CFRONT

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
797 - laser1

  ####     #       #
 #    #    #       #
 #       #####   #####
 #         #       #
 #    #    #       #
  ####

Request id: laser-9680     Printer: laser1    From: sfbc (laser-9004)    Via: nsc
Transit time: 1 minutes    Sat Feb  9 19:35:33 EST 1985
```

~~#22~~

#13

```
# %Z% %M% %I% %H% %T%
#
# cfront makefile
#
CC=CC
CFLAGS=-c
YACC=yacc
YFLAGS=
OSUF=.o
HDRS=cfront.h   \
        size.h          \
        token.h         \
        typedef.h       \
        yystype.h
OBJS=alloc$(OSUF)               \
        dcl$(OSUF)              \
        dcl2$(OSUF)             \
        del$(OSUF)              \
        error$(OSUF)            \
        expand$(OSUF)   \
        expr$(OSUF)             \
        expr2$(OSUF)            \
        lex$(OSUF)              \
        main$(OSUF)             \
        norm$(OSUF)             \
        norm2$(OSUF)            \
        print$(OSUF)            \
        repr$(OSUF)             \
        simpl$(OSUF)            \
        size$(OSUF)             \
        table$(OSUF)            \
        typ$(OSUF)              \
        typ2$(OSUF)

all     :       cfront

cfront  :       $(OBJS) y.tab$(OSUF)
                $(CC) $(OBJS) y.tab$(OSUF) -o cfront

y.tab.c :       gram.y
                $(YACC) $(YFLAGS) gram.y

y.tab$(OSUF)    :       y.tab.c $(HDRS)
                $(CC) $(CFLAGS) +E y.tab.c

alloc$(OSUF)    :       alloc.c
                $(CC) $(CFLAGS) alloc.c

dcl$(OSUF)      :       dcl.c
                $(CC) $(CFLAGS) dcl.c

dcl2$(OSUF)     :       dcl2.c
                $(CC) $(CFLAGS) dcl2.c

del$(OSUF)      :       del.c
                $(CC) $(CFLAGS) del.c
```

```
error$(OSUF)    :       error.c
                $(CC) $(CFLAGS) error.c

expand$(OSUF)   :       expand.c
                $(CC) $(CFLAGS) expand.c

expr$(OSUF)     :       expr.c
                $(CC) $(CFLAGS) expr.c

expr2$(OSUF)    :       expr2.c
                $(CC) $(CFLAGS) expr2.c

lex$(OSUF)      :       lex.c
                $(CC) $(CFLAGS) lex.c

main$(OSUF)     :       main.c
                $(CC) $(CFLAGS) main.c

norm$(OSUF)     :       norm.c
                $(CC) $(CFLAGS) norm.c

norm2$(OSUF)    :       norm2.c
                $(CC) $(CFLAGS) norm2.c

print$(OSUF)    :       print.c
                $(CC) $(CFLAGS) print.c

repr$(OSUF)     :       repr.c
                $(CC) $(CFLAGS) repr.c

simpl$(OSUF)    :       simpl.c
                $(CC) $(CFLAGS) simpl.c

size$(OSUF)     :       size.c
                $(CC) $(CFLAGS) size.c

table$(OSUF)    :       table.c
                $(CC) $(CFLAGS) table.c

typ$(OSUF)      :       typ.c
                $(CC) $(CFLAGS) typ.c

typ2$(OSUF)     :       typ2.c
                $(CC) $(CFLAGS) typ2.c

$(OBJS) :       $(HDRS)

clean   :
                rm -f $(OBJS) y.tab.c y.tab$(OSUF) *.i *..c

clobber :       clean
                rm -f cfront
```

```
/* %Z% %M% %I% %H% %T% */
#include "cfront.h"

extern void free(char*);
extern char *malloc(unsigned);
extern void print_free();

typedef class header HEADER;

static HEADER *morecore(unsigned);

class header {  /* free block header */
public:
        HEADER  *next;  /* next free block */
        unsigned size;  /* size of this free block */
};


HEADER base;            /* empty list to get started */
HEADER *allocp = NULL;  /* last allocated block */

void print_free()
{
        register HEADER* p, *q = 0;
        register int amount = 0;
        register int number = 0;

        for (p=allocp; q!=allocp; q=p=p->next) {
                number++;
                amount += p->size;
        }
        fprintf(stderr,"free: %d %d\n",number,amount*sizeof(HEADER) );
}

char *malloc(unsigned nbytes)   /* general-purpose storage allocator */
{
        register HEADER *p, *q;
        register int nunits;

        Nalloc++;
        nunits = 1+(nbytes+sizeof(HEADER)-1)/sizeof(HEADER);
        if ((q = allocp) == NULL) {     /* no free list yet */
                base.next = allocp = q = &base;
                base.size = 0;
        }
        for (p=q->next; ; q=p, p=p->next) {
                if (p->size >= nunits) {        /* big enough */
                        if (p->size == nunits)  /* exactly */
                                q->next = p->next;
                        else {  /* allocate tail end */
                                p->size -= nunits;
                                p += (int)p->size;
                                p->size = nunits;
                        }
                        allocp = q;
```

```
/*fprintf(stderr,"malloc(%d %d)->%d %d\n",nbytes,nunits*sizeof(HEADER),p+1,p+nunits)

                        register int* x = (int*)(p+1);
                        register int* y = (int*)(p+nunits);
                        while (x < y) *--y = 0;
                        return (char*) x;
                }
                if (p == allocp)  /* wrapped around free list */
                        if ((p = morecore(nunits)) == NULL)
                                return(NULL);   /* none left */
        }
}

#define NALLOC  1024    /* #units to allocate at once */

static HEADER *morecore(unsigned nu)    /* ask system for memory */
{
        char *sbrk(int);
        register char *cp;
        register HEADER *up;
        register int rnu;
        register int rnu2;

        rnu = NALLOC * ((nu+NALLOC-1) / NALLOC);
        cp = sbrk(rnu2 = rnu*sizeof(HEADER));
        Nfree_store += rnu2;
/*fprintf(stderr,"morecore %d %d -> %d Nf=%d\n", nu, rnu2, cp, Nfree_store); fflush(
*/
        if ((int)cp == -1)      /* no space at all */
                error("free store exhausted");
        up = (HEADER *)cp;
        up->size = rnu;
        free((char *)(up+1));
        return(allocp);
}

int NFn, NFtn, NFbt, NFpv, NFf, NFe, NFs, NFc;

void free(char* ap)             /* put block on free list */
{
        register HEADER *p, *q;

        if (ap == 0) return;

        p = (HEADER*)ap - 1;    /* point to header */

        Nfree++;

if (Nspy) {
        Pname pp = (Pname) ap;
        TOK t = pp->base;
        char* s = 0;

        switch (t) {
/*
        case TNAME: case NAME:
```

```
			NFn++;

			fprintf(stderr,"??name %d %d sz=%d\n",pp,t,p->size); fflush(stderr)
			break;
*/
	case INT: case CHAR: case TYPE: case VOID: case SHORT: case LONG:
	case FLOAT: case DOUBLE: case COBJ: case EOBJ: case FIELD:
				NFbt++; break;

	case PTR: case VEC:
				NFpv++; break;

	case FCT:		NFf++; break;
/*
	case INCR: case DECR: case ASSIGN: case CALL: case PLUS: case MINUS:
	case DEREF: case MUL: case DIV: case ASPLUS: case MOD: case UMINUS:
	case DOT: case REF: case CAST: case NEW: case NOT: case COMPL: case ER:
	case EQ: case NE: case GT: case LT: case LE: case GE:
	case ANDAND: case AND: case OR: case OROR: case SIZEOF:
	case ILIST: case ELIST: case CM: case QUEST: case RS: case LS:
	case TEXT: case IVAL: case FVAL:
				NFe++;

			fprintf(stderr,"??expr %d %d sz=%d\n",pp,t,p->size); fflush(stderr)
				break;
*/
	case ICON: case CCON: case STRING: case FCON: case THIS:
				NFc++; break;
/*
	case IF: case SM: case FOR: case WHILE: case DO: case BLOCK:
	case BREAK: case CONTINUE: case DEFAULT: case SWITCH: case CASE:
	case PAIR: case LABEL: case GOTO: case RETURN: case DELETE: case ASM:
				NFs++; break;
*/
	/*default:		if (0<t && t<140) fprintf(stderr,"delete tok %d\n",t);
*/
	}
}
/*fprintf(stderr,"free(%d)\n",ap);*/
	for (q=allocp; !(p > q && p < q->next); q=q->next)
		if (q >= q->next && (p > q || p < q->next))
			break;	/* at one end or other */

	if (p+p->size == q->next) { /* join to upper nbr */
		p->size += q->next->size;
		p->next = q->next->next;
	} else
		p->next = q->next;
	if (q+q->size == p) {	/* join to lower nbr */
		q->size += p->size;
		q->next = p->next;
	} else
		q->next = p;
	allocp = q;
}
```

```
/* %Z% %M% %I% %H% %T% */

/**************************************************************************

        C++ source for cfront, the C++ compiler front-end
        written in the computer science research center of Bell Labs

        Copyright (c) 1984 AT&T Technologies, Inc. All rigths Reserved
        THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

        If you ignore this notice the ghost of Ma Bell will haunt you forever.

cfront.h:

        Here is all the class definitions for cfront, and most of the externs

**************************************************************************/

/*      WARNING:
        This program relies on non-initialized class members being ZERO.
        This will be true as long as they are allocated using the "new" operator
*/

#include "token.h"
#include "typedef.h"

extern char* prog_name;         /* compiler name and version */
extern bit old_fct_accepted;    /* if set:
                                        old style function definitions are legal,
                                        implicit declarations are legal
                                */
extern bit fct_void;            /* if set:
                                        f() means f of no arguments
                                        f() means f(...)
                                   if not:
                                        f() illegal
                                        f() means f of no arguments
                                */
extern TOK scope_default;       /*      default scope of  externals
                                        STATIC or EXTERN
                                */
extern bit st_init;             /* static objects can be initialized by ctor */
extern int inline_restr;        /* inline expansion restrictions */
/*      free lists */
extern Pname name_free;
extern Pexpr expr_free;
extern Pstmt stmt_free;

extern int Nspy, Nn, Nbt, Nt, Ne, Ns, Nstr, Nc, Nl;

extern TOK      lex();
extern Pname    syn();
extern bit      print_mode;

        /* stage initializers: */
extern void     init_print();
extern void     init_lex();
extern void     int_syn();
extern void     ext(int);

extern char*    make_name(TOK);


class loc       /* a source file location */
{
public:
        short   file;   /* index into file_name[], or zero */
        short   line;

        void    put(FILE*);
        void    putline();
};

extern Loc curloc;
overload error;
extern int error(int, loc*, char* ...);
extern int error(int, char* ...);
extern int error(loc*, char* ...);
extern int error(char* ...);
extern int error_count;
extern bit debug;

extern FILE* in_file;
extern FILE* out_file;
extern char scan_started;
extern bit warn;
extern int br_level;
extern int bl_level;
extern Ptable ktbl;             /* keywords and typedef names */
extern char*    oper_name(TOK);
extern Ptable gtbl;             /* global names */
extern Pclass ccl;
extern Pbase defa_type;
extern Pbase moe_type;

extern Pstmt Cstmt;     /* current statement, or 0 */
extern Pname Cdcl;      /* name currently being declared, or 0 */
extern void put_dcl_context();

extern Ptable any_tbl;  /* table of undefined struct members */
extern Pbase any_type;
extern Pbase int_type;
extern Pbase char_type;
extern Pbase short_type;
extern Pbase long_type;
extern Pbase uchar_type;
extern Pbase ushort_type;
extern Pbase uint_type;
extern Pbase ulong_type;
extern Ptype Pchar_type;
extern Ptype Pint_type;
extern Ptype Pfctvec_type;
extern Pbase float_type;
extern Pbase double_type;
extern Pbase void_type;
extern Ptype Pvoid_type;
extern Pbase zero_type;
extern Ptype char2_type;
extern Ptype char3_type;
extern Ptype char4_type;

extern int byte_offset;
extern int bit_offset;
extern int max_align;
extern int stack_size;
extern int enum_count;
extern int const_save;

extern Pname class_name(Ptable,char*,bit);
extern Pname gen_find(Pname,Pfct);
extern char* gen_name(char*,char);

extern Pexpr dummy;     /* the empty expression */
extern Pexpr zero;
extern Pexpr one;
extern Pname sta_name;  /* qualifier for unary :: */

#define DEL(p) if (p && (p->permanent==0)) p->del()
#define PERM(p) p->permanent=1
#define UNPERM(p) p->permanent=0

struct node {
        TOK     base;
        TOK     n_key;  /* for names in table: class */
        bit     permanent;
};

extern Pclass Ebase, Epriv;     /* lookc return values */

class table : public node {
/*      a table is a node only to give it a "base" for debugging */
        short   size;
        short   hashsize;
        Pname*  entries;
        short*  hashtbl;
        short   free_slot;      /* next free slot in entries */
public:
        short   init_stat;      /* ==0 if block(s) of table not simplified,
                                   ==1 if simplified but had no initializers,
                                   ==2 if simplified and had initializers.
                                */
        Pstmt   real_block;     /* the last block the user wrote,
                                   not one of the ones cfront created
                                */
                table(short, Ptable, Pname);
        Ptable  next;           /* table for enclosing scope */
        Pname   t_name;         /* name of the table */
        Pname   look(char*, TOK);
        Pname   insert(Pname, TOK);
        void    grow(int);
        void    set_scope(Ptable t)     { next = t; };
        void    set_name(Pname n)       { t_name = n; };
        Pname   get_mem(int);
                int     max()           { return free_slot-1; };
        void    dcl_print(TOK,TOK);
        Pname   lookc(char*, TOK);
        Pexpr   find_name(Pname, bit, Pexpr);
        void    del();
};

extern bit Nold;
extern bit vec_const;
extern void restore();
extern void set_scope(Pname);
extern Plist modified_tn;
extern Pbase start_cl(TOK, Pname, Pname);
extern void end_cl();
extern Pbase end_enum(Pname, Pname);

/************ types : basic types, aggregates, declarators ************/

extern bit new_type;
extern Pname cl_obj_vec;
extern Pname eobj;


struct type : public node {
        bit     defined;        /*      0       if only declared
                                        1       if only defined
                                        2       if simplified
                                        not used systematically yet
                                */
        void    print();
        void    dcl_print(Pname);
        void    base_print();
        void    del();

        Pname   is_cl_obj();    /* sets cl_obj_vec */
        void    dcl(Ptable);
        int     tsizeof();
        bit     tconst();
        int     align();
        TOK     kind(TOK,TOK);
        TOK     integral(TOK oo)        { return kind(oo,I); };
        TOK     numeric(TOK oo)         { return kind(oo,N); };
        TOK     num_ptr(TOK oo)         { return kind(oo,P); };
        bit     fct_type();
        bit     vec_type();
        bit     check(Ptype, TOK);
        Ptype   deref();
        Pptr    addrof();
        char*   signature(char*);
```

```
};

extern bit vrp_equiv;

        class enumdef : public type {   /* ENUM */
        public:
                Pname   mem;
                bit     e_body;
                int     no_of_enumerators;
                        enumdef(Pname n)        { base=ENUM; mem=n; };
                void    print();
                void    dcl_print(Pname);
                void    dcl(Pname, Ptable);
                void    simpl();
        };

        class classdef : public type {  /* CLASS */
        public:
                Pname   clbase;
                bit     pubbase;
                bit     c_body;         /* print definition only once */
                TOK     csu;            /* CLASS, STRUCT, UNION, or ANON */
                char*   string;         /* name of class */
                Pname   pubmem;
        Ptable  memtbl;
        short   obj_size;
        char    obj_align;
        Ptable  _m; short _o; char _a;

                classdef(TOK, Pname);

        void    print();
        void    dcl_print(Pname);
        void    simpl();

        Pname   privmem;
        Plist   friend_list;
        Pname   pubdef;
                Plist   tn_list;        /* list of member names hiding type names a
                Pclass  in_class;       /* enclosing class, or 0 */
                Ptype   this_type;
                char    virt_count;     /* number of virtual functions
                                                incl. virtuals in base classes
                                        */
                Pname*  virt_init;      /* vector of jump table initializers */
                Pname   itor;           /* constructor X(X&) */
                Pname   conv;           /* operator T() chain */
        void    print_members();
        void    dcl(Pname, Ptable);
        bit     has_friend(Pname);
                TOK     is_simple()     { return (csu==CLASS)?0:csu; };
        Pname   has_oper(TOK);
        Pname   has_ctor()      { return memtbl->look("_ctor",0); }
        Pname   has_dtor()      { return memtbl->look("_dtor",0); }
        Pname   has_itor()      { return itor; }
        Pname   has_ictor();
```

```
};

class basetype : public type
        /*      ZTYPE CHAR SHORT INT LONG FLOAT DOUBLE
                FIELD EOBJ COBJ TYPE ANY
        */
        /*      used for gathering all the attributes
                for a list of declarators

                ZTYPE is the (generic) type of ZERO
                ANY is the generic type of an undeclared name
        */
{
public:
        bit     b_unsigned;
        bit     b_const;

        bit     b_typedef;
        bit     b_inline;
        bit     b_virtual;
        bit     b_short;
        bit     b_long;

        char    b_offset;
        TOK     b_sto;          /* AUTO STATIC EXTERN REGISTER 0 */
        Pname   b_name;         /* name of non-basic type */
        Pexpr   b_field;        /* field size expression for a field */
        char    b_bits;         /* number of bits in field */
        Ptable  b_table;        /* memtbl for b_name, or 0 */
        Pname   b_xname;        /* extra name */

                basetype(TOK, Pname);

        Pbase   type_adj(TOK);
        Pbase   base_adj(Pbase);
        Pbase   name_adj(Pname);
        Pbase   check(Pname);
        Pname   aggr();
        void    normalize();
        void    dcl_print();
        Pbase   arit_conv(Pbase);
};

struct fct : public type                /* FCT */
{
        Ptype   returns;
        Pname   argtype;
        Ptype   s_returns;
        Pname   f_this;
        Pblock  body;
        Pexpr   f_init;
        short   frame_size;
        TOK     nargs;
        TOK     nargs_known;    /* KNOWN, ELLIPSIS, or 0 */
        char    f_virtual;      /* 1+index in virtual table, or 0 */
```

```
	char	f_inline;	/* 1 if inline, 2 if being expanded, else 0 */
	Pexpr	f_expr;		/* body expanded into an expression */
	Pexpr	last_expanded;
		fct(Ptype, Pname, TOK);

	void	argdcl(Pname);
	Ptype	normalize(Ptype);
	void	dcl_print();
	void	dcl(Pname);
	bit	declared() { return (nargs_known); };
	void	simpl();
	Pexpr	expand(Pname,Ptable,Pexpr);
};


struct name_list {
	Pname	f;
	Plist	l;
		name_list(Pname ff, Plist ll) { f=ff; l=ll; };
};

struct gen: public type		/* OVERLOAD */
{
	Plist	fct_list;
	char*	string;
		gen(char*);
	Pname	add(Pname, int);
	Pname	find(Pfct);
};

struct vec : public type			/* VEC */
	/*	typ [ dim ] */
{
	Ptype	typ;
	Pexpr	dim;
	int	size;

		vec(Ptype t, Pexpr e) { Nt++; base=VEC; typ=t; dim=e; };

	Ptype	normalize(Ptype);
	void	print();
};

struct ptr : public type			/* PTR RPTR*/
{
	Ptype	typ;
	bit	rdo;	/* *CONST */

		ptr(TOK b, Ptype t, bit r = 0) { Nt++; base=b; typ=t; rdo=r; };
	Ptype	normalize(Ptype);
};
```

```
/******************************** constants ********************************/

                /* STRING ZERO ICON FCON CCON ID */
                /* IVAL FVAL LVAL */

/******************************* expressions ********************************/


extern Pexpr next_elem();
extern void new_list(Pexpr);
extern void list_check(Pname, Ptype, Pexpr);
extern Pexpr ref_init(Pptr,Pexpr,Ptable);
extern Pexpr class_init(Pexpr,Ptype,Pexpr,Ptable);
extern Pexpr check_cond(Pexpr, TOK, Ptable);

        class expr : public node        /* PLUS, MINUS, etc. */
                /* IMPORTANT:   all expressions are of sizeof(expr) */
                /*      DEREF           =>      *e1 (e2==0) OR e1[e2]
                        UMINUS          =>      -e2
                        INCR (e1==0)    =>      ++e2
                        INCR (e2==0)    =>      e1++
                        CM              =>      e1 , e2
                        ILIST           =>      LC e1 RC   (an initializer list)
                        a Pexpr may denote a name
                */
        {
        public:
                union {
                Ptype   tp;
                int     syn_class;
                };
                union {
                Pexpr e1;
                char* string;
                };
                union {
                Pexpr   e2;
                Pexpr   n_initializer;
                };
                union {                         /* used by the derived classes */
                Ptype   tp2;
                Pname   fct_name;
                Pexpr   cond;
                Pname   mem;
                Ptype   as_type;
                Ptable  n_table;
                Pin     il;
                };

                        expr(TOK, Pexpr, Pexpr);
                        ~expr();

                void    del();
                void    print();
                Pexpr   typ(Ptable);
                int     eval();
                int     lval(TOK);
                Ptype   fct_call(Ptable);
                Pexpr   address();
                Pexpr   contents();
                void    simpl();
                Pexpr   expand();
                bit     not_simple();
        };

        extern char* Neval;

        struct typed_obj : public expr {
                typed_obj(TOK t, char* s) : (t,(Pexpr)s,0) { this=0; }
        };

        struct texpr : public expr      /* NEW CAST VALUE */
        {
                texpr(TOK bb, Ptype tt, Pexpr ee) : (bb,ee,0) { this=0; tp2=tt; }
        };

        struct call : public expr       /* CALL */
        {
                call(Pexpr aa, Pexpr bb) : (CALL,aa,bb) { this=0; }

                void    simpl();
                Pexpr   expand(Ptable);
        };

        struct qexpr : public expr      /* QUEST */
        /* cond ? e1 : e2 */
        {
                qexpr(Pexpr ee, Pexpr ee1, Pexpr ee2) : (QUEST,ee1,ee2) { this=0; c
        };

        struct ref : public expr        /* REF DOT */
        /* e1->mem OR e1.mem */
        {
                ref(TOK ba, Pexpr a, Pname b) : (ba,a,0) { this=0; mem=b; }
        };



/************************** names (are expressions) ****************************/

        class name : public expr {
                /* NAME TNAME and the keywords in the ktbl */
        public:
        /*      Pexpr   n_initializer;  */
                int     n_val;          /* the value of n_initializer */
                TOK     n_oper;         /* name of operator or 0 */
                TOK     n_sto;          /* STO keyword: EXTERN, STATIC, AUTO, REGIS
                TOK     n_stclass;      /* STATIC AUTO REGISTER 0 */
                TOK     n_scope;        /* EXTERN STATIC FCT ARG PUBLIC 0 */
                short   n_offset;       /* byte offset in frame or struct */
                Pname   n_list;
                Pname   n_tbl_list;
        /*      Ptable  n_table;        */
                short   n_used;
                short   n_addr_taken;
                short   n_assigned_to;
                char    n_union;        /* 0 or union index */
                bit     n_evaluated;    /* 0 or n_val holds the value */
                short   lex_level;
                Loc     where;
                union {
                Pname   n_qualifier;
                Ptable  n_realscope;    /* for labels (always entered in
                                           function table) the table for the actual
                                           scope in which label occurred.
                                        */
                };

                        name(char* =0);
                        ~name();

                void    del();
                void    print();
                void    dcl_print(TOK);
                Pname   normalize(Pbase, Pblock, bit);
                Pname   tdef();
                Pname   tname(TOK);
                Pname   dcl(Ptable,TOK);
                int     no_of_names();
                void    hide();
                void    unhide()        { n_key=0; n_list=0; };
                void    use()           { n_used++; };
                void    assign();
                void    call()          { n_used++; };
                void    take_addr()     { n_addr_taken++; };
                void    check_oper(Pname);
                void    simpl();
        };


/********************** statements *********************************/

        class stmt : public node {      /* BREAK CONTINUE DEFAULT */
        /*      IMPORTANT: all statement nodes have sizeof(stmt) */
        public:
                Pstmt   s;
                Pstmt   s_list;
                Loc     where;
                union {
                Pname   d;
                Pexpr   e2;
                Pstmt   has_default;
                int     case_value;
                };
                union {
                Pexpr   e;
                bit     own_tbl;
                Pstmt   s2;
                };
                Ptable  memtbl;
                union {
                Pstmt   for_init;
                Pstmt   else_stmt;
                Pstmt   case_list;
                };

                        stmt(TOK, loc, Pstmt);
                        ~stmt();

                void    del();
                void    print();
                void    dcl();
                void    reached();
                Pstmt   simpl();
                Pstmt   expand();
                Pstmt   copy();
        };

extern Pname dcl_temp(Ptable, Pname);
extern char* temp(char*, char*, char*);
extern Ptable scope;
extern Ptable expand_tbl;
extern Pname expand_fn;

        struct estmt : public stmt      /* SM WHILE DO SWITCH RETURN DELETE CASE */
                /* SM (e!=0)    =>      e;
                in particular assignments and function calls
                SM (e==0)       =>      ;       (the null statement)

                CASE            =>      case e : s ;
                */
        {
                estmt(TOK t, loc ll, Pexpr ee, Pstmt ss) : (t,ll,ss) { this=0; e=ee
        };

        struct ifstmt : public stmt     /* IF */
                /* else_stme==0 =>      if (e) s
                else_stmt!=0 => if (e) s else else_stmt
                */
        {
                ifstmt(loc ll, Pexpr ee, Pstmt ss1, Pstmt ss2)
                        : (IF,ll,ss1) { this=0; e=ee; else_stmt=ss2; };
        };

        struct lstmt : public stmt      /* LABEL GOTO */
                /*
                        d : s
                        goto d
                */
        {
                lstmt(TOK bb, loc ll, Pname nn, Pstmt ss) : (bb,ll,ss) { this=0; d=
        };
```

```
	struct forstmt : public stmt	/* FOR */
	{
		forstmt(loc ll, Pstmt fss, Pexpr ee1, Pexpr ee2, Pstmt ss)
			: (FOR,ll,ss) { this=0; for_init=fss; e=ee1; e2=ee2; }
	};

	struct block : public stmt	/* BLOCK */
		/* { d s } */
	{
		block(loc ll, Pname nn, Pstmt ss) : (BLOCK,ll,ss) { this=0; d=nn; }
		void	dcl(Ptable);
		Pstmt	simpl();
	};

	struct pair : public stmt	/* PAIR */
	{
		pair(loc ll, Pstmt a, Pstmt b) : (PAIR,ll,a) { this=0; s2 = b; }
	};

class nlist {
public:
	Pname	head;
	Pname	tail;
		nlist(Pname);
	void	add(Pname n)	{ tail->n_list = n; tail = n; };
	void	add_list(Pname);
};

extern Pname name_unlist(nlist*);

class slist {
public:
	Pstmt	head;
	Pstmt	tail;
		slist(Pstmt s)	{ Nl++; head = tail = s; };
	void	add(Pstmt s)	{ tail->s_list = s; tail = s; };
};

extern Pstmt stmt_unlist(slist*);

class elist {
public:
	Pexpr	head;
	Pexpr	tail;
		elist(Pexpr e)	{ Nl++; head = tail = e; };
	void	add(Pexpr e)	{ tail->e2 = e; tail = e; };
};

extern Pexpr expr_unlist(elist*);

extern class dcl_context * cc;

class dcl_context {
public:
	Pname	c_this;	/* current fct's "this" */
```

```
        Ptype   tot;    /* type of "this" or 0 */
        Pname   not;    /* name of "this"'s class or 0 */
        Pclass  cot;    /* the definition of "this"'s class */
        Ptable  ftbl;   /* current fct's symbol table */
        Pname   nof;    /* current fct's name */

        void    stack()         { cc++; *cc = *(cc-1); };
        void    unstack()       { cc--; };
};

#define MAXCONT 20
extern dcl_context ccvec[MAXCONT];

extern bit can_coerce(Ptype, Ptype);
extern void yyerror(char*);
extern TOK back;

                /* "spy" counters: */
extern int Nspy;
extern int Nfile, Nline, Ntoken, Nname, Nfree_store, Nalloc, Nfree;
extern int NFn, NFtn, NFpv, NFbt, NFf, NFs, NFc, NFe, NFl;
extern char* line_format;

extern Plist isf_list;
extern Pstmt st_ilist;
extern Pstmt st_dlist;

extern Ptype np_promote(TOK, TOK, TOK, Ptype, Ptype, TOK);
extern void new_key(char*, TOK, TOK);

extern Pname dcl_list;
extern int over_call(Pname, Pexpr);
extern Pname Nover;
extern Pname Ncoerce;
extern Nover_coerce;

const MIA = 8;
struct iline {
        Pname   fct_name;       /* fct called */
        Pin     i_next;
        Ptable  i_table;
        Pname   local[MIA];     /* local variable for arguments */
        Pexpr   arg[MIA];       /* actual arguments for call */
        Ptype   tp[MIA];        /* type of formal arguments */
};

extern Pexpr curr_expr;
extern Pin curr_icall;
#define FUDGE111 111

extern Pstmt curr_loop;
extern Pblock curr_block;
extern Pstmt curr_switch;
extern bit arg_err_suppress;
extern loc last_line;
extern no_of_undcl;
extern Pname undcl1, undcl2;

extern int strlen(char*);
extern int strcpy(char*,char*);
extern int strcmp(char*,char*);
extern int str_to_int(char*);

extern Pname vec_new_fct;
extern Pname vec_del_fct;
/* end */
/* testing edget */
/* %Z% %M% %I% %H% %T% */
/***************************************************************************

        C++ source for cfront, the C++ compiler front-end
        written in the computer science research center of Bell Labs

        Copyright (c) 1984 AT&T Technologies, Inc. All rigths Reserved
        THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

        If you ignore this notice the ghost of Ma Bell will haunt you forever.

dcl.c:

        ``declare'' all names, that is insert them in the appropriate symbol tables

        Calculate the size for all objects (incl. stack frames),
        and find store the offsets for all members (incl. auto variables).
        "size.h" holds the constants needed for calculating sizes.

        Note that (due to errors) functions may nest

****************************************************************************/


#include "cfront.h"
#include "size.h"

class dcl_context ccvec[MAXCONT], * cc = ccvec;
int byte_offset;
int bit_offset;
int max_align;
int stack_size;
int enum_count;
int friend_in_class;

void name.check_oper(Pname cn)
{
        switch (n_oper) {
        case CALL:
                if (cn == 0) error("operator() must be aM");
                break;
        case DEREF:
                if (cn == 0) error("operator[] must be aM");
                break;
        case 0:
        case TNAME:     /* may be a constructor */
                if (cn && strcmp(cn->string,string)==0) {
                        if (tp->base == FCT) {
                                Pfct f = (Pfct)tp;
                                if (f->returns!=defa_type && fct_void==0)
                                        error("%s::%s() with returnT",string,string
                                f->returns = void_type;
                                string = "_ctor";
                                n_oper = CTOR;
                        }
                        else
```

```
                                error('s',"struct%cnM%n",cn,cn);
                }
                else
                        n_oper = 0;
                break;
        case DTOR:      /* must be a destructor */
                if (cn == 0) {
                        n_oper = 0;
                        error("destructor ~%s() not inC",string);
                }
                else if (strcmp(cn->string,string) == 0) {
                        Pfct f = (Pfct)tp;
                        string = "_dtor";
                        if (tp->base != FCT) {
                                error("%s::~%s notF",cn->string,cn->string);
                                tp = new fct(void_type,0,1);
                        }
                        else if (f->returns!=defa_type && fct_void==0)
                                error("%s::~%s() with returnT",cn->string,cn->strin
                        if (f->argtype) {
                                if (fct_void==0) error("%s::~%s() withAs",cn->strin
                                f->nargs = 0;
                                f->nargs_known = 1;
                                f->argtype = 0;
                        }
                        f->returns = void_type;
                }
                else {
                        error("~%s in%s",string,cn->string);
                        n_oper = 0;
                }
                break;
        case TYPE:
                if (cn == 0) {
                        error("operator%t() not aM",(Ptype)n_initializer);
                        n_oper = 0;
                        n_initializer = 0;
                }
                else {
                        Pfct f = (Pfct)tp;
                        Ptype tx = (Ptype)n_initializer;
/*error('d',"operator%t()",tx);*/
                        n_initializer = 0;
                        if (f->base != FCT) error("badT for%n::operator%t()",cn,tx)
                        if (f->returns != defa_type) {
                                if (f->returns->check(tx,0)) error("bad resultT for
                                DEL(f->returns);
                        }
                        if (f->argtype) {
                                error("%n::operator%t() withAs",cn,tx);
                                f->argtype = 0;
                        }
                        f->returns = tx;
                        Pname nx = tx->is_cl_obj();
                        if (nx && can_coerce(tx,cn->tp)) error("both %n::%n(%n) and
                        char buf[128];
                        char* bb = tx->signature(buf);
                        int l2 = bb-buf-1;
                        char* p = new char[l2+1];
                        strcpy(p,buf);
                        string = p;
                }
                break;
        }
}

Pname name.dcl(Ptable tbl, TOK scope)
/*
        enter a copy of this name into symbol table "tbl";
                - create local symbol tables as needed

        "scope" gives the scope in which the declaration was found
                - EXTERN, FCT, ARG, PUBLIC, or 0
        Compare "scope" with the specified storage class "n_sto"
                - AUTO, STATIC, REGISTER, EXTERN, OVERLOAD, FRIEND, or 0

        After name.dcl()
        n_stclass ==    0               class or enum member
                        REGISTER        auto variables declared register
                        AUTO            auto variables not registers
                        STATIC          statically allocated object
        n_scope ==      0               private class member
                        PUBLIC          public class member
                        EXTERN          name valid in this and other files
                        STATIC          name valid for this file only
                        FCT             name local to a function
                        ARG             name of a function argument
                        ARGT            name of a type defined in an argument list

        typecheck function bodies;
        typecheck initializers;

        note that functions (error recovery) and classes (legal) nest

        The return value is used to chain symbol table entries, but cannot
        be used for printout because it denotes the sum of all type information
        for the name

        names of typenames are marked with n_oper==TNAME

        WARNING: The handling of scope and storage class is cursed!
*/
{
        Pname nn;
        Ptype nnt = 0;
        Pname odcl = Cdcl;

        if (this == 0) error('i',"0->name.dcl()");
        if (tbl == 0) error('i',"%n->name.dcl(tbl=0,%k)",this,scope);
        if (tbl->base != TABLE) error('i',"%n->name.dcl(tbl=%d,%k)",this,tbl->base,
        if (tp == 0) error('i',"name.dcl(%n,%k)T missing",this,scope);
/*fprintf(stderr,"(%d %s)->dcl(tbl=%d,scope=%d) tp = (%d %d)\n",this,string,tbl,scop
	Cdcl = this;
	switch (base) {
	case TNAME:
		tp->dcl(tbl);
		PERM(tp);
		nn = new name(string);
		nn->base = TNAME;
		nn->tp = tp;
		tbl->insert(nn,0);
		delete nn;
		Cdcl = odcl;
		return this;
	case NAME:
		switch (n_oper) {
		case TNAME:
			if (tp->base != FCT) n_oper = 0;
			break;
		case COMPL:
			if (tp->base != FCT) {
				error("~%s notF",string);
				n_oper = 0;
			}
			break;
		}
		break;
	default:
		error('i',"NX in name.dcl()");
	}

	if (n_qualifier) {	/*	class function: c.f(); */
		if (tp->base != FCT) {
			error("QdN%n inD of nonF",this);
			Cdcl = odcl;
			return 0;
		}

		Pname cn = n_qualifier;
		switch (cn->base) {
		case TNAME:
			break;
		case NAME:
			cn = gtbl->look(cn->string,0);
			if (cn && cn->base==TNAME) break;
		default:
			error("badQr%n for%n",n_qualifier,this);
			Cdcl = odcl;
			return 0;
		}
		cn = ((Pbase)cn->tp)->b_name;
		if (n_oper) check_oper(cn);

		Pclass cl = (Pclass)cn->tp;
		if (cl == cc->cot) {
			n_qualifier = 0;
			goto xdr;
		}
		else if (cl->defined == 0) {
			error("C%nU",cn);
			Cdcl = odcl;
			return 0;
		}

		Ptable etbl = cl->memtbl;
		Pname x = etbl->look(string,0);
		if(x==0 || x->n_table!=etbl) {
			error("%n is not aM of%n",this,cn);
			Cdcl = odcl;
			return 0;
		}
	}
xdr:
	if (n_oper && tp->base!=FCT && n_sto!=OVERLOAD)
		error("operator%k not aF",n_oper);


	/*	if a storage class was specified
			check that it is legal in the scope
		else
			provide default storage class
		some details must be left until the type of the object is known
	*/

	n_stclass = n_sto;
	n_scope = scope;	/* default scope & storage class */

	if (n_sto==0 && scope==EXTERN) {
		if (scope_default==STATIC) {
			switch (tp->base) {
			case FCT:
			{
				Pfct f = (Pfct)tp;
				if ( strcmp(string,"main") )
					n_scope = (f->body) ? STATIC : EXTERN;
				break;
			}
			case CLASS:
			case ENUM:
			default:
				n_scope = STATIC;
			}
		}
	}

	switch (n_sto) {
	default:
		error('i',"unX %k",n_sto);
	case FRIEND:
	{
		Pclass cl = cc->cot;

		switch (scope) {
		case 0:
```

```
                case PUBLIC:
                        break;
                default:
                        error("friend%n not in classD(%k)",this,scope);
                        base = 0;
                        Cdcl = odcl;
                        return 0;
                }

                switch (n_oper) {
                case 0:
                case NEW:
                case DELETE:
                case CTOR:
                case DTOR:
                        n_sto = 0;
                        break;
                default:
                        n_sto = OVERLOAD;
                }

                switch (tp->base) {
        /*      case INT:       undefined: implicitly define as class
                        nn = tname(CLASS);
                        nn->tp->dcl(gtbl);
                        break;
        */
                case COBJ:
                        nn = ((Pbase)tp)->b_name;
                        break;
                case CLASS:
                        nn = this;
                        break;
                case FCT:
                        cc->stack();
                        cc->not = 0;
                        cc->tot = 0;
                        cc->cot = 0;
                        friend_in_class++;
                        nn = dcl(gtbl,EXTERN);
                        friend_in_class--;
/*fprintf(stderr,"ff %s %d\n",nn->string,nn->tp->base);*/
                        cc->unstack();
                        if (nn->tp->base == OVERLOAD) {
                                Pgen g = (Pgen)nn->tp;
                                nn = g->find( (Pfct)tp );
                        }
                        break;
                default:
                        error("badT%t of friend%n",tp,this);
                }
                PERM(nn);
                cl->friend_list = new name_list(nn,cl->friend_list);
                Cdcl = odcl;
                return nn;
        }
```

```
	case OVERLOAD:
		n_sto = 0;
		switch (scope) {
		case 0:
		case PUBLIC:
			error('w',"overload inCD (ignored)");
			switch (tp->base) {
			case INT:
				base = 0;
				Cdcl = odcl;
				return this;
			case FCT:
				return dcl(tbl,scope);
			}
		}
		if (n_oper && tp->base==FCT) break;
		nn = tbl->insert(this,0);

		if (Nold) {
			if (nn->tp->base != OVERLOAD) {
				error("%n redefined as overloaded",this);
				nn->tp = new gen(string);
			}
		}
		else {
			nn->tp = new gen(string);
		}

		switch (tp->base) {
		case INT:
			base = 0;
			Cdcl = odcl;
			return nn;
		case FCT:
			break;
		default:
			error("N%n ofT%k cannot be overloaded",this,tp->base);
			Cdcl = odcl;
			return nn;
		}
		break;
	case REGISTER:
		if (tp->base == FCT) {
			error('w',"%n: register (ignored)",this);
			goto ddd;
		}
	case AUTO:
		switch (scope) {
		case 0:
		case PUBLIC:
		case EXTERN:
			error("%k not inF",n_sto);
			goto ddd;
		}
		break;
	case EXTERN:
```

```
                switch (scope) {
                case ARG:
                        error("externA");
                        goto ddd;
                case 0:
                case PUBLIC:
                        /* extern is provided as a default for functions without bo
                        if (tp->base != FCT) error("externM%n",this);
                        goto ddd;
                }
                n_stclass = STATIC;
                n_scope = EXTERN;       /* avoid FCT scoped externs to allow better
                break;
        case STATIC:
                switch (scope) {
                case ARG:
                        error("static used forA%n",this);
                        goto ddd;
                case 0:
                case PUBLIC:
                        n_stclass = STATIC;
                        n_scope = scope;
                        break;
                default:
                        n_scope = STATIC;
                }
                break;
        case 0:
        ddd:
                switch (scope) {        /* default storage classes */
                case EXTERN:
                        switch (tp->base) {
                        case FCT:       /* anomaly:     f(int); => extern f(int); *
                                break;
                        default:
                                n_scope = scope_default;
                        }
                        n_stclass = STATIC;
                        break;
                case FCT:
                        if (tp->base == FCT) {
                                n_stclass = STATIC;
                                n_scope = EXTERN;
                        }
                        else
                                n_stclass = AUTO;
                        break;
                case ARG:
                        if (tp->base == FCT) error("%n asA",this);
                        n_stclass = AUTO;
                        break;
                case 0:
                case PUBLIC:
                        n_stclass = 0;
                        break;
                }
```

```
	}


	/*
		now insert the name into the appropriate symbol table,
		and compare types with previous declarations of that name

		do type dependent adjustments of the scope
	*/

	switch (tp->base) {
	case ASM:
	{	Pbase b = (Pbase)tp;
		Pname n = tbl->insert(this,0);
		n->assign();
		n->use();
		return this;
	}

	case CLASS:
	{	Pclass cl;
		Pbase bt;
		Pname bn;
		Pclass nest;
		Pname nx = ktbl->look(string,0);		/* TNAME */
/*fprintf(stderr,"%s: nx %d\n",string,nx);*/
		if (nx == 0) {
			/*	search for hidden name for
					(1) nested class declaration
					(2) local class declaration
			*/
			for (nx=ktbl->look(string,HIDDEN); nx; nx=nx->n_tbl_list) {
				if (nx->n_key != HIDDEN) continue;
				if (nx->tp->base != COBJ) continue;
				bt = (Pbase)nx->tp;
				bn = bt->b_name;
				cl = (Pclass)bn->tp;
				if (cl == 0) continue;
				if ((nest=cl->in_class) && nest==cc->cot)
					goto bbb;
				else if (cc->nof	/* fudge */
					&& cc->nof->where.line<nx->where.line)
					goto bbb;
			}
			error('i',"%n is not aTN",this);
		}
		else {
			bt = (Pbase)nx->tp;			/* COBJ */
			bn = bt->b_name;
			nest = 0;
		}
bbb:
/*fprintf(stderr,"bbb: bt %d %d\n",bt,bt->base); fflush(stderr);*/
		bn->where = nx->where;
		Pname bnn = tbl->insert(bn,CLASS);	/*copy for member lookup */
		cl = (Pclass)bn->tp;
```

```
                                                                /* CLASS */
/*fprintf(stderr,"cl %d %d\n",cl,cl->base); fflush(stderr);*/
                if (cl->defined)
                        error("C%n defined twice",this);
                else {
                        if (bn->n_scope == ARG) bn->n_scope = ARGT;
                        cl->dcl(bn,tbl);
                        if (nest) {
                                int l1 = strlen(cl->string);
                                int l2 = strlen(nest->string);
                                char* s = new char[l1+l2+2];
                                strcpy(s,nest->string);
                                s[l2] = '_';
                                strcpy(s+l2+1,cl->string);
                                cl->string = s;
                        /*      cl->memtbl->t_name->string = s;*/
                        }
                }
                tp = cl;
                Cdcl = odcl;
                return bnn;
        }

        case ENUM:
        {       Pname nx = ktbl->look(string,0);                /* TNAME */
                if (nx == 0) {
                        nx = ktbl->look(string,HIDDEN);         /* hidden TNAME */
                }
                Pbase bt = (Pbase)nx->tp;                       /* EOBJ */
                Pname bn = bt->b_name;
                Pname bnn = tbl->insert(bn,CLASS);
                Penum en = (Penum)bn->tp;                       /* ENUM */
                if (en->defined)
                        error("enum%n defined twice",this);
                else {
                        if (bn->n_scope == ARG) bn->n_scope = ARGT;
                        en->dcl(bn,tbl);
                }
                tp = en;
                Cdcl = odcl;
                return bnn;
        }

        case FCT:
        {       Pfct f = (Pfct)tp;
                Pname class_name;
                Ptable etbl;
                int can_overload;
                int in_class_dcl = (int)cc->not;
                int just_made = 0;

                if (f->f_inline) n_sto = STATIC;

                if (f->argtype) {
                        Pname a;
                        int oo = const_save;
                const_save = 1;
                for (a=f->argtype; a; a=a->n_list) {
                        Pexpr init;
                        if (init = a->n_initializer) {
                                int i = 0;
                                init = init->typ(tbl);
                                if (a->tp->check(init->tp,ARG)==0
                                || (i=can_coerce(a->tp,init->tp))) {
                                        if (1 < i) error("%d possible conve
                                        if (Ncoerce) {
                                                Pname cn = init->tp->is_cl_
                                                Pclass cl = (Pclass)cn->tp;
                                                Pref r = new ref(DOT,init,N
                                                init = new expr(G_CALL,r,0)
                                                init->fct_name = Ncoerce;
                                                init->tp = a->tp;
                                        }
                                        init->simpl();
                                        init->permanent = 2;
                                        a->n_initializer = init;
                                }
                                else {
                                        error("badIrT%t forA%n",init->tp,a)
                                        DEL(init);
                                        a->n_initializer = 0;
                                }
                        }

                flatten1:
                        switch (a->tp->base) {
                        case TYPE:
                                a->tp = ((Pbase)a->tp)->b_name->tp;
                                goto flatten1;
                        case CHAR:
                        case SHORT:
                        /*      error('w',"A ofT%k (becomes int)",a->tp->ba
                                a->tp = int_type;
                                break;
                        case FLOAT:
                        /*      error('w',"A ofT float (becomes double)");
                                a->tp = double_type;
                                break;
                        }
                }
                const_save = oo;
        }

        tp->dcl(tbl); /* must be done before the type check */

        if (n_qualifier) {      /* qualified name: c.f() checked above */
                if (in_class_dcl) {
                        error("unXQN%n",this);
                        Cdcl = odcl;
                        return 0;
                }
                class_name = ((Pbase)n_qualifier->tp)->b_name;
```

```
                        etbl = ((Pclass)class_name->tp)->memtbl;
                }
                else {
                        class_name = cc->not;
                        /* beware of local function declarations in member function
                        if (class_name && tbl!=cc->cot->memtbl) {
                                class_name = 0;
                                in_class_dcl = 0;
                        }
                        if (n_oper) check_oper(class_name);
                        etbl = tbl;
                }

                if (etbl==0 || etbl->base!=TABLE) error('i',"N.dcl: etbl=%d",etbl);

                switch (n_oper) {
                case NEW:
                case DELETE:
                        switch (scope) {
                        case 0:
                        case PUBLIC:
                                error("%nMF",this);
                        }
                case 0:
                        can_overload = in_class_dcl;
                        break;
                case CTOR:
                        if (f->f_virtual) {
                                error("virtual constructor");
                                f->f_virtual = 0;
                        }
                case DTOR:
                        if (fct_void) n_scope = PUBLIC;
                        can_overload = in_class_dcl;
                        break;
                default:
                        can_overload = 1;       /* all operators are overloaded */
                }

                switch (scope) {
                case FCT:
                case ARG:
                {       Pname nx = gtbl->insert(this,0);
                        n_table = 0;
                        n_tbl_list = 0;
                        /* no break */
                }
                default:
                        nn = etbl->insert(this,0);
                        nn->assign();
                        n_table = etbl;
                        break;
                }


                if (Nold) {
                        Pfct nf = (Pfct)nn->tp;
/*error('d',"%n: tp%t nf%t",nn,tp,nf);*/
                        if (nf->base==ANY || f->base==ANY)
                                ;
                        else if (nf->base == OVERLOAD) {
                                Pgen g = (Pgen) nf;
                                nn = g->add(this,0);
                                string = nn->string;
                                if (Nold == 0) {
                                        if (f->body) {
                                                if (n_qualifier) {
                                                        error(0,"badAL for overload
                                                        Cdcl = odcl;
                                                        return 0;
                                                }
                                                else if (f->f_inline==0 && n_oper==
                                                        error('w',"overloaded %n de
                                        }
                                        goto thth;
                                }
                                else {
                                        if (f->body==0 && friend_in_class==0) error
                                }

                                nf = (Pfct)nn->tp;

                                if (f->body && nf->body) {
                                        error("two definitions of overloaded%n",nn)
                                        Cdcl = odcl;
                                        return 0;
                                }

                                if (f->body) goto bdbd;

                                goto stst;
                        }
                        else if (nf->base != FCT) {
                                error("%n declared both as%t and asF",this,nf);
                                f->body = 0;
                        }
                        else if (can_overload) {
                                if (nf->check(f,OVERLOAD) || vrp_equiv) {
                                        if (f->body && n_qualifier) {
                                                error("badAT for%n",nn);
                                                Cdcl = odcl;
                                                return 0;
                                        }
                                        Pgen g = new gen(string);
                                        Pname n1 = g->add(nn,in_class_dcl);
                                        Pname n2 = g->add(this,0);
/*error('d',"n1%n n2%n\n",n1,n2);*/
                                        nn->tp = (Ptype)g;
                                        nn->string = g->string;
                                        nn = n2;
                                        goto thth;
                                }
                                if (in_class_dcl) {
                                        error("two declarations of%n",this);
                                        f->body = 0;
                                        Cdcl = odcl;
                                        return 0;
                                }

                                if (nf->body && f->body) {
                                        error("two definitions of%n",this);
                                        f->body = 0;
                                        Cdcl = odcl;
                                        return 0;
                                }

                                if (f->body) goto bdbd;

                                goto stst;
                        }
                        else if (nf->check(f,0)) {
                                switch (n_oper) {
                                case CTOR:
                                case DTOR:
                                        f->s_returns = nf->s_returns;
                                }
                                error("%nT mismatch:%t and%t",this,nf,f);
                                f->body = 0;
                        }
                        else if (nf->body && f->body) {
                                error("two definitions of%n",this);
                                f->body = 0;
                        }
                        else if (f->body) {
                                Pname a1, a2;
                        bdbd:
                                if (f->nargs_known && nf->nargs_known)
                                for (a1=f->argtype, a2=nf->argtype; a1; a1=a1->n_li
                                        int i1 = a1->n_initializer || a1->n_evaluat
                                        int i2 = a2->n_initializer || a2->n_evaluat
                                        if (i1) {
                                                if (i2
                                                && (     a1->n_evaluated==0
                                                        ||a2->n_evaluated==0
                                                        || a1->n_val!=a2->n_val)
                                                )
                                                        error("twoIrs for%nA%n",nn,
                                        }
                                        else if (i2) {
                                                a1->n_initializer = a2->n_initializ
                                                a1->n_evaluated = a2->n_evaluated;
                                                a1->n_val = a2->n_val;
                                        }
                                }
                                f->f_virtual = nf->f_virtual;
                                f->f_this = nf->f_this;
/*fprintf(stderr,"bdbd %s: f %d inl %d nf %d inl %d\n",string,f,f->f_inline,nf,nf->f
                                        nn->tp = f;
                                        if (f->f_inline) {
                                                if (nf->f_inline==0 && nn->n_used) error("%
                                                nf->f_inline = 1;
                                                nn->n_sto = STATIC;
                                        }
                                        else if (nf->f_inline) {
                                                /*error("%n defined as inline but not decla
                                                f->f_inline = 1;
                                        }
                                        goto stst2;
                                }
                                else {  /* two declarations */
                                        Pname a1, a2;
                                        f->f_this = nf->f_this;
                                stst:
                                        if (f->nargs_known && nf->nargs_known)
                                        for (a1=f->argtype, a2=nf->argtype; a1; a1=a1->n_li
                                                int i1 = a1->n_initializer || a1->n_evaluat
                                                int i2 = a2->n_initializer || a2->n_evaluat
                                                if (i1) {
                                                        if (i2) {
                                                                if (a1->n_evaluated==0
                                                                || a2->n_evaluated==0
                                                                || a1->n_val!=a2->n_val)
                                                                        error("twoIrs for%n
                                                        }
                                                        else if (class_name)
                                                                error("defaultA for%n",nn);
                                                }
                                                else if (i2) {
                                                        a1->n_initializer = a2->n_initializ
                                                        a1->n_evaluated = a2->n_evaluated;
                                                        a1->n_val = a2->n_val;
                                                }
                                        }
                                stst2:
                                        if (f->f_inline) n_sto = STATIC;
                                        if (n_sto) {
                                                if (nn->n_scope!=n_sto && f->f_inline==0)
                                                        error("%n both%k and%k",this,n_sto,
                                        }
                                        else {
                                                if (nn->n_scope==STATIC && n_scope==EXTERN)
                                        }
                                        n_scope = nn->n_scope; /* first specifier wins */
                                /*      n_sto = nn->n_sto;*/

                                }
                        /*      ((Pfct)nn->tp)->nargs_known = nf->nargs_known;  */
                        }
                        else {  /* new function: make f_this for member functions */
                        thth:
                                just_made = 1;
                                if (f->f_inline) nn->n_sto = STATIC;
/*fprintf(stderr,"thth %s: f %d nn->tp %d inl %d\n",string,f,nn->tp,f->f_inline);*/
```

```
                if (class_name && etbl!=gtbl) { /* beware of implicit decla
                        Pname cn = nn->n_table->t_name;
                        Pname tt = new name("this");
                        tt->n_scope = ARG;
                        tt->n_sto = REGISTER;
                        tt->tp = ((Pclass)class_name->tp)->this_type;
                        PERM(tt);
                        ((Pfct)nn->tp)->f_this = f->f_this = tt;
                        tt->n_list = f->argtype;
                }

                if (f->f_virtual) {
                        switch (nn->n_scope) {
                        default:
                                error("nonC virtual%n",this);
                                break;
                        case 0:
                        case PUBLIC:
                                cc->cot->virt_count = 1;
                                ((Pfct)nn->tp)->f_virtual = 1;
                                break;
                        }
                }
        }

        /*      an operator must take at least one class object or
                reference to class object argument
        */
        switch (n_oper) {
        case CTOR:
                if (f->nargs == 1) {    /* check for X(X) and X(X&) */
                        Ptype t = f->argtype->tp;
                cl11:
                        switch (t->base) {
                        case TYPE:
                                t = ((Pbase)t)->b_name->tp;
                                goto cl11;
                        case RPTR:                      /* X(X&) ? */
                                t = ((Pptr)t)->typ;
                        cx11:
                                switch (t->base) {
                                case TYPE:
                                        t = ((Pbase)t)->b_name->tp;
                                        goto cx11;
                                case COBJ:
                                        if (class_name == ((Pbase)t)->b_nam
                                                ((Pclass)class_name->tp)->i
                                }
                                break;
                        case COBJ:                      /* X(X) ? */
                                if (class_name == ((Pbase)t)->b_name)
                                        error("impossible constructor: %s(%
                        }
                }
                break;
        case TYPE:
```

```
/*error('d',"just_made %d %n",just_made,this);*/
                        if (just_made) {
                                nn->n_list = ((Pclass)class_name->tp)->conv;
                                ((Pclass)class_name->tp)->conv = nn;
                        }
                        break;
                case DTOR:
                case NEW:
                case DELETE:
                case CALL:
                case 0:
                        break;
                default:
                        if (f->nargs_known != 1) {
                                error("ATs must be fully specified for%n",nn);
                        }
                        else if (class_name == 0) {
                                Pname a;
                                switch (f->nargs) {
                                case 1:
                                case 2:
                                        for (a=f->argtype; a; a=a->n_list) {
                                                Ptype tx = a->tp;
                                                if (tx->base == RPTR) tx = ((Pptr)t
                                                if (tx->is_cl_obj()) goto cok;
                                        }
                                        error("%n must take at least oneCTA",nn);
                                        break;
                                default:
                                        error("%n must take 1 or 2As",nn);
                                }
                        }
                        else {
                                switch (f->nargs) {
                                case 0:
                                case 1:
                                        break;
                                default:
                                        error("%n must take 0 or 1As",nn);
                                }
                        }
                cok:;
                }

                /*
                        the body cannot be checked until the name
                        has been checked and entered into its table
                */
                if (f->body) f->dcl(nn);
                break;
        }

        case FIELD:
        {       Pbase fld = (Pbase)tp;
                char x;
                if (cc->not==0 || cc->cot->csu==UNION) {
                        if (cc->not)
                                error("field in union");
                        else
                                error("field not inC");
                        PERM(tp);
                        Cdcl = odcl;
                        return this;
                }

                if (string) {
                        nn = tbl->insert(this,0);
                        n_table = nn->n_table;
                        if (Nold) error("twoDs of field%n",this);
                }

                tp->dcl(tbl);
                if (fld->b_bits == 0) { /* force word alignment */
                        int b;
                        if (bit_offset)
                                fld->b_bits = BI_IN_WORD - bit_offset;
                        else if (b = byte_offset%SZ_WORD)
                                fld->b_bits = b * BI_IN_BYTE;
                }
                x = bit_offset += fld->b_bits;
                if (BI_IN_WORD < x) {
                        fld->b_offset = 0;
                        byte_offset += SZ_WORD;
                        bit_offset = fld->b_bits;
                }
                else {
                        fld->b_offset = bit_offset;
                        if (BI_IN_WORD == x) {
                                bit_offset = 0;
                                byte_offset += SZ_WORD;
                        }
                        else
                                bit_offset = x;
                }
                n_offset = byte_offset;
                break;
        }

        case COBJ:
        {       Pclass cl = (Pclass) ((Pbase)tp)->b_name->tp;
/*fprintf(stderr,"COBJ %d %s -> (%d %d)\n",tp,((Pbase)tp)->b_name->string,cl,cl->bas
                if (cl->csu == ANON) {  /* export member names to enclosing scope *
                        Pname nn;
                        int i;
                        int uindex;
                        Ptable mtbl = cl->memtbl;
                        char* p = cl->string;

                        if (tbl == gtbl) error('s',"global anonomous union");
                        while (*p++ != 'C');    /* UGH!!! */
                        uindex = str_to_int(p);
                        for ( nn=mtbl->get_mem(i=1); nn; nn=mtbl->get_mem(++i) ) {
                                Ptable tb = nn->n_table;
                                nn->n_table = 0;
                                Pname n = tbl->insert(nn,0);
                                n->n_union = uindex;
                                nn->n_table = tb;
                        }
                }
                goto cde;
        }

        case VEC:
        case PTR:
        case RPTR:
                tp->dcl(tbl);

        default:
        cde:
                nn = tbl->insert(this,0);

                n_table = nn->n_table;
/*error('d',"Nold %d tbl %d nn %d%n tp%t",Nold,tbl,nn,nn,nn->tp);*/
                if (Nold) {
                        if (nn->tp->base == ANY) goto zzz;
                        if (tp->check(nn->tp,0)) {
                                error("twoDs of%n;Ts:%t and%t",this,nn->tp,tp);
                                Cdcl = odcl;
                                return 0;
                        }

                        if (n_sto && n_sto!=nn->n_scope)
                                error("%n both%k and%k",this,n_sto,nn->n_scope);
                        else if (nn->n_scope==STATIC && n_scope==EXTERN)
                                error("%n both%k and%k",this,n_sto,nn->n_scope);
                        else if (nn->n_scope == STATIC)
                                error("static%n declared twice",this);

                /*      n_sto = nn->n_sto;      first scope specifier wins */
                        n_scope = nn->n_scope;

                        switch (scope) {
                        case FCT:
                                if (nn->n_stclass==STATIC && n_stclass==STATIC) bre
                                error("twoDs of%n",this);
                                Cdcl = odcl;
                                return 0;
                        case ARG:
                                error("two arguments%n",this);
                                Cdcl = odcl;
                                return 0;
                        case 0:
                        case PUBLIC:
                                error("twoDs ofM%n",this);
                                Cdcl = odcl;
                                return 0;
                        }
```

```
/* n_val */
                        if (n_initializer) {
                                if (nn->n_initializer) error("twoIrs for%n",this);
                                nn->n_initializer = n_initializer;
                        }
                }

        zzz:
                if (base != TNAME) {
                        Ptype t = nn->tp;
/*fprintf(stderr,"tp %d %d nn->tp %d %d\n",tp,tp->base,nn->tp,nn->tp?nn->tp->base:0)
                        switch (nn->n_stclass) {
                        default:
                                switch (t->base) {
                                case FCT:
                                case OVERLOAD:
                                        break;
                                default:
                                {       int x = t->align();
                                        int y = t->tsizeof();

                                        if (max_align < x) max_align = x;

                                        while (0 < bit_offset) {
                                                byte_offset++;
                                                bit_offset -= BI_IN_BYTE;
                                        }
                                        bit_offset = 0;

                                        if (byte_offset && 1<x) byte_offset = ((byt
                                        nn->n_offset = byte_offset;
                                        byte_offset += y;
                                }
                                }
                                break;
                        case STATIC:
                                switch (t->base) {
                                case FCT:
                                case OVERLOAD:
                                        break;
                                default:
                                        t->tsizeof();   /* check that size is known
                                }
                                break;
                        }
                }

        {       Ptype t = nn->tp;
                int const_old = const_save;
                bit vec_seen = 0;
                Pexpr init = n_initializer;

                if (init) {
                        switch (n_scope) {
                        case 0:
                        case PUBLIC:
                                if (n_stclass!=STATIC) error("Ir forM%n",this);
                                break;
                        }
                }

        /*      if (n_scope == EXTERN) break;           */

        lll:
                switch (t->base) {
                case RPTR:
/*fprintf(stderr,"RPTR init=%d\n",init);*/
                        if (init) {
                                init = init->typ(tbl);
                                nn->n_initializer = n_initializer = ref_init((Pptr)
                                nn->assign();
                        }
                        else {
                                switch (nn->n_scope) {
                                default:
                                        error("unId reference%n",this);
                                        break;
                                case ARG:
                                case PUBLIC:
                                case 0:
                                        break;
                                }
                        }
                        break;
                case COBJ:
/*fprintf(stderr,"COBJ %s init=%d scope %d n_scope %d\n",string,init,scope,nn->n_sco
                                        /*      TEMPORARY fudge
                                                to allow initialization of
                                                global objects
                                        */
                        if (init && st_init==0)
                                switch (nn->n_scope) {
                                case EXTERN:
                                case STATIC:
                                        if (init->base == ILIST) goto str;
                                }
                {       Pname cn = ((Pbase)t)->b_name;
                        Pclass cl = (Pclass)cn->tp;
                        Pname ctor = cl->has_ctor();
                        Pname dtor = cl->has_dtor();
                        if (dtor) {
                                Pstmt dls;
                                switch ( nn->n_scope ) {
                                case EXTERN:
                                        if (n_sto==EXTERN) break;
                                case STATIC:
                                        if (st_init==0) {
                                                if (ctor==0) error('s',"static0 %n
                                                break;
                                        }
                                        if (vec_seen) { /*  _vec_delete(vec,noe,sz,
                                                int esz = cl->tsizeof();
                                                Pexpr noe = new expr(IVAL, (Pexpr)(
                                                Pexpr sz = new expr(IVAL,(Pexpr)esz
                                                Pexpr arg = new expr(ELIST,dtor,0);
                                                dtor->lval(ADDROF);
                                                arg = new expr(ELIST,sz,arg);
                                                arg = new expr(ELIST,noe,arg);
                                                arg = new expr(ELIST,nn,arg);
                                                arg = new call(vec_del_fct,arg);
                                                arg->base = G_CALL;
                                                arg->fct_name = vec_del_fct;
                                                dls = new estmt(SM,nn->where,arg,0)
                                        }
                                        else {  /* nn->cl.~cl(0); */
                                                Pref r = new ref(DOT,nn,dtor);
                                                Pexpr ee = new expr(ELIST,zero,0);
                                                Pcall dl = new call(r,ee);
                                                dls = new estmt(SM,nn->where,dl,0);
                                                dl->base = G_CALL;
                                                dl->fct_name = dtor;
                                        }
                                        if (st_dlist) dls->s_list = st_dlist;
                                        st_dlist = dls;
                                }
                        }
                        if (ctor)       {
                                Pexpr oo = (vec_seen) ? nn->contents() : nn;
/*error('d',"ctor init=%d n_scope=%d",init,nn->n_scope);*/
                                switch (nn->n_scope) {
                                case EXTERN:
                                        if (init==0 && n_sto==EXTERN) goto ggg;
                                case STATIC:
                                        if (st_init==0) {
                                                error('s',"static0%n ofC%n that has
                                                nn->n_initializer = n_initializer =
                                                goto ggg;
                                        }
                                default:
                                        if (vec_seen && init) error("Ir forC0%n\[\]
                                        break;
                                case ARG:
                                        if (init == 0) goto ggg;
                                case PUBLIC:
                                case 0:
                                        init = new texpr(VALUE,cl,0);
                                        init->e2 = oo;
                                        nn->n_initializer = n_initializer = init =
                                        goto ggg;
                                }
                                const_save = 1;
                                nn->assign();
                                if (init) {
                                        if (init->base==VALUE && init->tp2==cl) {
                                                init->e2 = oo;
                                                init = init->typ(tbl);
                                        }
                                        else {
                                                init = init->typ(tbl);
                                                init = class_init(nn,nn->tp,init,tb
                                        }
                                }
                                else {
                                        init = new texpr(VALUE,cl,0);
                                        init->e2 = oo;
                                        init = init->typ(tbl);
                                }
                                if (init && st_init) {
                                        switch (nn->n_scope) {
                                        case EXTERN:
                                        case STATIC:
                                                if (vec_seen) { /*  _vec_new(vec,no
                                                        Pname c = cl->has_ictor();
                                                        if (c == 0) error("vector o
                                                        int esz = cl->tsizeof();
                                                        Pexpr noe = new expr(IVAL,(
                                                        Pexpr sz = new expr(IVAL,(P
                                                        Pexpr arg = new expr(ELIST,
                                                        c->lval(ADDROF);
                                                        arg = new expr(ELIST,sz,arg
                                                        arg = new expr(ELIST,noe,ar
                                                        arg = new expr(ELIST,nn,arg
                                                        init = new call(vec_new_fct
                                                        init->base = G_CALL;
                                                        init->fct_name = vec_new_fc
                                                }
                                        {       Pstmt ist = new estmt(SM,nn->where,
                                                static Pstmt itail = 0;
                                                if (st_ilist == 0)
                                                        st_ilist = ist;
                                                else
                                                        itail->s_list = ist;
                                                itail = ist;
                                                init = 0;
                                        }
                                        }
                                }
                                nn->n_initializer = n_initializer = init;
                                const_save = const_old;
                        }
                        else if (init == 0)             /* no initializer */
                                goto str;
                        else if (cl->is_simple())       /* struct */
                                goto str;
                        else {                          /* bitwise copy ok? */
                                init = init->typ(tbl);
                                if ( nn->tp->check(init->tp,ASSIGN)==0 )
                                        goto str;
                                else
                                        error("cannotI%n:C %s has privateMs but no
                        }
                        break;
                }
                case VEC:
                        t = ((Pvec)t)->typ;
                        vec_seen = 1;
                        goto lll;
                case TYPE:
                        t = ((Pbase)t)->b_name->tp;
                        goto lll;
                default:
                str:
                        if (init == 0) {
                                switch (n_scope) {
                                case ARG:
                                case 0:
                                case PUBLIC:
                                        break;
                                default:
                                        if (n_sto!=EXTERN && t->tconst())
                                                error('w',"unId const%n",this);
                                }

                                break;
                        }

                        const_save = const_save || n_scope==ARG || (t->tconst() &&
                        nn->n_initializer = n_initializer = init = init->typ(tbl);
                        if (const_save) PERM(init);
                        nn->assign();
                        const_save = const_old;

                        switch (init->base) {
                        case ILIST:
                                new_list(init);
                                list_check(nn,nn->tp,0);
                                if (next_elem()) error("IrL too long");
                                break;
                        case STRING:
                                if (nn->tp->base == VEC) {
                                        Pvec v = (Pvec)nn->tp;
                                        if (v->typ->base == CHAR) {
                                        /*      error('w',"\"char[] = string\"");*/
                                                v->size = Pvec(init->tp)->size;
                                                break;
                                        }
                                }
                        default:
                        {       Ptype nt = nn->tp;

                                if (vec_seen) {
                                        error("badIr for vector%n",nn);
                                        break;
                                }
                        tlx:
                                switch (nt->base) {
                                case TYPE:
                                        nt = ((Pbase)nt)->b_name->tp;
                                        goto tlx;
                                case INT:
```

```
                        case CHAR:
                        case SHORT:
                                if (init->base==ICON && init->tp==long_type
                                        error('w',"longIr constant for%k%n"
                        case LONG:
                                if (((Pbase)nt)->b_unsigned
                                && init->base==UMINUS
                                && init->e2->base==ICON)
                                        error('w',"negativeIr for unsigned%
                                if ( ((Pbase)nt)->b_const ) {
                                        int i;
                                        Neval = 0;
                                        i = init->eval();
                                        if (Neval == 0) {
                                                DEL(init);
                                                nn->n_evaluated = n_evaluat
                                                nn->n_val = n_val = i;
                                                nn->n_initializer = n_initi
                                        /*      if (i) {
                                                        nn->n_initializer =
                                                        nn->n_val = i;
                                                        n_initializer = 0;
                                                        n_val = i;
                                                }
                                                else {
                                                        nn->n_initializer =
                                                        n_initializer = zer
                                                }
                                        */
                                        }
                                }
                                goto cvcv;
                case PTR:
                {       Pfct ef = (Pfct)((Pptr)nt)->typ;
                        if (ef->base == FCT) {
                                Pfct f;
                                Pname n = 0;
                                switch (init->base) {
                                case NAME:
                                        f = (Pfct)init->tp;
                                        n = Pname(init);
                                        switch (f->base) {
                                        case FCT:
                                        case OVERLOAD:
                                                init = new expr(G_ADDROF,0,init);
                                                init->tp = f;
                                        }
                                        goto ad;
                                case DOT:
                                case REF:
                                        f = (Pfct) init->mem->tp;
                                        switch (f->base) {
                                        case FCT:
                                        case OVERLOAD:
                                                n = Pname(init->mem);
                                                init = new expr(G_ADDROF,0,init);
                                                init = init->typ(tbl);
                                        }
                                        goto ad;
                                case ADDROF:
                                case G_ADDROF:
                                        f = (Pfct)init->e2->tp;
                                ad:
                                        if (f->base == OVERLOAD) {
                                                Pgen g = (Pgen)f;
                                                n = g->find(ef);
                                                if (n == 0) {
                                                        error("cannot deduceT for &
                                                }
                                                init->e2 = n;
                                                n_initializer = init;
                                                n->lval(ADDROF);
                                                goto stgg;
                                        }
                                        if (n) n->lval(ADDROF);
                                }
                        }
                }
                                }
        cvcv:
                {       Pname cn;
                        int i;
                        if ((cn=init->tp->is_cl_obj())
                        && (i=can_coerce(nt,init->tp))
                        && Ncoerce) {
                                if (1 < i) error("%d possible conversions forIr");
/*error('d',"dcl %t<-%t",nt,init->tp);*/
                                Pclass cl = (Pclass)cn->tp;
                                Pref r = new ref(DOT,init,Ncoerce);
                                Pexpr c = new expr(G_CALL,r,0);
                                c->fct_name = Ncoerce;
                                c->tp = nt;
                                n_initializer = c;
                                goto stgg;
                        }
                }
                                if (nt->check(init->tp,ASSIGN))
                                        error("badIrT%t for%n (%tX)",init->tp,this,
                        else {
                        stgg:
                                if (init && n_stclass== STATIC) {
                                        /* check if non-static variables are used *
                                        /* INCOMPLETE */
                                        switch (init->base) {
                                        case NAME:
                                                if (init->tp->tconst()==0) error("v
                                                break;
                                        case DEREF:
                                        case DOT:
                                        case REF:
                                        case CALL:
                                        case G_CALL:
```

```
                                                                error("%k inIr of static%n",init->b
                                                        }
                                        }
                                        }
                                }
                                } /* switch */
                        } /* block */
                        } /* default */

                        } /* switch */
ggg:
        PERM(nn);
        switch (n_scope) {
        case FCT:
                nn->n_initializer = n_initializer;
                break;
        default:
        {/*     Pexpr ii = nn->n_initializer;*/
                Ptype t = nn->tp;
        /*      if (ii) PERM(ii);*/
        px:
                PERM(t);
                switch (t->base) {
                case PTR:
                case RPTR:      t = ((Pptr)t)->typ; goto px;
                case VEC:       t = ((Pvec)t)->typ; goto px;
                case TYPE:      t = ((Pbase)t)->b_name->tp; goto px;
                case FCT:       t = ((Pfct)t)->returns; goto px; /* args? */
                }
        }
        }

        Cdcl = odcl;
        return nn;
}
int inline_restr;       /* report use of constructs that the inline expanded cannot
                           handle here
                        */

void fct.dcl(Pname n)
{
        int nmem = TBLSIZE;
        Pname a;
        Pname ll;
        Ptable ftbl;

        Pptr cct = 0;
        int const_old = const_save;

        int bit_old = bit_offset;
        int byte_old = byte_offset;
        int max_old = max_align;
        int stack_old = stack_size;

        if (base != FCT) error('i',"fct.dcl(%d)",base);
        if (body==0 || body->memtbl) error('i',"fct.dcl(body=%d)",body);
```

```
	if (n==0 || n->base!=NAME) error('i',"fct.dcl(name=%d %d)",n,(n)?n->base:0)

	body->memtbl = ftbl = new table(nmem+3,n->n_table,0);
	body->own_tbl = 1;

	max_align = AL_FRAME;
	stack_size = byte_offset = SZ_BOTTOM;
	bit_offset = 0;

	cc->stack();
	cc->nof = n;
	cc->ftbl = ftbl;

	switch (n->n_scope) {
	case 0:
	case PUBLIC:
		cc->not = n->n_table->t_name;
		cc->cot = (Pclass)cc->not->tp;
		cc->tot = cc->cot->this_type;
		if (f_this==0 || cc->tot==0) error('i',"fct.dcl(%n): f_this=%d cc->
		f_this->n_table = ftbl;		/* fake for inline printout */
		cc->c_this = f_this;

	}

	Pname ax;
	for (a=argtype, ll=0; a; a=ax) {
		ax = a->n_list;
		Pname nn = a->dcl(ftbl,ARG);
		nn->n_assigned_to = nn->n_used = nn->n_addr_taken = 0;
		nn->n_list = 0;
		switch (a->tp->base) {
		case CLASS:
		case ENUM:	/* unlink types declared in arg list */
			a->n_list = dcl_list;
			dcl_list = a;
			break;
		default:
			if (ll)
				ll->n_list = nn;
			else {
				argtype = nn;
				if (f_this) f_this->n_list = argtype;
			}
			ll = nn;
			delete a;
		}
	}


	/* handle initializer for base class constructor */
	if (n->n_oper == CTOR) {
		Pname bn = cc->cot->clbase;

		if (bn) {
			Pclass bcl = (Pclass)bn->tp;
```

```
                        Pname bnw = bcl->has_ctor();

                        if (bnw) {
                                Ptype bnwt = bnw->tp;
                                Pfct bnwf = (Pfct) ((bnwt->base==FCT) ? bnwt : ((Pg
                                Ptype ty = bnwf->f_this->tp;
                                Pexpr v = new texpr(VALUE,bcl,f_init);
                                Pexpr th = new texpr(CAST,ty,f_this);
                                v->e2 = new expr(DEREF,th,0);
                                const_save = 1;
                                f_init = v->typ(ftbl);
                                const_save = const_old;
                        }
                        else if (f_init)
                                error(0,"unXAL: noBC constructor");
                }
                else if (f_init)
                        error( "unXAL: noBC" );
        }
        else if (f_init)
                error(0,"unXAL: not a constructor");

        PERM(returns);
        if (returns->base != VOID) {
                Pname rv = new name("_result");
                rv->tp  = returns;
                ftbl->insert(rv,0);
                delete rv;
        }

        const_save = f_inline?1:0;
        inline_restr = 0;
        body->dcl(ftbl);
        if( f_inline && inline_restr ) {
                f_inline = 0;
                error( 'w', "\"inline\" ignored, %n contains%s%s%s%s",n,
                        (inline_restr & 8) ? " loop" : "",
                        (inline_restr & 4) ? " switch" : "",
                        (inline_restr & 2) ? " goto" : "",
                        (inline_restr & 1) ? " label" : "" );
        }
        const_save = const_old;

        if (f_inline) {
                isf_list = new name_list(n,isf_list);
        }

        defined = 1;

        frame_size = stack_size + SZ_TOP;
        frame_size = ((frame_size-1)/AL_FRAME)*AL_FRAME+AL_FRAME;
        bit_offset = bit_old;
        byte_offset = byte_old;
        max_align = max_old;
        stack_size = stack_old;
```

```
	cc->unstack();
}
```

```
/* %Z% %M% %I% %H% %T% */
/*************************************************************************

        C++ source for cfront, the C++ compiler front-end
        written in the computer science research center of Bell Labs

        Copyright (c) 1984 AT&T Technologies, Inc. All rigths Reserved
        THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

        If you ignore this notice the ghost of Ma Bell will haunt you forever.

dcl2.c:

*************************************************************************/

#include "cfront.h"
#include "size.h"

Pname classdef.has_ictor()
/*
        does this class have a constructor taking no arguments?
 */
{
        Pname c = has_ctor();
        Pfct f;
        Plist l;

        if (c == 0) return 0;

        f = (Pfct)c->tp;

        switch (f->base) {
        default:
                error('i',"%s: bad constructor (%k)",string,c->tp->base);

        case FCT:
                switch (f->nargs) {
                case 0:         return c;
                default:        if (f->argtype->n_initializer) return c;
                }
                return 0;

        case OVERLOAD:
                for (l=((Pgen)f)->fct_list; l; l=l->l) {
                        Pname n = l->f;
                        f = (Pfct)n->tp;
                        switch (f->nargs) {
                        case 0:         return n;
                        default:        if (f->argtype->n_initializer) return n;
                        }
                }
                return 0;
        }
}

gen.gen(char* s)
```

```
{
        char * p = new char[ strlen(s)+1 ];
        base = OVERLOAD;
        strcpy(p,s);
        string = p;
        fct_list = 0;
}

Pname gen.add(Pname n,int sig)
/*
        add "n" to the tail of "fct_list"
        (overloaded names are searched in declaration order)

        detect:         multiple identical declarations
                        declaration after use
                        multiple definitions
*/
{
        Pfct f = (Pfct)n->tp;
        Pname nx;

        if (f->base != FCT) error(0,"%n: overloaded non-F",n);

        if ( fct_list && (nx=find(f)) ) {
/*
                Pfct nf = (Pfct)nx->tp;

                if (nf->body) {
                        if (f->body) error("two definitions for overloaded%n",n);
                }
                else {
                        if (f->body) nf->body = f->body;
                }
*/
                Nold = 1;
        }
        else {
                char* s = string;

                if (fct_list || sig) {
                        char buf[128];
                        char* bb = n->tp->signature(buf);
                        int l1 = strlen(s);
                        int l2 = bb-buf-1;
                        char* p = new char[l1+l2+1];
                        strcpy(p,s);
                        strcpy(p+l1,buf);
                        n->string = p;
                }
                else
                        n->string = s;

                nx = new name;
                *nx = *n;
                PERM(nx);
                Nold = 0;
                if (fct_list) {
                        Plist gl;
                        for (gl=fct_list; gl->l; gl=gl->l) ;
                        gl->l = new name_list(nx,0);
                }
                else
                        fct_list = new name_list(nx,0);
                nx->n_list = 0;
        }
        return nx;
}

Pname gen.find(Pfct f)
{
        Plist gl;

        for (gl=fct_list; gl; gl=gl->l) {
                Pname nx = gl->f;
                Pfct fx = (Pfct)nx->tp;
                Pname a, ax;
/*fprintf(stderr,"find %s\n",nx->string); fflush(stderr);*/

                if (fx->nargs_known != f->nargs_known) continue;

                for (ax=fx->argtype, a=f->argtype; a&&ax; ax=ax->n_list, a=a->n_lis
/*fprintf(stderr,"ax %d %d a %d %d\n",ax->tp,ax->tp->base,a->tp,a->tp->base); fflush
                        Ptype at = ax->tp;
                        if ( at->check(a->tp,0) || vrp_equiv ) goto xx;
                        switch (at->base) {
                        case CHAR:
                        case SHORT:
                        case INT:
                        case LONG:
                                if (((Pbase)at)->b_unsigned ^ ((Pbase)a->tp)->b_uns
                        }
                }

                if (ax) {
                        if (ax->n_initializer)
                                error("Ir makes overloaded %s() ambiguous",string);
                        continue;
                }

                if (a) {
                        if (a->n_initializer)
                                error("Ir makes overloaded %s() ambiguous",string);
                        continue;
                }

                if ( fx->returns->check(f->returns,0) )
                        error("two different return valueTs for overloaded %s: %t a

                return nx;
        xx:;
        }
        return 0;
}

void classdef.dcl(Pname cname, Ptable tbl)
{
        int nmem;
        Pname p;
        Pptr cct;
        Pbase bt;
        Pname px;
        Ptable btbl;
        int bvirt;
        Pclass bcl;
        int i;

        int byte_old = byte_offset;
        int bit_old = bit_offset;
        int max_old = max_align;
        int boff;

        int in_union;
        int usz;

        /* this is the place for paranoia */
        if (this == 0) error('i',"0->Cdef.dcl(%d)",tbl);
        if (base != CLASS) error('i',"Cdef.dcl(%d)",base);
        if (cname == 0) error('i',"unNdC");
        if (cname->tp != this) error('i',"badCdef");
        if (tbl == 0) error('i',"Cdef.dcl(%n,0)",cname);
        if (tbl->base != TABLE) error('i',"Cdef.dcl(%n,tbl=%d)",cname,tbl->base);

        nmem = pubmem->no_of_names() + privmem->no_of_names() + pubdef->no_of_names
        in_union = (csu==UNION || csu==ANON);

        if (clbase) {
                if (clbase->base != TNAME) error("BC%nU",clbase);
                clbase = ((Pbase)clbase->tp)->b_name;
                bcl = (Pclass)clbase->tp;
                if (bcl->defined == 0) error("BC%nU",clbase);
                tbl = bcl->memtbl;
                if (tbl->base != TABLE) error('i',"badBC table %d",tbl);
                btbl = tbl;
                bvirt = bcl->virt_count;
                if (bcl->csu == UNION) error('s',"C derived from union");
                if (in_union)
                        error("derived union");
                else
                        csu = (pubbase) ? bcl->csu : CLASS;
                boff = bcl->tsizeof();
                max_align = bcl->align();
        }
        else {
                btbl = 0;
                bvirt = 0;
                boff = 0;
                if (!in_union) csu = (virt_count) ? CLASS : STRUCT;
                while (tbl!=gtbl && tbl->t_name) tbl = tbl->next; /* nested classes
                max_align = AL_STRUCT;
        }

        memtbl->set_scope(tbl);
        memtbl->set_name(cname);
        if (nmem) memtbl->grow((nmem<=2)?3:nmem);

        cc->stack();
        cc->not = cname;
        cc->cot = this;

        byte_offset = usz = boff;
        bit_offset = 0;

        bt = new basetype(COBJ,cname);
        bt->b_table = memtbl;
        this_type = cc->tot = cct = new ptr(PTR,bt,0);
        PERM(cct);
        PERM(bt);

        for (p=privmem; p; p=px) {
                Pname m;
                px = p->n_list;
                if (p->tp->base==FCT) {
                        Pfct f = (Pfct)p->tp;
                        Pblock b = f->body;
                        f->body = 0;
                        switch( p->n_sto ) {
                        case AUTO:
                        case STATIC:
                        case REGISTER:
                        case EXTERN:
                                error("M%n cannot be%k",p,p->n_sto);
                                p->n_sto = 0;
                        }
                        m =  p->dcl(memtbl,0);
                        if (b) {
                                if (m->tp->defined)
                                        error("two definitions of%n",m);
                                else if (p->where.line!=m->where.line)
                                        error('s',"previously declared%n cannot be
                                else
                                        ((Pfct)m->tp)->body = b;
                        }
                }
                else {
                        m = p->dcl(memtbl,0);
                        if (m) {
                                if (m->n_stclass==STATIC
                                && m->n_initializer)
                                        error('s',"staticM%n withIr",m);
                                if (in_union) {
                                        if (usz < byte_offset) usz = byte_offset;
                                        byte_offset = 0;
                                }
                                }
                        }
                }
                if (privmem && csu==STRUCT) csu = CLASS;

                for (p=pubmem; p; p=px) {
                        Pname m;
                        px = p->n_list;
                        if (p->tp->base == FCT) {
                                Pfct f = (Pfct)p->tp;
                                Pblock b = f->body;
                                f->body = 0;
                                switch(p->n_sto) {
                                case AUTO:
                                case STATIC:
                                case REGISTER:
                                case EXTERN:
                                        error("M%n cannot be%k",p,p->n_sto);
                                        p->n_sto = 0;
                                }
                                m = p->dcl(memtbl,PUBLIC);
                                if (b) {
                                        if (m->tp->defined)
                                                error("two definitions of%n",m);
                                        else if (p->where.line!=m->where.line)
                                                error('s',"previously declared%n cannot be
                                        else
                                                ((Pfct)m->tp)->body = b;
                                }
                        }
                        else {
                                m = p->dcl(memtbl,PUBLIC);
                                if (m) {
                                        if (m->n_stclass==STATIC
                                        && m->n_initializer)
                                                error('s',"staticM%n withIr",m);
                                        if (in_union) {
                                                if (usz < byte_offset) usz = byte_offset;
                                                byte_offset = 0;
                                        }
                                }
                        }
                        /*delete p;*/
                }
/*              pubmem = 0;
*/
                if (in_union) byte_offset = usz;

                if (virt_count || bvirt) {      /* assign virtual indices */
                        Pname vp[100];
                        Pname nn;

                        nn = has_ctor();
                        if (nn==0 || nn->n_table!=memtbl)
                                error('s',"C%n with virtual but no constructor",cname);
                {       /*      FUDGE vtbl
                                so that the name can be used in initializers
                        */
                        char* s = new char[20];
                        sprintf(s,"%s__vtbl",string);
                        Pname n = new name(s);
                        n->tp = Pfctvec_type;
                        Pname nn = gtbl->insert(n,0);
                        nn->use();
                }

                if (virt_count = bvirt)
                        for (i=0; i<bvirt; i++) vp[i] = bcl->virt_init[i];

for ( nn=memtbl->get_mem(i=1); nn; nn=memtbl->get_mem(++i) ) {
        switch (nn->tp->base) {
        case FCT:
        {       Pfct f = (Pfct)nn->tp;
                if (bvirt) {
                        Pname vn = btbl->look(nn->string,0);
                        if (vn) {       /* match up with base class */
                                if (vn->n_table==gtbl) goto vvv;
                                Pfct vnf;
                                switch (vn->tp->base) {
                                case FCT:
                                        vnf = (Pfct)vn->tp;
                                        if (vnf->f_virtual) {
                                                if (vnf->check(f,0)) error("virtual
                                                f->f_virtual = vnf->f_virtual;
                                                vp[f->f_virtual-1] = nn;
                                        }
                                        else
                                                goto vvv;
                                        break;
                                case OVERLOAD:
                                {       Pgen g = (Pgen)vn->tp;
                                        if (f->f_virtual
                                        || ((Pfct)g->fct_list->f->tp)->f_virtual)
                                                error('s',"virtual%n overloaded inB
                                        break;
                                }
                                default:
                                        goto vvv;
                                }
                        }
                        else
                                goto vvv;
                }
                else {
                vvv:
/*error('d',"vvv: %n f_virtual %d virt_count %d",nn,f->f_virtual,virt_count);*/
                        if (f->f_virtual)  {
                                f->f_virtual = ++virt_count;
                                switch (f->f_virtual) {
                                case 1:
                                {       Pname vpn = new name("_vptr");
```

```
                                                vpn->tp = Pfctvec_type;
                                                (void) vpn->dcl(memtbl,PUBLIC);
                                                delete vpn;
                                        }
                                        default:
                                                vp[f->f_virtual-1] = nn;
                                        }
                                }
                        }
                        break;
                }

                case OVERLOAD:
                {       Plist gl;
                        Pgen g = (Pgen)nn->tp;
/*error('d',"overload%n bvirt==%d",nn,bvirt);*/
                        if (bvirt) {
                                Pname vn = btbl->look(nn->string,0);
                                Pgen g2;
                                Pfct f2;
                                if (vn) {
/*error('d',"vn%n tp%k",vn,vn->tp->base);*/
                                        if (vn->n_table == gtbl) goto ovvv;
                                        switch (vn->tp->base) {
                                        default:
                                                goto ovvv;
                                        case FCT:
                                                f2 = (Pfct)vn->tp;
                                                if (f2->f_virtual
                                                || ((Pfct)g->fct_list->f->tp)->f_virtual)
                                                        error('s',"virtual%n overloaded in
                                                break;
                                        case OVERLOAD:
                                                g2 = (Pgen)vn->tp;

                                                for (gl=g->fct_list; gl; gl=gl->l) {
                                                        Pname fn = gl->f;
                                                        Pfct f = (Pfct)fn->tp;
                                                        Pname vn2 = g2->find(f);

                                                        if (vn2 == 0) {
                                                                if (f->f_virtual) error('s'
                                                        }
                                                        else {
                                                                Pfct vn2f = (Pfct)vn2->tp;
                                                                if (vn2f->f_virtual) {
                                                                        f->f_virtual = vn2f
                                                                        vp[f->f_virtual-1]
                                                                }
                                                        }
                                                }
                                                break;
                                        }
                                }
                                else
                                        goto ovvv;
                }
                else {
                ovvv:
                        for (gl=g->fct_list; gl; gl=gl->l) {
                                Pname fn = gl->f;
                                Pfct f = (Pfct)fn->tp;

/*fprintf(stderr,"fn %s f %d %d %d count %d\n",fn->string,f,f->base,f->f_virtual,vir
                                if (f->f_virtual) {
                                        f->f_virtual = ++virt_count;
                                        switch (f->f_virtual) {
                                        case 1:
                                        {       Pname vpn = new name("_vptr");
                                                vpn->tp = Pfctvec_type;
                                                (void) vpn->dcl(memtbl,0);
                                                delete vpn;
                                        }
                                        default:
                                                vp[f->f_virtual-1] = fn;
                                        }
                                }
                        }
                }
                break;
        }
}
                }
                virt_init = new Pname[virt_count];
                for (i=0; i<virt_count; i++) virt_init[i] = vp[i];
        }

        for (p=pubdef, pubdef=0; p; p=p->n_list) {
                char* qs = p->n_qualifier->string;
                char* ms = p->string;
                Pname cx;
                Ptable ctbl;
                Pname mx;

                if (strcmp(ms,qs)==0) ms = "_ctor";

                for (cx = clbase; cx; cx = ((Pclass)cx->tp)->clbase) {
                        if (strcmp(cx->string,qs) == 0) goto ok;
                }
                error("publicQr %s not aBC",qs);
                continue;
        ok:
                ctbl = ((Pclass)cx->tp)->memtbl;
                mx = ctbl->lookc(ms,0);

                if (Ebase) {
                        if (!Ebase->has_friend(cc->nof)) error("QdMN%n is in privat
                }
                else if (Epriv) {
                        if (!Epriv->has_friend(cc->nof)) error("QdMN%n is private",
                }
                if (mx == 0) {
```

```
				error("C%n does not have aM %s",cx,p->string);
				p->tp = any_type;
			}
			else {
				if (mx->tp->base==OVERLOAD)
				   error('s',"public specification of overloaded%n",mx);
				p->base = PUBLIC;
			}

			p->n_qualifier = mx;
			(void) memtbl->insert(p,0);
			if (Nold) error("twoDs of CM%n",p);
		}

		if (bit_offset) byte_offset += SZ_WORD;
		if (byte_offset < SZ_STRUCT) {
			Pname n = new name("_dummy");
			switch (SZ_STRUCT-obj_size) {
			case 1:		n->tp = char_type; break;
			case 2:		n->tp = char2_type; break;
			case 3:		n->tp = char3_type; break;
			case 4:		n->tp = char4_type; break;
			default:	n->tp = new vec(char_type,0);
					Pvec(n->tp)->size = SZ_STRUCT-obj_size;
			}
			(void) n->dcl(memtbl,0);
			delete n;
/*error('d',"dummy bo=%d",byte_offset);*/
		}
		int waste = byte_offset%max_align;
		if (waste) {	/* fudge, ensure derived class get right sizeof */
			waste = max_align-waste;
/*error('d',"%s: waste %d tbl=%d",string,waste,memtbl);*/
			Pname n = new name("_waste");
			switch (waste) {
			case 1:		n->tp = char_type; break;
			case 2:		n->tp = char2_type; break;
			case 3:		n->tp = char3_type; break;
			case 4:		n->tp = char4_type; break;
			default:	n->tp = new vec(char_type,0);
					Pvec(n->tp)->size = waste;
			}
			(void) n->dcl(memtbl,0);
			delete n;
			if (byte_offset%max_align) error('i',"failed to align %s",string);
		}
/*error('d',"sz=%d al=%d",byte_offset,max_align);*/
		obj_size = byte_offset;
		obj_align = max_align;

		if ( has_dtor() && has_ctor()==0)
			error('w',"%s has destructor but no constructor",string);

		if ( itor==0 && has_oper(ASSIGN) )
			error('w',"%s has assignment defined but not initialization (no %s(
	defined = 1;

	for (p=memtbl->get_mem(i=1); p; p=memtbl->get_mem(++i)) {
	/* define members defined inline */
		switch (p->tp->base) {
		case FCT:
		{	Pfct f = (Pfct)p->tp;
			if (f->body) {
				f->f_inline = 1;
				p->n_sto = STATIC;
				f->dcl(p);
			}
			break;
		}
		case OVERLOAD:
		{	Pgen g = (Pgen)p->tp;
			Plist gl;
			for (gl=g->fct_list; gl; gl=gl->l) {
				Pname n = gl->f;
				Pfct f = (Pfct)n->tp;
				if (f->body) {
					f->f_inline = 1;
					n->n_sto = STATIC;
					f->dcl(n);
				}
			}
		}
		}
	}

	Plist fl;				/* define friends defined inline */
	for (fl=friend_list; fl; fl=fl->l) {
		Pname p = fl->f;
		switch (p->tp->base) {
		case FCT:
		{	Pfct f = (Pfct)p->tp;
			if (f->body && f->defined==0) {
				f->f_inline = 1;
				p->n_sto = STATIC;
				f->dcl(p);
			}
			break;
		}
		case OVERLOAD:
		{	Pgen g = (Pgen)p->tp;
			Plist gl;
			for (gl=g->fct_list; gl; gl=gl->l) {
				Pname n = gl->f;
				Pfct f = (Pfct)n->tp;
				if (f->body && f->defined==0) {
					f->f_inline = 1;
					n->n_sto = STATIC;
					f->dcl(n);
				}
			}
		}
```

```
                }
        }

        byte_offset = byte_old;
        bit_offset = bit_old;
        max_align = max_old;

        cc->unstack();
}

void enumdef.dcl(Pname, Ptable tbl)
{
#define FIRST_ENUM 0
        int nmem = mem->no_of_names();
        Pname p;
        Pname ns = 0;
        Pname nl;
        int enum_old = enum_count;
        no_of_enumerators = nmem;

        enum_count = FIRST_ENUM;

        if (this == 0) error('i',"0->enumdef.dcl(%d)",tbl);

        for(p=mem, mem=0; p; p=p->n_list) {
                Pname nn;
                if (p->n_initializer) {
                        Pexpr i = p->n_initializer->typ(tbl);
                        Neval = 0;
                        enum_count = i->eval();
                        if (Neval) error("%s",Neval);
                        DEL(i);
                        p->n_initializer = 0;
                }
                p->n_evaluated = 1;
                p->n_val = enum_count++;
                nn = tbl->insert(p,0); /* ??? */
                if (Nold) {
                        if (nn->n_stclass == ENUM) {
                                if (p->n_val != nn->n_val) error("twoDs of enum con
                        }
                        else
                                error("incompatibleDs of%n",nn);
                }
                else {
                        nn->n_stclass = ENUM; /* no store will be allocated */
                        if (ns)
                                nl->n_list = nn;
                        else
                                ns = nn;
                        nl = nn;
                }
                delete p;
        }

        mem = ns;
        enum_count = enum_old;
        defined = 1;
}
/*
void fct.dcl(Ptable tbl)

        The argument names are placed in the memtable of the body.
        This makes
                f(int a) { int a; };
        illegal

        The argument names/types remain linked even after they are entered
        into the symbol table,
        but class and enum declarations are unlinked
{
        int nmem = TBLSIZE;
        Pname a;
        Pname ll;
        int bit_old = bit_offset;
        int byte_old = byte_offset;
        int max_old = max_align;
        int stack_old = stack_size;

        if (base != FCT) error('i',"fct.dcl(%d)",base);
        if (body==0 || body->memtbl) error('i',"fct.dcl(%d)",body);
        if (tbl->base != TABLE) error('i',"fct.dcl(tbl=%d)",tbl->base);

        body->memtbl = new table(nmem,tbl,0);
        body->own_tbl = 1;

        max_align = AL_FRAME;
        stack_size = byte_offset = SZ_BOTTOM;
        bit_offset = 0;

        for (a=argtype, ll=0; a; a=a->n_list) {
                Pname n = a->dcl(body->memtbl,ARG);
                n->n_list = 0;
                switch (a->tp->base) {
                case CLASS:
                case ENUM:
                        break;
                default:
                        if (ll)
                                ll->n_list = n;
                        else
                                argtype = n;
                        ll = n;
                }
        }

        frame_size = stack_size + SZ_TOP;
        frame_size = ((frame_size-1)/AL_FRAME)*AL_FRAME+AL_FRAME;
        bit_offset = bit_old;
        byte_offset = byte_old;
        max_align = max_old;
```

```
	stack_size = stack_old;
}*/

Pstmt curr_loop;
Pstmt curr_switch;
Pblock curr_block;

void stmt.reached()
{
	register Pstmt ss = s_list;

	if (ss == 0) return;

	switch (ss->base) {
	case LABEL:
	case CASE:
	case DEFAULT:
		break;
	default:
		error('w',"statement not reached");
		/* delete unreacheable code */
		for (; ss; ss=ss->s_list) {
			switch (ss->base) {
			case LABEL:
			case CASE:
			case DEFAULT:	/* reachable */
				s_list = ss;
				return;
			case IF:
			case DO:
			case WHILE:
			case SWITCH:
			case FOR:
			case BLOCK:	/* may hide a label */
				s_list = ss;
				return;
			}
		}
		s_list = 0;
	}
}

bit arg_err_suppress;

Pexpr check_cond(Pexpr e, TOK b, Ptable)
{
	Pname cn;
	if (cn = e->tp->is_cl_obj()) {
		Pclass cl = (Pclass)cn->tp;
		int i = 0;
		Pname found = 0;
		for (Pname on = cl->conv; on; on=on->n_list) {
			Pfct f = (Pfct)on->tp;
			Ptype t = f->returns;
		xx:
			switch (t->base) {
```

```
                        case TYPE:
                                t = ((Pbase)t)->b_name->tp;
                                goto xx;
                        case CHAR:
                        case SHORT:
                        case INT:
                        case LONG:
                        case EOBJ:
                        case FLOAT:
                        case DOUBLE:
                        case PTR:
                                i++;
                                found = on;
                        }
                }
                switch (i) {
                case 0:
                        error("%n0 in%k expression",cn,b);
                        return e;
                case 1:
                {
/*error('d',"cond%t<-%t",((Pfct)found->tp)->returns,e->tp);*/
                        Pclass cl = (Pclass)cn->tp;
                        Pref r = new ref(DOT,e,found);
                        Pexpr c = new expr(G_CALL,r,0);
                        c->fct_name = found;
                        c->tp = ((Pfct)found->tp)->returns;
                        return c;
                }
                default:
                        error("%d possible conversions for%n0 in%k expression",i,cn
                        return e;
                }

        }
        e->tp->num_ptr(b);
        return e;
}

void stmt.dcl()
/*
        typecheck statement "this" in scope "curr_block->tbl"
*/
{
        Pstmt ss;
        Pname n;
        Pname nn;
        Pstmt ostmt = Cstmt;

        for (ss=this; ss; ss=ss->s_list) {
                Pstmt old_loop, old_switch;
                Cstmt = ss;
                Ptable tbl = curr_block->memtbl;
/*error('d',"ss %d%k tbl %d e %d%k s %d%k sl %d%k", ss, ss->base, tbl, ss->e, (ss->e
                switch (ss->base) {
                case BREAK:
```

```
				if (curr_loop==0 && curr_switch==0)
					error("%k not in loop or switch",BREAK);
				ss->reached();
				break;

			case CONTINUE:
				if (curr_loop == 0) error("%k not in loop",CONTINUE);
				ss->reached();
				break;

			case DEFAULT:
				if (curr_switch == 0) {
					error("default not in switch");
					break;
				}
				if (curr_switch->has_default) error("two defaults in switch
				curr_switch->has_default = ss;
				ss->s->s_list = ss->s_list;
				ss->s_list = 0;
				ss->s->dcl();
				break;

			case SM:
				ss->e = (ss->e != dummy) ? ss->e->typ(tbl) : 0;
				break;

			case DELETE:
			{	int i;
				ss->e = ss->e->typ(tbl);
				i = ss->e->tp->num_ptr(DELETE);
				if (i != P) error("nonP deleted");
				break;
			}

			case RETURN:
			{	Pname fn = cc->nof;
				Ptype rt = ((Pfct)fn->tp)->returns;
				Pexpr v = ss->e;
				if (v != dummy) {
					if (rt->base == VOID)
						error('w',"unX return value");
					else {
						v = v->typ(tbl);
					lx:
						switch (rt->base) {
						case TYPE:
							rt = ((Pbase)rt)->b_name->tp;
							goto lx;
						case RPTR:
							ss->e = ref_init((Pptr)rt,v,tbl);
							break;
						case COBJ:
						{	Pname rv = tbl->look("_result",0);
							ss->e = class_init(rv,rt,v,tbl);
							break;
						}
```

```
                                case ANY:
                                        break;
                                case INT:
                                case CHAR:
                                case LONG:
                                case SHORT:
                                        if (((Pbase)rt)->b_unsigned
                                        && v->base==UMINUS
                                        && v->e2->base==ICON)
                                                error('w',"negative retured
                                default:
                                        ss->e = v;
                                        if (rt->check(v->tp,ASSIGN))
                                                error("bad return valueT fo
                                }
                        }
                }
                else {
                        if (rt->base != VOID) error('w',"return valueX");
                }
                ss->reached();
                break;
        }

        case DO:        /* in DO the stmt is before the test */
                old_loop = curr_loop;
                curr_loop = ss;
                if (ss->s->base == DCL) error('s',"D as onlyS in do-loop");
                ss->s->dcl();
        /*      tbl = curr_block->memtbl;*/
                ss->e = ss->e->typ(tbl);
                ss->e = check_cond(ss->e,DO,tbl);
                curr_loop = old_loop;
                break;

        case WHILE:
                inline_restr |= 8;
                old_loop = curr_loop;
                curr_loop = ss;
                ss->e = ss->e->typ(tbl);
                /*ss->e->tp->num_ptr(ss->base);*/
                ss->e = check_cond(ss->e,WHILE,tbl);
                if (ss->s->base == DCL) error('s',"D as onlyS in while-loop
                ss->s->dcl();
                curr_loop = old_loop;
                break;

        case SWITCH:
        {       int ne = 0;
                inline_restr |= 4;
                old_switch = curr_switch;
                curr_switch = ss;
                ss->e = ss->e->typ(tbl);
        /*      ss->e->tp->num_ptr(SWITCH);*/
                ss->e = check_cond(ss->e,SWITCH,tbl);
                {       Ptype tt = ss->e->tp;
```

```
                        sii:
                                switch (tt->base) {
                                case TYPE:
                                        tt = ((Pbase)tt)->b_name->tp; goto sii;
                                case EOBJ:
                                        ne = Penum(Pbase(tt)->b_name->tp)->no_of_en
                                case ZTYPE:
                                case ANY:
                                case CHAR:
                                case SHORT:
                                case INT:
                                case LONG:
                                        break;
                                default:
                                        error('s',"%t switch expression",ss->e->tp)
                                }
                        }
                        ss->s->dcl();
                        if (ne) {       /* see if the number of cases is "close to"
                                           but not equal to the number of enumerato
                                        */
                                int i = 0;
                                Pstmt cs;
                                for (cs=ss->case_list; cs; cs=cs->case_list) i++;
                                if (i && i!=ne) {
                                        if (ne < i) {
                                ee:             error('w',"switch (%t) with %d case
                                        }
                                        else {
                                                switch (ne-i) {
                                                case 1: if (3<ne) goto ee;
                                                case 2: if (7<ne) goto ee;
                                                case 3: if (23<ne) goto ee;
                                                case 4: if (60<ne) goto ee;
                                                case 5: if (99<ne) goto ee;
                                                }
                                        }
                                }
                        }
                        curr_switch = old_switch;
                        break;
                }
                case CASE:
                        if (curr_switch == 0) {
                                error("case not in switch");
                                break;
                        }
                        ss->e = ss->e->typ(tbl);
                        ss->e->tp->num_ptr(CASE);
                        {       Ptype tt = ss->e->tp;
                        iii:
                                switch (tt->base) {
                                case TYPE:
                                        tt = ((Pbase)tt)->b_name->tp; goto iii;
                                case ZTYPE:
                                case ANY:
```

```
                        case CHAR:
                        case SHORT:
                        case INT:
                        case LONG:
                                break;
                        default:
                                error('s',"%t case expression",ss->e->tp);
                        }
                }
                if (1) {
                        Neval = 0;
                        int i = ss->e->eval();
                        if (Neval == 0) {
                                Pstmt cs;
                                for (cs=curr_switch->case_list; cs; cs=cs->
                                        if (cs->case_value == i) error("cas
                                }
                                ss->case_value = i;
                                ss->case_list = curr_switch->case_list;
                                curr_switch->case_list = ss;
                        }
                }
                if (ss->s->s_list) error('i',"case%k",ss->s->s_list->base);
                ss->s->s_list = ss->s_list;
                ss->s_list = 0;
                ss->s->dcl();
                break;

        case GOTO:
                inline_restr |= 2;
                ss->reached();
        case LABEL:
                /* Insert label in function mem table;
                   labels have function scope.
                */
                n = ss->d;
                nn = cc->ftbl->insert(n,LABEL);

                /* Set a ptr to the mem table corresponding to the scope
                   in which the label actually occurred.  This allows the
                   processing of goto's in the presence of ctors and dtors
                */
                if(ss->base == LABEL) {
                        nn->n_realscope = curr_block->memtbl;
                        inline_restr |= 1;
                }

                if (Nold) {
                        if (ss->base == LABEL) {
                                if (nn->n_initializer) error("twoDs of labe
                                nn->n_initializer = (Pexpr)1;
                        }
                        if (n != nn) ss->d = nn;
                }
                else {
                        if (ss->base == LABEL) nn->n_initializer = (Pexpr)1
```

```
				nn->where = ss->where;
			}
			if (ss->base == GOTO)
				nn->use();
			else {
				if (ss->s->s_list) error('i',"label%k",ss->s->s_lis
				ss->s->s_list = ss->s_list;
				ss->s_list = 0;
				nn->assign();
			}
			if (ss->s) ss->s->dcl();
			break;

		case IF:
		{	Pexpr ee = ss->e->typ(tbl);
			if (ee->base == ASSIGN) {
				Neval = 0;
				(void)ee->e2->eval();
				if (Neval == 0) error('w',"constant assignment in c
			}
			ss->e = ee = check_cond(ee,IF,tbl);
			switch (ee->tp->base) {
			case INT:
			case ZTYPE:
			{	int i;
				Neval = 0;
				i = ee->eval();
				if (Neval == 0) {
/*fprintf(stderr,"if (%d) %d %d\n",i,ss->e,ss->e->base);*/
					Pstmt sl = ss->s_list;
					if (i) {
						DEL(ss->else_stmt);
						ss->s->dcl();
						*ss = *ss->s;
					}
					else {
						DEL(ss->s);
						if (ss->else_stmt) {
							ss->else_stmt->dcl();
							*ss = *ss->else_stmt;
						}
						else {
							ss->base = SM;
							ss->e = dummy;
							ss->s = 0;
						}
					}
					ss->s_list = sl;
					continue;
				}
			}
			}
			ss->s->dcl();
			if (ss->else_stmt) ss->else_stmt->dcl();
			break;
		}
		case FOR:
			inline_restr |= 8;
			old_loop = curr_loop;
			curr_loop = ss;
			if (ss->for_init) {
				Pstmt fi = ss->for_init;
				switch (fi->base) {
				case SM:
					if (fi->e == dummy) {
						ss->for_init = 0;
						break;
					}
				default:
					fi->dcl();
					break;
				case DCL:
					fi->dcl();
/*error('d',"dcl=>%k",fi->base);*/
					switch (fi->base) {
					case BLOCK:
					{
					/* { ... for( { a } b ; c) d ; e }
						=>
					   { ... { a for ( ; b ; c) d ; e }}
					*/
						Pstmt tmp = new stmt (SM,curloc,0);
						*tmp = *ss;	/* tmp = for */
						tmp->for_init = 0;
						*ss = *fi;	/* ss = { } */
						if (ss->s)
							ss->s->s_list = tmp;
						else
							ss->s = tmp;
						curr_block = (Pblock)ss;
						tbl = curr_block->memtbl;
						ss = tmp;	/* rest of for and
						break;
					}
					}
				}
			}
			if (ss->e == dummy)
				ss->e = 0;
			else {
				ss->e = ss->e->typ(tbl);
				ss->e = check_cond(ss->e,FOR,tbl);
			}
			if (ss->s->base == DCL) error('s',"D as onlyS in for-loop")
			ss->s->dcl();
			ss->e2 = (ss->e2 == dummy) ? 0 : ss->e2->typ(tbl);
			curr_loop = old_loop;
			break;

		case DCL:	/* declaration after statement */
if (0)
		{	Pname n;
```

```
                        Pexpr in;
                        if (curr_block->own_tbl==0) {
                                curr_block->memtbl = tbl = new table(8,tbl,0);
                                curr_block->own_tbl = 1;
                        }
                        Pname dd = ss->d;
                        if (dd->n_list) error('s',"list ofDs not at head of block")
                        n = dd->dcl(tbl,FCT);
                        in = n->n_initializer;
                        ss->base = SM;
                        if (n->n_stclass == STATIC && in) {
                                error('s',"Id static not at head of block");
                                goto dum;
                        }
                        Pname cln = n->tp->is_cl_obj();
                        if (cln && ((Pclass)cln->tp)->has_dtor())
                                error('s',"%n ofC%n with destructor not at head of
                        if (in) {
                                n->n_initializer = 0;
                                switch (in->base) {
                                case G_CALL:    /* constructor? */
                                 {
                                        Pname fn = in->fct_name;
                                        if (fn==0 || fn->n_oper!=CTOR) goto ass;
                                        break;
                                }
                                case ILIST:
                                        error('s',"Ir list not at head of block");
                                        goto dum;
                                case STRING:
                                        n->n_initializer = in;  /* constant */
                                        goto dum;
                                default:
                                ass:
                                        in = new expr(ASSIGN,n,in);
                                }
                                ss->e = in;
                        }
                        else {
                        dum:
                                ss->e = dummy;
                        }
                        break;
                }

                {
                        /*      collect all the contiguous DCL nodes from the
                                head of the s_list. find the next statement
                        */
                        int non_trivial = 0;
                        int count = 0;
                        Pname tail = ss->d;
                        for (Pname nn=tail; nn; nn=nn->n_list) {
                                /*      find tail;
                                        detect non-trivial declarations
                                 */
                                count++;
                                if (nn->n_list) tail = nn->n_list;
                                Pname n = tbl->look(nn->string,0);
                                if (n && n->n_table==tbl) non_trivial = 2;
                                if (non_trivial) continue;
                                Pexpr in = nn->n_initializer;
/*error('d',"in %d",in);*/
                                if (in == 0) continue;
                                if (non_trivial == 0) non_trivial = 1;
                                if (nn->n_stclass==STATIC) {
                                        non_trivial = 2;
                                        continue;
                                }
                                switch (in->base) {
                                case ILIST:
                                case STRING:
                                        non_trivial = 2;
                                        continue;
                                }
                                Pname cln = nn->tp->is_cl_obj();
                                if (cln == 0) continue;
                                if (((Pclass)cln->tp)->has_dtor()) non_trivial = 2;
                        }
/*error('d',"non_trivial %d",non_trivial);*/
                        while( ss->s_list && ss->s_list->base==DCL ) {
                                Pstmt sx = ss->s_list;
                                tail = tail->n_list = sx->d;    /* add to tail */
                                for (nn=sx->d; nn; nn=nn->n_list) {
                                        /*      find tail;
                                                detect non-trivial declarations
                                        */
                                        count++;
                                        if (nn->n_list) tail = nn->n_list;
                                        Pname n = tbl->look(nn->string,0);
                                        if (n && n->n_table==tbl) non_trivial = 2;
                                        if (non_trivial) continue;
                                        Pexpr in = nn->n_initializer;
                                        if (in == 0) continue;
                                        if (non_trivial == 0) non_trivial = 1;
                                        if (nn->n_stclass==STATIC) {
                                                non_trivial = 2;
                                                continue;
                                        }
                                        switch (in->base) {
                                        case ILIST:
                                        case STRING:
                                                non_trivial = 2;
                                                continue;
                                        }
                                        Pname cln = nn->tp->is_cl_obj();
                                        if (cln == 0) continue;
                                        if (((Pclass)cln->tp)->has_dtor()) non_triv
                                }
                                ss->s_list = sx->s_list;
                        /*      delete sx;      */
                        }
                        Pstmt next_st = ss->s_list;
/*error('d',"non_trivial %d curr_block->own_tbl %d inline_restr %d",non_trivial,curr
                        if (non_trivial==2      /* must */
                        || (non_trivial==1      /* might */
                                && ( curr_block->own_tbl==0     /* just as well */
                                || inline_restr&3               /* label seen */)
                                )
                        ) {
                                /*      Create a new block,
                                        put all the declarations at the head,
                                        and the remainder of the slist as the
                                        statement list of the block.
                                */
                                ss->base = BLOCK;

                                /*      check that there are no redefinitions since
                                        "real" (user-written, non-generated) block
                                */
                                for( nn=ss->d; nn; nn=nn->n_list ) {
                                        Pname n;
                                        if( curr_block->own_tbl
                                        &&  (n=curr_block->memtbl->look(nn->string,
                                        &&  n->n_table->real_block==curr_block->mem
                                                error("twoDs of%n",n);
                                }

                                /*      attach the remainder of the s_list
                                        as the statement part of the block.
                                */
                                ss->s = next_st;
                                ss->s_list = 0;

                                /*      create the table in advance, in order to se
                                        real_block ptr to that of the enclosing tab
                                */
                                ss->memtbl = new table(count+4,tbl,0);
                                ss->memtbl->real_block = curr_block->memtbl->real_b

                                ((Pblock)ss)->dcl(ss->memtbl);
                        }
                        else {  /*      to reduce the number of symbol tables,
                                        do not make a new block,
                                        instead insert names in enclosing block,
                                        and make the initializers into expression
                                        statements.
                                */
                                Pstmt sss = ss;
                                for( nn=ss->d; nn; nn=nn->n_list ) {
                                        Pname n = nn->dcl(tbl,FCT);
/*error('d',"%n->dcl(%d) -> %d init %d sss=%d",nn,tbl,n,n->n_initializer,sss);*/
                                        if (n == 0) continue;
                                        Pexpr in = n->n_initializer;
                                        n->n_initializer = 0;
                                        if (ss) {
                                                sss->base = SM;
                                                ss = 0;
                                                }
                                                else
                                                        sss = sss->s_list = new estmt(SM,ss
                                                if (in) {
                                                        switch (in->base) {
                                                        case G_CALL:    /* constructor? */
                                                        {
                                                                Pname fn = in->fct_name;
                                                                if (fn && fn->n_oper==CTOR)
                                                        }
                                                        default:
                                                                in = new expr(ASSIGN,n,in);
                                                        }
                                                        sss->e = in->typ(tbl);
                                                }
                                                else
                                                        sss->e = dummy;
                                        }
                                        ss = sss;
                                        ss->s_list = next_st;
                                }
                                break;
                        }

                        case BLOCK:
                                ((Pblock)ss)->dcl(tbl);
                                break;

                        case ASM:
                                /* save string */
                                break;

                        default:
                                error('i',"badS(%d %d)",ss,ss->base);
                        }
                }

                Cstmt = ostmt;
}

void block.dcl(Ptable tbl)
/*
        Note: for a block without declarations memtbl denotes the table
        for the enclosing scope.
        A function body has its memtbl created by fct.dcl().
*/
{
        int bit_old = bit_offset;
        int byte_old = byte_offset;
        int max_old = max_align;
        Pblock block_old = curr_block;

        if (base != BLOCK) error('i',"block.dcl(%d)",base);

        curr_block = this;
        if (d) {
                Pname n;
                own_tbl = 1;
                if (memtbl == 0) {
                        int nmem = d->no_of_names()+4;
                        memtbl = new table(nmem,tbl,0);
                        memtbl->real_block = this;
                        /*      this is a "real" block from the
                                source text, and not one created by DCL's
                                inside a block. */
                }
                else
                        if (memtbl != tbl) error('i',"block.dcl(?)");

                Pname nx;
                for (n=d; n; n=nx) {
                        nx = n->n_list;
                        n->dcl(memtbl,FCT);
                        switch (n->tp->base) {
                        case CLASS:
                        case ANON:
                        case ENUM:
                                break;
                        default:
                                delete n;
                        }
                }
        }
        else
                memtbl = tbl;

        if (s) {
                Pname odcl = Cdcl;
                Pname m;
                int i;

                s->dcl();

                if (own_tbl)
                for (m=memtbl->get_mem(i=1); m; m=memtbl->get_mem(++i)) {
                        Ptype t = m->tp;

                        if (t == 0) {
                                if (m->n_assigned_to == 0) error('w',"undefined lab
                                if (m->n_used == 0) error('w',"label %s not used",
                                continue;
                        }
                ll:
                        switch (t->base) {
                        case TYPE:      t=((Pbase)t)->b_name->tp; goto ll;
                        case CLASS:
                        case ENUM:
                        case FCT:
                        case VEC:       continue;
                        }
                        if (m->n_addr_taken == 0) {
                                if (m->n_used) {
                                        if (m->n_assigned_to) {
                                        }
                                        else {
                                                switch (m->n_scope) {
                                                case FCT:
                                                        Cdcl = m;
                                                        error('w',"%n used but not
                                                }
                                        }
                                }
                                else {
                                        if (m->n_assigned_to) {
                                        }
                                        else {
                                                switch (m->n_scope) {
                                                case ARG:
                                                        if (m->string[0]=='_' && m-
                                                case FCT:
                                                        Cdcl = m;
                                                        error('w',"%n not used",m);
                                                }
                                        }
                                }
                        }
                }
                Cdcl = odcl;
        }

        d = 0;

        if (bit_offset) byte_offset += SZ_WORD;
        if (stack_size < byte_offset) stack_size = byte_offset;
        bit_offset = bit_old;
        byte_offset = byte_old;
        curr_block = block_old;
}

int name.no_of_names()
{
        register int i = 0;
        register Pname n;
        for (n=this; n; n=n->n_list) i++;
        return i;
}

static Pexpr lvec[20], *lll;
static Pexpr list_back = 0;
#define list_put_back(x) list_back = x;

void new_list(Pexpr lx)
{
        if (lx->base != ILIST) error('i',"IrLX");

        lll = lvec;
```

```
		lll++;
		*lll = lx->e1;
}

Pexpr next_elem()
{
		Pexpr e;
		Pexpr lx;

		if (lll == lvec) return 0;

		lx = *lll;

		if (list_back) {
			e = list_back;
			list_back = 0;
			return e;
		}

		if (lx == 0) {				/* end of list */
			lll--;
			return 0;
		}

		switch (lx->base) {
		case ELIST:
			e = lx->e1;
			*lll = lx->e2;
			switch (e->base) {
			case ILIST:
				lll++;
				*lll = e->e1;
				return (Pexpr)1;	/* start of new ILIST */
			case ELIST:
				error("nestedEL");
				return 0;
			default:
				return e;
			}
		default:
			error('i',"IrL");
		}
}

void list_check(Pname nn, Ptype t, Pexpr il)
/*
		see if the list lll can be assigned to something of type t
		nn is the name of the variable for which the assignment is taking place.
		il is the last list element returned by next_elem()
*/
{
		Pexpr e;
		bit lst = 0;
		int i;
		Pclass cl;
```

```
	switch ( (int)il ) {
	case 0:		break;
	case 1:		lst = 1; break;
	default:	list_put_back(il);
	}

zzz:
	switch (t->base) {
	case TYPE:
		t = ((Pbase)t)->b_name->tp;
		goto zzz;

	case VEC:
	{	Pvec v = (Pvec)t;
		Ptype vt = v->typ;

		if (v->size) {  /* get at most v->size initializers */
			for (i=0; i<v->size; i++) { /* check next list element type
			ee:
				e = next_elem();

				 /* "too few" initializers are legal */
				if (e == 0) goto xsw;
			vtz:
				switch (vt->base) {
				case TYPE:
					vt = ((Pbase)vt)->b_name->tp;
					goto vtz;
				case VEC:
				case COBJ:
					list_check(nn,vt,e);
					break;
				default:
					if (e == (Pexpr)1) {
						error("unXIrL");
						goto ee;
					}
					if (vt->check(e->tp,ASSIGN))
						error("badIrT for%n:%t (%tX)",nn,e-
				}
			}
			if ( lst && (e = next_elem()) ) error("end of IrLX after ve
		xsw:;
		}
		else {		/* determine v->size */
			i = 0;
		xx:
			while ( e=next_elem() ) {	/* get another initializer
				i++;
			vtzz:
				switch (vt->base) {
				case TYPE:
					vt = ((Pbase)vt)->b_name->tp;
					goto vtzz;
				case VEC:
				case COBJ:
```

```
                                                list_check(nn,vt,e);
                                                break;
                                        default:
                                                if (e == (Pexpr)1) {
                                                        error("unXIrL");
                                                        goto xx;
                                                }
                                                if (vt->check(e->tp,ASSIGN))
                                                        error("badIrT for%n:%t (%tX)",nn,e-
                                        }
                                }
                                v->size = i;
                        }
                        break;
                }

                case CLASS:
                        cl = (Pclass)t;
                        goto ccc;

                case COBJ:      /* initialize members */
                        cl = (Pclass)((Pbase)t)->b_name->tp;
                ccc:
                {       Ptable tbl = cl->memtbl;
                        Pname m;

                        if (cl->clbase) {
                                list_check(nn,cl->clbase->tp,0);
                        }
                        for (m=tbl->get_mem(i=1); m; m=tbl->get_mem(++i)) {
                                Ptype mt = m->tp;
                                switch (mt->base) {
                                case FCT:
                                case OVERLOAD:
                                case CLASS:
                                case ENUM:
                                        continue;
                                }
                                if (m->n_stclass == STATIC) continue;
                                /* check assignment to next member */
                        dd:
                                e = next_elem();
                                if (e == 0) break;
                        mtz:
                                switch (mt->base) {
                                case TYPE:
                                        mt = ((Pbase)mt)->b_name->tp;
                                        goto mtz;
                                case CLASS:
                                case ENUM:
                                        break;
                                case VEC:
                                case COBJ:
                                        list_check(nn,m->tp,e);
                                        break;
                                default:
```

```
                                if (e == (Pexpr)1) {
                                        error("unXIrL");
                                        goto dd;
                                }
                                if (mt->check(e->tp,ASSIGN))
                                        error("badIrT for %s .%n:%t (%tX)",cl->stri
                        }
                }
                if (lst && (e = next_elem()) ) error("end of IrLX afterO");
                break;
        }
        default:
                e = next_elem();

                if (e == 0) {
                        error("noIr forO");
                        break;
                }

                if (e == (Pexpr)1) {
                        error("unXIrL");
                        break;
                }
                if (t->check(e->tp,ASSIGN))
                        error("badIrT for%n:%t (%tX)",nn,e->tp,t);
                if (lst && (e = next_elem()) ) error("end of IrLX afterO");
                break;
        }
}
```

```
/* %Z% %M% %I% %H% %T% */
/************************************************************************

	C++ source for cfront, the C++ compiler front-end
	written in the computer science research center of Bell Labs

	Copyright (c) 1984 AT&T Technologies, Inc.
		All rigths Reserved
	THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

	If you ignore this notice the ghost of Ma Bell will haunt you forever.

del.c:

	walk the trees to reclaim storage

************************************************************************/

#include "cfront.h"

void name.del()
{
/*fprintf(stderr,"%d->name.del: %s %d\n",this,(string)?string:"?",base);fflush(stder
	Pexpr i = n_initializer;

	NFn++;
	DEL(tp);
	if(i && i!=(Pexpr)1) DEL(i);
	n_tbl_list = name_free;
	name_free = this;
}

void type.del()
{
/*fprintf(stderr,"DEL(type=%d %d)\n",this,base);*/
	permanent = 3;	/* do not delete twice */
	switch (base) {
	case TNAME:
	case NAME:
		error('i',"%d->T.del():N %s %d",this,((Pname)this)->string,base);
	case TYPE:
	{	Pbase b = (Pbase)this;
		break;
	}
	case FCT:
	{	Pfct f = (Pfct) this;
		DEL(f->returns);
		/*DEL(f->argtype);
*/
		break;
	}
	case VEC:
	{	Pvec v = (Pvec) this;
		DEL(v->dim);
		DEL(v->typ);
		break;
```

```
        }
        case PTR:
        case RPTR:
        {       Pptr p = (Pptr) this;
                DEL(p->typ);
                break;
        }
/*      case CLASS:
        {       Pclass cl = (Pclass)this;
                memtbl.del();
                break;
        }
        case ENUM:
        case OVERLOAD:
                break;*/
        }

        delete this;
}

void expr.del()
{
/*fprintf(stderr,"DEL(expr=%d: %d %d %d)\n",this,base,e1,e2); fflush(stderr);*/
        permanent = 3;
        switch (base) {
        case IVAL:
                if (this == one) return;
        case ICON:
        case FCON:
        case CCON:
        case THIS:
        case STRING:
        case TEXT:
        case FVAL:
                goto dd;
        case DUMMY:
        case ZERO:
        case NAME:
                return;
        case CAST:
        case SIZEOF:
        case NEW:
        case VALUE:
                DEL(tp2);
                break;
        case REF:
        case DOT:
                DEL(e1);
                DEL(mem);
                goto dd;
        case QUEST:
                DEL(cond);
                break;
        case ICALL:
                delete il;
                goto dd;
```

```
	}

	DEL(e1);
	DEL(e2);
/*	DEL(tp);*/
dd:
	e1 = expr_free;
	expr_free = this;
	NFe++;
}

void stmt.del()
{
/*fprintf(stderr,"DEL(stmt %d %s)\n",this,keys[base]); fflush(stderr);*/
	permanent = 3;
	switch (base) {
	case SM:
	case WHILE:
	case DO:
	case DELETE:
	case RETURN:
	case CASE:
	case SWITCH:
		DEL(e);
		break;
	case PAIR:
		DEL(s2);
		break;
	case BLOCK:
		DEL(d);
		DEL(s);
		if (own_tbl) DEL(memtbl);
		DEL(s_list);
		goto dd;
	case FOR:
		DEL(e);
		DEL(e2);
		DEL(for_init);
		break;
	case IF:
		DEL(e);
		DEL(else_stmt);
		break;
	}

	DEL(s);
	DEL(s_list);
dd:
	s_list = stmt_free;
	stmt_free = this;
	NFs++;
}

void table.del()
{
	register i;
```

```
/*fprintf(stderr,"tbl.del %s %d size=%d used=%d)\n", (t_name)?t_name->string:"?", th

        for (i=1; i<free_slot; i++) {
                Pname n = entries[i];
                if (n==0) error('i',"table.del(0)");
                switch (n->n_scope) {
                case ARG:
                case ARGT:
                        break;
                default:
                {       char* s = n->string;
                        if (s && (s[0]!='_' || s[1]!='X')) delete s;
                        /* delete n; */
                        n->del();
                }
                }
        }
        delete entries;
        delete hashtbl;
        delete this;
}
```

```
/* %Z% %M% %I% %H% %T% */
/*****************************************************************************

        C++ source for cfront, the C++ compiler front-end
        written in the computer science research center of Bell Labs

        Copyright (c) 1984 AT&T Technologies, Inc. All rigths Reserved
        THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

        If you ignore this notice the ghost of Ma Bell will haunt you forever.

error.c :

        write error messages

        Until scan_started != 0 no context can be assumed

*****************************************************************************/

#include "size.h"
#include "cfront.h"

int error_count;
static int no_of_warnings;
char scan_started;

#define ERRTRACE    20

static char * abbrev_tbl['Z'+1];

extern void error_init();
void error_init()
{
        static char errbuf[BUFSIZ];
        setbuf(stderr,errbuf);

        abbrev_tbl['A'] = " argument";
        abbrev_tbl['B'] = " base";
        abbrev_tbl['C'] = " class";
        abbrev_tbl['D'] = " declaration";
        abbrev_tbl['E'] = " expression";
        abbrev_tbl['F'] = " function";
        abbrev_tbl['I'] = " initialize";
        abbrev_tbl['J'] = " J";
        abbrev_tbl['K'] = " K";
        abbrev_tbl['L'] = " list";
        abbrev_tbl['M'] = " member";
        abbrev_tbl['N'] = " name";
        abbrev_tbl['O'] = " object";
        abbrev_tbl['P'] = " pointer";
        abbrev_tbl['Q'] = " qualifie";
        abbrev_tbl['R'] = " R";
        abbrev_tbl['S'] = " statement";
        abbrev_tbl['T'] = " type";
        abbrev_tbl['U'] = " undefined";
        abbrev_tbl['V'] = " variable";
```

```
	abbrev_tbl['W'] = " W";
	abbrev_tbl['X'] = " expected";
	abbrev_tbl['Y'] = " Y";
	abbrev_tbl['Z'] = " Z";

}

#define INTERNAL 127

void ext(int n)
/*
	remove temp_file and exit
*/
{
/*	if (n==INTERNAL) abort();*/
	exit(n);
}

static void print_loc()
{
	class loc * sl = (Cstmt) ? &Cstmt->where : 0;
	class loc * dl = (Cdcl) ? &Cdcl->where : 0;

	if (sl && dl && sl->file==dl->file) {
		if (sl->line<=dl->line)
			dl->put(out_file);
		else
			sl->put(out_file);
		return;
	}

	if (sl) {
		if (sl->file == curloc.file) {
			sl->put(out_file);
			return;
		}
	}

	if (dl) {
		if (dl->file == curloc.file) {
			dl->put(out_file);
			return;
		}
	}

	curloc.put(out_file);
}

static void print_context()
{
	putc('\n',out_file);
}

static char in_error = 0;
loc dummy_loc;
```

```
void yyerror(char* s)
{
        error(0,&dummy_loc,s);
}

int error(char* s ...)
{
        register* a = (int*)&s;
        return error(0,&dummy_loc, s, a[1], a[2], a[3], a[4], a[5], a[6], a[7], a[8
}

int error(int t, char* s ...)
{
        register* a = (int*)&s;
        return error(t,&dummy_loc, s, a[1], a[2], a[3], a[4], a[5], a[6], a[7], a[8
}

int error(loc* l, char* s ...)
{
        register* a = (int*)&s;
        return error(0, l, s, a[1], a[2], a[3], a[4], a[5], a[6], a[7], a[8] );
}

int error(int t, loc* lc, char* s ...)
/*
        "int" not "void" because of "pch" in lex.c
        subsequent arguments fill in %mumble fields

        legal error types are:
                'w'             warning  (not counted in error count)
                'd'             debug
                's'             "not implemented" message
                0               error
                'i'             internal error (causes abort)
                't'             error while printing error message
*/
{
        FILE * of = out_file;
        int c;
        char format[3]; /* used for "% mumble" sequences */
        int * a = &t;
        int argn = 3;

        /* check variable argument passing mechanism */
        int si = sizeof(int);
        int scp = sizeof(char*);
        int ssp = sizeof(Pname);

        if (si!=ssp || si!=scp || ssp!=scp || &a[2]!=(int*)&s) {
                fprintf(stderr,
                        "\n%s: this c can't handle varargs (%d,%d,%d -- %d %d)\n",
                        prog_name, si, scp, ssp, &a[1], &s);
                ext(12);
        }

        if (t == 'w' && warn==0) return 0;
        if (in_error++)
                if (t!='t' || 4<in_error) {
                        fprintf(stderr,"\nUPS!, error while handling error\n");
                        ext(13);
                }
        else if (t == 't')
                t = 'i';

        out_file = stderr;
        if (!scan_started)
                /*fprintf(out_file,"error during %s initializing: ",prog_name);*/
                putch('\n');
        else if (t=='t')
                putch('\n');
        else if (lc != &dummy_loc)
                lc->put(out_file);
        else
                print_loc();

    switch (t) {
        case 0:
                fprintf(out_file,"error: ");
                break;
        case 'w':
                no_of_warnings++;
                fprintf(out_file,"warning: ");
                break;
        case 's':
                fprintf(out_file,"sorry, not implemented: ");
                break;
        case 'i':
                if (error_count) {
                        fprintf(out_file,"sorry, %s cannot recover from earlier err
                        ext(INTERNAL);
                }
                else
                        fprintf(out_file,"internal %s error: ",prog_name);
                break;
        }

    while (c = *s++) {
        if ('A'<=c && c<='Z' && abbrev_tbl['A'])
                putstring(abbrev_tbl[c]);
        else if (c == '%')
                switch (c = *s++) {
                case 'k':
                {       TOK x = a[argn];
                        if (0<x && x<MAXTOK && keys[x])
                                fprintf(out_file," %s",keys[x]);
                        else
                                fprintf(out_file," token(%d)",x);
                        argn++;
                        break;
                }
                case 't':       /* Ptype */
                {       Ptype tt = (Ptype)a[argn];
                        if (tt) {
                                TOK pm = print_mode;
                                extern int ntok;
                                int nt = ntok;
                                print_mode = ERROR;
                                fprintf(out_file," ");
                                tt->dcl_print(0);
                                print_mode = pm;
                                ntok = nt;
                                argn++;
                        }
                        break;
                }
                case 'n':       /* Pname */
                {       Pname nn = (Pname)a[argn];
                        if (nn) {
                                TOK pm = print_mode;
                                print_mode = ERROR;
                                fprintf(out_file," ");
                                nn->print();
                                print_mode = pm;
                        }
                        else
                                fprintf(out_file," ?");
                        argn++;
                        break;
                }
                default:
                        format[0] = '%';
                        format[1] = c;
                        format[2] = '\0';
                        fprintf(out_file,format,a[argn++]);
                        break;
                }
                else
                        putch(c);
        }

        if (!scan_started) ext(4);

        switch (t) {
        case 'd':
        case 't':
        case 'w':
                putch('\n');
                break;
        default:
                print_context();
        }
        fflush(stderr);
    /* now we may want to carry on */

        out_file = of;

        switch (t) {
```

```
        case 't':
                if (--in_error) return 0;
        case 'i':
                ext(INTERNAL);
        case 0:
        case 's':
                if (MAXERR<++error_count) {
                        fprintf(stderr,"Sorry, too many errors\n");
                        ext(7);
                }
        }

        in_error = 0;
        return 0;
}
```

```
/* %Z% %M% %I% %H% %T% */
/**************************************************************************

	C++ source for cfront, the C++ compiler front-end
	written in the computer science research center of Bell Labs

	Copyright (c) 1984 AT&T Technologies, Inc. All rigths Reserved
	THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

	If you ignore this notice the ghost of Ma Bell will haunt you forever.

expand.c:

	expand inline functions

**************************************************************************/

#include "cfront.h"

char* temp(char* vn, char* fn, char* cn)
/*
	make the name of the temporary: _X_vn_fn_cn
*/
{	if (vn[0]!='_' || vn[1]!='X') {
		int vnl = strlen(vn);
		int fnl = strlen(fn);
		int cnl = (cn)?strlen(cn):0;
		char* s = new char[vnl+fnl+cnl+6];

		s[0] = '_';
		s[1] = 'X';
		strcpy(s+2,vn);
		s[vnl+2] = '_';
		strcpy(s+vnl+3,fn);
		if (cnl) {
			s[vnl+fnl+3] = '_';
			strcpy(s+vnl+fnl+4,cn);
		}
		return s;
	}
	else
		return vn;

}

Pname dcl_local(Ptable scope, Pname an, Pname fn)
{
	if (scope == 0) {
		error('s',"cannot expand inlineF needing temporary variable in nonF
		return an;
	}
	if (an->n_stclass == STATIC) error('s',"static%n in inlineF",an);
	Pname cn = fn->n_table->t_name;
	char* s = temp(an->string,fn->string,(cn)?cn->string:0);
	Pname nx = new name(s);
/*error('d',"%n: %d->dcl_local(%s)",fn,scope,s); */
```

```
        nx->tp = an->tp;
        PERM(nx->tp);
        nx->n_used = an->n_used;
        nx->n_assigned_to = an->n_assigned_to;
        nx->n_addr_taken = an->n_addr_taken;
        Pname r = scope->insert(nx,0);
        delete nx;
        return r;
}

Pstmt stmt.expand()
/*
        copy the statements with the formal arguments replaced by ANAMES

        called once only per inline function
        expand_tbl!=0 if the function should be transformed into an expression
        and expand_tbl is the table for local variables
*/
{
        if (this == 0) error('i',"0->stmt.expand() for%n",expand_fn);
/*error('d',"stmt %d:%k s=%d e=%d l=%d",this,base,s,e,s_list);*/

        if (memtbl) {   /* check for static variables */
                register Ptable t = memtbl;
                register int i;
                for (register Pname n = t->get_mem(i=1); n; n=t->get_mem(++i))
                        if (n->n_stclass == STATIC) {
                                error('s',"static%n in inlineF",n);
                                n->n_stclass = AUTO;
                        }
        }

        if (expand_tbl) {       /* make expression */
                Pexpr ee;

                if (memtbl && base!=BLOCK) { /* temporaries */
                        int i;
                        Pname n;
                        Ptable tbl = memtbl;
                        for (n = tbl->get_mem(i=1); n; n=tbl->get_mem(++i)) {
/*error('d',"%n: %n",expand_fn,n);*/
                                Pname nn = dcl_local(scope,n,expand_fn);
                                nn->base = NAME;
                                n->string = nn->string;
                        }
                }

                switch (base) {
                default:
                        error('s',"%kS in inline%n",base,expand_fn);
                        return (Pstmt)dummy;

                case BLOCK:
                        if (s_list) {
                                ee = (Pexpr) s_list->expand();
                                if (s) {
                                        ee = new expr(CM, (Pexpr)s->expand(), ee);
                                        PERM(ee);
                                }
                                return (Pstmt) ee;
                        }

                        if (s) return s->expand();

                        return (Pstmt) zero;

                case PAIR:
                        ee = s2 ? (Pexpr)s2->expand() : 0;
                        ee = new expr(CM, s?(Pexpr)s->expand():0, ee);
                        if (s_list) ee = new expr(CM, ee, (Pexpr)s_list->expand());
                        PERM(ee);
                        return (Pstmt) ee;

                case RETURN:
                        s_list = 0;
                        return (Pstmt) e->expand();

                case SM:
                        ee = (e==0 || e->base==DUMMY) ? zero : e->expand();
                        if (s_list) {
                                ee = new expr(CM, ee, (Pexpr)s_list->expand());
                                PERM(ee);
                        }
                        return (Pstmt)ee;

                case IF:
                {       Pexpr qq = new expr(QUEST,(Pexpr)s->expand(),0);
                        qq->cond = e->expand();
                        qq->e2 = else_stmt ? (Pexpr)else_stmt->expand() : zero;
                        if (s_list) qq = new expr(CM,qq,(Pexpr)s_list->expand());
                        PERM(qq);
                        return (Pstmt)qq;
                }
                }
        }

        switch (base) {
        default:
                if (e) e = e->expand();
                break;
        case PAIR:
                if (s2) s2 = s2->expand();
                break;
        case BLOCK:
                break;
        case FOR:
                if (for_init) for_init = for_init->expand();
                if (e2) e2 = e2->expand();
                break;
        case LABEL:
        case GOTO:
        case RETURN:
```

```
        case BREAK:
        case CONTINUE:
                error('s',"%kS in inline%n",base,expand_fn);
        }

        if (s) s = s->expand();
        if (s_list) s_list = s_list->expand();
        PERM(this);
        return this;
}

Pexpr expr.expand()
{
        if (this == 0) error('i',"expr.expand(0)");
/*fprintf(stderr,"%s(): expr %d: b=%d e1=%d e2=%d\n",expand_fn->string,this,base,e1,
        switch (base) {
        case NAME:
                if (expand_tbl && ((Pname)this)->n_scope==FCT) {
                        Pname n = (Pname)this;
                        char* s = n->string;
                        if (s[0]=='_' && s[1]=='X') break;
                        Pname cn = expand_fn->n_table->t_name;
                        n->string = temp(s,expand_fn->string,(cn)?cn->string:0);
                }
        case DUMMY:
        case ICON:
        case FCON:
        case CCON:
        case IVAL:
        case FVAL:
        case LVAL:
        case STRING:
        case ZERO:
        case SIZEOF:
        case TEXT:
        case ANAME:
                break;
        case ICALL:
                if (expand_tbl && e1==0) {
                        Pname fn = il->fct_name;
                        Pfct f = (Pfct)fn->tp;
                        if (f->returns==void_type && fn->n_oper!=CTOR)
                                error('s',"non-value-returning inline%n called in v
                        else
                                error("inline%n called before defined",fn);
                }
                break;
        case QUEST:
                cond = cond->expand();
        default:
                if (e2) e2 = e2->expand();
        case REF:
        case DOT:
                if( e1 ) e1 = e1->expand();
                break;
        case CAST:
                PERM(tp2);
                e1 = e1->expand();
                break;
        }

        PERM(this);
        return this;
}

bit expr.not_simple()
/*
        is a temporary variable needed to hold the value of this expression
        as an argument for an inline expansion?
        return 1; if side effect
        return 2; if modifies expression
*/
{
        int s;
/*error('d',"not_simple%k",base);*/
        switch (base) {
        default:
                return 2;
        case ZERO:
        case IVAL:
        case FVAL:
        case ICON:
        case CCON:
        case FCON:
        case STRING:
        case NAME:      /* unsafe (alias) */
        case SIZEOF:
        case G_ADDROF:
        case ADDROF:
                return 0;
        case CAST:
        case DOT:
        case REF:
                return e1->not_simple();
        case UMINUS:
        case NOT:
        case COMPL:
                return e2->not_simple();
        case DEREF:
                s = e1->not_simple();
                if (1<s) return 2;
                if (e2==0) return s;
                s |= e2->not_simple();
                return s;
        case MUL:
        case DIV:
        case MOD:
        case PLUS:
        case MINUS:
        case LS:
        case RS:
        case AND:
```

```
        case OR:
        case ER:
        case LT:
        case LE:
        case GT:
        case GE:
        case EQ:
        case NE:
        case ANDAND:
        case OROR:
        case CM:
                s = e1->not_simple();
                if (1<s) return 2;
                s |= e2->not_simple();
                return s;
        case QUEST:
                s = cond->not_simple();
                if (1<s) return 2;
                s |= e1->not_simple();
                if (1<s) return 2;
                s |= e2->not_simple();
                return s;
        case ANAME:
                if (curr_icall) {
                        Pname n = (Pname)this;
                        int argno = n->n_val;
                        Pin il;
                        for (il=curr_icall; il; il=il->i_next)
                                if (n->n_table == il->i_table) goto aok;
                        goto bok;
                aok:
                        return (il->local[argno]) ? 0 : il->arg[argno]->not_simple(
                }
        bok:    error('i',"expand aname%n",this);
        case VALUE:
        case NEW:
        case CALL:
        case G_CALL:
        case ICALL:
        case ASSIGN:
        case INCR:
        case DECR:
        case ASPLUS:
        case ASMINUS:
        case ASMUL:
        case ASDIV:
        case ASMOD:
        case ASAND:
        case ASOR:
        case ASER:
        case ASLS:
        case ASRS:
                return 2;
        }
}
```

```
Pexpr fct.expand(Pname fn, Ptable scope, Pexpr ll)
/*
        expand call to (previously defined) inline function in "scope"
        with the argument list "ll"
        (1) declare variables in "scope"
        (2) initialize argument variables
        (3) link to body
*/
{
/*error('d',"expand%n inline=%d last_exp=%d curr_expr=%d",fn,f_inline,last_expanded,
        if ((body==0 && f_expr==0)                      /* called before defined */
        ||  (((Pfct)fn->tp)->body->memtbl==scope)       /* called while defining */
        ||  (f_inline==2)                               /* recursive call */
        ||  (last_expanded && last_expanded==curr_expr)
                                                        /* twice in an expression *
        ) {
                fn->take_addr();                        /* so don't expand */
                if (fn->n_addr_taken == 1) {
                        Pname nn = new name;            /* but declare it */
                        *nn = *fn;
                        nn->n_list = dcl_list;
                        nn->n_sto = STATIC;
                        dcl_list = nn;
                }
                return 0;
        }

        f_inline = 2;

        Pin il = new iline;
        Pexpr ic = new texpr(ICALL,0,0);
        il->fct_name = fn;
        ic->il = il;
        ic->tp = returns;
        Pname n;
        Pname at = (f_this) ? f_this : argtype;
        if (at) il->i_table = at->n_table;
        int i = 0;
        int not_simple = 0;     /* is a temporary argument needed? */

        for (n=at; n; n=n->n_list, i++) {
                /*      check formal/actual argument pairs
                        and generate temporaries as necessary
                */
                if (ll == 0) error('i',"simpl.call:AX for %n",fn);
                Pexpr ee;

                if (ll->base == ELIST) {
                        ee = ll->e1;
                        ll = ll->e2;
                }
                else {
                        ee = ll;
                        ll = 0;
                }
                int s;  /* could be avoided when expanding into a block */

                if (n->n_assigned_to == FUDGE111) {     /* constructor's this */
                        if (ee != zero) {               /* automatic or static
                                                           then we can use the
                                                           actual variable
                                                        */
                                il->local[i] = 0;
                                goto zxc;
                        }
                }
/*error('d',"n=%n addr %d ass %d used %d s %d",n,n->n_addr_taken,n->n_assigned_to,n-
                if (n->n_addr_taken
                || n->n_assigned_to) {
                        Pname nn = dcl_local(scope,n,fn);
                        nn->base = NAME;
                        il->local[i] = nn;
                        ++not_simple;
                }
                else if (n->n_used
                        && (s=ee->not_simple())
                        && (1<s || 1<n->n_used) ) {     /* not safe */
                        Pname nn = dcl_local(scope,n,fn);
                        nn->base = NAME;
                        il->local[i] = nn;
                        ++not_simple;
                }
                else
                        il->local[i] = 0;
        zxc:
                il->arg[i] = ee;
                il->tp[i] = n->tp;
        }

        if (f_expr) {   /* generate comma expression */
                char loc_var = 0;
                /* look for local variables needing declaration: */
                Ptable tbl = body->memtbl;
                for (n=tbl->get_mem(i=1); n; n=tbl->get_mem(++i) ) {
                        if (n->base==NAME
                        && (n->n_used||n->n_assigned_to||n->n_addr_taken)) {
                                Pname nn = dcl_local(scope,n,fn);
                                nn->base = NAME;
                                n->string = nn->string;
                                loc_var++;
                        }
                }
/*error('d',"not_simple=%d loc_var=%d last_expanded=%d curr_expr=%d",not_simple,loc_
                if (not_simple || loc_var) last_expanded = curr_expr;

                Pexpr ex;
                if (not_simple) {
                        Pexpr etail = ex = new expr(CM,0,0);
                        for (i=0; i<MIA; i++) {
                                Pname n = il->local[i];
                                if (n==0) continue;
                                Pexpr e = il->arg[i];
                                etail->e1 = new expr(ASSIGN,n,e);
/*error('d',"%n = %k",n,e->base);*/
                                if (--not_simple)
                                        etail = etail->e2 = new expr(CM,0,0);
                                else
                                        break;
                        }
                        etail->e2 = f_expr;
                }
                else {
                        ex = f_expr;
                }
                ic->e1 = ex;
        }
        else {  /* generate block */
                Pstmt ss;
                if (not_simple) {
                        last_expanded = curr_expr;
                        Pstmt st = new estmt(SM,curloc,0,0);
                        Pstmt stail = st;
                        for (i=0; i<MIA; i++) {
                                Pname n = il->local[i];
                                if (n == 0) continue;
                                Pexpr e = il->arg[i];
                                stail->e = new expr(ASSIGN,n,e);
                                if (--not_simple)
                                        stail = stail->s_list = new estmt(SM,curloc
                                else
                                        break;
                        }
                        stail->s_list = body;
                        ss = new block(curloc,0,st);
                }
                else {
                        ss = body;
                }
                ic->e2 = (Pexpr)ss;
        }

        f_inline = 1;
        return ic;
}
```

```
/* %Z% %M% %I% %H% %T% */
/***************************************************************************

	C++ source for cfront, the C++ compiler front-end
	written in the computer science research center of Bell Labs

	Copyright (c) 1984 AT&T Technologies, Inc. All rigths Reserved
	THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

	If you ignore this notice the ghost of Ma Bell will haunt you forever.

expr.c:

	type check expressions

****************************************************************************/

#include "cfront.h"
#include "size.h"

int const_save;

Pexpr expr.address()
{
	if (base==DEREF && e2==0) return e1;	/* &* */
	if (base == CM) {
		e2 = e2->address();
		return this;
	}
	register Pexpr ee = new expr(G_ADDROF,0,this);
	ee->tp = new ptr(PTR,tp,0);
	if (base == NAME) ((Pname)this)->take_addr();
	return ee;
}

Pexpr expr.contents()
{
	if (base==ADDROF || base==G_ADDROF) return e2;		/* *& */
	register Pexpr ee = new expr(DEREF,this,0);
	if (tp) ee->tp = ((Pptr)tp)->typ;			/* tp==0 ??? */
	return ee;
}

Pexpr table.find_name(register Pname n, bit f, Pexpr args)
/*
	find the true name for "n", implicitly define if undefined
	if "n" was called f==1 and "args" were its argument list
	if n was qualified r->n or o.n  f==2
*/
{
	Pname q = n->n_qualifier;
	register Pname qn = 0;
	register Pname nn;
	Pclass cl;	/* class specified by q */
/*if (q)
 error('d',"%d->find_name%s::%s f=%d args=%d ntbl=%d cc->tot=%d\n",this,(q!=sta_name
else
 error('d',"%d->find_name %s f=%d args=%d ntbl=%d cc->tot=%d\n",this,n->string,f,arg
        if (n->n_table) {
                nn = n;
                n = 0;
                goto xx;
        }

        if (q) {
                Ptable tbl;

                if (q == sta_name)
                        tbl = gtbl;
                else {
                        Ptype t = (Pclass)q->tp;
                        if (t == 0) error('i',"Qr%n'sT missing",q);

                        if (q->base == TNAME) {
                                if (t->base != COBJ) {
                                        error("badT%k forQr%n",t->base,q);
                                        goto nq;
                                }
                                t = ((Pbase)t)->b_name->tp;
                        }
                        if (t->base != CLASS) {
                                error("badQr%n(%k)",q,t->base);
                                goto nq;
                        }
                        cl = (Pclass)t;
                        tbl = cl->memtbl;
                }

                qn = tbl->look(n->string,0);
                if (qn == 0) {
                        n->n_qualifier = 0;
                        nn = 0;
                        goto def;
                }

                if (q == sta_name) {    /* explicitly global */
                        qn->use();
                        delete n;
                        return qn;
                }
                /* else check visibility */
        }

nq:
        if (cc->tot) {
        {       for (Ptable tbl = this;;) {
                        nn = lookc(n->string,0);
/*error('d',"cc->tot:%n nn=%n sto%k sco%k",n,nn,nn->n_stclass,nn->n_scope);*/
                        if (nn == 0) goto qq;   /* try for friend */

                        switch (nn->n_scope) {
                        case 0:
```

```
				case PUBLIC:
					if (nn->n_stclass == ENUM) break;

					if (nn->tp->base == OVERLOAD) break;

					if (Ebase
					&& Ebase!=cc->cot->clbase->tp
					&& !Ebase->has_friend(cc->nof))
						error("%n is from a privateBC",n);

					if (Epriv
					&& Epriv!=cc->cot
					&& !Epriv->has_friend(cc->nof))
						error("%n is private",n);
				}

				if (qn==0 || qn==nn) break;

				if ((tbl=tbl->next) == 0) {	/* qn/cl test necessary? */
					if ( (qn->n_stclass==STATIC
						|| qn->tp->base==FCT
						|| qn->tp->base==OVERLOAD)
					&&   (  qn->n_scope==PUBLIC
						|| cl->has_friend(cc->nof)) ) {
						/*qn->use();
						delete n;
						return qn;
						*/
						nn = qn;
						break;
					}
					else {
						error("QdN%n not in scope",n);
						goto def;
					}
				}
			}
		}
		xx:
/*error('d',"xx: nn=%n qn=%n n=%n f=%d",nn,qn,n,f);*/
			if (nn == 0) goto def;
			nn->use();
			if (f == 2) {
				if (qn && nn->n_stclass==0)
					switch (nn->n_scope) {
					case 0:
					case PUBLIC:	/* suppress virtual */
						switch (qn->tp->base) {
						case FCT:
						case OVERLOAD:
							*n = *qn;
							n->n_qualifier = q;
							return n;
						}
					}
				if (n) delete n;
```

```
                                return nn;
                        }

                        switch (nn->n_scope) {
                        case 0:
                        case PUBLIC:
                                if (nn->n_stclass == 0) {
                                        if (qn) {       /* suppress virtual */
                                                switch (qn->tp->base) {
                                                case FCT:
                                                case OVERLOAD:
                                                        *n = *qn;
                                                        n->n_qualifier = q;
                                                        /*return n; */
                                                        nn = n;
                                                        n = 0;
                                                }
                                        }

                                        if (cc->c_this == 0) {
                                                switch (nn->n_oper) {
                                                case CTOR:
                                                case DTOR:
                                                        break;
                                                default:
                                                        /* in static member initializer */
                                                        error("%n cannot be used here",nn);
                                                        return nn;
                                                }
                                        }

                                        Pref r = new ref(REF,cc->c_this,nn);
                                        cc->c_this->use();
                                        r->tp = nn->tp;
                                        if (n) delete n;
                                        return r;
                                }
                        default:
                                if (n) delete n;
                                return nn;
                        }
                }
qq:
/*error('d',"qq: n%n nn%n qn%n",n,nn,qn);*/
                if (qn) {       /* static member? */
                        if (qn->n_scope==0  && !cl->has_friend(cc->nof) ) {
                                error("%n is private",qn);
                                if (n) delete n;
                                return qn;
                        }

                        switch (qn->n_stclass) {
                        case STATIC:
                                break;
                        default:
                                switch (qn->tp->base) {
```

```
				case FCT:
				case OVERLOAD:	/* suppress virtual */
					if (f == 1) error("0 missing for%n",qn);
					*n = *qn;
					n->n_qualifier = q;
					return n;
				default:
					if (f != 2) error("0 missing for%n",qn);
				}
			}

			if (n) delete n;
			return qn;
		}

		if ( nn = lookc(n->string,0) ) {
			switch (nn->n_scope) {
			case 0:
			case PUBLIC:
				if (nn->n_stclass == ENUM) break;

				if (nn->tp->base == OVERLOAD) break;
				if (Ebase && !Ebase->has_friend(cc->nof) )
					error("%n is from privateBC",n);

				if (Epriv && !Epriv->has_friend(cc->nof) )
					error("%n is private",n);
			}
		}

		if (nn) {
			nn->use();
			if (n) delete n;
			return nn;
		}

def:	/* implicit declaration */
/*error('d',"implicit f %d",f);*/
		n->n_qualifier = 0;
		if (f == 1) {	/* function */
			if (n->tp) error('i',"find_name(fct_type?)");
			if (fct_void) {
				n->tp = new fct(defa_type,0,0);
			}
			else {
				Pexpr e;
				Pname at = 0;
				Pname att;

				for (e=args; e; e=e->e2) {
					Pname ar = new name;
					if (e->base != ELIST) error('i',"badA %k",e->base);
					e->e1 = e->e1->typ(this);
					ar->tp = e->e1->base==STRING ? Pchar_type : e->e1->
					switch (ar->tp->base) {
					case ZTYPE:
						ar->tp = defa_type;
						break;
					case FIELD:
						ar->tp = int_type;
						break;
					case ANY:
					default:
						PERM(ar->tp);
					}
					if (at)
						att->n_list = ar;
					else
						at = ar;
					att = ar;
				}
				n->tp = new fct(defa_type,at,1);

			}
		}
		else {
			n->tp = any_type;
			if (this != any_tbl)
				if (cc->not && cc->cot->defined==0)
					error("C%n isU",cc->not);
				else
					error("%n isU",n);
		}

		nn = n->dcl(gtbl,EXTERN);
		nn->n_list = 0;
		nn->use();
		nn->use();	/* twice to cope with "undef = 1;" */
		if (n) delete n;

		if (f==1)
			switch (no_of_undcl++) {
			case 0:		undcl1 = nn; break;
			default:	undcl2 = nn; break;
			}

		return nn;
}

Pexpr expr.typ(Ptable tbl)
/*
		find the type of "this" and place it in tp;
		return the typechecked version of the expression:
		"tbl" provides the scope for the names in "this"
*/
{
if (this == 0) error('i',"0->expr.typ");
		Pname n;
		Ptype t = 0;
		Ptype t1, t2;
		TOK b = base;
		TOK r1, r2;
```

```
#define nppromote(b)    t=np_promote(b,r1,r2,t1,t2,1)
#define npcheck(b)      (void)np_promote(b,r1,r2,t1,t2,0)
        if (tbl->base != TABLE) error('i',"expr.typ(%d)",tbl->base);
//if (b == NAME) error('d',"name %d %d %s",this,string,string?string:"?");
        if (tp) {
/*error('d',"expr.typ %d (checked) tbl=%d",this,tbl);*/
                if (b == NAME) ((Pname)this)->use();
                return this;
        }
//error('d',"expr.typ %d%k e1 %d%k e2 %d%k tbl %d\n",this,base,e1,e1?e1->base:0,e2,e
        switch (b) {            /* is it a basic type */
        case DUMMY:
                error("emptyE");
                tp = any_type;
                return this;
        case ZERO:
                tp = zero_type;
                return this;
        case IVAL:
                tp = int_type;
                return this;
        case FVAL:
                tp = float_type;
                return this;
        case ICON:
                /*      is it long?
                        explicit long?
                        decimal larger than largest signed int
                        octal or hexadecimal larger than largest unsigned int
                 */
        {       int ll = strlen(string);
                switch (string[ll-1]) {
                case 'l':
                case 'L':
                lng:
                        tp = long_type;
                        goto save;
                }

                if  (string[0] == '0') {        /* assume 8 bits in byte */
                        switch (string[1]) {
                        case 'x':
                        case 'X':
                                if (SZ_INT+SZ_INT < ll-2) goto lng;
                                goto nrm;
                        default:
                                if (BI_IN_BYTE*SZ_INT < (ll-1)*3) goto lng;
                                goto nrm;
                        }
                }
                else {
                        if (ll</*sizeof(LARGEST_INT)-1*/10) {
                nrm:
                                tp = int_type;
                                goto save;
                        }
                        if (ll>10) goto lng;
                        char* p = string;
                        char* q = LARGEST_INT;
                        do if (*p++>*q++) goto lng; while (*p);
                }

                goto nrm;
        }
        case CCON:
                tp = char_type;
                goto save;
        case FCON:
                tp = float_type;
                goto save;
        case STRING:
        {       int ll = strlen(string);        /* type of "asdf" is char[5] */
                Pvec v = new vec(char_type,0);
                v->size = ll+1;
                tp = v;
                goto save;
        }
        save:
/*error('d',"%s const_save %d",string,const_save);*/
                if (const_save) {
                        int ll = strlen(string);
                        char* p = new char[ll+1];
                        strcpy(p,string);
                        string = p;
                }
                return this;

        case THIS:
                delete this;
                if (cc->tot) {
                        cc->c_this->use();
                        return cc->c_this;
                }
                error("this used in nonC context");
                n = new name("this");
                n->tp = any_type;
                return tbl->insert(n,0);

        case NAME:
/*error('d',"name %s",string);*/
        {       Pexpr ee = tbl->find_name((Pname)this,0,0);
                if (ee->tp->base == RPTR) return ee->contents();
                return ee;
        }
        case SIZEOF:
                t = tp2;
                if (t) {
                        t->dcl(tbl);
                        if (e1 && e1!=dummy) {
                                e1 = e1->typ(tbl);
                                DEL(e1);
```

```
                                        e1 = dummy;
                                }
                        }
                        else {
                                e1 = e1->typ(tbl);
                                tp2 = e1->tp;
                        }
                        tp = int_type;
                        return this;

                case CAST:
                {
                        Ptype tt = t = tp2;
                        tt->dcl(tbl);
                zaq:                                    /* is the cast legal? */
/*error('d',"tt %d %d",tt,tt?tt->base:0);*/
                        switch (tt->base) {
                        case TYPE:
                                tt = ((Pbase)tt)->b_name->tp;   goto zaq;
                        case RPTR:      // necessary?
                        case PTR:
                                if ( ((Pptr)tt)->rdo ) error("*const in cast");
                                tt = ((Pptr)tt)->typ;
                                goto zaq;
                        case VEC:
                                tt = ((Pvec)tt)->typ;
                                goto zaq;
                        case FCT:
                                tt = ((Pfct)tt)->returns;
                                goto zaq;
                        default:
                                if ( ((Pbase)tt)->b_const ) error("const in cast");
                        }

                        /* now check cast against value, INCOMPLETE */

/*error('d',"cast e1 %d %d",e1,e1->base);*/
                        tt = t;

                        if (e1 == dummy) {
                                error("expression missing for cast");
                                tp = any_type;
                                return this;
                        }
                        e1 = e1->typ(tbl);
                        Ptype etp = e1->tp;
                        while (etp->base == TYPE) etp = ((Pbase)etp)->b_name->tp;

                        if (etp->base == COBJ) {
                                int i = can_coerce(tt,etp);
/*error('d',"cast%t->%t -- %d%n",tt,etp,i,Ncoerce);*/
                                if (i==1 && Ncoerce) {
                                        Pname cn = ((Pbase)etp)->b_name;
                                        Pclass cl = (Pclass)cn->tp;
                                        Pref r = new ref(DOT,e1,Ncoerce);
                                        Pexpr c = new expr(G_CALL,r,0);
                                c->fct_name = Ncoerce;
                                c->tp = tt;
                                *this = *(Pexpr)c;
                                delete c;
                                return this;
                        }
                }

                switch (etp->base) {
                case VOID:
                        if (tt->base == VOID) {
                                tp = t;
                                return this;
                        }
                        error("cast of void value");
                case ANY:
                        tp = any_type;
                        return this;
                }

        legloop:
                switch (tt->base) {
                case TYPE:
                        tt = ((Pbase)tt)->b_name->tp; goto legloop;
                case VOID:
                        switch (etp->base) {
                        case COBJ:
                                switch (e1->base) {
                                case VALUE:
                                case CALL:
                                case G_CALL:
                                {       Pname cln = etp->is_cl_obj();
                                        Pclass cl = (Pclass)cln->tp;
                                        if (cl->has_dtor()) error('s',"cannot castC
                                }
                                }
                                break;
                        }
                        break;
                case PTR:
                        switch (etp->base) {
                        case COBJ:
                                error("cannot castCO toP");
                                break;
                        }
                        break;
                case RPTR:                      // can be simplified?
                {       Ptype t1 = etp;
                refl:
                        switch (t1->base) {
                        case TYPE:
                                t1 = ((Pbase)t1)->b_name->tp;
                                goto refl;
                //      case PTR:
                //      case RPTR:
                //      case VEC:
```

```
		//		break;
			case COBJ:
				e1 = e1->address();
				break;
			default:
				error(0,"cannot cast%t to reference",e1->tp);
			}
			break;
		}
		case COBJ:
			if (e1->lval(0)) {	/* (x)a => *(x*)&a */
				Ptype pt = new ptr(PTR,t);
				e1 = e1->address();
				e1 = new texpr(CAST,pt,e1);
				e1 = e1->contents();
				*this = *e1;
			}
			else
				error('s',"cannot cast toCO");
			break;
		case CHAR:
		case INT:
		case SHORT:
		case LONG:
		case FLOAT:
		case DOUBLE:
			switch (etp->base) {
			case COBJ:
				error("cannot castCO to%k",tt->base);
				break;
			}
			break;

		}
		tp = t;
		return this;
	}

	case VALUE:
	{	Ptype tt = tp2;
		Pclass cl;
		Pname cn;
/*error('d',"value %d %d (%d %d)",tt,tt?tt->base:0,e1,e1?e1->base:0);*/

		tt->dcl(tbl);
	vv:
/*error('d',"vv %d %d",tt,tt?tt->base:0);*/
		switch (tt->base) {
		case TYPE:
			tt = ((Pbase)tt)->b_name->tp;
			goto vv;
		case EOBJ:
		default:
			if (e1 == 0) {
				error("value missing in conversion to%t",tt);
				tp = any_type;
```

```
                                return this;
                        }
                        base = CAST;
                        return typ(tbl);
                case CLASS:
                        cl = (Pclass)tt;
                        goto nn;
                case COBJ:
                        cn = ((Pbase)tt)->b_name;
                        cl = (Pclass)cn->tp;
                nn:
                        if (e1 && e1->e2==0) {          /* single argument */
                                e1->e1 = e1->e1->typ(tbl);
                                Pname acn=e1->e1->tp->is_cl_obj();
/*error('d',"acn%n itor%n",acn,cl->itor);*/
                                if (acn && acn->tp==cl) {       /* x(x_obj) */
                                        if (cl->has_itor() == 0) return e1->e1;
                                }
                        }

                {       /* x(a) => obj.ctor(a); where e1==obj */
                        Pexpr ee;
                        Pexpr a = e1;
                        Pname ctor = cl->has_ctor();
                        if (ctor == 0) {
                                error("cannot make a%n",cn);
                                base = SM;
                                e1 = dummy;
                                e2 = 0;
                                return this;
                        }
/*error('d',"value %n.%n",e2,ctor);*/
                        if (e2 == 0) {  /*  x(a) => x temp; (temp.x(a),temp) */
                                Ptable otbl = tbl;
                                if (Cstmt) { /* make Cstmt into a block */
                                        if (Cstmt->memtbl == 0) Cstmt->memtbl = new
                                        tbl = Cstmt->memtbl;
                                }
                                char* s = make_name('V');
/*error('d',"%s: %d %d",s,otbl,tbl);*/
                                Pname n = new name(s);
                                n->tp = tp2;
                                n = n->dcl(tbl,ARG); /* no init! */
                                n->n_scope = FCT;
                                n->assign();
                                e2 = n;
                                ee = new expr(CM,this,n);
                                tbl = otbl;
                        }
                        else
                                ee = this;

                        base = G_CALL;
                        e1 = new ref(DOT,e2,ctor);
                        e2 = a;
                        return ee->typ(tbl);
```

```
				}
				}
		}

		case NEW:
		{	Ptype tt = tp2;
			Ptype tx = tt;
			bit v = 0;
			bit old = new_type;
			new_type = 1;
/*error('d',"new%t e1 %d %d",tt,e1,e1?e1->base:0);*/
			tt->dcl(tbl);
			new_type = old;
			if (e1) e1 = e1->typ(tbl);
		ll:
/*error('d',"tt %d %d",tt,tt?tt->base:0);*/
			switch (tt->base) {
			default:
				if (e1) error('i',"Ir for new non-C");
				break;
			case VEC:
				v = 1;
				tt = ((Pvec)tt)->typ;
				goto ll;
			case TYPE:
				tt = ((Pbase)tt)->b_name->tp;
				goto ll;
			case COBJ:
			{	Pname cn = ((Pbase)tt)->b_name;
				Pclass cl = (Pclass)cn->tp;
				if (cl->defined == 0) {
					error("new%n;%n isU",cn,cn);
				}
				else {
					Pname ctor = cl->has_ctor();
					TOK su;
					if (ctor) {
/*error('d',"cobj%n tp%t",ctor,ctor->tp);*/
						e1 = new call(ctor,e1);
						e1 = e1->typ(tbl);
						/*(void) e1->fct_call(tbl);*/
					}
					else if (su=cl->is_simple()) {
/*error('d',"simple cobj%k",su);*/
						if (e1) error("new%n withIr",cn);
					}
					else {
/*error('d',"not simple and no constructor?");*/
					}
				}
			}
			}
/*error('d',"v==%d",v);*/
			tp = (v) ? (Ptype)tx : (Ptype)new ptr(PTR,tx,0);
			return this;
		}
```

```
	}


	if (e1==0 && e2==0) error('i',"no operands for%k",b);

	switch(b) {
	case ILIST:	/* an ILIST is pointer to an ELIST */
		e1 = e1->typ(tbl);
		tp = any_type;
		return this;

	case ELIST:
		{	Pexpr e;
			Pexpr ex;

			if (e1 == dummy && e2==0) {
				error("emptyIrL");
				tp = any_type;
				return this;
			}

			for (e=this; e; e=ex) {
				Pexpr ee = e->e1;
/*error('d',"e %d %d ee %d %d",e,e?e->base:0,ee,ee?ee->base:0);*/
				if (e->base != ELIST) error('i',"elist%k",e->base);
				if (ex = e->e2) {	/* look ahead for end of li
					if (ee == dummy) error("EX in EL");
					if (ex->e1 == dummy && ex->e2 == 0) {
						/* { ... , } */
						DEL(ex);
						e->e2 = ex = 0;
					}
				}
				e->e1 = ee->typ(tbl);
				t = e->e1->tp;

			}
			tp = t;
			return this;
		}

	case DOT:
	case REF:
	{
		Pbase b;
		Ptable atbl;
		Pname nn;
		char* s;
		Pclass cl;

		e1 = e1->typ(tbl);
		t = e1->tp;

		if (base == REF) {
		xxx:
			switch (t->base) {
```

```
                        case TYPE:      t = ((Pbase)t)->b_name->tp;     goto xxx;
                        default:        error("nonP ->%n",mem);
                        case ANY:       atbl = any_tbl;                 goto mm;
                        case PTR:
                        case VEC:       b = (Pbase)((Pptr)t)->typ;      break;
                        }
                }
                else {
                qqq:
                        switch (t->base) {
                        case TYPE:      t = ((Pbase)t)->b_name->tp;     goto qqq;
                        default:        error("nonO .%n",mem);
                        case ANY:       atbl = any_tbl;                 goto mm;
                        case COBJ:      break;
                        }

                        switch (e1->base) {     /* FUDGE, but cannot use lval (cons
                        case CM:
                                /* ( ... , x). => ( ... , &x)-> */
                        {       Pexpr ex = e1;
                        cfr:    switch (ex->e2->base) {
                                case NAME:
                                        base = REF;
                                        ex->e2 = ex->e2->address();
                                        goto xde;
                                case CM:
                                        ex = ex->e2;
                                        goto cfr;
                                }
                        }
                        case CALL:
                        case G_CALL:
                                if (e1->fct_name==0
                                || ((Pfct)e1->fct_name->tp)->f_inline==0) {
                                        /* f(). => (tmp=f(),&tmp)-> */
                                        Ptable otbl = tbl;
                                        if (Cstmt) { /* make Cstmt into a block */
                                                if (Cstmt->memtbl == 0) Cstmt->memt
                                                tbl = Cstmt->memtbl;
                                        }
                                        char* s = make_name('T');
                                        Pname tmp = new name(s);
                                        tmp->tp = e1->tp;
                                        tmp = tmp->dcl(tbl,ARG); /* no init! */
                                        tmp->n_scope = FCT;
                                        e1 = new expr(ASSIGN,tmp,e1);
                                        e1->tp = tmp->tp;
                                        Pexpr aa = tmp->address();
                                        e1 = new expr(CM,e1,aa);
                                        e1->tp = aa->tp;
                                        base = REF;
                                        tbl = otbl;
                                }
                                break;
                        case QUEST:
                                error("non-lvalue .%n",mem);
                                break;
                        case NAME:
                                ((Pname)e1)->take_addr();
                        }
                xde:
                        b = (Pbase)t;
                }

        xxxx:
                switch (b->base) {
                case TYPE:      b = (Pbase) b->b_name->tp;      goto xxxx;
                default:        error("badT before %k%n",base,mem);
                case ANY:       atbl = any_tbl;                 goto mm;

                case COBJ:
                        if (atbl = b->b_table) goto mm;

                        s = b->b_name->string;  /* lookup the class name */
                        if (s == 0) error('i',"%kN missing",CLASS);
                        nn = tbl->look(s,CLASS);
                        if (nn == 0) error('i',"%k %sU",CLASS,s);
                        if (nn != b->b_name) b->b_name = nn;
                        cl = (Pclass) nn->tp;
                        PERM(cl);
                        if (cl == 0) error('i',"%k %s'sT missing",CLASS,s);
                        b->b_table = atbl = cl->memtbl;
                mm:
                        if (atbl->base != TABLE) error('i',"atbl(%d)",atbl->base);
                        nn = (Pname)atbl->find_name(mem,2,0);
/*error('d',"nn%n %d %d",nn,nn->n_stclass,nn->n_scope);*/
                        switch (nn->n_stclass) {
                        case 0:
                                mem = nn;
                                tp = nn->tp;
                                return this;
                        case STATIC:
                                return nn;
                        }
                }
        }

        case CALL:      /* handle undefined function names */
                if (e1->base==NAME && e1->tp==0) e1 = tbl->find_name((Pname)e1,1,e2
                break;
        case QUEST:
                cond = cond->typ(tbl);
        }


        if (e1) {
                e1 = e1->typ(tbl);
                if (e1->tp->base == RPTR) e1 = e1->contents();
                t1 = e1->tp;
        }
        else
                t1 = 0;
```

```
	if (e2) {
		e2 = e2->typ(tbl);
		if (e2->tp->base == RPTR) e2 = e2->contents();
		t2 = e2->tp;
	}
	else
		t2 = 0;


	TOK bb;
	switch (b) {				/* filter non-overloadable operators out */
	default:	bb = b; break;
	case DEREF:	bb = (e2) ? DEREF : MUL; break;
	case CM:
	case QUEST:
	case G_ADDROF:
	case G_CALL:	goto not_overloaded;
	}

	Pname n1;
	if (e1) {
		Ptype tx = t1;
		while (tx->base == TYPE) tx = ((Pbase)tx)->b_name->tp;
		n1 = tx->is_cl_obj();
	}
	else
		n1 = 0;

	Pname n2;
	if (e2) {
		Ptype tx = t2;
		while (tx->base == TYPE) tx = ((Pbase)tx)->b_name->tp;
		n2 = tx->is_cl_obj();
	}
	else
		n2 = 0;
/*error('d',"overload %k: %s %s\n", bb, n1?n1->string:"1", n2?n2->string:"2");*/
	if (n1==0 && n2==0) goto not_overloaded;
{
	/* first try for non-member function:   op(e1,e2) or op(e2) or op(e1) */
	Pexpr oe2 = e2;
	Pexpr ee2 = (e2 && e2->base!=ELIST) ? e2 = new expr(ELIST,e2,0) : 0;
	Pexpr ee1 = (e1) ? new expr(ELIST,e1,e2) : ee2;
	char* obb = oper_name(bb);
	Pname gname = gtbl->look(obb,0);
	int go = gname ? over_call(gname,ee1) : 0;
	int nc = Nover_coerce;	/* first look at member functions
				*/
	if (go) gname = Nover;
/*error('d',"global%n go=%d nc=%d",gname,go,nc);fflush(stderr);*/

	if (n1) {				/* look for member of n1 */
		Ptable ctbl = ((Pclass)n1->tp)->memtbl;
		Pname mname = ctbl->look(obb,0);
		if (mname == 0) goto glob;
```

```
                switch (mname->n_scope) {
                default:        goto glob;
                case 0:
                case PUBLIC:    break;          /* try e1.op(?) */
                }

                int mo = over_call(mname,e2);
/*error('d',"n1%n %d",mname,mo);*/
                switch (mo) {
                case 0:
                        if (1 < Nover_coerce) goto am1;
                        goto glob;
                case 1: if (go == 2) goto glob;
                        if (go == 1) {
                        am1:
                                error("ambiguous operandTs%n%t for%k",n1,t2,b);
                                tp = any_type;
                                return this;
                        }
                        else {
                                Pclass cl = (Pclass)n1->tp;
                                if (cl->conv) error('w',"overloaded%k may be ambigu
                        }
                }

                if (bb==ASSIGN && mname->n_table!=ctbl) {       /* inherited = */
                        error("assignment not defined for class%n",n1);
                        tp = any_type;
                        return this;
                }

                base = G_CALL;                  /* e1.op(e2) or e1.op() */
                e1 = new ref(DOT,e1,Nover);
                if (ee1) delete ee1;
                return typ(tbl);
        }

        if (n2 && e1==0) {                      /* look for unary operator */
                Ptable ctbl = ((Pclass)n2->tp)->memtbl;
                Pname mname = ctbl->look(obb,0);
                if (mname == 0) goto glob;
                switch (mname->n_scope) {
                default:        goto glob;
                case 0:
                case PUBLIC:    break;          /* try e2.op() */
                }

                int mo = over_call(mname,0);
/*error('d',"n2%n %d",mname,mo);*/
                switch (mo) {
                case 0:
                        if (1 < Nover_coerce) goto am2;
                        goto glob;
                case 1: if (go == 2) goto glob;
                        if (go == 1) {
                        am2:
                                error("ambiguous operandT%n for%k",n2,b);
                                tp = any_type;
                                return this;
                        }
                }

                base = G_CALL;                  /* e2.op() */
                e1 = new ref(DOT,oe2,Nover);
                e2 = 0;
                if (ee2) delete ee2;
                if (ee1 && ee1!=ee2) delete ee1;
                return typ(tbl);

        }

glob:   if (1 < nc) {
                error("ambiguous operandTs%t%t for%k",t1,t2,b);
                tp = any_type;
                return this;
        }
        if (go) {
                if (go == 1) {  /* conversion necessary => binary */
                        if (n1) {
                                Pclass cl = (Pclass)n1->tp;
                                if (cl->conv) error('w',"overloaded%k may be ambigu
                        }
                        else if (n2) {
                                Pclass cl = (Pclass)n2->tp;
                                if (cl->conv) error('w',"overloaded%k may be ambigu
                        }
                }
                base = G_CALL;                  /* op(e1,e2) or op(e1) or op(e2) */
                e1 = gname;
                e2 = ee1;
                return typ(tbl);
        }

        if (ee2) delete ee2;
        if (ee1 && ee1!=ee2) delete ee1;
        e2 = oe2;
/*error('d',"bb%k",bb);*/
        switch(bb) {
        case ASSIGN:
        case ADDROF:
        case CALL:
        case DEREF:
                break;
        default:        /* look for conversions to basic types */
        {       int found = 0;
                if (n1) {
                        int val = 0;
                        Pclass cl = (Pclass)n1->tp;
                        for ( Pname on = cl->conv; on; on=on->n_list) {
/*error('d',"oper_coerce n1%n %t",on,(on)?Pfct(on->tp)->returns:0);*/
                                Pfct f = (Pfct)on->tp;
                                if (bb==ANDAND || bb==OROR) {
```

```
                                                e1 = check_cond(e1,bb,0);
                                                goto not_overloaded;
                                        }
                                        if (n2
                                        || f->returns->check(t2,ASSIGN)==0
                                        || t2->check(f->returns,ASSIGN)==0) {
                                                Ncoerce = on;
                                                val++;
                                        }
                                }
                                switch (val) {
                                case 0:
                                        break;
                                case 1:
                                {       Pref r = new ref(DOT,e1,Ncoerce);
                                        e1 = new expr(G_CALL,r,0);
                                        found = 1;
                                        break;
                                }
                                default:
                                        error('s',"ambiguous coercion of%n to basicT",n1);
                                }
                        }
                        if (n2) {
                                int val = 0;
                                Pclass cl = (Pclass)n2->tp;
                                for ( Pname on = cl->conv; on; on=on->n_list) {
/*error('d',"oper_coerce n2%n %t",on,(on)?on->tp:0);*/
                                        Pfct f = (Pfct)on->tp;
                                        if (bb==ANDAND || bb==OROR) {
                                                e2 = check_cond(e2,bb,0);
                                                goto not_overloaded;
                                        }
                                        if (n1
                                        || f->returns->check(t1,ASSIGN)==0
                                        || t1->check(f->returns,ASSIGN)==0) {
                                                Ncoerce = on;
                                                val++;
                                        }
                                        //if (n1 || t1->check(f->returns,COERCE)==0) {
                                        //      Ncoerce = on;
                                        //      val++;
                                //      }
                                }
                                switch (val) {
                                case 0:
                                        break;
                                case 1:
                                {       Pref r = new ref(DOT,e2,Ncoerce);
                                        e2 = new expr(G_CALL,r,0);
                                        found++;
                                        break;
                                }
                                default:
                                        error('s',"ambiguous coercion of%n to basicT",n2);
                                }
```

```
                }
                if (found) {
                /*      if (found == 2) error('w',"coercions of operands of%k may b
                        return typ(tbl);
                }
                if (t1 && t2)
                        error("bad operandTs%t%t for%k",t1,t2,b);
                else
                        error("bad operandT%t for%k",t1?t1:t2,b);
                tp = any_type;
                return this;
        }
        }
}
not_overloaded:
        t = (t1==0) ? t2 : (t2==0) ? t1 : 0;
/*fprintf(stderr,"%s: e1 %d %d e2 %d %d\n",oper_name(b),e1,e1?e1->base:0,e2,e2?e2->b
        switch (b) {            /* are the operands of legal types */
        case G_CALL:
        case CALL:
                tp = fct_call(tbl);     /* two calls of use() for e1's names */
                if (tp->base == RPTR) return contents();
                return this;

        case DEREF:
                if (e1 == dummy) error("O missing before []\n");
                if (t) {        /*      *t      */
                        t->vec_type();
                        tp = t->deref();
                }
                else {          /*      t1[t2]  */
                        t1->vec_type();
                        t2->integral(b);
                        tp = t1->deref();
                }
                if (tp->base == RPTR) return contents();
                return this;

        case G_ADDROF:
        case ADDROF:
                if (e2->lval(b) == 0) {
                        tp = any_type;
                        return this;
                }
                tp = t->addrof();
                        /* look for &p->member_function */
                switch (e2->base) {
                case DOT:
                case REF:
                {       Pname m = e2->mem;
                        Pfct f = (Pfct)m->tp;
                        if (f->base==FCT && (f->f_virtual==0 || m->n_qualifier)) {
                                DEL(e2);
                                e2 = m;
                        }
                }
                }
                return this;

        case UMINUS:
                t->numeric(b);
                tp = t;
                return this;

        case NOT:
                e2 = check_cond(e2,NOT,tbl);
                tp = int_type;
                return this;
        case COMPL:
                t->integral(b);
                tp = t;
                return this;

        case INCR:
        case DECR:
                if (e1) e1->lval(b);
                if (e2) e2->lval(b);
                r1 = t->num_ptr(b);
                tp = t;
                return this;
        }

        if (e1==dummy || e2==dummy || e1==0 || e2==0) error("operand missing for%k"
        switch (b) {
        case MUL:
        case DIV:
                r1 = t1->numeric(b);
                r2 = t2->numeric(b);
                nppromote(b);
                break;
        case MOD:
                r1 = t1->integral(b);
                r2 = t2->integral(b);
                nppromote(b);
                break;
        case PLUS:
                r1 = t1->num_ptr(b);
                r2 = t2->num_ptr(b);
                if (r1==P && r2==P) error("P +P");
                nppromote(b);
                break;
        case MINUS:
                r1 = t1->num_ptr(b);
                r2 = t2->num_ptr(b);
                if (r2==P && r1!=P && r1!=A) error("P - nonP");
                nppromote(b);
                break;
        case LS:
        case RS:
        case AND:
        case OR:
        case ER:
                r1 = t1->integral(b);
                r2 = t2->integral(b);
                nppromote(b);
                break;
        case LT:
        case LE:
        case GT:
        case GE:
        case EQ:
        case NE:
                r1 = t1->num_ptr(b);
                r2 = t2->num_ptr(b);
                npcheck(b);
                t = int_type;
                break;
        case ANDAND:
        case OROR:
        /*      t1->num_ptr(b);
                t2->num_ptr(b);
        */
                e1 = check_cond(e1,b,tbl);
                e2 = check_cond(e2,b,tbl);
                t = int_type;
                break;
        case QUEST:
                cond = check_cond(cond,b,tbl);
                if (t1 == t2) {         /* not general enough */
                        t = t1;
                }
                else {
                        r1 = t1->num_ptr(b);
                        r2 = t2->num_ptr(b);
                        nppromote(b);
                        if (t != t1) e1 = new texpr(CAST,t,e1);
                        if (t != t2) e2 = new texpr(CAST,t,e2);
                }
                break;
        case ASPLUS:
                r1 = t1->num_ptr(b);
                r2 = t2->num_ptr(b);
                if (r1==P && r2==P) error("P +=P");
                nppromote(b);
                goto ass;
        case ASMINUS:
                r1 = t1->num_ptr(b);
                r2 = t2->num_ptr(b);
                if (r2==P && r1!=P && r1!=A) error("P -= nonP");
                nppromote(b);
                goto ass;
        case ASMUL:
        case ASDIV:
                r1 = t1->numeric(b);
                r2 = t1->numeric(b);
                nppromote(b);
                goto ass;
        case ASMOD:
                r1 = t1->integral(b);
                r2 = t2->integral(b);
                nppromote(b);
                goto ass;
        case ASAND:
        case ASOR:
        case ASER:
        case ASLS:
        case ASRS:
                r1 = t1->integral(b);
                r2 = t2->integral(b);
                npcheck(b);
                t = int_type;
                goto ass;
        ass:
                as_type = t;    /* the type of the rhs */
                t2 = t;
        case ASSIGN:
                if (e1->lval(b) == 0) {
                        tp = any_type;
                        return this;
                }
        lkj:
                switch (t1->base) {
                case INT:
                case CHAR:
                case SHORT:
                        if (e2->base==ICON && e2->tp==long_type)
                                error('w',"long constant assigned to%k",t1->base);
                case LONG:
                        if (b==ASSIGN
                        && ((Pbase)t1)->b_unsigned
                        && e2->base==UMINUS
                        && e2->e2->base==ICON)
                                error('w',"negative assigned to unsigned");
                        break;
                case TYPE:
                        t1 = ((Pbase)t1)->b_name->tp;
                        goto lkj;
                case COBJ:
                {       Pname c1 = t1->is_cl_obj();

                        if (c1) {
                                Pname c2 = t2->is_cl_obj();
/*error('d',"%t=%t %d %d",t1,t2,c1,c2);*/
                                if (c1 != c2) {
                                        e2 = new expr(ELIST,e2,0);
                                        e2 = new texpr(VALUE,t1,e2);
                                        e2->e2 = e1;
                                        e2 = e2->typ(tbl);
                                        *this = *e2;
                                        tp = t1;
                                        return this;
                                }
                        }
                        break;
                }
                case PTR:
/*error('d',"ptr %d %d",t1,t1?t1->base:0);*/
                {       Pfct ef = (Pfct)((Pptr)t1)->typ;
                        if (ef->base == FCT) {
                                Pfct f;
                                Pname n = 0;
                                switch (e2->base) {
                                case NAME:
                                        f = (Pfct)e2->tp;
                                        n = Pname(e2);
                                        switch (f->base) {
                                        case FCT:
                                        case OVERLOAD:
                                                e2 = new expr(G_ADDROF,0,e2);
                                                e2->tp = f;
                                        }
                                        goto ad;
                                case DOT:
                                case REF:
/*error('d',"dot %d %d",e2->mem->tp,e2->mem->tp?e2->mem->tp->base:0);*/
                                        f = (Pfct)e2->mem->tp;
                                        switch (f->base) {
                                        case FCT:
                                        case OVERLOAD:
                                                n = Pname(e2->mem);
                                                e2 = new expr(G_ADDROF,0,e2);
                                                e2 = e2->typ(tbl);
                                        }
                                        goto ad;
                                case ADDROF:
                                case G_ADDROF:
                                        f = (Pfct)e2->e2->tp;
                                ad:
                                        if (f->base == OVERLOAD) {
                                                Pgen g = (Pgen)f;
                                                n = g->find(ef);
                                                if (n == 0) {
                                                        error("cannot deduceT for &
                                                        tp = any_type;
                                                }
                                                else
                                                        tp = t1;
                                                e2->e2 = n;
                                                n->lval(ADDROF);
                                                return this;
                                        }
                                        if (n) n->lval(ADDROF);
                                }
                        }
                        break;
                }
                }
                {       Pname cn;
                        int i;
                        if ((cn=t2->is_cl_obj())
                        && (i=can_coerce(t1,t2))
                        && Ncoerce) {
                                if (1 < i) error("%d possible conversions for assig
/*error('d',"%t =%t",t1,t2);*/
                                Pclass cl = (Pclass)cn->tp;
                                Pref r = new ref(DOT,e2,Ncoerce);
                                Pexpr c = new expr(G_CALL,r,0);
                                c->fct_name = Ncoerce;
                                c->tp = t1;
                                e2 = c;
                                tp = t1;
                                return this;
                        }
                }
/*error('d',"check(%t,%t)",e1->tp,t2);*/
                if (e1->tp->check(t2,ASSIGN)) error("bad assignmentT:%t =%t",e1->tp
                t = e1->tp;                             /* the type of the lhs */
                break;
        case CM:
                t = t2;
                break;
        default:
                error('i',"unknown operator%k",b);
        }

        tp = t;
        return this;
}
```

```
/* %Z% %M% %I% %H% %T% */
/*****************************************************************************

        C++ source for cfront, the C++ compiler front-end
        written in the computer science research center of Bell Labs

        Copyright (c) 1984 AT&T Technologies, Inc. All rigths Reserved
        THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

        If you ignore this notice the ghost of Ma Bell will haunt you forever.

expr2.c:

        type check expressions

****************************************************************************/

#include "cfront.h"
#include "size.h"

void name.assign()
{
        if (n_assigned_to++ == 0) {
                switch (n_scope) {
                case FCT:
                        if (n_used && n_addr_taken==0)  {
                                Ptype t = tp;
                        ll:
                                switch (t->base) {
                                case TYPE:
                                        t=((Pbase)t)->b_name->tp; goto ll;
                                case VEC:
                                        break;
                                default:
                                        if (curr_loop)
                                                error('w',"%n may have been used be
                                        else
                                                error('w',"%n used before set",this
                                }
                        }
                }
        }
}

int expr.lval(TOK oper)
{
        register Pexpr ee = this;
        register Pname n;
        int deref = 0;
        char* es;

        if (this==0 || tp==0) error('i',"%d->lval(0)",this);

        switch (oper) {
        case ADDROF:
        case G_ADDROF:
                es = "address of";
                break;
        case INCR:
        case DECR:
                es = "increment of";
                goto def;
        case DEREF:
                es = "dereference of";
                break;
        default:
                es = "assignment to";
        def:
                if (tp->tconst()) {
                        if (oper) error("%s constant",es);
                        return 0;
                }
        }

        forever {
                switch (ee->base) {
                default:
                        if (deref==0) {
                                if (oper) error("%s %k",es,ee->base);
                                return 0;
                        }
                        return 1;
                case ZERO:
                case CCON:
                case ICON:
                case FCON:
                        if (oper) error("%s numeric constant",es);
                        return 0;
                case STRING:
                        if (oper) error('w',"%s string constant",es);
                        return 1;

                case DEREF:
                {       Pexpr ee1 = ee->e1;
                        if (ee1->base == ADDROF) /* *& vanishes */
                                ee = ee1->e2;
                        else {
                                ee = ee1;
                                deref = 1;
                        }
                        break;
                }
                case INCR:
                case DECR:
                        ee = (ee->e1) ? ee->e1 : ee->e2;
                        break;

                case DOT:
                        n = ee->mem;
                        if (deref==0 && ee->e1->tp->tconst()) {
                                if (oper) error("%sM%n of%t",es,n,ee->e1->tp);
                                return 0;
                }
                goto xx;
        case REF:
                n = ee->mem;
                if (deref==0) {
                        Ptype p = ee->e1->tp;
                zxc:
                        switch (p->base) {
                        case TYPE:      p = ((Pbase)p)->b_name->tp; goto zx
                        case PTR:       break;
                        default:        error('i',"%t->%n",p,n);
                        }
                        if ( ((Pptr)p)->typ->tconst() ) {
                                if (oper) error("%sM%n of%t",es,n,((Pptr)p)
                                return 0;
                        }
                }
                goto xx;
        case NAME:
                n = (Pname)ee;
        xx:
                if (deref || oper==0) return 1;
                switch (oper) {
                case ADDROF:
                case G_ADDROF:
                {       Pfct f = (Pfct)n->tp;
                        if (n->n_sto == REGISTER) {
                                if (oper) error("& register%n",n);
                                return 0;
                        }
                        if (f == 0) {
                                if (oper) error("& label%n",n);
                                return 0;
                        }
                /*      if (f->base == OVERLOAD) {
                                if (oper) error("& overloaded%n",n);
                                return 0;
                        }
                */
                        if (n->n_stclass == ENUM) {
                                if (oper) error("& enumerator%n",n);
                                return 0;
                        }
                        n->n_used--;
                        n->take_addr();
                        if (n->n_evaluated
                        || (f->base==FCT && f->f_inline) ) {
                                /* address of const or inline: allocate it
                                Pname nn = new name;
                                if (n->n_evaluated) {
                                        n->n_evaluated = 0;     /* use allo
                                        n->n_initializer = new expr(IVAL,(P
                                }
                                *nn = *n;
                                nn->n_sto = STATIC;
                                nn->n_list = dcl_list;
```

```
                                        dcl_list = nn;
                                }
                                break;
                        }
                        case ASSIGN:
                                n->n_used--;
                                n->assign();
                                break;
                        default:        /* incr ops, and asops */
                                if (cc->tot && n==cc->c_this) {
                                        error("%n%k",n,oper);
                                        return 0;
                                }
                                n->assign();
                        }
                        return 1;
                }
        }
}

Pexpr Ninit;

bit gen_match(Pname n, Pexpr arg)
/*
        look for an exact match between "n" and the argument list "arg"
*/
{
        Pfct f = (Pfct) n->tp;
        register Pexpr e;
        register Pname nn;

        for (e=arg, nn=f->argtype; e; e=e->e2, nn=nn->n_list) {
                Pexpr a = e->e1;
                Ptype at = a->tp;
                if (at->base == ANY) return 0;
                if (nn == 0) return f->nargs_known==ELLIPSIS;

                Ptype nt = nn->tp;

                switch (nt->base) {
                case RPTR:
                        if (nt->check(at,COERCE)) {
                                if (((Pptr)nt)->typ->check(at,0)) return 0;
                        }
                        break;
                default:
                        if (nt->check(at,COERCE)) return 0;
                }
        }

        if (nn) {
                Ninit = nn->n_initializer;
                return Ninit!=0;
        }
        return 1;
}
```

```
Pname Ncoerce;

bit can_coerce(Ptype t1, Ptype t2)
/*      return number of possible coercions of t2 into t1,
        Ncoerce holds a coercion function (not constructor), if found
*/
{
/*error('d',"can_coerce %t->%t",t1,t2);*/
        Ncoerce = 0;
        if (t2->base == ANY) return 0;
        switch (t1->base) {
        case RPTR:
        rloop:
                switch (t2->base) {
                case TYPE:
                        t2 = ((Pbase)t2)->b_name->tp;
                        goto rloop;
        //      case VEC:
        //      case PTR:
        //      case RPTR:
        //              if (t1->check(t2,COERCE) == 0) return 1;
                default:
                {       Ptype tt2 = t2->addrof();
                        if (t1->check(tt2,COERCE) == 0) return 1;
                        Ptype tt1 = ((Pptr)t1)->typ;
                        int i = can_coerce(tt1,t2);
                        return i;
                }
                }
        }

        Pname c1 = t1->is_cl_obj();
        Pname c2 = t2->is_cl_obj();
        int val = 0;

        if (c1) {
                Pclass cl = (Pclass)c1->tp;
                if (c2 && c2->tp==cl) return 1;

                /*      look for constructor
                                with one argument
                                or with default for second argument
                        of acceptable type
                */
                Pname ctor = cl->has_ctor();
                if (ctor == 0) goto oper_coerce;
                register Pfct f = (Pfct)ctor->tp;

                switch (f->base) {
                case FCT:
                        switch (f->nargs) {
                        case 1:
                        one:
                                if (f->argtype->tp->check(t2,COERCE)==0) val = 1;
                                goto oper_coerce;
                        default:
                                if (f->argtype->n_list->n_initializer) goto one;
                        case 0:
                                goto oper_coerce;
                        }
                case OVERLOAD:
                {       register Plist gl;

                        for (gl=Pgen(f)->fct_list; gl; gl=gl->l) {      /* look for
                                Pname nn = gl->f;
                                Pfct ff = (Pfct)nn->tp;
                                switch (ff->nargs) {
                                case 0:
                                        break;
                                case 1:
                                over_one:
                                        if (ff->argtype->tp->check(t2,COERCE)==0) v
                                        if (ff->argtype->tp->base == RPTR
                                        && ((Pptr)ff->argtype->tp)->typ->check(t2,C
                                                val = 1;
                                                goto oper_coerce;
                                        }
                                        break;
                                default:
                                        if (ff->argtype->n_list->n_initializer) got
                                }
                        }
                        goto oper_coerce;
                }
                default:
                        error('i',"cannot_coerce(%k)\n",f->base);
                }
        }
oper_coerce:
        if (c2) {
                Pclass cl = (Pclass)c2->tp;
                for (register Pname on=cl->conv; on; on=on->n_list) {
/*error('d'," oper_coerce%n %t %d",on,(on)?on->tp:0,on);*/
                        Pfct f = (Pfct)on->tp;
                        if (t1->check(f->returns,COERCE) == 0) {
                                Ncoerce = on;
                                val++;
                        }
                }
        }

        if (val) return val;
        if (t1->check(t2,COERCE)) return 0;
        return 1;
}

int gen_coerce(Pname n, Pexpr arg)
/*
        look to see if the argument list "arg" can be coerced into a call of "n"
        1: it can
        0: it cannot or it can be done in more than one way
*/
```

```
{
        Pfct f = (Pfct) n->tp;
        register Pexpr e;
        register Pname nn;
/*error('d',"gen_coerce%n %d",n,arg);*/
        for (e=arg, nn=f->argtype; e; e=e->e2, nn=nn->n_list) {
                if (nn == 0) return f->nargs_known==ELLIPSIS;
                Pexpr a = e->e1;
                Ptype at = a->tp;
                int i = can_coerce(nn->tp,at);
/*error('d',"a1 %k at%t argt%t -> %d",a->base,at,nn->tp,i);*/
                if (i != 1) return 0;
        }
        if (nn && nn->n_initializer==0) return 0;
        return 1;
}


Pname Nover;
int Nover_coerce;

int over_call(Pname n, Pexpr arg)
/*
        return 2 if n(arg) can be performed without user defined coercion of arg
        return 1 if n(arg) can be performed only with user defined coercion of arg
        return 0 if n(arg) is an error
*/
{
        register Plist gl;
        Pgen g = (Pgen) n->tp;
        if (arg && arg->base!= ELIST) error('i',"ALX");
/*error('d',"over_call%n base%k arg%d%k", n, g->base, arg, arg?arg->tp->base:0);*/
        Nover_coerce = 0;
        switch (g->base) {
        default:         error('i',"over_call(%t)\n",g);
        case OVERLOAD:  break;
        case FCT:
                Nover = n;
                Ninit = 0;
                if (gen_match(n,arg) && Ninit==0) return 2;
                if (gen_coerce(n,arg)) return 1;
                return 0;
        }

        for (gl=g->fct_list; gl; gl=gl->l) {            /* look for match */
                Nover = gl->f;
                Ninit = 0;
/*error('d',"over_call: gen_match(%n,%k) %d",Nover,arg->e1->base,gen_match(Nover,arg
                if (gen_match(Nover,arg) && Ninit==0) return 2;
        }

        Nover = 0;
        for (gl=g->fct_list; gl; gl=gl->l) {            /* look for coercion */
                Pname nn = gl->f;
/*error('d',"over_call: gen_coerce(%n,%k) %d",nn,arg->e1->base,gen_coerce(nn,arg));*
                if (gen_coerce(nn,arg)) {
```

```
                                if (Nover) {
                                        Nover_coerce = 2;
                                        return 0;               /* ambiguous */
                                }
                                Nover = nn;
                        }
                }

                return Nover ? 1 : 0;
}


Ptype expr.fct_call(Ptable tbl)
/*
        check "this" call:
                 e1(e2)
        e1->typ() and e2->typ() has been done
*/
{
        Pfct f;
        Pname fn;
        int x;
        int k;
        Pname nn;
        Pexpr e;
        Ptype t;
        Pexpr arg = e2;
        Ptype t1;
        int argno;
        Pexpr etail = 0;
        Pname no_virt;  /* set if explicit qualifier was used: c::f() */
/*error('d',"fct_call");*/
        switch (base) {
        case CALL:
        case G_CALL:
                break;
        default:
                error('i',"fct_call(%k)",base);
        }

        if (e1==0 || (t1=e1->tp)==0) error('i',"fct_call(e1=%d,e1->tp=%t)",e1,t1);
        if (arg && arg->base!=ELIST) error('i',"badAL%d%k",arg,arg->base);

        switch (e1->base) {
        case NAME:
                fn = (Pname)e1;
                no_virt = fn->n_qualifier;
                break;
        case REF:
        case DOT:
                fn = e1->mem;
                no_virt = fn->n_qualifier;
                break;
        default:
                fn = 0;
                no_virt = 0;
```

```
	};
/*error('d',"fn%n t1%k",fn,t1->base);*/
	switch (t1->base) {
	default:
		error("call of%n;%n is a%t)",fn,fn,e1->tp);

	case ANY:
		return any_type;

	case OVERLOAD:
	{	register Plist gl;
		Pgen g = (Pgen) t1;
		Pname found = 0;

		for (gl=g->fct_list; gl; gl=gl->l) {	/* look for match */
			register Pname nn = gl->f;
/*fprintf(stderr,"gen_match %s %d\n",nn->string?nn->string:"?",arg->base);*/
			if (gen_match(nn,arg)) {
				found = nn;
				goto fnd;
			}
		}

		for (gl=g->fct_list; gl; gl=gl->l) {	/* look for coercion */
			register Pname nn = gl->f;
/*fprintf(stderr,"gen_coerce %s %d\n",nn->string?nn->string:"?",arg->base);*/
			if (gen_coerce(nn,arg)) {
				if (found) {
					error("ambiguousA for overloaded%n",fn);
					goto fnd;
				}
				found = nn;
			}
		}

	fnd:
//error('d',"found%n",found);
		if (found) {
			Pbase b;
			Ptable tblx;

			f = (Pfct)found->tp;
			fct_name = found;

			/* is fct_name visible? */
//error('d',"e1 %d%k",e1,e1?e1->base:0);
			switch (e1->base) {
			case REF:
				if (e1->e1 == 0) break; /* constructor: this==0 */
				b = (Pbase) ((Pptr)e1->e1->tp)->typ; goto xxxx;
			case DOT:
				b = (Pbase)e1->e1->tp;
			xxxx:
				switch (b->base) {
				case TYPE:	b = (Pbase) b->b_name->tp; goto xxx
				case ANY:	break;
```

```
                                        case COBJ:      tblx = b->b_table;
                                        }

                                        if ( tblx->lookc(g->string,0) == 0)
                                                error('i',"fct_call overload check");
//error('d',"scope %d epriv %d ebase %d cc %d",found->n_scope,Epriv,Ebase,cc);
                                        switch (found->n_scope) {
                                        case 0:
                                                if (Epriv
                                                && Epriv!=cc->cot
                                                && !Epriv->has_friend(cc->nof)) {
                                                        error("%n is private",found);
                                                        break;
                                                }
                                                /* no break */
                                        case PUBLIC:
                                                if (Ebase
                                                && (cc->cot==0
                                                        || ( Ebase!=cc->cot->clbase->tp
                                                        && !Ebase->has_friend(cc->nof))
                                                ) {
                                                  error("%n is from a privateBC",found);
                                                }
                                        }
                                }
                        }
                        else {
                                error("badAL for overloaded%n",fn);
                                return any_type;
                        }
                        break;
        }
        case FCT:
                f = (Pfct)t1;
                if (fn) fct_name = fn;
                break;
        }

        if (no_virt) fct_name = 0;

        t = f->returns;
        x = f->nargs;
        k = f->nargs_known;
/*error('d',"fct_name%n",fct_name);*/

        if (k == 0) return t;
/*
        if (arg == 0) {
                switch (x) {
                default:        error("AX for%n",fn);
                case 0:         return t;
                }
        }
*/

        for (e=arg, nn=f->argtype, argno=1; e||nn; nn=nn->n_list, e=etail->e2, argn
                Pexpr a;

                if (e) {
                        a = e->e1;
/*error('d',"e %d%k a %d%k e2 %d",e,e->base,a,a->base,e->e2);*/
                        etail = e;

                        if (nn) {       /* type check */
                                Ptype t1 = nn->tp;
                        lx:
/*error('d',"lx: t1%t a->tp%t",t1,a->tp);*/
                                switch (t1->base) {
                                case TYPE:
                                        t1 = ((Pbase)t1)->b_name->tp;
                                        goto lx;
                                case RPTR:
                                        e->e1 = ref_init(Pptr(t1),a,tbl);
                                        break;
                                case COBJ:
                                        e->e1 = class_init(0,t1,a,tbl);
                                        break;
                                case ANY:
                                        return t;
                case PTR:
                {       Pfct ef = (Pfct)((Pptr)t1)->typ;
                        if (ef->base == FCT) {
                                Pfct f;
                                Pname n = 0;
                                switch (a->base) {
                                case NAME:
                                        f = (Pfct)a->tp;
                                        switch (f->base) {
                                        case FCT:
                                        case OVERLOAD:
                                                e->e1 = new expr(G_ADDROF,0,a);
                                                e->e1->tp = f;
                                        }
                                        n = Pname(a);
                                        goto ad;
                                case DOT:
                                case REF:
                                        f = (Pfct)a->mem->tp;
                                        switch (f->base) {
                                        case FCT:
                                        case OVERLOAD:
                                                n = Pname(a->mem);
                                                a = new expr(G_ADDROF,0,a);
                                                e->e1 = a->typ(tbl);
                                        }
                                        goto ad;
                                case ADDROF:
                                case G_ADDROF:
                                        f = (Pfct)a->e2->tp;
                                ad:
                                        if (f->base == OVERLOAD) {
                                                Pgen g = (Pgen)f;
```

```
                                                        n = g->find(ef);
                                                        if (n == 0) {
                                                                error("cannot deduceT for &
                                                                return any_type;
                                                        }
                                                        e->e1->e2 = n;
                                                }
                                                if (n) n->lval(ADDROF);
                                        }
                                        break;

                                }
                                goto def;
                        }
                                        case CHAR:
                                        case SHORT:
                                        case INT:
                                                if (a->base==ICON && a->tp==long_type)
                                                        error('w',"long constantA for%n,%kX
                                        case LONG:
                                                if (((Pbase)t1)->b_unsigned
                                                && a->base==UMINUS
                                                && a->e2->base==ICON)
                                                        error('w',"negativeA for%n, unsigne
                                        default:
                                        def:
                                                {       Pname cn;
                                                        int i;
                                                        if ((cn=a->tp->is_cl_obj())
                                                        && (i=can_coerce(t1,a->tp))
                                                        && Ncoerce) {
                                                                if (1 < i) error("%d possib
/*error('d',"%t<-%t",t1,a->tp);*/
                                                                Pclass cl = (Pclass)cn->tp;
                                                                Pref r = new ref(DOT,a,Ncoe
                                                                Pexpr c = new expr(G_CALL,r
                                                                c->fct_name = Ncoerce;
                                                                c->tp = t1;
                                                                e->e1 = c;
                                                                return t1;
                                                        }
                                                }
                                                if (t1->check(a->tp,ARG)) {
                                                        if (arg_err_suppress==0) error("bad
                                                        return any_type;
                                                }
                                        }
                                }
                                else {
                                        if (k != ELLIPSIS) {
                                                if (arg_err_suppress==0) error("unX %dA for
                                                return any_type;
                                        }
                                        return t;
                                }
                        }
```

```
                else {  /* default argument? */
                        a = nn->n_initializer;

                        if (a == 0) {
                                if (arg_err_suppress==0) error("A %d ofT%tX for%n",
                                return any_type;
                        }
                        e = new expr(ELIST,a,0);
                        if (etail)
                                etail->e2 = e;
                        else
                                e2 = e;
                        etail = e;
                }
        }

        return t;
}

int refd;

Pexpr ref_init(Pptr p, Pexpr init, Ptable tbl)
/*
        initialize the "p" with the "init"
*/
{
        register Ptype it = init->tp;
        Ptype p1;
        Pname c1;
        Pexpr a;

rloop:
/*error('d',"rloop: %d%k",it,it->base);*/
        switch (it->base) {
        case TYPE:
                it = ((Pbase)it)->b_name->tp; goto rloop;
//      case VEC:
//      case PTR:
//              if (p->check(it,ASSIGN) == 0) return init;
//              break;
        default:
                {       Ptype tt = it->addrof();
                        if (p->check(tt,ASSIGN) == 0) {
                                if (init->lval(0)) return init->address();
                                if (init->base==G_CALL  /* &inline function call? *
                                && init->fct_name
                                && ((Pfct)init->fct_name->tp)->f_inline )
                                        return init->address();
                                p1 = p->typ;
                                goto xxx;
                        }
                }
        }

        p1 = p->typ;
        c1 = p1->is_cl_obj();
```

```
	if (c1) {
		refd = 1;	/* disable itor */
		a = class_init(0,p1,init,tbl);
		refd = 0;
		if (a == init) goto xxx;
		switch (a->base) {
		case G_CALL:
		case CM:
			init = a;
			goto xxx;
		}
		return a->address();
	}

	if (p1->check(it,ASSIGN)) {
		error("badIrT:%t (%tX)",it,p);
		init->tp = any_type;
		return init;
	}

xxx:
/*error('d',"xxx: %k",init->base);*/
	switch (init->base) {
	case NAME:
	case DEREF:
	case REF:
	case DOT:				/* init => &init */
		init->lval(ADDROF);
		return init->address();
	case CM:
/*error('d',"cm%k",init->e2->base);*/
		switch (init->e2->base) {	/* (a, b) => (a, &b) */
		case NAME:
		case DEREF:
			return init->address();
		}
	default:				/* init = > ( temp=init, &temp) */
	{	Ptable otbl = tbl;
		if (Cstmt) {	/*	make Cstmt into a block */
			if (Cstmt->memtbl == 0) Cstmt->memtbl = new table(4,tbl,0);
			tbl = Cstmt->memtbl;
		}
		char* s = make_name('I');
		Pname n = new class name(s);

/*error('d',"ref_init tmp %s n=%d tbl %d init=%d%k",s,n,tbl,init,init->base);*/
		if (tbl == gtbl) error('s',"Ir for global reference not an lvaue");
		n->tp = p1;
		n = n->dcl(tbl,ARG); /* no initialization! */
		n->n_scope = FCT;
		n->assign();
		a = n->address();
		Pexpr as = new class expr(ASSIGN,n,init);
		a = new class expr(CM,as,a);
		a->tp = a->e2->tp;
```

```
                tbl = otbl;
                return a;
        }
        }
}

Pexpr class_init(Pexpr nn, Ptype tt, Pexpr init, Ptable tbl)
/*
        initialize "nn" of type "tt" with "init"
        if nn==0 make a temporary,
        nn may not be a name
*/
{       Pname c1 = tt->is_cl_obj();
        Pname c2 = init->tp->is_cl_obj();

/*error('d',"class_init%n%n%n refd=%d",nn,c1,c2,refd);*/
        if (c1) {
                if (c1!=c2
                || (refd==0 && Pclass(c1->tp)->has_itor())) {
                        /*      really ouht to make a temp if refd,
                                but ref_init can do that
                        */
                        int i = can_coerce(tt,init->tp);
                        if (Ncoerce) {
                                if (1 < i) {
                                        error("%d possible ways of making a%n from
                                        return init;
                                }
/*error('d',"coerce%n=(%d%k).%n",nn,init,init->base,Ncoerce);*/
                                switch (init->base) {
                                case CALL:
                                case G_CALL:
                                case CM:
                                case NAME:      /* init.coerce() */
                                {       Pref r = new ref(DOT,init,Ncoerce);
                                        Pexpr c = new expr(G_CALL,r,0);
                                        c->fct_name = Ncoerce;
                                        init = c;
                                        break;
                                }
                                default:        /* (temp=init,temp.coerce()) */
                                {       Ptable otbl = tbl;
                                        if (Cstmt) { /* make Cstmt into a block */
                                                if (Cstmt->memtbl == 0) Cstmt->memt
                                                tbl = Cstmt->memtbl;
                                        }
                                        char* s = make_name('U');
                                        Pname tmp = new name(s);
                                        tmp->tp = init->tp;
                                        tmp = tmp->dcl(tbl,ARG); /* no init! */
                                        tmp->n_scope = FCT;
                                        Pexpr ass = new expr(ASSIGN,tmp,init);
                                        ass->tp = tt;
                                        Pref r = new ref(DOT,tmp,Ncoerce);
                                        Pexpr c = new expr(G_CALL,r,0);
                                        c->fct_name = Ncoerce;
```

```
                                                init = new expr(CM,ass,c);
                                                tbl = otbl;
                                        }
                                        }
                                        return init->typ(tbl);
                                }
                                Pexpr a = new class expr(ELIST,init,0);
                                a = new class texpr(VALUE,tt,a);
                                a->e2 = nn;
                                a = a->typ(tbl);
/*error('d',"class_init%n: %k %t",nn,a->base,tt);*/
                                return a;
                        }
/*error('d',"class_init%n: init%t",nn,init->base,init->tp);*/
                        return init;
        }

        if (tt->check(init->tp,ASSIGN) && refd==0) {
                error("badIrT:%t (%tX)",init->tp,tt);
                init->tp = any_type;
        }
        return init;
}

int char_to_int(char* s)
/*      assume s points to a string:
                'c'
        or      '\c'
        or      '\0'
        or      '\ddd'
        or multi-character versions of the above
*/
{
        register int i = 0;
        register char c, d, e;

        switch (*s) {
        default:
                error('i',"char constant store corrupted");
        case '`':
                error('s',"bcd constant");
                return 0;
        case '\'':
                break;
        }

        forever                         /* also handle multi-character constants */
        switch (c = *++s) {
        case '\'':
                return i;
        case '\\':                              /* special character */
                switch (c = *++s) {
                case '0': case '1': case '2': case '3': case '4':
                case '5': case '6': case '7':   /* octal representation */
                        c -= '0';
                        switch (d = *++s) {             /* try for 2 */
                        case '0': case '1': case '2': case '3': case '4':
                        case '5': case '6': case '7':
                                d -= '0';
                                switch (e = *++s) {     /* try for 3 */

                                case '0': case '1': case '2': case '3': case '4':
                                case '5': case '6': case '7':
                                        c = c*64+d*8+e-'0';
                                        break;
                                default:
                                        c = c*8+d;
                                        s--;
                                }
                                break;
                        default:
                                s--;
                        }
                        break;
                case 'b':
                        c = '\b';
                        break;
                case 'f':
                        c = '\f';
                        break;
                case 'n':
                        c = '\n';
                        break;
                case 'r':
                        c = '\r';
                        break;
                case 't':
                        c = '\t';
                        break;
                case '\\':
                        c = '\\';
                        break;
                case '\'':
                        c = '\'';
                        break;
                }
                /* no break */
        default:
                if (i) i <<= BI_IN_BYTE;
                i += c;
        }
}

const A10 = 'A'-10;
const a10 = 'a'-10;

int str_to_int(register char* p)
/*
        read decimal, octal, or hexadecimal integer
*/
{
        register c;
        register i = 0;

        if ((c=*p++) == '0') {
                switch (c = *p++) {
                case 0:
                        return 0;

                case 'l':
                case 'L':       /* long zero */
                        return 0;

                case 'x':
                case 'X':       /* hexadecimal */
                        while (c=*p++)
                                switch (c) {
                                case 'l':
                                case 'L':
                                        return i;
                                case 'A':
                                case 'B':
                                case 'C':
                                case 'D':
                                case 'E':
                                case 'F':
                                        i = i*16 + c-A10;
                                        break;
                                case 'a':
                                case 'b':
                                case 'c':
                                case 'd':
                                case 'e':
                                case 'f':
                                        i = i*16 + c-a10;
                                        break;
                                default:
                                        i = i*16 + c-'0';
                                }
                        return i;

                default:        /* octal */
                        do
                                switch (c) {
                                case 'l':
                                case 'L':
                                        return i;
                                default:
                                        i = i*8 + c-'0';
                                }
                        while (c=*p++);
                        return i;
                }
        }
                                /* decimal */
        i = c-'0';
        while (c=*p++)
```

```
                switch (c) {
                case 'l':
                case 'L':
                        return i;
                default:
                        i = i*10 + c-'0';
                }
        return i;


}

char* Neval;

int expr.eval()
{
        if (Neval) return 1;

        switch (base) {
        case ZERO:      return 0;
        case IVAL:      return (int)e1;
        case ICON:      return str_to_int(string);
        case CCON:      return char_to_int(string);
        case FCON:      Neval = "float in constant expression"; return 1;
        case STRING:    Neval = "string in constant expression"; return 1;
        case EOBJ:      return ((Pname)this)->n_val;
        case SIZEOF:    return tp2->tsizeof();
        case NAME:
        {       Pname n = (Pname)this;
                if (n->n_evaluated) return n->n_val;
                Neval = "cannot evaluate constant";
                return 1;
        }
        case ICALL:
                if (e1) {
                        il->i_next = curr_icall;
                        curr_icall = il;
                        int i = e1->eval();
                        curr_icall = il->i_next;
                        return i;
                }
                Neval = "void inlineF";
                return 1;
        case ANAME:
        {       Pname n = (Pname)this;
                int argno = n->n_val;
                Pin il;
                for (il=curr_icall; il; il=il->i_next)
                        if (il->i_table == n->n_table) goto aok;
                goto bok;
        aok:
                if (il->local[argno]) {
        bok:
                        Neval = "inlineF call too complicated for constant expressi
                        return 1;
                }
                Pexpr aa = i1->arg[argno];
                return aa->eval();
        }
        case CAST:
        {       int i = e1->eval();
                Neval = "cast in constant expression";
                return i;
        }
        case UMINUS:
        case NOT:
        case COMPL:
        case PLUS:
        case MINUS:
        case MUL:
        case LS:
        case RS:
        case NE:
        case LT:
        case LE:
        case GT:
        case GE:
        case AND:
        case OR:
        case ER:
        case DIV:
        case MOD:
        case QUEST:
        case EQ:
        case ANDAND:
        case OROR:
                break;
        default:
                Neval = "bad operator in constant expression";
                return 1;
        }

        int i1 = (e1) ? e1->eval() : 0;
        int i2 = (e2) ? e2->eval() : 0;

        switch (base) {
        case UMINUS:    return -i2;
        case NOT:       return !i2;
        case COMPL:     return ~i2;
        case CAST:      return i1;
        case PLUS:      return i1+i2;
        case MINUS:     return i1-i2;
        case MUL:       return i1*i2;
        case LS:        return i1<<i2;
        case RS:        return i1>>i2;
        case NE:        return i1!=i2;
        case EQ:        return i1==i2;
        case LT:        return i1<i2;
        case LE:        return i1<=i2;
        case GT:        return i1>i2;
        case GE:        return i1>=i2;
        case AND:       return i1&i2;
        case OR:        return i1|i2;
        case OROR:      return i1||i2;
        case ER:        return i1^i2;
        case MOD:       return (i2==0) ? 1 : i1%i2;
        case QUEST:     return (cond->eval()) ? i1 : i2;
        case DIV:       if (i2==0) {
                                Neval = "divide by zero";
                                return 1;
                        }
                        return i1/i2;
        }
}

bit classdef.has_friend(Pname f)
/*
        does this class have function "f" as its friend?
*/
{
        Plist l;
        Ptable ctbl = f->n_table;
/*fprintf(stderr,"(%d %s)->has_friend(%d %s)\n",this,string,f,(f)?f->string:""); ffl
        for (l=friend_list; l; l=l->l) {
                Pname fr = l->f;
/*fprintf(stderr,"fr %d %d %d\n",fr,fr->tp,fr->tp->base); fflush(stderr);*/
                switch (fr->tp->base) {
                case CLASS:
                        if (((Pclass)fr->tp)->memtbl == ctbl) return 1;
                        break;
                case COBJ:
                        if (((Pbase)fr->tp)->b_table == ctbl) return 1;
                        break;
                case FCT:
                        if (fr == f) return 1;
                        break;
                case OVERLOAD:
                {/*     Pgen g = (Pgen)fr->tp;
                        Plist ll;
                        for (ll=g->fct_list; ll; ll=ll->l) {
                                if (ll->f == f) return 1;
                        }*/
                        l->f = fr = ((Pgen)fr->tp)->fct_list->f; /* first fct */
                        if (fr == f) return 1;
                        break;
                }
                default:
                        error('i',"bad friend %k",fr->tp->base);
                }
        }
        return 0;
}
```

```
/* %Z% %M% %I% %H% %T% */
/*************************************************************************

        C++ source for cfront, the C++ compiler front-end
        written in the computer science research center of Bell Labs

        Copyright (c) 1984 AT&T Technologies, Inc. All rigths Reserved
        THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

        If you ignore this notice the ghost of Ma Bell will haunt you forever.
gram.y:

        This is the syntax analyser.

        Old C features not recognized:
        (1) "+ =" as the operator "+="
        (2) any construct using one of the new keywords as an identifier
        (3) initializers without "=" operator
        (4) structure tags used as identifier names

        Additions:
        (1) Classes (keywords: CLASS THIS PUBLIC FRIEND and VIRTUAL)
                (classes incorporate STRUCT and UNION)
        (2) the new and delete operators (keywords: NEW DELETE)
        (3) inline functions (keyword INLINE)
        (4) overloaded function names (keyword OVERLOAD)
        (5) overloaded operators (keyword OPERATOR)
        (6) constructors and destructors
        (7) constant types (keyword: CONST)
        (8) argument types part of function function type (token: ...)
        (9) new argument syntax ( e.g. char f(int a, char b) { ... })
        (10) names can be left out of argument lists

        Syntax extensions for error handling:
        (1) nested functions
        (2) any expression can be empty
        (3) any expression can be a constant_expression

        note that a call to error() does not change the parser's state
*/

%{
#include "size.h"
#include "cfront.h"

#define YYMAXDEPTH 300

Pbase defa_type;
Pbase moe_type;
Pexpr dummy;
Pexpr zero;

Pclass ccl;
int cdi = 0;
Pname cd = 0, cd_vec[BLMAX];
```

```
char stmt_seen = 0, stmt_vec[BLMAX];
Plist modified_tn = 0, tn_vec[BLMAX];

Pname sta_name = (Pname)&sta_name;

bit cm_warn;

extern TOK back;
TOK back;
#define lex_unget(x) back = x

#define Ndata(a,b)       ((Pname)b)->normalize((Pbase)a,0,0)
#define Ncast(a,b)       ((Pname)b)->normalize((Pbase)a,0,1)
#define Nfct(a,b,c)      ((Pname)b)->normalize((Pbase)a,(Pblock)c,0)
#define Ntype(p)         ((Pname)p)->tp
#define Nstclass(p)      ((Pname)p)->n_stclass
#define Nlist(p)         ((Pname)p)->n_list
#define Ncopy(n)         (((Pname)n)->base==TNAME) ? new name(((Pname)n)->string) :
(Pname)
#define Nhide(n)         ((Pname)n)->hide()
/*      #define Ntname(t,n)      ((Pname)n)->tname(t)     */

#define fieldN(e)        new basetype(FIELD,(Pname)e)
#define enumdefN(m)      new enumdef(m)
#define Fargtype(p)      ((Pfct)p)->argtype
#define Finit(p)         ((Pfct)p)->f_init
#define Finline(p)       ((Pfct)p)->f_inline = 1
#define Fargdcl(p,q)     ((Pfct)p)->argdcl(q)
#define Freturns(p)      ((Pfct)p)->returns
#define fctN(t,a,k)      new fct(t,a,k)
#define vecN(e)          new vec(0,e)
#define Vtype(v)         ((Pvec)v)->typ
#define Ptyp(p)          ((Pptr)p)->typ

#define conN(t,v)        new expr(t,(Pexpr)v,0)

#define nlistN(n)        (PP)new nlist((Pname)n)
#define Nadd(l,n)        ((class nlist *)l)->add((Pname)n)
#define Nadd_list(l,n)   ((class nlist *)l)->add_list((Pname)n)
#define Nunlist(l)       name_unlist((nlist*)l)
#define slistN(s)        (PP)new slist((Pstmt)s)
#define Sadd(l,s)        ((slist*)l)->add((Pstmt)s)
#define Sunlist(l)       stmt_unlist((slist*)l)
#define Eadd(l,e)        ((elist*)l)->add((Pexpr)e)
#define Eunlist(l)       expr_unlist((elist*)l)

                /* avoid redefinitions */
#undef EOFTOK
#undef ASM
#undef BREAK
#undef CASE
#undef CONTINUE
#undef DEFAULT
#undef DELETE
#undef DO
#undef ELSE
#undef ENUM
```

```
#undef FOR
#undef FORTRAN
#undef GOTO
#undef IF
#undef NEW
#undef OPERATOR
#undef PUBLIC
#undef RETURN
#undef SIZEOF
#undef SWITCH
#undef THIS
#undef WHILE
#undef LP
#undef RP
#undef LB
#undef RB
#undef REF
#undef DOT
#undef NOT
#undef COMPL
#undef MUL
#undef AND
#undef PLUS
#undef MINUS
#undef ER
#undef OR
#undef ANDAND
#undef OROR
#undef QUEST
#undef COLON
#undef ASSIGN
#undef CM
#undef SM
#undef LC
#undef RC
#undef ID
#undef STRING
#undef ICON
#undef FCON
#undef CCON
#undef ZERO
#undef ASOP
#undef RELOP
#undef EQUOP
#undef DIVOP
#undef SHIFTOP
#undef ICOP
#undef TYPE
#undef TNAME
#undef EMPTY
#undef NO_ID
#undef NO_EXPR
#undef ELLIPSIS
#undef AGGR
#undef MEM
#undef CAST
```

```
Pname syn()
{
        return (Pname) yyparse();
}

%}

%union {
        char*   s;
        TOK     t;
        int     i;
        loc     l;
        Pname   pn;
        Ptype   pt;
        Pexpr   pe;
        Pstmt   ps;
        Pbase   pb;
        PP      p;      /* fudge: pointer to all class node objects
                                neccessary only because unions of class
                                pointers are not implemented by cpre
                        */
}
%{
extern YYSTYPE yylval;
%}
/*
        the token definitions are copied from token.h,
        and all %token replaced by %token
*/
                        /* keywords in alphabetical order */
%token EOFTOK           0
%token ASM              1
%token BREAK            3
%token CASE             4
%token CONTINUE         7
%token DEFAULT          8
%token DELETE           9
%token DO               10
%token ELSE             12
%token ENUM             13
%token FOR              16
%token FORTRAN          17
%token GOTO             19
%token IF               20
%token NEW              23
%token OPERATOR         24
%token PUBLIC           25
%token RETURN           28
%token SIZEOF           30
%token SWITCH           33
%token THIS             34
%token WHILE            39

                        /* operators in priority order (sort of) */
%token LP               40
```

```
%token RP               41
%token LB               42
%token RB               43
%token REF              44
%token DOT              45
%token NOT              46
%token COMPL            47
%token MUL              50
%token AND              52
%token PLUS             54
%token MINUS            55
%token ER               64
%token OR               65
%token ANDAND           66
%token OROR             67
%token QUEST            68
%token COLON            69
%token ASSIGN           70
%token CM               71
%token SM               72
%token LC               73
%token RC               74
%token CAST             113

                        /* constants etc. */
%token ID               80
%token STRING           81
%token ICON             82
%token FCON             83
%token CCON             84

%token ZERO             86

                        /* groups of tokens */
%token ASOP             90      /* op= */
%token RELOP            91      /* LE GE LT GT */
%token EQUOP            92      /* EQ NE */
%token DIVOP            93      /* DIV MOD */
%token SHIFTOP          94      /* LS RS */
%token ICOP             95      /* INCR DECR */

%token TYPE             97      /*      INT FLOAT CHAR DOUBLE
                                        REGISTER STATIC EXTERN AUTO
                                        CONST INLINE VIRTUAL FRIEND
                                        LONG SHORT UNSIGNED
                                        TYPEDEF */
%token TNAME            123
%token EMPTY            124
%token NO_ID            125
%token NO_EXPR          126
%token ELLIPSIS         155     /* ... */
%token AGGR             156     /* CLASS STRUCT UNION */
%token MEM              160     /* :: */


%type <p>       external_def fct_dcl fct_def att_fct_def arg_dcl_list
```

```
                base_init
                data_dcl ext_def vec ptr
                type tp enum_dcl moe_list
                moe
                tag class_head class_dcl mem_list cl_mem_list
                cl_mem dl decl_list
                fname decl initializer stmt_list
                block statement simple ex_list elist e  term prim
                cast_decl cast_type c_decl c_type c_tp
                arg_decl at arg_type arg_list arg_type_list
                new_decl new_type
                condition
                TNAME tn_list
%type <l>       LC RC SWITCH CASE DEFAULT FOR IF DO WHILE GOTO RETURN DELETE
                BREAK CONTINUE
%type <t>       oper
                EQUOP DIVOP SHIFTOP ICOP RELOP ASOP
                ANDAND OROR PLUS MINUS MUL ASSIGN OR ER AND
                LP LB NOT COMPL AGGR
                TYPE
%type <s>       ID CCON ZERO ICON FCON STRING

%left   EMPTY
%left   NO_ID
%left   RC LC ID BREAK CONTINUE RETURN GOTO DELETE DO IF WHILE FOR CASE DEFAULT
        AGGR ENUM TYPE
%left   NO_EXPR

%left   CM
%right  ASOP ASSIGN
%right  QUEST COLON
%left   OROR
%left   ANDAND
%left   OR
%left   ER
%left   AND
%left   EQUOP
%left   RELOP
%left   SHIFTOP
%left   PLUS MINUS
%left   MUL DIVOP
%right  NOT COMPL NEW
%right  CAST ICOP SIZEOF
%left   LB LP DOT REF MEM

%start ext_def

%%
/*
        this parser handles declarations one by one,
        NOT a complete .c file
*/
```

```
/*************** DECLARATIONS in the outermost scope: returns Pname *******/

ext_def         :  external_def
                        {       return $<i>1; }
                |  SM
                        {       return 1; }
                |  EOFTOK
                        {       return 0; }
                ;

external_def    :  data_dcl
                        {       modified_tn = 0; if ($<pn>1==0) $<i>$ = 1; }
                |  att_fct_def
                        {       goto mod; }
                |  fct_def
                        {       goto mod; }
                |  fct_dcl
                        { mod:  if (modified_tn) {
                                        restore();
                                        modified_tn = 0;
                                }
                        }
                |  ASM LP STRING RP SM
                        {       Pname n = new name(make_name('A'));
                                n->tp = new basetype(ASM,$<pn>3);
                                $$ = n;
                        }
                ;

fct_dcl         :  decl SM
                        {       Pname n = $<pn>1;
                                switch (n->tp->base) {
                                case FCT:
                                        $$ = Nfct(defa_type,n,0);
                                        break;
                                default:
                                        error("TX for%n",n);
                                        $$ = Ndata(defa_type,$1);
                                }
                        }
                ;

att_fct_def     :  type decl arg_dcl_list base_init block
                        {       $$ = Nfct($1,$2,$5);
                                Fargdcl(Ntype($$),Nunlist($3));
                                Finit(Ntype($$)) = $<pe>4;
                        }
                ;

fct_def         :  decl arg_dcl_list base_init block
                        {       $$ = Nfct(defa_type,$1,$4);
                                Fargdcl(Ntype($$),Nunlist($2));
                                Finit(Ntype($$)) = $<pe>3;
                                }
                        ;

base_init               :   COLON LP elist RP
                                {       $$ = $3; }
                        |   %prec EMPTY
                                {       $$ = 0; }
                        ;



/**************** declarations: returns Pname ********************/

arg_dcl_list            :   arg_dcl_list data_dcl
                                {       if ($<pn>2 == 0)
                                                error("badAD");
                                        else if ($<pn>2->tp->base == FCT)
                                                error("FD inAL (%n)",$<pn>2);
                                        else if ($1)
                                                Nadd_list($1,$2);
                                        else
                                                $$ = nlistN($2);
                                }
                        |   %prec EMPTY
                                {       $$ = 0; }
                        ;

dl                      :   decl
                        |   ID COLON e                  %prec CM
                                {       $$ = new name($<s>1);
                                        Ntype($$) = fieldN($<pe>3);
                                }
                        |   COLON e                     %prec CM
                                {       $$ = new name;
                                        Ntype($$) = fieldN($<pe>2);
                                }
                        |   decl ASSIGN initializer
                                {       $<pn>1->n_initializer = $<pe>3; }
                        ;

decl_list               :   dl
                                {       if ($1) $$ = nlistN($1); }
                        |   decl_list CM dl
                                {       if ($1)
                                                if ($3)
                                                        Nadd($1,$3);
                                                else
                                                        error("DL syntax");
                                        else {
                                                if ($3) $$ = nlistN($3);
                                                error("DL syntax");
                                        }
                                }
                        ;
```

```
data_dcl        :   type decl_list SM
                        {       $$ = Ndata($1,Nunlist($2)); }
                |   type SM
                        {       $$ = $<pb>1->aggr(); }
                ;

tp              :   TYPE
                        {       $$ = new basetype($<t>1,0); }
                |   TNAME
                        {       $$ = new basetype(TYPE,$<pn>1); }
                |   class_dcl
                |   enum_dcl
                |   AGGR tag
                        {       Pname n = $<pn>2;
                                TOK t = $<t>1;
                                if (n->base == NAME) {  /* implicit dcl */
                                        n = n->tname(t);
                                        modified_tn = modified_tn->l;   /* not loca
                                        n->lex_level = 0;
                                }
                                $$ = n->tp;
                        }
                |   ENUM tag
                        {       Pname n = $<pn>2;
                                if (n->base == NAME) {  /* implicit dcl */
                                        n = n->tname(ENUM);
                                        modified_tn = modified_tn->l;
                                        n->lex_level = 0;
                                }
                                $$ = n->tp;
                        }
                ;

type            :   tp
                |   type TYPE
                        {       $$ = $<pb>1->type_adj($<t>2); }
                |   type TNAME
                        {       $$ = $<pb>1->name_adj($<pn>2); }
                |   type class_dcl
                        {       $$ = $<pb>1->base_adj($<pb>2); }
                |   type enum_dcl
                        {       $$ = $<pb>1->base_adj($<pb>2); }
                |   type AGGR tag
                        {       Pname n = $<pn>3;
                                TOK t = $<t>2;
                                if (n->base == NAME) {  /* implicit dcl */
                                        n = n->tname(t);
                                        modified_tn = modified_tn->l;
                                        n->lex_level = 0;
                                }
                                $$ = $<pb>1->base_adj((Pbase)n->tp);
                        }
                |   type ENUM tag
                        {       Pname n = $<pn>3;
                                if (n->base == NAME) {  /* implicit dcl */
                                        n = n->tname(ENUM);
```

```
                                                modified_tn = modified_tn->l;
                                                n->lex_level = 0;
                                        }
                                        $$ = $<pb>1->base_adj((Pbase)n->tp);
                                }
                        ;



/****************** aggregate: returns Pname ******************/

enum_dcl        :       ENUM LC moe_list RC
                                {       $$ = end_enum(0,$<pn>3); }
                |       ENUM tag LC moe_list RC
                                {       $$ = end_enum($<pn>2,$<pn>4); }
                ;

moe_list        :       moe
                                {       if ($1) $$ = nlistN($1); }
                |       moe_list CM moe
                                {       if( $3) if ($1) Nadd($1,$3); else $$ = nlistN($3);
                ;

moe             :       ID
                                {       $$ = new name($<s>1); Ntype($$) = moe_type; }
                |       ID ASSIGN e
                                {       $$ = new name($<s>1);
                                        Ntype($$) = moe_type;
                                        $<pn>$->n_initializer = $<pe>3;
                                }
                |       /* empty */
                                {       $$ = 0; }
                ;


class_dcl       :       class_head mem_list RC
                                {       end_cl(); }
                |       class_head mem_list RC TYPE
                                {       end_cl();
                                        error("`;' or declaratorX afterCD");
                                        lex_unget($4);
                                        /* lex_unget($4); but only one unget, sorry */
                                 }
                ;

class_head      :       AGGR LC
                                {       $$ = start_cl($<t>1,0,0); }
                |       AGGR tag LC
                                {       $$ = start_cl($<t>1,$<pn>2,0); }
                |       AGGR tag COLON tag LC
                                {       $$ = start_cl($<t>1,$<pn>2,$<pn>4); }
                |       AGGR tag COLON PUBLIC tag LC
                                {       $$ = start_cl($<t>1,$<pn>2,$<pn>5);
                                        ccl->pubbase = 1;
                                }
                        ;

tag                     :  ID
                                {       $$ = new name($<s>1); }
                        |  TNAME
                        ;

mem_list                :  cl_mem_list
                                {       Pname n = Nunlist($1);
                                        if (ccl->is_simple())
                                                ccl->pubmem = n;
                                        else
                                                ccl->privmem = n;
                                        $$ = 0;
                                }
                        |  cl_mem_list PUBLIC cl_mem_list
                                {       error("``:'' missing after ``public''");
                                        ccl->pubmem = Nunlist($3);
                                        goto priv;
                                }
                        |  cl_mem_list PUBLIC COLON cl_mem_list
                                {       TOK t;
                                        ccl->pubmem = Nunlist($4);
                                priv:
                                        t = ccl->is_simple();
                                        if (t) error("public in%k",t);
                                        ccl->privmem = Nunlist($1);
                                        $$ = 0;
                                 }
                        ;

cl_mem_list             :  cl_mem_list cl_mem
                                {       if ($2) if ($1) Nadd_list($1,$2); else $$ = nlistN(
                        |  %prec EMPTY
                                {       $$ = 0; }
                        ;

cl_mem                  :  data_dcl
                        |  att_fct_def SM
                        |  att_fct_def
                        |  fct_def SM
                        |  fct_def
                        |  fct_dcl
                        |  tn_list tag SM       /* public declaration */
                                {       Pname n = Ncopy($2);
                                        n->n_qualifier = (Pname)$1;
                                        n->n_list = ccl->pubdef;
                                        ccl->pubdef = n;
                                        $$ = 0;
                                }
                        ;


/************** declarators:     returns Pname **********************/
```

```
/*      a ``decl'' is used for function and data declarations,
                and for member declarations
                (it has a name)
        an ``arg_decl'' is used for argument declarations
                (it may or may not have a name)
        an ``cast_decl'' is used for casts
                (it does not have a name)
        a ``new_decl'' is used for type specifiers for the NEW operator
                (it does not have a name, and PtoF and PtoV cannot be expressed)
*/

fname           :  ID
                        {       $$ = new name($<s>1); }
                |  COMPL TNAME
                        {       $$ = Ncopy($2); $<pn>$->n_oper = DTOR; }
                |  DELETE
                        {       if (fct_void==0) error("deleteF (use destructor)");
                                $$ = new name("_dtor");
                                $<pn>$->n_oper = DTOR;
                        }
                |  NEW
                        {       if (fct_void==0) error("newF (use constructor)");
                                $$ = new name("_ctor");
                                $<pn>$->n_oper = CTOR;
                        }
                |  OPERATOR oper
                        {       $$ = new name(oper_name($2));
                                $<pn>$->n_oper = $<t>2;
                        }
                |  OPERATOR c_type
                        {       Pname n = $<pn>2;
                                n->string = "_type";
                                n->n_oper = TYPE;
                                n->n_initializer = (Pexpr)n->tp;
                                n->tp = 0;
                                $$ = n;
                        }
                ;

oper            :  PLUS
                |  MINUS
                |  MUL
                |  AND
                |  OR
                |  ER
                |  SHIFTOP
                |  EQUOP
                |  DIVOP
                |  RELOP
                |  ANDAND
                |  OROR
                |  LP RP        {       $$ = CALL; }
                |  LB RB        {       $$ = DEREF; }
                |  NOT
                |  COMPL
```

```
                        |  ICOP
                        |  ASOP
                        |  ASSIGN
                        |  NEW            {       $$ = NEW; }
                        |  DELETE         {       $$ = DELETE; }
                        ;

tn_list                 :  TNAME DOT
                        |  tn_list TNAME DOT
                                {       error('s',"MF of nestedC"); }
                        |  tn_list ID DOT
                                {       error('s',"MF of nestedC"); }
                        ;

decl                    :  decl arg_list
                                {       Freturns($2) = Ntype($1);
                                        Ntype($1) = (Ptype)$2;
                                }
                        |  TNAME arg_list
                                {       Pname n = (Pname)$1;
                                        $$ = Ncopy(n);
                                        if (ccl && strcmp(n->string,ccl->string)) Nhide(n);
                                        $<pn>$->n_oper = TNAME;
                                        Freturns($2) = Ntype($$);
                                        Ntype($$) = (Ptype)$2;
                                }
                        |  decl LP elist RP
                                /*      may be class object initializer,
                                        class object vector initializer,
                                        if not elist will be a CM or an ID
                                */
                                {       TOK k = 1;
                                        Pname l = $<pn>3;
                                        if (fct_void && l==0) k = 0;
                                        Ntype($1) = fctN(Ntype($1),l,k);
                                }
                        |  TNAME LP elist RP
                                {       TOK k = 1;
                                        Pname l = $<pn>3;
                                        if (fct_void && l==0) k = 0;
                                        $$ = Ncopy($1);
                                        $<pn>$->n_oper = TNAME;
                                        Ntype($$) = fctN(0,l,k);
                                }

                        |  fname
                        |  ID DOT fname
                                {       $$ = Ncopy($3);
                                        $<pn>$->n_qualifier = new name($<s>1);
                                }
                        |  tn_list fname
                                {       $$ = $2;
                                        set_scope($<pn>1);
                                        $<pn>$->n_qualifier = $<pn>1;
                                }
                        |  tn_list TNAME
```

```
                {       $$ = Ncopy($2);
                        set_scope($<pn>1);
                        $<pn>$->n_oper = TNAME;
                        $<pn>$->n_qualifier = $<pn>1;
                }
        |  ptr decl     %prec MUL
                {       Ptyp($1) = Ntype($2);
                        Ntype($2) = (Ptype)$1;
                        $$ = $2;
                }
        |  ptr TNAME    %prec MUL
                {       $$ = Ncopy($2);
                        $<pn>$->n_oper = TNAME;
                        Nhide($2);
                        Ntype($$) = (Ptype)$1;
                }
        |  TNAME vec    %prec LB
                {       $$ = Ncopy($1);
                        $<pn>$->n_oper = TNAME;
                        Nhide($1);
                        Ntype($$) = (Ptype)$2;
                }
        |  decl vec     %prec LB
                {       Vtype($2) = Ntype($1);
                        Ntype($1) = (Ptype)$2;
                }
        |  LP decl RP arg_list  /* xxxxx need a CAST here? */
                {       Freturns($4) = Ntype($2);
                        Ntype($2) = (Ptype)$4;
                        $$ = $2;
                }
        |  LP decl RP vec /* xxx */
                {       Vtype($4) = Ntype($2);
                        Ntype($2) = (Ptype)$4;
                        $$ = $2;
                }
        ;

arg_decl        :  ID
                {       $$ = new name($<s>1); }
        |  %prec NO_ID
                {       $$ = new name; }
        |  ptr arg_decl         %prec MUL
                {       Ptyp($1) = Ntype($2);
                        Ntype($2) = (Ptype)$1;
                        $$ = $2;
                }
        |  arg_decl vec         %prec LB
                {       Vtype($2) = Ntype($1);
                        Ntype($1) = (Ptype)$2;
                }
        |  LP arg_decl RP arg_list
                {       Freturns($4) = Ntype($2);
                        Ntype($2) = (Ptype)$4;
                        $$ = $2;
                }
                | LP arg_decl RP vec
                        {       Vtype($4) = Ntype($2);
                                Ntype($2) = (Ptype)$4;
                                $$ = $2;
                        }
                ;

new_decl        : %prec NO_ID
                        {       $$ = new name; }
                | ptr new_decl          %prec MUL
                        {       Ptyp($1) = Ntype($2);
                                Ntype($2) = (Ptype)$1;
                                $$ = $2;
                        }
                | new_decl vec          %prec LB
                        {       Vtype($2) = Ntype($1);
                                Ntype($1) = (Ptype)$2;
                        }
                ;

cast_decl       :  %prec NO_ID
                        {       $$ = new name; }
                | ptr cast_decl                         %prec MUL
                        {       Ptyp($1) = Ntype($2);
                                Ntype($2) = (Ptype)$1;
                                $$ = $2;
                        }
                | cast_decl vec                         %prec LB
                        {       Vtype($2) = Ntype($1);
                                Ntype($1) = (Ptype)$2;
                        }
                | LP cast_decl RP arg_list
                        {       Freturns($4) = Ntype($2);
                                Ntype($2) = $<pt>4;
                                $$ = $2;
                        }
                | LP cast_decl RP vec
                        {       Vtype($4) = Ntype($2);
                                Ntype($2) = $<pt>4;
                                $$ = $2;
                        }
                ;

c_decl          :  %prec NO_ID
                        {       $$ = new name; }
                | ptr c_decl                            %prec MUL
                        {       Ptyp($1) = Ntype($2);
                                Ntype($2) = (Ptype)$1;
                                $$ = $2;
                        }
                ;


/***************** statements: returns Pstmt *****************/
```

```
stmt_list       :  stmt_list statement
                        {       if ($2)
                                        if ($1)
                                                Sadd($1,$2);
                                        else {
                                                $$ = slistN($2);
                                                stmt_seen = 1;
                                        }
                        }
                |  statement
                        {       if ($1) {
                                        $$ = slistN($1);
                                        stmt_seen = 1;
                                }
                        }
                ;

condition       :  LP e RP
                        {       $$ = $2;
                                if ($<pe>$ == dummy) error("empty condition");
                                stmt_seen = 1;
                        }
                ;

block           :  LC
                        {       cd_vec[cdi] = cd;
                                stmt_vec[cdi] = stmt_seen;
                                tn_vec[cdi++] = modified_tn;
                                cd = 0;
                                stmt_seen = 0;
                                modified_tn = 0;
                        }
                        stmt_list RC
                        {       Pname n = Nunlist(cd);
                                Pstmt ss = Sunlist($3);
                                $$ = new block($<l>1,n,ss);
                                if (modified_tn) restore();
                                cd = cd_vec[--cdi];
                                stmt_seen = stmt_vec[cdi];
                                modified_tn = tn_vec[cdi];
                                if (cdi < 0) error('i',"block level(%d)",cdi);
                        }
                |  LC RC
                        {       $$ = new block($<l>1,0,0); }
                |  LC error RC
                        {       $$ = new block($<l>1,0,0); }
                ;

simple          :  e
                        {       $$ = new estmt(SM,curloc,$<pe>1,0);     }
                |  BREAK
                        {       $$ = new stmt(BREAK,$<l>1,0); }
                |  CONTINUE
                        {       $$ = new stmt(CONTINUE,$<l>1,0); }
                |  RETURN e
                        {       $$ = new estmt(RETURN,$<l>1,$<pe>2,0); }
                | GOTO ID
                        {       Pname n = new name($<s>2);
                                $$ = new lstmt(GOTO,$<l>1,n,0);
                        }
                | DELETE e
                        {       $$ = new estmt(DELETE,$<l>1,$<pe>2,0); }
                | DO { stmt_seen=1; } statement WHILE condition
                        {       $$ = new estmt(DO,$<l>1,$<pe>5,$<ps>3); }
                ;

statement       : simple SM
                | ASM LP STRING RP SM
                        {
                                if (stmt_seen)
                                        $$ = new estmt(ASM,curloc,(Pexpr)$<s>3,0);
                                else {
                                        Pname n = new name(make_name('A'));
                                        n->tp = new basetype(ASM,(Pname)$<s>3);
                                        if (cd) Nadd_list(cd,n); else cd=(Pname)nli
                                        $$ = 0;
                                }
                        }
        /*      | simple
                        {       error("';' missing after simpleS"); }*/
                | data_dcl
                        {
                                if ($<pn>1)
                                if (stmt_seen) {
                                        Pname n = $<pn>1;
                                        $$ = new block(n->where,n,0);
                                        $<ps>$->base = DCL;
                                }
                                else
                                        goto dddd;
                        }
                | att_fct_def
                        {
                                lex_unget(RC);
                                error("nestedFD (did you forget a ``}''?)");
                        dddd:
                                if (cd) Nadd_list(cd,$1); else cd = (Pname)nlistN($
                                $$ = 0;
                        }
                | block
                | IF condition statement
                        {       $$ = new ifstmt($<l>1,$<pe>2,$<ps>3,0); }
                | IF condition statement ELSE statement
                        {       $$ = new ifstmt($<l>1,$<pe>2,$<ps>3,$<ps>5); }
                | WHILE condition statement
                        {       $$ = new estmt(WHILE,$<l>1,$<pe>2,$<ps>3); }
                /*|  FOR LP { stmt_seen=1; cm_warn++; } e SM e SM e RP statement
                        {       $$ = new forstmt($<l>1,$<pe>4,$<pe>6,$<pe>8,$<ps>10
                                cm_warn--;
                        }*/
                | FOR LP { stmt_seen=1; cm_warn++; } statement e SM e RP statement
                        {       $$ = new forstmt($<l>1,$<ps>4,$<pe>5,$<pe>7,$<ps>9)
```

```
                                cm_warn--;
                        }
                |  FOR CAST { stmt_seen=1; cm_warn++; } statement e SM e RP stateme
                        {       $$ = new forstmt($<l>1,$<ps>4,$<pe>5,$<pe>7,$<ps>9)
                                cm_warn--;
                        }
                |  SWITCH condition statement
                        {       $$ = new estmt(SWITCH,$<l>1,$<pe>2,$<ps>3); }
                |  ID COLON { $$ = new name($<s>1); stmt_seen=1; } statement
                        {       Pname n = $<pn>3;
                                $$ = new lstmt(LABEL,n->where,n,$<ps>4);
                        }
                |  CASE { stmt_seen=1; } e COLON statement
                        {       if ($<pe>3 == dummy) error("empty case label");
                                $$ = new estmt(CASE,$<l>1,$<pe>3,$<ps>5);
                        }
                |  DEFAULT COLON { stmt_seen=1; } statement
                        {       $$ = new stmt(DEFAULT,$<l>1,$<ps>4); }
                ;


/*********************** expressions: returns Pexpr ***************/


elist           : ex_list
                        {       Pexpr e = Eunlist($1);
                                while (e && e->e1==dummy) {
                                        if (e->e2) error("EX inEL");
                                        delete e;
                                        e = e->e2;
                                }
                                $$ = e;
                        }

ex_list         :  initializer          %prec CM
                        {       Pexpr e = new expr(ELIST,$<pe>1,0);
                                $$ = (PP)new elist(e);
                        }
                |  ex_list CM initializer
                        {       Pexpr e = new expr(ELIST,$<pe>3,0);
                                Eadd($1,e);
                        }
                ;

initializer     :  e                            %prec CM
                |  LC elist RC
                        {       Pexpr e;
                                if ($2)
                                        e = $<pe>2;
                                else
                                        e = new expr(ELIST,dummy,0);
                                $$ = new expr(ILIST,e,0);
                        }
                ;
```

```
e               :  e ASSIGN e
                        {       binop:  $$ = new expr($<t>2,$<pe>1,$<pe>3); }
                |  e PLUS e     {       goto binop; }
                |  e MINUS e    {       goto binop; }
                |  e MUL e      {       goto binop; }
                |  e AND e      {       goto binop; }
                |  e OR e       {       goto binop; }
                |  e ER e       {       goto binop; }
                |  e SHIFTOP e  {       goto binop; }
                |  e EQUOP e    {       goto binop; }
                |  e DIVOP e    {       goto binop; }
                |  e RELOP e    {       goto binop; }
                |  e ANDAND e   {       goto binop; }
                |  e OROR e     {       goto binop; }
                |  e ASOP e     {       goto binop; }
                |  e CM e
                        {       if (cm_warn==0) error('w',"comma not in parentheses
                                goto binop;
                        }
                |  e QUEST e COLON e
                        {       $$ = new qexpr($<pe>1,$<pe>3,$<pe>5); }
                |  term
                ;

term            :  TYPE LP elist RP
                        {       TOK b = $<t>1;
                                Ptype t;
                                switch (b) {
                                case CHAR:      t = char_type; break;
                                case SHORT:     t = short_type; break;
                                case INT:       t = int_type; break;
                                case LONG:      t = long_type; break;
                                case UNSIGNED:  t = uint_type; break;
                                case FLOAT:     t = float_type; break;
                                case DOUBLE:    t = double_type; break;
                                case VOID:      t = void_type; break;
                                default:
                                        error("illegal constructor:%k",b);
                                        t = int_type;
                                }
                                $$ = new texpr(VALUE,t,$<pe>3);
                        }
                |  TNAME LP elist RP
                        {       Ptype t = Ntype($1);
                                $$ = new texpr(VALUE,t,$<pe>3);
                        }
                |  NEW new_type
                        {       Ptype t = Ntype($2); $$ = new texpr(NEW,t,0); }
                |  NEW LP new_type RP
                        {       Ptype t = Ntype($3); $$ = new texpr(NEW,t,0); }
        /*      |  NEW new_type LP elist RP
                        {       Ptype t = Ntype($2); $$ = new texpr(NEW,t,$<pe>4);
                |  NEW LP new_type LP elist RP RP
                        {       Ptype t = Ntype($3); $$ = new texpr(NEW,t,$<pe>5);
```

```
*/      |   term ICOP
                {       $$ = new expr($<t>2,$<pe>1,0); }
        |   CAST cast_type RP term    /* lex() returns CAST instead of LP */
                {       Ptype t = Ntype($2);
                        $$ = new texpr(CAST,t,$<pe>4);
                }
        |   MUL term
                {       $$ = new expr(DEREF,$<pe>2,0); }
        |   AND term
                {       $$ = new expr(ADDROF,0,$<pe>2); }
        |   MINUS term
                {       $$ = new expr(UMINUS,0,$<pe>2); }
        |   NOT term
                {       $$ = new expr(NOT,0,$<pe>2); }
        |   COMPL term
                {       $$ = new expr(COMPL,0,$<pe>2); }
        |   ICOP term
                {       $$ = new expr($<t>1,0,$<pe>2); }
        |   SIZEOF term
                {       Pexpr e = $<pe>2;
                        if (e->base == CAST) {
                                Pexpr ee = e->e1;
                                TOK k = ee->base;
                                switch (k) {
                                case UMINUS:
                                        ee = new expr(MINUS,e,ee->e2);
                                        goto kk;
                                case DEREF:
                                        if (ee->e2) goto dd;
                                        ee = new expr(MUL,e,ee->e1);
                                        goto kk;
                                case ADDROF:
                                        ee = new expr(AND,e,ee->e2);
                                kk:
                                        e->base = SIZEOF;
                                        e->e1 = 0;
                                        $$ = ee;
                                        break;
                                default:
                                dd:
                                        e->base = SIZEOF;
                                        $$ = $2;
                                }
                        }
                        else
                                $$ = new texpr(SIZEOF,0,e);
                }
        |   term LB e RB
                {       $$ = new expr(DEREF,$<pe>1,$<pe>3); }
        |   term LP elist RP
                {       Pexpr ee = $<pe>3;
                        Pexpr e = $<pe>1;
                        if (e->base == NEW)
                                e->e1 = ee;
                        else
                                $$ = new call(e,ee);
                                }
                        |       term REF prim
                                {       $$ = new ref(REF,$<pe>1,$<pn>3); }
                        |       term REF TNAME
                                {       Pname n = Ncopy($3); $$ = new ref(REF,$<pe>1,n); }
                        |       term DOT prim
                                {       $$ = new ref(DOT,$<pe>1,$<pn>3); }
                        |       term DOT TNAME
                                {       Pname n = Ncopy($3); $$ = new ref(DOT,$<pe>1,n); }
                        |       MEM tag
                                {       $$ = Ncopy($2); $<pn>$->n_qualifier = sta_name; }
                        |       prim
                        |       LP { cm_warn++; } e RP
                                { $$ = $3; cm_warn--; }
                        |       ZERO
                                {       $$ = zero; }
                        |       ICON
                                {       $$ = conN(ICON,$1); }
                        |       FCON
                                {       $$ = conN(FCON,$1); }
                        |       STRING
                                {       $$ = conN(STRING,$1); }
                        |       CCON
                                {       $$ = conN(CCON,$1); }
                        |       THIS
                                {       $$ = conN(THIS,0); }
                        |       %prec NO_EXPR
                                {       $$ = dummy; }
                        ;


prim                    :       ID
                                {       $$ = new name($<s>1); }
                        |       TNAME MEM tag
                                {       $$ = Ncopy($3);
                                        $<pn>$->n_qualifier = $<pn>1;
                                }
                        |       OPERATOR oper
                                {       $$ = new name(oper_name($2));
                                        $<pn>$->n_oper = $<t>2;
                                }
                        |       TNAME MEM OPERATOR oper
                                {       $$ = new name(oper_name($4));
                                        $<pn>$->n_oper = $<t>4;
                                        $<pn>$->n_qualifier = $<pn>1;
                                }
                        ;

/****************** abstract types (return type Pname) *************/

cast_type               :       type cast_decl
                                {       $$ = Ncast($1,$2); }
                        ;

c_tp                    :       TYPE
```

```
                                {       $$ = new basetype($<t>1,0); }
                        | TNAME
                                {       $$ = new basetype(TYPE,$<pn>1); }
                        ;

c_type                  : c_tp c_decl
                                {       $$ = Ncast($1,$2); }
                        ;

new_type                : type new_decl
                                {       $$ = Ncast($1,$2); }
                        ;

arg_type                : type arg_decl
                                {       $$ = Ndata($1,$2); }
                        | type arg_decl ASSIGN initializer
                                {       $$ = Ndata($1,$2);
                                        $<pn>$->n_initializer = $<pe>4;
                                }
                        ;

arg_list                : CAST arg_type_list RP
                                {       TOK k = 1;
                                        Pname l = $<pn>2;
                                        if (fct_void && l==0) k = 0;
                                        $$ = fctN(0,Nunlist(l),k);
                                }
                        | CAST arg_type_list ELLIPSIS RP
                                {       TOK k = ELLIPSIS;
                                        Pname l = $<pn>2;
                                        if (fct_void && l==0) k = 0;
                                        $$ = fctN(0,Nunlist(l),k);
                                }
                        | CAST arg_type_list CM ELLIPSIS RP
                                {       TOK k = ELLIPSIS;
                                        Pname l = $<pn>2;
                                        if (fct_void && l==0) k = 0;
                                        error('w',"syntax error: comma before ellipsis");
                                        $$ = fctN(0,Nunlist(l),k);
                                }
                        | LP arg_type_list RP
                                {       TOK k = 1;
                                        Pname l = $<pn>2;
                                        if (fct_void && l==0) k = 0;
                                        $$ = fctN(0,Nunlist(l),k);
                                }
                        | LP arg_type_list ELLIPSIS RP
                                {       TOK k = ELLIPSIS;
                                        Pname l = $<pn>2;
                                        if (fct_void && l==0) k = 0;
                                        $$ = fctN(0,Nunlist(l),k);
                                }
                        | LP arg_type_list CM ELLIPSIS RP
                                {       TOK k = ELLIPSIS;
                                        Pname l = $<pn>2;
                                        if (fct_void && l==0) k = 0;
                                error('w',"syntax error: comma before ellipsis");
                                $$ = fctN(0,Nunlist(1),k);
                        }
                ;

arg_type_list   :       arg_type_list CM at
                        {       if ($3)
                                        if ($1)
                                                Nadd($1,$3);
                                        else {
                                                error("AD syntax");
                                                $$ = nlistN($3);
                                        }
                                else
                                        error("AD syntax");
                        }
                |       at      %prec CM
                        {       if ($1) $$ = nlistN($1); }
                ;

at              :       arg_type
                |       %prec EMPTY
                        {       $$ = 0; }

ptr             :       MUL
                        {       $$ = new ptr(PTR,0); }
                |       AND
                        {       $$ = new ptr(RPTR,0); }
                |       MUL TYPE
                        {       TOK c = $<t>2;
                                if (c == CONST)
                                        $$ = new ptr(PTR,0,1);
                                else {
                                        $$ = new ptr(PTR,0);
                                        error("syntax error: *%k",c);
                                }
                        }
                |       AND TYPE
                        {       TOK c = $<t>2;
                                if (c == CONST)
                                        $$ = new ptr(RPTR,0,1);
                                else {
                                        $$ = new ptr(RPTR,0);
                                        error("syntax error: &%k",c);
                                }
                        }
                ;

vec             :       LB e RB
                        {       Pexpr d = $<pe>2;
                                $$ = vecN( (d!=dummy)?d:0 );
                        }
                ;

%%
```

```
/* %Z% %M% %I% %H% %T% */
/*****************************************************************************

        C++ source for cfront, the C++ compiler front-end
        written in the computer science research center of Bell Labs

        Copyright (c) 1984 AT&T Technologies, Inc. All rigths Reserved
        THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

        If you ignore this notice the ghost of Ma Bell will haunt you forever.

lex.c:
        lexical analyser based on pcc's and cpre's scanners
        modified to handle classes:
        new keywords:   class
                        public
                        call
                        etc.
        names are not entered in the symbol table by lex()
        names can be of arbitrary length
        error() is used to report errors
        {} and () must match
        numeric constants are not converted into internal representation
        but stored as strings

*****************************************************************************/

#include "cfront.h"
#include "yystype.h"
#include "size.h"

# define  CCTRANS(x) x

        /* lexical actions */

#define A_ERR   0               /* illegal character */
#define A_LET   1               /* saw a letter */
#define A_DIG   2               /* saw a digit */
#define A_1C    3               /* return a single character */
#define A_STR   4               /* string */
#define A_CC    5               /* character constant */
#define A_BCD   6               /* GCOS BCD constant */
#define A_SL    7               /* saw a / */
#define A_DOT   8               /* saw a . */
#define A_2C    9               /* possible two character symbol */
#define A_WS    10              /* whitespace (not \n) */
#define A_NL    11              /* \n */
#define A_LC    12              /* { */
#define A_RC    13              /* } */
#define A_L     14              /* ( */
#define A_R     15              /* ) */
#define A_EOF   16
#define A_ASS   17
#define A_LT    18
#define A_GT    19              /* > */
#define A_ER    20
```

```
#define A_OR    21
#define A_AND   22
#define A_MOD   23
#define A_NOT   24
#define A_MIN   25
#define A_MUL   26
#define A_PL    27
#define A_COL   28              /* : */

        /* character classes */

# define LEXLET 01
# define LEXDIG 02
/* no LEXOCT because 8 and 9 used to be octal digits */
# define LEXHEX 010
# define LEXWS 020
# define LEXDOT 040

        /* text buffer */
char inbuf[TBUFSZ];
char* txtmax = &inbuf[TBUFSZ-1];
char* txtstart;
char* txtfree;
#define pch(c) ((txtmax<=txtfree)?error('i',"input buffer overflow"):(*txtfree++=c))
#define start_txt()     txtstart = txtfree
#define del_txt()       txtfree = txtstart

char* file_name[MAXFILE];       /* stack of source file names */
                                /* file_name[0] == 0 means stdin */
class loc curloc;
FILE * out_file = stdout;
FILE * in_file = stdin;
Ptable ktbl;
int br_level = 0;               /* number of unmatched ``('''s */
int bl_level = 0;               /* number of unmatched ``{'''s */

# ifdef ibm

# define CSMASK 0377
# define CSSZ 256

# else

# define CSMASK 0177
# define CSSZ 128

# endif

short lxmask[CSSZ+1];

int saved;      /* putback character, avoid ungetchar */
int lastseen;   /* last token returned */
extern int lxtitle();

#define get(c)          (c=getc(in_file))
#define unget(c)        ungetc(c,in_file)
```

```
#define reti(a,b)       { yylval.t = b; return lastseen=a; }
#define retn(a,b)       { yylval.p = (Pnode)b; return lastseen=a; }
#define rets(a,b)       { yylval.s = b; return lastseen=a; }
#define retl(a)         { yylval.l = curloc; return lastseen=a; }

void ktbl_init()
/*
        enter keywords into keyword table for use by lex()
        and into keyword representation table used for output
*/
{
        ktbl = new table(KTBLSIZE,0,0);

        new_key("asm",ASM,0);
        new_key("auto",AUTO,TYPE);
        new_key("break",LOC,BREAK);
        new_key("case",LOC,CASE);
        new_key("continue",LOC,CONTINUE);
        new_key("char",CHAR,TYPE);
        new_key("do",LOC,DO);
        new_key("double",DOUBLE,TYPE);
        new_key("default",LOC,DEFAULT);
        new_key("enum",ENUM,0);
/*      new_key("fortran",FORTRAN);     */
        new_key("else",LOC,ELSE);
        new_key("extern",EXTERN,TYPE);
        new_key("float",FLOAT,TYPE);
        new_key("for",LOC,FOR);
        new_key("fortran",FORTRAN,0);
        new_key("goto",LOC,GOTO);
        new_key("if",LOC,IF);
        new_key("int",INT,TYPE);
        new_key("long",LONG,TYPE);
        new_key("return",LOC,RETURN);
        new_key("register",REGISTER,TYPE);
        new_key("static",STATIC,TYPE);
        new_key("struct",STRUCT,AGGR);
        new_key("sizeof",SIZEOF,0);
        new_key("short",SHORT,TYPE);
        new_key("switch",LOC,SWITCH);
        new_key("typedef",TYPEDEF,TYPE);
        new_key("unsigned",UNSIGNED,TYPE);
        new_key("union",UNION,AGGR);
        new_key("void",VOID,TYPE);
        new_key("while",LOC,WHILE);

        new_key("class",CLASS,AGGR);
        new_key("delete",LOC,DELETE);
        new_key("friend",FRIEND,TYPE);
        new_key("operator",OPERATOR,0);
        new_key("new",NEW,0);
        new_key("public",PUBLIC,0);
        new_key("const",CONST,TYPE);
        new_key("this",THIS,0);
        new_key("inline",INLINE,TYPE);
        new_key("virtual",VIRTUAL,TYPE);
        new_key("overload",OVERLOAD,TYPE);
}

extern char* src_file_name;
extern char* line_format;
loc last_line;

void loc.putline()
{
        if (file==0 && line==0) return;
        if (0<=file && file<MAXFILE) {
                char* f = file_name[file];
                if (f==0) f = (src_file_name) ? src_file_name : "";
                fprintf(out_file,line_format,line,f);
                last_line = *this;
        }
}

void loc.put(FILE* p)
{
        if (0<=file && file<MAXFILE) {
                char* f = file_name[file];
                if (f==0) f = (src_file_name) ? src_file_name : "";
                fprintf(p,"\"%s\", line %d: ",f,line);
        }
}

void lxenter( s, m ) register char *s; register short m;
/* enter a mask into lxmask */
{
        register c;

        while( c= *s++ ) lxmask[c+1] |= m;

}


void lxget(c,m) register c, m;
/*
        put 'c' back then scan for members of character class 'm'
        terminate the string read with \0
        txtfree points to the character position after that \0
*/
{
        pch(c);
        while ( (get(c), lxmask[c+1]&m) ) pch(c);
        unget(c);
        pch('\0');
}

struct LXDOPE {
        short lxch;     /* the character */
        short lxact;    /* the action to be performed */
        TOK   lxtok;    /* the token number to be returned */
} lxdope[] = {
        '$',    A_ERR,  0,      /* illegal characters go here... */
        '_',    A_LET,  0,      /* letters point here */
        '0',    A_DIG,  0,      /* digits point here */
        ' ',    A_WS,   0,      /* whitespace goes here */
        '\n',   A_NL,   0,
        '"',    A_STR,  0,      /* character string */
        '\'',   A_CC,   0,      /* ASCII character constant */
        '`',    A_BCD,  0,      /* 'foreign' character constant, e.g. BCD */
        '(',    A_L,    LP,
        ')',    A_R,    RP,
        '{',    A_LC,   LC,
        '}',    A_RC,   RC,
        '[',    A_1C,   LB,
        ']',    A_1C,   RB,
        '*',    A_MUL,  MUL,
        '?',    A_1C,   QUEST,
        ':',    A_COL,  COLON,
        '+',    A_PL,   PLUS,
        '-',    A_MIN,  MINUS,
        '/',    A_SL,   DIV,
        '%',    A_MOD,  MOD,
        '&',    A_AND,  AND,
        '|',    A_OR,   OR,
        '^',    A_ER,   ER,
        '!',    A_NOT,  NOT,
        '~',    A_1C,   COMPL,
        ',',    A_1C,   CM,
        ';',    A_1C,   SM,
        '.',    A_DOT,  DOT,
        '<',    A_LT,   LT,
        '>',    A_GT,   GT,
        '=',    A_ASS,  ASSIGN,
        EOF,    A_EOF,  EOFTOK
        };
/* note: EOF is used as sentinel, so must be <=0 and last entry in table */

struct LXDOPE *lxcp[CSSZ+1];

extern void lex_init();
void lex_init()
{
        register struct LXDOPE *p;
        register i;
        register char *cp;
        /* set up character classes */

        /* first clear lexmask */
        for(i=0; i<=CSSZ; i++) lxmask[i] = 0;

        lxenter( "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_", LEXLET );
        lxenter( "0123456789", LEXDIG );
        lxenter( "0123456789abcdefABCDEF", LEXHEX );
                /* \013 should become \v someday; \013 is OK for ASCII and EBCDIC *
        lxenter( " \t\r\b\f\013", LEXWS );
        lxmask['.'+1] |= LEXDOT;

        /* make lxcp point to appropriate lxdope entry for each character */
        /* initialize error entries */

        for( i= 0; i<=CSSZ; ++i ) lxcp[i] = lxdope;

        /* make unique entries */

        for( p=lxdope; ; ++p ) {
                lxcp[p->lxch+1] = p;
                if( p->lxch < 0 ) break;
                }

        /* handle letters, digits, and whitespace */
        /* by convention, first, second, and third places */

        cp = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
        while( *cp ) lxcp[*cp++ + 1] = &lxdope[1];
        cp = "123456789";
        while( *cp ) lxcp[*cp++ + 1] = &lxdope[2];
        cp = "\t\b\r\f\013";
        while( *cp ) lxcp[*cp++ + 1] = &lxdope[3];

        file_name[0] = src_file_name;
        curloc.file = 0;
        curloc.line = 1;

        ktbl_init();

        lex_clear();

        saved = lxtitle();
}

void lex_clear()
{
        txtstart = txtfree = inbuf;
}


char * chconst()
/*
        read a character constant into inbuf
*/
{
        register c;
        int nch = 0;

        pch('\'');

        forever {
                if (SZ_INT < nch++) {
                        error("char constant too long");
                        goto ex;
                }

                switch (get(c)) {
```

```
                case '\'':
                        goto ex;
                case EOF:
                        error("eof in char constant");
                        goto ex;
                case '\n':
                        error("newline in char constant");
                        goto ex;
                case '\\':
                        pch(c);
                        switch (get(c)){
                        case '\n':
                                ++curloc.line;
                        default:
                                pch(c);
                                break;
                        case '0': case '1': case '2': case '3': case '4':
                        case '5': case '6': case '7': case '8': case '9':
                                pch(c);
                                get(c);  /* try for 2 */
                                if( lxmask[c+1] & LEXDIG ){
                                        pch(c);
                                        get(c);  /* try for 3 */
                                        if (lxmask[c+1] & LEXDIG) pch(c);
                                        else unget(c);
                                }
                                else unget(c);
                                break;
                        };
                        break;
                default:
                        pch(c);
                }
        }
ex:
        pch('\'');
        pch('\0');
        return txtstart;
}

void lxcom()
/* process a "block comment" */
{
        register c;

        forever
        switch (get(c)) {
        case EOF:
                error("eof in comment");
                return;
        case '\n':
                curloc.line++;
                Nline++;
                break;
        case '*':
                if (get(c) == '/') return;
                unget(c);
                break;
        case '/':
                if (get(c) == '*') error('w',"``/*'' in comment");
                unget(c);
                break;
        }
}


void linecom()
/* process a "line comment" */
{
        register c;

        forever
        switch (get(c)) {
        case EOF:
                error("eof in comment");
                return;
        case '\n':
                curloc.line++;
                Nline++;
                saved = lxtitle();
                return;
        }
}

struct xyzzy {
        TOK t;
        int y;  /* fake for yystype */
};
xyzzy bck;

TOK lex()
{
        TOK ret;
        Pname n;

        if( bck.t ) {
                xyzzy tmp = bck;
                bck.t = 0;
                if( tmp.t==LC || tmp.t==RC )
                        retl( tmp.t )
                else
                        rets( tmp.t, (char *)tmp.y )
        }

        Ntoken++;

        forever {
                register lxchar;
                register struct LXDOPE *p;

                start_txt();
```

```
		if (saved) {
			lxchar = saved;
			saved = 0;
		}
		else
			get(lxchar);

		switch( (p=lxcp[lxchar+1])->lxact ){
		case A_1C:
			/* eat up a single character, and return an opcode */

			reti(p->lxtok,p->lxtok);

		case A_EOF:
			if (br_level || bl_level)
				error("'%s' missing at end of input",(bl_level) ? "
			reti(EOFTOK,0);

		case A_ERR:
			error("illegal character (0%o)",lxchar);
			break;

		case A_LET:
			/* collect an identifier, check for reserved word, and retu
			lxget( lxchar, LEXLET|LEXDIG );

			if (n = ktbl->look(txtstart,0)) {
				TOK x;
				del_txt();
				switch (x=n->base) {
				case TNAME:
					retn(TNAME,n);
					break;
				case LOC:
					retl(n->syn_class);
				default:
					reti(n->syn_class,x);
				}
			}
			else {
				rets(ID,txtstart);
			}

		case A_DIG:

			ret = ICON;

			if (lxchar=='0') {	/* octal or hexadecimal number */
				pch('0');
				switch (get(lxchar)) {
				case 'l':
				case 'L':
					pch('L');
					pch(0);
					rets(ICON,txtstart);
				case 'x':
```

```
                case 'X':
                        lxget('X',LEXHEX);
                        switch (get(lxchar)) {
                        case 'l':
                        case 'L':
                                txtfree--;
                                pch('L');
                                pch(0);
                                break;
                        default:
                                saved = lxchar;
                        }
                        rets(ICON,txtstart);
                case '8':
                case '9':
                        error("8 or 9 used as octal digit");
                case '0':
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                case '7':
                        pch(lxchar);
                ox:
                        switch (get(lxchar)) {
                        case '8':
                        case '9':
                                error("8 or 9 used as octal digit")
                        case '0':
                        case '1':
                        case '2':
                        case '3':
                        case '4':
                        case '5':
                        case '6':
                        case '7':
                                pch(lxchar);
                                goto ox;
                        case 'l':
                        case 'L':
                                pch('L');
                                pch(0);
                                break;
                        default:
                                pch(0);
                                saved = lxchar;
                        }
                        rets(ICON,txtstart);
                case '.':
                        lxget('.',LEXDIG);
                        goto getfp;
                default:
                        saved = lxchar;
                        reti(ZERO,0);
```

```
                                }
                        }
                        else
                                lxget(lxchar,LEXDIG);

                        if (get(lxchar) == '.') {
                                txtfree--;
                                lxget('.', LEXDIG );
                getfp:
                                ret = FCON;
                                get(lxchar);
                        };

                        switch (lxchar) {
                        case 'e':
                        case 'E':
                                txtfree--;
                                switch (get(lxchar)) {
                                case '-':
                                case '+':
                                        pch('e');
                                        break;
                                default:
                                        unget(lxchar);
                                        lxchar = 'e';
                                };
                                lxget( lxchar, LEXDIG );
                                ret = FCON;
                                break;
                        case 'l':
                        case 'L':
                                txtfree--;
                                pch('L');
                                break;
                        default:
                                saved = lxchar;
                        };

                        pch(0);
                        rets(ret,txtstart);

                case A_DOT:
                        if (get(lxchar) == '.') {       /* look for ellipsis */
                                if (get(lxchar) != '.') {
                                        error("token .. ?");
                                        saved = lxchar;
                                }
                                reti(ELLIPSIS,0);
                        }
                        if( lxmask[lxchar+1] & LEXDIG ){/* look for floating consta
                                unget(lxchar);
                                lxget( '.', LEXDIG );
                                goto getfp;
                        }
                        saved = lxchar;
                        reti(DOT,0);
```

```
		case A_STR:
			/* save string constant in buffer */
			forever
			switch (get(lxchar)) {
			case '\\':
				pch('\\');
				get(lxchar);
				pch(lxchar);
				break;
			case '"':
				pch(0);
				rets(STRING,txtstart);
			case '\n':
				error("newline in string");
				pch(0);
				rets(STRING,txtstart);
			case EOF:
				error("eof in string");
				pch(0);
				rets(STRING,txtstart);
			default:
				pch(lxchar);
			}

		case A_CC:
			/* character constant */
			rets(CCON,chconst());

		case A_BCD:
			{
				register i;
				int j;

				pch('`');

				for (i=0; i<7; ++i) {
					pch(get(j));
					if (j == '`' ) break;
				}
				pch(0);
				if (6<i)
					error("bcd constant exceeds 6 characters" )
				rets(CCON,txtstart);
			}

		case A_SL:	/* / */
			switch (get(lxchar))  {
			case '*':
				lxcom();
				break;
			case '/':
				linecom();
				break;
			case '=':
				reti(ASOP,ASDIV);
```

```
                default:
                        saved = lxchar;
                        reti(DIVOP,DIV);
                }

        case A_WS:
                continue;

        case A_NL:
                ++curloc.line;
                Nline++;
                saved = lxtitle();
                continue;

        case A_LC:
                if (BLMAX <= bl_level++) {
                        error('s',"blocks too deaply nested");
                        ext(3);
                }
                retl(LC);

        case A_RC:
                if (bl_level-- <= 0) {
                        error("unX '}'");
                        bl_level = 0;
                }
                retl(RC);

        case A_L:
        /*
                return CAST if the LP is the start of a cast LP otherwise
                only
                        ( type-name (
                is a real problem
        */
                br_level++;
                switch (lastseen) {     /* f( => all bets are off */
                case NAME:
                case TNAME:
                case TYPE:
                        reti(LP,0);
                }

                if( saved )
                        lxchar = saved;
                else
                        get( lxchar );
                while( ( p = lxcp[ lxchar+1 ] )->lxact == A_WS )
                        get( lxchar );
                saved = lxchar;
                if( p->lxact != A_LET ) reti( LP,  0 );

                bck.t = lex();
                bck.y = int( yylval.s );
                switch (bck.t) {
                case TYPE:
```

```
                case TNAME:
                        break;
                case AGGR:
                case ENUM:
                        reti(CAST,0);
                default:
                        reti(LP,0);
                }

                if( saved )
                        lxchar = saved;
                else
                        get( lxchar );
                while( ( p = lxcp[ lxchar+1 ] )->lxact == A_WS )
                        get( lxchar );
                saved = lxchar;
                switch (lxchar) {
                case ':':       reti(LP,0);     /* (classname::memname */
                case '(':       break;
                default:        reti(CAST,0);
                }

                /*      here is the real problem:
                        CAST:   ( int ( * ) ( ) ) p;
                        LP:     ( int ( * p ) )

                        ignore
                                ( int ( &
                        and     ( int ( [
                        and     ( int ( (       problems
                */
                get( lxchar );
                while( ( p = lxcp[ lxchar+1 ] )->lxact == A_WS ) get( lxcha
                if (lxchar != '*') {
                        unget(lxchar);
                        reti(LP,0);
                }

                get( lxchar );
                while( ( p = lxcp[ lxchar+1 ] )->lxact == A_WS ) get( lxcha
                unget(lxchar);
                unget('*');
                if (lxchar == ')') reti(CAST,0);

                reti(LP,0);

        case A_R:
                if (br_level-- <= 0) {
                        error("unX ')'");
                        br_level = 0;
                }
                reti(RP,0);
        case A_ASS:
                switch (get(lxchar)) {
                case '=':
                        reti(EQUOP,EQ);
```

```
			default:
				saved = lxchar;
				reti(ASSIGN,ASSIGN);
			}
		case A_COL:
			switch (get(lxchar)) {
			case ':':
				reti(MEM,0);
			case '=':
				error("':=' is not a c++ operator");
				reti(ASSIGN,ASSIGN);
			default:
				saved = lxchar;
				reti(COLON,COLON);
			}
		case A_NOT:
			switch (get(lxchar)) {
			case '=':
				reti(EQUOP,NE);
			default:
				saved = lxchar;
				reti(NOT,NOT);
			}
		case A_GT:
			switch(get(lxchar)) {
			case '>':
				switch (get(lxchar)) {
				case '=':
					reti(ASOP,ASRS);
					break;
				default:
					saved = lxchar;
					reti(SHIFTOP,RS);
				}
			case '=':
				reti(RELOP,GE);
			default:
				saved = lxchar;
				reti(RELOP,GT);
			}
		case A_LT:
			switch (get(lxchar)) {
			case '<':
				switch (get(lxchar)) {
				case '=':
					reti(ASOP,ASLS);
				default:
					saved = lxchar;
					reti(SHIFTOP,LS);
				}
			case '=':
				reti(RELOP,LE);
			default:
				saved = lxchar;
				reti(RELOP,LT);
			}
```

```
                case A_AND:
                        switch (get(lxchar)) {
                        case '&':
                                reti(ANDAND,ANDAND);
                        case '=':
                                reti(ASOP,ASAND);
                        default:
                                saved = lxchar;
                                reti(AND,AND);
                        }
                case A_OR:
                        switch (get(lxchar)) {
                        case '|':
                                reti(OROR,OROR);
                        case '=':
                                reti(ASOP,ASOR);
                        default:
                                saved = lxchar;
                                reti(OR,OR);
                        }
                case A_ER:
                        switch (get(lxchar)) {
                        case '=':
                                reti(ASOP,ASER);
                        default:
                                saved = lxchar;
                                reti(ER,ER);
                        }
                case A_PL:
                        switch (get(lxchar)) {
                        case '=':
                                reti(ASOP,ASPLUS);
                        case '+':
                                reti(ICOP,INCR);
                        default:
                                saved = lxchar;
                                reti(PLUS,PLUS);
                        }
                case A_MIN:
                        switch (get(lxchar)) {
                        case '=':
                                reti(ASOP,ASMINUS);
                        case '-':
                                reti(ICOP,DECR);
                        case '>':
                                reti(REF,REF);
                        default:
                                saved = lxchar;
                                reti(MINUS,MINUS);
                        }
                case A_MUL:
                        switch (get(lxchar)) {
                        case '=':
                                reti(ASOP,ASMUL);
                        case '/':
                                error('w',"*/ not as end of comment");
                        default:
                                saved = lxchar;
                                reti(MUL,MUL);
                        }
                case A_MOD:
                        switch (get(lxchar)) {
                        case '=':
                                reti(ASOP,ASMOD);
                        default:
                                saved = lxchar;
                                reti(DIVOP,MOD);
                        }
                default:
                        error('i',"lex act==%d getc()->%d",p,lxchar);

                }

                error('i',"lex, main switch");

        }

}

int lxtitle()
/*
        called after a newline; set linenumber and file name
*/
{
        register c;

        forever
        switch ( get(c) ) {
        default:
                return c;
/*      case EOF:
                return EOF;     */
        case '\n':
                curloc.line++;
                Nline++;
                break;
        11:
                break;
        case '#':       /* # lineno "filename" */
                curloc.line = 0;
                forever
                switch (get(c)) {
                case '"':
                        start_txt();
                        forever
                        switch (get(c)) {
                        case '"':
                                pch('\0');
                                if (get(c) != '\n') error("unX eol on # line");
                                if (*txtstart) {
                                        /*      maintain stack of file names */
                                        int f = curloc.file;
                                        char* fn;
                                        if (f == 0) goto push;
                                        if ( (fn=file_name[f]) && (strcmp(txtstart,
                                                /* same file: ignore */
                                        }
                                        else if ( (fn=file_name[f-1]) && (strcmp(tx
                                                /* previous file: pop */
                                        /*      delete(file_name[f]);*/
                                                curloc.file--;
                                        }
                                        else {
                                                /* new file name: push */
                                                char * p;
                                push:
                                                Nfile++;
                                                p = new char[txtfree-txtstart+1];

                                                if (MAXFILE<=++f) error('i',"fileN
                                                file_name[curloc.file=f] = p;
                                                (void) strcpy(p,txtstart);
                                                Nstr++;
                                        }
                                }
                                else {
                                        /* back to the original .c file: "" */
/*
                                        int f = curloc.file;
                                        if (1<f) error('i',"fileN buffer (%d)",f);
                                        if (f) delete file_name[f];
*/
                                        curloc.file = 0;
                                }
                                del_txt();
/*
                                curloc.putline();
*/
                                goto l1;
                        case '\n':
                                error("unX end of line on '# line'");
                        default:
                                pch(c);
                        }
                case ' ':
                        break;
                case '0':
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                case '7':
                case '8':
                case '9':
                        curloc.line = curloc.line*10+c-'0';
                        break;
                case 'c':       /* ignore #class */
                        if (get(c) == 'l')
                                while (get(c) != '\n') ;
                        curloc.line++;
                        Nline++;
                        goto l1;
                case '\n':
                        curloc.putline();
                        goto l1;
                default:
                        error("unX character on '# line'");
                }
        }
}
```

```
/* %Z% %M% %I% %H% %T% */
/*****************************************************************************

        C++ source for cfront, the C++ compiler front-end
        written in the computer science research center of Bell Labs

        Copyright (c) 1984 AT&T Technologies, Inc. All rigths Reserved
        THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

        If you ignore this notice the ghost of Ma Bell will haunt you forever.

main.c:

        Initialize global environment
        Read argument line
        Start compilation
        Clean up end exit

*****************************************************************************/

/*#include <signal.h>
*/

#include <time.h>
char* ctime(long*);
long time(long*);
long start_time, stop_time;

#include "cfront.h"

char* prog_name = "<<cfront (release E) 1/30/85>>";

extern char* src_file_name;
char* src_file_name = 0;

bit Styp = 1;
bit Ssimpl = 1;

bit old_fct_accepted = 1;
TOK scope_default = STATIC;
bit fct_void;
bit st_init;

char* line_format = "\n# %d \"%s\"\n";

Plist isf_list;
Pstmt st_ilist;
Pstmt st_dlist;

int Nspy;
int Nfile = 1 , Nline, Ntoken;
int Nfree_store, Nalloc, Nfree;
int Nname;
int Nn, Nbt, Nt, Ne, Ns, Nc, Nstr, Nl;
extern int NFn, NFtn, NFbt, NFpv, NFf, NFe, NFs, NFc, NFl;
```

```
simpl_init();
typ_init();
syn_init();
lex_init();
error_init();
print_free();
read_align(char*);
print_align(char*);

void spy(char* s)
{
        if (s) fprintf(stderr,"%s:\n",s);
        fprintf(stderr,"files=%d lines=%d tokens=%d\n",Nfile, Nline, Ntoken);
        fprintf(stderr,"Names: distinct=%d global=%d type=%d\n",
                Nname, gtbl->max(), ktbl->max());
        fflush(stderr);
        if (start_time && stop_time) {
                fprintf(stderr,"start time: %s", ctime(&start_time) );
                fprintf(stderr,"stop time:  %s", ctime(&stop_time) );
                fprintf(stderr,"real time delay %ld: %d lines per second\n",
                        stop_time-start_time, Nline/(stop_time-start_time) );
                fflush(stderr);
        }
        fprintf(stderr,"free store=%dbytes alloc()=%d free()=%d ",
                Nfree_store, Nalloc, Nfree);
        print_free();
        fflush(stderr);
        fprintf(stderr,"sizeof: n=%d bt=%d f=%d p=%d v=%d e=%d c=%d l=%d\n",
                sizeof(name), sizeof(basetype), sizeof(fct),
                sizeof(ptr), sizeof(vec),
                sizeof(expr), sizeof(typed_obj), /*sizeof(elist)*/16 );
        fprintf(stderr,"alloc(): n=%d bt=%d t=%d e=%d s=%d c=%d str=%d l=%d\n",
                Nn, Nbt, Nt, Ne, Ns, Nc, Nstr,Nl);
        fprintf(stderr,"free(): n=%d bt=%d t=%d e=%d s=%d c=%d str=? l=%d\n",
                NFn, NFbt, NFpv+NFf, NFe, NFs, NFc, NFl);
        fflush(stderr);
        fprintf(stderr,"%d errors\n",error_count);
        fflush(stderr);
}

Pname dcl_list; /* declarations generated while declaring something else */

char *st_name( int ); /* generates names of static ctor, dtor callers */

void run()
/*
        run the appropriate stages
*/
{
        Pname n;
        int i = 1;

        while (n=syn()) {
                Pname nn;
                Pname nx;
```

```
		if (n == (Pname)1) continue;

		if (Styp == 0) {
			n->dcl_print(SM);
			lex_clear();
			continue;
		}

		for (nn=n; nn; nn=nx) {
			nx = nn->n_list;
			nn->n_list = 0;
			if (nn->dcl(gtbl,EXTERN) == 0) continue;

			if (error_count) continue;

			if (Ssimpl) nn->simpl();

			/* handle generated declarations */
			for (Pname dx, d=dcl_list; d; d=dx) {
				dx = d->n_list;
				d->dcl_print(0);
				delete d;
			}
			dcl_list = 0;

			if (nn->base) nn->dcl_print(0);

			switch (nn->tp->base) { /* clean up */
			default:
			{	Pexpr i = nn->n_initializer;
				if (i && i!=(Pexpr)1) DEL(i);
			}

			case FCT:
			{	Pfct f = (Pfct)nn->tp;
				if (f->body && (debug || f->f_inline==0)) {
					DEL(f->body);
				/*	f->body = 0;  leave to detect re-definition
				}
				break;
			}

			case CLASS:
			{	Pclass cl = (Pclass)nn->tp;
				register Pname p;
				for (p=cl->pubmem; p; p=p->n_list) {
					switch (p->tp->base) {
					case FCT:
					{	Pfct f = (Pfct)p->tp;
						if (f->body && (debug || f->f_inlin
							DEL(f->body);
							f->body = 0;
						}
					}
					case CLASS:
					case ENUM:
```

```
                                                break;
                                        case COBJ:
                                        case EOBJ:
                                                DEL(p);
                                                break;
                                        default:
                                                delete p;
                                        }
                                }
                                cl->pubmem = 0;


                                for (p=cl->privmem; p; p=p->n_list) {
                                        switch (p->tp->base) {
                                        case FCT:
                                        {       Pfct f = (Pfct)p->tp;
                                                if (f->body && (debug || f->f_inlin
                                                        DEL(f->body);
                                                        f->body = 0;
                                                }
                                        }
                                        case CLASS:
                                        case ENUM:
                                                break;
                                        case COBJ:
                                        case EOBJ:
                                                DEL(p);
                                                break;
                                        default:
                                                delete p;
                                        }
                                }
                                cl->privmem = 0;
                                cl->permanent = 3;
                                break;
                        }
                        }

                        DEL(nn);
                }

                lex_clear();
        }

        switch (no_of_undcl) {
        case 0:
                break;
        case 1:
                error('w',"undeclaredF%n called",undcl1);
                break;
        case 2:
                error('w',"%d undeclaredFs called:%n and%n",no_of_undcl,undcl1,undc
                break;
        default:
                error('w',"%d undeclaredFs called,%n,%n etc",no_of_undcl,undcl1,und
        }
```

```
	Pname m;
	if (fct_void == 0)
	for (m=gtbl->get_mem(i=1); m; m=gtbl->get_mem(++i)) {
/*error('d',"global:%k n_key%k perm %d %n", m->base, m->n_key, m->permanent, m );*/
		if (m->base==TNAME
		|| m->n_scope==EXTERN
		|| m->n_stclass == ENUM) continue;

		Ptype t = m->tp;
		if (t == 0) continue;
	ll:
		switch (t->base) {
		case TYPE:	t=((Pbase)t)->b_name->tp; goto ll;
		case CLASS:
		case ENUM:
		case COBJ:
		case OVERLOAD:
		case VEC:	continue;
		case FCT:	if ( ((Pfct)t)->f_inline ) continue;

		}

		if (m->n_addr_taken==0 && m->n_used==0) {
			Cdcl = m;
			if (m->tp->tconst() ==0)
				error('w',"static%n declared but not used",m);
		}
	}

	if (st_ilist) { /*	make an "init" function;
				it calls all constructors for static objects
			*/
		Pname n = new name( st_name('I') );
		Pfct f = new fct(void_type,0,1);
		n->tp = f;
		f->body = new block(st_ilist->where,0,0);
		n->n_sto = EXTERN;
		(void) n->dcl(gtbl,EXTERN);
		n->simpl();
		f->body->s = st_ilist;
		f->simpl();
		n->dcl_print(0);
	}

	if (st_dlist) { /*	make a "done" function;
				it calls all destructors for static objects
			*/
		Pname n = new name( st_name('D') );
		Pfct f = new fct(void_type,0,1);
		n->tp = f;
		f->body = new block(st_dlist->where,0,0);
		n->n_sto = EXTERN;
		(void) n->dcl(gtbl,EXTERN);
		n->simpl();
		f->body->s = st_dlist;
```

```
                f->simpl();
                n->dcl_print(0);
        }

        if (debug==0) { /* print inline function definitions */
                Plist l;
                for (l=isf_list; l; l=l->l) {
                        Pname n = l->f;
                        Pfct f = (Pfct)n->tp;

                        switch (f->base) {
                        case FCT: break;
                        default: error('i',"inline list corrupted\n");
                        case OVERLOAD:
                                n = ((Pgen)f)->fct_list->f;     /* first fct */
                                f = (Pfct)n->tp;
                        }

/*fprintf(stderr,"%s() tp (%d %d) %d %d\n", n->string, n->tp, n->tp?n->tp->base:0, n
                        if (n->n_addr_taken || f->f_virtual) {
/*                              if (st_init) putst("asm(\"library\");");*/
                                n->tp->dcl_print(n);
                        }
                }
        }

        fprintf(out_file,"\n/* the end */\n");

}

bit warn = 1;   /* printout warning messages */
bit debug = 0;  /* code generation for debugger */
char* afile = "default";

int no_of_undcl;
Pname undcl1, undcl2;

main(int argc, char* argv[])
/*
        read options, initialize, and run
*/
{
        extern char* mktemp();
        register char * cp;
        short i;

        /*(void) signal(SIGINT,&sig_exit);
        (void) signal(SIGTERM,sig_exit);
        (void) signal(SIGQUIT,sig_exit);
*/


        error_init();

        for (i=1; i<argc; ++i) {
                switch (*(cp=argv[i])) {
```

```
                case '+':
                        while (*++cp) {
                                switch(*cp) {
                                case 't':
                                        fprintf(stderr,"type check only\n");
                                        Ssimpl = 0;
                                        break;
                                case 's':
                                        fprintf(stderr,"syntax check only\n");
                                        Styp = Ssimpl = 0;
                                        break;
                                case 'w':
                                        warn = 0;
                                        break;
                                case 'd':
                                        debug = 1;
                                        break;
                                case 'f':
                                        src_file_name = cp+1;
                                        goto xx;
                                case 'x':       /* read in table for cross compilat
                                        if (read_align(afile = cp+1)) {
                                                fprintf(stderr,"bad size-table (opt
                                                exit(11);
                                        }
                                        goto xx;
                                case 'C':       /* preserve comments */
                                        error('s',"cannot preserve comments");
                                        break;
                                case 'V':       /* C with classes compatability */
                                        fct_void = 1;
                                        /* no break */
                                case 'E':
                                        scope_default = EXTERN;
                                        break;
                                case 'S':
                                        Nspy++;
                                        break;
                                case 'L':
                                        line_format = "\n#line %d \"%s\"\n";
                                        break;
                                case 'I':
                                        st_init = 1;
                                        break;
                                default:
                                        fprintf(stderr,"%s: unexpected option: -%c

                                        break;
                                }
                        }
                xx:
                        break;
                default:
                        fprintf(stderr,"%s: bad argument \"%s\"\n",prog_name,cp);
                        exit(11);
                }
```

```
        }


        fprintf(out_file,"\n/* %s */\n",prog_name);
        if (src_file_name) fprintf(out_file,"/* < %s */\n",src_file_name);

        if (Nspy) {
                start_time = time(0);
                print_align(afile);
        }
        fflush(stderr);
        if (Ssimpl) print_mode = SIMPL;
        otbl_init();
        lex_init();
        syn_init();
        typ_init();
        simpl_init();
        scan_started = 1;
        curloc.putline();
        run();
        if (Nspy) {
                stop_time = time(0);
                spy(src_file_name);
        }

        return (0<=error_count && error_count<127) ? error_count : 127;
}

extern int strcat(char*, char*);
char * st_name( int iord ) {
        static char *name = 0;
        static char *prefix = "_ST_"; /* first character must be valid in a
                                         C identifier */
        if ( iord != 'I' && iord != 'D' )
                error( 'i', "bad ST_ type %d\n", iord );
        if ( !name ) {
                int stilen = strlen( prefix ) + 1 +
                        ( src_file_name ) ? strlen( src_file_name ) : 0;
                name = new char[ stilen ];
                strcpy( name, prefix );
                if ( src_file_name ) strcat( name, src_file_name );
                char *p = name;
                while ( *++p ) {
                        if ( 'a' <= *p && *p <= 'z' ||
                        'A' <= *p && *p <= 'Z' ||
                        '0' <= *p && *p <= '9' ) continue;
                        *p = '_';
                }
        }
        name[ strlen( prefix ) - 1 ] = iord;
        return name;
}
```

```
/* %Z% %M% %I% %H% %T% */
/* empty */
/* @(#) norm.c 1.1 1/2/85 17:58:42 */
/*************************************************************************

        C++ source for cfront, the C++ compiler front-end
        written in the computer science research center of Bell Labs

        Copyright (c) 1984 AT&T Technologies, Inc. All rigths Reserved
        THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

        If you ignore this notice the ghost of Ma Bell will haunt you forever.

norm.c:

        "normalization" handles problems which could have been handled
        by the syntax analyser; but has not been done. The idea is
        to simplify the grammar and the actions accociated with it,
        and to get a more robust error handling

**************************************************************************/

#include "cfront.h"
#include "size.h"

extern void syn_init();
void syn_init()
{
        any_type = new basetype(ANY,0);
        PERM(any_type);
        dummy = new expr(DUMMY,0,0);
        PERM(dummy);
        dummy->tp = any_type;
        zero = new expr(ZERO,0,0);
        PERM(zero);
}


char* make_name(TOK c)
{
        static stcount;

        char* s = new char[8];  /* as it happens: fits in two words */


        if (99999 <= ++stcount) error('i',"too many generated names");

        s[0] = '_';
        s[1] = c;
        int count = stcount;
        int i = 2;
        if (10000 <= count) {
                s[i++] = '0' + count/10000;
                count %= 10000;
        }
        if (1000 <= count) {
                s[i++] = '0' + count/1000;
                count %= 1000;
        }
        else if (2<i) s[i++] = '0';

        if (100 <= count) {
                s[i++] = '0' + count/100;
                count %= 100;
        }
        else if (2<i) s[i++] = '0';

        if (10 <= count) {
                s[i++] = '0' + count/10;
                count %= 10;
        }
        else if (2<i) s[i++] = '0';

        s[i++] = '0' + count;
        s[i] = 0;

        return s;
}

Pbase basetype.type_adj(TOK t)
{
        switch (base) {
        case COBJ:
        case EOBJ:
        {       Pbase bt = new basetype(0,0);
                *bt = *this;
                DEL(this);
                this = bt;
        }
        }

        if (b_xname) {
                if (base)
                        error("badBT:%n%k",b_xname,t);
                else {
                        base = TYPE;
                        b_name = b_xname;
                }
                b_xname = 0;
        }

        switch (t) {
        case TYPEDEF:   b_typedef = 1;  break;
        case INLINE:    b_inline = 1;   break;
        case VIRTUAL:   b_virtual = 1;  break;
        case CONST:     b_const = 1;    break;
        case UNSIGNED:  b_unsigned = 1; break;
        case SHORT:     b_short = 1;    break;
        case LONG:      b_long = 1;     break;
        case FRIEND:
        case OVERLOAD:
        case EXTERN:
        case STATIC:
        case AUTO:
```

```
	case REGISTER:
		if (b_sto)
			error("badBT:%k%k",b_sto,t);
		else
			b_sto = t;
		break;
	case VOID:
	case CHAR:
	case INT:
	case FLOAT:
	case DOUBLE:
		if (base)
			error("badBT:%k%k",base,t);
		else
			base = t;
		break;
	default:
		error('i',"basetype.type_adj(%k)",t);
	}
	return this;
}

Pbase basetype.name_adj(Pname n)
{
	if (b_xname) {
		if (base)
			error("badBT:%n%n",b_xname,n);
		else {
			base = TYPE;
			b_name = b_xname;
		}
		b_xname = 0;
	}
	b_xname = n;
	return this;
}

Pbase basetype.base_adj(Pbase b)
{
	Pname bn = b->b_name;

	switch (base) {
	case COBJ:
	case EOBJ:
		error("NX after%k%n",base,b_name);
		return this;
	}

	if (base) {
		if (b_name)
			error("badBT:%k%n%k%n",base,b_name,b->base,bn);
		else
			error("badBT:%k%k%n",base,b->base,bn);
	}
	else {
		base = b->base;
```

```
                b_name = bn;
                b_table = b->b_table;
        }
        return this;
}

Pbase basetype.check(Pname n)
/*
        "n" is the first name to be declared using "this"
        check the consistency of "this"
        and use "b_xname" for "n->string" if possible and needed
*/
{
        b_inline = 0;
        b_virtual = 0;
/*fprintf(stderr,"check n: %d %s b: %d %d %s\n",n,(n)?n->string:"",this,base,(b_name
        if (b_xname && (n->tp || n->string)) {
                if (base)
                        error("badBT:%k%n",base,b_xname);
                else {
                        base = TYPE;
                        b_name = b_xname;
                }
                b_xname = 0;
        }

        if (b_xname) {
                if (n->string)
                        error("twoNs inD:%n%n",b_xname,n);
                else {
                        n->string = b_xname->string;
                        b_xname->hide();
                }
                b_xname = 0;
        }

        switch (base) {
        case 0:
                base = INT;
                break;
        case EOBJ:
        case COBJ:
                if (b_name->base == TNAME)
                        error('i',"TN%n inCO %d",b_name,this);
        }

        if (b_long || b_short) {
                TOK sl = (b_short) ? SHORT : LONG;
                if (b_long && b_short) error("badBT:long short%k%n",base,n);
                if (base != INT)
                        error("badBT:%k%k%n",sl,base,n);
                else
                        base = sl;
                b_short = b_long = 0;
        }
        if (b_typedef && b_sto) error("badBT:typedef%k%n",b_sto,n);
        b_typedef = b_sto = 0;

        if (Pfctvec_type == 0) return this;

        if (b_const) {
                return this;
/*
                switch (base) {
                case INT:
                        ;
                }
*/
        }
        else if (b_unsigned) {
                switch (base) {
                case LONG:
                /*      error('s',"unsigned long");*/
                        delete this;
                        return ulong_type;
                case SHORT:
                        delete this;
                        return ushort_type;
                case INT:
                        delete this;
                        return uint_type;
                case CHAR:
                        delete this;
                        return uchar_type;
                default:
                        error("badBT: unsigned%k%n",base,n);
                        b_unsigned = 0;
                        return this;
                }
        }
        else {
                switch (base) {
                case LONG:
                        delete this;
                        return long_type;
                case SHORT:
                        delete this;
                        return short_type;
                case INT:
                        if (this != int_type) delete this;
                        return int_type;
                case CHAR:
                        delete this;
                        return char_type;
                case VOID:
                        delete this;
                        return void_type;
                case TYPE:
                        /* use a single base saved in the keyword */
/*fprintf(stderr,"type %d bn %d %s q %d\n",this,b_name,b_name->string,b_name->n_qual
                        if (b_name->n_qualifier) {
```

```
                                        delete this;
                                        return (Pbase)b_name->n_qualifier;
                                }
                                else {
                                        PERM(this);
                                        b_name->n_qualifier = (Pname)this;
                                        return this;
                                }
                default:
                        return this;
                }
        }
}

Pname basetype.aggr()
/*
        "type SM" seen e.g.     struct s {};
                                class x;
                                enum e;
                                int tname;
                                friend cname;
                                friend class x;
                                int;

        convert
                union { ... };
        into
                union name { ... } name ;
*/
{
        Pname n;

        if (b_xname) {
                if (base) {
                        Pname n = new name(b_xname->string);
                        b_xname->hide();
                        b_xname = 0;
                        return n->normalize(this,0,0);
                }
                else {
                        base = TYPE;
                        b_name = b_xname;
                        b_xname = 0;
                }
        }


        switch (base) {
        case COBJ:
        {       Pclass cl = (Pclass)b_name->tp;
                char* s = cl->string;
/*fprintf(stderr,"COBJ (%d %s) -> (%d %d) ->(%d %d)\n",this,b_name->string,b_name,b_
                if (b_name->base == TNAME) error('i',"TN%n inCO",b_name);
                if (b_const) error("const%k%n",cl->csu,b_name);

                if (cl->c_body == 2) {  /* body seen */
```

```
				if (s[0]=='_' && s[1]=='C') {
					char* ss = new char[5];
					Pname obj = new name(ss);
					if (cl->csu == UNION) {
						strcpy(ss,s);
						ss[1] = 'O';
						cl->csu = ANON;
						return obj->normalize(this,0,0);
					}
					error('w',"un-usable%k ignored",cl->csu);
				}
				cl->c_body = 1;
				return b_name;
			}
			else {	/* really a typedef for cfront only: class x; */
				if (b_sto == FRIEND) goto frr;
				return 0;
			}
		}

		case EOBJ:
		{	Penum en = (Penum)b_name->tp;
/*fprintf(stderr,"EOBJ (%d %s) -> (%d %d) ->(%d %d)\n",this,b_name->string,b_name,b_
			if (b_name->base == TNAME) error('i',"TN%n in enumO",b_name);
			if (b_const) error("const enum%n",b_name);
			if (en->e_body == 2) {
				en->e_body = 1;
				return b_name;
			}
			else {
				if (b_sto == FRIEND) goto frr;
				return 0;
			}
		}

		default:
			if (b_typedef) error('w',"illegal typedef ignored");

			if (b_sto == FRIEND) {
			frr:
				Pname fr = ktbl->look(b_name->string,0);
				if (fr == 0) error('i',"cannot find friend%n",b_name);
				n = new name(b_name->string);
				n->n_sto = FRIEND;
				n->tp = fr->tp;
				return n;
			}
			else {
				n = new name(make_name('D'));
				n->tp = any_type;
				error('w',"NX inDL");
				return n;
			}
		}
}
```

```
void name.hide()
/*
        hide "this": that is, "this" should not be a keyword in this scope
*/
{
        if (base != TNAME) return;
        if (n_key == 0) {
/*error('d',"hide%n",this);*/
                if (lex_level == bl_level) error('w',"%n redefined",this);
                modified_tn = new name_list(this,modified_tn);
                n_key = HIDDEN;
        }
}
void set_scope(Pname tn)
/*
        enter the scope of class tn after seeing "tn.f"
*/
{
        Pbase b;
        Pclass cl;
        Plist l;
        if (tn->base != TNAME) error('i',"set_scope: not aTN %d %d",tn,tn->base);
        b = (Pbase)tn->tp;
        if (b->b_name->tp->base != CLASS) error('i',"T of%n not aC (%k)",tn,b->b_na
        cl = (Pclass)b->b_name->tp;
        for (l=cl->tn_list; l; l=l->l) {
                Pname n = l->f;
                n->n_key = (n->lex_level) ? 0 : HIDDEN;
                modified_tn = new name_list(n,modified_tn);
        }
}

void restore()
/*
*/
{
        Plist l;

        for (l=modified_tn; l; l=l->l) {
                Pname n = l->f;
                if (n->lex_level <= bl_level) {
                        n->n_key = 0;
                }
                else {
                        n->n_key = HIDDEN;
                }
        }
}

Pbase start_cl(TOK t, Pname c, Pname b)
{
        if (c == 0) c = new name(make_name('C'));
        Pname n = c->tname(t);                          /* t ignored */
        n->where = curloc;
        Pbase bt = (Pbase)n->tp;                        /* COBJ */
        if (bt->base != COBJ) {
                error("twoDs of%n:%t andC",n,bt);
                exit(88);
        }
        Pclass occl = ccl;
        ccl = (Pclass)bt->b_name->tp;           /* CLASS */
        ccl->in_class = occl;
        ccl->tn_list = modified_tn;
        modified_tn = 0;
        ccl->string = n->string;
        ccl->csu = t;                           /* ! */
        if (b) ccl->clbase = b->tname(t);
        return bt;
}

void end_cl()
{
        Pclass occl = ccl->in_class;
        Plist ol = ccl->tn_list;
        ccl->c_body = 2;
        ccl->tn_list = modified_tn;
        if (modified_tn) restore();
        modified_tn = ol;
        ccl = occl;
}

Pbase end_enum(Pname n, Pname b)
{
        if (n == 0) n = new name(make_name('E'));
        n = n->tname(ENUM);
        Pbase bt = (Pbase)n->tp;
        if (bt->base != EOBJ) {
                error("twoDs of%n:%t and enum",n,bt);
                exit(88);
        }
        Penum en = (Penum)bt->b_name->tp;
        en->e_body = 2;
        en->mem = name_unlist((class nlist *)b);
        if (en->defined) error("enum%n defined twice",n);
        return bt;
}

Pname name.tdef()
/*
        typedef "this"
*/
{
        Pname n = ktbl->insert(this,0);
        if (tp == 0) error('i',"typedef%n tp==0",this);
        n->base = base = TNAME;
        PERM(n);
        PERM(tp);
        modified_tn = new name_list(n,modified_tn);
        return n;
}

Pname name.tname(TOK csu)
```

```
/*
        "csu" "this" seen, return typedef'd name for "this"
        return  (TNAME,x)
        x:      (COBJ,y)
        y:      (NAME,z)
        z:      (CLASS,ae);
*/
{
        switch (base) {
        case TNAME:
                return this;
        case NAME:
        {       Pname tn = ktbl->insert(this,0);
                Pname on = new name;
                tn->base = TNAME;
                modified_tn = new name_list(tn,modified_tn);
                tn->n_list = n_list = 0;
                string = tn->string;
                *on = *this;
                switch (csu) {
                case ENUM:
                        tn->tp = new basetype(EOBJ,on);
                        on->tp = new enumdef(0);
                        break;
                default:
                        on->tp =  new classdef(csu,0);
                        ((Pclass)on->tp)->string = tn->string;
                        tn->tp = new basetype(COBJ,on);
                        ((Pbase)tn->tp)->b_table = ((Pclass)on->tp)->memtbl;
                }
                PERM(tn);
                PERM(tn->tp);
                PERM(on);
                PERM(on->tp);
/*fprintf(stderr,"tname %s -> n (%d %d) n->tp (%d %d)\n",string,tn,tn->base,tn->tp,t
                return tn;
        }
        default:
                error('i',"tname(%s %d %k)",string,this,base);
        }
}


Pname name.normalize(Pbase b, Pblock bl, bit cast)
/*
        if (bl) : a function definition (check that it really is a type

        if (cast) : no name string

        for each name on the name list
        invert the declarator list(s) and attatch basetype
        watch out for class object initializers

        convert
                struct s { int a; } a;
        into
```

```
                struct s { int a; }; struct s a;
*/
{
        Pname n;
        Pname nn;
        TOK stc = b->b_sto;
        bit tpdf = b->b_typedef;
        bit inli = b->b_inline;
        bit virt = b->b_virtual;
        Pfct f;
        Pname nx;

        if (b == 0) error('i',"%d->N.normalize(0)",this);
        if (this == 0) error('i',"0->N.normalize(%k)",base);

        if (inli && stc==EXTERN)  {
                error("both extern and inline");
                inli = 0;
        }
/*fprintf(stderr,"name.norm(%d %s) tp (%d %d)\n",this,string,tp,tp->base);*/

        if (stc==FRIEND && tp==0) {
                        /*      friend x;
                                must be handled during syntax analysis to cope with
                                        class x { friend y; y* p; };
                                "y" is not local to "x":
                                        class x { friend y; ... }; y* p;
                                is legal
                        */
                if (b->base) error(0,"T specified for friend");
                if (n_list) {
                        error("L of friends");
                        n_list = 0;
                }
                Pname nx = tname(CLASS);
                modified_tn = modified_tn->l;   /* global */
                n_sto = FRIEND;
                tp = nx->tp;
                return this;
        }

        if (cast) string = "";
        b = b->check(this);

        switch (b->base) {      /*      separate class definitions
                                        from object and function type declarations
                                */
        case COBJ:
                nn = b->b_name;
/*fprintf(stderr,"COBJ (%d %s) -> (%d %d body=%d)\n",nn,nn->string,nn->tp,nn->tp->ba
                if (((Pclass)nn->tp)->c_body==2) {      /* first occurrence */
                        if (tp && tp->base==FCT) {
                                error('s',"C%n defined as returnT for%n (did you fo
                                nn = this;
                                break;
                        }
                        nn->n_list = this;
                        ((Pclass)nn->tp)->c_body = 1;   /* other occurences */
                }
                else
                        nn = this;
                break;
        case EOBJ:
                nn = b->b_name;
                if (((Penum)nn->tp)->e_body==2) {
                        if (tp && tp->base==FCT) {
                                error('s',"enum%n defined as returnT for%n (did you
                                nn = this;
                                break;
                        }
                        nn->n_list = this;
                        ((Penum)nn->tp)->e_body = 1;
                }
                else
                        nn = this;
                break;
        default:
                nn = this;
        }

        for (n=this; n; n=nx) {
                Ptype t = n->tp;
                nx = n->n_list;
                n->n_sto = stc;

                if (t
                && n_oper==TNAME
                && t->base==FCT) {      /* HORRIBLE FUDGE: fix the bad grammar */
                        Pfct f = (Pfct)t;
                        Pfct f2 = (Pfct)f->returns;
                        if (f2 && f2->base==FCT) {
                                Pexpr e = f2->argtype;
/*error('d',"%s: mis-analyzedP toF",n->string);*/
                                if  (e->base == ELIST) {
                                        /*      get the real name,
                                                fix its type
                                        */
                                        if (e->e2 || e->e1->base!=DEREF) goto zse;
                                        Pname rn = (Pname)e->e1->e1;
                                        if (rn->base!=NAME) goto zse;
/*error('d',"realN %n b==%t",rn,b);*/
                                        f->returns = new ptr(PTR,0);
                                        b = new basetype(TYPE,ktbl->look(n->string,
                                        n->n_oper = 0;
                                        n->string = rn->string;
                                        n->base = NAME;
                                }
                        }
                }
zse:
                if (n->base == TNAME) error('i',"redefinition ofTN%n",n);
```

```
		if (t == 0) {
			if (bl == 0)
				n->tp = t = b;
			else {
				error("body of nonF%n",n);
				t = new fct(defa_type,0,0);
			}
		}

		switch (t->base) {
		case PTR:
		case RPTR:
			n->tp = ((Pptr)t)->normalize(b);
			break;
		case VEC:
			n->tp = ((Pvec)t)->normalize(b);
			break;
		case FCT:
			n->tp = ((Pfct)t)->normalize(b);
			break;
		case FIELD:
			if (n->string == 0) n->string = make_name('F');
			n->tp = t;
			Pbase tb = b;
		flatten:
			switch (tb->base) {
			case TYPE:   /* chase typedefs */
				tb = (Pbase)tb->b_name->tp;
				goto flatten;
			case INT:
				((Pbase)t)->b_unsigned = b->b_unsigned;
				((Pbase)t)->b_const = b->b_const;
				break;
			default:
				error("non-int field");
				n->tp = defa_type;
			}
			break;
		}

		f = (Pfct) n->tp;

		if (f->base != FCT) {
			if (bl) {
				error("body for nonF%n",n);
				n->tp = f = new fct(defa_type,0,0);
				continue;
			}
			if (inli) error("inline nonF %n",n);
			if (virt) error("virtual nonF %n",n);

			if (tpdf) {
				if (n->n_initializer) {
					error("Ir for typedefN%n",n);
					n->n_initializer = 0;
				}
```

```
                                n->tdef();
                        }
                        continue;
                }

                f->f_inline = inli;
                f->f_virtual = virt;

                if (tpdf) error("typedef%n",n);

                if (f->body = bl) continue;

                /*
                        Check function declarations.
                        Look for class object instansiations
                        The real ambiguity:             ; class x fo();
                                is interpreted as an extern function
                                declaration NOT a class object with an
                                empty initializer
                */
                {       Pname cn = f->returns->is_cl_obj();
                        bit clob = (cn || cl_obj_vec);
/*error('d',"%n: fr%t cn%n",n,f->returns,cn);*/
                        if (f->argtype) { /* check argument/initializer list */
                                Pname nn;

                                for (nn=f->argtype; nn; nn=nn->n_list) {
                                        if (nn->base != NAME) {
                                                if (!clob) {
                                                        error("ATX for%n",n);
                                                        goto zzz;
                                                }
                                                goto is_obj;
                                        }
/*
                                        if (nn->string) {
                                                error("AN%n inD of%n",nn,n);
                                                nn->string = 0;
                                        }
*/
                                        if (nn->tp) goto ok;
                                }
                                if (!clob) {
                                        error("FALX");
                                        goto zzz;
                                }
                is_obj:
/*fprintf(stderr,"is_obj: %d %s tp = %d %d\n",this,string,f->returns,f->returns->bas
                                /* it was an initializer: expand to constructor */
                                n->tp = f->returns;
                                if (f->argtype->base != ELIST) f->argtype = (Pname)
                                n->n_initializer = new texpr(VALUE,cn->tp,(Pexpr)f-
                                goto ok;
                        zzz:
                                if (f->argtype) {
                                        DEL(f->argtype);
```

```
                                                f->argtype = 0;
                                                f->nargs = 0;
                                                f->nargs_known = !fct_void;
                                        }
                                }
                                else {  /* T a(); => function declaration */
/*
                                        if (clob) {
                                                DEL(n->tp);
                                                n->tp = f->returns;
                                        }
*/
                                }
                        ok:
                                ;
                        }
                }
                return nn;
}

Ptype vec.normalize(Ptype vecof)
/*
*/
{
        Ptype t = typ;
        if (this == 0) error('i',"0->vec.normalize()");
        typ = vecof;

        if (t == 0) return this;

xx:
        switch (t->base) {
        case TYPE:      t = ((Pbase)t)->b_name->tp;     goto xx;
        case PTR:
        case RPTR:      return ((Pptr)t)->normalize(this);
        case VEC:       return ((Pvec)t)->normalize(this);
        case FCT:       return ((Pfct)t)->normalize(this);
        default:        error('i',"bad vectorT(%d)",t->base);
        }
}

Ptype ptr.normalize(Ptype ptrto)
{
        Ptype t = typ;
        if (this == 0) error('i',"0->ptr.normalize()");
        typ = ptrto;

        if (t == 0) {
                Pbase b = (Pbase) ptrto;
                if (Pfctvec_type
                && rdo==0
                && b->b_unsigned==0
                && b->b_const==0
                && base==PTR) {
                        switch (b->base) {
                        case INT:
```

```
                                        delete this;
                                        return Pint_type;
                                case CHAR:
                                        delete this;
                                        return Pchar_type;
                                case VOID:
                                        delete this;
                                        return Pvoid_type;
                                case TYPE:
                                        break;
                                }
                        }
                        if (base==RPTR && b->base==VOID) error("void& is not a validT");
                        return this;
                }
xx:
        switch (t->base) {
        case TYPE:      t = ((Pbase)t)->b_name->tp; goto xx;
        case PTR:
        case RPTR:      return ((Pptr)t)->normalize(this);
        case VEC:       return ((Pvec)t)->normalize(this);
        case FCT:       return ((Pfct)t)->normalize(this);
        default:        error('i',"badPT(%d)",t->base);
        }
}

Ptype fct.normalize(Ptype ret)
/*
        normalize return type
*/
{
        register Ptype t = returns;

        if (this==0 || ret==0) error('i',"%d->fct.normalize(%d)",this,ret);

        returns = ret;
        if (t == 0) return this;

        if (argtype) {
                if (argtype->base != NAME) {
                        error('i',"syntax: ANX");
                        argtype = 0;
                        nargs = 0;
                        nargs_known = 0;
                }
/*
                else {
                        Pname n;
                        for (n=argtype; n; n=n->n_list) {
                                if (n->string) {
                                        error("N inATL");
                                        n->string = 0;
                                }
                        }
                }
```

```
*/
        }

xx:
        switch (t->base) {
        case PTR:
        case RPTR:      return ((Pptr)t)->normalize(this);
        case VEC:       return ((Pvec)t)->normalize(this);
        case FCT:       return ((Pfct)t)->normalize(this);
        case TYPE:      t = ((Pbase)t)->b_name->tp;     goto xx;
        default:        error('i',"badFT:%k",t->base);
        }

}


void fct.argdcl(Pname dcl)
/*
        sort out the argument types for old syntax:
                        f(a,b) int a; char b; { ... }
        beware of
                        f(a) struct s { int a; }; struct s a;
*/
{
        Pname n;
/*fprintf(stderr,"%d argtype %d %d dcl %d %d\n",this, argtype, argtype?argtype->base
        switch (base) {
        case FCT:       break;
        case ANY:       return;
        default:        error('i',"argdcl(%d)",base);
        }

        if (argtype) {
                switch (argtype->base) {
                case NAME:
                        if (dcl) error("badF definition syntax");
                        for (n=argtype; n; n=n->n_list) {
                                if (n->string == 0) {
                                /*      error('w',"AN missing inF definition");*/
                                        n->string = make_name('A');
                                }
                        }
                        return;
                case ELIST:     /* expression list:     f(a ...) */
                {       Pexpr e;
                        Pname nn;
                        Pname tail = 0;
                        n = 0;
                        if (old_fct_accepted == 0) error("old styleF definition");
                        for (e=(Pexpr)argtype; e; e=e->e2) {
                                /* scan the elist and build a NAME list */
                                Pexpr id = e->e1;
                                if (id->base != NAME) {
                                        error("NX inAL");
                                        argtype = 0;
                                        dcl = 0;
                                        goto xxx;
                                }
                                nn = new name(id->string);
                                if (n)
                                        tail = tail->n_list = nn;
                                else
                                        tail = n = nn;
                        }
                xxx:
                        argtype = n;
                        break;
                }
                default:
                        error("ALX(%d)",argtype->base);
                        argtype = 0;
                        dcl = 0;
                }
        }
        else {
                nargs_known = 1;
                nargs = 0;
                if (dcl) error("ADL forF withoutAs");
                return;
        }

        nargs_known = 0;

        if (dcl) {
                Pname d;
                Pname dx;
                /*      for each  argument name see if its type is specified
                        in the declaration list otherwise give it the default type
                */

                for (n=argtype; n; n=n->n_list) {
                        char* s = n->string;
                        if (s == 0) {
                                error("AN missing inF definition");
                                n->string = s = make_name('A');
                        }
                        else if (n->tp) error("twoTs forA %s",n->string);

                        for (d=dcl; d; d=d->n_list) {
                                if (strcmp(s,d->string) == 0) {
                                        if (d->tp->base == VOID) {
                                                error("voidA%n",d);
                                                d->tp = any_type;
                                        }
                                        n->tp = d->tp;
                                        n->n_sto = d->n_sto;
                                        d->tp = 0;      /* now merged into argtype
                                        goto xx;
                                }
                        }
                        n->tp = defa_type;
                xx:;
                        if (n->tp == 0) error('i',"noT for %s",n->string);
                }

                /*      now scan the declaration list for "unused declarations"
                        and delete it
                */
                for (d=dcl; d; d=dx) {
                        dx = d->n_list;
                        if (d->tp) {    /* not merged with argtype list */
                                /*if (d->base == TNAME)  ??? */
                                switch (d->tp->base) {
                                case CLASS:
                                case ENUM:
                                        /* WARNING: this will reverse the order of
                                           class and enum declarations
                                        */
                                        d->n_list = argtype;
                                        argtype = d;
                                        break;
                                default:
                                         error("%n inADL not inAL",d);
                                }
                        }
                }
        }

        /* add default argument types if necessary */
        for (n=argtype; n; n=n->n_list) {
                if (n->tp == 0) n->tp = defa_type;
                nargs++;
        }
}

Pname cl_obj_vec;       /* set if is_cl_obj() found a vector of class objects */
Pname eobj;             /* set if is_cl_obj() found an enum */

Pname type.is_cl_obj()
{
        bit v = 0;
        register Ptype t = this;

        eobj = 0;
        cl_obj_vec = 0;
xx:
        switch (t->base) {
        case TYPE:
                t = ((Pbase)t)->b_name->tp;
                goto xx;

        case COBJ:
                if (v) {
                        cl_obj_vec = ((Pbase)t)->b_name;
                        return 0;
                }
                else
```

```
				return ((Pbase)t)->b_name;

	case VEC:
		t = ((Pvec)t)->typ;
		v=1;
		goto xx;

	case EOBJ:
		eobj = ((Pbase)t)->b_name;
	default:
		return 0;
	}
}
```

```
/* %Z% %M% %I% %H% %T% */
/**************************************************************************

        C++ source for cfront, the C++ compiler front-end
        written in the computer science research center of Bell Labs

        Copyright (c) 1984 AT&T Technologies, Inc.
                All rigths Reserved
        THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

        If you ignore this notice the ghost of Ma Bell will haunt you forever.

norm2.c:

        "normalization" handles problems which could have been handled
        by the syntax analyser; but has not been done. The idea is
        to simplify the grammar and the actions accociated with it,
        and to get a more robust error handling

****************************************************************************/

#include "cfront.h"
#include "size.h"
extern char* malloc(int);

fct.fct(Ptype t, Pname arg, TOK known)
{
        Nt++;
        base = FCT;
        nargs_known = known;
        returns = t;
        argtype = arg;
/*fprintf(stderr,"fct t %d %d arg %d %d -> %d\n",t, t?t->base:0, arg, arg?arg->base:

        if (arg==0 || arg->base==ELIST) return;

        register Pname n;
        for (n=arg; n; n=n->n_list) {
                switch (n->tp->base) {
                case VOID:
                        argtype = 0;
                        nargs = 0;
                        nargs_known = 1;
                        if (n->string)
                                error("voidFA%n",n);
                        else if (nargs || n->n_list) {
                                error("voidFA");
                                nargs_known = 0;
                        }
                        break;
                case CLASS:
                case ENUM:
                        break;
                default:
                        nargs++;
                }
        }
}

Pexpr expr_free;
#define EBITE 250

expr.expr(TOK ba, Pexpr a, Pexpr b)
{
        register Pexpr p;

        if (this) goto ret;

        if ( (p=expr_free) == 0 ) {
                register Pexpr q = (Pexpr) malloc(EBITE*sizeof(class expr));
                for (p=expr_free=&q[EBITE-1]; q<p; p--) p->e1 = p-1;
                (p+1)->e1 = 0;
/*fprintf(stderr, "malloc %d expr_free=%d p+1=%d\n", EBITE*sizeof(class expr), expr_
        }
        else
                expr_free = p->e1;

        /* beware of alignment differences */
        if ( sizeof(expr)&1 ) {
                register char* pp = (char*)(p+1);
                while ( (char*)p<pp) *--pp = 0;
        }
        else if (sizeof(expr)&2 ) {
                register short* pp = (short*)(p+1);
                while ( (short*)p<pp ) *--pp = 0;
        }
        else {
                register int* pp = (int*)(p+1);
                while ( (int*)p<pp ) *--pp = 0;
        }

        this = p;
/*fprintf(stderr,"expr.ctor(%d,%d,%d)->%d\n",ba,a,b,this); fflush(stderr);*/

ret:
        Ne++;
        base = ba;
        e1 = a;
        e2 = b;
}

expr.~expr()
{
        NFe++;
/*fprintf(stderr,"%d->expr.dtor(%d %d %d)\n",this,base,e1,e2); */
        e1 = expr_free;
        expr_free = this;
        this = 0;
}

Pstmt stmt_free;
#define SBITE 250
```

```
stmt.stmt(TOK ba, loc ll, Pstmt a)
{
	register Pstmt p;

	if ( (p=stmt_free) == 0 ) {
		register Pstmt q = (Pstmt) malloc(SBITE*sizeof(class stmt));
		for (p=stmt_free=&q[SBITE-1]; q<p; p--) p->s_list = p-1;
		(p+1)->s_list = 0;
	}
	else
		stmt_free = p->s_list;

	/* beware of alignment differences */
	if ( sizeof(stmt)&1 ) {
		register char* pp = (char*)(p+1);
		while ( (char*)p<pp) *--pp = 0;
	}
	else if (sizeof(stmt)&2 ) {
		register short* pp = (short*)(p+1);
		while ( (short*)p<pp ) *--pp = 0;
	}
	else {
		register int* pp = (int*)(p+1);
		while ( (int*)p<pp ) *--pp = 0;
	}

	this = p;

	Ns++;
	base=ba;
	where = ll;
	s=a;
}

stmt.~stmt()
{
	NFs++;
	s_list = stmt_free;
	stmt_free = this;
	this = 0;
}

classdef.classdef(TOK b, Pname n)
{
	base = CLASS;
	csu = b;
	pubmem = n;
	memtbl = new table(CTBLSIZE,0,0);
}

basetype.basetype(TOK b, Pname n)
{
/*fprintf(stderr,"%d->basetype.basetype(%d %d)\n",this,b,n);*/
	Nbt++;
	switch (b) {
```

```
	case 0:				break;
	case TYPEDEF:	b_typedef = 1;	break;
	case INLINE:	b_inline = 1;	break;
	case VIRTUAL:	b_virtual = 1;	break;
	case CONST:	b_const = 1;	break;
	case UNSIGNED:	b_unsigned = 1;	break;
	case FRIEND:
	case OVERLOAD:
	case EXTERN:
	case STATIC:
	case AUTO:
	case REGISTER:	b_sto = b;	break;
	case SHORT:	b_short = 1;	break;
	case LONG:	b_long = 1;	break;
	case ANY:
	case ZTYPE:
	case VOID:
	case CHAR:
	case INT:
	case FLOAT:
	case DOUBLE:	base = b;	break;
	case TYPE:
	case COBJ:
	case EOBJ:
	case FIELD:
	case ASM:
		base = b;
		b_name = n;
		break;
	default:
		error('i',"badBT:%k",b);
	}
}

#define NBITE 250
Pname name_free;

name.name(char* s) : (NAME,(Pexpr)s,0)
{
	register Pname p;

	if ( (p=name_free) == 0 ) {
		register Pname q = (Pname) malloc(NBITE*sizeof(class name));
		for (p=name_free=&q[NBITE-1]; q<p; p--) p->n_tbl_list = p-1;
		(p+1)->n_tbl_list = 0;
/*fprintf(stderr, "malloc %d name_free=%d p+1=%d\n", NBITE*sizeof(class name), name_
	}
	else
		name_free = p->n_tbl_list;

	/* beware of alignment differences */
	if ( sizeof(name)&1 ) {
		register char* pp = (char*)(p+1);
		while ( (char*)p<pp) *--pp = 0;
	}
	else if (sizeof(name)&2 ) {
```

```
                register short* pp = (short*)(p+1);
                while ( (short*)p<pp ) *--pp = 0;
        }
        else {
                register int* pp = (int*)(p+1);
                while ( (int*)p<pp ) *--pp = 0;
        }

        this = p;
/*fprintf(stderr,"%d: new name %s %d\n",this,s,base); fflush(stderr);*/

        Nn++;
        where = curloc;
        lex_level = bl_level;
}


name.~name()
{
        NFn++;
/*fprintf(stderr,"delete %d: %s %d\n",this,string,base);*/
        n_tbl_list = name_free;
        name_free = this;
        this = 0;
}


nlist.nlist(Pname n)
{
        Pname nn;

        if (n==0) error('i',"nlist.nlist(0)");

        head = n;
        for (nn=n; nn->n_list; nn=nn->n_list);
        tail = nn;
        Nl++;
}

void nlist.add_list(Pname n)
{
        Pname nn;

        tail->n_list = n;
        for (nn=n; nn->n_list; nn=nn->n_list);
        tail = nn;
}

int NFl;

Pname name_unlist(class nlist * l)
{
        Pname n;
        if (l == 0) return 0;
        n = l->head;
        NFl++;
        delete l;
        return n;
}

Pstmt stmt_unlist(class slist * l)
{
        Pstmt s;
        if (l == 0) return 0;
        s = l->head;
        NFl++;
        delete l;
        return s;
}

Pexpr expr_unlist(class elist * l)
{
        Pexpr e;
        if (l == 0) return 0;
        e = l->head;
        NFl++;
        delete l;
        return e;
}
```

```
/* %Z% %M% %I% %H% %T% */
/***************************************************************************

        C++ source for cfront, the C++ compiler front-end
        written in the computer science research center of Bell Labs

        Copyright (c) 1984 AT&T Technologies, Inc. All rigths Reserved
        THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

        If you ignore this notice the ghost of Ma Bell will haunt you forever.

print.c:

        print the output of simpl, typ, or syn in a form suitable for cc input

****************************************************************************/

#include "cfront.h"

extern FILE* out_file;

/*
        print the declaration tree
*/

bit print_mode = 0;
extern int ntok;
int ntok = 0;
int forced_sm = 0;
bit Cast = 0;
Pin curr_icall;

void puttok(TOK t)
/*
        print the output representation of "t"
*/
{
        char * s;
        if (t<=0 || MAXTOK<=t) error("illegal token %d",t);
        s = keys[t];
        if (s == 0) error("V representation token %d",t);
        putst(s);
        if (12<ntok++) {
                forced_sm = 1;
                ntok = 0;
        /*      putch('\n');*/
                last_line.putline();
        }
        else if (t == SM) {
                forced_sm = 1;
                ntok = 0;
                putch('\n');
                last_line.line++;
        }
}
```

```
#define MX      20
#define NTBUF   10
class dcl_buf {
        /*
                buffer for assembling declaration (or cast)
                left contains CONST_PTR => *CONST
                              CONST_RPTR => &CONST
                                 PTR     => *
                                 RPTR    => &
                                 LP      => (
                right contains   RP      => )
                                 VEC     => [ rnode ]
                                 FCT     => ( rnode )
                                 FIELD   => : rnode
        */
        Pbase b;
        Pname n;
        TOK left[MX], right[MX];
        Pnode  rnode[MX];
        int li, ri;
public:
        void    init(Pname nn)          { b=0; n=nn; li=ri=0; };
        void    base(Pbase bb)          { b = bb; };
        void    front(TOK t)            { left[++li] = t; };
        void    back(TOK t, Pnode nod)  { right[++ri] = t; rnode[ri] = nod; };
        void    paran()                 { front(LP); back(RP,0); };
        void    put();
} *tbufvec[NTBUF], *tbuf;

int freetbuf = 0;

void dcl_buf.put()
{
        int i;

        if (MX<=li || MX<=ri) error('i',"T buffer overflow");
        if (b == 0) error('i',"noBT%s",Cast?" in cast":"");

        if (n) {
                if (n->n_sto)
                        puttok(n->n_sto);
                else if (n->n_scope==STATIC && scope_default==STATIC)
                        puttok(STATIC);
        }

        b->dcl_print();

        for( ; li; li--)
                switch (left[li]) {
                case LP:
                        puttok(LP);
                        break;
                case CONST_PTR:
                        puttok(MUL);
                        if (print_mode != SIMPL) puttok(CONST);
                        break;
```

```
		case CONST_RPTR:
			if (print_mode == SIMPL)
				puttok(MUL);
			else
				puttok(ADDROF);
			if (print_mode != SIMPL) puttok(CONST);
			break;
		case PTR:
			puttok(MUL);
			break;
		case RPTR:
			if (print_mode == SIMPL)
				puttok(MUL);
			else
				puttok(ADDROF);
		}

	if (n) n->print();

	for(i=1; i<=ri; i++)
		switch (right[i]) {
		case RP:
			puttok(RP);
			break;
		case VEC:
			puttok(LB);
			{	Pvec v = (Pvec) rnode[i];
				Pexpr d = v->dim;
				int s = v->size;
				if (d) d->print();
				if (s) fprintf(out_file,"%d",s);
			}
			puttok(RB);
			break;
		case FCT:
			{	Pfct f = (Pfct) rnode[i];
				f->dcl_print();
			}
			break;
		case FIELD:
			{	Pbase f = (Pbase) rnode[i];
				Pexpr d = (Pexpr)f->b_name;
				int s = f->b_bits;
				puttok(COLON);
				if (d) d->print();
				if (s) fprintf(out_file,"%d",s);
			}
			break;
		}
}

#define eprint(e) if (e) Eprint(e)

void Eprint(Pexpr e)
{
	switch (e->base) {
```

```
	case DUMMY:
		break;
	case NAME:
	case ID:
	case ZERO:
	case ICON:
	case CCON:
	case FCON:
	case STRING:
	case IVAL:
	case TEXT:
	case CM:
	case ELIST:
	case COLON:
	case ILIST:
	case DOT:
	case REF:
	case THIS:
	case CALL:
	case G_CALL:
	case ICALL:
	case ANAME:
		e->print();
		break;
	default:
		puttok(LP);
		e->print();
		puttok(RP);
		break;
	}
}

void name.dcl_print(TOK list)
/*
	Print the declaration for a name (list==0) or a name list (list!=0):
		For each name
		(1) print storage class
		(2) print base type
		(3) print the name with its declarators
	Avoid (illegal) repetition of basetypes which are class or enum declaration
	(A name list may contain names with different base types)
	list == SM :	terminator SM
	list == 0:	single declaration with terminator SM
	list == CM :	separator CM
*/
{
	Pname n;

	if (this == 0) error("0->name.dcl_print()");

	for (n=this; n; n=n->n_list) {
		Ptype t = n->tp;
		int sm = 0;

		if (t == 0) error('i',"name.dcl_print(%n)T missing",n);
		if (print_mode==SIMPL && n->n_stclass==ENUM) continue;
	if (n->n_stclass == STATIC) n->where.putline();

	switch (t->base) {
	case CLASS:
	{	Pclass cl = (Pclass)t;
		if (n->base == TNAME) break;
	/*	if (n->n_sto) puttok(n->n_sto); */
		cl->dcl_print(n);
		sm = 1;
		break;
	}

	case ENUM:
		((Penum)t)->dcl_print(n);
		sm = 1;
		break;

	case FCT:
	{	Pfct f = (Pfct) t;
		if (n->base == TNAME) puttok(TYPEDEF);
		if (debug==0 && f->f_inline) {
			if (print_mode==SIMPL) {
				if (f->f_virtual || n->n_addr_taken) {
					TOK st = n->n_sto;
					Pblock b = f->body;
					f->body = 0;
				/*	n->n_sto = 0;	*/
					t->dcl_print(n);
					n->n_sto = st;
					f->body = b;
				}
			}
			else {
				if (print_mode != SIMPL)
					puttok(INLINE);
				else
					putst("/* inline */");
				t->dcl_print(n);
			}
		}
		else
			t->dcl_print(n);
		break;
	}

	case OVERLOAD:
	{	Pgen g = (Pgen) t;
		Plist gl;
		fprintf(out_file,"\t/* overload %s: */\n",g->string);
		for (gl=g->fct_list; gl; gl=gl->l) {
			Pname nn = gl->f;
			nn->dcl_print(0);
			sm = 1;
		}
		break;
```

```
                }

                case ASM:
                        fprintf(out_file,"asm(\"%s\")\n",(char*)((Pbase)t)->b_name)
                        break;

                case INT:
                case CHAR:
                case LONG:
                case SHORT:
                        if (print_mode==SIMPL
                        && ((Pbase)t)->b_const
                        && (n->n_scope==STATIC || n->n_scope==FCT) ) {
                                /* do not allocate space for constants unless neces
                                if (n->n_evaluated) {
                                        sm = 1; /* no ; */
                                        break;
                                }
                        }

                default:
                {       Pexpr i = n->n_initializer;
                        if (n->base == TNAME) puttok(TYPEDEF);
/*fprintf(stderr,"%s: init %d %d tbl %d %d sto %d sc %d scope %d\n",n->string?n->str
                        if (i && n->n_sto==EXTERN && n->n_stclass==STATIC) {
                                n->n_initializer = 0;
                                t->dcl_print(n);
                                puttok(SM);
                                n->n_initializer = i;
                                n->n_sto = 0;
                                t->dcl_print(n);
                                n->n_sto = EXTERN;
                        }
                        else
                                t->dcl_print(n);

                        if (n->n_scope!=ARG) {
                                if (i) {
                                        puttok(ASSIGN);
                                        if (t!=i->tp && i->base!=ZERO && i->base!=I
                                                Ptype t1 = n->tp;
                                        cmp:
                                                switch (t1->base) {
                                                default:
                                                        i->print();
                                                        break;
                                                case TYPE:
                                                        t1 = ((Pbase)t1)->b_name->t
                                                        goto cmp;
                                                case VEC:
                                                        if (((Pvec)t1)->typ->base==
                                                                i->print();
                                                                break;
                                                        }
                                                case PTR:
                                                        puttok(LP);
```

```
                                                                {       bit oc = Cast;
                                                                        Cast = 1;
                                                                        t->print();
                                                                        Cast = oc;
                                                                }
                                                                puttok(RP);
                                                                eprint(i);
                                                        }
                                                }
                                                else
                                                        i->print();
                                        }
                                        else if (n->n_evaluated) {
                                                puttok(ASSIGN);
                                                if (n->tp->base != INT) {
                                                        puttok(LP);
                                                        puttok(LP);
                                                        {       bit oc = Cast;
                                                                Cast = 1;
                                                                n->tp->print();
                                                                Cast = oc;
                                                        }
                                                        fprintf(out_file,")%d)",n->n_val);
                                                }
                                                else
                                                        fprintf(out_file,"%d",n->n_val);
                                        }
                                }
                        }
                }

                switch (list) {
                case SM:
                        if (sm==0) puttok(SM);
                        break;
                case 0:
                        if (sm==0) puttok(SM);
                        return;
                case CM:
                        if (n->n_list) puttok(CM);
                        break;
                }
        }
}

void name.print()
/*
        print just the name itself
*/
{
        if (this == 0) error('i',"0->name.print()");

        if (string == 0) {
                if (print_mode == ERROR) putst(" ?");
                return;
        }
```

```
	switch (base) {
	default:
		error('i',"%d->name.print() base=%d",this,base);
	case TNAME:
		putst(string);
		return;
	case NAME:
	case ANAME:
		break;
	}

	switch (print_mode) {
	case SIMPL:
	{	Ptable tbl;
		int i = n_union;
		if (tp) {
			switch (tp->base) {
			default:
				if (tbl=n_table) {
					Pname tn;
					if (tbl == gtbl) break;
					if (tn=tbl->t_name) {
						if (i)
							fprintf(out_file,"_%s__O%d.
						else
							fprintf(out_file,"_%s_",tn-
						break;
					}
				}
				switch (n_stclass) {
				case STATIC:
				case EXTERN:
					if (i) fprintf(out_file,"_O%d.__C%d_",i,i);
					break;
				default:
					if (i)
						fprintf(out_file,"_auto__O%d.__C%d_
					else
						fprintf(out_file,"_auto_");
				}
				break;
			case CLASS:
			case ENUM:
				break;
			}
		}
		break;
	}
	case ERROR:
	{	Ptable tbl;
		char* cs;
		bit f = 0;
		if (tp) {
			switch (tp->base) {
			case OVERLOAD:
				case FCT:
					f = 1;
				default:
					if (tbl=n_table) {
						if (tbl == gtbl) {
							if (f == 0) putstring("::");
						}
						else {
							if (tbl->t_name) {
								cs = tbl->t_name->string;
								fprintf(out_file,"%s::",cs)
							}
						}
					}
					if (n_sto==REGISTER
					&& n_scope==ARG
					&& strcmp(string,"this")==0) {
						Ptype tt = ((Pptr)tp)->typ;
						Pname cn = ((Pbase)tt)->b_name;
						fprintf(out_file,"%s::",cn->string);
					}
					break;
				case CLASS:
				case ENUM:
				case TYPE:
					break;
				}

				switch (n_oper) {
				case 0:
				case TYPE:
					putstring(string);
					break;
				case DTOR:
					puttok(COMPL);
				case CTOR:
					putstring(cs);
					break;
				default:
					putstring("operator");
					putstring(keys[n_oper]);
					break;
				}
				if (f) putstring("()");
			}
			else
				putstring(string);
			return;
		}
		default:
			if (n_qualifier) {
				n_qualifier->print();
				puttok(DOT);
			}
		}
```

```
        putst(string);
}


void type.print()
{
/*fprintf(stderr,"type %d %d\n",this,base); fflush(stderr);*/
        switch (base) {
        case PTR:
        case RPTR:
                ((Pptr)this)->dcl_print(0);
                break;
        case FCT:
                ((Pfct)this)->dcl_print();
                break;
        case VEC:
                ((Pvec)this)->dcl_print(0);
                break;
        case CLASS:
        case ENUM:
                if (print_mode == ERROR)
                        fprintf(out_file,"%s",base==CLASS?"class":"enum");
                else
                        error('i',"%d->T.print(%k)",this,base);
                break;
        case TYPE:
                if (Cast) {
                        ((Pbase)this)->b_name->tp->print();
                        break;
                }
        default:
                ((Pbase)this)->dcl_print();
        }
}

char* type.signature(register char* p)
/*
        take a signature suitable for argument types for overloaded
        function names
*/
{
#define SDEL    '_'

        Ptype t = this;
        int pp = 0;


xx:
        switch (t->base) {
        case TYPE:      t = ((Pbase)t)->b_name->tp;     goto xx;
        case PTR:       *p++ = 'P';     t = ((Pptr)t)->typ;     pp=1;   goto xx;
        case RPTR:      *p++ = 'R';     t = ((Pptr)t)->typ;     pp=1;   goto xx;
        case VEC:       *p++ = 'V';     t = ((Pvec)t)->typ;     pp=1;   goto xx;
        case FCT:
        {       Pfct f = (Pfct)this;
                Pname n;
                t = (f->s_returns) ? f->s_returns : f->returns;
                *p++ = 'F';
/*
                if (t) p = t->signature(p);
                *p++ = SDEL;
*/
                for (n=f->argtype; n; n=n->n_list) {
                        p = n->tp->signature(p);
                        *p++ = SDEL;
                }
                *p++ = SDEL;
                *p =0;
                return p;
        }
        }

        if ( ((Pbase)t)->b_unsigned ) *p++ = 'U';

        switch (t->base) {
        case ANY:       *p++ = 'A';     break;
        case ZTYPE:     *p++ = 'Z';     break;
        case VOID:      *p++ = 'V';     break;
        case CHAR:      *p++ = (pp)?'C':'I';    break;
        case SHORT:     *p++ = (pp)?'S':'I';    break;
        case EOBJ:
        case INT:       *p++ = 'I';     break;
        case LONG:      *p++ = 'L';     break;
        case FLOAT:     *p++ = 'F';     break;
        case DOUBLE:    *p++ = 'D';     break;
        case COBJ:      *p++ = 'C';
                        strcpy(p,((Pbase)t)->b_name->string);
                        while (*p++) ;
                        *(p-1) = SDEL;
                        break;
        case FIELD:
        default:
                error('i',"signature of %k",t->base);
        }

        *p = 0;
        return p;
}

void basetype.dcl_print()
{
        Pname nn;
        Pclass cl;

        if (print_mode != SIMPL) {
                if (b_virtual) puttok(VIRTUAL);
                if (b_inline) puttok(INLINE);
                if (b_const) puttok(CONST);
        }
        if (b_unsigned) puttok(UNSIGNED);

        switch (base) {
```

```
	case ANY:
		putst("any");
		break;

	case ZTYPE:
		putst("zero");
		break;

	case VOID:
		if (print_mode == SIMPL) {
			puttok(INT);
			break;
		}
	case CHAR:
	case SHORT:
	case INT:
	case LONG:
	case FLOAT:
	case DOUBLE:
		puttok(base);
		break;

	case EOBJ:
		nn = b_name;
	eob:
		if (print_mode == SIMPL)
			puttok(INT);
		else {
			puttok(ENUM);
			nn->print();
		}
		break;

	case COBJ:
		nn = b_name;
	cob:
		cl = (Pclass)nn->tp;
		switch (cl->csu) {
		case UNION:
		case ANON:	puttok(UNION); break;
		default:	puttok(STRUCT);
		}
		putst(cl->string);
		break;

	case TYPE:
		if (print_mode == SIMPL) {
			switch (b_name->tp->base) {
			case COBJ:
				nn = ((Pbase)b_name->tp)->b_name;
				goto cob;
			case EOBJ:
				nn = ((Pbase)b_name->tp)->b_name;
				goto eob;
			}
		}
```

```
                b_name->print();
                break;

        default:
                if (print_mode == ERROR) {
                        if (0<base && base<MAXTOK && keys[base])
                                fprintf(out_file," %s",keys[base]);
                        else
                                fprintf(out_file,"?");
                }
                else
                        error('i',"%d->basetype.print(%d)",this,base);
        }
}

void type.dcl_print(Pname n)
/*
        "this" type is the type of "n". Print the declaration
*/
{
        Ptype t = this;
        Pfct f;
        Pvec v;
        Pptr p;
        TOK pre = 0;

        if (t == 0) error('i',"0->dcl_print()");
        if (n && n->tp!=t) error('i',"not %n'sT (%d)",n,t);

        if (base == OVERLOAD) {
                if (print_mode == ERROR) {
                        puttok(OVERLOAD);
                        return;
                }
                Pgen g = (Pgen) this;
                Plist gl;
                fprintf(out_file,"\t/* overload %s: */\n",g->string);
                for (gl=g->fct_list; gl; gl=gl->l) {
                        Pname nn = gl->f;
                        nn->tp->dcl_print(nn);
                        if (gl->l) puttok(SM);
                }
                return;
        }

        tbuf = tbufvec[freetbuf];
        if (tbuf == 0) {
                if (freetbuf == NTBUF-1) error('i',"AT nesting overflow");
                tbufvec[freetbuf] = tbuf = new class dcl_buf;
        }
        freetbuf++;
        tbuf->init(n);

        while (t) {
                TOK k;
```

```
		switch (t->base) {
		case PTR:
			p = (Pptr)t;
			k = (p->rdo) ? CONST_PTR : PTR;
			goto ppp;
		case RPTR:
			p = (Pptr)t;
			k = (p->rdo) ? CONST_RPTR : RPTR;
		ppp:
			tbuf->front(k);
			pre = PTR;
			t = p->typ;
			break;
		case VEC:
			v = (Pvec)t;
			if (Cast) {
				tbuf->front(PTR);
				pre = PTR;
			}
			else {
				if (pre == PTR) tbuf->paran();
				tbuf->back(VEC,v);
				pre = VEC;
			}
			t = v->typ;
			break;
		case FCT:
			f = (Pfct)t;
			if (pre == PTR) tbuf->paran();
			tbuf->back(FCT,f);
			pre = FCT;
			t = (f->s_returns) ? f->s_returns : f->returns;
			break;
		case FIELD:
			tbuf->back(FIELD,t);
			tbuf->base(defa_type);
			t = 0;
			break;
		case 0:
			error('i',"noBT(B=0)%s",Cast?" in cast":"");
		case TYPE:
			if (Cast) { /* unravel type in case it contains vectors */
				t = ((Pbase)t)->b_name->tp;
				break;
			}
		default:
			/* the base has been reached */
			tbuf->base((Pbase)t);
			t = 0;
			break;
		}
	}

	tbuf->put();
	freetbuf--;
}
```

```
void fct.dcl_print()
{
        Pname nn;

        if (print_mode == ERROR) {
                puttok(LP);
                for (nn=argtype; nn;) {
                        nn->tp->dcl_print(0);
                        if (nn=nn->n_list) puttok(CM); else break;
                }
                switch (nargs_known) {
                case ELLIPSIS:  puttok(ELLIPSIS); break;
                case 0:         putst("?"); break;
                }
                puttok(RP);
                return;
        }

        Pname at = (f_this) ? f_this : argtype;
        puttok(LP);
        if (body && Cast==0) {
                Ptable tbl = body->memtbl;

                for (nn=at; nn;) {
                        nn->print();
                        if (nn=nn->n_list) puttok(CM); else break;
                }
                puttok(RP);
        /*
                int i;
                if (tbl)
                        for (nn=tbl->get_mem(i=1); nn; nn=tbl->get_mem(++i) )
                                if (nn->n_scope==ARGT && nn->n_union==0) nn->dcl_pr
        */
                if (at) at->dcl_print(SM);

                if (f_init && print_mode!=SIMPL) {
                        puttok(COLON);
                        puttok(LP);
                        f_init->print();
                        puttok(RP);
                }
if (0)
                switch (nargs_known) {
                case 0:
                        putst("/* ? */");
                        break;
                case ELLIPSIS:
                        putst("/* ... */");
                }
                body->print();
        }
        else {
if (0) {
                if (print_mode == SIMPL) putst("/*");
                if (at) at->dcl_print(CM);
                switch (nargs_known) {
                case 0:
                        puttok(QUEST);
                        break;
                case ELLIPSIS:
                        puttok(ELLIPSIS);
                }
                if (print_mode == SIMPL) putst("*/");
}
                puttok(RP);
        }
}

void classdef.print_members()
{
        int i;
        Pname nn;

        if (clbase) {
                Pclass bcl = (Pclass)clbase->tp;
                bcl->print_members();
                /*      fprintf(out_file," int :0;\n"); force word alignment */
        }
        for (nn=memtbl->get_mem(i=1); nn; nn=memtbl->get_mem(++i)) {
                if (nn->base==NAME
                && nn->n_union==0
                && nn->tp->base!=FCT
                && nn->tp->base!=OVERLOAD
                && nn->tp->base!=CLASS
                && nn->tp->base!=ENUM
                && nn->n_stclass != STATIC) {
                        Pexpr i = nn->n_initializer;
                        nn->n_initializer = 0;
                        nn->dcl_print(0);
                        nn->n_initializer = i;
                }
        }
}


void classdef.dcl_print(Pname)
{
        Plist l;
        TOK c = csu;
        if (c==CLASS && print_mode==SIMPL) c = STRUCT;

        if (print_mode == SIMPL) {      /* cope with nested classes */
                int i;
                Pname nn;

                for ( nn=memtbl->get_mem(i=1); nn; nn=memtbl->get_mem(++i) ) {
/*fprintf(stderr, "mem %d %s %d union %d tp %d %d\n", nn, nn->string, nn->base, nn->
                        if (nn->base==NAME && nn->n_union==0) {
                                if (nn->tp->base == CLASS) ((Pclass)nn->tp)->dcl_pr
                        }
```

```
		}
	}

	puttok(c);
	putst(string);

	if (c_body == 0) return;
	c_body = 0;

	if (print_mode == SIMPL) {
		int i;
		int sm = 0;
		Pname nn;
		int sz = tsizeof();

		puttok(LC);
		fprintf(out_file,"/* sizeof = %d */\n",sz);
		print_members();
		puttok(RC);
		puttok(SM);


		if (virt_count) {	/* print initialized jump-table */

			for (nn=memtbl->get_mem(i=1); nn; nn=memtbl->get_mem(++i) )
				if (nn->base==NAME && nn->n_union==0) { /* declare
					Ptype t = nn->tp;
					switch (t->base) {
					case FCT:
					{	Pfct f =(Pfct) t;
						if (f->f_virtual == 0) break;
						f->returns->print();
						nn->print();
						putst("()");
						puttok(SM);
						break;
					}
					case OVERLOAD:
					{	Pgen g = (Pgen)t;
						Plist gl;
						for (gl=g->fct_list; gl; gl=gl->l)
							Pfct f = (Pfct) gl->f->tp;
							if (f->f_virtual == 0) brea
							f->returns->print();
							gl->f->print();
							putst("()");
							puttok(SM);
						}
					}
					}
				}
			}

			fprintf(out_file,"static int (*%s__vtbl[])() =",string);
			puttok(LC);
			for (i=0; i<virt_count; i++) {
```

```
                                fprintf(out_file," (int(*)()) ");
                                virt_init[i]->print();
                                puttok(CM);
                        }
                        puttok(ZERO);
                        puttok(RC);
                        puttok(SM);
                }

                for (nn=memtbl->get_mem(i=1); nn; nn=memtbl->get_mem(++i) ) {
                        if (nn->base==NAME && nn->n_union==0) {
                                Ptype t = nn->tp;
                                switch (t->base) {
                                case FCT:
                                case OVERLOAD:
                                        break;
                                default:
                                        if (nn->n_stclass == STATIC) {
                                                nn->n_sto = 0;
                                                nn->dcl_print(0);
                                        }
                                }
                        }
                }

                for (nn=memtbl->get_mem(i=1); nn; nn=memtbl->get_mem(++i) ) {
                        if (nn->base==NAME && nn->n_union==0) {
                                Pfct f = (Pfct)nn->tp;
                                switch (f->base) {
                                case FCT:
                                        /* suppress duplicate or spurious declarati
                                        if (debug==0 && f->f_virtual) break;
                                        if (debug==0 && f->f_inline) break;
                                        nn->dcl_print(0);
                                        break;
                                case OVERLOAD:
                                        nn->dcl_print(0);
                                        break;
                                }
                        }
                }

                for (l=friend_list; l; l=l->l) {
                        Pname nn = l->f;
/*fprintf(stderr,"friend %s %d\n",nn->string,nn->tp->base);*/
                        switch (nn->tp->base) {
                        case FCT:
                                putst("/* friend */");
                                nn->dcl_print(0);
                                break;
                        case OVERLOAD: /* first fct */
                                l->f = nn = ((Pgen)nn->tp)->fct_list->f;
                                putst("/* friend */");
                                nn->dcl_print(0);
                                break;
                        }
                }
                return;
        }

        if (clbase) {
                puttok(COLON);
                if (pubbase) puttok(PUBLIC);
                clbase->print();
        }
        puttok(LC);

                for (l=friend_list; l; l=l->l) {
                        Pname fr = l->f;
                        puttok(FRIEND);
                        switch (fr->tp->base) {
                        case FCT:
                        default:
                                fr->print();
                                puttok(SM);
                        }
                }

        if (privmem) privmem->dcl_print(SM);
        if (memtbl) memtbl->dcl_print(NE,PUBLIC);
        puttok(PUBLIC);
        puttok(COLON);
        if (pubmem) pubmem->dcl_print(SM);
        if (memtbl) memtbl->dcl_print(EQ,PUBLIC);

        if (pubdef) {
                puttok(PUBLIC);
                puttok(COLON);
                pubdef->print();
                puttok(SM);
        }

        puttok(RC);
}

void enumdef.dcl_print(Pname n)
{
        if (print_mode == SIMPL) {
                if (mem) {
                        fprintf(out_file,"/* enum %s */\n",n->string);
                        mem->dcl_print(SM);
                }
        }
        else {
                puttok(ENUM);
                if (n) n->print();
                puttok(LC);
                if (mem) mem->dcl_print(SM);
                puttok(RC);
        }
}
```

```
int addrof_cm;

void expr.print()
{
        if (this == 0) error('i',"0->expr.print()");
        if (this==e1 || this==e2) error('i',"(%d%k)->expr.print(%d %d)",this,base,e
/*error('d',"expr %d%k e1=%d e2=%d tp2=%d",this,base,e1,e2,tp2);*/
        switch (base) {
        case NAME:
        {       Pname n = (Pname) this;
                if (n->n_evaluated) {
                        if (n->tp->base != INT) {
                                puttok(LP);
                                puttok(LP);
                                {       bit oc = Cast;
                                        Cast = 1;
                                        n->tp->print();
                                        Cast = oc;
                                }
                                fprintf(out_file,")%d)",n->n_val);
                        }
                        else
                                fprintf(out_file,"%d",n->n_val);
                }
                else
                        n->print();
                break;
        }
        case ANAME:
                if (curr_icall) {       /*in expansion: look it up */
                        Pname n = (Pname)this;
                        int argno = n->n_val;
                        Pin il;
                        for (il=curr_icall; il; il=il->i_next)
                                if (n->n_table == il->i_table) goto aok;
                        goto bok;
                aok:
                        if (n = il->local[argno])
                                n->print();
                        else {
                                Pexpr ee = il->arg[argno];
                                Ptype t = il->tp[argno];
                                if (ee==0 || ee==this) error('i',"%d->expr.print(A
                                if (t!=ee->tp && t->is_cl_obj()==0 && eobj==0) {
                                        puttok(LP);
                                        puttok(LP);
                                        {       bit oc = Cast;
                                                Cast = 1;
                                                t->print();
                                                Cast = oc;
                                        }
                                        puttok(RP);
                                        eprint(ee);
                                        puttok(RP);
                                }
                                else
```

```
                                        eprint(ee);
                        }
                }
                else {
                bok:    /* in body: print it: */
                        ((Pname)this)->print();
                }
                break;

        case ICALL:
        {       il->i_next = curr_icall;
                curr_icall = il;
                if (il == 0) error('i',"expr.print: iline missing");
                Pexpr a0 = il->arg[0];
                int val = QUEST;
                if (il->fct_name->n_oper != CTOR) goto dumb;

                /*
                        find the value of "this"
                        if the argument is a "this" NOT assigned to
                        by the programmer, it was initliazed
                */

                switch (a0->base) {
                case ZERO:
                        val = 0;
                        break;
                case ADDROF:
                case G_ADDROF:
                        val = 1;
                        break;
                case CAST:
                        if (a0->e1->base == ANAME) {
                                Pname a = (Pname)a0->e1;
                                if (a->n_assigned_to == FUDGE111) val = FUDGE111;
                        }
                }

                if (val==QUEST) goto dumb;
/*error('d',"%n's this == %d",il->fct_name,val);*/
                /*
                        now find the test:  "(this==0) ? _new(sizeof(X)) : 0"

                        e1 is a comma expression,
                        the test is either the first sub-expression
                                or the first sub-expression after the assignments
                                        initializing temporary variables
                 */

        {       Pexpr e = e1;
        lx:
                switch (e->base) {
                case CM:
                /*      if (val==1 && e->e1->base==ASSIGN) {
                                Pexpr ass = e->e1;
                                Pname a = e->e1->e1;
                                if (a->base==ANAME && 1) {
                                }
                        }
                */
                        e = (e->e2->base==QUEST || e->e1->base==ASSIGN) ? e->e2 : e
                        goto lx;

                case QUEST:
                {       Pexpr q = e->cond;
                        if (q->base==EQ && q->e1->base==ANAME && q->e2==zero) {
                                Pname a = (Pname)q->e1;
                                Pexpr saved = new expr(0,0,0);
                                *saved = *e;
                                *e = (val==0) ? *e->e1 : *e->e2;
                                eprint(e1);
                                *e = *saved;
                                delete saved;
                                curr_icall = il->i_next;
                                return;
                        }
                }
                }
        }
        dumb:
                eprint(e1);
                if (e2) ((Pstmt)e2)->print();
                curr_icall = il->i_next;
                break;
        }
        case REF:
        case DOT:
                eprint(e1);
                puttok(base);
                mem->print();
                break;

        case VALUE:
                tp2->print();
                puttok(LP);
                if (e2) {
                        putst("/* &");
                        e2->print();
                        putst(", */");
                }
                if (e1) e1->print();
                puttok(RP);
                break;

        case SIZEOF:
                puttok(SIZEOF);
                if (e1 != dummy) {
                        eprint(e1);
                }
                else if (tp2) {
                        puttok(LP);
                        tp2->print();
```

```
				puttok(RP);
			}
			break;

		case NEW:
			puttok(NEW);
			tp2->print();
			if (e1) {
				puttok(LP);
				e1->print();
				puttok(RP);
			}
			break;

		case CAST:
			puttok(LP);
			puttok(LP);
			if (tp2->base == VOID)
				puttok(VOID);
			else {
				bit oc = Cast;
				Cast = 1;
				tp2->print();
				Cast = oc;
			}
			puttok(RP);
			puttok(LP);
			e1->print();
			puttok(RP);
			puttok(RP);
			break;

		case ICON:
		case FCON:
		case CCON:
		case ID:
			putst(string);
			break;

		case STRING:
			fprintf(out_file,"\"%s\"",string);
			break;

		case THIS:
		case ZERO:
			puttok(base);
			break;

		case IVAL:
			fprintf(out_file,"%d",(int)e1);
			break;

		case TEXT:
			if (e2)
				fprintf(out_file, " %s_%s", (char*)e1, (char*)e2);
			else
```

```
                                fprintf(out_file, " %s", (char*)el);
                        break;

                case DUMMY:
                        break;

                case G_CALL:
                case CALL:
                {       Pname fn = fct_name;
                        Pname at;
                        if (fn && print_mode==SIMPL) {
                                Pfct f = (Pfct)fn->tp;
                                if (f->base==OVERLOAD) { /* overloaded after call */
                                        Pgen g = (Pgen)f;
                                        fct_name = fn = g->fct_list->f;
                                        f = (Pfct)fn->tp;
                                }
                                fn->print();
                                at = (f->f_this) ? f->f_this : f->argtype;
                        }
                        else {
/*error('d',"e1%k e1->tp %d %d%t",e1->base,e1->tp,e1->tp->base,e1->tp);*/
                                eprint(e1);
                                if (e1->tp) {   /* pointer to fct */
                                        at = ((Pfct)e1->tp)->argtype;
                                }
                                else {          /* virtual: argtype encoded */
                                        at = (Pname)e1->e1->tp;
                                }
                        }
                        puttok(LP);
                        if (e2) {
                                if (at && print_mode==SIMPL) {
                                        Pexpr e = e2;
                                        while (at) {
                                                Pexpr ex;
                                                Ptype t = at->tp;
/*fprintf(stderr,"at %s tp (%d %d)\n", at->string?at->string:"?", t, t?t->base:0);*/
                                                if (e == 0) error('i',"A missing for %s()",
                                                if (e->base == ELIST) {
                                                        ex = e->e1;
                                                        e =  e->e2;
                                                }
                                                else
                                                        ex = e;

                                                if (ex==0) error('i',"A ofT%t missing",t);

                                                if (t!=ex->tp && t->is_cl_obj()==0 && eobj=
                                                        puttok(LP);
                                                        {       bit oc = Cast;
                                                                Cast = 1;
                                                                t->print();
                                                                Cast = oc;
                                                        }
                                                        puttok(RP);
                                /*              puttok(LP);
                                                ex->print();
                                                puttok(RP);
                                */
                                                eprint(ex);
                                        }
                                        else
                                                ex->print();
                                        at = at->n_list;
                                        if (at) puttok(CM);
                                }
                                if (e) {
                                        puttok(CM);
                                        e->print();
                                }
                        }
                        else
                                e2->print();
                }
                puttok(RP);
                break;
        }
        case ASSIGN:
                if (e1->base==ANAME && ((Pname)e1)->n_assigned_to==FUDGE111) {
                        /* suppress assignment to "this" that has been optimized aw
                        Pname n = (Pname)e1;
                        int argno = n->n_val;
                        Pin il;
                        for (il=curr_icall; il; il=il->i_next)
                                if (il->i_table == n->n_table) goto akk;
                        goto bkk;
                akk:
                        if (il->local[argno] == 0) {
                                e2->print();
                                break;
                        }
                }
        case EQ:
        case NE:
        case GT:
        case GE:
        case LE:
        case LT:
        bkk:
                eprint(e1);
                puttok(base);
                if (e1->tp!=e2->tp && e2->base!=ZERO) { /* cast, but beware of int!
                        Ptype t1 = e1->tp;
                cmp:
                        switch (t1->base) {
                        default:        break;
                        case TYPE:      t1 = ((Pbase)t1)->b_name->tp; goto cmp;
                        case PTR:
                        case VEC:
                                puttok(LP);
                                {       bit oc = Cast;
```

```
                                                Cast = 1;
                                                e1->tp->print();
                                                Cast = oc;
                                        }
                                        puttok(RP);
                                }
                        }
                        eprint(e2);
                        break;

                case DEREF:
                        if (e2) {
                                eprint(e1);
                                puttok(LB);
                                e2->print();
                                puttok(RB);
                        }
                        else {
                                puttok(MUL);
                                eprint(e1);
                        }
                        break;

                case ILIST:
                        puttok(LC);
                        if (e1) e1->print();
                        puttok(RC);
                        break;

                case ELIST:
                {       Pexpr e = this;
                        forever {
                                if (e->base == ELIST) {
                                        e->e1->print();
                                        if (e = e->e2)
                                                puttok(CM);
                                        else
                                                return;
                                }
                                else {
                                        e->print();
                                        return;
                                }
                        }
                }
                case QUEST:
                        eprint(cond);
                        puttok(QUEST);
                        eprint(e1);
                        puttok(COLON);
                        eprint(e2);
                        break;

                case CM:        /* do &(a,b) => (a,&b) for previously checked inlines */
                        switch (e1->base) {
                        case ZERO:
```

```
		case IVAL:
		case ICON:
		case NAME:
		case DOT:
		case REF:
		case FCON:
		case FVAL:
		case STRING:
			puttok(LP);
			goto le2;
		default:
			puttok(LP);
			{	int oo = addrof_cm;
				addrof_cm = 0;
				eprint(e1);
				addrof_cm = oo;
			}
			puttok(CM);
		le2:
			if (addrof_cm) {
				switch (e2->base) {
				case CAST:
					switch (e2->e2->base) {
					case CM:
					case ICALL:	goto ec;
					}
				case NAME:
				case DOT:
				case DEREF:
				case REF:
				case ANAME:
					puttok(ADDROF);
					addrof_cm--;
					eprint(e2);
					addrof_cm++;
					break;
				case ICALL:
				case CM:
				ec:
					eprint(e2);
					break;
				case G_CALL:
					/* & ( e, ctor() ) with temporary optimized
					if (e2->fct_name
					&& e2->fct_name->n_oper==CTOR) {
						addrof_cm--;
						eprint(e2);
						addrof_cm++;
						break;
					}
				default:
					error('i',"& inlineF call (%k)",e2->base);
				}
			}
			else {
				eprint(e2);
				}
				puttok(RP);
			}
			break;

		case UMINUS:
		case NOT:
		case COMPL:
			puttok(base);
			eprint(e2);
			break;
		case ADDROF:
		case G_ADDROF:
			switch (e2->base) {
			case DEREF:
				if (e2->e2 == 0) {
					e2->e1->print();
					return;
				}
				break;
			case ICALL:
				addrof_cm++;
				eprint(e2);
				addrof_cm--;
				return;
			}

			switch (e2->tp->base) {
			case FCT:
				break;	/* suppress cc warning on &f */
			default:
				puttok(ADDROF);
			}
			eprint(e2);
			break;

		case PLUS:
		case MINUS:
		case MUL:
		case DIV:
		case MOD:
		case LS:
		case RS:
		case AND:
		case OR:
		case ER:
		case ANDAND:
		case OROR:
		case ASPLUS:
		case ASMINUS:
		case ASMUL:
		case ASMOD:
		case ASDIV:
		case ASLS:
		case ASRS:
		case ASOR:
        case ASER:
        case ASAND:
        case DECR:
        case INCR:
                eprint(e1);
                puttok(base);
                eprint(e2);
                break;

        default:
                error('i',"%d->expr.print%k",this,base);
        }
}

Pexpr aval(Pname a)
{
        int argno = a->n_val;
        Pin il;
        for (il=curr_icall; il; il=il->i_next)
                if (il->i_table == a->n_table) goto aok;
        return 0;
aok:
        Pexpr aa = il->arg[argno];
/*error('d',"aval(%n) -> %k",a,aa->base);*/
ll:
        switch (aa->base) {
        case CAST:      aa = aa->e1; goto ll;
        case ANAME:     return aval((Pname)aa);
        default:        return aa;
        }
}

#define putcond()       puttok(LP); e->print(); puttok(RP)

void stmt.print()
{
        if (forced_sm) {
                forced_sm = 0;
                where.putline();
        }
/*error('d',&where,"stmt.print %d:%k s %d s_list %d",this,base,s,s_list);*/

        if (memtbl && base!=BLOCK) { /* also print declarations of temporaries */
                puttok(LC);
                Ptable tbl = memtbl;
                memtbl = 0;
                Pname n;
                int i;
                int bl = 1;
                for (n=tbl->get_mem(i=1); n; n=tbl->get_mem(++i)){
                        /* avoid double declarartion of temporaries from inlines */
                        char* s = n->string;
                        if (s[0]!='_' || s[1]!='X') {
                                n->dcl_print(0);
                                bl = 0;
                        }
                        Pname cn;
                        if (bl && (cn=n->tp->is_cl_obj()) && Pclass(cn->tp)->has_dt
                }
/*error('d',"%d (tbl=%d) list %d",this,tbl,s_list);*/
                if (bl) {
                        Pstmt sl = s_list;
                        s_list = 0;
                        print();
                        memtbl = tbl;
                        puttok(RC);
                        if (sl) {
                                s_list = sl;
                                sl->print();
                        }
                }
                else {
                        print();
                        memtbl = tbl;
                        puttok(RC);
                }
                return;
        }

        switch (base) {
        default:
                error('i',"stmt.print(base=%k)",base);
        case ASM:
                fprintf(out_file,"asm(\"%s\");\n",(char*)e);
                break;
        case DCL:
                d->dcl_print(SM);
                break;
        case BREAK:
        case CONTINUE:
                puttok(base);
                puttok(SM);
                break;
        case DEFAULT:
                puttok(base);
                puttok(COLON);
                s->print();
                break;
        case SM:
/*if (e->base==CALL || e->base==G_CALL) error('d',"%n",(Pname)e->e1);*/
                if (e) {
                        e->print();
                        if (e->base==ICALL && e->e2) break;     /* a block: no SM *
                }
                puttok(SM);
                break;
        case WHILE:
                puttok(WHILE);
                putcond();
                s->print();
                break;
        case DO:
                puttok(DO);
                s->print();
                puttok(WHILE);
                putcond();
                puttok(SM);
                break;
        case SWITCH:
                puttok(SWITCH);
                putcond();
                s->print();
                break;
        case RETURN:
                puttok(RETURN);
                if (e) e->print();
                puttok(SM);
                break;
        case DELETE:
                puttok(DELETE);
                e->print();
                puttok(SM);
                break;
        case CASE:
                puttok(CASE);
                eprint(e);
                puttok(COLON);
                s->print();
                break;
        case GOTO:
                puttok(GOTO);
                d->print();
                puttok(SM);
                break;
        case LABEL:
                d->print();
                puttok(COLON);
                s->print();
                break;
        case IF:
        {       int val = QUEST;
/*error('d',"if (%k%k%k)",e->e1->base,e->base,e->e2->base);*/
                if (e->base == ANAME) {
                        Pname a = (Pname)e;
                        Pexpr arg = aval(a);
//error('d',"arg %d%k",arg,arg->base);
                        if (arg == 0)
                                ;
                        else if (arg == zero)
                                val = 0;
                        else if (arg->base==ADDROF || arg->base==G_ADDROF)
                                val = 1;
                }
                else if (e->base == ANDAND
                && e->e1->base==ANAME
                && e->e2->base==ANAME) {
                        /* suppress spurious tests: if (this&&0) */
                        Pname a1 = (Pname)e->e1;
```

```
                                Pname a2 = (Pname)e->e2;
/*error('d',"aname%n %d %d%n %d %d",a1,a1->n_val,a1->n_table,a2,a2->n_val,a2->n_tabl
                                Pexpr arg2 = aval(a2);
                                if (arg2==zero) val = 0;        /* unsafe, sideeffects */
                        }
/*error('d',"val %d",val);*/
                        switch (val) {
                        case 1:
                                s->print();
                                break;
                        case 0:
                                if (else_stmt)
                                        else_stmt->print();
                                else
                                        puttok(SM);     /* null statement */
                                break;
                        default:
                                puttok(IF);
                                putcond();
                                if (s->s_list) {
                                        puttok(LC);
                                        s->print();
                                        puttok(RC);
                                }
                                else
                                        s->print();
                                if (else_stmt) {
                                        puttok(ELSE);
                                        if (else_stmt->s_list) {
                                                puttok(LC);
                                                else_stmt->print();
                                                puttok(RC);
                                        }
                                        else
                                                else_stmt->print();
                                }
                        }
                        break;
                }
                case FOR:
                {       int fi = for_init && for_init->base!=SM;
                        if (fi) {
                                puttok(LC);
                                for_init->print();
                        }
                        puttok(FOR);
                        puttok(LP);
                        if (fi==0 && for_init) for_init->e->print();
                        putch(';');     /* to avoid newline: not puttok(SM) */
                        eprint(e);
                        putch(';');
                        eprint(e2);
                        puttok(RP);
                        s->print();
                /*      if (for_init) {
                        *       if (s_list) s_list->print();
```

```
                                puttok(RC);
                                return;*
                                puttok(RC);
                        }*/
                        if (fi) puttok(RC);
                        break;
                }
        case PAIR:
                if (s&&s2) {
                        puttok(LC);
                        s->print();
                        s2->print();
                        puttok(RC);
                }
                else {
                        if (s) s->print();
                        if (s2) s2->print();
                }
                break;
        case BLOCK:
                puttok(LC);
                where.putline();
                if (d) d->dcl_print(SM);
                if (memtbl && own_tbl) {
                        Pname n;
                        int i;
                        for (n=memtbl->get_mem(i=1); n; n=memtbl->get_mem(++i)) {
                                if (n->tp && n->n_union==0)
                                        switch (n->n_scope) {
                                        case ARGT:
                                        case ARG:
                                                break;
                                        default:
                                                n->dcl_print(0);
                                        }
                        }
                }
                if (s) s->print();
                puttok(RC);
        }
        if (s_list) s_list->print();
}

void table.dcl_print(TOK s, TOK pub)
/*
        print the declarations of the entries in the order they were inserted
        ignore labels (tp==0)
*/
{
        register Pname* np;
        register int i;

        if (this == 0) return;

        np = entries;
        for (i=1; i<free_slot; i++) {
                register Pname n = np[i];
                switch (s) {
                case 0:
                        n->dcl_print(0);
                        break;
                case EQ:
                        if (n->tp && n->n_scope == pub) n->dcl_print(0);
                        break;
                case NE:
                        if (n->tp && n->n_scope != pub) n->dcl_print(0);
                        break;
                }
        }
}
```

```
/* %Z% %M% %I% %H% %T% */
/************************************************************************

        repr.c: stage main (views: main err)
*/

#include "cfront.h"

char* oper_name(TOK op)
/*
        return the string representation of operator "op"
*/
{
        switch (op) {
        default:        error('i',"oper_name(%k)",op);
        case CM:        return "_comma";
        case NEW:       return "_new";
        case DELETE:    return "_delete";
        case MUL:       return "_mul";
        case DIV:       return "_div";
        case MOD:       return "_mod";
        case PLUS:      return "_plus";
        case MINUS:
        case UMINUS:    return "_minus";
        case LS:        return "_lshift";
        case RS:        return "_rshift";
        case EQ:        return "_eq";
        case NE:        return "_ne";
        case LT:        return "_lt";
        case GT:        return "_gt";
        case LE:        return "_le";
        case GE:        return "_ge";
        case AND:
        case ADDROF:    return "_and";
        case OR:        return "_or";
        case ER:        return "_er";
        case ANDAND:    return "_andand";
        case OROR:      return "_oror";
        case NOT:       return "_not";
        case COMPL:     return "_compl";
        case INCR:      return "_incr";
        case DECR:      return "_decr";
        case CALL:      return "_call";
        case DEREF:     return "_vec";
        case ASSIGN:    return "_assign";
        case ASPLUS:    return "_asplus";
        case ASMINUS:   return "_asminus";
        case ASMUL:     return "_asmul";
        case ASDIV:     return "_asdiv";
        case ASMOD:     return "_asmod";
        case ASLS:      return "_asls";
        case ASRS:      return "_asrs";
        case ASAND:     return "_asand";
        case ASOR:      return "_asor";
        case ASER:      return "_aser";
        case SIZEOF:    return "sizeof";
        }
}

#define new_op(ss,v) keys[v]=ss

void otbl_init()
/*
        operator representation table
*/
{
        new_op("->",REF);
        new_op("." ,DOT);
        new_op("!" ,NOT);
        new_op("~" ,COMPL);
        new_op("++",INCR);
        new_op("--",DECR);
        new_op("*" ,MUL);
        new_op("&" ,AND);
        new_op("&" ,ADDROF);
        new_op("&" ,G_ADDROF);
        new_op("/" ,DIV);
        new_op("%" ,MOD);
        new_op("+" ,PLUS);
        new_op("-" ,MINUS);
        new_op("-" ,UMINUS);
        new_op("<<",LS);
        new_op(">>",RS);
        new_op("<" ,LT);
        new_op(">" ,GT);
        new_op("<=",LE);
        new_op(">=",GE);
        new_op("==",EQ);
        new_op("!=",NE);
        new_op("^" ,ER);
        new_op("|" ,OR);
        new_op("&&",ANDAND);
        new_op("||",OROR);
        new_op("?" ,QUEST);
        new_op(":" ,COLON);
        new_op("=" ,ASSIGN);
        new_op("," ,CM);

        new_op(";" ,SM);
        new_op("{" ,LC);
        new_op("}" ,RC);
        new_op("(" ,LP);
        new_op(")" ,RP);
        new_op("[" ,LB);
        new_op("]" ,RB);

        new_op("+=",ASPLUS);
        new_op("-=",ASMINUS);
        new_op("*=",ASMUL);
        new_op("/=",ASDIV);
        new_op("%=",ASMOD);
        new_op("&=",ASAND);
        new_op("|=",ASOR);
        new_op("^=",ASER);
        new_op(">>=",ASRS);
        new_op("<<=",ASLS);

        new_op("sizeof",SIZEOF);

        new_op("0" ,ZERO);
        new_op("," ,ELIST);
        new_op("[]" ,DEREF);
        new_op("expression list", ELIST);
        new_op("function call", CALL);
        new_op("generated function call",G_CALL);
        new_op("inline function call",ICALL);
        new_op("cast",CAST);
        new_op("inline argument",ANAME);

        new_op("class", COBJ);
        new_op("enum", EOBJ);
        new_op("union", ANON);

        new_op("function",FCT);
        new_op("pointer",PTR);
        new_op("reference",RPTR);
        new_op("vector",VEC);
        new_op("identifier",ID);
        new_op("name",NAME);
        new_op("...",ELLIPSIS);
        new_op("::",MEM);
        new_op("type name",TYPE);
        new_op("{}",BLOCK);
        new_op("pair",PAIR);
        new_op("declaration",DCL);
        new_op("character constant",CCON);
        new_op("integer constant",ICON);
        new_op("float constant",FCON);
        new_op("string",STRING);
}
```

```
/* %Z% %M% %I% %H% %T% */
/*************************************************************************

        C++ source for cfront, the C++ compiler front-end
        written in the computer science research center of Bell Labs

        Copyright (c) 1984 AT&T Technologies, Inc. All rigths Reserved
        THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

        If you ignore this notice the ghost of Ma Bell will haunt you forever.

simpl.c:

        simplify the typechecked function
        remove:         classes:
                                class fct-calls
                                operators
                                value constructors and destructors
                        new and delete operators (replace with function calls)
                        initializers            (turn them into statements)
                        constant expressions            (evaluate them)
                        inline functions                (expand the calls)
                        enums                           (make const ints)
                        unreachable code                (delete it)
        make implicit coersions explicit

        in general you cannot simplify something twice

*************************************************************************/

#include "cfront.h"
#include "size.h"
#include <ctype.h>

Pname new_fct;
Pname del_fct;
Pname vec_new_fct;
Pname vec_del_fct;
Pstmt del_list;
Pstmt block_del_list;
Pname ret_var;
bit not_inl;    /* is the current function an inline? */
Pname curr_fct; /* current function */
Pexpr init_list;
Pexpr one;


extern void simpl_init();
void simpl_init()
{
        Pname a;
        Pname al;
        Pname nw = new name(oper_name(NEW));
        Pname dl = new name(oper_name(DELETE));
        Pname vn = new name("_vec_new");
        Pname vd = new name("_vec_delete");
```

```
	new_fct = gtbl->insert(nw,0);	/* void* operator new(long); */
	delete nw;
	a = new name;
	a->tp = uint_type;		/* !!!!! */
	new_fct->tp = new fct(Pvoid_type,a,1);
	new_fct->n_scope = EXTERN;
	PERM(new_fct);
	PERM(new_fct->tp);
	new_fct->dcl_print(0);

	del_fct = gtbl->insert(dl,0);	/* void operator delete(void*); */
	delete dl;
	a = new name;
	a->tp = Pvoid_type;
	del_fct->tp = new fct(void_type,a,1);
	del_fct->n_scope = EXTERN;
	PERM(del_fct);
	PERM(del_fct->tp);
	del_fct->dcl_print(0);

	a = new name;
	a->tp = Pvoid_type;
	al = a;
	a = new name;
	a->tp = int_type;
	a->n_list = al;
	al = a;
	a = new name;
	a->tp = int_type;
	a->n_list = al;
	al = a;
	a = new name;
	a->tp = Pvoid_type;
	a->n_list = al;
	al = a;				/* (Pvoid, int, int, Pvoid) */

	vec_new_fct = gtbl->insert(vn,0);
	delete vn;
	vec_new_fct->tp = new fct(Pvoid_type,al,1);
	vec_new_fct->n_scope = EXTERN;
	PERM(vec_new_fct);
	PERM(vec_new_fct->tp);
	vec_new_fct->dcl_print(0);

	vec_del_fct = gtbl->insert(vd,0);
	delete vd;
	vec_del_fct->tp = new fct(void_type,al,1);
	vec_del_fct->n_scope = EXTERN;
	PERM(vec_del_fct);
	PERM(vec_del_fct->tp);
	vec_del_fct->dcl_print(0);

	one = new expr(IVAL,(Pexpr)1,0);
	one->tp = int_type;
	PERM(one);
```

```
}

Ptable scope = 0;        /* current scope for simpl() */
Pname expand_fn = 0;     /* name of function being expanded or 0 */
Ptable expand_tbl = 0;   /* scope for inline function variables */



Pname classdef.has_oper(TOK op)
{
        char* s = oper_name(op);
        Pname n;
        if (this == 0) error('i',"0->has_oper(%s)",s);
        n = memtbl->lookc(s,0);
        if (n == 0) return 0;
        switch (n->n_scope) {
        case 0:
        case PUBLIC:    return n;
        default:        return 0;
        }
}

int no_of_returns;

void name.simpl()
{
/*fprintf(stderr,"%s.simpl(%d %d)\n",string,tp,tp?tp->base:0); fflush(stderr);*/
        if (base == PUBLIC) return;

        if (tp == 0) error('i',"%n->name.simple(tp==0)",this);

        switch (tp->base) {
        case 0:
                error('i',"%n->name.simpl(tp->base==0)",this);

        case OVERLOAD:
        {       Plist gl;
                for (gl = ((Pgen)tp)->fct_list; gl; gl=gl->l) gl->f->simpl();
                break;
        }

        case FCT:
        {       Pfct f = (Pfct)tp;
                Pname n;
                Pname th = f->f_this;
/*error('d',"simpl%n tp=%t defined=%d th=%d n_oper%k",this,tp,tp->defined,th,n_oper)
                if (th) {
                        th->n_list = f->argtype;
                        if (n_oper == CTOR) f->s_returns = th->tp;
                }

                if (tp->defined != 1) return;
                tp->defined = 2;
                for (n=(th)?th:f->argtype; n; n=n->n_list) n->simpl();

                if (f->body) {
                        Ptable oscope = scope;
                        scope = f->body->memtbl;
                        if (scope == 0) error('i',"%n memtbl missing",this);
                        curr_fct = this;
                        f->simpl();
                        if (f->f_inline && debug==0) {
                                if (MIA<=f->nargs) {
                                        error('w',"too many arguments for inline%n
                                        f->f_inline = 0;
                                        scope = oscope;
                                        break;
                                }
                                int i = 0;
                                for (n=(th)?th:f->argtype; n; n=n->n_list) {
                                        n->base = ANAME;
                                        n->n_val = i++;
                                        if (n->n_table != scope) error('i',"%s %d %
                                }
                                expand_tbl = (f->returns->base!=VOID || n_oper==CTO
                                expand_fn = this;
                                if (expand_tbl) {
                                        f->f_expr = (Pexpr)f->body->expand();
                                        /* the body still holds the memtbl */
                                }
                                else {
                                        f->f_expr = 0;
                                        f->body = (Pblock)f->body->expand();
                                }
                                expand_fn = 0;
                                expand_tbl = 0;
                        }
                        scope = oscope;
                }
                break;
        }

        case CLASS:
                ((Pclass)tp)->simpl();
                break;
/*
        case EOBJ:
                tp->base = INT;
                break;
*/
        default:
                break;
        }

        if (n_initializer) n_initializer->simpl();
}

void fct.simpl()
/*
```

```
        call only for the function definition (body != 0)

        simplify argument initializers, and base class initializer, if any
        then simplify the body, if any

        for constructor:call allocator if this==0 and this not assigned to
                        (auto and static objects call constructor with this!=0,
                        the new operator generates calls with this==0)
                        call base & member constructors
        for destructor: call deallocator (no effect if this==0)
                        case base & member destructors

        for arguments  and function return values look for class objects
        that must be passed by constructor "operator X(X&)".

        Allocate temporaries for class object expressions, and see if
        class object return values can be passed as pointers.

        call constructor and destructor for local class variables.
*/
{
        Pexpr th = f_this;
        Ptable tbl = body->memtbl;
        Pstmt ss = 0;
        Pstmt tail;
        Pname cln;
        Pclass cl;
        Pstmt dtail = 0;

        not_inl = debug || f_inline==0;
        ret_var = tbl->look("_result",0);
        if (ret_var && not_inl==0) /* "_result" not used in inlines */
                ret_var->n_used = ret_var->n_assigned_to = ret_var->n_addr_taken =
        del_list = 0;
        block_del_list = 0;
        scope = tbl;
        if (scope == 0) error('i',"fct.simpl()");

        if (th) {
                Pptr p = (Pptr)th->tp;
                cln = ((Pbase)p->typ)->b_name;
                cl = (Pclass)cln->tp;
        }

        if (curr_fct->n_oper == DTOR) {         /* initialize del_list */
                Pexpr ee;
                Pexpr cc;
                Pestmt es;
                class ifstmt * ifs;
                Pname bcln = cl->clbase;
                Pclass bcl;
                Pname d;

                Pname fa = new name("_free");   /* fake argument for dtor */
                fa->tp = int_type;
                Pname free_arg = fa->dcl(body->memtbl,ARG);
```

```
		delete fa;
		f_this->n_list = free_arg;

		Ptable tbl = cl->memtbl;
		int i;
		Pname m;

		/* generate calls to destructors for all members of class cl */
		for (m=tbl->get_mem(i=1); m; m=tbl->get_mem(++i) ) {
			Ptype t = m->tp;
			Pname cn;
			Pclass cl;
			Pname dtor;
			if (m->n_stclass == STATIC) continue;

			if (cn = t->is_cl_obj()) {
				cl = (Pclass)cn->tp;
				if (dtor = cl->has_dtor()) {
					/*	dtor(this,0);	*/
					Pexpr aa = new expr(ELIST,zero,0);
					ee = new ref(REF,th,m);
					ee = new ref(DOT,ee,dtor);
					ee = new call(ee,aa);
					ee->fct_name = dtor;
					ee->base = G_CALL;
					es = new estmt(SM,curloc,ee,0);
					if (dtail)
						dtail->s_list = es;
					else
						del_list = es;
					dtail = es;
				}
			}
			else if (cl_obj_vec) {
				cl = (Pclass)cl_obj_vec->tp;
				if (dtor = cl->has_dtor()) {
					int esz = cl->tsizeof();
					Pexpr noe = new expr(IVAL,(Pexpr)(t->tsizeo
					Pexpr sz = new expr(IVAL,(Pexpr)esz,0);
					Pexpr mm = new ref(REF,th,m);
					Pexpr arg = new expr(ELIST,dtor,0);
					/*dtor->take_addr();*/
					dtor->lval(ADDROF);
					arg = new expr(ELIST,sz,arg);
					arg = new expr(ELIST,noe,arg);
					arg = new expr(ELIST,mm,arg);
					ee = new call(vec_del_fct,arg);
					ee->base = G_CALL;
					if (dtail)
						dtail->s_list = es;
					else
						del_list = es;
					dtail = es;
				}
			}
		}
		/* generate:	if (this) base.dtor(this,_free); or
				if (this && _free) _delete(this);
		*/
		if ( bcln
		&& (bcl=(Pclass)bcln->tp)
		&& (d=bcl->has_dtor()) ) {
			Pexpr aa = new expr(ELIST,free_arg,0);
			cc = th;
			ee = new ref(REF,th,d);
			ee = new call(ee,aa);
			/*ee->fct_name = d; NO suppress virtual */
		}
		else {
			Pexpr aa  = new expr(ELIST,th,0);
			cc = new expr(ANDAND,th,free_arg);
			ee = new call(del_fct,aa);
			ee->fct_name = del_fct;
		}
		free_arg->use();
		((Pname)th)->use();
		ee->base = G_CALL;
		es = new estmt(SM,curloc,ee,0);
		ifs = new ifstmt(curloc,cc,es,0);
		if (dtail)
			dtail->s_list = ifs;
		else
			del_list = ifs;
		dtail = ifs;

		if (del_list) del_list->simpl();
	}

int ass_count;
	if (curr_fct->n_oper == CTOR) {
		Pexpr ee;
		Ptable tbl = cl->memtbl;
		Pname m;
		int i;

		if (f_init) {		/* generate: this=base.ctor(this,args) */
			Pcall cc = (Pcall)f_init;
			Pname bn = cc->fct_name;
			Pname tt = ((Pfct)bn->tp)->f_this;
			ass_count = tt->n_assigned_to;
			f_init->simpl();
			init_list = new expr(ASSIGN,th,f_init);
		}
		else {
			ass_count = 0;
			init_list = 0;
		}

		if (cl->virt_count) {	/* generate: this->_vptr=virt_init; */
			Pname vp = cl->memtbl->look("_vptr",0);
			Pexpr vtbl = new expr(TEXT,(Pexpr)cl->string,(Pexpr)"_vtbl"
                        ee = new ref(REF,th,vp);
                        ee = new expr(ASSIGN,ee,vtbl);
                        init_list =  (init_list) ? new expr(CM,init_list,ee) : ee;
                }

                /* generate cl.new(0) for all members of cl */
                for (m=tbl->get_mem(i=1); m; m=tbl->get_mem(++i) ) {
                        Ptype t = m->tp;
                        Pname cn;
                        Pclass cl;
                        Pname ctor;
                        if (m->n_stclass == STATIC) continue;

                        if (cn=t->is_cl_obj()) {
                                cl = (Pclass)cn->tp;
                                if (ctor = cl->has_ictor()) {
                                        ee = new ref(REF,th,m);
                                        ee = new ref(DOT,ee,ctor);
                                        ee = new call(ee,0);
                                        ee->fct_name = ctor;
                                        ee->base = G_CALL;
                                        ee = ee->typ(tbl);      /* look for default
                                        ee->simpl();
                                        if (init_list)
                                                init_list = new expr(CM,init_list,e
                                        else
                                                init_list = ee;
                                }
                                else if (cl->has_ctor()) {
                                        error("%s%n, no default constructor",cl->st
                                }
                        }
                        else if (cl_obj_vec) {
                                cl = (Pclass)cl_obj_vec->tp;
                                if (ctor = cl->has_ictor()) {   /*  _new_vec(vec,no
                                        int esz = cl->tsizeof();
                                        Pexpr noe = new expr(IVAL,(Pexpr)(t->tsizeo
                                        Pexpr sz = new expr(IVAL,(Pexpr)esz,0);
                                        Pexpr mm = new ref(REF,th,m);
                                        Pexpr arg = new expr(ELIST,ctor,0);
                                        /*ctor->take_addr();*/
                                        ctor->lval(ADDROF);
                                        arg = new expr(ELIST,sz,arg);
                                        arg = new expr(ELIST,noe,arg);
                                        arg = new expr(ELIST,mm,arg);
                                        ee = new call(vec_new_fct,arg);
                                        ee->fct_name = vec_new_fct;
                                        ee->base = G_CALL;
                                /*      ee = ee->typ(tbl);      look for default a
                                        ee->simpl();
                                        if (init_list)
                                                init_list = new expr(CM,init_list,e
                                        else
                                                init_list = ee;
                                }
                                else if (cl->has_ctor()) {
                                        error("%s%n[], no default constructor",cl->
                                }
                        }
                }
        }

        no_of_returns = 0;

        tail = body->simpl();

        if (returns->base != VOID) {    /* return must have been seen */
                if (no_of_returns) {
                        switch (tail->base) {
                        case SM:
                                switch (tail->e->base) {
                                case ICALL:
                                case G_CALL:            /* not good enough */
                                        goto dontknow;
                                };
                        default:
/*fprintf(stderr,"t %d %d\n",tail->base,tail->e->base);*/
                                if (strcmp(curr_fct->string,"main"))
                                        error('w',"maybe no value returned from%n",

                                if (del_list) goto zaq;
                                break;
                        case RETURN:
                        case IF:
                        case SWITCH:
                        case DO:
                        case FOR:
                        case LABEL:
                        case BLOCK:
                        case PAIR:
                        case GOTO:
                        dontknow:
                                break;
                        }
                }
                else {
                        if (strcmp(curr_fct->string,"main"))
                                error('w',"no value returned from%n",curr_fct);
                        if (del_list) goto zaq;
                }
        }
        else if (del_list) {    /* return may not have been seen */
        zaq:
                if (tail)
                        tail->s_list = del_list;
                else
                        body->s = del_list;
                tail = dtail;
        }

        if (curr_fct->n_oper == CTOR) {
                if  ( ((Pname)th)->n_assigned_to == 0 ) {
                /* generate:    if (this==0) this=_new( sizeof(class cl) );
                                init_list ;
                */
                        ((Pname)th)->n_assigned_to = ass_count ? ass_count : FUDGE1
                        Pexpr sz = new expr(IVAL,(Pexpr)cl->tsizeof(),0);
                        Pexpr ee = new expr(ELIST,sz,0);
                        ee = new call(new_fct,ee);
                        ee->fct_name = new_fct;
                        ee->base = G_CALL;
                        ee->simpl();
                        ee = new expr(ASSIGN,th,ee);
                        Pstmt es = new estmt(SM,curloc,ee,0);
                        ee = new expr(EQ,th,zero);
                        ifstmt* ifs = new ifstmt(curloc,ee,es,0);
                        /*ifs->simpl();
                        do not simplify or "this = " will cause an extra call of ba
                        if (init_list) {
                                es = new estmt(SM,curloc,init_list,0);
                                es->s_list = body->s;
                                body->s = es;
                                if (tail == 0) tail = es;
                        }
                        ifs->s_list = body->s;
                        body->s = ifs;
                        if (tail == 0) tail = ifs;
                }

                Pstmt st = new estmt(RETURN,curloc,th,0);
                if (tail)
                        tail->s_list = st;
                else
                        body->s = st;
                tail = st;
        }
}

Pstmt block.simpl()
{
        int i;
        Pname n;
        Pstmt ss=0, sst;
        Pstmt dd=0, ddt;
        Pstmt stail;
        Ptable old_scope = scope;

        if (own_tbl == 0) {
                Pstmt obd = block_del_list;
                block_del_list = 0;
                ss = (s) ? s->simpl() : 0;
                block_del_list = obd;
                return ss;
        }

        scope = memtbl;
        if(scope->init_stat == 0) scope->init_stat = 1; /* table is simplified. */
```

```
	for (n=scope->get_mem(i=1); n; n=scope->get_mem(++i)) {
		Pstmt st = 0;
		Pname cln;
		Pexpr in = n->n_initializer;

		if (in) scope->init_stat = 2; /* initializer in this scope */

		switch (n->n_scope) {
		case ARG:
		case 0:
		case PUBLIC:
			continue;
		}

		if (n->n_stclass == STATIC) continue;

		if (in->base == ILIST)
			error('s', "initialization of automatic aggregates");

		if (n->tp == 0) continue; /* label */

		if (n->n_evaluated) continue;

		/* construction and destruction of temporaries is handled locally *
		{	char* s = n->string;
			register char c3 = s[3];
			if (s[0]=='_' && s[1]=='D' && isdigit(c3)) continue;
		}

		if ( cln=n->tp->is_cl_obj() ) {
			Pclass cl = (Pclass)cln->tp;
			Pname d = cl->has_dtor();

			if (d) {	/* n->cl.delete(0); */
				Pref r = new ref(DOT,n,d);
				Pexpr ee = new expr(ELIST,zero,0);
				Pcall dl = new call(r,ee);
				Pstmt dls = new estmt(SM,n->where,dl,0);
				dl->base = G_CALL;
				dl->fct_name = d;
				if (dd)
					ddt->s_list = dls;
				else
					dd = dls;
				ddt = dls;
			}

			if (in) {
				if (in->base == G_CALL) {	/* constructor? */
					Pname fn = in->fct_name;
					if (fn==0 || fn->n_oper!=CTOR) goto ddd;
					st = new estmt(SM,n->where,in,0);
					n->n_initializer = 0;
				}
				else
```

```
                                goto ddd;
                }
        }
        else if (cl_obj_vec) {  /* never "new x" is a pointer */
                Pclass cl = (Pclass)cl_obj_vec->tp;
                Pname d = cl->has_dtor();
                Pname c = cl->has_ictor();

                if (in) {
                        if (c) {        /*  _vec_new(vec,noe,sz,ctor); */
                                int esz = cl->tsizeof();
                                Pexpr noe = new expr(IVAL,(Pexpr)(n->tp->ts
                                Pexpr sz = new expr(IVAL,(Pexpr)esz,0);
                                Pexpr arg = new expr(ELIST,c,0);
                                /*c->take_addr();*/
                                c->lval(ADDROF);
                                arg = new expr(ELIST,sz,arg);
                                arg = new expr(ELIST,noe,arg);
                                arg = new expr(ELIST,n,arg);
                                arg = new call(vec_new_fct,arg);
                                arg->base = G_CALL;
                                arg->fct_name = vec_new_fct;
                                st = new estmt(SM,n->where,arg,0);
                                n->n_initializer = 0;
                        }
                        else
                                goto ddd;
                }
                if (d) {        /*  _vec_delete(vec,noe,sz,dtor); */
                        Pstmt dls;
                        int esz = cl->tsizeof();
                        Pexpr noe = new expr(IVAL, (Pexpr)(n->tp->tsizeof()
                        Pexpr sz = new expr(IVAL,(Pexpr)esz,0);
                        Pexpr arg = new expr(ELIST,c,0);
                        /*c->take_addr();*/
                        c->lval(ADDROF);
                        arg = new expr(ELIST,sz,arg);
                        arg = new expr(ELIST,noe,arg);
                        arg = new expr(ELIST,n,arg);
                        arg = new call(vec_del_fct,arg);
                        arg->base = G_CALL;
                        arg->fct_name = vec_del_fct;
                        dls = new estmt(SM,n->where,arg,0);
                        if (dd)
                                ddt->s_list = dls;
                        else
                                dd = dls;
                        ddt = dls;
                }
        }
        else if (in /*&& n->n_scope==FCT*/) {
                switch (in->base) {
                case ILIST:
                        switch (n->n_scope) {
                        case FCT:
                        case ARG:
```

```
                                        error('s',"Ir list for localV%n",n);
                                }
                                break;
                        case STRING:
                                if (n->tp->base==VEC) break; /* BUG char vec only *
                        default:
                        ddd:
                        {       Pexpr ee = new expr(ASSIGN,n,in);
                                st = new estmt(SM,n->where,ee,0);
                                n->n_initializer = 0;
                        }
                        }
                }

                if (st) {
                        if (ss)
                                sst->s_list = st;
                        else
                                ss = st;
                        sst = st;
                }
        }

        if (dd) {
                Pstmt od = del_list;
                Pstmt obd = block_del_list;

                dd->simpl();
                /*PERM(dd);
*/
                if (od)
                        del_list = new pair(curloc,dd,od);
                else
                        del_list = dd;
                block_del_list = dd;

                stail  = (s) ? s->simpl() : 0;

                Pfct f = (Pfct)curr_fct->tp;
                if (this!=f->body
                || f->returns->base==VOID
                || strcmp(curr_fct->string,"main")==0 ) {
                /* not dropping through the bottom of a value returning function */
                        if (stail)
                                stail->s_list = dd;
                        else
                                s = dd;
                        stail = ddt;
                }

                del_list = od;
                block_del_list = obd;
        }
        else
                stail  = (s) ? s->simpl() : 0;
```

```
	if (ss) {	/* place constructor calls */
		ss->simpl();
		sst->s_list = s;
		s = ss;
		if (stail == 0) stail = sst;
	}

	scope = old_scope;

	return stail;
}


void classdef.simpl()
{
	int i;
	Pname m;
	Pclass oc = in_class;
	int ct = has_ctor()==0;
	int dt = has_dtor()==0;
	int un = csu==UNION;

	in_class = this;

	for (m=memtbl->get_mem(i=1); m; m=memtbl->get_mem(++i) ) {
		/* should really be checked in classdef.dcl() */
		Ptype t = m->tp;
		Pexpr i = m->n_initializer;
		Pname cn;

		if ( (ct || dt || un)
		&& ( (cn=t->is_cl_obj()) || (cn=cl_obj_vec) ) ) {
			Pclass cl = (Pclass)cn->tp;
			Pname ctor = cl->has_ctor();
			Pname dtor = cl->has_dtor();

			if (ctor) {
				if (m->n_stclass==STATIC)
					error('s',"staticM%n ofC%n with constructor
				else if (un)
					error("M%n ofC%n with constructor in union"
				else if (ct)
					error('s',"M%n ofC%n with constructor inC %
			}
			if (dtor) {
				if (m->n_stclass==STATIC)
					error('s',"staticM%n ofC%n with destructor"
				else if (un)
					error("M%n ofC%n with destructor in union",
				else if (dt)
					error('s',"M%n ofC%n with destructor inC %s
			}
		}
		m->n_initializer = 0;
		m->simpl();
		m->n_initializer = i;
	}
	in_class = oc;

	Plist fl;				/* simplify friends */
	for (fl=friend_list; fl; fl=fl->l) {
		Pname p = fl->f;
		switch (p->tp->base) {
		case FCT:
		case OVERLOAD:
			p->simpl();
		}
	}
}

void expr.simpl()
{
	if (this==0 || permanent==2) return;
/*fprintf(stderr,"expr.simpl %d %d e1=%d e2=%d tp2=%d cf %d\n",this,base,e1,e2,tp2,c
	switch (base) {
	case BLOCK:
	case SM:
	case IF:
	case FOR:
	case WHILE:
	case SWITCH:
		error('i',"%k inE",base);

	case VALUE:
		error('i',"expr.simpl(value)");

	case G_ADDROF:
	case ADDROF:
		e2->simpl();
		switch (e2->base) {
		case DOT:
		case REF:
		{	Pref r = (Pref)e2;
			Pname m = r->mem;
			if (m->n_stclass == STATIC) {	/* & static member */
				Pexpr x;
			delp:
				x = e2;
				e2 = m;
				r->mem = 0;
				DEL(x);
			}
			else if (m->tp->base == FCT) {	/* & member fct */
				Pfct f = (Pfct)m->tp;
				if (f->f_virtual) {
					/* &p->f ==> p->vtbl[fi] */
					int index = f->f_virtual;
					Pexpr ie = (1<index) ? new expr(IVAL, (Pexp
					Pname vp = m->n_table->look("_vptr",0);
					r->mem = vp;
					base = DEREF;
					e1 = e2;
```

```
                                                e2 = ie;
                                        }
                                        else {
                                                goto delp;
                                        }
                                }
                        }
                        }
                        break;

        default:
                        if (e1) e1->simpl();
                        if (e2) e2->simpl();
                        break;

        case NAME:
        case DUMMY:
        case ICON:
        case FCON:
        case CCON:
        case IVAL:
        case FVAL:
        case LVAL:
        case STRING:
        case ZERO:
        case ILIST:
                        return;

        case SIZEOF:
                        base = IVAL;
                        e1 = (Pexpr)tp2->tsizeof();
                        DEL(tp2);
                        tp2 = 0;
                        break;

        case G_CALL:
        case CALL:
                        ((Pcall)this)->simpl();
                        break;

        case QUEST:
                        cond->simpl();
                        e1->simpl();
                        e2->simpl();
                        break;

        case NEW:               /* change NEW node to CALL node */
        {               Pname cln;
                        Pname ctor;
                        int sz = 1;
                        int esz;
                        Pexpr var_expr = 0;
                        Pexpr const_expr;
                        Ptype tt = tp2;
                        Pexpr arg;
```

```
		if ( cln=tt->is_cl_obj() ) {
			Pclass cl = (Pclass)cln->tp;
			if ( ctor=cl->has_ctor() ) {	/* 0->cl_ctor(args) */
				Pexpr p = zero;
				if (ctor->n_table != cl->memtbl) {
				/*	no derived constructor: pre-allocate */
					int dsz = cl->tsizeof();
					Pexpr ce = new expr(IVAL,(Pexpr)dsz,0);
					ce = new expr(ELIST,ce,0);
					p = new expr(G_CALL,new_fct,ce);
					p->fct_name = new_fct;
				}
				Pcall c = (Pcall)e1;
				c->e1 = new ref(REF,p,(Pname)c->e1);
			/*	c->set_fct_name(ctor);*/
				c->simpl();
				*this = *((Pexpr)c);
				return;
			}
		}
		else if (cl_obj_vec) {
			Pclass cl = (Pclass)cl_obj_vec->tp;
			ctor = cl->has_ictor();
			if (ctor == 0) {
				if (cl->has_ctor()) error("new %s[], no default con
				cl_obj_vec = 0;
			}
		}

	xxx:
		switch (tt->base) {
		case TYPE:
			tt = ((Pbase)tt)->b_name->tp;
			goto xxx;
		default:
			esz = tt->tsizeof();
			break;
		case VEC:
		{	Pvec v = (Pvec)tt;
			if (v->size)
				sz *= v->size;
			else if (v->dim) {
				if (var_expr)
					var_expr = new expr(MUL,var_expr,v->dim);
				else
					var_expr = v->dim;
			}
			else {
				sz = SZ_WPTR;
				break;
			}
			tt = v->typ;
			goto xxx;
		}
		}
```

```
                if (cl_obj_vec) {
                        /* call _vec_new(0,no_of_elements,element_size,ctor) */
                        const_expr = new expr(IVAL,(Pexpr)sz,0);
                        Pexpr noe = (var_expr) ? (sz!=1) ? new expr(MUL,const_expr,
                        const_expr = new expr(IVAL,(Pexpr)esz,0);
                        base = CALL;
                        arg = new expr(ELIST,ctor,0);
                        /*ctor->take_addr();*/
                        ctor->lval(ADDROF);
                        arg = new expr(ELIST,const_expr,arg);
                        arg = new expr(ELIST,noe,arg);
                        e2 = new expr(ELIST,zero,arg);
                        e1 = vec_new_fct;
                        fct_name = vec_new_fct;
                        break;
                }
                /* call _new(element_size*no_of_elements) */
                sz *= esz;
                const_expr = new expr(IVAL,(Pexpr)sz,0);
                arg = (var_expr) ? (sz!=1) ? new expr(MUL,const_expr,var_expr) :var
/*              arg->simpl();*/
                base = G_CALL;
                e2 = new expr(ELIST,arg,0);
                e1 = new_fct;
                fct_name = new_fct;
                simpl();
                break;
        }
        case CAST:
                e1->simpl();
                break;

        case REF:
                e1->simpl();
                break;
        case DOT:
                e1->simpl();
                if (e1->base == CM) {   /* &( , name). => ( ... , &name)-> */
                        Pexpr ex = e1;
                        cfr:
                        switch (ex->e2->base) {
                        case NAME:
                                base = REF;
                                ex->e2 = ex->e2->address();
                                break;
                        case CM:
                                ex = ex->e2;
                                goto cfr;
                        }
                }
                break;

        case ASSIGN:
        {       Pfct f = (Pfct)curr_fct->tp;
                Pexpr th = f->f_this;
```

```
		if (e1) e1->simpl();
		if (e2) e2->simpl();

		if (th && th==e1) {
			if (curr_fct->n_oper == CTOR) {
				if (init_list) {
					/* this=e2 => (this=e2,init_list) */
					base = CM;
					e1 = new expr(ASSIGN,e1,e2);
					e2 = init_list;
				}
			}
		}
		break;
	}
	}

	if (tp && tp->base==INT) {
		Neval = 0;
		int i = eval();
		if (Neval == 0) {
			base = IVAL;
			e1 = (Pexpr)i;
		}
	}
}


void call.simpl()
/*
	fix member function calls:
		p->f(x) becomes f(p,x)
		o.f(x)  becomes f(&o,x)
	or if f is virtual:
		p->f(x) becomes ( *p->_vptr[ type_of(p).index(f)-1 ] )(p,x)
	replace calls to inline functions by the expanded code
*/
{
	Pname fn = fct_name;
	Pfct f = (fn) ? (Pfct)fn->tp : 0;

	if (f) {
		switch(f->base) {
		case ANY:
			return;
		case FCT:
			break;
		case OVERLOAD:
		{	Pgen g = (Pgen)f;
			fct_name = fn = g->fct_list->f;
			f = (Pfct)fn->tp;
		}
		}
	}
```

```
        if (f && curr_expr==this) {     /* check for class object retuening fct */
                Pname cln = f->returns->is_cl_obj();
                if (cln && Pclass(cln->tp)->has_dtor()) error('s',"%n returned by%n
        }

        switch (e1->base) {
        case DOT:
        case REF:
        {       Pref r = (Pref)e1;
                Pexpr a1 = r->e1;
/*fprintf(stderr,"fn %s f %d fv %d\n",fn?fn->string:"?",f,f?f->f_virtual:0);*/
                if (f && f->f_virtual) {
                        Pexpr a11 = 0;

                        switch(a1->base) {      /* see if temporary might be
                                                   needed/avoided
                                                */
                /*      case ASPLUS:
                        case ASMINUS:
                        case ASMUL:
                        case ASDIV:
                        case ASMOD:
                        case ASAND:
                        case ASOR:
                        case ASER:
                        case ASLS:
                        case ASRS:
                        case ASSIGN:
                                a11 = a1->e1;
                                break;*/
                        case NAME:
                                a11 = a1;
                                break;
                        case ADDROF:
                        case G_ADDROF:
                                if (a1->e2->base == NAME) a11 = a1;
                                break;
                /*      case CM:
                        {       Pexpr ee = a1;
                        cm1:
                                switch (ee->e2->base) {
                                case NAME:
                                        a11 = ee->e2;
                                        break;
                                case ADDROF:
                                case G_ADDROF:
                                        if (ee->e2->e2->base == NAME) a11 = ee->e2;
                                        break;
                                case CM:
                                        ee = ee->e2;
                                        goto cm1;
                                }
                        }*/
                        }
/*error('d',"expression(%k)%k%n: %d",r->e1->base,e1->base,fct_name,a11);*/
                        if (e1->base == DOT) {
                                if (a11) a11 = a11->address();
                                a1 = a1->address();

                        }

                        if (a11 == 0) {
                                /* temporary (maybe) needed
                                   e->f() => (t=e,t->f(t))
                                */
                                char* s = make_name('K');
                                Pname n = new name(s);
                                n->tp = a1->tp;
                                n = n->dcl(scope,ARG); /* no init! */
                                n->n_scope = FCT;
                                n->assign();
                                a11 = n;
                                a1 = new expr(ASSIGN,n,a1);
                                a1->tp = n->tp;
                                a1->simpl();
                                Pcall cc = new call(0,0);
                                *cc = *this;
                                base = CM;
                                e1 = a1;
                                e2 = cc;
                                this = cc;
                        }
                        e2 = new expr(ELIST,a11,e2);
                        int index = f->f_virtual;
                        Pexpr ie = (1<index) ? new expr(IVAL,(Pexpr)(index-1),0) :
                        Pname vp = fn->n_table->look("_vptr",0);
                        Pexpr vptr = new ref(REF,a11,vp);       /* p->vptr */
                        Pexpr ee = new expr(DEREF,vptr,ie);     /* p->vptr[i] */
                        Ptype pft = new ptr(PTR,f);
                        ee = new texpr(CAST,pft,ee);            /* (T)p->vptr[i] */
                        ee->tp = (Ptype)f->f_this;              /* encode argtype *
                        e1 = new expr(DEREF,ee,0);              /* *(T)p->vptr[i] *
                                                                /* e1->tp must be 0
                        fct_name = 0;
                        fn = 0;
                        e2->simpl();
                        return;                         /* (*(T)p->vptr[i])(e2) */
                }
                else {
                        if (e1->base == DOT) a1 = a1->address();
                        e2 = new expr(ELIST,a1,e2);
                        e1 = r->mem;
                }
        }
        }

        e2->simpl();
        if (e1->base==NAME && e1->tp->base==FCT) {
                /* reconstitute fn destroyed to suppress "virtual" */
                fct_name = fn = (Pname)e1;
                f = (Pfct)fn->tp;
        }
        if (fn && f->f_inline && debug==0) {
                Pexpr ee = f->expand(fn,scope,e2);
                if (ee) *((Pexpr)this) = *ee;
        }
}

Pexpr curr_expr;        /* to protect against an inline being expanded twice
                           in a simple expression keep track of expressions
                           being simplified
                        */

Pstmt stmt.simpl()
/*
        return a pointer to the last statement in the list, or 0
*/
{
        if (this == 0) error('i',"0->stmt.simpl()");
/*error('d',"stmt.simpl %d%k e %d%k s %d%k sl %d%k\n",this,base,e,e?e->base:0,s,s?s-

        curr_expr = e;

        switch (base) {
        default:
                error('i',"stmt.simpl(%k)",base);

        case ASM:
                break;

        case BREAK:
        case CONTINUE:
                if (block_del_list) {
                /*      break           =>      { _dtor()s; break; }
                        continue        =>      { _dtor()s; continue; }
                */
                        Pstmt bs = new stmt(base,where,0);
                        Pstmt dl = block_del_list->copy();
                        base = BLOCK;
                        s = new pair(where,dl,bs);
                        break;
                }
                break;

        case DEFAULT:
                s->simpl();
                break;

        case SM:
                if (e) e->simpl();
                break;

        case RETURN:
        {       /*      return x;       =>
                                { _ret_var = x; _dtor()s;  return _ret_var; }
                        return ctor(x); =>
                                { ctor(&_result,x); _dtor()s;  return _ret_var; }
```

```
				return;		=>
					{ _dtor()s; return; } OR (in constructors)
					{ _dtor()s; return _this; }
			*/

			no_of_returns++;

			if (not_inl) {
				Pstmt as;
				if (e && e!=dummy) {
					Pexpr ee;
					if (e->base==G_CALL
					&& e->fct_name
					&& e->fct_name->n_oper==CTOR
					&& e->e1->base==DOT ) {
						Pref r = (Pref)e->e1;
						r->e1 = ret_var;
						ee = e;
					}
					else {
						ee = new expr(ASSIGN,ret_var,e);
					}
					ee->simpl();
					as = new estmt(SM,where,ee,0);
				}
				else
					as = 0;

				base = BLOCK;
				s = 0;
				d = 0;
				own_tbl = (memtbl) ? 1 : 0;
				((Pblock)this)->simpl();

				Pstmt dl = (del_list) ? del_list->copy() : 0;
				if (s) dl = (dl) ? new pair(where,s,dl) : s;

				Pstmt rs = new estmt(RETURN,where,(ret_var)?(Pexpr)ret_var:
				if (as) {
					if (dl) as = new pair(where,as,dl);
					s = new pair(where,as,rs);
				}
				else {
					if (curr_fct->n_oper == CTOR) {
						rs->e = ((Pfct)(curr_fct->tp))->f_this;
					}
					s = (dl) ? new pair(where,dl,rs) : rs;
				}
			}
			else {
				if (e->base == VALUE) error('s',"inlineF returns constructo
				e->simpl();
			}
			break;
		}
```

```
	case WHILE:
	case DO:
		e->simpl();
		s->simpl();
		break;
	case SWITCH:
		e->simpl();
		s->simpl();
		switch (s->base) {
		case DEFAULT:
		case LABEL:
		case CASE:
			break;
		case BLOCK:
			switch (s->s->base) {
			case BREAK:	/* to cope with the "break; case" macro */
			case CASE:
			case LABEL:
			case DEFAULT:
				break;
			default:
				goto df;
			}
			break;
		default:
		df:
			error('w',&s->where,"statement not reached: case label miss
		}
		break;
	case DELETE:	/* change DELETE node to SM node
			   delete p; => _delete(p);
			            or  cl.delete(p,1);
			 */
	{	Pname cln;
		Pclass cl;
		Pname n;
		Pexpr ee;
		Ptype tt = e->tp;
	ttloop:
		switch (tt->base) {
		case TYPE:	tt = ((Pbase)tt)->b_name->tp; goto ttloop;
		case VEC:
		case PTR:	tt = ((Pptr)tt)->typ; break;
		}

		base = SM;
		cln = tt->is_cl_obj();
		if (cln) cl = (Pclass)cln->tp;
		if ( cln && (n=cl->has_dtor()) ) {	/* e->cl.dtor() */
			Pexpr aa = new expr(ELIST,one,0);
			ee = new ref(REF,e,n);
			e = new call(ee,aa);
			e->fct_name = n;
			e->base = G_CALL;
		}
		else if (cl_obj_vec) {
```

```
                                error('w',"delete vector ofC %n with destructor",cl_obj_vec
                        }
                        else {                                  /* _delete(e) */
                                n = del_fct;
                                ee = new expr(ELIST,e,0);
                                e = new call(n,ee);
                                e->fct_name = n;
                                e->base = G_CALL;
                        }
                        ((Pcall)e)->simpl();
                        break;
                }
        case CASE:
                e->simpl();
                s->simpl();
                break;
        case LABEL:
                if (del_list) error('s',"label in block with destructors");
                s->simpl();
                break;
/*      case GOTO:
                if (del_list) error('s',"goto in block with destructors");
                break;
*/
        case GOTO:
                /* If the goto is going to a different (effective) scope,
                 * then it is necessary to activate all relevant destructors
                 * on the way out of nested scopes, and issue errors if there
                 * are any constructors on the way into the target. */

                /* Only bother if the goto and label have different effective
                 * scopes. (If mem table of goto == mem table of label, then
                 * they're in the same scope for all practical purposes. */

                {
                Pname n = scope->look( d->string, LABEL );
                if (n == 0) error('i',&where,"label%n missing",d);
                if( n->n_realscope != scope ) {

                        /* Find the root of the smallest subtree containing
                         * the path of the goto.  This algorithm is quadratic
                         * only if the goto is to an inner or unrelated scope. */

                        Ptable r = 0;

                        for(Ptable q=n->n_realscope; q!=gtbl; q=q->next) {
                                for( Ptable p = scope; p != gtbl; p = p->next ) {
                                        if( p==q ) {
                                                r = p; /* found root of subtree! */
                                                goto xyzzy;
                                        }
                                }
                        }
xyzzy:                  if( r==0 ) error( 'i',&where,"finding root of subtree" );
```

```
                        /* At this point, r = root of subtree, n->n_realscope
                         * = mem table of label, and scope = mem table of goto. */

                        /* Climb the tree from the label mem table to the table
                         * preceding the root of the subtree, looking for
                         * initializers and ctors.  If the mem table "belongs"
                         * to an unsimplified block(s), the n_initializer field
                         * indicates presence of initializer, otherwise initializer
                         * information is recorded in the init_stat field of
                         * mem table. */

                        for( Ptable p=n->n_realscope; p!=r; p=p->next )
                                if( p->init_stat == 2 ) {
                                        error(&where,"goto%n pastD withIr",d);
                                        goto plugh; /* avoid multiple error msgs */
                                }
                                else if( p->init_stat == 0 ) {
                                        int i;
                                        for(Pname nn=p->get_mem(i=1);nn;nn=p->get_m
                                                if(nn->n_initializer||nn->n_evaluat
                                                        error(&nn->where,"goto%n pa
                                                        goto plugh;
                                                }
                                }
plugh:

                        /* Proceed in a similar manner from the point of the goto,
                         * generating the code to activate dtors before the goto. *
                        /* There is a bug in this code.  If there are class objects
                         * of the same name and type in (of course) different mem
                         * tables on the path to the root of the subtree from the
                         * goto, then the innermost object's dtor will be activated
                         * more than once. */

                        {
                        Pstmt dd = 0, ddt;

                        for( Ptable p=scope; p!=r; p=p->next ) {
                                int i;
                                for(Pname n=p->get_mem(i=1);n;n=p->get_mem(++i)) {
                Pname cln;
                if (n->tp == 0) continue; /* label */

                if ( cln=n->tp->is_cl_obj() ) {
                        Pclass cl = (Pclass)cln->tp;
                        Pname d = cl->has_dtor();

                        if (d) {        /* n->cl.delete(0); */
                                Pref r = new ref(DOT,n,d);
                                Pexpr ee = new expr(ELIST,zero,0);
                                Pcall dl = new call(r,ee);
                                Pstmt dls = new estmt(SM,n->where,dl,0);
                                dl->base = G_CALL;
                                dl->fct_name = d;
                                if (dd)
                                        ddt->s_list = dls;
                                else
                                        dd = dls;
                                ddt = dls;
                        }

                }
                else if (cl_obj_vec) {  /* never "new x" is a pointer */
                        Pclass cl = (Pclass)cl_obj_vec->tp;
                        Pname c = cl->has_ictor();
                        Pname d = cl->has_dtor();

                        if (d) {        /*  _vec_delete(vec,noe,sz,dtor); */
                                Pstmt dls;
                                int esz = cl->tsizeof();
                                Pexpr noe = new expr(IVAL, (Pexpr)(n->tp->tsizeof()
                                Pexpr sz = new expr(IVAL,(Pexpr)esz,0);
                                Pexpr arg = new expr(ELIST,c,0);
                                /*c->take_addr();*/
                                c->lval(ADDROF);
                                arg = new expr(ELIST,sz,arg);
                                arg = new expr(ELIST,noe,arg);
                                arg = new expr(ELIST,n,arg);
                                arg = new call(vec_del_fct,arg);
                                arg->base = G_CALL;
                                arg->fct_name = vec_del_fct;
                                dls = new estmt(SM,n->where,arg,0);
                                if (dd)
                                        ddt->s_list = dls;
                                else
                                        dd = dls;
                                ddt = dls;
                        }
                }
                                } /* end mem table scan */
                        } /* end dtor loop */

                        /* "activate" the list of dtors obtained. */

                        if( dd ) {
                                dd->simpl();
                                Pstmt bs = new stmt( base, where, 0 );
                                *bs = *this;
                                base = PAIR;
                                s = dd;
                                s2 = bs;
                        }
                        }
                } /* end special case for non-local goto */
                }
                break;

        case IF:
                e->simpl();
                s->simpl();
                if (else_stmt) else_stmt->simpl();
                break;
```

```
	case FOR:
		/* "for (s;e;e2) s2; => s; while(e) {s2;e3}" */
		if (for_init) for_init->simpl();
		if (e) e->simpl();
		if (e2) {
			curr_expr = e2;
			e2->simpl();
			if (e2->base==ICALL && e2->tp==void_type)
			   error('s',"call of inline voidF in for-expression");
		}
		s->simpl();
		break;
	case BLOCK:
		((Pblock)this)->simpl();
		break;
	case PAIR:
		break;
	}

	/*if (s) s->simpl();*/
	if (base!=BLOCK && memtbl) {
		int i;
		Pstmt t1 = (s_list) ? s_list->simpl() : 0;
		Pstmt ss = 0;
		Pname cln;
		for (Pname tn = memtbl->get_mem(i=1); tn; tn=memtbl->get_mem(++i))
/*fprintf(stderr,"tmp %s tbl %d\n",tn->string,memtbl);*/
			if ( cln=tn->tp->is_cl_obj() ) {
				Pclass cl = (Pclass)cln->tp;
				Pname d = cl->has_dtor();
				if (d) {	/* n->cl.delete(0); */
					Pref r = new ref(DOT,tn,d);
					Pexpr ee = new expr(ELIST,zero,0);
					Pcall dl = new call(r,ee);
					Pstmt dls = new estmt(SM,tn->where,dl,0);
					dl->base = G_CALL;
					dl->fct_name = d;
					dls->s_list = ss;
					ss = dls;
/*error('d',"%d (tbl=%d): %n.%n %d->%d",this,memtbl,tn,d,ss,ss->s_list);*/
				}
			}
		}
		if (ss) {
			Pstmt t2 = ss->simpl();
			switch (base) {
			case IF:
				Pstmt es = ss->copy();
				if (else_stmt) {
					for (Pstmt t=es; t->s_list; t=t->s_list);
					t->s_list = else_stmt;
				}
				else_stmt = es;
				t2->s_list = s;
				s = ss;
				break;
```

```
                                case RETURN:
                                case WHILE:
                                case FOR:
                                case DO:
                                case SWITCH:
                                case DELETE:
                                        error('s',"E in%kS needs temporary ofC%n with destr
                                        break;
                                default:
                                        if (t1) {
                                                t2->s_list = s_list;
                                                s_list = ss;
                                                return t1;
                                        }
                                        s_list = ss;
                                        return t2;
                                }
                        }
                        return (t1) ? t1 : this;
                }

                return (s_list) ? s_list->simpl() : this;
        }

        Pstmt stmt.copy()
        // now handles dtors in the expression of an IF stmt
        // not general!
        {
                Pstmt ns = new stmt(0,curloc,0);

                *ns = *this;
                if (s) ns->s = s->copy();
                if (s_list) ns->s_list = s_list->copy();

                switch (base) {
                case PAIR:
                        ns->s2 = s2->copy();
                        break;
                }

                return ns;
        }
```

```
/* @(#) size.c 1.2 2/8/85 14:33:46 */
/***********************************************************************

        C++ source for cfront, the C++ compiler front-end
        written in the computer science research center of Bell Labs

        Copyright (c) 1984 AT&T Technologies, Inc.
                All rigths Reserved
        THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF
                AT&T TECHNOLOGIES, INC.

        If you ignore the above notice the gost of Ma Bell will haunt you forever.

size.c:

        initialize alignment and sizeof "constants"

***********************************************************************/

#include <stdio.h>
#include "size.h"
extern int strcmp(char*,char*);
extern int strcpy(char*,char*);
extern int strlen(char*);

int BI_IN_WORD = 32;
int BI_IN_BYTE = 8;

int SZ_CHAR = 1;
int AL_CHAR = 1;

int SZ_SHORT = 2;
int AL_SHORT = 2;

int SZ_INT = 4;
int AL_INT = 4;

int SZ_LONG = 4;
int AL_LONG = 4;

int SZ_FLOAT = 4;
int AL_FLOAT = 4;

int SZ_DOUBLE = 8;
int AL_DOUBLE = 4;

int SZ_STRUCT = 4;      /* minimum struct size */
int AL_STRUCT = 4;

int SZ_FRAME = 4;
int AL_FRAME = 4;

int SZ_WORD = 4;

int SZ_WPTR = 4;
int AL_WPTR = 4;
int SZ_BPTR = 4;
int AL_BPTR = 4;

                                        /*      space at top and bottom of stack frame
                                                (for registers, return ptr, etc.)
                                        */
int SZ_TOP = 0;
int SZ_BOTTOM = 0;

char* LARGEST_INT = "2147483647";       /* 2**31 - 1 */

int arg1;
int arg2;

int get_line(FILE* fp) {
        char s[32];

        if (fscanf(fp," %s %d %d",s,&arg1,&arg2) == EOF) return 0;

        if (strcmp("char",s) == 0) {
                SZ_CHAR = arg1;
                AL_CHAR = arg2;
                return 1;
        }
        if (strcmp("short",s) == 0) {
                SZ_SHORT = arg1;
                AL_SHORT = arg2;
                return 1;
        }
        if (strcmp("int",s) == 0) {
                SZ_INT = arg1;
                AL_INT = arg2;
                if (fscanf(fp," %s",s) == EOF) return 0;
                int ll = strlen(s);
                LARGEST_INT = new char[ll+1];
                strcpy(LARGEST_INT,s);
                return 1;
        }
        if (strcmp("long",s) == 0) {
                SZ_LONG = arg1;
                AL_LONG = arg2;
                return 1;
        }
        if (strcmp("float",s) == 0) {
                SZ_FLOAT = arg1;
                AL_FLOAT = arg2;
                return 1;
        }
        if (strcmp("double",s) == 0) {
                SZ_DOUBLE = arg1;
                AL_DOUBLE = arg2;
                return 1;
        }
        if (strcmp("bit",s) == 0) {
                BI_IN_BYTE = arg1;
```

```
		BI_IN_WORD = arg2;
		return 1;
	}
	if (strcmp("struct",s) == 0) {
		SZ_STRUCT = arg1;
		AL_STRUCT = arg2;
		return 1;
	}
	if (strcmp("frame",s) == 0) {
		SZ_FRAME = arg1;
		AL_FRAME = arg2;
		return 1;
	}
	if (strcmp("word",s) == 0) {
		SZ_WORD = arg1;
		return 1;
	}
	if (strcmp("wptr",s) == 0) {
		SZ_WPTR = arg1;
		AL_WPTR = arg2;
		return 1;
	}
	if (strcmp("bptr",s) == 0) {
		SZ_BPTR = arg1;
		AL_BPTR = arg2;
		return 1;
	}
	if (strcmp("top",s) == 0) {
		SZ_TOP = arg1;
		SZ_BOTTOM = arg2;
		return 1;
	}
	return 0;
}

extern int read_align(char* f)
{
	FILE* fp = fopen(f,"r");
	if (fp == 0) return 1;
	while (get_line(fp)) ;
	return 0;
}

extern print_align(char* s)
{
	fprintf(stderr,"%s sizes and alignments\n\n",s);

	fprintf(stderr,"	size	align\n");
	fprintf(stderr,"char	%d	%d\n",SZ_CHAR,AL_CHAR);
	fprintf(stderr,"short	%d	%d\n",SZ_SHORT,AL_SHORT);
	fprintf(stderr,"int	%d	%d\n",SZ_INT,AL_INT);
	fprintf(stderr,"long	%d	%d\n",SZ_LONG,AL_LONG);
	fprintf(stderr,"float	%d	%d\n",SZ_FLOAT,AL_FLOAT);
	fprintf(stderr,"double	%d	%d\n",SZ_DOUBLE,AL_DOUBLE);
	fprintf(stderr,"bptr	%d	%d\n",SZ_BPTR,AL_BPTR);
	fprintf(stderr,"wptr	%d	%d\n",SZ_WPTR,AL_WPTR);
```

```
        fprintf(stderr,"struct  %d      %d\n",SZ_STRUCT,AL_STRUCT);
        fprintf(stderr,"frame   %d      %d\n",SZ_FRAME,AL_FRAME);
        fprintf(stderr,"large   %s\n\n",LARGEST_INT);

        fprintf(stderr,"%d bits in a byte, %d bits in a word, %d bytes in a word\n"
                        BI_IN_BYTE, BI_IN_WORD, SZ_WORD);
        return 1;
}
```

```
/* %Z% %M% %I% %H% %T% */
/*      used in typ.c type.sizeof() for implementing sizeof */

extern BI_IN_WORD;
extern BI_IN_BYTE;
                                /*      byte sizes */
extern SZ_CHAR;
extern AL_CHAR;

extern SZ_SHORT;
extern AL_SHORT;

extern SZ_INT;
extern AL_INT;

extern SZ_LONG;
extern AL_LONG;

extern SZ_FLOAT;
extern AL_FLOAT;

extern SZ_DOUBLE;
extern AL_DOUBLE;

extern SZ_STRUCT;       /* minimum struct size */
extern AL_STRUCT;

extern SZ_FRAME;
extern AL_FRAME;

extern SZ_WORD;

extern SZ_WPTR;
extern AL_WPTR;

extern SZ_BPTR;
extern AL_BPTR;         /*      space at top and bottom of stack frame
                                        (for registers, return ptr, etc.)
                                */
extern SZ_TOP;
extern SZ_BOTTOM;

extern char* LARGEST_INT;
#if 0
                                /*      byte sizes */
#define SZ_CHAR         1
#define SZ_SHORT        2
#define SZ_INT          4
#define SZ_LONG         4
#define SZ_FLOAT        4
#define SZ_DOUBLE       8

#define SZ_WORD         4
#define SZ_WPTR         4
#define SZ_BPTR         4
                                /*      bit sizes */
```

```
#define BI_IN_WORD      32
#define BI_IN_BYTE      8
                                /*      alignment requirements */
#define AL_CHAR         1
#define AL_SHORT        2
#define AL_INT          4
#define AL_LONG         4
#define AL_FLOAT        4
#define AL_DOUBLE       4
#define AL_PTR          4
#define AL_STRUCT       4
#define AL_FRAME        4
                                /*      space at top and bottom of stack frame
                                        (for registers, return ptr, etc.)
                                */
#define SZ_TOP          0
#define SZ_BOTTOM       0
#endif
                        /*      table sizes */
#define KTBLSIZE        123
#define GTBLSIZE        257
                                /*      initial class table size */
#define CTBLSIZE        12
                                /*      initial block table size */
#define TBLSIZE         20

#define BLMAX           50      /*      max block nesting */
#define TBUFSZ          24*1024 /*      (lex) input buffer size */
#define MAXFILE         30      /*      max include file nesting */

#define MAXERR          20      /* maximum number of errors before terminating */
```

```
/* %Z% %M% %I% %H% %T% */
#include "cfront.h"

char * keys[MAXTOK];
/*
        keys[]  holds the external form for tokens with fixed representation
        illegal tokens and those with variable representation have 0 entries
*/

/*
        the class table functions assume that new initializes store to 0
*/

table.table(short sz, Ptable nx, Pname n)
/*
        create a symbol table with "size" entries
        the scope of table is enclosed in the scope of "nx"

        both the vector of class name pointers and the hash table
        are initialized containing all zeroes

        to simplify hashed lookup entries[0] is never used
        so the size of "entries" must be "size+1" to hold "size" entries
*/
{
        base = TABLE;
        t_name = n;
        size = sz = (sz<=0) ? 2 : sz+1;
        entries = new Pname[sz];
        hashsize = sz = (sz*3)/2;
        hashtbl = new short[sz];
        next = nx;
        free_slot = 1;
/* fprintf(stderr,"table.table %d %s %d\n", this, (n)?n->string:"?", size); fflush(s
}


Pname table.look(char* s, TOK k)
/*
        look for "s" in table, ignore entries which are not of "k" type
        look and insert MUST be the same lookup algorithm
*/
{
        Ptable t;
        register char * p;
        register char * q;
        register int i;
        Pname n;
        int rr;

        if (s == 0) error('i',"%d->look(0)",this);
        if (this == 0) error('i',"0->look(%s)",s);
        if (base != TABLE) error('i',"(%d,%d)->look(%s)",this,base,s);

        /* use simple hashing with linear search for overflow */
        p = s;
        i = 0;
        while (*p) i += (i + *p++); /* i<<1 ^ *p++ better?*/
        rr = (0<=i) ? i : -i;

        for (t=this; t; t=t->next) {
                /* in this and all enclosing scopes look for name "s" */
                Pname* np = t->entries;
                int mx = t->hashsize;
                short* hash = t->hashtbl;
                int firsti = i = rr%mx;

                do {
                        if (hash[i] == 0) goto not_found;
                        n = np[hash[i]];
                        if (n == 0) error('i',"hashed lookup");
                        p = n->string;          /* strcmp(n->n_string,s) */
                        q = s;
                        while (*p && *q)
                                if (*p++ != *q++) goto nxt;
                        if (*p == *q) goto found;
                nxt:
                        if (mx <= ++i) i = 0;           /* wrap around */
                } while (i != firsti);

        found:
                for (; n; n=n->n_tbl_list){     /* for  all name "s"s look for a ke
                        if (n->n_key == k) return n;
                }

        not_found:;
        }

        return 0;       /* not found && no enclosing scope */
}

bit Nold;       /* non-zero if last insert() failed */

Pname table.insert(Pname nx, TOK k)
/*
        the lookup algorithm MUST be the same as look
        if nx is found return the older entry otherwise a copy of nx;
        Nold = (nx found) ? 1 : 0;
*/
{
        register char * p;
        register int i;
        Pname n;
        Pname* np = entries;
        Pname* link;
        int firsti;
        int mx = hashsize;
        short* hash = hashtbl;
        char* s = nx->string;

        if (s==0) error('i',"%d->insert(0,%d)",this,k);
        nx->n_key = k;
        if (nx->n_tbl_list || nx->n_table) error('i',"%n in two tables",nx);
        /* use simple hashing with linear search for overflow */

        p = s;
        i = 0;
        while (*p) i += (i + *p++);
        if (i<0) i = -i;
        firsti = i = i%mx;

        do {    /* look for name "s" */
                if (hash[i] == 0) {
                        hash[i] = free_slot;
                        goto add_np;
                }
                n = np[hash[i]];
                if (n == 0) error('i',"hashed lookup");
                if (strcmp(n->string,s) == 0) goto found;
/*
                p = n->string;
                q = s;
                while (*p && *q) if (*p++ != *q++) goto nxt;
                if (*p == *q) goto found;
        nxt:
*/
                if (mx <= ++i) i = 0;   /* wrap around */
        } while (i != firsti);

        error("N table full");

found:


        forever {
                if (n->n_key == k) { Nold = 1; return n; }

                if (n->n_tbl_list)
                        n = n->n_tbl_list;
                else {
                        link = &(n->n_tbl_list);
                        goto re_allocate;
                }
        }

add_np:
        if (size <= free_slot) {
                grow(2*size);
                return insert(nx,k);
        }

        link = &(np[free_slot++]);

re_allocate:
        {
                Pname nw = new class name(0);
                *nw = *nx;
                {
                        int ll = strlen(s)+1;
                        char *ps = new char[ll];
/*fprintf(stderr,"tbl.cpy %s sz=%d %d->%d\n", s, ll, s, ps); fflush(stderr);*/
                        strcpy(ps,s);   /*      copy string to safer store */
                        Nstr++;
                        nw->string = ps;
                }

                nw->n_table = this;
                *link = nw;
                Nold = 0;
                Nname++;
                return nw;
        }
}

void table.grow(int g)
{
        short* hash;
        register int j;
        int mx;
        register Pname* np;
        Pname n;

        if (g <= free_slot) error('i',"table.grow(%d,%d)",g,free_slot);
        if (g <= size) return;
/* fprintf(stderr,"tbl.grow %d %s %d->%d\n", this, (t_name)?t_name->string:"?", size
        size = mx = g+1;

        np = new Pname[mx];
        for (j=0; j<free_slot; j++) np[j] = entries[j];
        delete entries;
        entries = np;

        delete hashtbl;
        hashsize = mx = (g*3)/2;;
        hash = hashtbl = new short[mx];

        for (j=1; j<free_slot; j++) {   /* rehash(np[j]); */
                char * s = np[j]->string;
                register char * p;
                char * q;
                register int i;
                int firsti;

                p = s;
                i = 0;
                while (*p) i += (i + *p++);
                if (i<0) i = -i;
                firsti = i = i%mx;

                do {    /* look for name "s" */
                        if (hash[i] == 0) {
                                hash[i] = j;
                                goto add_np;
                        }
                        n = np[hash[i]];
                        if (n == 0) error('i',"hashed lookup");
                        p = n->string;  /* strcmp(n->n_string,s) */
                        q = s;
                        while (*p && *q) if (*p++ != *q++) goto nxt;
                        if (*p == *q) goto found;
                nxt:
                        if (mx <= ++i) i = 0;    /* wrap around */
                } while (i != firsti);

                error('i',"rehash??");

        found:
                error('i',"rehash failed");

        add_np:;
        }
}

Pclass Ebase;
Pclass Epriv;   /* extra return values from lookc() */

Pname table.lookc(char* s, TOK)
/*
        like look().

        look and insert MUST be the same lookup algorithm

*/
{
        Ptable t;
        register char * p;
        register char * q;
        register int i;
        Pname n;
        int rr;

        if (s == 0) error('i',"%d->look(0)",this);
        if (this == 0) error('i',"0->look(%s)",s);
        if (base != TABLE) error('i',"(%d,%d)->look(%s)",this,base,s);

        Ebase = 0;
        Epriv = 0;

        /* use simple hashing with linear search for overflow */

        p = s;
        i = 0;
        while (*p) i += (i + *p++);
        rr = (0<=i) ? i : -i;

        for (t=this; t; t=t->next) {
                /* in this and all enclosing scopes look for name "s" */
                Pname* np = t->entries;
                int mx = t->hashsize;
                short* hash = t->hashtbl;
                int firsti = i = rr%mx;
                Pname tname = t->t_name;

                do {
                        if (hash[i] == 0) goto not_found;
                        n = np[hash[i]];
                        if (n == 0) error('i',"hashed lookup");
                        p = n->string;          /* strcmp(n->n_string,s) */
                        q = s;
                        while (*p && *q)
                                if (*p++ != *q++) goto nxt;
                        if (*p == *q) goto found;
                nxt:
                        if (mx <= ++i) i = 0;           /* wrap around */
                } while (i != firsti);

        found:
                if (tname) {
                        if (n->base == PUBLIC)
                                n = n->n_qualifier;
                        else if (n->n_scope == 0)
                                Epriv = (Pclass)tname->tp;
                }
                return n;

        not_found:
                if (tname) {
                        Pclass cl = (Pclass)tname->tp;
                        if (cl && cl->clbase && cl->pubbase==0) Ebase = (Pclass)cl-
                }
        }

        Ebase = Epriv = 0;
        return 0;       /* not found && no enclosing scope */
}


Pname table.get_mem(int i)
/*
        return a pointer to the i'th entry, or 0 if it does not exist
*/
{
        return (i<=0 || free_slot<=i) ? 0 : entries[i];
}

void new_key(char* s, TOK toknum, TOK yyclass)
/*
        make "s" a new keyword with the representation (token) "toknum"
        "yyclass" is the yacc token (for example new_key("int",INT,TYPE); )
        "yyclass==0" means yyclass=toknum;
*/
{
        Pname n = new class name(s);
        Pname nn = ktbl->insert(n,0);
        if (Nold) error("keyword %sD twice",s);
        nn->base = toknum;
        nn->syn_class = (yyclass) ? yyclass : toknum;
        keys[(toknum==LOC)?yyclass:toknum] = s;
        delete n;
}
```

```
/* %Z% %M% %I% %H% %T% */
#include <stdio.h>
extern void lex_clear();
extern void ktbl_init();
extern void otbl_init();

#define yylex() lex()

#define putstring(s)     fputs(s,out_file)
#define putst(ss)        fprintf(out_file,"%s ",ss)
#define putch(c)         putc(c,out_file)

                         /* token numbers for C parser    */

#define MAXTOK 256
extern char* keys[MAXTOK];

#define EOFTOK  0        /*      EOF     */
                         /* keywords in alphabetical order */
#define ASM              1
#define AUTO             2
#define BREAK            3
#define CASE             4
#define CHAR             5
#define CLASS            6
#define CONTINUE         7
#define DEFAULT          8
#define DELETE           9
#define DO               10
#define DOUBLE           11
#define ELSE             12
#define ENUM             13
#define EXTERN           14
#define FLOAT            15
#define FOR              16
#define FORTRAN          17
#define FRIEND           18
#define GOTO             19
#define IF               20
#define INT              21
#define LONG             22
#define NEW              23
#define OPERATOR         24
#define PUBLIC           25
#define CONST            26
#define REGISTER         27
#define RETURN           28
#define SHORT            29
#define SIZEOF           30
#define STATIC           31
#define STRUCT           32
#define SWITCH           33
#define THIS             34
#define TYPEDEF          35
#define UNION            36
#define UNSIGNED         37
```

```
#define VOID              38
#define WHILE             39

                          /* operators in priority order (sort of) */
#define LP                40
#define RP                41
#define LB                42
#define RB                43
#define REF               44
#define DOT               45
#define NOT               46
#define COMPL             47
#define INCR              48
#define DECR              49
#define MUL               50
#define DIV               51
#define AND               52
#define MOD               53
#define PLUS              54
#define MINUS             55
#define LS                56
#define RS                57
#define LT                58
#define LE                59
#define GT                60
#define GE                61
#define EQ                62
#define NE                63
#define ER                64
#define OR                65
#define ANDAND            66
#define OROR              67
#define QUEST             68
#define COLON             69
#define ASSIGN            70
#define CM                71
#define SM                72
#define LC                73
#define RC                74

#define INLINE            75
#define OVERLOAD          76
#define VIRTUAL           77
#define COERCE            78

                          /* constants etc. */
#define ID                80
#define STRING            81
#define ICON              82
#define FCON              83
#define CCON              84
#define NAME              85
#define ZERO              86

                          /* groups of tokens */
#define ASOP              90       /* op= */
```

```
#define RELOP           91      /* LE GE LT GT */
#define EQUOP           92      /* EQ NE */
#define DIVOP           93      /* DIV MOD */
#define SHIFTOP         94      /* LS RS */
#define ICOP            95      /* INCR DECR */
#define UNOP            96      /* NOT COMPL */
#define TYPE            97
        /* TYPE =       INT FLOAT CHAR DOUBLE REGISTER STATIC EXTERN AUTO
                        LONG SHORT UNSIGNED INLINE FRIEND VIRTUAL */

                        /* new tokens generated by syn() */
#define UMINUS          107
#define FCT             108
#define CALL            109
#define VEC             110
#define DEREF           111
#define ADDROF          112
#define CAST            113
#define FIELD           114
#define LABEL           115
#define BLOCK           116
#define QUA             117
#define DCL             118
#define COBJ            119
#define EOBJ            121
#define TNAME           123
#define ILIST           124
#define PTR             125

#define ASPLUS          126
#define ASMINUS         127
#define ASMUL           128
#define ASDIV           129
#define ASMOD           130
#define ASAND           131
#define ASOR            132
#define ASER            133
#define ASLS            134
#define ASRS            135

#define ARG             136
#define KNOWN           137
#define ZTYPE           138
#define ARGT            139
#define ELIST           140
#define ANY             141
#define TABLE           142
#define LOC             143
#define DUMMY           144
#define G_ADDROF        145
#define G_CALL          146
#define IVAL            150
#define FVAL            151
#define LVAL            152
#define ELLIPSIS        155
#define AGGR            156
```

```
#define VALUE           157
#define RPTR            158
#define HIDDEN          159
#define MEM             160
#define CTOR            161
#define DTOR            162
#define CONST_PTR       163
#define CONST_RPTR      164
#define TEXT            165
#define PAIR            166
#define ANON            167
#define ICALL           168
#define ANAME           169

#define A       'A'
#define I       'I'
#define Z       'Z'
#define F       'F'
#define P       'P'
#define C       'C'
#define N       'N'
#define U       'U'
#define S       'S'

#define SYN     1
#define TYP     2
#define SIMPL   3
#define ERROR   4
```

```
/* %Z% %M% %I% %H% %T% */
/************************************************************************

        C++ source for cfront, the C++ compiler front-end
        written in the computer science research center of Bell Labs

        Copyright (c) 1984 AT&T Technologies, Inc. All rigths Reserved
        THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

        If you ignore this notice the ghost of Ma Bell will haunt you forever.

typ.c:

*************************************************************************/

#include "cfront.h"
#include "size.h"

Pbase short_type;
Pbase int_type;
Pbase char_type;
Pbase long_type;

Pbase uchar_type;
Pbase ushort_type;
Pbase uint_type;
Pbase ulong_type;

Pbase zero_type;
Pbase float_type;
Pbase double_type;
Pbase void_type;
Pbase any_type;

Ptype Pint_type;
Ptype Pchar_type;
Ptype Pvoid_type;
Ptype Pfctvec_type;

Ptype char2_type;
Ptype char3_type;
Ptype char4_type;

Ptable gtbl;
Ptable any_tbl;

Pname Cdcl = 0;
Pstmt Cstmt = 0;

bit new_type = 0;

extern Ptype np_promote(TOK, TOK, TOK, Ptype, Ptype, TOK);
Ptype np_promote(TOK oper, TOK r1, TOK r2, Ptype t1, Ptype t2, TOK p)
/*
        an arithmetic operator "oper" is applied to "t1" and "t2",
	types t1 and t2 has been checked and belongs to catagories
	"r1" and "r2", respectively:
		A	ANY
		Z	ZERO
		I	CHAR, SHORT, INT, LONG, FIELD, or EOBJ
		F	FLOAT DOUBLE
		P	PTR (to something) or VEC (of something)
	test for compatability of the operands,
	if (p) return the promoted result type
*/
{
	if (r2 == A) return t1;

	switch (r1) {
	case A: return t2;
	case Z:
		switch (r2) {
		case Z:		return int_type;
		case I:
		case F:		return (p) ? ((Pbase)t2)->arit_conv(0) : 0;
		case P:		return t2;
		default:	error('i',"zero(%d)",r2);
		}
	case I:
		switch (r2) {
		case Z: t2 = 0;
		case I:
		case F: return (p) ? ((Pbase)t1)->arit_conv((Pbase)t2) : 0;
		case P: switch (oper) {
			case PLUS:
			case ASPLUS:	break;
			default:	error("int%kP",oper); return any_type;
			}
			return t2;
		default: error('i',"int(%d)",r2);
		}
	case F:
		switch (r2) {
		case Z: t2 = 0;
		case I:
		case F: return (p) ? ((Pbase)t1)->arit_conv((Pbase)t2) : 0;
		case P: error("float%kP",oper); return any_type;
		default: error('i',"float(%d)",r2);
		}
	case P:
		switch (r2) {
		case Z: return t1;
		case I:
			switch (oper) {
			case PLUS:
			case MINUS:
			case ASPLUS:
			case ASMINUS: break;
			default: error("P%k int",oper); return any_type;
			}
			return t1;
```

```
				case F: error("P%k float",oper); return any_type;
				case P:
					if (t1->check(t2,ASSIGN)) {
						switch (oper) {
						case EQ:
						case NE:
						case LE:
						case GE:
						case GT:
						case LT:
						case QUEST:
							if (t2->check(t1,ASSIGN) == 0) goto zz;
						}
						error("T mismatch:%t %k%t",t1,oper,t2);
						return any_type;
					}
					zz:
					switch (oper) {
					case MINUS:
					case ASMINUS:	return int_type;
					case PLUS:
					case ASPLUS:	error("P +P"); return any_type;
					default:	return t1;
					}
				case FCT:	return t1;
				default:	error('i',"pointer(%d)",r2);
				}
		case FCT:
				error("F%k%t",oper,t2);
				return any_type;
		default:
				error('i',"np_promote(%d,%d)",r1,r2);
		}
}

TOK type.kind(TOK oper, TOK v)
/*	v ==	I	integral
		N	numeric
		P	numeric or pointer
*/
{
	Ptype t = this;
	char* s = (oper) ? keys[oper] : 0;
xx:
	switch (t->base) {
	case ANY:	return A;
	case ZTYPE:	return Z;
	case FIELD:
	case CHAR:
	case SHORT:
	case INT:
	case LONG:
	case EOBJ:	return I;
	case FLOAT:
	case DOUBLE:	if (v == I) error("float operand for %s",s);	return F;
	case PTR:	if (v != P) error("P operand for %s",s);
```

```
                        switch (oper) {
                        case INCR:
                        case DECR:
                        case MINUS:
                        case PLUS:
                        case ASMINUS:
                        case ASPLUS:
                                Pptr(t)->typ->tsizeof();        /* get increment */
                        }
                        return P;
        case RPTR:      //if (v != P) error("P operand for %s",s);
                        //if (oper != ASSIGN) error("reference operand for %s",s);
                        //return P;
                         error("reference operand for %s",s);
                        return A;
        case VEC:       if (v != P) error("V operand for %s",s);        return P;
        case TYPE:      t = ((Pbase)t)->b_name->tp;                     goto xx;
        case FCT:       if (v != P) error("F operand for %s",s);        return FCT;
        default:        error("%t operand for %s",this,s);      return A;
        }
}

void type.dcl(Ptable tbl)
/*
        go through the type (list) and
        (1) evaluate vector dimentions
        (2) evaluate field sizes
        (3) lookup struct tags, etc.
        (4) handle implicit tag declarations
*/
{
        Ptype t = this;

        if (this == 0) error('i',"type.dcl(this==0)");
        if (tbl->base != TABLE) error('i',"type.dcl(%d)",tbl->base);

xx:
        switch (t->base) {
        case PTR:
        case RPTR:
        {       Pptr p = (Pptr)t;
                t = p->typ;
                goto xx;
        }

        case VEC:
        {       Pvec v = (Pvec)t;
                Pexpr e = v->dim;
                if (e) {
                        Ptype et;
                        v->dim = e = e->typ(tbl);
                        et = e->tp;
                        if (et->integral(0) == A)  {
                                error("UN in array dimension");
                        }
                        else if (!new_type) {
                                int i;
                                Neval = 0;
                                i = e->eval();
                                if (Neval) error("%s",Neval);
                                else if (i == 0)
                                        error('w',"array dimension == 0");
                                else if (i < 0) {
                                        error("negative array dimension");
                                        i = 1;
                                }
                                v->size = i;
                                DEL(v->dim);
                                v->dim = 0;
                        }
                }
                t = v->typ;
                goto xx;
        }

        case FCT:
        {       Pfct f = (Pfct)t;
                Pname n;
                for (n=f->argtype; n; n = n->n_list) n->tp->dcl(tbl);
                t = f->returns;
                goto xx;
        }

        case FIELD:
        {       Pbase f = (Pbase)t;
                Pexpr e = (Pexpr)f->b_name;
                int i;
                Ptype et;
                e = e->typ(tbl);
                f->b_name = (Pname)e;
                et = e->tp;
                if (et->integral(0) == A) {
                        error("UN in field size");
                        i = 1;
                }
                else {
                        Neval = 0;
                        i = e->eval();
                        if (Neval)
                                error("%s",Neval);
                        else if (i < 0) {
                                error("negative field size");
                                i = 1;
                        }
                        else if (SZ_INT*BI_IN_BYTE < i)
                                error("field size > sizeof(int)");
                        DEL(e);
                }
                f->b_bits = i;
                f->b_name = 0;
                break;
        }
        }
}

bit vrp_equiv;  /* vector == reference == pointer equivalence used in check() */

bit type.check(Ptype t,TOK oper)
/*
        check if "this" can be combined with "t" by the operator "oper"

        used for check of
                        assignment types                (oper==ASSIGN)
                        declaration compatability       (oper==0)
                        argument types                  (oper==ARG)
                        return types                    (oper==RETURN)
                        overloaded function  name match (oper==OVERLOAD)
                        overloaded function coercion    (oper==COERCE)

        NOT for arithmetic operators

        return 1 if the check failed
*/
{
        Ptype t1 = this;
        Ptype t2 = t;
        TOK b1, b2;
        bit first = 1;
        TOK r;

        if (t1==0 || t2==0) error('i',"check(%d,%d,%d)",t1,t2,oper);

        vrp_equiv = 0;

        while (t1 && t2) {
        top:
/*fprintf(stderr,"top: %d %d\n",t1->base,t2->base);*/
                if (t1 == t2) return 0;
                if (t1->base == ANY || t2->base == ANY) return 0;

                b1 = t1->base;
                b2 = t2->base;
                if (b1 !=  b2) {
                        if (b1 == TYPE) {
                                t1 = ((Pbase)t1)->b_name->tp;
                                goto top;
                        }
                        if (b2 == TYPE) {
                                t2 = ((Pbase)t2)->b_name->tp;
                                goto top;
                        }

                        switch (b1) {
                        case PTR:
                //      case RPTR:
                                if (b1 != b2) vrp_equiv = 1;
                                switch (b2) {
```

```
                        case PTR:
//                      case RPTR:
                        case VEC:
                                t1 = ((Pptr)t1)->typ;
                                t2 = ((Pvec)t2)->typ;
                                first = 0;
                                goto top;
                        case FCT:
                                t1 = ((Pptr)t1)->typ;
                                if (first==0 || t1->base!=b2) return 1;
                                first = 0;
                                goto top;
                        }
                        first = 0;
                        break;
                case VEC:
                        if (b1 != b2) vrp_equiv = 1;
                        first = 0;
                        switch (b2) {
                        case PTR:
//                      case RPTR:
                                switch(oper) {
                                case 0:
                                case ARG:
                                case ASSIGN:
                                case COERCE:
                                        break;
                                case OVERLOAD:
                                default:
                                        return 1;
                                }
                                t1 = ((Pvec)t1)->typ;
                                t2 = ((Pptr)t2)->typ;
                                goto top;
                        }
                        break;
                case TYPE:
                        t1 = ((Pbase)t1)->b_name->tp;
                        goto top;
                }
                goto base_check;
        }

        switch (b1) {
        case VEC:
                first = 0;
        {       Pvec v1 = (Pvec)t1;
                Pvec v2 = (Pvec)t2;
                if (v1->size != v2->size)
                        switch (oper) {
                        case OVERLOAD:
                        case COERCE:
                                return 1;
                        }
                t1 = v1->typ;
                t2 = v2->typ;
```

```
		}
			break;

		case PTR:
		case RPTR:
			first = 0;
		{	Pptr p1 = (Pptr)t1;
			Pptr p2 = (Pptr)t2;
			if (p2->rdo && p1->rdo==0) return 1;
			t1 = p1->typ;
			t2 = p2->typ;
		}
			break;

		case FCT:
			first = 0;
		{	Pfct f1 = (Pfct)t1;
			Pfct f2 = (Pfct)t2;
			Pname a1 = f1->argtype;
			Pname a2 = f2->argtype;
			TOK k1 = f1->nargs_known;
			TOK k2 = f2->nargs_known;
			int n1 = f1->nargs;
			int n2 = f2->nargs;
/*error('d',"k %d %d n %d %d body %d %d",k1,k2,n1,n2,f1->body,f2->body);*/
			if ( (k1 && k2==0) || (k2 && k1==0) ){
				if (f2->body == 0) return 1;
			}

			if (n1!=n2 && k1 && k2) {
				goto aaa;
			}
			else if (a1 && a2) {
				int i = 0;
				while (a1 && a2) {
					i++;
					if ( a1->tp->check(a2->tp,oper?OVERLOAD:0)
					a1 = a1->n_list;
					a2 = a2->n_list;
				}
				if (a1 || a2) goto aaa;
			}
			else if (a1 || a2) {
			aaa:
				if (k1 == ELLIPSIS) {
					switch (oper) {
					case 0:
						if (a2 && k2==0) break;
						return 1;
					case ASSIGN:
						if (a2 && k2==0) break;
						return 1;
					case ARG:
						if (a1) return 1;
						break;
					case OVERLOAD:
```

```
                                        case COERCE:
                                                return 1;
                                        }
                                }
                                else if (k2 == ELLIPSIS) {
                                        return 1;
                                }
                                else if (k1 || k2) {
                                        return 1;
                                }
                        }
                        t1 = f1->returns;
                        t2 = f2->returns;
                }
                        break;

                case FIELD:
                        goto field_check;
                case CHAR:
                case SHORT:
                case INT:
                case LONG:
                        goto int_check;
                case FLOAT:
                case DOUBLE:
                        goto float_check;
                case EOBJ:
                        goto enum_check;
                case COBJ:
                        goto cla_check;
                case ZTYPE:
                case VOID:
                        return 0;

                case TYPE:
                        t1 = ((Pbase)t1)->b_name->tp;
                        t2 = ((Pbase)t2)->b_name->tp;
                        break;

                default:
                        error('i',"type.check(o=%d %d %d)",oper,b1,b2);
                }
        }

        if (t1 || t2) return 1;
        return 0;


field_check:
        switch (oper) {
        case 0:
        case ARG:
                error('i',"check field?");
        }
        return 0;
```

```
float_check:
        if (first==0) {
                if (b1!=b2 && b2!=ZTYPE) return 1;
        }
        goto const_check;

enum_check:
int_check:
const_check:
        if (first==0 && t2->tconst() && t1->tconst()==0) return 1;
        return 0;

cla_check:
        {       Pbase c1 = (Pbase)t1;
                Pbase c2 = (Pbase)t2;
                Pname n1 = c1->b_name;
                Pname n2 = c2->b_name;
/*fprintf(stderr,"c1 %d c2 %d n1 %d %s n2 %d %s oper %d\n",c1,c2,n1,n1->string,n2,n2
                if (n1 == n2) goto const_check;

                switch (oper) {
                case 0:
                case OVERLOAD:
                        return 1;
                case ARG:
                case ASSIGN:
                case RETURN:
                case COERCE:
                {
                        /*      is c2 derived from c1 ? */
                        Pname b = n2;
                        Pclass cl;
                        while (b) {
                                cl = (Pclass) b->tp;
                                b = cl->clbase;
/*if (b)fprintf(stderr,"n2=(%d %s) b=(%d %s) n1=(%d %s) pub %d\n",n2,n2->string,b,b-
                                if (b && cl->pubbase==0) {
                                        return 1;
                                }
                                if (b == n1) goto const_check;
                        }
                        return 1;
                }
                }
        }
        goto const_check;

base_check:
/*error('d',"base_check t1=%t t2=%t oper=%d",t1,t2,oper);*/
        if (oper)
        if (first) {
                if (b1==VOID || b2==VOID) return 1;
        }
        else {
                if (b1==VOID || b2==VOID) {     /* check for void* */
```

```
                        register Ptype tx = this;
                txloop:
                        switch (tx->base) {
                        default:        return 1;
                        case VOID:      break;
                        case PTR:
                //      case RPTR:      tx = ((Pptr)tx)->typ; goto txloop;
                        case VEC:       tx = ((Pvec)tx)->typ; goto txloop;
                        case TYPE:      tx = ((Pbase)tx)->b_name->tp; goto txloop;
                        }

                        tx = b1==VOID ? t2 : t1;
                bloop:
                        switch (tx->base) {
                        default:        return 0;
                        case VEC:
                        case PTR:
                //      case RPTR:
                        case FCT:       return 1;
                        case TYPE:      tx = ((Pbase)tx)->b_name->tp; goto bloop;
                        }
                }
                if (b2 != ZTYPE) return 1;
        }

        switch (oper) {
        case 0:
                return 1;
        case OVERLOAD:
        case COERCE:
                switch (b1) {
                case EOBJ:
                case ZTYPE:
                case CHAR:
                case SHORT:
                case INT:
                        switch (b2) {
                        case EOBJ:
                        case ZTYPE:
                        case CHAR:
                        case SHORT:
                        case INT:
                        case FIELD:
                                goto const_check;
                        }
                        return 1;
                case LONG:      /* char, short, and int promotes to long */
                        switch (b2) {
                        case ZTYPE:
                        case EOBJ:
                        case CHAR:
                        case SHORT:
                        case INT:
                        case FIELD:
                                goto const_check;
                        }
```

```
				return 1;
			case FLOAT:
				switch (b2) {
				case FLOAT:
				case DOUBLE:
				case ZTYPE:
					goto const_check;
				}
				return 1;
			case DOUBLE:	/* char, short, int, and float promotes to double *
				switch (b2) {
				case FLOAT:
				case DOUBLE:
				case ZTYPE:
				case EOBJ:
				case CHAR:
				case SHORT:
				case INT:
					goto const_check;
				}
				return 1;
			case PTR:
				switch (b2) {
				case ZTYPE:
					goto const_check;
				}
			case RPTR:
			case VEC:
			case COBJ:
			case FCT:
				return 1;
			}
		case ARG:
		case ASSIGN:
		case RETURN:
			switch (b1) {
			case COBJ:
				return 1;
			case EOBJ:
			case ZTYPE:
			case CHAR:
			case SHORT:
			case INT:
			case LONG:
				r = t2->num_ptr(ASSIGN);
				switch (r) {
				case A: return 1;
				case Z:
				case I: break;
				case F: error('w',"double assigned to int"); break;
				case P: return 1;
				}
				break;
			case FLOAT:
			case DOUBLE:
				r = t2->numeric(ASSIGN);
				break;
			case VEC:
				return 1;
			case PTR:
				r = t2->num_ptr(ASSIGN);
				switch (r) {
				case A: return 1;
				case Z:
				case P: break;
				case I:
				case F: return 1;
				case FCT:
				{	Pptr p = (Pptr)t1;
					if (p->typ->base != FCT) return 1;
				}
				}
				break;
			case RPTR:
			//	r = t2->num_ptr(ASSIGN);
			//	switch (r) {
			//	case A: break;
			//	case Z: return 1;
			//	case P:
			//	case I:
			//	case F: break;
			//	case FCT:
			//	{	Pptr p = (Pptr)t1;
			//		if (p->typ->base != FCT) return 1;
			//	}
			//	}
			//	break;
				return 1;
			case FCT:
				switch (oper) {
				case ARG:
				case ASSIGN:
					return 1;
				}
			}
			break;
		}
		goto const_check;
}
```

```
/* %Z% %M% %I% %H% %T% */
/*******************************************************************************

        C++ source for cfront, the C++ compiler front-end
        written in the computer science research center of Bell Labs

        Copyright (c) 1984 AT&T Technologies, Inc. All rigths Reserved
        THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T TECHNOLOGIES, INC.

        If you ignore this notice the ghost of Ma Bell will haunt you forever.

typ2.c:

*******************************************************************************/

#include "cfront.h"
#include "size.h"

extern void typ_init();
void typ_init()
{
        defa_type = int_type = new class basetype(INT,0);
        PERM(int_type);

        moe_type = new class basetype(INT,0);
        PERM(moe_type);
        moe_type->b_const = 1;
        moe_type->check(0);

        uint_type = new class basetype(INT,0);
        PERM(uint_type);
        uint_type->type_adj(UNSIGNED);
        uint_type->check(0);

        long_type = new class basetype(LONG,0);
        PERM(long_type);
        long_type->check(0);

        ulong_type = new class basetype(LONG,0);
        PERM(ulong_type);
        ulong_type->type_adj(UNSIGNED);
        ulong_type->check(0);

        short_type = new class basetype(SHORT,0);
        PERM(short_type);
        short_type->check(0);

        ushort_type = new class basetype(SHORT,0);
        PERM(ushort_type);
        ushort_type->type_adj(UNSIGNED);
        ushort_type->check(0);

        float_type = new class basetype(FLOAT,0);
        PERM(float_type);
        double_type = new class basetype(DOUBLE,0);
        PERM(double_type);

        zero_type = new class basetype(ZTYPE,0);
        PERM(zero_type);
        zero->tp = zero_type;

        void_type = new class basetype(VOID,0);
        PERM(void_type);

        char_type = new class basetype(CHAR,0);
        PERM(char_type);

        uchar_type = new class basetype(CHAR,0);
        PERM(uchar_type);
        uchar_type->type_adj(UNSIGNED);
        uchar_type->check(0);

        Pchar_type = new class ptr(PTR,char_type,0);
        PERM(Pchar_type);

        Pint_type = new class ptr(PTR,int_type,0);
        PERM(Pint_type);

        Pvoid_type = new class ptr(PTR,void_type,0);
        PERM(Pvoid_type);

        Pfctvec_type = new class fct(int_type,0,0);             /* must be last, se
        Pfctvec_type = new class ptr(PTR,Pfctvec_type,0);
        Pfctvec_type = new class ptr(PTR,Pfctvec_type,0);
        PERM(Pfctvec_type);

        any_tbl = new class table(TBLSIZE,0,0);
        gtbl = new class table(GTBLSIZE,0,0);
        gtbl->t_name = new class name("global");

        if (SZ_SHORT == 2)
                char2_type = short_type;
        else {
                char2_type = new vec(char_type,0);
                PERM(char2_type);
                Pvec(char2_type)->size = 2;
        }
        char3_type = new vec(char_type,0);
        PERM(char3_type);
        Pvec(char3_type)->size = 3;
        if (SZ_INT == 4)
                char4_type = int_type;
        else if (SZ_LONG == 4)
                char4_type = long_type;
        else {
                char4_type = new vec(char_type,0);
                PERM(char4_type);
                Pvec(char4_type)->size = 4;
        }
}
```

```
Pbase basetype.arit_conv(Pbase t)
/*
        perform the "usual arithmetic conversions" C ref Manual 6.6
        on "this" op "t"
        "this" and "t" are integral or floating
        "t" may be 0
*/
{
        bit l;
        bit u;
        bit f;
        bit l1 = (base == LONG);
        bit u1 = b_unsigned;
        bit f1 = (base==FLOAT || base==DOUBLE);
        if (t) {
                bit l2 = (t->base == LONG);
                bit u2 = t->b_unsigned;
                bit f2 = (t->base==FLOAT || t->base==DOUBLE);
                l = l1 || l2;
                u = u1 || u2;
                f = f1 || f2;
        }
        else {
                l = l1;
                u = u1;
                f = f1;
        }

        if (f)          return double_type;
        if (l & u)      return ulong_type;
        if (l & !u)     return long_type;
        if (u)          return uint_type;
                        return int_type;
}

bit vec_const = 0;

bit type.tconst()
/*
        is this type a constant
*/
{
        Ptype t = this;
        vec_const = 0;
xxx:
        switch (t->base) {
        case TYPE:      if ( ((Pbase)t)->b_const ) return 1;
                        t = ((Pbase)t)->b_name->tp;
                        goto xxx;
        case VEC:       vec_const = 1; return 1; /*t = ((Pvec)t)->typ; goto xxx;*/
        case PTR:
        case RPTR:      return ((Pptr)t)->rdo;
        case ANY:       return 0;
        default:        return ((Pbase)t)->b_const;
        }
```

```
}

int type.align()
{
        Ptype t = this;
xx:
/*fprintf(stderr,"align %d %d\n",t,t->base);*/
        switch (t->base) {
        case TYPE:      t = ((Pbase)t)->b_name->tp; goto xx;
        case COBJ:      t = ((Pbase)t)->b_name->tp; goto xx;
        case VEC:       t = ((Pvec)t)->typ; goto xx;
        case ANY:       return 1;
        case CHAR:      return AL_CHAR;
        case SHORT:     return AL_SHORT;
        case INT:       return AL_INT;
        case LONG:      return AL_LONG;
        case FLOAT:     return AL_FLOAT;
        case DOUBLE:    return AL_DOUBLE;
        case PTR:
        case RPTR:      return AL_WPTR;
        case CLASS:     return ((Pclass)t)->obj_align;
        case ENUM:
        case EOBJ:      return AL_INT;
        case VOID:      error("illegal use of void"); return AL_INT;
        default:        error('i',"(%d,%k)->type.align",t,t->base);
        }
}

int type.tsizeof()
/*
        the sizeof type operator
        return the size in bytes of the types representation
*/
{
        Ptype t = this;
zx:
        if (t == 0) error('i',"typ.tsizeof(t==0)");
        switch (t->base) {
        case TYPE:
        case COBJ:
/*fprintf(stderr,"tsizeof %d %d %s%s\n",t,t->base,((Pbase)t)->b_name->string,(t->per
                t = ((Pbase)t)->b_name->tp; goto zx;
        case ANY:       return 1;
        case VOID:      return 0;
        case ZTYPE:     return SZ_WPTR; /* assume pointer */
        case CHAR:      return SZ_CHAR;
        case SHORT:     return SZ_SHORT;
        case INT:       return SZ_INT;
        case LONG:      return SZ_LONG;
        case FLOAT:     return SZ_FLOAT;
        case DOUBLE:    return SZ_DOUBLE;
        case VEC:
                {       Pvec v = (Pvec) t;
                        if (v->size == 0) return SZ_WPTR;
                        return v->size * v->typ->tsizeof();
                }
```

```
	case PTR:
	case RPTR:
		t = ((Pptr)t)->typ;
	xxx:
		switch (t->base) {
		default:	return SZ_WPTR;
		case CHAR:	return SZ_BPTR;
		case TYPE:	t = ((Pbase)t)->b_name->tp; goto xxx;
		}
	case FIELD:
	{	Pbase b = (Pbase)t;
		return b->b_bits/BI_IN_BYTE+1;
	}
	case CLASS:
	{	Pclass cl = (Pclass)t;
		int sz = cl->obj_size;
		if (cl->defined == 0) {
			error("%sU, size not known",cl->string);
			return SZ_INT;
		}
		return sz;
	}
	case EOBJ:
	case ENUM:	return SZ_INT;
	default:	error('i',"sizeof(%d)",t->base);
	}
}

bit type.fct_type()
{
	return 0;
}

bit type.vec_type()
{
	Ptype t = this;
xx:
	switch (t->base) {
	case ANY:
	case VEC:
	case PTR:
	case RPTR:	return 0;
	case TYPE:	t = ((Pbase)t)->b_name->tp; goto xx;
	default:	error("not a vector(%k)",base); return 1;
	}
}

Ptype type.deref()
{
	Ptype t = this;
xx:
	switch (t->base) {
	case PTR:
	case RPTR:
	case VEC:	return ((Pptr)t)->typ;
	case ANY:	return t;
	case TYPE:	t = ((Pbase)t)->b_name->tp; goto xx;
	default:	error("nonP dereferenced"); return any_type;
	}
}

Pptr type.addrof()
{
	return new class ptr(PTR,this,0);
}
```

```
/* %Z% %M% %I% %H% %T% */
typedef short TOK;
typedef class node * PP;
typedef char bit;
typedef int (*PFI)();
typedef void (*PFV)();
typedef class node * Pnode;
typedef struct key * Pkey;
typedef class name * Pname;
typedef class basetype * Pbase;
typedef class type * Ptype;
typedef class fct  * Pfct;
typedef class field * Pfield;
typedef class expr * Pexpr;
typedef class qexpr * Pqexpr;
typedef class texpr * Ptexpr;
typedef class classdef * Pclass;
typedef class enumdef * Penum;
typedef class stmt * Pstmt;
typedef class estmt * Pestmt;
typedef class tstmt * Ptstmt;
typedef class vec * Pvec;
typedef class ptr * Pptr;
typedef class block * Pblock;
typedef class table * Ptable;
typedef class loc Loc;
typedef class call * Pcall;
typedef class gen* Pgen;
typedef class ref * Pref;
typedef class name_list * Plist;
typedef class iline * Pin;

#define forever for(;;)
```

```
/* %Z% %M% %I% %H% %T% */

char    1       1
short   2       2
int     2       2         32767
long    4       4
double  8       4
bit     8       32
ptr     4       4
struct  1       1
frame   4       0
top     0       0
word    4       0
wptr    4       4
bptr    4       4
```