Clark Weissman

8-20K/65

((Primer for LISP 2 by Mike Levin and Ed Berkeley))

Draft, Sept. 20, 1965

PREFACE

The purpose of this primer for LISP 2 is to give a fairly simple and understandable explanation of several important parts of LISP 2, including the way in which information is given to the system (called Source Language) and the way in which the system operates with the information (called the Intermediate Language).

We assume that readers of this primer have had no previous familiarity with LISP.

We also assume that readers have some familiarity with mathematics including the binary and octal scales of notation, the algebra of the real number system and Boolean algebra. However such information is needed only seldom.

We try to explain each idea stated here in such a way that the explanation is all contained in this primer. However, every now and then an idea occurs in the primer marked "this will be explained later" or "this will be mentioned but not explained in the primer".

We invite suggestions, comments, and criticisms of this draft from every reader.

((Primer for LISP 2 by Mike Levin and Ed Berkeley))

Draft, Sept. 20, 1965

Chapter 1. INTRODUCTION

1. Three Basic Ideas

The first two ideas needed for the computer programming language LISP 2 are:

the Source Language

the Intermediate Language

The Source Language is a language which is relatively easy and natural for a programmer to learn, write, and use to express problems, and which is acceptable by a device (a set of rules or a computer program) called the Syntax Translator which produces Intermediate Language.

The Intermediate Language is a language which is much like LISP 1.5, which is acceptable to one or more computers and can be implemented on them, and which enables a computer to solve the problems that a programmer has expressed.

The Syntax Translator may be a computer-implemented system or it may be a set of rules which a programmer can manually apply or be guided by. A programmer can of course write in Intermediate Language if he chooses.

2. Example of Source Language and Intermediate Language

In order to illustrate the content of each of these ~~three~~ two symbolic systems, ?

let us take the problem of:

-- telling a computer (a person or a machine) the definition of the factorial of n, which is n times n-1 times n-2 and so on down to 3 times 2 times 1, except that factorial of 0 is 1;

-- directing the computer to compute the factorial of 5.

In these words just written we have expressed the problem in ordinary English.

In Source Language the problem is expressed:

```
FUNCTION FACTORIAL (N)
    IF N=0 THEN  1 ELSE
    N * FACTORIAL (N-1) ;
FACTORIAL (5);
```

In internal Language the problem is expressed:

```
(FUNCTION FACTORIAL (N)
    (IF (EQUAL N 0) 1
    (TIMES N (FACTORIAL (DIFFERENCE N 1))) ) )
(FACTORIAL 5)
```

space

The first thing we have to do is to describe source language and explain what is acceptable and what is not. To say what is acceptable requires a long series of statements and many examples.

3. Acceptable Characters

The standard acceptable characters for Source Language are:

- the 26 capital letters A to Z; capital o is written O;
- the 10 digits 0 to 9; the digit 0 is written Ø; the digit 1 for one is not the same character as the small letter L;
- the 24 characters in Table 1; each is shown there with its name and its usual meaning, if any.

In addition, from time to time, the absence of any character, i.e., nothing written, has meaning as the symbol "plus". In other ways the absence of any character ~~and this~~ is called an <u>empty</u>. For example, 4 and +4 have the same meaning; both are the negative of -4 (minus four). An <u>empty</u> is different from a space, such as the space between words in ordinary English, as produced by pressing the space-bar of a typewriter.

The last two signs in Table 1 are not literally expressed in Source Language by the characters ƀ, cr. Instead they are expressed in Source Language by pressing the space bar, and by pressing the carriage return key.

Table 1

No.	Class	Character	Name	Usual Meaning if Any
A1	Grouping signs	(	left parenthesis	start of an expression, list, etc.
A2		)	right parenthesis	finish of an expression, list, etc.
A3		[	left bracket	start of a block or an array of constants
A4		]	right bracket	finish of a block or an array of constants
B1	Operation signs	+	plus sign	PLUS
B2		-	minus sign	MINUS
B3		*	asterisk	TIMES
B4		/	slant or slash	DIVIDED BY
B5		\	backward slash	REMAINDER
B6		↑	up arrow	EXPONENT
B7		←	left arrow	ASSIGNMENT (SET....EQUAL to ....) lower case
C1	Punctuation	,	comma	separator between arguments of a function
C2		;	semicolon	separator between statements in sequence
C3		.	period or dot	the point in a scale of notation, (see note)
C4		:	colon	placed after labels in source language
D1	Relation signs	<	less than sign	LESS THAN lower case
D2		>	greater than sign	GREATER THAN lower case
D3		=	equal sign	EQUALS
E1	E	%	percent sign	escape character, enabling the next character to take a special assigned interpretation
E2		#	fence, (number sign)	start or end of a string
E3		$	dollar sign	tailing of names in sections of programs
E4		'	single quote	QUOTE
F1	F	ƀ	b slashed	space, in the sense of pressing the space bar on a typewriter
F2		(cr)	c r in a circle	carriage return, in the sense of pressing a typewriter key to produce carriage return

produce carriage return

4. The Period

Table 1 gives for the period or dot (.) "the point in a scale of notation, etc. (see note)".

In Source Language, the period is used in exactly four ways. We shall specify these ways here although the meaning of the information given here will not become clear until later.

1. The period is used for specifying the decimal point in a number (expressed according to the rules for numbers explained below) *
2. It is a permitted character in identifiers except in the first position of ~~a character~~ an identifier.
3. It is used in what is called the dot-notation in S-expressions in the precise form "space, dot, space" (see later explanation).
4. It is used as the infix operator for CONS in the form "space, dot, space".

((Primer for LISP 2 by Mike Levin and Ed Berkeley))

Draft, Sept. 20, 1965

Chapter 2. DATA -- Part 1

We will now consider the ideas used in Source Language for designating data. Data comprises the constant information which the implemented computer will accept as given. To designate data, a number of terms are used, which have special defined meanings.

1. Characters

A letter is one of the 26 capital letters of the alphabet.

An octal-digit is one of the eight digits Ø, 1, 2, 3, 4, 5, 6, 7.

A digit is a decimal digit; it is one of the digits from Ø to 9.

A sign is an empty (i.e., nothing written) or + or -.

A space is the result of pressing the space bar on a typewritter and occurs as a separator between expressions in source language. In talking about a space in situations where otherwise the meaning may not be clear, the sign b̸ (slashed b) is used, meaning a space. This sign however does not appear in Source Language itself.

A carriage return is the result of pressing the carriage return key on a typewriter. When Source Language is copied from one place to another, the end of one line of writing and the beginning of the next line of writing is not noticed, may vary, and the variation is not significant. In talking about carriage return, the symbol cr (c r in a circle) is used as its name, but this symbol does not appear in Source Language.

Other characters may occur in source language (such as / or %). They will be explained below as they occur.

2. Numbers

An integer is an acceptable expression of Source Language which stands for a positive or negative integer written in the usual way in the scale of 10, with an option for using a positive exponent.

Examples of acceptable numerical expressions are:

37	the decimal number 37
2EØ	the decimal number 2
-345E9	the decimal number minus 345 times 10 to the 9th power or minus 345-billion

Examples of unacceptable numerical expressions are:

E3	lc The absence of a digit in front of the E bars it as a number (It is a legal identifier, however, as noted below.)
2E-7	lc The negative exponent (the minus ~~in front of~~ after the E) bars it
2 E 7	lc The spaces bar it; the Syntax Translator would look on this as three expressions

An integer may be defined precisely as:

empty or + or -, followed by

one or more decimal digits, followed by

empty, or E followed by one or more decimal digits

The term integer (in Source Language) is short for decimal integer, or integer written in the decimal scale, as described above.

An octal is an acceptable expression of Source Language which stands for a positive or negative octal integer, with an option for using a positive exponent written in the decimal scale. Such a number is regularly used as a bit pattern, a pattern of ones and zeros produced by converting the octal digits Ø to 7 individually into binary equivalents ØØØ to 111 in the binary scale.

37Q	37 in the octal scale, 11111 in the binary scale
-2Q	minus 2 in the octal scale
+37Q	37 in the octal scale (the plus is not necessary but is acceptable)

5Q8 500000000 in the octal scale (note that the 8 is an exponent written in the scale of 10)

500000000Q this is the same as the last number, written in another acceptable way

Examples of expressions which are not acceptable are:

37 the absence of Q means that the number is decimal not octal

5Q-2 the negative exponent makes the octal number fractional; only positive integers in the octal scale are acceptable

40Q+3 the presence of a plus sign between Q and the next digit bars the expression

3.4Q the presence of the point in the octal scale makes the expression unacceptable

37 Q the space bars it; the Syntax Translator would treat this as two expressions

49Q the 9 makes the expression unacceptable as an octal

An octal may be precisely defined as:

empty or + or -, followed by

one or more octal-digits, followed by

Q, followed by

empty or one or more decimal digits

A real (in Source Language) is an acceptable expression of Source Language which stands for a positive or negative number with integral and fractional part expressed in the scale of 10, and with an option for using a positive or negative exponent.

Examples of acceptable reals are:

2. lc ~~T~~he number 2.0

2.3 lc ~~T~~he number 2 and 3 tenths

Ø.446	The decimal number 446 thousandths
-7.65	Minus seven point six five
7.65	Seven point six five
.2	The number two tenths
3.67E-4	3.67 times 10 to the minus 4 power
ØØØ.Ø32E8	.032 times 10 to the 8th power
ØØØ.ØØØ	A representation of zero

A real may be precisely defined as:

a sign, followed by

zero or more decimal digits, followed by

a period (a point), followed by

zero or more decimal digits, followed by

empty, or E followed by a sign followed by one or more decimal digits; provided that there is at least one decimal digit on one side or the other of the point.

A ~~N~~number (in Source Language) is an integer or an octal or a real.

~~A scale is a sign followed by one or more decimal digits.~~

These are all of the acceptable expressions which represent numbers.

In practice, there is a limit to the number of digits which may occur in the representation of a number. In Source Language as such, there is no specified limit.

3. Booleans

In reporting the truth values of statements there is need for reporting "true, yes, correct" or "false, no, wrong".

The first of these is TRUE which is an acceptable expression of Source Language.

The second of these is any one of the following expressions, all of which are interchangeable and equivalent:

FALSE, or

NIL, or

( )

A Boolean may be precisely defined as TRUE or FALSE or NIL or ( ).

4. Identifiers

In order to deal with functions, variables, and other operations of computing, we need a class of symbols which are here called "identifiers".

An identifier may be precisely defined as:

a letter, followed by

one or more letters or digits or periods

(excluding the special expressions TRUE, FALSE, NIL), or

else:

a percent sign, followed by

a string

We have not defined strings yet but we will come to them soon.

Examples of acceptable identifiers are:

CAR	PLUS	MER863.2	X
CDR	U	A1B2C3	FN
%#(((#	%#A B#	%#,, "#	

T. A. Weissman

8-20K/65

((Primer for LISP 2 by Mike Levin and Ed Berkeley))

Draft, Sept. 24, 1965

Chapter 2: DATA

Part 2

5. Strings, Comments, Hyphenators, and Spacers

From time to time as a computation proceeds we need to be able to write freely, disregarding all the conventions that we have hitherto set up. This happens for example when we want to write a sentence as a comment, or when we want to construct machine language programs using a suitable menemonic language which may not in any way agree with our Source Language so far defined.

What we write on one of these occasions may be called a <u>string</u>, a sequence of any characters whatever, which needs to be treated as a unit, as for example in a comment on a program.

In order to write a string in Source Language, basically, we begin the string with a fence and end the string with a fence.

Examples of acceptable strings are:

#THE POSITIVE ROOT IS#

#SUBROUTINE FOR SQUARE ROOT#

To define a string, we will first define "string-character".

A <u>string-character</u> is a letter or a digit or any of the following 19 characters:

Grouping signs (4)

Operation signs (7)

Punctuation signs except the semicolon (3)

Relation signs (3)

$, dollar sign (1)

a space, produced by the space bar of a typewriter (1)

Note that the following six characters are <u>not</u> string-characters:

fence

' single quote or quote mark

% percent sign

; semicolon

(cr) c r in a circle

ƀ slashed b

The slashed b is a name for a space, used when talking about strings, but not used in strings.

One might think from this definition that we could not use the five non-string characters in strings. But there is a way of avoiding this limitation. It appears in the definition of string.

A <u>string</u> may be precisely defined as:

a fence, followed by

one or more of any of the following

a string character

a semicolon

a quote mark followed by a quote mark

a quote mark followed by a fence

a quote mark followed by a percent sign

a quote mark followed by a carriage return

followed by a fence

The <u>meaning</u> of a string (in other words, what the Internal Language receives as translation from the Syntax Translator) is:

a string containing all the characters of the original in proper sequence, EXCEPT:

the initial fence is omitted;

the terminal fence is omitted;

all quote marks are omitted except that a pair of quote marks together yield a single quote mark in the translated string

Examples of strings and their translation appear below:

Source Language String	Translated String (or Meaning) in Intermediate Language
#ABC#	ABC
#''A#	'A
#CAR SUBR; CHOOSE CR X ''A#	CAR SUBR; CHOOSE X 'A
#'#ABC'##	#ABC#

A <u>comment</u> is:

a percent sign, followed by

a space, followed by

zero or more of:

- a string character, or
- a fence, or
- a quote mark,

followed by

a semicolon or a carriage return

Examples of acceptable comments are:

% THIS IS A COMMENT;

% COMMENT; A + B % THE PRECEDING EXPRESSION IS NOT A COMMENT

Examples of expressions which are not acceptable as comments are:

% THE SECOND PERCENT WILL CAUSE % AN ERROR;

% THIS COMMENT IS NOT PROPERLY TERMINATED- A+B

A <u>hyphenator</u> is the equivalent in Source Language of a hyphen in ordinary English. It is used to show that, although the end of a line has been reached in the middle of a "word" (or a string), no break whatever is intended.

One example of the use of a hyphenator is:

ABC%

DEF

The translation of this in Intermediate Language is:

ABCDEF

A hyphenator may even interrupt a number, so a second example is:

3.42%

53

which means 3.4253.

A third example is:

3.4E%

2

which means 3.4E2, which is the same as 340.

Another example is:

ABC%;DEF

which is translated:

ABCDEF

A <u>spacer</u> is the equivalent in Source Language of the space between words or other expressions when writing in ordinary English or mathematics. One or more spaces in Source Language is a spacer.

A <u>spacer</u> (in Source Language) may be precisely defined as one of (or a sequence of two or more of) spaces, carriage returns, and comments.

Examples of spacers are:

(1) space, space, space as in: A1   B1

(2) carriage return as in:

A1

B1

(3) %   THE SQUARE ROOT ROUTINE;

6. Tokens

A <u>token</u> (in Source Language) is any one of the following:

a Boolean, or a number, or an identifier, or a string, or one of the following 24 special tokens:

,	comma
;	semicolon
:	colon
.	This is "space, period, space", and has a special meaning
<	less than sign
>	greater than sign
=	equal sign, which means EQUAL
/=	slashed equal sign, which means NOT EQUAL
<=	which means LESS THAN OR EQUAL
>=	which means GREATER THAN OR EQUAL
+	PLUS
-	MINUS
*	TIMES
\	REMAINDER
/	DIVIDED BY
↑	up arrow, EXPONENT
←	left arrow, ASSIGNMENT (LET ... EQUAL)
-:	which means DIVIDED BY
(	left parenthesis
)	right parenthesis
[	left bracket
]	right bracket
'	quote mark
$	dollar sign

7. Atoms, S-Expressions, and Constants

In any discussion of any subject, we find it necessary to give names to the ideas we are going to talk about, both those which are defined at the beginning and those which are defined from time to time during the course of the discussion.

To name these ideas in LISP 2 systematically, we make use of identifiers, numbers, Booleans, and strings, and we put them together into what are called "acceptable symbolic expressions". This name is abbreviated to S-expression.

Some examples of S-expressions and their uses are:

CAR	a function of lists
(PLUS U V)	a sum of two variables U and V
(U . (V . W))	a list of two elements one of which is a sublist
( )	NIL
(3 #EXPONENT# R #RADIVS# PI #USUAL MEANING#)	a list of identifiers, and comments about them
(((A B)) (C D))	a list of sublists

One elementary type of S-expression is atom. An atom may be precisely defined as any one of the following:

a number, or a Boolean, or a string, or an identifier, or an array.

(The term array will not be defined or discussed in this primer, but in order to make definitions complete, it will be mentioned from time to time. The reason for this is that if LISP 2 without arrays is understood first, then the inclusion of arrays later is fairly easy.)

An S-expression may be precisely defined as:

an atom; or

a left parenthesis, followed by one or more S-expressions separated by spaces, followed by

a space, followed by a dot, followed by a space, followed by an S-expression, followed by a right parenthesis; or a left parenthesis, followed by zero or more S-expressions separated by

spaces, followed by

a right parenthesis

Some examples of S-expressions were given earlier.

A <u>constant</u> is:

a number, or a

a Boolean, or

a string, or

the quote mark followed by an S-expression

Examples of constants are:

7.6E2

( )

#THE END#

'(CDR'(U V W))

Examples of expressions that are not constants are:

CAR	an identifier
(CAR (CDR (QUOTE (U V W))))	an S-expression <u>not</u> preceded by a quote mark