KRYSTYNA JERZYKIEWICZ
JERZY SZCZEPKOWICZ

ALGOL 1204

SYSTEM PROGRAMOWANIA MASZYNY CYFROWEJ ODRA 1204

WARSZAWA 1973
PAŃSTWOWE WYDAWNICTWO NAUKOWE

Książka zawiera pełny opis obszernej realizacji międzynarodowego języka algorytmicznego ALGOL 60 — o nazwie ALGOL 1204 — dla używanych w Polsce zestawów maszyny cyfrowej ODRA 1204. Omówiono m. in. procedury wejścia i wyjścia, przyjęte rozstrzygnięcia znanych niejednoznaczności ALGOLu 60, zasady optymalizacji programów, kod wewnętrzny maszyny i sposoby korzystania z translatorów ALGOLu 1204. Najważniejsze informacje zilustrowano przykładami.

Książka może służyć jako podręcznik programowania, a także jako poradnik programisty i operatora maszyny ODRA 1204.

REDAKTOR WYDAWNICZY
CENTRUM OBLICZENIOWEGO PAN
Jan Lipski

Printed in Poland

PAŃSTWOWE WYDAWNICTWO NAUKOWE

Wydanie II Nakład 3000+150 egz. Ark. wyd. 9,50. Ark. druk. 10,50.
Papier piśm. mgł. kl. III, 80 g, 70×100. Podpisano do druku 15 XII 1972 r.
Druk ukończono w styczniu 1973 r. Zam. nr 742. D-10. Cena zł 28,—

Zakład Graficzny Wydawnictw Naukowych
Łódź, ul. Gdańska 162

Spis treści

WSTĘP

0.0. Informacje ogólne 7
0.1. Sprawność różnych systemów programowania 8
0.2. Terminologia stosowana w dalszym ciągu opisu . . 10
0.3. Tytuły następnych punktów 10

ROZDZIAŁ 1. Język ALGOL 1204

1.0. Związek ALGOLu 1204 z międzynarodowym językiem algorytmicznym ALGOL 60 11
1.1. Alfabet ALGOLu 1204 12
1.2. Odstępy i zmiany wiersza 14
1.3. Symbole podkreślone i przekreślone 15
1.4. Komentarze 16
1.5. Programy i dane na wielu taśmach 16
1.6. Znaki sterujące maszyny OPTIMA 17
1.7. Pamiętanie informacji w programie przetłumaczonym 17
1.8. Symbole tape i drum 17
1.9. Niezależne tłumaczenie procedur 18
1.10. Działanie instrukcji tape 19
1.11. Działanie instrukcji drum 20
1.12. Segmentacja 21

ROZDZIAŁ 2. Rozstrzygnięcia w ALGOLu 1204 ważniejszych niejednoznaczności ALGOLu 60

2.0. Uwagi ogólne 23
2.1. Zmienne z mianem own (zmienne własne) 23
2.2. Obszar działania nazw standardowych 24
2.3. Instrukcja warunkowa 25
2.4. Wyrażenia warunkowe 26
2.5. Wyrażenia proste 26
2.6. Instrukcja "dla" 27
2.7. Powtórzenia w zbiorze parametrów formalnych . . . 30
2.8. Zgodność parametru aktualnego ze specyfikacją parametru formalnego 31
2.9. Parametr o specyfikacji label w zbiorze wartości 32
2.10. Nazwa wartości funkcji w roli instrukcji procedury 32
2.11. Wartość procedury 32
2.12. Podstawienie pod nazwę procedury 33
2.13. Kolejność obliczania parametrów w zbiorze wartości 33
2.14. Skutki uboczne treści procedur 33

ROZDZIAŁ 3. Procedury i zmienne standardowe

3.0. Uwagi ogólne 37
3.1. Postać liczb czytanych przez program 38
3.2. Standardowe funkcje matematyczne 39
3.3. Procedura ref 40
3.4. Procedura key 40
3.5. Procedura setinput - wybór rodzaju wejścia . . . 40
3.6. Procedura read 41

3.7. Procedury ininteger i inreal 42
3.8. Procedura inchar 42
3.9. Procedura instring - czytanie łańcucha 43
3.10. Procedura setoutput - wybór rodzaju wyjścia . . . 43
3.11. Procedura outchar - drukowanie jednego znaku . . 44
3.12. Procedura space - drukowanie odstępów 45
3.13. Procedura line - drukowanie zmian wiersza 45
3.14. Procedura outstring - drukowanie łańcucha 45
3.15. Procedura print 46
3.16. Procedura format - wybór wzorca druku liczb . . . 47
3.17. (D) Drukarka wierszowa 50
3.18. Procedury wait i stop - zatrzymanie programu . . 52
3.19. Zmienne lastchar, lastinteger, lastreal, time, drumplace 53
3.20. Procedury copy i exch 54
3.21. (D) Procedury todrum i fromdrum 55

ROZDZIAŁ 4. Przykłady programowania w ALGOLu 1204

4.0. Uwagi ogólne 57
4.1. Warunkowe drukowanie wyników pośrednich 57
4.2. Czytanie danych z powtórzeniami 57
4.3. Czytanie danych logicznych 58
4.4. Drukowanie liczby całkowitej z minimalną ilością cyfr 59
4.5. Przykład operowania ciągiem o nieznanej liczbie elementów 60
4.6. Porównanie dwóch metod porządkowania 60
4.7. Pomiar czasu trwania operacji bardzo szybkich . . 62
4.8. Pomiar czasu bez wielokrotnego powtarzania operacji 62
4.9. Drukowanie wykresu rodziny funkcji 63
4.10. (D) Porządkowanie obiektów 66
4.11. Przejście z klawiatury P na klawiaturę A 68

ROZDZIAŁ 5. Zasady optymalizacji programu

5.0. Uwagi ogólne 71
5.1. Ważniejsze wyniki pomiarów czasu 71
5.2. Optymalizacja najprostszych elementów programu . 74
5.3. Zmienne ze wskaźnikami 75
5.4. Ogólne zasady optymalizacji instrukcji 76
5.5. Instrukcja "dla" 77
5.6. Instrukcja warunkowa i wyrażenie warunkowe . . . 78
5.7. Procedury i funkcje niestandardowe 79
5.8. Procedury rekursywne 80

ROZDZIAŁ 6. Kod wewnętrzny maszyny ODRA 1204 w ALGOLu 1204

6.0. Informacje wstępne 84
6.1. Rejestry maszyny i budowa słowa rozkazowego . . . 84
6.2. Reprezentacja liczb w maszynie 85
6.3. Schemat wykonywania rozkazu 87
6.4. Schematy warunków 88
6.5. Reguły zastępowania parametrów aktualnych procedur specjalnych adresami 88
6.6. Rozkazy skoków 89
6.7. Rozkazy działań na słowach krótkich 90
6.8. Rozkazy działań na słowach długich 91
6.9. Rozkazy przesuwań 93

6.10. Rozkazy generowania warunków 94
6.11. Rozkazy modyfikacji 94
6.12. Rozkaz skw 95
6.13. Uwagi o stosowaniu procedur specjalnych 95

ROZDZIAŁ 7. Przykłady stosowania kodu wewnętrznego

7.0. Uwagi ogólne 96
7.1. Obliczenie wielkości obszaru pamięci dostępnej dla rezerwacji dynamicznej 96
7.2. (D) Obliczanie wielkości obszaru roboczego pamięci bębnowej 96
7.3. Mnożenie liczby całkowitej długiej przez krótką . 97
7.4. Drukowanie słów maszynowych w postaci ósemkowej . 98
7.5. Drukowanie liczby całkowitej z minimalną ilością cyfr 98
7.6. Optymalizacja programu 99

ROZDZIAŁ 8. Wykrywanie i korygowanie błędów w programach

8.0. Uwagi ogólne 101
8.1. Sygnalizacja błędów w czasie tłumaczenia programu 102
8.2. Postać gramatyki pamiętanej przez translator . . 107
8.3. Drukowanie symboli podstawowych niedostępnych na klawiaturze P 109
8.4. Sygnalizacja błędów w czasie działania programu . 109
8.5. Błędy nie wykrywane 110
8.6. Ślad dynamiczny i ślad retroaktywny programu . . 110
8.7. Korygowanie i kopiowanie programu przy pomocy translatora 112
8.8. Numerowanie wierszy i punkty orientacyjne 113
8.9. Postać wykazu poprawek 114

ROZDZIAŁ 9. Maszyna i translatory

9.0. Typowe zestawy maszyny ODRA 1204 118
9.1. (M) Translatory A, B i podprogramy pomocnicze PP 119
9.2. (D) Translator D 120
9.3. Uruchamianie translatorów i programów 120
9.4. Interpretacja sygnałów systemu obsługi, translatorów i programów 121

ROZDZIAŁ 10. Systemy obsługi MASON i MASON (D)

10.0. Informacje wstępne 122
10.1. Wprowadzenie i uruchomienie systemu obsługi . . . 123
10.2. Pisanie zleceń 123
10.3. Obsługa urządzeń zewnętrznych 125
10.4. Stany maszyny i wykonalność zleceń 127
10.5. (D) Podział pamięci bębnowej 128
10.6. (D) Zapisanie systemu MASON (D) na bębnie 129
10.7. (D) Biblioteka systemu 130

ROZDZIAŁ 11. Wykazy

11.0. Kody znaków 131
11.1. Zlecenia dla translatorów ALGOLu 1204 134
11.2. Słowne opisy błędów 138
11.3. Sygnały zatrzymania translatorów i programów . . 139
11.4. Standardowe znaczenie klawiszy nr 0,1,2,3 i 4 w ALGOLu 1204 143

11.5. Zlecenia standardowe systemów MASON i MASON (D) . 143
11.6. Początkowy obszar pamięci operacyjnej w ALGOLu 1204 153

ROZDZIAŁ 12. Wersje specjalne ALGOLu 1204

12.0. Uwagi ogólne 156
12.1. Wersja DISP ALGOLu 1204 156
12.2. Translator D-COAN 158

Prace cytowane 158

Skorowidz nazw, sygnałów i symboli 159

WSTĘP

0.0. Informacje ogólne

W chwili obecnej pracuje w Polsce około 100 egzemplarzy maszyny cyfrowej ODRA 1204; liczba ta w najbliższym czasie wzrośnie. Niniejszy podręcznik jest przeznaczony dla tych użytkowników maszyny, którzy znają popularny i szeroko używany w Europie międzynarodowy język algorytmiczny ALGOL 60 i chcą go stosować jako język programowania. Czytelnik, który nie zna ALGOLu 60, może nauczyć się tego języka studiując np. ogólnie dostępną książkę S. Paszkowskiego [1]. Podręcznik nasz jest uzupełnieniem tej książki, ważnym oczywiście tylko dla maszyny ODRA 1204. Nie powtarzamy żadnych informacji zawartych w [1], ograniczając się do dokładnego opisania specyficznych cech konkretnej realizacji ALGOLu 60, o nazwie ALGOL 1204.

ALGOL 1204 jest systemem programowania automatycznego o wysokiej - w granicach możliwości maszyny - sprawności użytkowej. Najistotniejsze cechy tego systemu są następujące:

- język ALGOL 1204 jest bardzo obszerną realizacją ALGOLu 60, tzn. nakłada niewiele ograniczeń na ALGOL 60 (p. 1.0),
- alfabet ALGOLu 1204 praktycznie nie różni się od alfabetu ALGOLu 60 (p. 1.1),
- sprawność programów napisanych w ALGOLu 1204 mało różni się od sprawności równoważnych programów napisanych w kodzie wewnętrznym maszyny przez człowieka (p. 0.1),
- język ALGOL 1204 zawiera znaczną część kodu wewnętrznego maszyny (p. 6.0 - 6.13),
- istnieje obecnie sześć porównywalnych translatorów ALGOLu 1204 dostosowanych do możliwości różnych zestawów maszyny (p. rozdz. 9 i 12),
- wszystkie translatory ALGOLu 1204 zapewniają istotną pomoc przy wykrywaniu i korygowaniu błędów w programach (p. 8.0 - 8.9),

Oprócz podstawowych elementów ALGOLu 1204 - w rodzaju procedur czytania danych i drukowania wyników - w podręczniku omówio-

WSTĘP

0.0. Informacje ogólne

W chwili obecnej pracuje w Polsce około 100 egzemplarzy maszyny cyfrowej ODRA 1204; liczba ta w najbliższym czasie wzrośnie. Niniejszy podręcznik jest przeznaczony dla tych użytkowników maszyny, którzy znają popularny i szeroko używany w Europie międzynarodowy język algorytmiczny ALGOL 60 i chcą go stosować jako język programowania. Czytelnik, który nie zna ALGOLu 60, może nauczyć się tego języka studiując np. ogólnie dostępną książkę S. Paszkowskiego [1]. Podręcznik nasz jest uzupełnieniem tej książki, ważnym oczywiście tylko dla maszyny ODRA 1204. Nie powtarzamy żadnych informacji zawartych w [1], ograniczając się do dokładnego opisania specyficznych cech konkretnej realizacji ALGOLu 60, o nazwie ALGOL 1204.

ALGOL 1204 jest systemem programowania automatycznego o wysokiej - w granicach możliwości maszyny - sprawności użytkowej. Najistotniejsze cechy tego systemu są następujące:

- język ALGOL 1204 jest bardzo obszerną realizacją ALGOLu 60, tzn. nakłada niewiele ograniczeń na ALGOL 60 (p. 1.0),
- alfabet ALGOLu 1204 praktycznie nie różni się od alfabetu ALGOLu 60 (p. 1.1),
- sprawność programów napisanych w ALGOLu 1204 mało różni się od sprawności równoważnych programów napisanych w kodzie wewnętrznym maszyny przez człowieka (p. 0.1),
- język ALGOL 1204 zawiera znaczną część kodu wewnętrznego maszyny (p. 6.0 - 6.13),
- istnieje obecnie sześć porównywalnych translatorów ALGOLu 1204 dostosowanych do możliwości różnych zestawów maszyny (p. rozdz. 9 i 12),
- wszystkie translatory ALGOLu 1204 zapewniają istotną pomoc przy wykrywaniu i korygowaniu błędów w programach (p. 8.0 - 8.9),

Oprócz podstawowych elementów ALGOLu 1204 - w rodzaju procedur czytania danych i drukowania wyników - w podręczniku omówio-

no przyjęte w ALGOLu 1204 rozstrzygnięcia znanych niejednoznaczności ALGOLu 60, zasady optymalizacji programów i kod wewnętrzny maszyny. Najważniejsze informacje zilustrowano przykładami, m. in. kompletnych programów napisanych w ALGOLu 1204.

0.1. Sprawność różnych systemów programowania

ALGOL 1204 nie jest jedynym systemem programowania maszyny ODRA 1204, ani nawet jedyną realizacją ALGOLu 60 dla tej maszyny. Wybierając jakiś system programowania jako podstawowy, użytkownik interesuje się najczęściej pracochłonnością procesu prowadzącego od sformułowania zadania do wydrukowania przez maszynę wyników. Pytania bardziej szczegółowe mogą być następujące:

p1. Czy programowanie w danym języku jest łatwe (przy założeniu odpowiedniej znajomości tego języka)?
p2. Jakie są możliwości korzystania z materiałów gotowych, na przykład opisów metod i algorytmów publikowanych w literaturze naukowej albo znajdujących się w bibliotece procedur maszyny?
p3. Jakie są możliwości sprawdzania i korygowania programów?
p4. Jaka jest sprawność translatora?
p5. W jakim stopniu program utworzony przez translator wykorzystuje możliwości maszyny?

Odpowiadamy krótko na te pytania. Programowanie w ALGOLu 60 - w porównaniu z innymi językami - należy uznać za łatwe; m. in. z tego powodu ALGOL 60 zyskał dużą popularność nie tylko jako język programowania, ale także jako symbolika uzupełniająca tradycyjną symbolikę matematyczną w publikacjach naukowych. Wiele czasopism (np. Communications of the ACM, Numerische Mathematik, Zastosowania Matematyki) publikuje algorytmy, z których znaczną większość można bez zmian wykorzystać w programach napisanych w ALGOLu 1204. W Instytucie Matematycznym Uniwersytetu Wrocławskiego opracowano specjalnie dla maszyny ODRA 1204 dużą bibliotekę procedur, rozpowszechnianą przez producenta maszyny. Poszczególne ośrodki obliczeniowe dysponują również własnymi bibliotekami algorytmów, nie zawsze publikowanymi, ale udostępnianymi użytkownikom maszyny.

Wystarczające odpowiedzi na pytania p3-p4 znajdują się w rozdziale 8 podręcznika. Odpowiedź na pytanie p5 ma istotne znaczenie tylko w przypadku wielokrotnego powtarzania obliczeń dla

zmieniających się danych. Przypadek ten nie jest rzadki; z tego powodu może być interesujący opis eksperymentu, w którym zbadano kilka równoważnych programów napisanych w różnych językach dla maszyny ODRA 1204, mianowicie

- w ODRA-ALGOLu (jest to starsza - znacznie uboższa od ALGOLu 1204 - realizacja ALGOLu 60),
- w autokodzie MOST 2 (jest to rozszerzenie autokodu MOST 1 opracowanego dla starszych maszyn ODRA 1003 i 1013; w autokodzie nie ma np. pojęcia lokalności, co praktycznie wyklucza tworzenie biblioteki podprogramów),
- w ALGOLu 1204,
- w języku wewnętrznym maszyny (chodzi o język, w którym można korzystać ze wszystkich bez wyjątku możliwości maszyny).

W każdym z tych języków napisano ten sam algorytm rozwiązywania układów równań liniowych. Programy sprawdzono na maszynie dla tych samych danych i otrzymano identyczne rozwiązania. Programy miały następujące własności:

- program w ALGOLu był tak zredagowany, aby dał się przetłumaczyć przez translator ODRA-ALGOLu i ALGOLu 1204,
- w programie w ALGOLu macierz układu zapamiętano jako wektor, aby umożliwić łatwe i bezpośrednie przepisanie programu na autokod i na język wewnętrzny.

Tak zaplanowany eksperyment przy badaniu (p5) powinien dać zdecydowanie niekorzystne wyniki dla ALGOLu, z następujących powodów:

- programy w ALGOLu rezerwują pamięć maszyny dynamicznie i m. in. z tego powodu zawierają dynamiczną kontrolę wskaźników (przy każdym odwołaniu do zmiennej ze wskaźnikami),
- programy w autokodzie i w języku wewnętrznym nie zawierały żadnej kontroli wskaźników.

Odpowiednie pomiary przeprowadzono dla układu 23 równań liniowych; wyniki podajemy w tabeli poniżej:

	długość programu w pamięci	czas działania programu
ODRA-ALGOL	416	48 sek
autokod MOST 2	nieznana	21 sek
ALGOL 1204	312	15 sek
język wewnętrzny	214	10 sek

Szybsze działanie programu w ALGOLu 1204 w porównaniu z programem w autokodzie tłumaczy się tym, że stratę czasu wynikającą z odwołań do zmiennych ze wskaźnikami zrekompensowała wyższa sprawność innych elementów programu.

0.2. Terminologia stosowana w dalszym ciągu opisu

Stosujemy konsekwentnie terminologię ustaloną w [1]. Używamy także terminów, których w [1] nie ma. Te nowe terminy są dwóch rodzajów:

1° Drobne uproszczenia terminów znanych, ale niewygodnych, kiedy trzeba je często wymieniać. Na przykład, zamiast "zmienna typu real albo typu integer" mówimy "zmienna arytmetyczna", a zamiast "zmienna typu Boolean" mówimy "zmienna logiczna". Podobnie należy rozumieć terminy

tablica arytmetyczna,
tablica logiczna.

Terminologię tego rodzaju przyjmujemy za oczywistą i nigdzie jej nie wyjaśniamy.

2° Pojęcia niezbędne dla opisu konkretnej reprezentacji ALGOLu i niektórych ograniczeń spowodowanych np. skończoną wielkością pamięci maszyny. Te pojęcia wyjaśniamy szczegółowo wszędzie tam, gdzie pojawiają się po raz pierwszy.

0.3. Tytuły następnych punktów

Istnieje sześć porównywalnych translatorów ALGOLu 1204 dla różnych zestawów maszyny (p. rozdz. 9 i 12). Informacje podane dalej są na ogół ważne dla wszystkich zestawów. Nieliczne wyjątki są opisane w miejscach oznaczonych literami M lub D w nawiasach. Litera M oznacza, że podana informacja jest ważna tylko dla zestawu minimalnego, a litera D oznacza, że informacja dotyczy zestawu D (p. 9.0). Jeżeli cały punkt dotyczy tylko jednego zestawu, to odpowiednią z liter M lub D podano w tytule tego punktu, po jego numerze, na przykład: "3.21. (D) Procedury todrum i fromdrum". Takich punktów jest w książce niewiele, co odzwierciedla fakt, że typowy program w ALGOLu 1204 można uruchomić na dowolnym zestawie maszyny.

ROZDZIAŁ 1. Język ALGOL 1204

1.0. Związek ALGOLu 1204 z międzynarodowym językiem algorytmicznym ALGOL 60

ALGOL 1204 jest w sensie gramatycznym podzbiorem ALGOLu 60. Rozumiemy przez to, że jeżeli pominiemy pewne szczegóły typograficzne związane z możliwościami maszyny do pisania, to każdy program napisany w ALGOLu 1204 jest pod względem gramatycznym programem w ALGOLu 60. W szczególności z urządzeń zewnętrznych maszyny ODRA 1204 oraz z rozkazów w kodzie wewnętrznym tej maszyny korzysta się za pomocą konstrukcji językowych zgodnych z gramatyką języka ALGOL 60 (patrz Dodatek A w [1]).

Niżej podano pełny wykaz ograniczeń obowiązujących w ALGOLu 1204.

G. Ograniczenia gramatyczne

G1. Każdy parametr formalny musi mieć specyfikację.

G2. Etykieta musi być nazwą.

S. Ograniczenia i zmiany semantyczne

S1. Skok przy nie określonym przełączeniu powoduje zatrzymanie programu i sygnalizację błędu.

S2. Wynik potęgowania jest typu integer, jeżeli podstawa jest typu integer, a wykładnik jest liczbą całkowitą; w przeciwnym razie wynik potęgowania jest typu real.

S3. Liczba, która w sensie ALGOLu 60 jest typu integer, ale nie należy do przedziału $\langle -2^{23}+1,\ 2^{23}-1\rangle$, w ALGOLu 1204 jest typu real.

I. Ograniczenia ilościowe

I1. Liczby całkowite muszą należeć do przedziału $\langle -2^{23}+1,\ 2^{23}-1\rangle$ (czyli $\langle -8388607,\ 8388607\rangle$); wartości zmiennych całkowitych muszą należeć do przedziału $\langle -2^{23},\ 2^{23}-1\rangle$.

I2. Liczby rzeczywiste muszą należeć do przedziału $\langle -2^{511},\ 2^{511}-2^{474}\rangle$ (czyli ok. $\langle -6.703_{10}153,\ 6.703_{10}153\rangle$).

I3. Nazwy, których 63 początkowe znaki są takie same, uważa się za identyczne.

I4. Wykaz par granicznych w opisie tablic z mianem own może zawierać co najwyżej 10 par granicznych.

I5. Stopień procedury nie może przekraczać 3.

UWAGA 1

Stopniem procedury nazywamy w I5 liczbę całkowitą określoną następująco:

1° Program główny jest procedurą stopnia zero.

2° Procedura jest stopnia n, jeżeli najwęższy opis procedury zawierający opis tej pierwszej jest stopnia n-1.

Stopień procedury określa więc głębokość zanurzenia opisu procedury w innych opisach procedur.

UWAGA 2

Istnieją również inne ograniczenia ilościowe spowodowane skończoną wielkością rejestrów maszyny i sposobem pamiętania informacji przez translator. Nie wymieniamy wszystkich tych ograniczeń, ponieważ są one tak dobrane, że ich przekroczenie jest mało prawdopodobne bez uprzedniego wypełnienia całej dostępnej pamięci operacyjnej informacjami o tłumaczonym programie.

UWAGA 3

Przekroczenie dowolnego ograniczenia jest sygnalizowane, z wyjątkiem przekroczenia przedziału liczb całkowitych przy wykonywaniu działań na takich liczbach (p. 8.5).

UWAGA 4

Niektóre własności języka ALGOL 60 nie zostały dotąd jednoznacznie ustalone (np. identyczność zmiennych własnych, konflikt między opisami i specyfikacjami itp.). Odpowiednie wyjaśnienia - ważne tylko w ALGOLu 1204 - podajemy w rozdziale 2.

1.1. Alfabet ALGOLu 1204

Alfabet ALGOLu 1204 jest związany z elektryczną maszyną do pisania OPTIMA 527/528, której używa się do pisania programów i danych dla maszyny cyfrowej ODRA 1204; otrzymuje się zwykły maszynopis programu lub danych i ośmiokanałową taśmę perforowaną przeznaczoną do czytania przez maszynę cyfrową.

Używane są dwie wersje maszyny OPTIMA, różniące się liczbą i rodzajem znaków dostępnych na klawiaturze. Pierwsza z tych

klawiatur, którą będziemy nazywać klawiaturą A (algolowską), zawiera następujące znaki drukowane

a b c d e f g h i j k l m n o p q r s t u v w x y z

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

+ - × / ÷ ↑

= < >

¬ ∧ ∨ ⊃ ≡

. $_{10}$ ( ) := ; : [ ] ' ' ␣

| _ ?

i pewną liczbę znaków pomocniczych, np. odstęp, zmianę wiersza i inne znaki sterujące (p. 11.0), których obecności na taśmie można domyślać się na podstawie układu graficznego maszynopisu. Przytoczony wyżej zestaw znaków, z wyjątkiem trzech znaków ostatnich, jest omówiony w [1]; w ALGOLu 1204 ten zestaw ma dokładnie takie znaczenie, jak w [1]. Znaki | _ (przekreślanie i podkreślanie) znajdują się na klawiszu, którego naciśnięcie nie przesuwa karetki maszyny do pisania. W ten sposób pisząc kolejno | = otrzymujemy na papierze napis postaci ≠ oznaczający w ALGOLu 1204 relację "nie równe". Podobnie, pisząc kolejno znaki _ < otrzymujemy na papierze ≤ (czyli "mniejsze lub równe"), a złożenie znaków _ > daje znak ≥ (czyli "większe lub równe"). Posługując się klawiszem podkreślania można tak samo utworzyć inne symbole algolowskie, np. if then else end true itd.; szczegóły dotyczące tych symboli podano w p. 1.3.

Znak zapytania ? w ALGOLu 60 nie występuje. W ALGOLu 1204 jest to m. in. symbol końca taśmy; w określonych przypadkach (p. 1.5) przeczytanie tego znaku przez maszynę powoduje chwilowe wstrzymanie jej pracy.

Programy w ALGOLu 1204 można również pisać na klawiaturze P (polskiej), która różni się od klawiatury A głównie tym, że niektóre znaki algolowskie zastąpiono literami charakterystycznymi dla języka polskiego, zgodnie z poniższą tabelą:

klawiatura A: ↑ := ÷ ␣ ⊃ < ≡ > ¬ ∧ ∨ $_{10}$ × | ' '
klawiatura P: Ż ż ź ó Ł ł ę ą ś ń ć ◇ * _ ' `

Napis I↓J - i inne podobne - wykonany na klawiaturze P będzie przez translator potraktowany tak, jak napis I<J wykonany na klawiaturze A. Aby umożliwić bardziej czytelne pisanie programów na klawiaturze P, w ALGOLu 1204 dopuszczono następującą transkrypcję znaków algolowskich niedostępnych na klawiaturze P:

×	÷	↑	¬	∧	∨	⊃	≡	≠	<	≤	>	≥	10	:=	'	'	␣
*	div	**	not	and	or	imp	eqv	ne	lt	le	gt	ge	^	:=	'	'	ó

Ponieważ translator nie może rozpoznać wersji maszyny do pisania użytej do przygotowania programu, więc również używając klawiatury A można zamiast

```
b:=x<y∧eps>u↑v
```

napisać

```
b:=x<y and eps>u↑v
```

albo też

```
b:=x lt y and eps gt uxxv
```

Tego jednak nie zalecamy, gdyż odpowiednio przygotowany na klawiaturze A algorytm w ALGOLu 1204 nadaje się np. do bezpośredniej publikacji jako algorytm w ALGOLu 60. Własność ta jest bardzo istotna; wiadomo bowiem, że większość błędów w publikowanych algorytmach powstaje w czasie przepisywania na ALGOL 60 programu uprzednio uruchomionego na maszynie i w trakcie tradycyjnego składu w drukarni.

Programy w ALGOLu 1204 można również pisać na niektórych wersjach automatu produkcji firmy FRIDEN; ogólne zasady pisania są takie same, jak dla maszyny OPTIMA.

We wszystkich przykładach występujących w dalszym ciągu stosujemy konsekwentnie znaki z klawiatury A i najbardziej naturalną transkrypcję alfabetu ALGOLu 60.

1.2. Odstępy i zmiany wiersza

Zmianę wiersza można wykonać na maszynie OPTIMA dwoma sposobami:

a. przez naciśnięcie klawisza "nowa linia" (NL), co powoduje powrót karetki na początek wiersza, wysunięcie papieru o jeden wiersz i wydziurkowanie jednego znaku na taśmie,
b. przez kolejne naciśnięcie klawiszy "powrót karetki" (CR) i "wysuw papieru" (LF); położenie papieru jest wtedy takie, jak po naciśnięciu klawisza "nowa linia", a na taśmie zostaną wydziurkowane dwa znaki.

Przy dziurkowaniu taśm programów i danych należy brać pod uwagę, że znak "powrót karetki" jest pomijany przez translator, a znak "wysuw papieru" jest traktowany tak, jak znak "nowa linia". Wyjątek stanowią wnętrza łańcuchów, w których obowiązują odrębne reguły interpretacji znaków (p. 3.14, UWAGA).

Przy tłumaczeniu programu translator ALGOLu 1204 pomija odstępy i zmiany wiersza wszędzie poza łańcuchami; można to wykorzystać w celu nadania maszynopisowi programu czytelnej postaci graficznej.

Symbolu odstępu ␣ można użyć tylko w komentarzu, we wnętrzu łańcucha lub jako symbolu końca liczby na taśmie z danymi. Przy drukowaniu łańcucha symbol odstępu zostanie zamieniony na odstęp.

1.3. Symbole podkreślone i przekreślone

Jeżeli litera użyta poza łańcuchem jest podkreślona, to nie bierze się pod uwagę wielkości tej litery. Dodatkowo, zamiast podkreślania liter dopuszcza się ich przekreślanie (na klawiaturze A), bez zmiany znaczenia tych liter. Tak więc symbole

Boolean
boolean
Boolean

są równoważne.

Znaku podkreślonego nie można przekreślić, a znaku przekreślonego nie można podkreślić. Po znaku podkreślania można powtórnie użyć znaku podkreślania (wielokrotne podkreślanie), a po znaku przekreślania można powtórnie użyć znaku przekreślania (wielokrotne przekreślanie).

Znaki < > nie mogą być przekreślone, a znak = nie może być podkreślony, z wyjątkiem przypadku, gdy znaki te występują we wnętrzu łańcucha lub w komentarzu zaczynającym się od comment.

Odstępy i zmiany wiersza użyte we wnętrzu symbolu złożonego ze znaków podkreślonych lub przekreślonych, albo między takimi symbolami, są również pomijane. Tak więc fragment programu

```
begin
 integer procedure f(g,h);
  integer procedure g,h;
```

można również napisać w postaci

```
beginintegerproceduref(g,h);integerprocedureg,h;
```

W szczególności wszystkie cztery podane niżej symbole

go to
goto
go to
g o t o

są równoważne.

1.4. Komentarze

Komentarze pisane po end, a przed średnikiem, end lub else muszą tworzyć ciąg symboli podstawowych ALGOLu 1204. Tak więc ciąg znaków postaci

end a lt b imp c lt d;

kończący instrukcję złożoną lub blok jest równoważny ciągowi

end;

a ciąg znaków

end opisu procedury p;

jest błędny.

Komentarza zaczynającego się od comment w ALGOLu 1204 można użyć także przed programem; komentarz taki jest kopiowany na tym urządzeniu, które operator wyznaczył do wyprowadzania wykazu błędów lub punktów orientacyjnych programu (p. 8.1, 8.8) pod warunkiem, że klawisz nr 1 pulpitu maszyny jest zwolniony lub klawisz nr 2 jest wciśnięty (p. 8.7, 8.8).

W czasie kopiowania komentarza zamiast symbolu comment i średnika kończącego komentarz na odpowiednie urządzenie wyprowadza się znaki zmiany wiersza.

1.5. Programy i dane na wielu taśmach

Jeżeli w czasie czytania programu lub taśmy z danymi zostanie przeczytany znak zapytania, który nie jest zawarty we wnętrzu łańcucha, to maszyna zatrzyma się, a na monitorze zostanie wydrukowany znak zapytania. Po napisaniu na monitorze dowolnego znaku działanie translatora lub programu zostanie wznowione.

Znak zapytania użyty jako symbol końca taśmy programu jest pod wszystkimi innymi względami pomijany. Znak zapytania kończący taśmę z danymi jest jednym z dopuszczalnych symboli końca liczby.

Ze względu na sposób czytania znaków przez translator wymaga się, aby po znaku zapytania, który ma spowodować tymczasowe zatrzymanie maszyny, na taśmie było jeszcze co najmniej 16 odstępów lub zmian wiersza.

1.6. Znaki sterujące maszyny OPTIMA

Elektryczna maszyna do pisania OPTIMA zapewnia możliwość czytania i dziurkowania pewnych znaków sterujących, które nie są widoczne w maszynopisie, np. tabulatora, stopu czytnika maszyny do pisania, znaku kasowania błędu itp. (p. 11.0). W ALGOLu 1204 znaki te nie mają żadnego wpływu na interpretację programu lub taśmy z danymi i można ich używać dowolnie. Znaki sterujące umieszczone we wnętrzu łańcucha są pamiętane razem z innymi znakami tego łańcucha. Jeśli łańcuch będzie wyprowadzony na taśmę, to znaki te będą również wyprowadzone; na monitorze i na drukarce wierszowej znaki sterujące są automatycznie pomijane, z wyjątkiem odstępu i zmiany wiersza.

1.7. Pamiętanie informacji w programie przetłumaczonym

Program przetłumaczony przez translator zawiera między innymi wykaz użytych liczb całkowitych, wykaz liczb rzeczywistych i wykaz łańcuchów; każdy z tych wykazów zawiera wyłącznie elementy parami różne.

Wartości zmiennych typu integer i typu Boolean zajmują po jednej komórce pamięci maszyny, a wartości zmiennych typu real - po dwie komórki.

Informację o wielkości wolnej pamięci operacyjnej lub pamięci bębnowej otrzymuje się w sposób opisany w p. 11.1 i 11.5 (zlec. "dd 21 22" i "drum").

1.8. Symbole tape i drum

W ALGOLu 60 treścią procedury może być - oprócz dowolnej instrukcji - element zwany kodem (p. [1], Dodatek A, 5.41), którego z oczywistych powodów nie zdefiniowano. W ALGOLu 1204 przyjęto następującą składnię tego elementu

⟨kod⟩ ::= tape | drum

z tym ograniczeniem, że symbol tape lub drum może być użyty wyłącznie jako treść procedury stopnia 1 (p. 1.0, UWAGA 1). Tak

więc blok w ALGOLu 1204 może zaczynać się tak

```
begin
 real procedure Det(A,n);
  value n;
  integer n;
  array A;
  drum; comment koniec opisu funkcji Det;
 procedure Proces1(x,y);
  real x,y;
  tape; comment koniec opisu procedury Proces1;
 ......
```

W nagłówku tego bloku mamy dwa kompletne opisy procedur o nazwach Det i Proces1.

Z podanych przykładów wynika, że symbole tape i drum można pod względem gramatycznym traktować jako instrukcje, a zatem można mówić o ich działaniu.

Działanie instrukcji tape lub drum rozpoczyna się natychmiast po wywołaniu procedury, której treścią jest ta instrukcja i powoduje wprowadzenie z zewnątrz - z uprzednio przygotowanej taśmy papierowej lub z pamięci bębnowej - do aktualnie wolnych komórek pamięci operacyjnej właściwej treści procedury i wykonanie obliczeń według tej treści. Po zakończeniu tych obliczeń komórki zajmowane przez treść procedury są ponownie dostępne dla rezerwacji dynamicznej. Tak więc treść procedury może - w razie potrzeby - zmieniać się po kolejnych wywołaniach tej procedury.

Opisany mechanizm umożliwia wykonanie segmentacji programu, tzn. podzielenie go na części, z których nie wszystkie jednocześnie znajdują się w pamięci operacyjnej. Z tego powodu informację wprowadzaną do pamięci operacyjnej w wyniku działania instrukcji tape lub drum będziemy nazywać segmentem programu. Sposób otrzymywania segmentów i ważniejsze szczegóły operacyjne opisano w następnych punktach. Ogólne wskazówki dotyczące segmentacji podano w p. 1.12.

1.9. Niezależne tłumaczenie procedur

Jeżeli zakończony średnikiem opis procedury spełnia następujące warunki:

(a) treść procedury nie zawiera nazw w niej nielokalnych,

(b) treść procedury nie zawiera zmiennych z mianem own,
(c) treścią procedury nie jest ani tape ani drum,

to można go przetłumaczyć niezależnie od programu i wyprodukować jego taśmę binarną (p. 11.1) w dowolnej liczbie egzemplarzy. Taśmy takiej używa się jako segmentu programu (p. 1.8), tzn. jako informacji wprowadzanej do pamięci w wyniku działania instrukcji tape lub drum (p. 1.10 i 1.11).

Nagłówek procedury, której treścią jest instrukcja tape lub drum i nagłówki segmentów wprowadzanych w wyniku działania tej instrukcji mogą różnić się nazwami i zbiorami wartości, ale muszą mieć te same typy i równoważne zbiory specyfikacji.

PRZYKŁAD

Opisowi procedury

```
real procedure Det(A,n);
 array A;
 integer n;
 tape
```

może odpowiadać segment otrzymany po przetłumaczeniu opisu procedury

```
real procedure WYZNACZNIK(MA) stopnia:(s);
 value s;
 integer s;  array MA;
 begin
  comment obliczenie wyznacznika MA[1:s,1:s];
  ......
 end WYZNACZNIK;
```

Zgodność nagłówka procedury Det z nagłówkiem segmentu WYZNACZNIK sprawdza się przy każdym wprowadzaniu tego segmentu.

Ponieważ produkowanie taśmy segmentu odbywa się tylko na żądanie operatora (p. 11.1), więc tłumaczenie opisu procedury niezależnie od programu jest również wygodnym sposobem sprawdzenia formalnej poprawności lub skorygowania (reperforacji) tego opisu; typowe opisy procedur spełniają na ogół warunki (a), (b) i (c) wymienione na początku.

1.10. Działanie instrukcji tape

Jeżeli treścią procedury o nazwie P jest instrukcja tape, to po wywołaniu procedury P są wykonywane następujące czynności:

c1. Na monitorze program drukuje w nowym wierszu nazwę P, po czym czeka na napisanie na monitorze dowolnego znaku przez operatora.

c2. Po napisaniu tego znaku program czyta segment i sprawdza jego poprawność; w przypadku wykrycia błędu program wraca do czynności c1, a w przeciwnym razie po przeczytaniu całego segmentu program czeka ponownie na napisanie na monitorze dowolnego znaku przez operatora.

c3. Po napisaniu tego znaku następuje uruchomienie przeczytanego segmentu.

Opisane tutaj czynności wyglądają tak samo dla wszystkich zestawów maszyny (p. 9.0). Wynika stąd, że na zestawie D z dwoma czytnikami taśma segmentu jest czytana z czytnika pierwszego.

Bezpośrednio przed czytaniem segmentu z taśmy program powinien zakończyć czytanie kolejnej taśmy z danymi, gdyż przejście do czytania segmentu powoduje wyzerowanie zapasu poprzednio przeczytanych znaków (zapas zawiera co najwyżej 16 znaków). Dla zestawu D z dwoma czytnikami to żądanie dotyczy oczywiście tylko taśmy z danymi czytanej z pierwszego czytnika.

1.11. Działanie instrukcji drum

Dla zestawu M maszyny instrukcje tape i drum mają dokładnie to samo znaczenie. Przy korzystaniu z translatora D ALGOLu 1204, instrukcja tape działa tak, jak opisano w p. 1.10, a instrukcja drum ma inne własności.

Jeżeli treścią procedury P jest drum, to w momencie wywołania procedury P wyróżnia się dwa przypadki:

p1. Jeżeli od chwili wprowadzenia programu do pamięci operacyjnej nie była jeszcze wykonana instrukcja procedury P, to instrukcja drum działa najpierw dokładnie tak samo, jak instrukcja tape, czyli powoduje czytanie segmentu z uprzednio przygotowanej taśmy binarnej (p. 1.9), po czym zapisuje się przeczytaną taśmę w pamięci bębnowej, w końcowej części obszaru roboczego (p. 10.5); jednocześnie w programie następuje odnotowanie obecności segmentu w pamięci bębnowej i takie uaktualnienie liczby ograniczającej wartość zmiennej standardowej drumplace (p. 11.6), aby program nie mógł zniszczyć zapisu tego segmentu (p. 3.21).

p2. Jeżeli od chwili wprowadzenia programu do pamięci była już wykonana instrukcja procedury P, to segment znajduje się w pamięci bębnowej. W tym przypadku wszystkie czynności związane z wprowadzeniem i uruchomieniem segmentu odbywają się automatycznie, bez interwencji operatora.

Zapis segmentów bębnowych jest chroniony przed zniszczeniem tylko w czasie istnienia programu głównego w pamięci operacyjnej maszyny. Po ponownym wprowadzeniu programu głównego do pamięci operacyjnej czynności zapisywania segmentów bębnowych muszą być powtórzone.

1.12. Segmentacja

W ALGOLu 1204 segmentacji programu nie wykonuje się zbyt często, ponieważ wielkość pamięci operacyjnej maszyny jest wystarczająca dla zapamiętania przekładu typowych programów obliczeniowych. Jeżeli przekład programu nie mieści się w pamięci maszyny, tzn. w czasie tłumaczenia tego programu translator sygnalizuje SPACE OVERFLOW, to - pomijając przypadek stosowania wersji A translatora (p. 9.1) lub nieoptymalnego programowania - mamy do czynienia z programem dość złożonym. W programie tego rodzaju można prawie zawsze wyodrębnić pewną liczbę procesów obliczeniowych rozłącznych w tym sensie, że po zakończeniu każdego z nich instrukcje realizujące ten proces przez dłuższy czas - być może do zakończenia obliczeń - nie będą wykonywane. Programy tego rodzaju dobrze nadają się do segmentacji. Aby wykonać segmentację programu, należy wyodrębnione procesy obliczeniowe opisać w postaci procedur do niezależnego tłumaczenia (p. 1.9), z parametrami oznaczającymi - w najogólniejszym przypadku - dane i wyniki tego procesu. Procedury te należy przetłumaczyć i wyprodukować ich taśmy binarne (p. 11.1), które są segmentami.

Z informacji podanych w poprzednich punktach wynika, że samo przygotowanie jakiegoś segmentu programu nie przesądza jeszcze o tym, w jakim programie segment ten będzie użyty, ani też o tym, czy będzie to segment taśmowy, czy bębnowy (tzn. czytany w wyniku wykonania instrukcji tape, czy też instrukcji drum). Tak więc raz wykonany komplet segmentów może być używany w wielu różnych programach.

O tym, czy segment programu ma być umieszczony w pamięci bębnowej, czy też wprowadzać go z taśmy przy każdym wywołaniu odpo-

wiedniej procedury o treści tape, łatwo można zdecydować biorąc pod uwagę, że wprowadzanie segmentu z pamięci bębnowej jest szybkie i nie wymaga żadnej interwencji operatora, a wprowadzenie segmentu z taśmy trwa dość długo, głównie z powodu czynności operatorskich (p. 1.10). Tak więc segment działający długo lub wprowadzany jednokrotnie w czasie działania programu może bez szkody dla sprawności obliczeń pozostać na taśmie, a segment wprowadzany wielokrotnie do pamięci i działający krótko należy koniecznie umieścić w pamięci bębnowej; w przeciwnym razie maszyna będzie bardzo często i długo czekać na wykonanie niezbędnych operacji taśmowych.

UWAGA

Program używający segmentów powinien być przetłumaczony za pomocą tego samego wydania translatora, którego użyto do wyprodukowania segmentów.

ROZDZIAŁ 2. Rozstrzygnięcia w ALGOLu 1204 ważniejszych niejednoznaczności ALGOLu 60

2.0. Uwagi ogólne

Od chwili opublikowania opisu ALGOLu 60 (tzw. Raport, p. Dodatek A w [1]) przedmiotem rozważań specjalistów były pewne niedoskonałości tego opisu uniemożliwiające jednoznaczną interpretację niektórych konstrukcji algolowskich. Sprawy te, jakkolwiek bez większego znaczenia, powinny być rozstrzygnięte w każdej konkretnej realizacji ALGOLu 60. W tym rozdziale omawiamy tylko najbardziej - naszym zdaniem - istotne dla praktyki rozstrzygnięcia niejednoznaczności ALGOLu 60, przyjęte w ALGOLu 1204. Czytelnika zainteresowanego pełną analizą niejednoznaczności ALGOLu 60 odsyłamy do pracy [2] i do prac tam cytowanych. Czytelnik traktujący ALGOL 1204 wyłącznie jako system programowania, a nie przedmiot nieco spekulatywnych rozważań, znajdzie w każdym przypadku wystarczające wyjaśnienia w [1] i tutaj.

Warto dodać, że dwa ograniczenia gramatyczne ALGOLu 1204 (p. 1.0, G1 i G2) zmniejszają liczbę możliwych konstrukcji niejednoznacznych.

2.1. Zmienne z mianem own (zmienne własne)

Po ponownym wejściu do bloku, zmienne własne lokalne w tym bloku będą miały wartości takie, jak przy ostatnim wyjściu z tego bloku. Dotyczy to również przypadku, kiedy blok stanowi część opisu procedury. Identyczność zmiennych własnych jest ustalona na cały czas wykonywania programu (por. przykład).

Początkowe wartości prostych zmiennych własnych typu integer lub real są określone i równe odpowiednio 0 lub 0.0; należy to rozumieć w ten sposób, że bezpośrednio przed uruchomieniem programu od początku, zmiennym wymienionego rodzaju maszyna automatycznie nadaje wartości równe zero. Wartości początkowe pozostałych zmiennych nie są określone.

PRZYKŁAD

Podany dalej program może służyć jako ilustracja powyższych wyjaśnień. W wyniku działania tego programu maszyna wydrukuje

ciąg liczb 1,2,3,4,5 (w formacie standardowym - p. 3.16).

```
begin
 procedure P(k);
  value k;
  integer k;
  begin
   own integer N;
   N:=N+1;
   begin
    own integer array A[1:N];
    A[N]:=N;
    if k>0
     then P(k-1)
     else print(A)
   end
  end P;
 P(4)
end
```

2.2. Obszar działania nazw standardowych

Opis ALGOLu 60 pozostawia m. in. następujące wątpliwości:

a. Czy nazwę zastrzeżoną, np. sqrt, można opisać ponownie, np. jako zmienną całkowitą?
b. Jaki jest obszar działania etykiety umieszczonej przed programem, lub we wnętrzu instrukcji złożonej tworzącej program?

Jednoznaczne odpowiedzi na te pytania wynikają z podanych dalej informacji oraz z definicji obszaru działania nazwy w ALGOLu 60:

A. W ALGOLu 1204 niektóre nazwy, zwane nazwami standardowymi, mają ustalone znaczenie; w tym znaczeniu nazwy te mogą być używane bez opisywania (wykaz nazw standardowych znajduje się w skorowidzu).
B. Każdy program tłumaczony przez translator jest automatycznie zanurzany w standardowym bloku, w którego nagłówku są opisane wszystkie nazwy standardowe.

Tak więc opisanie nazwy standardowej w programie pozbawia ją (lokalnie) standardowego znaczenia, a etykiety o własności z (b) są lokalne w bloku standardowym wymienionym w (B); z (B) wynika

również, że etykiety o własności z (b) muszą mieć nazwy różne od wszystkich nazw standardowych.

2.3. Instrukcja warunkowa

W ALGOLu 60 pewne wątpliwości budzi działanie instrukcji warunkowej postaci:

```
if ⟨wyrażenie boolowskie WB⟩
 then ⟨instrukcja bezwarunkowa Ib⟩
 else ⟨instrukcja I⟩
```

(zwanej potocznie pełną instrukcją warunkową) szczególnie przy próbie interpretacji następujących przypadków:

a. skok z zewnątrz do wnętrza instrukcji Ib lub I;
b. skok z wnętrza Ib do I lub na odwrót.

W celu wyjaśnienia tych wątpliwości zauważmy najpierw, że dwie inne postacie instrukcji warunkowej, mianowicie

```
if WB then Ib;
if WB then ⟨instrukcja "dla"⟩
```

są dobrze określone w opisie ALGOLu 60, z czego skorzystamy.

W ALGOLu 1204 (także w innych realizacjach ALGOLu 60) instrukcja postaci

```
if WB then Ib else I
```

działa tak, jak ciąg instrukcji postaci:

```
if WB
 then begin
       Ib;
       go to NEXT
      end;
I;
NEXT:
```

Jeśli I jest także pełną instrukcją warunkową, to należy ją interpretować podobnie. Tak więc instrukcję postaci

```
if WB1
 then Ib1
 else if WB2
       then Ib2
       else if WB3
```

```
                then Ib3
                else I
```

należy interpretować dokładnie tak, jak instrukcje

```
      if WB1 then begin Ib1; go to NEXT end;
      if WB2 then begin Ib2; go to NEXT end;
      if WB3 then begin Ib3; go to NEXT end;
      I;
NEXT:
```

Schemat ten, odpowiadający ściśle sposobowi tłumaczenia pełnej instrukcji warunkowej przez translator, rozstrzyga m. in. o interpretacji instrukcji skoku wymienionych w (a) i (b) powyżej.

2.4. Wyrażenia warunkowe

Wartość W wyrażenia warunkowego postaci

```
  if ⟨wyrażenie boolowskie WB⟩
   then ⟨proste wyrażenie arytmetyczne PWA⟩
   else ⟨wyrażenie arytmetyczne WA⟩
```

maszyna oblicza według schematu odpowiadającego instrukcji:

```
  if WB
   then W:=PWA
   else W:=WA
```

(p. 2.3). Analogicznie oblicza się wartość warunkowego wyrażenia boolowskiego i desygnującego.

Wyrażenie postaci

```
  if WB then PWA else WA
```

jest typu integer tylko wtedy, jeśli oba wyrażenia PWA, WA są typu integer, a w przeciwnym razie wyrażenie jest typu real.

2.5. Wyrażenia proste

Jeżeli obliczanie wartości A przesądza już o wartości wyrażenia A op B, gdzie op jest dowolnym operatorem dwuargumentowym, to także i w tym przypadku maszyna oblicza wartość B i wykonuje czynności związane z wykonaniem działania op.

PRZYKŁAD

Jeśli w czasie wykonywania instrukcji

```
if a<.5Vq/a>eps
 then go to L
```

mamy a=0, to maszyna zatrzyma się (sygnał REAL OVERFLOW, p.11.3) wskutek dzielenia przez zero przy obliczaniu wartości relacji q/a>eps; z punktu 2.4 wynika, że obliczania tej relacji w przypadku a=0 unikniemy zastępując podaną instrukcję przez

```
if if a<.5 then true else q/a>eps
 then go to L
```

Zauważmy, że jeśli tylko obliczanie wyrażenia nie jest związane z jakimiś efektami ubocznymi, to postępowanie zilustrowane przykładem jest zawsze możliwe, bez zmiany wyników programu (program będzie na ogół działać szybciej).

2.6. Instrukcja "dla"

Instrukcja "dla" w ALGOLu 60 jest bardzo rozbudowana, a jej opis jest na tyle niekompletny, że nawet stosunkowo proste (choć zwykle bez praktycznego znaczenia) przykłady można interpretować na dziesiątki sposobów (por. 4 interpretacje "liberalne" i 23 "konserwatywne" przykładu na str. 13 w [2]).

Ogólnie biorąc, realizacja instrukcji "dla" w ALGOLu 1204 jest taka, że żaden przykład z książki [1] nie jest z tą realizacją sprzeczny. Dla wyjaśnienia istotnych dla praktyki szczegółów, niniejszy punkt podzielimy na części, których tytuły odpowiadają kłopotliwym szczegółom definicji instrukcji "dla".

D1. Wykaz "dla" z wieloma elementami wykazu "dla"

Instrukcja postaci

```
for Z:=E1,E2,...,En do I
```

gdzie Ei (i=1,2,...,n) oznacza jeden element wykazu "dla", działa tak, jak ciąg instrukcji postaci:

```
for Z:=E1 do I;
for Z:=E2 do I;
......
for Z:=En do I
```

D2. Wyjściowa wartość zmiennej sterowanej

Po zakończeniu wykonywania instrukcji "dla" postaci

```
for Z:=a step b until c do I
```

lub

for Z:=a while B do I

wartość zmiennej sterowanej jest równa pierwszej takiej wartości, dla której instrukcja "dla" już nie jest wykonana - w sensie definicji podanych w [1]. Na przykład, po wykonaniu instrukcji

for Z:=1 step 1 until 3 do I

wartość Z jest równa 4, a po wykonaniu ciągu instrukcji

Z:=0; for Z:=Z+1 while Z<3 do I

wartość Z jest równa 3 (w obu przypadkach przy założeniu, że instrukcja I nie zmienia wartości Z).

Jeżeli E jest wyrażeniem arytmetycznym, to instrukcja

for Z:=E do I

działa tak, jak ciąg instrukcji

Z:=E; I

Z podanych informacji i D1 wynika, że instrukcja

for Z:=1 step 1 until 3,Z+1 while Z<7 do I

gdzie I nie zmienia Z, spowoduje wykonanie I dla Z=1,2,3,5,6.

D3. Zmienna sterowana ze wskaźnikami

Jeżeli zmienna sterowana ma wskaźniki, to wartości tych wskaźników oblicza się przed każdym badaniem, czy należy wykonać instrukcję sterowaną. Tak więc instrukcje

i:=1; for x[i]:=5,10,a-b,sin(a) do i:=i+1

mają taki sam skutek, jak instrukcje

x[1]:=5; x[2]:=10; x[3]:=a-b; x[4]:=sin(a); i:=5

D4. Element postaci A step B until C

Z punktu 4.642 Raportu wynika, że instrukcja postaci

for V:=A step B until C do S

działa tak, jak instrukcje

```
      V:=A;
LOOP:if (V-C)×sign(B)>0 then go to FINISH;
      S; V:=V+B; go to LOOP;
FINISH: ......
```

Zauważmy najpierw, że ta definicja jest jednoznaczna tylko przy następujących założeniach:

z1. V jest zmienną prostą.

z2. B jest stałą lub zmienną prostą.

Jeśli bowiem nie jest spełnione z1 lub z2, to Raport nie daje jednoznacznej odpowiedzi na co najmniej jedno z następujących pytań:

p1. Ile razy obliczać wskaźniki zmiennej V?
p2. Ile razy obliczać wartość B?
p3. Jak interpretować ewentualne skutki uboczne obliczania wskaźników zmiennej V, oraz wartości B i C?

Aby odpowiedzieć na te pytania, wprowadzimy funkcję ref (p. 3.3) taką, że ref(V) jest adresem komórki pamięci, w której jest zapamiętana wartość zmiennej V, oraz tablicę mem oznaczającą pamięć maszyny, przy czym mem[K] oznacza zawartość komórki o adresie K. Przyjmujemy, że liczba mem[ref(V)] jest równa wartości V i ma typ zgodny z typem V. Przy tych oznaczeniach instrukcja postaci

```
for V:=A step B until C do S
```

w ALGOLu 1204 działa tak, jak ciąg instrukcji postaci:

```
        p:=ref(V); mem[p]:=A; q:=B; go to CHECK;
LOOP:   p:=ref(V); q:=B; mem[p]:=mem[p]+q;
CHECK:  r:=C; if (mem[p]-r)xsign(q)>0 then go to FINISH;
        S; go to LOOP;
FINISH:.....
```

gdzie p jest typu integer, a typy V,q,r są takie same. Zauważmy, że jeśli są spełnione założenia z1 i z2, to podana definicja jest równoważna definicji zawartej w Raporcie, a odpowiedzi na pytania p1-p3 są jednoznaczne także bez założeń z1 i z2. Widać na przykład, że jeżeli instrukcja S była wykonana N razy, to wartość A była obliczona 1 raz, a wskaźniki zmiennej V oraz wartości B i C obliczano N+1 razy. Program (por. [2], str. 13)

```
begin
 integer procedure I(s);
  string s;
  begin
   I:=1; K:=K+1;
   print(s,'␣'); comment p. rozdz. 3;
  end I;
 array V,A,B,C[1:1];
 integer K;
```

```
A[1]:=B[1]:=1;
C[1]:=3;
K:=0;
format('K=12?');
line(1);
for V[I('V')]:=A[I('A')] step B[I('B')] until C[I('C')]
   do print(K);
print('KONIEC:␣',K)
end
```

wydrukuje następujące wyniki:

```
V A B C K= 4
V B C K= 7
V B C K=10
V B C KONIEC: K=13
```

Należy dodać, że podaną tutaj definicję elementu postaci A step B until C sformułowano dla przypadku najogólniejszego. W przypadkach szczególnych translatory ALGOLu 1204 stosują równoważne, ale znacznie uproszczone postacie tej definicji, co oznacza na przykład, że jeżeli są spełnione założenia z1 i z2, to organizacja pętli jest bardzo sprawna (p. 5.1 i 5.5).

D5. Skok z zewnątrz do wnętrza instrukcji "dla"

Skutki skoku z zewnątrz do wnętrza instrukcji "dla" w ALGOLu 60 nie są określone. W ALGOLu 1204 skok taki można stosować, ale tylko pod warunkiem, że wyjście z instrukcji sterowanej nastąpi także przez skok; w przeciwnym razie skutków skoku do wnętrza instrukcji "dla" nie można przewidzieć (może nastąpić dezorganizacja programu lub nawet systemu obsługi).

2.7. Powtórzenia w zbiorze parametrów formalnych

W ALGOLu 60 nie zabrania się konstruować opisów procedur zaczynających się na przykład tak:

```
procedure P(q,q1,q2,q);
```

przy czym nie jest określony sens powtórnego użycia tej samej nazwy w zbiorze parametrów.

Translatory ALGOLu 1204 konstrukcje takie sygnalizują jako błędne (q REPEATED, p. 11.2).

2.8. Zgodność parametru aktualnego ze specyfikacją parametru formalnego

Opis ALGOLu 60 żąda wymienionej w tytule tego punktu zgodności nie precyzując jednak bliżej sensu słowa "zgodność". W ALGOLu 1204 wymaga się zgodności parametrów formalnych i aktualnych w sensie podanej niżej tabeli:

Specyfikacja parametru formalnego	Rodzaj i typ parametru aktualnego
integer	wyrażenie arytmetyczne typu integer
real	wyrażenie arytmetyczne typu real
Boolean	wyrażenie boolowskie
integer array	nazwa tablicy typu integer
real array, array	nazwa tablicy typu real
Boolean array	nazwa tablicy typu Boolean
integer procedure	nazwa funkcji typu integer
real procedure, procedure	nazwa funkcji typu real lub nazwa procedury (p. 2.9 i 2.10)
Boolean procedure	nazwa funkcji typu Boolean
label	wyrażenie desygnujące
switch	nazwa przełącznika
string	łańcuch lub parametr formalny o specyfikacji string

Jedynymi dopuszczalnymi wyjątkami od tej tabeli są parametry aktualne procedur standardowych (p. 3.0).

Niezgodność zbioru parametrów aktualnych ze zbiorem parametrów formalnych jest sygnalizowana w czasie działania programu (sygnał PARAMETER LIST).

PRZYKŁAD 1

Jeżeli opis procedury zaczyna się tak, jak poniżej

```
procedure P(x);
 value x;
 real x;
 ......
```

to instrukcja P(1) jest błędna, ponieważ parametr aktualny jest typu integer. Należy tę instrukcję zmienić na P(1.0). Jeżeli K jest zmienną typu integer, to instrukcja P(K) jest również błędna; można ją zastąpić na przykład poprawną instrukcją P(K+.0).

PRZYKŁAD 2

Jeżeli nagłówek procedury porządkowania ciągu

A[1],A[2],...,A[n]

ma postać

```
procedure sort(n,A);
 value n;
 integer n;
 integer array A;
```

to instrukcji tej procedury można użyć tylko do porządkowania ciągu zapamiętanego w tablicy typu integer. Aby uporządkować ciąg zapamiętany w tablicy typu real, należy specyfikację parametru A zmienić na

```
array A
```

Przy sposobności zwracamy uwagę na to, że drobne zmiany tego rodzaju łatwo wprowadzić za pomocą tzw. tłumaczenia z korygowaniem (p. 8.7).

2.9. Parametr o specyfikacji label w zbiorze wartości

W ALGOLu 1204 parametru o specyfikacji label nie można umieścić w zbiorze wartości. ALGOL 60 sprawy tej nie precyzuje.

2.10. Nazwa wartości funkcji w roli instrukcji procedury

Użycie nazwy wartości funkcji w roli instrukcji procedury jest dopuszczalne i powoduje zwykłe przejście do wykonania treści funkcji. Na użytek czytelnika interesującego się kodem wewnętrznym maszyny (p. rozdz. 6) podajemy, że po wykonaniu takiej instrukcji wartość funkcji znajduje się w akumulatorze maszyny (w A - jeśli funkcja jest typu integer lub Boolean, w AW - jeśli funkcja jest typu real), a rejestry warunków opisują stan akumulatora tak, jak po wykonaniu rozkazu pobrania. Jeżeli w czasie wykonywania treści funkcji nie nadano jej wartości, to zawartość akumulatora nie jest określona.

2.11. Wartość procedury

W ALGOLu 1204 opisy lub specyfikacje zaczynające się od symbolu procedure oznaczają to samo, co odpowiednie opisy lub specyfikacje zaczynające się od real procedure (analogia do array

i real array). Tak więc wartość procedury jest liczbą typu real; wartość ta jest określona, jeśli w treści procedury wykonano podstawienie pod nazwę procedury.

Stąd i z treści p. 2.10 wynika, że w ALGOLu 1204 nie odróżnia się procedur od funkcji, co jest w pełni zgodne z gramatyką ALGOLu 60 (por. Raport, p. 5.4). Jedynym wyjątkiem są tzw. procedury specjalne (p. 3.0, 6.5-6.12).

W tym podręczniku użyto terminu "funkcja" tylko w odniesieniu do standardowych funkcji matematycznych (p. 3.2). Jednak również i tych funkcji można użyć do utworzenia instrukcji procedury; sens tej konstrukcji wynika z treści p. 2.10.

2.12. Podstawienie pod nazwę procedury

W ALGOLu 1204 podstawienie pod nazwę procedury może wystąpić tylko w treści tej procedury. Tak więc fragment programu (zaczerpnięty z [2]):

```
integer procedure A; A:=B:=0;
integer procedure B(k); if k>0 then A else B:=2
```

poprawny pod każdym względem w ALGOLu 60, jest błędny w ALGOLu 1204 (sygnał B:= OUT OF PROCEDURE BODY w czasie tłumaczenia, p. 11.2).

2.13. Kolejność obliczania parametrów w zbiorze wartości

Kolejność obliczania parametrów umieszczonych w zbiorze wartości jest zgodna z ich kolejnością w zbiorze parametrów formalnych. Kolejność parametrów w zbiorze wartości lub w zbiorze specyfikacji nie ma żadnego znaczenia.

2.14. Skutki uboczne treści procedur

W ALGOLu 1204 nie nakłada się żadnych ograniczeń na rodzaj skutków ubocznych w procedurach. Może używać w nich nazw nielokalnych, w szczególności wykonywanie treści procedury można przerwać dowolnym skokiem prowadzącym poza treść procedury.

Interpretacja skutków ubocznych w ALGOLu 1204 jest w pełni zgodna z Raportem. Stwierdzenie to nie rozstrzyga jednak interpretacji niektórych możliwych skutków ubocznych, ponieważ Raport nie określa na przykład porządku obliczania wartości wyrażeń pierwotnych, chociaż bardzo dokładnie określono w nim porządek wykonywania działań i role nawiasów.

PRZYKŁAD 1

Niżej podajemy program (zaczerpnięty z [2]), który ma co najmniej 10 różnych interpretacji; żadna z tych interpretacji nie jest sprzeczna z Raportem:

```
begin
 integer a;
 integer procedure f(x,y);
  value y,x;
  integer y,x;
  a:=f:=x+1;
 integer procedure g(x);
  integer x;
  x:=g:=a+2;
 a:=0;
 print(a+f(a,g(a))/g(a))
end
```

Studiując Raport i przytoczony przykład można stwierdzić, że po ustaleniu kolejności obliczania wartości parametrów umieszczonych w zbiorze wartości i kolejności obliczania wartości wyrażeń pierwotnych liczba drukowana przez podany program będzie określona jednoznacznie.

Wobec treści p. 2.13 wystarczy podać kolejność obliczania wartości wyrażeń w ALGOLu 1204. Translatory ALGOLu 1204 stosują ściśle kolejność "od lewej do prawej" opisaną w [1] (rozdział 12 i 16) z następującym wyjątkiem związanym ze skutkami ubocznymi:

(R) Jeżeli program - analizowany ściśle "od lewej do prawej" zawiera wyrażenie postaci

Z op W

gdzie

Z - jest zmienną prostą (ale nie parametrem formalnym nie umieszczonym w zbiorze wartości),

op- jest dowolnym operatorem dwuargumentowym,

W - jest wyrażeniem, którego wartość trzeba obliczyć przed wykonaniem operacji op, przy czym obliczenie W powoduje zmianę wartości Z,

to do wykonania operacji op bierze się zmienioną wartość Z.

Wynika z tego, że program podany w przykładzie 1 drukuje liczbę równą 10/3. Jeżeli wyrażenie

```
a+f(a,g(a))/g(a)
```

występujące w tym programie zastąpić wyrażeniem

```
(a)+f(a,g(a))/g(a)
```

to wynik jest równy 1/3, gdyż to ostatnie wyrażenie nie ma własności omawianej w (R). Wartość 1/3 otrzymuje się również stosując do któregokolwiek z tych dwóch wyrażeń reguły omawiane w [1], rozdział 12 i w p. 2.13 tego rozdziału.

Własność (R) wynika z wykonywanej przez translatory ALGOLu 1204 optymalizacji programu, która polega m. in. na zmniejszeniu liczby zmiennych roboczych wprowadzanych przez translator i wykorzystaniu wielu spośród operacji maszyny. Kolejność wykonywania działań pozostaje przy tym taka, jak opisano w Raporcie.

PRZYKŁAD 2

Jeżeli program zawiera opisy

```
Boolean a,b; real u,v,w,y,z
```

to przekład instrukcji

```
if aVbΛ.0>1.0+uxv↑w/y+z
 then y:=u+v/w
```

jest dokładnie taki sam, jak przekład instrukcji

```
if v↑wxu/y+1.0+z<.0ΛbVa
 then y:=v/w+u
```

PRZYKŁAD 3

Po wykonaniu fragmentu programu

```
begin
 procedure f;
  f:=a:=b:=a+1.0
 real a,b,c;
 a:=5.0;
 c:=b+axf;
 ......
```

wartość zmiennej c będzie równa 42 (6x6+6), gdyż przekład instrukcji podstawienia c:=b+axf jest taki sam, jak przekład in-

strukcji c:=fxa+b. Jeżeli omawianą instrukcję zastąpimy instrukcją c:=(b)+(a)xf, to zmienna c nie ma określonej wartości, zgodnie z rozdziałem 12 w [1].

ROZDZIAŁ 3. Procedury i zmienne standardowe

3.0. Uwagi ogólne

W ALGOLu 1204 rozróżnia się dwa rodzaje procedur standardowych:

a. Procedury standardowe, których nazwy mogą być używane do tworzenia takich samych konstrukcji, jak nazwy procedur niestandardowych (p. 2.10 i 2.11); w szczególności nazw tych można używać jako parametrów aktualnych odpowiadających parametrom formalnym specyfikowanym jako procedury z odpowiednim typem. Typ i wartość każdej procedury standardowej tego rodzaju są podane w następnych punktach tego rozdziału.

b. Procedury standardowe, zwane dalej procedurami specjalnymi, do których należą cztery procedury omówione na końcu tego rozdziału (copy, exch, todrum, fromdrum) i procedury odpowiadające rozkazom kodu wewnętrznego maszyny ODRA 1204 (p. rozdział 6); specjalna własność tych procedur polega na tym, że nazwa procedury specjalnej może być użyta tylko do utworzenia instrukcji procedury, przy czym translator sprawdza tylko składnię tej instrukcji. Z tego powodu semantycznie błędne instrukcje procedur specjalnych mogą spowodować dezorganizację programu, a nawet systemu obsługi.

Liczbę i rodzaj parametrów aktualnych procedur standardowych translator sprawdza w czasie tłumaczenia programu i w razie potrzeby zmienia automatycznie typ z integer na real (lub odwrotnie). Jeśli natomiast wywołanie procedury standardowej następuje w wyniku odwołania do parametru formalnego oznaczającego nazwę procedury, to wykaz parametrów aktualnych takiego wywołania jest sprawdzany w czasie wykonywania programu; w tym przypadku żąda się takiej zgodności parametrów, jak dla procedur niestandardowych.

PRZYKŁAD

Jeśli fragment programu ma postać

```
procedure F(G);
 Boolean procedure G;
 begin
  if G(7.2) then go to E;
  ......
 end F;
.....
F(key);
.....
```

to w czasie działania tego programu zostanie zasygnalizowany błąd (p. 2.8), ponieważ parametrowi formalnemu G odpowiada nazwa procedury standardowej key, której parametrem aktualnym ma być wyrażenie typu integer (p. 3.4).

3.1. Postać liczb czytanych przez program

Program w ALGOLu 1204 może czytać dane z monitora albo z taśmy założonej do czytnika (p. 3.5). W obu przypadkach obowiązują te same reguły pisania liczb.

Składnia liczb czytanych przez program jest taka sama, jak składnia liczb występujących w programach (p. 1.0, S3). Pojedyncze odstępy między dwoma kolejnymi cyframi są przy czytaniu danych pomijane, co umożliwia grupowanie cyfr w liczbach. Symbolem końca liczby może być zmiana wiersza, dwa kolejne odstępy, dowolny symbol oznaczający koniec liczby w programie, a także symbol odstępu ␣. Niepotrzebne symbole końca liczby i wszystkie inne znaki niewidoczne w maszynopisie są pomijane przez program.

Specjalną rolę pełni znak zapytania. Po jego przeczytaniu program drukuje na monitorze znak zapytania i czeka na napisanie na monitorze dowolnego znaku, po czym kontynuuje obliczenia. Znak zapytania służy więc jako symbol końca taśmy w przypadku, gdy wygodnie jest pisać dane na kilku taśmach. Niezależnie od tego znak zapytania jest jednym z możliwych symboli końca liczby.

Jeżeli program wykonuje instrukcję czytania liczby typu integer (p. 3.6 i 3.7) i czytana liczba jest typu real, to program zatrzymuje się (sygnał NUMBER, p. 11.3).

Z powodów opisanych w p. 11.6, każda taśma z danymi powinna zawierać na końcu co najmniej 16 znaków pomijanych, na przykład odstępów. Z tych samych powodów przygotowane na jednej taśmie

kolejne komplety danych, dla których program jest uruchamiany od początku przez operatora, też powinny być oddzielone znakami pomijanymi.

3.2. Standardowe funkcje matematyczne

Oprócz funkcji standardowych ALGOLu 60 o nazwach

abs, sqrt, sin, cos, arctan, exp, ln, sign, entier

w ALGOLu 1204 można także używać bez opisywania następujących funkcji o wartościach typu real:

tan - tangens (argument w radianach),
arcsin - arcus sinus (wartość w radianach),
arccos - arcus cosinus (wartość w radianach).

Parametrem aktualnym każdej z tych funkcji może być dowolne wyrażenie arytmetyczne typu real.

Algorytmy obliczania wartości standardowych funkcji matematycznych, z wyjątkiem funkcji abs, sign i entier, mają ograniczoną dokładność. Szczegóły zawiera podana niżej tabela:

Nazwa wartości funkcji w ALGOLu 1204	Błąd względny wartości (w symbolice matematycznej)
sqrt(a)	2^{-37}
ln(a)	$\max(2^{-36}, 2^{-37}/\ln a)$
exp(a)	$\max(\|a\|\times 2^{-37}, 2^{-35})$
arcsin(a), arccos(a), arctan(a)	2^{-35}
sin(a), cos(a), tan(a)	$\max(\|a\|\times 2^{-37}, 2^{-37})$

Argumenty funkcji, z wyjątkiem funkcji abs, arctan i sign, muszą spełniać pewne szczególne warunki. Ich niespełnienie jest wykrywane i sygnalizowane w czasie działania programu utworzonego przez translator (p. 11.3). Szczegóły podano w tabeli poniżej:

Nazwa wartości funkcji	Wartość argumentu	Sygnał zatrzymania
sqrt(a)	a < 0	SQRT
ln(a)	a ≤ 0	LN
exp(a)	a > 510.99999998 ln 2	EXP

Nazwa wartości funkcji	Wartość argumentu	Sygnał zatrzymania
arcsin(a), arccos(a)	$\|a\| > 1$	ARC
sin(a), cos(a)	$\|a\| > \pi \times 2^{38}$	TRIG
tan(a)	$\|a\| > \pi \times 2^{38} \vee \cos a = 0$	TRIG
entier(a)	$a > 2^{23}-1 \vee a < -2^{23}$	RI CONVERSION

3.3. Procedura ref

Procedura ref jest typu integer i ma jeden parametr, którym może być zmienna dowolnego typu. Wykonanie treści tej procedury powoduje podstawienie pod nazwę ref adresu komórki zarezerwowanej na zmienną (parametr aktualny). Dla zmiennej rzeczywistej jest to adres drugiej komórki z dwóch kolejnych komórek zajmowanych przez tę zmienną.

Procedura ref jest użyteczna w programach zawierających kod wewnętrzny maszyny ODRA 1204 (p. rozdz. 6).

3.4. Procedura key

Wartością procedury typu Boolean

key(i)

jest true, jeśli i-ty klawisz pulpitu sterowania maszyny jest wciśnięty. W przeciwnym razie procedura ma wartość false. Wartość parametru aktualnego (typu integer) musi należeć do przedziału ⟨0,23⟩.

3.5. Procedura setinput - wybór rodzaju wejścia

Programy napisane w ALGOLu 1204 mogą czytać dane z monitora maszyny i z taśmy założonej do czytnika, a programy uruchamiane na zestawie D maszyny - także z taśmy założonej do drugiego czytnika. Jeżeli program czyta dane z monitora, to bezpośrednio przed czytaniem liczby całkowitej, liczby rzeczywistej lub łańcucha program automatycznie drukuje na monitorze odpowiednio literę i, r lub s, po czym zapalają się lampki PISZ na pulpicie monitora. Przed czytaniem pojedynczego znaku program nie drukuje żadnej informacji; zapalają się jedynie lampki PISZ. Dane na monitorze można pisać tylko w tych momentach pracy programu, kiedy palą się lampki PISZ.

Aby odróżnić wymienione wyżej rodzaje wejść w programach, przyporządkowano im w ALGOLu 1204 numery w następujący sposób:

Numer wejścia	Wejście
0	monitor
1	czytnik taśmy
2	drugi czytnik taśmy (tylko zestaw D)

Wszystkie instrukcje czytania odnoszą się do tzw. wejścia aktualnego. Wejście aktualne jest zawsze ustawiane na początku wykonywania programu zależnie od zlecenia uruchamiającego program (p. 11.1).

Do ustawienia wejścia aktualnego w programie służy procedura typu real

```
setinput(N)
```

gdzie N jest wyrażeniem arytmetycznym typu integer o wartości równej jednemu z numerów wejścia. Wykonanie treści tej procedury powoduje zmianę aktualnego wejścia na wejście o numerze N i podstawienie wartości .0 pod nazwę setinput.

3.6. Procedura read

Instrukcja postaci

```
read(p1,p2,...,pn)
```

gdzie n jest dowolną liczbą naturalną, służy do czytania liczb z wejścia aktualnego i podstawiania ich pod odpowiednie zmienne. Każdy parametr aktualny p1,p2,...,pn może być jednym z następujących elementów:

1° zmienną całkowitą,
2° zmienną rzeczywistą,
3° nazwą tablicy całkowitej,
4° nazwą tablicy rzeczywistej.

Procedura read ma typ real. Wykonanie jej treści polega na tym, że najpierw wykonuje się wszystkie czynności związane z czytaniem p1, następnie wszystkie czynności związane z czytaniem p2 itd. Czynności związane z czytaniem kolejnego parametru aktualnego zależą od rodzaju tego parametru w następujący sposób:

Rodzaj parametru	Czynności wykonywane w treści procedury
Zmienna całkowita	Czytanie liczby całkowitej (p. 1.0), podstawienie jej pod parametr aktualny i pod zmienną standardową lastinteger (p. 3.19), podstawienie kodu znaku kończącego liczbę pod zmienną standardową lastchar, podstawienie wartości .0 pod nazwę read.
Zmienna rzeczywista	Czytanie liczby rzeczywistej (p. 1.0), podstawienie jej pod parametr aktualny i pod zmienną standardową lastreal, podstawienie kodu znaku kończącego liczbę pod zmienną standardową lastchar, podstawienie wartości .0 pod nazwę read.
Nazwa tablicy całkowitej	Wykonanie czynności związanych z czytaniem zmiennej całkowitej dla wszystkich leksykograficznie kolejnych elementów tablicy.
Nazwa tablicy rzeczywistej	Wykonanie czynności związanych z czytaniem zmiennej rzeczywistej dla wszystkich leksykograficznie kolejnych elementów tablicy.

3.7. Procedury ininteger i inreal

Bezparametrowe procedury ininteger i inreal są odpowiednio typu integer i real. W treści każdej z tych procedur czyta się z aktualnego wejścia liczbę odpowiedniego typu. Tę liczbę podstawia się pod nazwę procedury i pod właściwą ze zmiennych standardowych lastinteger albo lastreal (p. 3.19), a pod zmienną standardową lastchar podstawia się kod znaku kończącego przeczytaną liczbę.

Zastosowanie procedur ininteger i inreal pozwala skrócić programy, w których czytanych liczb nie trzeba pamiętać, gdyż są one używane jednorazowo do obliczania wartości wyrażeń (p. 4.9).

3.8. Procedura inchar

Bezparametrowa procedura inchar jest typu integer. Wykonanie treści tej procedury powoduje przeczytanie jednego znaku z aktualnego wejścia i podstawienie kodu tego znaku pod nazwę inchar i pod zmienną standardową .lastchar. Przy pomocy proce-

dury inchar można czytać wszystkie znaki dostarczane przez czytnik taśmy i monitor; w szczególności przeczytanie znaku zapytania nie powoduje w tym przypadku zawieszenia pracy programu (p. 3.1).

Kody znaków maszyny OPTIMA i ich transkrypcję na drukarce wierszowej podano w p. 11.0.

3.9. Procedura instring - czytanie łańcucha

Procedura instring jest typu integer i ma jeden parametr, który musi być zmienną całkowitą ze wskaźnikami. Wykonanie instrukcji

instring(A[i1,i2,...,in])

gdzie A jest nazwą tablicy całkowitej, powoduje wykonanie następujących czynności:

1° pominięcie na aktualnym wejściu wszystkich znaków aż do najbliższego otwierającego nawiasu łańcuchowego ' ,

2° przeczytanie łańcucha i zapamiętanie go w komórkach pamięci maszyny zarezerwowanych na leksykograficznie kolejne elementy tablicy A począwszy od elementu leksykograficznie następnego po A[i1,i2,...,in], po trzy znaki w jednej komórce; jeżeli z1, z2 i z3 są kodami kolejnych znaków łańcucha zapamiętanymi w jednej komórce, to zawartość tej komórki jest równa

(z1x256+z2)x256+z3,

3° podstawienie pod zmienną A[i1,i2,...,in] wartości równej liczbie znaków przeczytanego łańcucha bez skrajnych nawiasów łańcuchowych,

4° podstawienie pod nazwę instring wartości równej liczbie komórek (kolejnych elementów tablicy A) zawierających przeczytany łańcuch,

5° podstawienie pod zmienną standardową lastchar kodu nawiasu łańcuchowego zamykającego.

Jeżeli czytany łańcuch nie mieści się w tablicy A, to program zatrzymuje się (sygnał STRING, p. 11.3). Przykład zastosowania procedury instring jest podany w p. 4.10.

3.10. Procedura setoutput - wybór rodzaju wyjścia

Wyniki obliczeń mogą być wyprowadzane przez program na monitor maszyny albo na taśmę ośmiokanałową, której treść można później wydrukować na maszynie do pisania niezależnej od maszyny

cyfrowej. Programy uruchamiane na zestawie D maszyny mogą także wyprowadzać wyniki na drukarkę wierszową i na taśmę ośmiokanałową dziurkowaną przez drugi perforator. Termin "drukowanie wyników" oznacza tutaj dowolny z wymienionych sposobów wyprowadzania wyników.

W ALGOLu 1204 wyjścia maszyny są ponumerowane w następujący sposób:

Numer wyjścia	Wyjście
0	monitor
1	perforator taśmy
2	drugi perforator taśmy (tylko zestaw D)
3	drukarka wierszowa (tylko zestaw D)

Wszystkie instrukcje drukowania umieszczone w programie odnoszą się do tzw. wyjścia aktualnego. Wyjście aktualne jest zawsze ustawiane na początku wykonywania programu zależnie od zlecenia uruchamiającego program (p. 11.1).

Do ustawienia wyjścia aktualnego w programie służy procedura typu real

setoutput(N)

gdzie N jest wyrażeniem typu integer o wartości równej jednemu z numerów wyjścia. Wykonanie treści tej procedury powoduje, że wyjściem aktualnym staje się wyjście o numerze N, a pod nazwę setoutput podstawia się wartość .0.

UWAGA

Jeżeli w czasie drukowania wyników jest wciśnięty klawisz nr 0, to niezależnie od wykonywanych instrukcji ustawiania wyjścia, wyjściem aktualnym jest perforator taśmy (p. 11.4). Po zwolnieniu klawisza nr 0 wyniki są drukowane na tym wyjściu, które było ostatnio wybrane jako wyjście aktualne.

3.11. Procedura outchar - drukowanie jednego znaku

Wykonanie instrukcji

outchar(N)

gdzie N jest wyrażeniem arytmetycznym typu integer, powoduje wydrukowanie na aktualnym wyjściu znaku o kodzie równym wartości parametru aktualnego N (p. 11.0) i podstawienie wartości .0 pod nazwę outchar. Procedura outchar ma typ real, a wartości pa-

rametru aktualnego tej procedury muszą należeć do przedziału ⟨0, 127⟩.

3.12. Procedura space - drukowanie odstępów

Wykonanie instrukcji

```
space(N)
```

gdzie N jest wyrażeniem arytmetycznym typu integer, powoduje wydrukowanie na aktualnym wyjściu N odstępów i podstawienie wartości .0 pod nazwę space. Procedura space ma typ real. Jeśli wartość wyrażenia N nie jest dodatnia, to instrukcja space(N) jest równoważna instrukcji pustej.

3.13. Procedura line - drukowanie zmian wiersza

Wykonanie instrukcji

```
line(N)
```

gdzie N jest wyrażeniem arytmetycznym typu integer, powoduje wydrukowanie na aktualnym wyjściu trzech odstępów, znaku nowej linii i N-1 znaków wysuwu papieru, oraz podstawienie wartości .0 pod nazwę line. Trzy odstępy są drukowane z powodu własności znaku "nowa linia", który na niektórych egzemplarzach maszyny do pisania OPTIMA nie działa, jeżeli karetka jest na początku wiersza.

Procedura line ma typ real. Jeśli wartość wyrażenia N nie jest dodatnia, to instrukcja line(N) jest równoważna instrukcji pustej.

3.14. Procedura outstring - drukowanie łańcucha

Procedura outstring jest procedurą typu integer z jednym parametrem, który musi być zmienną całkowitą ze wskaźnikami. Wykonanie instrukcji

```
outstring(A[i1,i2,...,in])
```

gdzie A jest nazwą tablicy całkowitej, powoduje wykonanie następujących czynności:

1° wydrukowanie na aktualnym wyjściu łańcucha, którego początek (liczba znaków łańcucha) znajduje się w komórce zarezerwowanej na zmienną A[i1,i2,...,in], a następne leksykograficznie kolejne elementy tablicy A zawierają kolejne trójki znaków

tego łańcucha (p. 3.9),

2^0 podstawienie pod nazwę outstring wartości równej liczbie komórek zajętych przez łańcuch.

UWAGA

Jeśli we wnętrzu łańcucha występuje znak zapytania, to zamiast drukowania tego znaku program wykonuje instrukcję line(1) (p. 3.13), a zamiast symbolu odstępu ␣ program drukuje odstęp. Ta sama uwaga dotyczy drukowania łańcuchów za pomocą procedury print (p. 3.15) i drukowania tekstów występujących we wzorcu druku liczb (p. 3.16). Tak więc znak zapytania i symbol odstępu mogą być wydrukowane przez program tylko za pomocą instrukcji outchar (p. 3.11).

3.15. Procedura print

Instrukcja

```
print(p1,p2,...,pn)
```

gdzie n jest dowolną liczbą naturalną, służy do drukowania wyników na wyjściu aktualnym. Każdy parametr aktualny p1,p2,..., pn może być

1^0 wyrażeniem arytmetycznym,

2^0 nazwą tablicy arytmetycznej,

3^0 łańcuchem.

Procedura print jest typu real, a wykonanie jej treści polega na tym, że najpierw wykonuje się wszystkie czynności związane z drukowaniem p1, następnie wszystkie czynności związane z drukowaniem p2, itd. Czynności związane z drukowaniem kolejnego parametru aktualnego zależą od rodzaju tego parametru w następujący sposób:

Rodzaj parametru aktualnego	Czynności wykonywane w treści procedury
Wyrażenie arytmetyczne	Obliczenie wartości wyrażenia, wydrukowanie jej według aktualnego wzorca (p. 3.16), podstawienie wartości .0 pod nazwę print.
Nazwa tablicy arytmetycznej	Wykonanie czynności związanych z drukowaniem wartości wszystkich leksykograficznie kolejnych elementów tablicy.

Łańcuch	Wydrukowanie łańcucha (p. 3.14, UWAGA) i podstawienie wartości .0 pod nazwę print.

3.16. Procedura format - wybór wzorca druku liczb

Instrukcja

format(S)

gdzie S jest łańcuchem, a procedura format ma typ integer, określa postać dalej drukowanych liczb, zależną od S (nie powoduje natomiast drukowania żadnego znaku) i nadaje nazwie format wartość 0. Łańcuch S nazywamy wzorcem druku. Postać liczb określona wzorcem druku obowiązuje aż do wykonania następnej instrukcji ustawiania wzorca, albo do ponownego uruchomienia programu.

Wnętrze łańcucha S nie może być zupełnie dowolne. Jego postać ma związek ze składnią liczb. Aby opisać ten związek, podamy najpierw ogólne reguły obowiązujące we wnętrzu wzorca:

r1. Znaki niewidoczne w maszynopisie nie mają wpływu na interpretację wzorca i można ich używać dowolnie.

r2. Wzorzec druku ma postać

'T0 L1 T1 L2 T2 ... Ln Tn'

gdzie L1,L2,...,Ln (n>0) są liczbami, a ciągi znaków T0,T1,T2,...,Tn nie zawierają żadnego ze znaków

\+ - . ␣ 0 1 2 3 4 5 6 7 8 9

Ciągi takie będziemy nazywać tekstami. Teksty T0 Tn mogą nie zawierać żadnego znaku, a pozostałe teksty muszą zawierać co najmniej jeden znak widoczny różny od symbolu odstępu ␣ lub co najmniej dwa symbole odstępu ␣␣.

r3. Wzorzec postaci podanej w r2 oznacza, że w dalszym ciągu działania programu kolejne liczby drukuje się w następujący sposób:

r30. bezpośrednio przed drukowaniem pierwszej liczby drukuje się tekst T0;

r31. dla i=1,2,...,n drukuje się liczbę o numerze i w postaci analogicznej do postaci liczby Li, po czym drukuje się tekst Ti;

r32. liczbę o numerze n+1 drukuje się tak, jak liczbę o numerze 1, liczbę o numerze n+2 - jak liczbę o numerze 2,

itd; wzorzec działa zatem cyklicznie, przy czym jeden cykl zawiera n kolejno drukowanych liczb, licząc od momentu ustawienia wzorca druku.

r4. Znaki ␣ ? mają we wzorcu analogiczne znaczenie, jak w łańcuchu (p. 3.14, UWAGA).

r5. Dodatkowo - pojedynczy symbol odstępu ␣ użyty między dwiema cyframi oznacza odstęp wewnątrz liczby i nie rozpoczyna tekstu; umożliwia to drukowanie liczb z grupowaniem cyfr.

PRZYKŁAD

Przypuśćmy, że w programie jest opisana tablica

```
array a[0:2,1:2]
```

przypuśćmy dalej, że leksykograficznie kolejne elementy tej tablicy mają wartości odpowiednio:

3.5, -1.55, 8.976, -3.14, 2.78, 6.6666666

Wykonanie instrukcji

```
format('?a[1,1]␣␣-1.123␣456');
for i:=0,1,2 do for j:=1,2 do print(i,j,a[i,j])
```

spowoduje wydrukowanie następujących wyników:

```
a[0,1] =  3.500 000
a[0,2] = -1.550 000
a[1,1] =  8.976 000
a[1,2] = -3.140 000
a[2,1] =  2.780 000
a[2,2] =  6.666 667
```

W podanym przykładzie wzorzec druku składa się z następujących części:

T0	L1	T1	L2	T2	L3	T3
?a[	1	,	1	]␣␣	-1.123␣456	

Tekst T3 nie zawiera tutaj żadnego znaku.

Tak więc wzorzec druku składa się ze wzorców liczb i tekstów. Wzorzec liczby jest liczbą. W pojedynczym wzorcu liczby można w oczywisty sposób wyróżnić wzorce fragmentów liczb, np. wzorzec znaku, mantysy, cechy, cyfry, kropki, dziesiątki algolowskiej itd. - analogicznie, jak w gramatyce ALGOLu wyróżnia się niektóre fragmenty liczb. W dalszym ciągu opiszemy własności poszcze-

gólnych fragmentów wzorca liczby.

Znak liczby może być wydrukowany na kilka różnych sposobów, zależnie od wzorca znaku. Brak wzorca znaku liczby oznacza, że znak liczby nie będzie wydrukowany przed liczbą nieujemną. Wzorzec znaku - (minus) oznacza, że liczby ujemne będą poprzedzone znakiem -, a liczby nieujemne będą miały na początku odstęp zamiast znaku. Wzorzec znaku + oznacza, że każda liczba będzie poprzedzona właściwym ze znaków + lub -.

Jeżeli pierwsza cyfra we wzorcu liczby jest zerem, to wszystkie liczby drukowane według tego wzorca będą miały nieznaczące początkowe zera przed kropką wydrukowane jako zera; w przeciwnym razie nieznaczące początkowe zera zostaną zastąpione odstępami, a znak liczby będzie wydrukowany bezpośrednio przed pierwszą wydrukowaną cyfrą, przed kropką albo przed cechą.

Wzorzec liczby nie może zawierać żądania drukowania poza cechą więcej, niż 12 cyfr, a wzorzec cechy liczby może zawierać co najwyżej 3 wzorce cyfry. Przy drukowaniu cyfr cechy wszystkie zera są drukowane jako zera, a mantysa różna od zera ma zawsze pierwszą cyfrę różną od zera.

Wzorzec liczby postaci

$$_{10}+12$$

i inne podobne powodują wydrukowanie liczby bez mantysy (ale z jej znakiem, jeżeli wzorzec go zawiera), a cechę drukuje się tak, jak gdyby mantysa M spełniała warunek

$$0.1 \leq |M| < 1 \quad (\text{dla } M \neq 0).$$

Wydrukowana w ten sposób cecha jest oczywistą informacją o rzędzie wielkości liczby. Dla M=0 cechę drukuje się jako liczbę równą zeru.

Jeżeli wzorzec liczby nie zawiera wzorca znaku, a liczba jest taka, że znaku nie można opuścić (przed całą liczbą, albo wewnątrz cechy), to liczbę drukuje się według aktualnego wzorca uzupełnionego w niezbędny sposób wzorcem znaku - (jednym lub dwoma); aktualny wzorzec nie zmienia się.

Jeżeli wielkość liczby uniemożliwia jej wydrukowanie zgodnie z aktualnym wzorcem, to wzorzec ten pomija się, a następnie drukuje się w nowej linii znak zapytania i samą liczbę zgodnie ze wzorcem

$$'-1.123\_456\_789_{10}+123'$$

Aktualny wzorzec nie zmienia się. W ten sposób w przypadku niezgodności liczby ze wzorcem nigdy nie traci się informacji, ale układ graficzny wyników zostanie naruszony.

Niekiedy zamiast drukowania liczby według kolejnego wzorca liczby jest wykonywana instrukcja

print('?UNDEFINED␣␣')

Oznacza to, że program zawiera drukowanie zmiennej typu real, której reprezentacja maszynowa nie jest liczbą zmiennoprzecinkową; może się to zdarzyć, jeśli wartość tej zmiennej nie została określona lub wykonano na niej niewłaściwe operacje za pomocą procedur specjalnych (p. 3.0, b). W obu przypadkach jest to błąd programowania.

Długość wzorca druku jest ograniczona. Jeżeli wzorzec zawiera n+1 tekstów T0,T1,...,Tn i n liczb L1,L2,...,Ln, a przez d(Ti) oznaczymy długość i-tego tekstu liczoną w znakach, to wzorzec musi spełniać warunek

$$3\times n+d(T0)\div 3+d(T1)\div 3+ \ldots +d(Tn)\div 3 \leq 60$$

Wynika stąd, że wzorzec druku może zawierać co najwyżej 20 wzorców liczb; jeżeli n=20, to każdy z tekstów T0,T1,...,Tn może zawierać co najwyżej 2 znaki.

Bezpośrednio przed uruchomieniem programu jest zawsze wykonywana instrukcja

format('?-1.123␣456␣789$_{10}$+123')

Tak więc program może nie zawierać ustawiania wzorca; w tym przypadku otrzymuje się wyniki w postaci kolumny liczb, każda z 10-cyfrową mantysą i 3-cyfrową cechą.

Jeżeli w czasie wykonywania programu zostanie wykryty błąd we wzorcu druku, to program zatrzymuje się (sygnał FORMAT, p. 11.3).

3.17. (D) Drukarka wierszowa

Drukarka wierszowa ma 63 znaki. Niektóre znaki klawiatury A maszyny CPTIMA nie mają odpowiedników na drukarce, a niektóre znaki drukarki nie mają odpowiedników na maszynie CPTIMA. Z tego powodu jest niemożliwe ustalenie w pełni zadowalającego przyporządkowania kodów znaków w ALGOLu 1204 znakom drukarki (p. 11.0). W szczególności jest niemożliwe prawidłowe wydrukowanie tekstu

algolowskiego na drukarce wierszowej. Jednak wyniki typowych programów obliczeniowych będą miały na drukarce postać w pełni równoważną postaci tych wyników wydrukowanych na maszynie OPTIMA lub na monitorze.

Jeśli w czasie działania programu wyjściem aktualnym (p. 3.10) jest drukarka wierszowa, to kolejne znaki, które mają utworzyć wiersz na drukarce umieszcza się w pamięci maszyny. Wyprowadzenie wiersza na drukarkę (tzn. przepisanie zapamiętanych znaków do pamięci drukarki i rozpoczęcie drukowania wiersza) następuje po każdym z wymienionych niżej zdarzeń:

a. Wykonanie instrukcji line(N). Wykonanie tej instrukcji bezpośrednio po takiej samej instrukcji powoduje wyprowadzenie N pustych wierszy; drukowanie wiersza pustego polega na wyprowadzeniu trzech odstępów i wysunięciu papieru o jeden wiersz.
b. Wykonanie instrukcji outchar(NL), gdzie NL jest jedną z liczb 15, 47, 79, 111 (są to kody znaków zmiany wiersza).
c. Wypełnienie wiersza, tzn. stan, w którym długość ciągu kolejno drukowanych znaków, nie zawierającego zmiany wiersza, osiąga wartość 120; wyprowadza się wtedy wiersz zawierający 119 znaków, a znak o numerze 120 będzie pierwszym znakiem w następnym wierszu.
d. Przejście do ostatniego symbolu end w programie.
e. Wykonanie zlecenia "end" (p. 11.1).

Tak więc używanie drukarki nie nakłada żadnych specjalnych wymagań co do sposobu pisania tych części programu, które drukują wyniki. W projektowaniu postaci wyników dla drukarki użyteczne są następujące informacje:

A. W jednym wierszu drukarki może wystąpić co najwyżej 119 znaków widocznych lub odstępów.
B. Typowy papier zakładany do drukarki jest łamany na przemian w jedną i w drugą stronę. Pomiędzy dwoma załamaniami papieru mieści się 66 wierszy.
C. Początkowe położenie papieru w drukarce można ustawiać za pomocą manipulacji ręcznych. W szczególności można papier tak ustawić, aby pierwszy wiersz był wydrukowany bezpośrednio pod zgięciem papieru.
D. System obsługi pomija pod każdym względem sygnał końca forma-

tu papieru, nadawany z czytnika mechanicznego drukarki.

Ze względu na sposób działania instrukcji line(N) (p. 3.13) zaleca się, aby żaden wiersz wyników drukowanych przez program nie zawierał więcej, niż 116 znaków. Jeżeli jest potrzebne wykorzystanie wszystkich 119 miejsc wiersza drukarki, to zmianę wiersza trzeba wykonywać za pomocą instrukcji procedury outchar (por. b powyżej).

3.18. Procedury wait i stop - zatrzymanie programu

Do zatrzymania pracy programu służą dwie procedury standardowe typu integer: wait i stop. Wykonanie instrukcji

```
wait(S)
```

gdzie S jest łańcuchem, powoduje wydrukowanie na monitorze (a jeśli zerowy klawisz pulpitu jest wciśnięty, to na perforatorze) sygnału postaci

```
wait S
```

a następnie przeczytanie jednego znaku z monitora (maszyna czeka na napisanie znaku przez operatora) i podstawienie kodu tego znaku pod nazwę wait.

Procedury wait używa się do chwilowego zatrzymania pracy programu. Łańcuch S może zawierać instrukcje dla operatora (np. oznaczenie taśmy z danymi, którą należy założyć do czytnika), a znak przeczytany z monitora może być wykorzystany do sterowania dalszym ciągiem pracy programu (np. może być numerem wariantu, według którego mają być wykonane obliczenia).

Wykonanie instrukcji

```
stop
```

powoduje przerwanie pracy programu i wydrukowanie sygnału zatrzymania STOP (p. 11.3). Po napisaniu zlecenia "go" (p. 11.5) pod nazwę stop podstawia się wartość zmiennej standardowej time (p. 3.19) i program kontynuuje obliczenia.

Używając procedury stop można na przykład skrócić fragmenty programu, które wykrywają błędy w danych. Instrukcja

```
if abs(Det(A,n))<eps
 then print('?macierz␣A␣osobliwa',stop)
```

działa podobnie, jak instrukcja

```
if abs(Det(A,n))<eps
 then begin
       print('?macierz_A_osobliwa');
       stop
      end
```

W pierwszym przypadku po napisaniu zlecenia "go" program wydrukuje aktualną wartość zmiennej time i przejdzie do wykonania następnej instrukcji programu. W drugim przypadku nastąpi tylko przejście do wykonania następnej instrukcji.

Bardziej użytecznym zastosowaniem procedury stop jest zatrzymanie programu w wybranym z góry momencie obliczeń po to, aby stan tych obliczeń przechować (w pamięci bębnowej lub na taśmie binarnej - p. 11.5) w celu późniejszej kontynuacji. Czynność taką można wykonać również bez użycia w programie procedury stop, ale wtedy moment przerwania obliczeń musi wybrać operator maszyny.

3.19. Zmienne lastchar, lastinteger, lastreal, time, drumplace

W ALGOLu 1204 można używać bez opisywania zmiennych standardowych lastchar, lastinteger, lastreal, time, a na zestawach D maszyny - także zmiennej drumplace. Wszystkie informacje o tych zmiennych zawiera tabela poniżej:

Nazwa zmiennej	Typ	Wartość początkowa	Wartość w czasie działania programu
lastchar	integer	0	kod ostatnio przeczytanego znaku
lastinteger	integer	0	ostatnio przeczytana liczba całkowita
lastreal	real	.0	ostatnio przeczytana liczba rzeczywista
time	integer	nie określona	wartość jest zwiększana o 1 po upływie każdej sekundy pracy programu i translatora
drumplace	integer	-1	zwiększony o 1 adres bębnowy ostatniego elementu zapisywanego lub odczytywanego z bębna (p. 3.21)

Wartości zmiennych standardowych można także zmieniać w programie (np. za pomocą instrukcji podstawienia). Przykłady używania tych zmiennych są podane w następnych rozdziałach.

3.20. Procedury copy i exch

W maszynie ODRA 1204 są zrealizowane między innymi dwie operacje szybkiego kopiowania i zamiany ciągu słów maszynowych. Operacje te wykorzystuje się w treści procedur specjalnych (p. 3.0) copy i exch, służących odpowiednio do kopiowania i zamiany ciągów leksykograficznie kolejnych elementów tablic.

Procedury copy i exch mają po trzy parametry, z których pierwszy jest wyrażeniem arytmetycznym, a dwa pozostałe muszą być zmiennymi ze wskaźnikami. Typ ostatniego parametru aktualnego decyduje o tym, jaką długość mają elementy kopiowane albo zamieniane (dla typu real - elementy zapamiętane w dwu kolejnych komórkach pamięci maszyny - p. 1.7). Typ drugiego parametru aktualnego nie jest uwzględniany.

Wykonanie instrukcji

```
copy(N,A[i,...],B[j,...])
```

powoduje skopiowanie N leksykograficznie kolejnych elementów tablicy A, począwszy od elementu A[i,...], do tablicy B, począwszy od elementu B[j,...].

Wykonanie instrukcji

```
exch(N,A[i,...],B[j,...])
```

powoduje zamianę N leksykograficznie kolejnych elementów tablicy A, począwszy od elementu A[i,...], z N elementami tablicy B, począwszy od elementu B[j,...].

PRZYKŁAD 1

Jeżeli w programie znajduje się opis

```
array A[1:N]
```

to instrukcja

```
copy(N-1,A[2],A[1])
```

jest równoważna instrukcji

```
for i:=2 step 1 until N do
 A[i-1]:=A[i]
```

W treściach procedur copy i exch nie sprawdza się, czy wszystkie elementy, na których wykonuje się manipulacje są elementami wskazanych tablic.

PRZYKŁAD 2

Jeżeli tablice A i B mają opis

```
array A,B[1:N]
```

to instrukcja

```
exch(2xN,A[1],B[1])
```

nie będzie sygnalizowana jako błędna, ale po jej wykonaniu program może działać źle (dezorganizacja).

3.21. (D) Procedury todrum i fromdrum

Procedury specjalne (p. 3.0) todrum i fromdrum służą do pamiętania na bębnie i odczytywania z bębna ciągu leksykograficznie kolejnych elementów tablicy. Procedury te mają po dwa parametry, z których pierwszy jest wyrażeniem arytmetycznym określającym liczbę elementów, a drugi jest zmienną ze wskaźnikami oznaczającą pierwszy element ciągu. Z procedurami todrum i fromdrum jest związana zmienna standardowa drumplace (p. 3.19), której wartość jest interpretowana jako adres bębnowy.

Wykonanie instrukcji

```
todrum(N,T[I,...])
```

powoduje zapisanie w pamięci bębnowej, poczynając od komórki o adresie równym wartości zmiennej drumplace, N leksykograficznie kolejnych elementów tablicy T, poczynając od elementu T[I,...]. Wykonanie tej czynności zmienia wartość zmiennej drumplace tak, jak wykonanie instrukcji

```
drumplace:=drumplace+K
```

gdzie

$$K = \begin{cases} N, & \text{jeżeli tablica T jest typu } \underline{\text{integer}} \text{ lub } \underline{\text{Boolean}}, \\ 2xN, & \text{jeżeli tablica T jest typu } \underline{\text{real}}. \end{cases}$$

Do czytania danych umieszczonych w pamięci bębnowej służy instrukcja postaci

```
fromdrum(N,T[I,...])
```

która działa analogicznie do instrukcji procedury todrum.

Za pomocą omówionych wyżej procedur w programie można używać tylko obszaru roboczego pamięci bębnowej (p. 10.5), to znaczy komórek o adresach 0,1,2,...,D-1, gdzie D jest adresem pierwszej komórki obszaru bębnowego chronionego przez system (p. 11.5, "drum"). Wartość D jest zawsze określona po przetłumaczeniu programu, a zapisanie segmentu na bębnie (p. 1.11) zmniejsza tę wartość o liczbę komórek zajmowanych przez segment (p. 7.2). Jeśli przed wykonaniem instrukcji todrum(N,T[I,...]) lub fromdrum(N,T[I,...]) nie są spełnione następujące warunki:

1° drumplace≥0,
2° drumplace+K≤D,
3° N>0,
4° liczba leksykograficznie kolejnych elementów tablicy T, od elementu T[I,...] do elementu ostatniego, jest nie mniejsza od N,

to program zatrzymuje się (sygnał DRUM PARAMETER, p. 11.3).

Przykład stosowania procedur todrum i fromdrum jest podany w p. 4.10.

ROZDZIAŁ 4. Przykłady programowania w ALGOLu 1204

4.0. Uwagi ogólne

Ponieważ ograniczenia ALGOLu 1204 (p. 1.0) nie są zbyt krępujące, więc znane z podręczników ALGOLu 60 przykłady programowania są ważne w ALGOLu 1204. W tym rozdziale omawiamy przykłady innego rodzaju; ilustrują one przede wszystkim specyficzne możliwości ALGOLu 1204, związane m. in. z czytaniem danych i drukowaniem wyników. Czytelnik zauważy, że przykłady te sugerują określone sposoby programowania; takie sugestie zawiera również rozdział 5.

Dalsze przykłady związane tylko z kodem wewnętrznym maszyny, czytelnik znajdzie w rozdziale 7.

4.1. Warunkowe drukowanie wyników pośrednich

Niekiedy zdarza się, że w programie występują zmienne, których wartości są oczywistą informacją o prawidłowości lub postępie obliczeń - na przykład mówią o zbieżności procesu iteracyjnego. Niech taką zmienną będzie Z. Jeżeli wewnątrz pętli iteracji umieścimy instrukcję

```
if key(23)
 then begin
       setoutput(0);
       format('?Z=+.1234₁₀+12');
       print(Z)
      end
```

to dysponujemy możliwością warunkowego wyprowadzenia na monitor lub perforator (p. 11.4) wartości zmiennej Z wraz z nazwą tej zmiennej.

4.2. Czytanie danych z powtórzeniami

Niektóre dane zawierają długie ciągi takich samych elementów. Warto w takim przypadku pisać dane w specjalny sposób, aby uniknąć wielokrotnego dziurkowania tych samych liczb. Można na przykład przyjąć, że podany niżej ciąg liczb na taśmie z danymi

```
5,1,1,1,1,2,0,0,0,0,0,3,-6;
```

wolno napisać tak

```
5,1x4,2,0x5,3,-6;
```

Powtarzającą się liczbę kończymy znakiem mnożenia, a dalej piszemy liczbę powtórzeń. Program, który ma czytać dane pisane w ten sposób powinien zawierać opis odpowiedniej procedury czytania:

```
real procedure CL;
 begin
  own integer k; comment p. 2.1, wartości początkowe;
  own real r;
   if k≤0
    then begin
           read(r);
           if lastchar=26
            then read(k)
          end k≤0;
   k:=k-1;
   CL:=r
  end CL
```

Czytanie wartości jakiejś zmiennej Z wykonujemy w programie za pomocą instrukcji podstawienia Z:=CL, a nie za pomocą procedur standardowych. Zauważmy, że wtedy dane zapisane w zwykły sposób są także czytane dobrze, a symbole takie jak 5x1, 5x0, 5x-1 są traktowane tak samo, jak liczba 5 zakończona przecinkiem.

Usunięcie któregokolwiek z symboli own w treści podanej procedury pozbawia ją sensu (p. 2.1).

4.3. Czytanie danych logicznych

W programach napisanych w ALGOLu 1204 nie można czytać wartości logicznych za pomocą procedur standardowych. Stałe logiczne true i false są przy czytaniu danych traktowane jako symbole końca liczby (p. 3.1). Podana niżej procedura służy do czytania wartości logicznych zapisanych za pomocą pojedynczych znaków 1 lub 0 odpowiadających wartościom true lub false. Wszystkie inne znaki są pomijane.

```
Boolean procedure CWL;
E:if inchar=1
   then CWL:=true
   else if lastchar=16
         then CWL:=false
         else go to E
```

Wartości logiczne można czytać za pomocą instrukcji podstawienia

```
B:=CWL
```

gdzie B jest zmienną typu Boolean.

4.4. Drukowanie liczby całkowitej z minimalną ilością cyfr

Podana niżej procedura drukuje liczbę całkowitą z minimalną ilością cyfr nie zmieniając ustalonego w programie wzorca druku liczb.

```
procedure dlc(l);
 value l;
 integer l;
 begin
  integer j,k;
  if l<0
   then begin
         outchar(32);
         l:=-l;
         if l<0
          then begin
                print('8388608');
                go to E
               end
        end l<0;
   if l=0
    then outchar(16)
    else begin
          for j:=10↑entier(ln(l)x.434294482),
                j÷10 while j≠0 do
           begin
            k:=l÷j; l:=l-kxj;
            outchar(if k=0 then 16 else k)
           end j;
          end;
E:end dlc
```

Opis tej procedury napisany z zastosowaniem procedur specjalnych jest podany w p. 7.5.

4.5. Przykład operowania ciągiem o nieznanej liczbie elementów

Taśma z danymi zawiera ciąg liczb

$$X1, X2, \ldots, Xn$$

Liczba n nie jest podana, ale wiadomo, że ostatnia i tylko ostatnia liczba na taśmie jest zakończona średnikiem. Należy obliczyć i wydrukować liczbę

$$\frac{1}{n}\sum_{1}^{n} X_i^2 - \left(\frac{1}{n}\sum_{1}^{n} X_i\right)^2$$

Rozwiązanie może mieć następującą postać:

```
begin
 real Sx,Sxx;
 integer n;
 n:=-1;
 Sx:=Sxx:=.0;
 for n:=n+1 while lastchar≠90 do
  begin
   Sx:=Sx+inreal;
   Sxx:=Sxx+lastrealXlastreal
  end n;
 print(Sxx/n-(Sx/n)↑2)
end
```

4.6. Porównanie dwóch metod porządkowania

Podajemy tutaj dwie istotnie różne procedury porządkowania zredagowane dla najprostszego przypadku porządkowania od "najmniejszej do największej" ciągu liczb całkowitych

$$A[1], A[2], \ldots, A[n]$$

W obu procedurach celowo zachowano pewne drobne nieoptymalności (np. until n-1 i until n+m); usunięcie tych nieoptymalności daje praktycznie niedostrzegalny zysk czasowy i zwiększa zarówno długość tekstu algolowskiego, jak i jego przekładu. Oto zapowiedziane procedury.

```
procedure SHORT SORT(A,n);
 value n;
 integer n;
 integer array A;
 begin
  integer i,j;
  for i:=1 step 1 until n-1 do
   for j:=i+1 step 1 until n do
    if A[i]>A[j] then exch(1,A[i],A[j])
 end SHORT SORT;

procedure FAST SORT(A,n);
 value n;
 integer n;
 integer array A;
 begin comment metoda Shella;
  integer i,j,m;
  m:=n;
  for m:=m÷2 while m<0 do
   for j:=1 step 1 until n+m do
    begin
     for i:=j step m until 1 do
      if A[i]≤A[i-m]
       then go to endj
       else exch(1,A[i],A[i-m]);
endj:end j,m
 end FAST SORT;
```

Obie procedury sprawdzono m. in. dla danych postaci
n,n-1,n-2,...,1
i otrzymano następujące wyniki:

Procedura	Czas działania w sekundach		
	n=200	n=400	n=800
SHORT SORT	79	318	1274
FAST SORT	5	12	26

Należy jeszcze wyjaśnić, dlaczego w treści procedury FAST SORT nie użyto nieco bardziej naturalnych instrukcji:

```
m:=n; for m:=m÷2 while m>0 do .....
        for i:=j step -m until 1 do    itd...
```

Chodzi o to, że jeżeli pomiędzy symbolami step i until znajduje się wyrażenie różne od stałej lub zmiennej prostej (p. 5.5), to organizacja pętli jest znacznie mniej sprawna, niż wynika to z dołączenia operacji zmiany znaku.

4.7. Pomiar czasu trwania operacji bardzo szybkich

Podany niżej program

```
begin
 integer i,k,m,n;
 format('T(for␣k)=999,␣T(i:=k)=99');
 time:=0;
 for k:=1 step 1 until 1 000 000 do;
 m:=time;
 time:=0;
 for k:=1 step 1 until 1 000 000 do i:=k;
 n:=time-m;
 print(m,n)
end
```

po upływie ok. 4.5 minut pracy wydrukuje wyniki

T(for k)=116, T(i:=k)=38

albo wyniki nieznacznie mniejsze lub większe (do 10%) zależnie od egzemplarza maszyny i jego regulacji. Wydrukowane wartości wyrażają odpowiednio czas (w mikrosekundach) organizacji pętli najczęściej spotykanego rodzaju i czas wykonania instrukcji i:=k dla zmiennych całkowitych, z błędem bezwzględnym nie przekraczającym odpowiednio 2 i 4.

4.8. Pomiar czasu bez wielokrotnego powtarzania operacji

Do szybkiego i dokładnego pomiaru czasu trwania bardziej złożonych czynności może służyć procedura timing opracowana przez Z. Cylkowskiego (Uniwersytet Wrocławski). Oto opis tej procedury:

```
real procedure timing(w,t); real w,t;
 begin integer k,m,t0,t1;
  t0:=time;
  for m:=0 while time=t0 do;
  timing:=w; t1:=time;
  for k:=3, k+1 while time=t1 do;
  t1:=time;
```

```
    for m:=m+1 while time=t1 do;
    t:=t1-t0-1-k/m
  end timing;
```

Chcąc zmierzyć np. czas T (zmienna typu real) wykonania instrukcji procedury postaci

P(x,y,...)

wystarczy napisać w programie instrukcję

timing(P(x,y,...),T)

Tak samo można zmierzyć czas obliczania wartości wyrażenia typu real; wartość tej procedury jest równa wartości wyrażenia, co można wykorzystać do połączenia obliczeń sprawdzających metodę numeryczną z pomiarem czasu. Jeżeli na przykład sprawdzamy algorytm obliczania wartości procedury F(x,y), to zamiast instrukcji

print(F(x,y))

można w programie użyć instrukcji

print(timing(F(x,y),T),T)

co spowoduje wydrukowanie wartości F i czasu obliczania tej wartości.

Błąd bezwzględny opisanego sposobu pomiaru czasu jest równy 0.0013. Wykonując instrukcję timing(.0,T) 3500 razy otrzymano wartości T spełniające nierówność

$$-.000854 \leq T \leq .000850$$

4.9. Drukowanie wykresu rodziny funkcji

Podany niżej program drukowania wykresu funkcji zawiera wiele przykładów użycia procedur i zmiennych standardowych. Niektóre instrukcje występujące w tym programie są napisane nieoptymalnie, na przykład zawierają dodawania wartości procedury print, czyli zera. W tym przypadku nie jest to istotne, gdyż czas działania instrukcji czytania lub drukowania zależy głównie od szybkości urządzeń zewnętrznych, a taki sposób pisania instrukcji może zwiększyć czytelność programu.

Aby skorzystać z programu należy przygotować dane w następujący sposób:

$$\begin{matrix} 1 & n \\ w_{1,1} & w_{1,2} \cdots w_{1,l} \\ w_{2,1} & w_{2,2} \cdots w_{2,l} \\ \cdots\cdots & \\ w_{n,1} & w_{n,2} \cdots w_{n,l} \end{matrix}$$

gdzie l jest liczbą wykreślanych funkcji, n jest liczbą punktów, w których funkcje są stablicowane, $w_{i,j}$ jest wartością i-tej funkcji w j-tym punkcie.

Jako odległość między dwoma kolejnymi punktami na osi odciętych przyjmuje się jedną zmianę wiersza, a odległości na osi rzędnych są obliczane przez program (w zależności od podanej liczby znaków w wierszu). Znaki, którymi program wykreśla funkcje, podaje się z monitora. Dodatkowo podaje się znak, którym mają być oznaczone punkty wspólne wykresów.

Dalej przytaczamy program, przykład danych, konwersację operatora z programem i wyniki programu.

```
begin
 integer n,lf,i,j,k,pw;
 read(lf,n);
 begin
  array W[1:n,1:lf];
  integer array znak,znak1,W1[1:lf];
  real min,max,skala;
  min:=10^100; max:=-10^100;
  for i:=1 step 1 until n do
   for k:=1 step 1 until lf do
    begin
     read(W[i,k]);
     if lastreal<min then min:=lastreal;
     if lastreal>max then max:=lastreal
    end k,i;
  format('?wykres␣funkcji␣numer␣123␣');
  setinput(0);
  setoutput(0);
  for i:=1 step 1 until lf do znak[i]:=print(i)+inchar;
  pw:=print('?punkty␣wspolne␣')+inchar;
  skala:=print('?liczba␣znakow␣w␣wierszu␣')+(ininteger-1)/
         (max-min);
```

```
      setoutput(print('?numen␣urzadzenia␣drukujacego␣wykres␣')
                +ininteger);
      for i:=1 step 1 until n do
       begin
        for k:=1 step 1 until lf do W1[k]:=(W[i,k]-min)xskala;
        copy(lf,znak[1],znak1[1]);
        for j:=1 step 1 until lf-1 do
         for k:=j+1 step 1 until lf do
          if W1[j]>W1[k]
           then begin
                 exch(1,W1[j],W1[k]);
                 exch(1,znak1[j],znak1[k])
                end k,j;
        j:=outchar(15);
        for k:=1 step 1 until lf do
         if if k<lf then W1[k]=W1[k+1] else false
          then znak1[k+1]:=pw
          else begin
                space(W1[k]-j);
                j:=znak1[k];
                outchar(j);
                j:=W1[k]+(if j=29Vj=93 then 0 else 1)
               end k
       end i
      end
     end
```

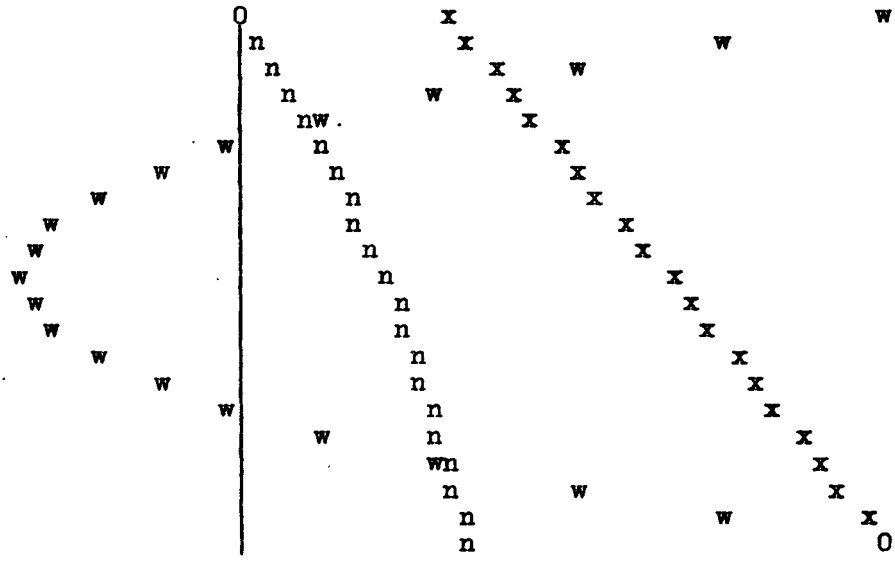
Dane do tego programu zostały wydrukowane przez następujący program:

```
     begin
      real x;
      print('4,21');
      for x:=1.5 step .1 until 3.51 do
       print(ln(x),4.0x(x-1.5)x(x-2.5),x,.0);
      space(20)
     end
```

Konwersacja programu z operatorem miała następujący przebieg:

```
wykres funkcji numer   1 n
wykres funkcji numer   2 w
wykres funkcji numer   3 x
wykres funkcji numer   4 |
punkty wspolne 0 .
liczba znakow w wierszu i55,
numer urzadzenia drukujacego wykres i1,
```

po czym program wydziurkował taśmę, po wydrukowaniu której otrzymano następujący wykres:



4.10. (D) Porządkowanie obiektów

Nazwijmy obiektem łańcuch i ciąg liczb całkowitych (cechy obiektu, liczba cech lc jest ustalona). Podany niżej program czyta liczbę lc i ciąg obiektów zakończony łańcuchem pustym, po czym drukuje obiekty i sumę wartości cech każdego z nich w ten sposób, że liczby odpowiadające sumie cech kolejno drukowanych obiektów tworzą ciąg nierosnący.

W programie obowiązują następujące ograniczenia:

$N \leq 3400$, gdzie N jest liczbą obiektów,

$\max_{1 \leq j \leq N} ((d_j+5)\div 3+1c) \leq 1000$, gdzie d_j jest liczbą znaków łańcucha numer j.

Kolejne obiekty program zapisuje w pamięci bębnowej, a działania wykonuje na wskaźnikach i adresach tych obiektów.

```
begin
 integer n,k,x,lc,j;
 Boolean B;
 integer array T[1:1000],pt,nr,s[1:3400];
 read(lc);
 drumplace:=n:=0;
 for k:=instring(T[1]) while k≠1 do
  begin
   n:=n+1; x:=0;
   for j:=1 step 1 until lc do
    begin
     x:=x+ininteger; k:=k+1;
     T[k]:=lastinteger
    end j;
   nr[n]:=n; s[n]:=x;
   pt[n]:=drumplace; todrum(k,T[1])
  end k;
 for k:=n,k-1 while B do
  begin
   B:=false;
   for j:=2 step 1 until k do
    if s[j-1]<s[j]
     then begin
           exch(1,s[j-1],s[j]);
           exch(1,nr[j-1],nr[j]);
           B:=true
          end j
  end k;
 format('-123␣');
 for j:=1 step 1 until n do
  begin
   drumplace:=pt[nr[j]]; fromdrum(1000,T[1]);
   print('?',j); k:=outstring(T[1]);
   space(25-T[1]); x:=k+lc;
   for k:=k+1 step 1 until x do print(T[k]);
   print(s[j])
  end j
end
```

Po przeczytaniu danych

```
4
'S. Kowalski - II' 1,2,3,4
'W. Kwiatkowski - X' 2,2,2,2
'L. Nowak -' -2,1,3,6
'Z. Kalinowski - V' 3,2,1,4
'A. Malinowski - VI' -3,2,2,2
''
```

program wydrukował następujące wyniki:

1	S. Kowalski - II	1	2	3	4	10
2	Z. Kalinowski - V	3	2	1	4	10
3	W. Kwiatkowski - X	2	2	2	2	8
4	L. Nowak -	-2	1	3	6	8
5	A. Malinowski - VI	-3	2	2	2	3

4.11. Przejście z klawiatury P na klawiaturę A .

Podany niżej program czyta ciąg symboli podstawowych ALGOLu 1204 i drukuje ciąg równoważny zamieniając symbole odpowiadające znakom algolowskim niedostępnym na klawiaturze P (p. 1.1) na odpowiednie znaki z klawiatury A (np. po przeczytaniu symbolu imp program wydrukuje znak ⊃, a po przeczytaniu le znaki ≤). Program nie rozpoznaje wszystkich symboli podstawowych ALGOLu, wobec czego wymaga się, aby symbole, które mają być zamienione, były oddzielone od innych przynajmniej jednym znakiem niepodkreślonym i nieprzekreślonym. Wszystkie symbole złożone z liter przekreślonych program drukuje jako symbole złożone z liter podkreślonych. Pomija on wszystkie znaki korekty występujące na czytanej taśmie. Po naciśnięciu klawisza nr 23 na pulpicie maszyny program zatrzymuje się (w wyniku wykonania procedury stop). Po zwolnieniu klawisza nr 23 i napisaniu zlecenia "go" program kontynuuje działanie.

```
begin
 integer p,P,z,j,k,Y;
 Boolean B;
 integer array ZZ,K[1:11],Z[0:2];
 integer procedure inch;
  begin
   if key(23) then stop;
 E:z:=inch:=inchar;
```

```
     if z=63Vz=127 then go to E
    end inch;
   p:=29; P:=93; j:=1;
   for ZZ[j]:=4499,4533,7059,7093,859285,4789,611091,4905,
              873493,807604,938535 do j:=j+1;
   j:=1;
   for K[j]:=59,3771,43,3755,81,11974,91,96,107,112,123 do
    j:=j+1;
E1:if inch=pVz=P
    then begin
         for Z[0]:=inch while z=pVz=P do;
         if z=70
          then begin
                print('≠'); go to E1
               end;
         if z=35Vz=55Vz=38Vz=52Vz=37Vz=53Vz=49Vz=57
          then go to L;
       E6:outchar(p); outchar(z);
       E4:if inch=pVz=P
           then begin
                for z:=inch while z=pVz=P do;
                go to E6
               end;
       E5:outchar(z); go to E1
         end;
  if z=26
    then begin
          if inch=26 then z:=109 else outchar(26)
         end;
   go to E5;
 L:Y:=z; B:=false;
   for j:=1,2 do
    if inch=pVz=P
     then begin
           for Z[j]:=inch while z=pVz=P do;
           Y:=Y×128+z
          end
     else begin
           j:=j-1; B:=true; go to C2
          end;
```

```
C2:for k:=1 step 1 until 11 do
    if ZZ[k]=Y then go to C4;
   for k:=0 step 1 until j do
    begin
     outchar(p); outchar(Z[k])
    end;
   if B
    then begin
          outchar(z); go to E1
         end;
   go to E4;
C4:k:=K[k];
   if k>127
    then begin
          j:=k÷128; outchar(j);
          k:=k-j×128
         end;
    outchar(k);
   if B then outchar(z);
   go to E1
  end
```

ROZDZIAŁ 5. Zasady optymalizacji programu

5.0. Uwagi ogólne

Optymalizacją programu nazywamy wszystkie zabiegi mające na celu skrócenie czasu wykonywania obliczeń, albo zmniejszenie długości przekładu programu (aby pozostało więcej miejsca dla danych lub dla projektowanych rozszerzeń tego programu). Dwa wymienione cele niekoniecznie są sprzeczne; nierzadko zmierzając do jednego z tych celów mimowolnie osiągamy także i drugi z nich.

Trzeba wyraźnie podkreślić, że mówimy tutaj tylko o optymalizacji za pomocą formalnych środków języka programowania, a nie o zmianach metody użytej w programie.

W tym rozdziale będziemy rozważać optymalizację czysto algolowską, tzn. bez używania kodu wewnętrznego maszyny. Najogólniejsze zasady takiej optymalizacji są dość proste i prawdopodobnie skuteczne w odniesieniu do wszystkich realizacji ALGOLu 60. Naszym zdaniem dobre nauczenie się tych zasad będzie skutkiem ubocznym nauki ALGOLu 60 ze wspomnianej już książki [1]; prawie wszystkie przykłady zawarte w [1] są zredagowane optymalnie w sensie ALGOLu 1204.

W tym rozdziale podamy pewne szczegóły ilościowe, z których wynika na przykład, jakie zabiegi optymalizacyjne są najbardziej skuteczne. Wiadomo bowiem, że w pełni zoptymalizowany program często staje się nieczytelny. W p. 4.6 czytelnik znajdzie dwa przykłady, gdzie zaniechano "groszowej" optymalizacji na rzecz zwartości i czytelności algorytmu.

Chcemy tutaj również zwrócić uwagę na to, że wiele na pozór skomplikowanych konstrukcji algolowskich odznacza się przyrodzoną optymalnością; takich konstrukcji zwykle nie warto zastępować innymi. Odpowiednie przykłady można znaleźć w następnych punktach.

W całym rozdziale czasy wykonania podajemy w mikrosekundach, a długości przekładu - w słowach maszynowych.

5.1. Ważniejsze wyniki pomiarów czasu

Niżej podajemy tabelę zawierającą czasy trwania pewnych typowych czynności wykonywanych w programach napisanych w ALGOLu 1204

W tabeli obowiązują następujące oznaczenia:

x,y,z	zmienne rzeczywiste,
i,j,k	zmienne całkowite,
B,C,D	zmienne logiczne,
a,b	nazwy tablic,
s	nazwa przełącznika
c	liczba całkowita.

Czasy te zostały zmierzone za pomocą odpowiednich programów (p. 4.7) na maszynie zainstalowanej w Centrum Obliczeniowym Uniwersytetu Wrocławskiego. Należy pamiętać, że w tabeli podano czasy średnie; dla działań, których czas trwania zależy od wartości argumentów, średnią liczono dla pewnego zbioru argumentów. Na przykład, dla instrukcji x:=y+1.0 wzięto średnią dla y=1.0,2.0, 3.0,...,100000.0.

Operacja	Średni czas wykonania w mikrosekundach
i:=j	38
x:=y	78
x:=i	327
i:=x	336
i:=j+k	54
i:=jxk	143
i:=j÷k	573
x:=y+1	1080
x:=y+1.0	248
x:=yxz	452
x:=y/z	1240
B:=x=y	457
B:=i=j	254
B:=i=x	900
B:=-C	50
B:=C∧D	54
B:=C∨D	54
odwołanie do zmiennej a[i]	567
odwołanie do zmiennej a[i,j]	890
odwołanie do zmiennej a[i,j,k]	1912
rezerwacja pamięci dla tablicy a[i:j]	1584

rezerwacja pamięci dla tablicy a[i:j,i:j] . .	2080
rezerwacja pamięci dla tablicy a[i:j,i:j,i:j] .	2570
rezerwacja pamięci dla tablic a,b[i:j]	1920
rezerwacja pamięci dla tablic a,b[i:j,i:j] . . .	2410
rezerwacja pamięci dla tablic a,b[i:j,i:j,i:j] .	2920
go to s[i]	250
obliczenie nowej wartości zmiennej sterowanej następującymi warunkami:	
for i:=j step c until k do	116
for i:=j step -c until k do	116
for i:=j step j until k do	288
for x:=y step y until z do	671
for i:=i+j while B do	130
for x:=x+y while B do	285
wywołanie procedury niestandardowej bez parametrów	1054
wywołanie procedury niestandardowej z jednym parametrem	1562
wywołanie procedury niestandardowej z dwoma parametrami	2050
odwołanie się do parametru formalnego, który nie jest nazwą tablicy, nazwą procedury ani łańcuchem	570
wyjście z procedury	164
obliczenie wartości funkcji standardowej	
abs(x)	130
sqrt(x)	1570
ln(x)	4230
exp(x)	5370
sin(x)	4190
cos(x)	4330
tan(x)	8170
arcsin(x)	8630
arccos(x)	8980
arctan(x)	5830
entier(x)	150
sign(x)	110
key(i)	320

Rezerwacja pamięci dla tablic z mianem own trwa na ogół dłużej, niż rezerwacja pamięci dla zwykłych tablic. Czas ten zależy między innymi od liczby tablic własnych w programie i od liczby elementów tych tablic.

5.2. Optymalizacja najprostszych elementów programu

Najważniejsze wnioski wynikające z budowy maszyny i translatora są następujące:

w1. Odwołania do stałych, zmiennych prostych, procedur i funkcji standardowych oraz działania na tych obiektach translator programuje najoszczędniej.

w2. Mieszanie typów integer i real w wyrażeniach i relacjach nie zawsze wydłuża program, ale znacznie zwiększa czas jego wykonania, ponieważ opęracje zmiany typu w maszynie ODRA 1204 trwają długo (p. 6.8). Ilustruje to przykład, w którym x, y oznaczają zmienne typu real.

Instrukcja	Długość przekładu	Czas wykonania
x:=y+1.0	3	248
x:=y+1	4	1080
B:=x<1.0	4	457
B:=x<1	4	900

Szczególną uwagę należy zwrócić na dzielenie / ; na przykład obliczenie 1/3 trwa ok. 2 razy dłużej niż obliczenie 1.0/3.0 i ok. 80 razy dłużej niż odwołanie się do stałej 0.333333333333.

w3. Działania na stałych i zmiennych typu real trwają znacznie dłużej od analogicznych działań na stałych i zmiennych typu integer i Boolean. Na przykład dodawanie dwóch liczb rzeczywistych może trwać nawet 10 razy dłużej od dodawania dwóch liczb całkowitych.

w4. Najszybciej maszyna wykonuje dodawania i odejmowania, znacznie wolniej - mnożenia, a najwolniej - dzielenie i potęgowanie.

w5. Translator programuje działania z uwzględnieniem wielu możliwości maszyny. Jeżeli na przykład i, j są zmiennymi całkowitymi, przy czym sign(i)=sign(j), to wartość i÷j będzie obliczona ok. 5 razy szybciej niż wartość entier(i/j).

w6. Potęgowanie liczb typu real jest operacją dość szybką. Jako przykład podajemy, że w czasie wykonywania instrukcji

```
y:=x↑(-18)
```

(5 słów przekładu) maszyna działa tak, jak w czasie wykonywania ciągu instrukcji

```
v:=1.0/x;  u:=1.0;
v:=vxv;    u:=uxv;
v:=vxv;
v:=vxv;
v:=vxv;    u:=uxv;
y:=u
```

a zatem maszyna wykonuje jedno dzielenie i sześć mnożeń.

w7. Potęgowanie liczb całkowitych dla wykładników naturalnych trwa - w porównaniu z innymi działaniami na liczbach całkowitych - bardzo długo. Zastąpienie tego działania mnożeniem zawsze przyspiesza działanie programu.

w8. Z podanych wcześniej informacji (por. w2 wyżej i 5.1) wynika, że czas obliczania relacji jest dość długi. Jeżeli więc ta sama relacja (dla tych samych wartości zmiennych) występuje w programie więcej niż jeden raz, to użycie zmiennej boolowskiej o wartości tej relacji optymalizuje program pod każdym względem. Oto przykład:

Instrukcja	Długość przekładu	Czas wykonania
if x<.005 then go to E	5	ok. 480
if B then go to E	3	ok. 40

gdzie: real x; Boolean B.

5.3. Zmienne ze wskaźnikami

Odwołanie do zmiennej ze wskaźnikami powoduje zawsze wykonanie następujących czynności:

c1. Obliczenie wartości kolejnych wskaźników.
c2. Sprawdzenie, czy liczba wskaźników jest zgodna z liczbą odpowiednich par granicznych.
c3. Sprawdzenie, czy wartość każdego wskaźnika należy do przedziału określonego odpowiednią parą graniczną.
c4. Wyznaczenie miejsca zmiennej w pamięci operacyjnej.

Wynika stąd, że błędna wartość wskaźnika nie może spowodować zniszczenia programu, ale odwołania do zmiennych ze wskaźnikami trwają dość długo (p. 5.1) i zajmują w programie więcej miejsca, niż odwołania do zmiennych prostych. Ilustruje to poniższa tabela, w której zmienne s, x i tablice a, b mają typ real, a zmienne i, j mają typ integer:

Instrukcja	Długość przekładu	Czas wykonania
s:=s+x	3	ok. 250
s:=s+a[i]	6	ok. 830
s:=s+a[i+j]	7	ok. 850
s:=s+b[i,j]	8	ok. 1150
a[i]:=a[i]+b[i,j]	14	ok. 2300

Jedynym sposobem optymalizacji fragmentów programu zawierającego zmienne ze wskaźnikami jest unikanie odwoływania się do tych zmiennych. Jeżeli na przykład instrukcję

```
I1: for i:=1 step 1 until 100 do s:=s+a[i]xa[i]
```

zastąpimy instrukcją o równoważnych skutkach

```
I2: for i:=1 step 1 until 100 do
    begin
     x:=a[i];
     s:=s+xxx
    end
```

to otrzymamy następujące wyniki:

Instrukcja	Długość przekładu	Czas wykonania
I1	22	ok. 190000
I2	21	ok. 140000

Inne przykłady dotyczące zmiennych ze wskaźnikami podano na końcu następnego punktu.

5.4. Ogólne zasady optymalizacji instrukcji

Na ogół nie warto zastępować pojedynczych instrukcji "długich" równoważnymi ciągami instrukcji "krótkich". Inaczej mówiąc, opłaca się korzystać z możliwości ALGOLu 60. Oto przykłady, o których zakładamy, że nie zawierają zmiany typu.

Instrukcje	Długość przekładu
if x>y then max:=x else max:=y	9
max:=if x>y then x else y	8
B:=if x>.0 then true else false	8
B:=x>.0	4
a:=x; b:=x; c:=x	6
a:=b:=c:=x	4
w:=axz; w:=w+b; w:=wxz; w:=w+c	12
w:=(axz+b)xz+c	6
w:=sin(a+b); print(w)	6
print(sin(a+b))	4
B:=x>.0; if B then go to E	7
if x>.0 then go to E	5
k:=p[i]; a[k]:=.0	10
a[p[i]]:=.0	8
rob:=x[i]; x[i]:=x[k]; x[k]:=rob	18
exch(1,x[i],x[k])	9
x[i]:=x[k]	8
copy(1,x[k],x[i])	9

5.5. Instrukcja "dla"

Jeżeli pominiemy sprawy omówione w p. 5.3, to jedynym sposobem optymalizacji instrukcji "dla" jest wybranie odpowiedniego warunku "dla" (p. 5.1). Zastąpienie instrukcji "dla" równoważnym układem instrukcji algolowskich innego rodzaju zawsze zwiększa czas działania programu, chociaż może nieznacznie skrócić długość przekładu. Oto przykład:

Instrukcje	Długość przekładu	Czas wykonania
for i:=1 step 1 until 100 do a[i]:=.0	17	ok. 76000
i:=0; E:if i<100 then begin i:=i+1; a[i]:=.0; go to E end	15	ok. 96000

Z tabeli podanej w p. 5.1 wynika, że najsprawniejszą organizację pętli daje warunek "dla", w którym zmienna sterowana jest typu integer, a pomiędzy symbolami step i until występuje liczba całkowita, ewentualnie ze znakiem - (minus). Najwolniej działa instrukcja, w której zmienna sterowana jest typu real. Dłuższym przekładem i wolniejszym działaniem odznaczają się również instrukcje "dla", w których pomiędzy symbolami step i until występuje wyrażenie różne od stałej lub zmiennej prostej.

Element zawierający while i dający się w naturalny sposób zastąpić elementem step - until działa zwykle wolniej od tego ostatniego.

Czas trwania czynności organizujących pętlę w instrukcji postaci

```
for i:=1 step 1 until k-1, k+1 step 1 until n do INSTR
```

jest ok. 3 razy krótszy od analogicznego czasu dla instrukcji

```
for i:=1 step 1 until n do
 if i≠k then INSTR
```

chociaż przekład pierwszej instrukcji jest o 8 słów dłuższy od przekładu drugiej z nich (jeżeli instrukcja INSTR jest pusta, to przekłady zawierają odpowiednio 24 i 16 słów).

Instrukcja postaci

```
for i:=1,2,...,n do INSTR
```

działa nieznacznie szybciej od instrukcji postaci

```
for i:=1 step 1 until n do INSTR
```

Dla n>2 przekład drugiej instrukcji jest zawsze krótszy od przekładu pierwszej z nich. Dotyczy to również innych podobnych wykazów stałych lub zmiennych prostych.

5.6. Instrukcja warunkowa i wyrażenie warunkowe

Jednym ze sposobów optymalizacji elementów wymienionych w tytule jest optymalizacja wyrażeń logicznych występujących w warunkach "jeśli", na przykład przez eliminację zbytecznych działań (p. 2.5), lub zastosowanie odpowiednich podstawień (p. 5.2, w8). Ważne (i oczywiste) znaczenie ma również wybór kolejności warunków "jeśli". W pewnych przypadkach szczególnych bardzo skuteczne jest zastosowanie przełącznika. Jeżeli na przykład instrukcja ma

postać

```
IW:if k=1 then Ib1
     else if k=2 then Ib2
      else if k=3 then Ib3
       else ......
```

to zastąpienie jej blokiem postaci

```
    begin
     switch S:=E1,E2,E3,..... ;
     go to S[k];
E1:Ib1; go to end;
E2:Ib2; go to end;
E3:Ib3; go to end;
     ..... ;
end:end
```

istotnie skraca długość przekładu i czas wykonania programu; to ostatnie wynika stąd, że obliczanie wartości relacji dla argumentów całkowitych trwa prawie tak długo, jak wykonanie instrukcji go to S[k] (p. 5.1). Analogiczny sposób można zastosować do obliczania wartości niektórych wyrażeń.

5.7. Procedury i funkcje niestandardowe

W optymalizacji programów z procedurami są użyteczne następujące informacje:

P1. Operowanie parametrami formalnymi oznaczającymi nazwy tablic i procedur jest praktycznie tak samo sprawne, jak operowanie zwykłymi tablicami i procedurami niestandardowymi (różnica wynika z P4 poniżej).

P2. Umieszczenie w zbiorze wartości parametru formalnego o specyfikacji real, integer lub Boolean optymalizuje treść procedury pod każdym względem przy założeniu, że w treści procedury odwołanie do tego parametru występuje więcej niż jeden raz.

P3. Umieszczenie parametru w zbiorze wartości nie ma wpływu na objętość przekładu instrukcji procedury (skraca się jedynie przekład treści procedury).

P4. Odwołania do nazw lokalnych w treści procedury (z wyjątkiem etykiet) trwają o 1.1 mikrosekund dłużej (czas b-modyfikacji, p. 6.3), niż analogiczne odwołania w programie głównym.

P5. Przekład instrukcji procedury lub nazwy wartości funkcji bez parametrów zajmuje 5 słów maszynowych.

P6. Przekład instrukcji procedury lub nazwy wartości funkcji z K parametrami najczęściej używanych rodzajów zajmuje nie mniej niż 4xK+6 słów maszynowych.

P7. Wywołanie procedury lub funkcji powoduje zawsze sprawdzenie zgodności zbioru parametrów formalnych ze zbiorem parametrów aktualnych, także dla zbioru pustego.

Z podanych tutaj i w p. 5.1 informacji wynika, że jeśli treść procedury jest bardzo krótka (w sensie zajmowanego miejsca i czasu wykonania), to stosowanie tej procedury może się nie opłacać. Przykładem takiej nieopłacalnej procedury może być

```
real procedure MAX(a,b);
 value a,b;
 real a,b;
 MAX:=if a>b then a else b
```

Jeżeli x, y są zmiennymi prostymi, to przekład wyrażenia MAX(x,y) zawiera 14 słów (por. P6 powyżej), a przekład wyrażenia if x>y then x else y zawiera 7 słów maszynowych. Oczywiście, w praktyce procedury mają treść zwykle dość długą i pytanie o opłacalność ich stosowania jest pozbawione sensu.

5.8. Procedury rekursywne

Translatory ALGOLu 1204 nie odróżniają procedur rekursywnych od innych procedur niestandardowych. Poprawność działania takich procedur wynika po prostu z dynamicznej rezerwacji pamięci. Z tego powodu nie można twierdzić, że sam fakt rekursywności prowadzi już do bardzo długich obliczeń w porównaniu z równoważną procedurą nierekursywną, chociaż niekiedy tak jest. Rozważmy przykład opisu procedury

```
real procedure T(n,t);
 value n,t;
 integer n;
 real t;
 T:=if n>1 then T(n-1,t)x2.0xt-T(n-2,t)
     else if n=1 then t else 1
```

który jest zoptymalizowaną wersją przykładu P18.63 z [1]. Niesprawność tej procedury wynika stąd, że program nie uwzględni wia-

domej z góry identyczności wielu wyników pośrednich. Dla n=10 treść procedury będzie wykonana aż 177 razy, chociaż wartość T(10,x) można obliczyć znając wartości T(9,x), T(8,x),... T(1,x), T(0,x). Zastąpienie rekursywnego opisu procedury T równoważnym opisem nierekursywnym (por. [1], P18.64) znacznie skróci obliczenia.

Inaczej jest w przypadku procedury o opisie

```
real procedure I(a,b,n);
 value a,b,n;
 integer n;
 real a,b;
 begin
  integer p;
  p:=n-1;
  I:=if n>1 then (tan(b)↑p-tan(a)↑p)/p-I(a,b,n-2)
      else if n=1 then ln(abs(cos(a)/cos(b))) else b-a
 end I;
```

Wartością tej procedury jest $\int_a^b tg^n x\,dx$. Napisanie równoważnego opisu nierekursywnego również i w tym przypadku nie jest trudne; jednak nie warto tego robić, ponieważ osiągnięty zysk czasowy będzie bardzo niewielki. W poprzednim przykładzie zależność rekurencyjna miała postać

T(n,t)=F(T(n-1,t),T(n-2,t))

(zależność od dwóch wartości "wcześniejszych"), a tutaj ma postać

I(a,b,n)=G(I(a,b,n-2))

(zależność od jednej wartości "wcześniejszej"). Wniosek wynikający z tych dwóch przykładów jest ważny dla wszystkich przykładów podobnych.

Rozważmy jeszcze jeden przykład naturalnej procedury rekursywnej, zaczerpnięty z książki "Introduction to ALGOL 60" (Baumanna i innych). Przypuśćmy, że rozwiązanie jakiegoś zadania można otrzymać w wyniku wykonania instrukcji postaci:

```
for p[N]:=1 step 1 until N do
 for p[N-1]:=1 step 1 until N do
  ......
   for p[1]:=1 step 1 until N do
    PROCES(p[1],p[2],...,p[N])
```

gdzie wartość N nie jest z góry znana. Jeżeli odrzucimy rozwiązanie polegające na wypisaniu wszystkich takich instrukcji dla potrzebnego zbioru wartości N, to jedynym naturalnym rozwiązaniem jest podany niżej program, w którym należy jedynie uzupełnić treść procedury PROCES, stosownie do rzeczywistej treści zadania.

```
begin
 procedure PROCES(a,k);
  value k;
  integer k;
  integer array a;
  begin
   .......
  end PROCES;
 integer N;
 read(N);
 begin
  procedure R(k);
   value k;
   integer k;
   if k=0
    then PROCES(p,N)
    else for p[k]:=1 step 1 until N do R(k-1);
  integer array p[1:N];
  R(N)
 end
end
```

Działanie procedury R można łatwo sprawdzić na maszynie przyjmując jako treść procedury PROCES na przykład instrukcję złożoną

```
begin
 line(1); format('␣1');
 for k:=k step -1 until 1 do print(a[k])
end
```

Jedynym ogólnie ważnym argumentem przeciwko procedurom rekursywnym jest to, że każde nowe wywołanie - bez wykonania wyjścia - powoduje rezerwację nowego (zwykle niewielkiego) obszaru pamięci. W pewnych przypadkach prowadzi to do bardzo szybkiego zajęcia całej dostępnej pamięci. Aby to pokazać rozważmy przykład, znany w literaturze jako "man-or-boy test", którego autorem jest

D. Knuth. Podana niżej wersja tego przykładu różni się od oryginału możliwością wykonania obliczeń dla różnych danych i pełnym zbiorem specyfikacji (p. 1.0, G1).

```
begin
 integer procedure A(k,x1,x2,x3,x4,x5);
  value k;
  integer k,x1,x2,x3,x4,x5;
  begin
   integer procedure B;
    begin
     k:=k-1;
     B:=A:=A(k,B,x1,x2,x3,x4)
    end B;
   if k≤0
    then A:=x4+x5
    else B
  end A;
 print(A(ininteger,1,-1,-1,1,0))
end
```

Dla k=10 po 8 sekundach pracy maszyna drukuje wynik równy -67; w tym czasie treść procedury A była wykonana 543 razy, a treść B - 542 razy i użyto ok. 6000 komórek roboczych pamięci.

Dla k=11 po 5 sekundach pracy maszyna sygnalizuje brak pamięci (SPACE OVERFLOW). Zwiększanie wartości k o 1 podwaja (w przybliżeniu) liczbę potrzebnych komórek roboczych. Przekład podanego programu zawiera 187 słów.

ROZDZIAŁ 6. Kod wewnętrzny maszyny ODRA 1204 w ALGOLu 1204

6.0. Informacje wstępne

W ALGOLu 1204 można używać 67 (spośród 115) rozkazów kodu wewnętrznego maszyny ODRA 1204. Służą do tego procedury specjalne (p. 3.0) z jednym parametrem lub bez parametrów. Nazwy tych procedur są identyczne z literowymi oznaczeniami części operacyjnych rozkazów stosowanymi w dokumentacji technicznej maszyny, a każdej instrukcji procedury odpowiada jeden rozkaz przekładu.

Stosując procedury specjalne można łatwo opisywać działania, których wyrażenie w ALGOLu 60 jest bardzo uciążliwe lub trudne. W wielu przypadkach za pomocą procedur specjalnych można także skrócić czas obliczeń i długość przekładu programu. Należy jednak pamiętać, że translator nie sprawdza semantycznej poprawności instrukcji tych procedur; niewłaściwe ich użycie może spowodować dezorganizację programu, a nawet systemu obsługi. W zamian za tę niedogodność uzyskujemy pewną swobodę korzystania z możliwości maszyny.

Korzystanie z kodu wewnętrznego maszyny w ALGOLu 1204 nie wymaga żadnej znajomości postaci programu tworzonego przez translator, ani też budowy translatora. W szczególności włączenie operacji opisanych przy pomocy procedur specjalnych w system dynamicznej rezerwacji pamięci jest całkowicie zautomatyzowane.

6.1. Rejestry maszyny i budowa słowa rozkazowego

Pamięć maszyny ODRA 1204 składa się z 16384 komórek, którym są przyporządkowane liczby całkowite 0,1,2,...,16383, zwane dalej adresami tych komórek. W każdej komórce pamięci można zapamiętać ciąg 24 zer lub jedynek - czyli bitów - zwany słowem maszynowym. Słowo maszynowe może przedstawiać liczbę całkowitą (tzw. stałoprzecinkową) lub słowo rozkazowe, czyli symbol elementarnej operacji maszynowej. Liczby rzeczywiste (tzw. zmiennoprzecinkowe) i liczby całkowite długie są zapamiętane w dwu kolejnych komórkach pamięci. Jeśli liczba jest zapamiętana w komórkach o adresach N-1 i N, to adresem tej liczby jest N.

Komórki o adresach 1,2,3 są wyróżnione; pełnią one role rejestrów b-modyfikacji (p. 6.3). W ALGOLu 1204 rejestry te są wykorzystane przez system dynamicznej rezerwacji pamięci.

W czasie działania maszyny informacje są przechowywane w pamięci oraz w następujących rejestrach:

A - 24-bitowy akumulator,
W - 24-bitowe wydłużenie akumulatora,
L - 16-bitowy licznik rozkazów,
U, Z, Nd - 1-bitowe rejestry warunków.

Przez AW będziemy oznaczać akumulator i jego wydłużenie, traktowane jako jeden rejestr 48-bitowy. Małymi literami będziemy oznaczać zawartości odpowiednich rejestrów (np. u oznacza zawartość rejestru U, aw - zawartość AW).

Słowo rozkazowe składa się z następujących części:

P - 1 bit - oznacza p-modyfikację (p. 6.3),
B - 2 bity - numer rejestru b-modyfikacji (p. 6.3),
OR- 7 bitów - część operacyjna rozkazu,
AR- 14 bitów - część adresowa rozkazu.

Części P, B, AR określają adres argumentu operacji (p. 6.3). Jeżeli w słowie rozkazowym OR=1, to AR jest traktowane jako dalszy ciąg części operacyjnej; taki rozkaz nazywamy rozkazem bezadresowym.

6.2. Reprezentacja liczb w maszynie

Każde słowo maszynowe przedstawia liczbę całkowitą. Jeśli kolejne bity słowa oznaczymy przez

b0 b1 ... b23

to o znaku liczby decyduje bit b0. Dla b0=0 słowo przedstawia liczbę całkowitą dodatnią równą

$$b1\times2^{22}+b2\times2^{21}+ \ldots +b22\times2^{1}+b23\times2^{0}$$

a dla b0=1, liczbę całkowitą ujemną równą

$$-2^{23}+b1\times2^{22}+b2\times2^{21}+ \ldots +b22\times2^{1}+b23\times2^{0}$$

W ALGOLu 1204 stałe typu integer i wartości zmiennych typu integer są zapamiętane jako liczby całkowite. Wartości logiczne

true i false są też zapamiętane jako liczby całkowite równe odpowiednio -1 (wszystkie bity są równe 1) i 1 (b23=1, pozostałe bity są równe zero). Stałe i wartości zmiennych typu real są zapamiętane jako liczby zmiennoprzecinkowe zajmujące po dwie komórki pamięci.

Liczba zmiennoprzecinkowa X różna od zera jest zapamiętana w postaci

$$m \times 2^c$$

gdzie liczba m - zwana mantysą - musi spełniać warunek

$$1/2 \leq m < 1 \quad \text{dla } X > 0,$$
$$-1 \leq m < -1/2 \quad \text{dla } X < 0,$$

a liczba c - zwana cechą - musi spełniać warunek

$$-512 \leq c \leq 511$$

Jeśli kolejne bity maszynowej reprezentacji liczby zmiennoprzecinkowej oznaczymy przez

b0 b1 ... b47

to mantysa m jest określona ciągiem bitów

b0 b1 ... b37

w następujący sposób:

$$m = b1 \times 2^{-1} + b2 \times 2^{-2} + \ldots + b37 \times 2^{-37}, \text{ dla } b0=0,$$
$$m = -1 + b1 \times 2^{-1} + b2 \times 2^{-2} + \ldots + b37 \times 2^{-37}, \text{ dla } b0=1,$$

a cecha c jest zapamiętana na bitach

b38 b39 ... b47

w ten sposób, że

$$c + 512 = b38 \times 2^9 + b39 \times 2^8 + \ldots + b47 \times 2^0.$$

Zero w postaci zmiennoprzecinkowej jest zapamiętane jako ciąg 48 zer.

PRZYKŁAD

Mantysa liczby 10.5 jest równa .65625, cecha jest równa 4, a reprezentacją tej liczby są dwa słowa maszynowe

0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0

zapamiętane w dwu kolejnych komórkach.

Ciąg 48 bitów (bity dwu kolejnych komórek)

b0 b1 ... b47

może także reprezentować liczbę całkowitą równą

$$b1\times2^{46}+b2\times2^{45}+ \ldots +b46\times2^{1}+b47\times2^{0} \quad \text{dla } b0=0,$$

albo

$$-2^{47}+b1\times2^{46}+b2\times2^{45}+ \ldots +b46\times2^{1}+b47\times2^{0} \quad \text{dla } b0=1.$$

Wprowadzanie liczb całkowitych długich i wykonywanie na nich działań w ALGOLu 1204 jest możliwe tylko przy użyciu procedur specjalnych.

6.3. Schemat wykonywania rozkazu

Wykonanie rozkazu w maszynie ODRA 1204 dzieli się na etapy:

e1. Przepisanie do rejestrów pomocniczych P1, B1, OR1, AR1 maszyny części P, B, OR, AR słowa rozkazowego z komórki o adresie równym l (zawartość L).

e2. Wykonanie b-modyfikacji: Jeśli B1≠0, to do AR1 dodaje się zawartość komórki o adresie równym B1.

e3. Wykonanie p-modyfikacji: Jeśli P1≠0, to do rejestrów P1, B1, AR1 podstawia się nowe wartości równe odpowiednim częściom słowa rozkazowego z komórki o adresie AR1 i przechodzi się do etapu e2.

e4. Jeśli wykonywany rozkaz był poprzedzony jednym z rozkazów: mok1, mok2, moa1, moa2, mow1, mow2 (p. 6.11), to do AR1 dodaje się odpowiednio: zawartość komórki o adresie równym wartości AR1 obliczonej dla poprzedniego rozkazu, podwojoną zawartość tej komórki, a, 2xa, w, 2xw.

e5. Wykonanie operacji wskazanej przez OR1. Jeśli jest to operacja skoku, to do rejestru L podstawia się wartość AR1; w przeciwnym razie zawartość L zwiększa się o jeden. W obu przypadkach przechodzi się do e1.

Z podanego schematu wynika, że część AR może być modyfikowana kilka razy (ponieważ p-modyfikacja odnawia b- i p-modyfikację, a część AR rozkazu modyfikującego następny rozkaz też może być modyfikowana). W maszynie ODRA 1204 liczba modyfikacji części adresowej jednego rozkazu jest ograniczona do 6; wykonanie modyfikacji po raz siódmy powoduje przerwanie pracy programu (sygnał WRONG INSTRUCTION, p. 11.3).

6.4. Schematy warunków

Wynikiem wykonania rozkazu jest określona zmiana zawartości rejestrów maszyny. Reguły określania wartości u, z, nd nazywamy schematami warunków i podajemy je w tabeli poniżej. Zawartość komórki o adresie wskazanym przez części P, B i AR rozkazu (p. 6.3) oznaczamy przez n.

Oznaczenie schematu warunków	Sposób podstawiania nowych wartości do rejestrów U, Z i Nd
a1	Jeśli a>0, to z:=0 i u:=0; jeśli a<0, to z:=0 i u:=1; jeśli a=0, to z:=1 i u:=0; nd:=0.
a2	Wartości u i z określa się tak, jak w schemacie a1. Jeśli wynik operacji nie należy do przedziału dopuszczalnych wartości liczb całkowitych, tzn. powstał nadmiar stałoprzecinkowy, to nd:=1, a w przeciwnym razie nd:=0.
a3	Schemat ten jest omówiony razem z rozkazami przesuwań (p. 6.9).
a4	Wartości u i z określa się tak, jak w schemacie a1, a wartość nd nie jest określona.
n1	Jeśli n>0, to z:=0 i u:=0; jeśli n<0, to z:=0 i u:=1; jeśli n=0, to z:=1 i u:=0; nd:=0.
n2	Wartości u i z określa się tak, jak w schemacie n1. Jeśli powstał nadmiar stałoprzecinkowy, to nd:=1, a w przeciwnym razie nd:=0.

6.5. Reguły zastępowania parametrów aktualnych procedur specjalnych adresami

Przekład instrukcji procedury specjalnej odpowiadającej rozkazowi maszyny zajmuje jedno słowo maszynowe. Procedury bez parametrów odpowiadają rozkazom bezadresowym. Możliwe rodzaje parametrów aktualnych procedur specjalnych z jednym parametrem podajemy poniżej:

Rodzaj parametru	Oznaczenie
etykieta	E
zmienna prosta arytmetyczna zmienna prosta poprzedzona znakiem +	Z

zmienna prosta
arytmetyczna zmienna prosta poprzedzona znakiem + } ZL
stała dowolnego typu
stała arytmetyczna poprzedzona znakiem +

Parametrom aktualnym procedur specjalnych przyporządkowuje się adresy komórek pamięci w następujący sposób:

Rodzaj parametru	Adres
etykieta	adres pierwszego słowa maszynowego przekładu instrukcji poprzedzonej tą etykietą,
zmienna prosta	adres komórki zarezerwowanej na tę zmienną wraz z odpowiednią b-modyfikacją (dla zmiennych rzeczywistych jest to adres drugiej komórki),
stała	adres komórki zawierającej tę stałą.

Jeżeli przed zmienną lub stałą występuje znak +, to do adresu określonego jak wyżej dodaje się p-modyfikację.

UWAGA

Parametrem aktualnym procedury specjalnej nie może być parametr formalny procedury niestandardowej nie umieszczony w zbiorze wartości

Oznaczeń E, Z, ZL będziemy używać w następnych punktach tego rozdziału do wskazywania możliwych postaci parametrów aktualnych procedur specjalnych. Symbol N będzie oznaczać adres przyporządkowany parametrowi aktualnemu i być może-zmodyfikowany, n - słowo maszynowe o adresie N, a nn - słowo długie zapamiętane w komórkach o adresach N-1 i N. Dla każdego rozkazu podano czas jego wykonania (w mikrosekundach) przy założeniu, że w rozkazie nie występuje p i b-modyfikacja. Każda taka modyfikacja zwiększa czas wykonania rozkazu o 11 mikrosekund. Pomiary czasów były wykonane na maszynie cyfrowej ODRA 1204 zainstalowanej w Centrum Obliczeniowym Uniwersytetu Wrocławskiego i mogą różnić się od czasów innych maszyn nawet o 10%.

6.6. Rozkazy skoków

Wykonanie rozkazu skoku polega na zmianie zawartości licznika rozkazów L, bez zmiany zawartości innych rejestrów (w szcze-

gólności rejestrów U, Z, Nd). W tabeli poniżej podano wszystkie dopuszczalne instrukcje procedur specjalnych odpowiadające rozkazom skoku.

Instrukcja	Treść rozkazu	Czas
skb(E)	l:=N	10
skd(E)	l:=if u=0∧z=0 then N else l+1	19
skz(E)	l:=if z=1 then N else l+1	19
sku(E)	l:=if u=1 then N else l+1	19
skn(E)	l:=if nd=1 then N else l+1	19
snd(E)	l:=if u=1∨z=1 then N else l+1	19
snz(E)	l:=if z=0 then N else l+1	19
snu(E)	l:=if u=0 then N else l+1	19

PRZYKŁAD

Przekład instrukcji

E1:go to E1

jest identyczny z przekładem instrukcji

E1:skb(E1)

6.7. Rozkazy działań na słowach krótkich

Wymienione w tabeli poniżej instrukcje procedur specjalnych służą do wykonywania działań na słowach 24 bitowych. Termin "działania logiczne" oznacza w tym punkcie opisane dalej działania na bitach. Jeśli a, n są odpowiednio ciągami bitów

a0 a1 ... a23
n0 n1 ... n23

i wynik działania logicznego ma postać

b0 b1 ... b23

to bi=1 (i=0,1,...,23) dla poszczególnych działań tylko w następujących przypadkach:

a dl n, jeśli ai=1 lub ni=1,
a ml n, jeśli ai=1 i ni=1,
a rl n, jeśli ai≠ni.

Instrukcja	Treść rozkazu	Schemat warunków	Czas
usak(ZL)	a:=n	a1	16
ssak(ZL)	a:=a+n	a2	16
osak(ZL)	a:=a-n	a2	16
rsak(ZL)	a:=n-a	a2	16
mlak(ZL)	a:=a ml n	a1	16
zmak(Z)	zamiana a, n	a1	27
uazk(Z)	a:=n; n:=0	a1	27
pska(Z)	n:=a	n1	22
sska(Z)	n:=a+n	n2	28
oska(Z)	n:=a-n	n2	28
rska(Z)	n:=n-a	n2	28
mlka(Z)	n:=a ml n	n1	27
dokj(Z)	n:=n+1	n2	28
pkza(Z)	n:=a; a:=0	n1	27
odkj(Z)	n:=n-1	n2	28
zerk(Z)	n:=0	bez zmiany	22
pakw(Z)	n:=w	n1	22
uswk(ZL)	w:=n	n1	16
dlak(ZL)	a:=a dl n	a1	16
olak(ZL)	a:=a rl n	a1	16
czkl	a:=stan klawiatury pulpitu maszyny	a1	19
usaw	a:=w	a1	19
pswa	w:=a	a1	19
zmaw	zamiana a, w	a1	25

PRZYKŁAD

Instrukcja

if k÷2×2=k then go to L

gdzie k jest typu integer, L jest etykietą występującą przed instrukcją, ma skutki takie, jak instrukcje

usak(k); mlak(1); skz(L)

6.8. Rozkazy działań na słowach długich

Słowo długie oznacza tutaj liczbę rzeczywistą albo liczbę całkowitą długą. Dla odróżnienia działań na tych dwu reprezentacjach liczb, działania zmiennoprzecinkowe są w tabeli podkreślone.

Instrukcja	Treść rozkazu	Schemat warunków	Czas
uzak(ZL)	aw:=nn	a1	31
szak(ZL)	aw:=aw+nn	a4	170
ozak(ZL)	aw:=aw-nn	a4	169
rzak(ZL)	aw:=nn-aw	a4	178
pzka(Z)	nn:=aw	a1	47
mzak(ZL)	aw:=aw×nn	a4	396
dzak(ZL)	aw:=aw/nn	a4	1156
sdak(ZL)	aw:=aw+nn	a2	32
odak(ZL)	aw:=aw-nn	a2	32
rdak(ZL)	aw:=nn-aw	a2	32
msak(ZL)	aw:=a×n	a1	86
dsak(ZL)	a:=aw/n; w:=reszta z dzielenia (o takim samym znaku, jak n)	a1	201
ddak(ZL)	aw:=aw/n	a1	356
mina	aw:=-aw	a4	87
absa	aw:=abs(aw) (wartość bezwzględna liczby zmiennoprzecinkowej)	a4	98
deca	aw:=aw×10	a1	38
zeaw	aw:=0	bez zmiany	22
stnd	zmiana aw z postaci liczby całkowitej na postać zmiennoprzecinkową	a1	700
enta	jeśli entier(aw) nie należy do przedziału liczb całkowitych, to l:=l+1; w przeciwnym razie a:=entier(aw) i l:=l+2	a1	49

PRZYKŁAD

Jeśli w czasie wykonywania instrukcji

```
X:=(N-J)×K×3.14; J:=entier(X)
```

gdzie X jest typu real, N,J,K są typu integer, program zatrzyma się i wydrukuje sygnał RI CONVERSION (p. 11.3), to w czasie wykonywania instrukcji

```
usak(N); osak(J); msak(K); stnd; mzak(3.14);
pzka(X); enta; skb(L); pska(J);
.....
L:print(' ?wynik poza przedziałem',stop)
```

dla tych samych wartości zmiennych program wydrukuje napis

wynik poza przedziałem

i zatrzyma się (p. 3.18).

6.9. Rozkazy przesuwań

Rozkazy przesuwań odgrywają istotną rolę przy programowaniu działań na grupach bitów słowa maszynowego. Jeśli przez

b0 ... b47

oznaczymy kolejne bity aw, a przez p oznaczymy resztę z dzielenia N przez 64, to treść rozkazów przesuwań można opisać w następujący sposób:

Instrukcja	Treść rozkazu (postać aw po wykonaniu rozkazu)	Schemat warunków	Czas (dla p=10)
pak(ZL)	$\overbrace{b0 \dots b0}^{p+1 \text{ razy}}\ b1 \dots b(23-p)$ b24 ... b47	a1	36
lnk(ZL)	bp ... b23 $\overbrace{0 \dots 0}^{p \text{ razy}}$ b24 ... b47	a3	39
pnk(ZL)	$\overbrace{0 \dots 0}^{p \text{ razy}}\ b0 \dots b(23-p)$ b24 ... b47	a1	36
pad(ZL)	$\overbrace{b0 \dots b0}^{p+1 \text{ razy}}\ b1 \dots b(47-p)$	a1	36
lnd(ZL)	bp ... b47 $\overbrace{0 \dots 0}^{p \text{ razy}}$	a3	39
pck(ZL)	b(23-p+1) ... b23 b0 ... b(23-p) b24 ... b47	a1	36

Schemat warunków a3 oznacza, że wartości u, z są takie, jak w schemacie a1, a nd=0 tylko wtedy, gdy b0=b1=...=b(p-1). W przeciwnym razie nd=1.

Jeżeli w czasie wykonywania rozkazów przesuwań wartość p jest większa od 23 (lub 47), to przyjmuje się, że p=23 (lub p=47).

PRZYKŁAD

Następujący ciąg instrukcji wykonuje zerowanie N-tego bitu w słowie J:

```
zeaw; usak(1); pad(+N); pad(+1);
usaw; olak(true); mlka(J)
```

6.10. Rozkazy generowania warunków

Trzy rozkazy o kodach symbolicznych pgw, psak, plak służą do ustawiania wartości u, z, nd bez zmiany zawartości rejestrów A, W i zawartości komórki o adresie N.

Instrukcja	Treść rozkazu	Czas
pgw(ZL)	generowanie warunków według schematu n1	16
psak(ZL)	nd:=0; jeśli a>n, to z:=0 i u:=0; jeśli a<n, to z:=0 i u:=1; jeśli a=n, to z:=1 i u:=0	23
plak(ZL)	nd:=0; jeśli a=n, to z:=1 i u:=0; w przeciwnym razie z:=0 i jeśli ai=ni (i=0,1,...,k) i a(k+1)≠n(k+1) i a(k+1)=1, to u:=0, a w przeciwnym razie u:=1.	23

PRZYKŁAD

Instrukcja

```
E:if inchar=29Vlastchar=73 then go to E
```

działa tak samo, jak ciąg instrukcji

```
E:inchar; psak(29); skz(E); psak(73); skz(E)
```

6.11. Rozkazy modyfikacji

Wykonanie rozkazu modyfikacji nie powoduje zmiany zawartości żadnego z rejestrów maszyny (z wyjątkiem oczywistej zmiany l:=l+1), a oznacza jedynie modyfikację części adresowej rozkazu wykonywanego bezpośrednio po rozkazie modyfikacji (p. 6.3, e4).

Instrukcja	Wartość modyfikująca część adresową następnego rozkazu	Czas
mok1(ZL)	n	20
mok2(ZL)	2xn	23
moa1	a	22
moa2	2xa	26
mow1	w	22
mow2	2xw	26

Przykład zastosowania rozkazu modyfikacji jest podany w p. 6.12.

6.12. Rozkaz skw

Pewną osobliwością maszyny ODRA 1204 jest rozkaz skw. Wykonanie instrukcji skw(E), gdzie E jest etykietą występującą przed inną instrukcją, powoduje wykonanie jednego rozkazu z komórki o adresie N, a do czasu wykonania tego rozkazu dodaje się 11 mikrosekund.

PRZYKŁAD

Wartość zmiennej X po wykonaniu instrukcji

```
if 0<i∧i≤5 then
 X:=if i=1 then a else if i=2 then b else
     if i=3 then c else if i=4 then d else e
```

jest taka sama, jak po wykonaniu ciągu instrukcji

```
if 0<i∧i≤5
 then begin
     E:go to E1;
       uzak(a); uzak(b); uzak(c); uzak(d); uzak(e);
    E1:mok1(i); skw(E); pzka(X)
       end
```

6.13. Uwagi o stosowaniu procedur specjalnych

Używając procedur specjalnych warto pamiętać, że po wykonaniu instrukcji podstawienia albo instrukcji procedury wartość podstawionego wyrażenia albo wartość procedury znajduje się w akumulatorze, a wartości u, z, nd są określone tak, jak w schemacie warunków a1.

Przetłumaczenie programu ze śladem etykiet (p. 8.6, klawisz nr 3) powoduje dołączenie rozkazu drukowania etykiety przed przekładem instrukcji poprzedzonej tą etykietą. Po wykonaniu takiego rozkazu wartość nd jest zerem, a zawartości innych rejestrów nie ulegają zmianie. Ponadto, jeśli przed rozkazem z etykietą występuje rozkaz modyfikacji (p. 6.11), to zmodyfikuje się rozkaz drukowania etykiety i nie można przewidzieć skutków działania tego rozkazu.

ROZDZIAŁ 7. Przykłady stosowania kodu wewnętrznego

7.0. Uwagi ogólne

Rozdział 7 zawiera dwa rodzaje przykładów stosowania kodu wewnętrznego:

a. Wykonywanie takich operacji, których nie można opisać za pomocą innych pojęć ALGOLu 1204.
b. Optymalizacja niektórych fragmentów programu.

Pisanie całego programu w kodzie wewnętrznym mija się z celem, gdyż niewiele skraca czas działania programu, a znacznie zmniejsza jego czytelność.

7.1. Obliczenie wielkości obszaru pamięci dostępnej dla rezerwacji dynamicznej

Komórki o adresach 17 i 18 w czasie działania programu zawierają informacje o obszarze pamięci dostępnej dla rezerwacji dynamicznej (p. 11.6). Ciąg instrukcji

```
    usak(+18); snz(E); usak(14816);
E:osak(+17); osak(1); pska(MAXP)
```

nadaje zmiennej MAXP wartość równą liczbie komórek dostępnych dla rezerwacji dynamicznej.

Zawartości komórek o adresach 17 i 18 są uaktualniane tylko bezpośrednio po wykonaniu rezerwacji pamięci, to znaczy po zarezerwowaniu miejsca na tablicę i przed wykonaniem treści procedury niestandardowej. Po wyjściu z obszaru działania tablicy (bez miana own) i po wyjściu z treści procedury obszar dostępnej pamięci jest większy od obszaru określonego zawartością komórek 17 i 18. Tak więc wykonywanie podanego wyżej ciągu instrukcji ma sens tylko na początku bloku lub na początku treści procedury.

7.2. (D) Obliczenie wielkości obszaru roboczego pamięci bębnowej

Po przetłumaczeniu programu zawartość komórki o adresie 38 jest równa wielkości obszaru roboczego pamięci bębnowej. Program może używać tego obszaru do przechowywania danych (w p. 3.21

zawartość tej komórki jest oznaczona symbolem D). Jeżeli program jest segmentowany, to po zapisaniu segmentu bębnowego (p. 1.11) wielkość dostępnego dla programu obszaru roboczego pamięci bębnowej i zawartość komórki o adresie 38 odpowiednio zmniejszy się.

W wyniku wykonania dwóch instrukcji

```
usak(+38); pska(MAXD)
```

pod zmienną MAXD będzie podstawiona aktualna wielkość dostępnego dla programu obszaru roboczego pamięci bębnowej.

7.3. Mnożenie liczby całkowitej długiej przez krótką

Rozważmy procedurę

```
procedure msdk(a,n,L);
 value a,n;
 integer a,n;
 label L;
 begin
  integer znak,b;
  real x;
  x:=.0;
  zerk(znak); pgw(n);     skd(E4);    odkj(znak);
  usak(0);    oska(n);
E4:uzak(+a);  snu(E);     dokj(znak); rdak(.0);
 E:pakw(b);   msak(n);    skz(E1);
E2:go to L;
E1:usak(b);   mlak(1);    skz(E5);    usak(n);
  pska(x);
E5:usak(b);   pakw(b);    pnk(+1);    msak(n);
  lnd(+1);    sdak(x);    ssak(b);    skn(E2);
  sku(E2);    pgw(znak);  skz(E3);    rdak(.0);
E3:pzka(x);   msdk:=x
 end msdk
```

Procedura ta mnoży liczbę całkowitą długą (p. 6.2) przez liczbę całkowitą krótką. Podstawienie iloczynu pod zmienną X następuje w wyniku wykonania instrukcji

```
X:=msdk(A,N,ET)
```

gdzie X jest zmienną typu real, A jest wyrażeniem arytmetycznym typu integer o wartości równej adresowi liczby długiej (wartość

funkcji ref), N jest wyrażeniem arytmetycznym typu integer, przez które mnoży się liczbę długą, a ET jest etykietą stojącą przed instrukcją, którą należy wykonać, jeśli wyniku mnożenia nie można zapamiętać na 48 bitach.

7.4. Drukowanie słów maszynowych w postaci ósemkowej

Podana niżej procedura drukuje 48-bitowe słowo maszynowe w postaci ciągu cyfr ósemkowych. Parametrem aktualnym instrukcji tej procedury jest adres liczby długiej, którą chcemy wydrukować.

```
procedure dlcd(a);
 value a;
 integer a;
 begin
  integer i,k;
  format('␣1');
  for i:=45 step -3 until 0 do
   begin
    uzak(+a); pad(+i); usaw;
    mlak(7);  pska(k); print(k)
   end i;
 end dlcd
```

7.5. Drukowanie liczby całkowitej z minimalną ilością cyfr

Procedura

```
procedure dlc(l);
 value l;
 integer l;
 begin
  integer i,j,sc;
  uswk(l); skz(DC); snu(PD); outchar(32); odak(l);
PD: zerk(sc); usak(7); pkza(l);
KC: dsak(1 000 000); sska(sc); skz(NZ); pzka(j); snz(DCC);
DC: i:=16;
DCC: outchar(i); uswk(j);
NZ: deca; odkj(l); skd(KC)
 end dlc
```

ma działanie równoważne procedurze podanej w p. 4.4. Zwracamy uwagę czytelnika na to, że umieszczona w treści proce-

dury instrukcja pzka(j) nie jest semantycznie błędna. Wykorzystano tutaj fakt, że jeśli w programie występuje opis

```
integer i,j,sc
```

to zmiennym i,j,sc translator przyporządkuje adresy w ten sposób, że adres zmiennej i jest o jeden mniejszy od adresu zmiennej j, a ten ostatni jest o jeden mniejszy od adresu zmiennej sc. Wobec tego po wykonaniu instrukcji pzka(j) pod zmienne i,j zostanie podstawiona odpowiednio zawartość akumulatora i zawartość wydłużenia akumulatora.

7.6. Optymalizacja programu

Za pomocą procedur specjalnych warto optymalizować przede wszystkim odwołania do zmiennych ze wskaźnikami, które występują wewnątrz wielu instrukcji "dla". W większości przypadków oznacza to rezygnację (całkowitą lub częściową) z dynamicznego sprawdzania wskaźników. Przekład zoptymalizowanych w ten sposób fragmentów programu jest na ogół dłuższy, ale czas wykonywania jest dużo krótszy.

PRZYKŁAD 1

Przekład instrukcji

```
for k:=1 step 1 until n do X[k]:=A[k,i]
```

zajmuje 22 słowa maszynowe, a czas wykonania wynosi ok. 1650×n mikrosekund. Jeśli tablica A ma opis array A[1:n,1:m], to rezygnując z kontroli wskaźników elementów tablicy X, można zastąpić tę instrukcję ciągiem instrukcji

```
aX:=ref(X[1]); aA:=ref(A[n,i]); m2:=m+m;
for k:=ref(A[1,i]) step m2 until aA do
 begin
  uzak(+k); pzka(+aX); usak(2); sska(aX)
 end
```

których przekład zajmuje 33 słów maszynowych, a czas działania wynosi ok. 2470+430×n mikrosekund. Rezygnując z kontroli wskaźników elementów tablicy A można tę samą instrukcję "dla" zastąpić ciągiem instrukcji

```
aX:=ref(X[n]); aA:=ref(A[1,i]);
for k:=ref(X[1]) step 2 until aX do
 begin uzak(+aA); pzka(+k); usak(m); ssak(m); sska(aA) end
```

o przekładzie 29 słów maszynowych i czasie wykonania ok. 2090+ 280×n mikrosekund.

PRZYKŁAD 2

Instrukcję

```
for i:=0 step 1 until n do s:=s×X+A[i]
```

(przekład 21 słów, czas wykonania ok. 1380×(n+1) mikrosekund) można zastąpić ciągiem instrukcji

```
aA:=ref(A[n]);
for i:=ref(A[0]) step 2 until aA do
 begin uzak(s); mzak(X); szak(+i); pzka(s) end
```

(przekład 22 słowa, czas wykonania ok. 1150+630×(n+1) mikrosekund), które też zawierają dynamiczną kontrolę wskaźników tablicy A (ściślej, jest badany tylko wskaźnik najmniejszy i największy w rozważanym fragmencie programu).

W podanych wyżej przykładach nie użyto procedur specjalnych do organizacji pętli, ponieważ zysk stąd wynikający jest bardzo mały, a traci się na czytelności programu.

PRZYKŁAD 3

W poniższej tabeli podajemy trzy przykłady stosowania procedur specjalnych do optymalizacji fragmentów programów nie zawierających zmiennych ze wskaźnikami. Każdy przykład składa się z ciągu instrukcji napisanych w ALGOLu i ciągu równoważnych instrukcji zawierającego procedury specjalne.

Ciąg instrukcji	Objętość przekładu	Czas wykonania
`a:=a+(b×c+3.0)/a; if a<0 then go to E`	11	2430
`a:=a+(b×c+3.0)/a; sku(E)`	7	1980
`i:=i+1; k:=k-1`	6	108
`dokj(i); odkj(k)`	2	56
`if 0<i∧i≠k then k:=i`	11	560
`usak(i); snd(E); psak(k); skz(E); pska(k);`	5	100
`E:.....`		

ROZDZIAŁ 8. Wykrywanie i korygowanie błędów w programach

8.0. Uwagi ogólne

Wykaz czynności prowadzących do poprawnie działającego programu jest dość długi i wykonując je łatwo popełnić błędy. Niektórych rodzajów błędów formalnych w dużych programach praktycznie nie można uniknąć, a wyszukiwanie takich błędów jest pracą niełatwą i czasochłonną. Translatory ALGOLu 1204 pomagają jednak szukać i usuwać te błędy. Każdy translator ALGOLu 1204 sprawdza m. in. formalną poprawność tłumaczonego programu. Jeżeli ten program (lub opis niezależnie tłumaczonej procedury) jest błędny, to translator drukuje lub perforuje symboliczne opisy zauważonych błędów wraz z odpowiednimi numerami wierszy programu (p. 8.1). Na oddzielne żądanie translator może również podać wykaz tzw. punktów orientacyjnych (p. 8.8) umożliwiający szybkie odszukanie wierszy o podanych numerach.

Translator może także korygować tłumaczony program na podstawie podanego uprzednio - z monitora lub z taśmy - wykazu poprawek (p. 8.9). Na oddzielne żądanie (p. 8.7) translator perforuje tekst tłumaczonego programu (lub opisu procedury), po ewentualnym skorygowaniu tego tekstu według wykazu poprawek. Skorygowana taśma jest produktem ubocznym procesu tłumaczenia.

Do kontroli działania programu może służyć tzw. ślad zawierający informacje o kolejności wykonywania niektórych instrukcji w czasie działania programu (p. 8.6).

Próbne tłumaczenia programu - mające na celu np. wykrycie błędów formalnych - nie są zbyt kłopotliwe (szczególnie na zestawie D), gdyż translatory ALGOLu 1204 działają szybko. Dla przykładu podajemy, że program, którego przekład zajmuje 3000 słów maszynowych, będzie przetłumaczony i przygotowany do wykonania w czasie na ogół krótszym od 1 minuty.

Należy pamiętać, że nie wszystkie błędy formalne są wykrywane w czasie tłumaczenia programu. Niektóre z nich mogą być wykryte dopiero w czasie działania programu. Dlatego w dalszym ciągu będziemy mówić oddzielnie o tych dwóch rodzajach błędów. Pewne - nieliczne zresztą - błędy formalne nie są w ogóle

wykrywane (p. 8.5).

Na zakończenie tego punktu trzeba wyraźnie podkreślić, że mówiąc o błędach wykrywanych w czasie tłumaczenia lub wykonywania programu nie mamy w żadnym wypadku na myśli błędów logicznych w metodzie numerycznej przyjętej w programie. Oczywiście, także i w takim przypadku użycie maszyny może ułatwić znalezienie błędu, ale będzie to wymagało zaplanowania i przeprowadzenia odpowiednich eksperymentów.

8.1. Sygnalizacja błędów w czasie tłumaczenia programu

Informacje o błędach translator wyprowadza bądź na monitor, bądź też na perforator - stosownie do tego, czy klawisz nr 0 pulpitu jest zwolniony, czy wciśnięty. Każdy sygnał błędu jest drukowany w jednym wierszu i ma postać:

E n ⟨opis błędu⟩

Litera E oznacza, że translator wykrył błąd, a n jest numerem wiersza (p. 8.8) aktualnie analizowanego przez translator; często n jest także równe numerowi tego wiersza, w którym błąd występuje. Różnica pomiędzy tymi numerami może być duża, jeżeli błąd polega na jakiejś niezgodności dwóch odległych od siebie części programu.

Do tworzenia opisów błędów translator używa symboli czterech rodzajów:

a. symboli podstawowych ALGOLu, które oznaczają siebie - np. if then else begin [ ] itd.
b. krótkich zwrotów lub słów zaczerpniętych z oryginalnej terminologii algolowskiej, np. OUT OF SCOPE, UNDECLARED; znaczenie tych zwrotów opisano dokładnie w p. 11.2,
c. symboli występujących w tłumaczonym tekście, np. nazw tam użytych,
d. symboli oznaczających elementy składniowe ALGOLu, różne od symboli podstawowych.

Te ostatnie symbole są utworzone z pierwszych liter odpowiednich terminów algolowskich. Na przykład symbol SIMAREX pochodzi od "simple arithmetic expression", FORCL - od "for clause", BAST - od "basic statement", CONST - od "conditional statement" itd. Znaczenia wszystkich takich symboli - po polsku i po angielsku - podano w skorowidzu na końcu podręcznika.

Wszystkie sygnały błędu translator tworzy wybierając odpowiednie symbole spośród wymienionych wyżej w punktach a-d; powstaje w ten sposób informacja, którą należy traktować zawsze jako komentarz do błędu wykrytego w programie, a nie jako błędny fragment programu. Aby wyjaśnić sposób interpretacji sygnałów błędu rozważymy przykłady tekstów algolowskich zawierających błędy. Są to przykłady dość sztuczne, ale dobrze ilustrujące postać sygnalizacji błędów.

PRZYKŁAD 1

Rozważmy tekst

```
P:begin
   integer i,k;
   array X,Y[1:k];
   read(k,y);
   for i=1 step 1 until k do
    begin
     integer s;
L:   read(s;
     if s-1=
      then print s+1)
      else X[i]:=if i÷2x2=i then i+k
    end i;
   if s=7
    then go to L;
   sort and output(X,Y,k);
   if wait(co dalej)=0
    then go to P
  end
```

W czasie tłumaczenia tego tekstu translator wydrukuje następujące informacje:

```
E   3  k IN BOUND PAIR LIST
E   4  y UNDECLARED
E   5  for SIMARVA RELOP
E   8  PROID ( SIMARVA ;
E  10  SIMAREX RELOP then
E  10  prints UNDECLARED
E  12  IFCL SIMAREX end
E  13  s OUT OF SCOPE
```

```
E  14  L OUT OF SCOPE
E  15  sortandoutput UNDECLARED
E  16  codalej UNDECLARED
```

Przy niewielkiej wprawie i znajomości odpowiedniej terminologii angielskiej przytoczony wykaz można zinterpretować bez żadnych dodatkowych informacji; zawsze jest to możliwe po przejrzeniu skorowidza na końcu podręcznika.

Spośród jedenastu przytoczonych sygnałów znaczenie siedmiu wynika natychmiast stąd, że

```
IN BOUND PAIR LIST  oznacza  "w wykazie par granicznych",
UNDECLARED          oznacza  "nie opisane",
OUT OF SCOPE        oznacza  "poza obszarem działania",
```

a symbole występujące przed tymi zwrotami są nazwami występującymi w tekście tłumaczonym przez translator.

Pozostałe cztery sygnały są opisami błędów gramatycznych. W skorowidzu znajdziemy następujące wyjaśnienia:

```
SIMARVA   simple arithmetic variable,
          prosta zmienna arytmetyczna,
RELOP     relational operator,
          operator relacji,
PROID     procedure identifier,
          nazwa procedury,
SIMAREX   simple arithmetic expression,
          proste wyrażenie arytmetyczne,
IFCL      if clause,
          warunek "jeśli"
```

Znając sens symboli występujących w sygnale, nietrudno już zinterpretować ten sygnał w całości. Na przykład sygnał

```
E  5  for SIMARVA RELOP
```

oznacza, że w czasie analizy wiersza nr 5 translator wykrył w programie obecność ciągu elementów składniowych

for ⟨prosta zmienna arytmetyczna⟩ ⟨operator relacji⟩

który w żadnym poprawnym programie nie może wystąpić. Analogicznie trzeba odczytać pozostałe trzy sygnały błędów gramatycznych.

Już z tego niewielkiego przykładu wynikają pewne ogólne wnioski:

w1. Sygnały translatora są wynikiem czysto mechanicznej analizy programu. Człowiek analizujący ten sam tekst prawdopodobnie powie, że np. w wierszu 5 brakuje dwukropka, w wierszach 8 i 10 brakuje nawiasów do pary i jeszcze jakiegoś wyrażenia po prawej stronie równości, a w wierszu 16 brakuje nawiasów łańcuchowych.

w2. Każdy z elementów wymienionych w sygnale błędu gramatycznego jest poprawny, a dopiero zestawienie tych poprawnych elementów jest błędem. Na przykład

```
if i÷2x2=i then
```

jest poprawnym warunkiem "jeśli" (IFCL),

```
i+k
```

jest poprawnym prostym wyrażeniem arytmetycznym (SIMAREX), ale ciąg elementów

```
IFCL SIMAREX end
```

wykryty przez translator w czasie analizy wiersza nr 12 jest błędem gramatycznym.

w3. Interpretując sygnały błędów trzeba pamiętać, że wydrukowany przez translator numer wiersza nie zawsze odnosi się do wiersza, w którym błąd występuje. Na przykład, błąd występujący zapewne w wierszu 9 jest sygnalizowany w czasie analizy wiersza 10. Stanie się to zrozumiałe, jeśli przypomnimy sobie, że układ graficzny programu nie ma żadnego wpływu na jego interpretację; w tej sytuacji brak wyrażenia po prawej stronie znaku równości można wykryć dopiero po przejściu do analizy następnego wiersza.

Niekiedy sygnał błędu zawiera tylko jeden symbol elementu składniowego. Oznacza to, że wymieniony element jest poprawny pod względem gramatycznym, ale błędny pod względem semantycznym.

PRZYKŁAD 2

W czasie tłumaczenia tekstu

```
 begin
  integer i;
  real r;
  Boolean B;
  switch s:=L,s[sin(B) div i];
L:i:=r:=2;
  read(B);
```

```
    go to s[i,r];
  end
```

translator wydrukuje następujący wykaz błędów:

```
E  5  ACPALIST
E  5  div
E  6  LEPA
E  7  ACPALIST
E  8  SULIST
```

który należy interpretować tak:

wiersz 5: błędny wykaz parametrów aktualnych (actual parameter list) funkcji sin i źle użyte dzielenie całkowite (p. 8.3),
wiersz 6: błędna lewa strona (left part),
wiersz 7: błędny wykaz parametrów aktualnych,
wiersz 8: błędny wykaz wskaźników (subscript list) - dwa wskaźniki zamiast jednego.

W omówionych przykładach wykaz błędów wydrukowany przez translator ma następujące własności:

A. wykaz jest pełny, tzn. zawiera informacje o wszystkich błędach występujących w tekście,
B. wykaz nie zawiera informacji fałszywych.

W praktyce wykazy błędów spełniają obie wymienione własności dość często, jednak nie zawsze. Niespełnienie warunku (A) wynika najczęściej stąd, że tłumaczenie programu jest wykonywane w dwóch kolejnych etapach, przy czym wykrycie błędu w pierwszym etapie wyklucza już działanie etapu drugiego. Ponadto, po wykryciu błędu, translator prawie zawsze pomija pewien fragment programu w otoczeniu błędu; na przykład, po wykryciu błędu we wnętrzu wyrażenia translator zwykle pominie co najmniej całą resztę tego wyrażenia, co oznacza również pominięcie wszystkich następnych błędów w tym wyrażeniu.

Warunek (B) może nie być spełniony dlatego, że dalsza analiza programu po wykryciu błędu wymaga wykonania zmiany dość licznych informacji o tłumaczonym programie; nie zawsze translator wykona taką zmianę poprawnie, ponieważ wymagałoby to np. odgadywania intencji programisty i innych kwalifikacji nie zawsze osiągalnych także dla człowieka.

Wynika stąd, że jeśli któryś z kolei sygnał na wydrukowanym wykazie nie daje się skojarzyć z tekstem programu, to trzeba ten sygnał pominąć. Przypadek taki w praktyce jest raczej rzadki. Niespełnienie warunku (A) zdarza się bez porównania częściej.

Niektóre sygnały błędu drukowane w czasie pierwszego etapu tłumaczenia (tzn. w czasie czytania taśmy) nie zawierają żadnego opisu błędu, tzn. sygnał zawiera tylko literę E i numer wiersza. Oznacza to, że w translatorze nie przewidziano komentarza dla wykrytego błędu. Należy w tym przypadku - podobnie jak w innych - sprawdzić wskazany wiersz programu i wiersze poprzednie. Najczęściej (ale nie zawsze) okaże się, że brakuje średnika po nawiasie zamykającym kończącym instrukcję, a następna instrukcja zaczyna się od niepodkreślonej litery.

PRZYKŁAD 3

W czasie tłumaczenia tekstu

```
  begin
 K:read(x,y)
   print(sqrt(x↑4+y↑4));
   go to K;
K:end
```

translator wydrukuje wykaz

```
E  3
E  5  K  REPEATED
```

i zatrzyma się. Braku opisów zmiennych x,y translator w tym przypadku nie wykryje.

Niekiedy translator zatrzymuje się natychmiast po wykryciu i zasygnalizowaniu jednego błędu. Oznacza to, że analizowany program nie jest ciągiem symboli podstawowych AIGOLu 1204 (p. 1.4) albo pamięć operacyjna jest już całkowicie wypełniona informacjami o tłumaczonym programie (w celu ochrony danych bębnowych także translator D w czasie tłumaczenia nie umieszcza żadnych informacji w pamięci bębnowej).

8.2. Postać gramatyki pamiętanej przez translator

Proces rozbioru gramatycznego programu jest sterowany za pomocą gramatyki zapamiętanej na stałe w translatorze. Gramatyka ta jest równoważna gramatyce zawartej w opisie języka wzorcowego, (por. [1], Dodatek A), ale nie jest z nią identyczna. Różnice

pochodzą stąd, że oryginalna gramatyka ALGOLu 60 nie nadaje się do jednoznacznego sterowania rozbiorem programu, a także stąd, że translator operuje tekstem przekształconym we wstępnym etapie tłumaczenia. Sygnały błędów gramatycznych są tak dobrane, aby wspomniane różnice w większości przypadków nie były widoczne w sygnałach błędów; jednak w pewnych przypadkach zatarcie różnic nie jest możliwe. Dla uniknięcia możliwych nieporozumień podajemy dalej te fragmenty wewnętrznej gramatyki ALGOLu 1204, które w sposób istotny różnią się od odpowiednich fragmentów gramatyki ALGOLu 60. Należy wyraźnie podkreślić, że nie chodzi tu o różnice w zbiorze tekstów poprawnych, ale o inny sposób definiowania tego zbioru i drobne różnice terminologiczne. Oto zapowiedziane fragmenty gramatyki, w której używamy symboli elementów składniowych wspomnianych w p. 8.1, (d).

Nazwa elementu	Symbol	Definicja
Boolean factor	⟨BOFAC⟩	⟨BOPRIM⟩|¬⟨BOPRIM⟩
Boolean term	⟨BOTER⟩	⟨BOFAC⟩|⟨BOTER⟩∧⟨BOFAC⟩
Boolean sum	⟨BOSUM⟩	⟨BOTER⟩|⟨BOSUM⟩∨⟨BOTER⟩
implication	⟨IMP⟩	⟨BOSUM⟩|⟨IMP⟩⊃⟨BOSUM⟩
arithmetic left part	⟨ARLEPA⟩	⟨ARVA⟩:=|⟨ARFUID⟩:=
arithmetic assignment statement	⟨ARAST⟩	⟨ARLEPA⟩⟨AREX⟩|⟨ARLEPA⟩⟨ARAST⟩
Boolean left part	⟨BOLEPA⟩	⟨BOVA⟩:=|⟨BOFUID⟩:=
Boolean assignment statement	⟨BOAST⟩	⟨BOLEPA⟩⟨BOEX⟩|⟨BOLEPA⟩⟨BOAST⟩
statement list	⟨STLIST⟩	⟨ST⟩|⟨STLIST⟩;⟨ST⟩
compound statement	⟨COMST⟩	begin ⟨STLIST⟩ end | ⟨LAB⟩:⟨COMST⟩
block	⟨BLOCK⟩	⟨BLOCK HEAD⟩; STLIST end | ⟨LAB⟩:⟨BLOCK⟩

Symbol blockbegin, który może się pojawić w sygnałach błędów, oznacza symbol begin rozpoczynający blok.

Pierwsze cztery z podanych formuł zawierają tylko różnice terminologiczne (nie ma nazwy ⟨Boolean secondary⟩), a terminologia jest analogiczna jak dla wyrażeń arytmetycznych. Definicja instrukcji podstawienia zawiera także informacje, które w opisie języka wzorcowego występują tylko w nieformalnych komentarzach. Element ⟨STLIST⟩ jest wnętrzem instrukcji złożonej lub bloku, które w gramatyce ALGOLu 60 nie jest wyróżnione.

8.3. Drukowanie symboli podstawowych niedostępnych na klawiaturze P

Aby umożliwić czytelne drukowanie wykazu błędów także na automacie z klawiaturą P, w sygnałach błędów używa się omówionej w p. 1.1 transkrypcji symboli podstawowych ALGOLu niedostępnych na klawiaturze P.

PRZYKŁAD

Jeżeli początek programu ma postać

```
begin
 real całka;
```

to przytoczony fragment programu będzie potraktowany tak, jak znaki

```
begin
 real ca<ka;
```

Wydrukowany przez translator sygnał błędu będzie miał postać

```
E  2  lt  UNEXPECTED
```

ponieważ niedostępny na klawiaturze P znak < transkrybuje się jako lt (p. 1.1).

8.4. Sygnalizacja błędów w czasie działania programu

Niektóre błędy w programie, na przykład

- niezgodność wykazu parametrów aktualnych z odpowiednim wykazem parametrów formalnych,
- niewłaściwa liczba wskaźników w zmiennej,
- niedopuszczalna wartość parametru aktualnego funkcji standardowej,

są wykrywane i sygnalizowane dopiero w czasie działania fragmentu programu zawierającego taki błąd. Sygnał błędu, drukowany zawsze na monitorze, jest symbolem lub zwrotem, który łatwo kojarzy się z rodzajem wykrytego błędu. Na przykład próba obliczenia sqrt(-2) spowoduje wydrukowanie sygnału SQRT, a pierwszy z trzech błędów wymienionych na początku tego punktu spowoduje wydrukowanie sygnału PARAMETER LIST.

Po wydrukowaniu sygnału błędu program zatrzymuje się. Dlatego sygnały takie nazywamy sygnałami zatrzymania. Wykaz sygnałów zatrzymania podano wraz z wyjaśnieniami w p. 11.3.

Niektóre błędy można próbować pominąć (zlecenie "skip" - p. 11.5), przy czym skutki pominięcia są w pełni określone; na przykład pominięcie błędu sygnalizowanego jako SQRT powoduje przyjęcie liczby 0.0 jako wartości funkcji sqrt i kontynuowanie obliczeń. Odpowiednie informacje podano również w p. 11.3.

8.5. Błędy nie wykrywane

Niektóre błędy formalne znajdujące się w programie nie są wykrywane ani w czasie tłumaczenia, ani w czasie wykonywania programu, więc oczywiście nie są sygnalizowane. Niżej podajemy pełny opis takich błędów i ich skutków.

b1. Jeżeli w czasie wykonywania działania na liczbach całkowitych przekroczono przedział liczb całkowitych (p. 1.0, I1), albo wykonano dzielenie całkowite przez zero, to wynik działania nie jest określony.

b2. Jeżeli w treści funkcji F nie wykonano podstawienia pod nazwę F, to wartość F nie jest określona.

b3. Jeżeli PA jest parametrem aktualnym odpowiadającym parametrowi formalnemu PF i są spełnione następujące warunki:
- PF ma jedną ze specyfikacji real, integer lub Boolean,
- PF nie jest umieszczony w zbiorze wartości,
- treść procedury zawiera podstawienie pod PF,
- PA nie jest zmienną,

to wykonanie podstawienia pod PF nie ma żadnych skutków, w szczególności nie powoduje nadania wartości żadnej zmiennej.

8.6. Ślad dynamiczny i ślad retroaktywny programu

Śladem programu nazywamy wykaz nazw procedur niestandardowych lub etykiet napotkanych w czasie pracy programu. Wykaz wydrukowany w czasie pracy programu nazywamy śladem dynamicznym, a wykaz drukowany na specjalne żądanie po zatrzymaniu pracy programu nazywamy śladem retroaktywnym. Ślad może być pomocą np. przy szukaniu błędów sygnalizowanych w czasie działania programu.

Aby skorzystać z możliwości drukowania śladu, należy na czas czytania przez translator taśmy programu - lub jej wybranych fragmentów - wcisnąć klawisze nr 3 i 4 - jeden lub oba - pulpitu maszyny.

Jeżeli w czasie czytania etykiety umieszczonej przed instrukcją klawisz nr 3 jest wciśnięty, to do programu zostanie dołączony rozkaz warunkowego drukowania tej etykiety.

Jeżeli w czasie wykonywania tak przetłumaczonego programu klawisz nr 3 jest wciśnięty, to przed każdym wykonaniem instrukcji z odpowiednią etykietą na aktualnym wyjściu będzie wydrukowana nazwa tej etykiety, każdorazowo w nowym wierszu, poprzedzona gwiazdką.

Klawisz nr 4 działa analogicznie, jak klawisz 3, ale w odniesieniu do procedur i funkcji niestandardowych. Rozkaz warunkowego drukowania nazwy jest dołączany do opisu procedury lub funkcji w momencie czytania pary symboli

procedure ⟨nazwa⟩

na początku opisu procedury lub funkcji. Jeżeli w czasie działania programu jest wciśnięty klawisz 4, to nazwa procedury jest drukowana po sprawdzeniu zgodności wykazu parametrów formalnych z wykazem parametrów aktualnych wywołania tej procedury.

Jeżeli program zawiera możliwość drukowania śladu (tzn. przetłumaczono go przy wciśniętych klawiszach 3 lub 4), to po zatrzymaniu tego programu - na przykład wskutek wykrycia błędu - można wydrukować na monitorze 10 ostatnich wierszy śladu dynamicznego. Służy do tego zlecenie "ditr" (p. 11.1). Po wykonaniu tego zlecenia maszyna drukuje sygnał END. Sygnał zatrzymania SORRY oznacza, że w programie nie ma możliwości drukowania śladu.

Typowy sposób korzystania ze śladu retroaktywnego polega na tłumaczeniu programu przy wciśniętych klawiszach 3 i 4 i wykonaniu obliczeń po zwolnieniu tych klawiszy. Ślad jest wówczas uwzględniony, ale nie jest drukowany. Po wykryciu błędu zlecenie "ditr" daje lokalizację błędu z dokładnością zależną od liczby i położenia etykiet i odwołań do procedur i funkcji niestandardowych.

Pisząc długie programy warto użyć pewnej liczby etykiet niepotrzebnych z punktu widzenia organizacji programu, ale użytecznych przy liczeniu wierszy (p. 8.8) i w razie drukowania śladu.

Program przetłumaczony z uwzględnieniem śladu zajmuje zawsze więcej pamięci i działa wolniej niż po przetłumaczeniu bez uwzględniania śladu.

W czasie niezależnego tłumaczenia procedury (p. 1.9) translator działa zawsze tak, jak gdyby klawisze 3 i 4 były zwolnione, bez względu na rzeczywiste położenie tych klawiszy.

8.7. Korygowanie i kopiowanie programu przy pomocy translatora

W praktyce korygowanie programów wykonuje się często, i to nie tylko w celu poprawienia programu błędnego; niekiedy są potrzebne wersje tego samego programu różniące się kilkoma instrukcjami. Translatory ALGOLu 1204 zawierają pewne proste możliwości korygowania programów i procedur w czasie tłumaczenia, co zmniejsza znacznie ilość czasu automatu OPTIMA potrzebnego do poprawiania i zmiany taśm.

Postępowanie związane z korygowaniem programu lub procedury przy użyciu translatora jest następujące:

a. Przygotowuje się wykaz poprawek (p. 8.9), który może zawierać żądania usunięcia i zamiany wiersza lub ciągu wierszy tłumaczonego tekstu, albo też żądanie wykonania wstawki w tekście.

b. Na oddzielne żądanie translator czyta i zapamiętuje wykaz poprawek - bezpośrednio z monitora lub z uprzednio przygotowanej taśmy (p. 11.1).

c. Po przeczytaniu wszystkich poprawek - ewentualnie z kilku kolejnych taśm - można żądać tłumaczenia programu lub procedury z uwzględnieniem uprzednio zapamiętanego wykazu poprawek. W tym przypadku translator analizuje znaki na przemian z taśmy i z wykazu poprawek, pomijając przy tym odpowiednie wiersze oryginalnego tekstu.

Korzystając z tych możliwości można na przykład przetłumaczyć i wypróbować kilka wersji tego samego programu nie dysponując w pełni poprawną taśmą tego programu. Jeżeli jest potrzebna taśma skorygowanego programu, to w czasie działania translatora powinien być wciśnięty klawisz nr 1 pulpitu; w tym przypadku translator perforuje formalnie poprawny program (lub opis procedury), jeśli tylko w dalszym ciągu tłumaczenia nie wykrył błędu. Jeśli klawisz 1 jest wciśnięty w czasie zwykłego tłumaczenia (tzn. bez korygowania), to otrzymuje się kopię taśmy programu. *Znaki zapytania kończące odcinki korygowanej* lub kopiowanej taśmy są zachowane, a po każdym z nich translator perforuje dodatkowo

50 odstępów i 100 znaków pustych. Znak zapytania, odstępy i znaki puste są również perforowane po ostatnim end programu lub po średniku kończącym niezależnie tłumaczoną procedurę (p. 1.9).

Stan klawisza 1 jest badany przy czytaniu każdego znaku, co umożliwia wykonanie kopii jakiejś części programu, np. opisu procedury. Wykrycie błędu w czasie tłumaczenia lub wciśnięcie klawisza 2 (p. 8.8) powoduje natychmiastowe przerwanie kopiowania taśmy.

8.8. Numerowanie wierszy i punkty orientacyjne

Ponumerowanie wierszy - wszystkich lub tylko niektórych - jest potrzebne do interpretacji wykazu błędów wykrytych w czasie tłumaczenia (p. 8.1), a także do przygotowania wykazu poprawek (p. 8.9). Numeracja wierszy zaczyna się od jedynki; liczy się tylko wiersze zawierające co najmniej jeden znak widoczny w maszynopisie.

Czynność numerowania wierszy długiego programu można znacznie uprościć drukując w czasie tłumaczenia wykaz tzw. punktów orientacyjnych.

Jeżeli klawisz nr 2 pulpitu jest wciśnięty (por. ostatnie zdanie p. 8.7), to w czasie czytania taśmy programu lub procedury translator drukuje na monitorze (jeśli klawisz 0 jest zwolniony) lub perforuje (jeśli klawisz 0 jest wciśnięty) wykaz tych fragmentów taśmy, które mają jedną z trzech postaci

⟨etykieta⟩ :
procedure ⟨nazwa⟩
for ⟨nazwa⟩

i nie wchodzą w skład łańcucha, komentarza, ani zbioru specyfikacji; są to zatem instrukcje z etykietą, początki opisów procedur i początki instrukcji "dla", czyli miejsca, które łatwo odnaleźć nawet w długim programie. Fragment jednej z trzech wymienionych postaci nazywamy punktem orientacyjnym. Przed każdym punktem orientacyjnym translator drukuje numer tego wiersza tłumaczonego programu lub procedury, w którym występuje pierwszy znak następujący po tym punkcie orientacyjnym. W praktyce będzie to najczęściej po prostu numer wiersza, w którym znajduje się punkt orientacyjny, a wyjątkiem będzie etykieta umieszczona w oddzielnym wierszu.

Numery wierszy drukowane przez translator w czasie tłumaczenia z korygowaniem (p. 8.7) odnoszą się do tekstu przed jego skorygowaniem. Jeżeli punkt orientacyjny lub błąd sygnalizowany przez translator znajduje się w treści poprawki (p. 8.9), to numer wiersza drukowany przez translator odnosi się do najbliższego wiersza, który będzie analizowany po wyczerpaniu tekstu poprawki. Wynika to stąd, że uwzględniając poprawkę translator najpierw pomija przeznaczone do zmiany wiersze tekstu (zwiększając odpowiednio licznik wierszy), a dopiero potem analizuje kolejne znaki treści poprawki (p. 8.9).

Stan klawisza 2 jest badany w czasie czytania każdego punktu orientacyjnego, co umożliwia wydrukowanie wykazu częściowego.

Jeżeli w czasie drukowania wykazu punktów orientacyjnych translator sygnalizuje błędy, to otrzymujemy wiersze wykazu błędów pomieszane z wierszami wykazu punktów orientacyjnych; łatwo można odróżnić te dwa rodzaje wierszy, ponieważ sygnały błędów zaczynają się od litery E (p. 8.1), a punkty orientacyjne - od odstępu.

8.9. Postać wykazu poprawek

Aby wykonać tłumaczenie z korygowaniem (p. 8.7), należy ponumerować (p. 8.8) odpowiednie wiersze programu i na tej podstawie przygotować wykaz potrzebnych poprawek. Opiszemy teraz dopuszczalną postać tego wykazu.

Poprawka jest ciągiem elementów postaci:

⟨nagłówek poprawki⟩
⟨treść poprawki⟩
⟨symbol końca poprawki⟩

Nagłówek poprawki musi być napisany w oddzielnym wierszu. Treść poprawki może zawierać dowolną liczbę wierszy zawierających dowolne znaki z wyjątkiem znaku zapytania napisanego bezpośrednio po zmianie wiersza. Symbolem końca poprawki jest znak zapytania napisany bezpośrednio po zmianie wiersza. Przy usuwaniu wierszy podaje się tylko nagłówek poprawki, a treść i symbol końca pomija się.

Wykaz poprawek jest ciągiem poprawek zakończonym znakiem zapytania.

Podana dalej tabela zawiera m. in. wszystkie możliwe postacie nagłówka poprawki. Litera N oznacza liczbę naturalną zakończoną

odstępem i ewentualnie poprzedzoną odstępami, a litera M oznacza liczbę naturalną zakończoną zmianą wiersza i ewentualnie poprzedzoną odstępami. Litery d,i,r użyte w pierwszej kolumnie tabeli pochodzą odpowiednio od słów "deletion", "insertion" i "replacement". Zamiast liter d,i,r można również używać odpowiednio liter u,w,z pochodzących od słów "usunięcie", "wstawka" i "zamiana".

nagłówek poprawki	interpretacja poprawki	dalszy ciąg poprawki
d M	usunąć wiersz nr M	nie ma
d N M	usunąć wiersze nr N,N+1,...,M (N ≤ M)	nie ma
i M	wstawić treść poprawki po wierszu nr M	⟨treść poprawki⟩ ?
r M	zastąpić wiersz nr M treścią poprawki	⟨treść poprawki⟩ ?
r N M	zastąpić wiersze nr N,N+1,...,M (N ≤ M) treścią poprawki	⟨treść poprawki⟩ ?

Ciąg liczb występujących w nagłówkach kolejnych poprawek musi być rosnący. Jeżeli wykaz poprawek jest wydziurkowany na taśmie, to po znaku zapytania kończącym wykaz należy wydziurkować jeszcze co najmniej 16 odstępów. Wykaz poprawek może być wydziurkowany na kilku kolejnych taśmach, z których każda jest również (w sensie podanej definicji) wykazem poprawek. O tym, czy taśma zawiera pełny wykaz poprawek, czy też jego część, decyduje operator w momencie wprowadzania poprawek do pamięci (p. 11.1).

Poprawki wprowadzane bezpośrednio z monitora maszyny można - w razie omyłki - częściowo korygować jeszcze w czasie ich pisania (p. 11.1). Poprawki wydziurkowane na taśmie można korygować wyłącznie przez reperforację na maszynie OPTIMA.

PRZYKŁAD

Jeżeli tekst

```
begin
 integer i,k
 read(n,y);
 begin
  array a[0:n];
```

```
        real y;
        x:=.0;
        for i:=1 step 1 until n do
         begin
          a[i]:=1;
          a[i]:=a[i-1]×y;
          b[i]:=x:=x+a[i]
        print(a,b)
       end
      end
```

chcemy skorygować tak, aby otrzymać tekst

```
      begin
       integer i,n;
       real x,y;
       read(n,y);
       begin
        array a,b[0:n];
        a[0]:=b[0]:=1.0;
        x:=.0;
        for i:=1 step 1 until n do
         begin
          a[i]:=a[i-1]×y;
          b[i]:=x:=x+a[i]
         end i;
        print(a,b)
       end
      end
```

to należy przygotować następujący wykaz poprawek:

```
      z 2
       integer i,n;
       real x,y;
      ?
      z 5 6
        array a,b[0:n];
        a[0]:=b[0]:=1.0;
      ?
      u 10
```

```
w 12
   end i;
?
?
```

UWAGA 1

Przypominamy (por. akapit poprzedzający tabelę), że liczba napisana na końcu nagłówka poprawki musi być zakończona zmianą wiersza (a nie, na przykład, dwoma odstępami).

UWAGA 2

Jeżeli korygujemy ostatni wiersz programu, to wiersz ten powinien być zakończony (znakiem NL lub LF - jak pozostałe wiersze); brak zakończenia spowoduje wypadnięcie taśmy z czytnika, ponieważ w czasie korygowania translator najpierw pomija wiersz przeznaczony do zmiany, a dopiero potem analizuje treść poprawki.

UWAGA 3

Para znaków tworzących symbol końca poprawki musi być parą również na taśmie, tzn znaki te można rozdzielić na taśmie tylko znakiem pustym lub znakami ML lub DL (trzy wymienione znaki są znakami sterującymi czytnika maszyny cyfrowej, która nie dostarcza ich translatorowi). Ograniczenie to wynika stąd, że treść poprawki może zawierać dowolne znaki, także znaki zapytania i zmiany wiersza.

ROZDZIAŁ 9. Maszyna i translatory

9.0. Typowe zestawy maszyny ODRA 1204

Wszystkie zestawy maszyny ODRA 1204, dla których można pisać programy w ALGOLu 1204, zawierają taką samą jednostkę centralną z pamięcią operacyjną o pojemności 16384 słów maszynowych i pracują pod kontrolą systemu obsługi typu MASON (Mało Aktywny System Obsługi i Nadzoru). Różnice pomiędzy zestawami sprowadzają się do liczby i rodzaju urządzeń zewnętrznych.

Minimalny zestaw maszyny, który w dalszym ciągu będziemy oznaczać literą M, zawiera następujące urządzenia zewnętrzne:

Urządzenie	Sposób podłączenia do maszyny nr kanału - nr urządzenia
czytnik taśmy	1 - 1
perforator taśmy	1 - 2
monitor	1 - 3

Zestaw M z tak podłączonymi urządzeniami może pracować pod kontrolą systemu obsługi i nadzoru MASON (p. rozdz. 10). Pod kontrolą tego systemu działają translatory A i B ALGOLu 1204 (p. 9.1), uwzględniające pewne cechy ograniczonego zestawu maszyny.

Istotnie większe możliwości mają zestawy D. Wspólną cechą wszystkich zestawów D jest dodatkowa pamięć bębnowa o pojemności co najmniej 65536 słów, a co najwyżej 262144 słów maszynowych. Aby określić możliwe zestawy D, opiszemy zestaw najmniejszy i największy; każdy zestaw pośredni będziemy również nazywać zestawem D.

Najmniejszy zestaw D określa tabela poniżej:

Urządzenie	Sposób podłączenia do maszyny nr kanału - nr urządzenia
czytnik taśmy	2 - 1
perforator taśmy	2 - 2
monitor	2 - 3
pamięć bębnowa	1 - 4

Największy zestaw D powstaje z najmniejszego przez dołączenie następujących trzech urządzeń dodatkowych:

Urządzenie	Sposób podłączenia do maszyny nr kanału - nr urządzenia
drugi czytnik taśmy	1 - 1
drugi perforator taśmy	1 - 2
drukarka wierszowa	2 - 4

Każdy z możliwych zestawów D może pracować pod kontrolą systemu obsługi MASON (D), który pod względem wykonywanych czynności jest rozszerzeniem systemu MASON dla zestawu M; rozszerzenie jest związane głównie z administracją biblioteki bębnowej systemu i obsługą urządzeń dodatkowych (p. 11.5).

Pod kontrolą systemu MASON (D) działa translator D ALGOLu 1204, który wykonuje na zestawach D m. in. wszystkie czynności wykonywane przez translatory A i B oraz podprogramy PP (p. 9.1) na zestawie M.

Z powyższego wynika, że program w ALGOLu 1204 działający na zestawie M maszyny będzie tak samo działał na dowolnym zestawie D.

Obsługa zestawu D jest istotnie wygodniejsza w porównaniu z zestawem M, ponieważ często używane programy i translator D można umieścić w bibliotece bębnowej (p. 11.5) eliminując w ten sposób używanie niektórych długich taśm. Zestaw D ma także istotnie większe możliwości obliczeniowe i wydawnicze, szczególnie zestaw z drukarką wierszową.

9.1. (M) Translatory A, B i podprogramy pomocnicze PP

Dysponując zestawem M, można używać jednego z dwóch translatorów, których taśmy binarne są oznaczone symbolami

ALGOL 1204 (A)
ALGOL 1204 (B)

Z translatorem B jest związana dodatkowa taśma tzw. podprogramów pomocniczych oznaczona symbolem ALGOL 1204 (PP).

Różnice pomiędzy translatorem A i translatorem B są następujące:

r1. Korzystając z translatora A można kolejno tłumaczyć i wykonywać obliczenia według programów przetłumaczonych.

r2. Korzystając z translatora B można kolejno tłumaczyć i produ-

kować taśmy binarne przekładów programów; obliczenia można wykonywać dopiero po wprowadzeniu do pamięci maszyny podprogramów pomocniczych PP i taśmy binarnej przekładu programu.

r3. Translator B może tłumaczyć programy ok. 2 razy dłuższe niż translator A. Długość programu liczymy tutaj bez rezerwacji dynamicznej; jeśli liczyć także rezerwację dynamiczną, to możliwości obu translatorów są dokładnie takie same: maksymalna liczba słów maszynowych zajętych przez przekład programu i wszystkie jego zmienne dla obu translatorów przekracza 11000.

Taśmy A, B, PP i taśmy programów wyprodukowane przez translator B wprowadza się do pamięci maszyny za pomocą systemu obsługi MASON (zlecenie "load" - p. 11.5).

9.2. (D) Translator D

Chcąc korzystać z możliwości zestawu D, należy używać translatora D; używanie translatorów A lub B pod kontrolą bębnowego systemu obsługi MASON (D) jest również możliwe, jeżeli tylko programy tłumaczone nie korzystają ze specyficznych możliwości zestawu D. Ośrodek obliczeniowy dysponujący zestawem D powinien jednak używać translatora D, gdyż taśma D po umieszczeniu w bibliotece bębnowej (p. 11.5) zastępuje pod każdym względem taśmy A, B i PP, przy czym wszystkie czynności związane z wprowadzaniem translatora i podprogramów z biblioteki bębnowej do pamięci operacyjnej są zautomatyzowane (po jednorazowym wykonaniu zlecenia postaci "load NP", gdzie NP jest nazwą biblioteczną translatora - p. 11.5). Straty czasu związane z wielokrotnym wprowadzaniem translatora do pamięci, dość duże na zestawie M, na zestawie D są nieznaczne.

Maksymalna liczba słów pamięci operacyjnej zajętych przez program i wszystkie jego zmienne jest nieco mniejsza, niż dla translatorów A i B, ale również przekracza 11000.

9.3. Uruchamianie translatorów i programów

Obsługa maszyny pracującej pod kontrolą systemu obsługi MASON (zwykłego lub bębnowego) polega m. in. na konwersacji pomiędzy operatorem maszyny i systemem obsługi; operator pisze na monitorze zlecenie wykonania pewnej pracy, a system na ogół odpowiada wydrukowaniem sygnału, który może być np. informacją o

wykonaniu tej pracy, albo o niemożności jej wykonania bez spełnienia jakiegoś warunku. System obsługi informuje także operatora o ważniejszych zdarzeniach (np. zakończenie działania programu, błąd urządzenia zewnętrznego, dzielenie przez 0.0 itd. - p. 11.3 i 11.5).

Operator może pisać na monitorze dwa rodzaje zleceń: standardowe (p. 11.5), tzn. wykonywane przez sam system obsługi i niestandardowe, tzn. takie, które system obsługi przekazuje do rozpoznania translatorowi (ogólniej - programowi aktualnie znajdującemu się w pamięci operacyjnej). Ważniejsze szczegóły dotyczące konwersacji pomiędzy operatorem i systemem obsługi podano w p. 10.2.

Tak więc do opisania obsługi translatora wystarczy podać wykaz zleceń niestandardowych rozpoznawanych przez translator. Zlecenia dla translatorów ALGOLu 1204 podano w p. 11.1, a zlecenia standardowe systemu - w p. 11.5.

Wszystkie translatory ALGOLu 1204 i tworzone przez nie programy można po jednorazowym wprowadzeniu uruchamiać wielokrotnie; jedynym wyjątkiem jest przypadek tłumaczenia dużego programu translatorem A lub B (sygnał LOAD ALGOL - p. 11.3).

Wykonując długotrwałe obliczenia warto pamiętać o możliwości przerwania pracy programu, zapamiętania stanu obliczeń na taśmie binarnej lub w pamięci bębnowej (p. 11.5) i późniejszej kontynuacji obliczeń.

9.4. Interpretacja sygnałów systemu obsługi, translatorów i programów

Wszystkie sygnały związane z ALGOLem 1204 są - przy niewielkiej wprawie i znajomości terminologii angielskiej - na ogół łatwo zrozumiałe bez żadnych dodatkowych informacji. W przypadku jakichkolwiek trudności z interpretacją sygnału, należy przejrzeć skorowidz alfabetyczny znajdujący się na końcu książki. Skorowidz taki znajduje się również na końcu [3], a jego uzupełnienie dla zestawu D - na końcu [4]. W skorowidzu można znaleźć co najmniej wskazówkę ułatwiającą znalezienie wyjaśnienia, a niekiedy - samo wyjaśnienie (por. np. SIMAREX).

ROZDZIAŁ 10. Systemy obsługi MASON i MASON (D)

10.0. Informacje wstępne

Niektóre informacje o systemach MASON i MASON (D) podano w rozdziale 9. Tutaj omówimy dalsze własności tych systemów, w szczególności obsługę urządzeń zewnętrznych, tworzenie biblioteki bębnowej itp.

Z punktu widzenia operatora system obsługi jest programem uzupełniającym układy elektroniczne maszyny i zapewniającym jej wygodną obsługę metodą konwersacji prowadzonej między operatorem i systemem. Systemy MASON i MASON (D) wymagają zajęcia dla swoich wewnętrznych potrzeb 1536 komórek pamięci operacyjnej maszyny; do dyspozycji translatorów pozostaje więc ponad 90% pamięci. Po początkowym przygotowaniu do pracy (p. 10.1) system obsługi powinien pracować stale; nie należy go zatrzymywać z pulpitu sterowania nawet w przypadku przerwy w obliczeniach. Ważniejsze czynności wykonywane przez dwa omawiane tutaj systemy obsługi są następujące:

a. czytanie i wykonywanie poleceń pisanych przez operatora na monitorze w momentach dowolnie wybranych przez operatora, niezależnie od czynności programu lub translatora,
b. sygnalizacja na monitorze ważniejszych zdarzeń - np. błędów - zachodzących w urządzeniach zewnętrznych i w programie,
c. organizacja jednoczesnej pracy urządzeń zewnętrznych - np. czytnika i drukarki wierszowej - i programu lub translatora,
d. administracja biblioteki bębnowej zestawu D.

Ogólne informacje o konwersacji systemu obsługi i translatorów ALGOLu 1204 z operatorem podano już w p. 9.3. Należy dodać, że w konstrukcji systemów MASON i MASON (D) nie wyróżniono w żaden sposób translatorów ALGOLu 1204. Oznacza to, że pod kontrolą tych samych systemów mogą również pracować inne translatory, uruchamiane równie wygodnie, jak translatory ALGOLu 1204.

System MASON (D) nadzorujący pracę zestawu D maszyny (p. 9.0) musi być zapisany w wybranym uprzednio (p. 10.6) obszarze pamięci bębnowej (ok. 4000 słów); obszar można wybrać dowolnie, co umożliwia koegzystencję zapisów bębnowych (także biblioteki) z zapi-

ami jakiegoś innego systemu. Trwałość zapisu systemu i jego biblioteki bębnowej praktycznie nie jest ograniczona (przy właściwej konserwacji i eksploatacji maszyny). W razie potrzeby zapisy można łatwo odtworzyć wprowadzając odpowiednie taśmy binarne (p. 10.6 i 10.7).

Po początkowym uruchomieniu oba omawiane systemy zachowują się tak samo (z punktu widzenia operatora), ale czynności przygotowawcze są różne. Dlatego najpierw omówimy te czynności.

10.1. Wprowadzenie i uruchomienie systemu obsługi

Taśma systemu MASON dla zestawu M składa się z dwóch części: pierwsza zawiera program wprowadzający, a druga - ciąg rozkazów tworzących system obsługi. Pierwszą część czyta program stały maszyny, a druga jest czytana po naciśnięciu klawisza START na pulpicie sterowania. Jeżeli w czasie czytania drugiej części nie wykryto błędu, to system uruchamia się automatycznie. Ujawnia się to miganiem lampek PISZ monitora, które gasną po naciśnięciu klawisza ZO; system jest wtedy gotowy do konwersacji z operatorem (p. 10.2).

Do uruchomienia systemu MASON (D) służy program STARTER wyprodukowany bezpośrednio po zapisaniu systemu na bębnie (p. 10.6). Program STARTER wprowadza się za pomocą programu stałego maszyny i uruchamia przez naciśnięcie klawisza START. Po upływie ok. 4 sekund pracy programu STARTER maszyna zachowuje się tak samo, jak opisano powyżej dla zestawu M.

Brak migania lampek PISZ po wprowadzeniu systemu oznacza wykrycie jakiegoś błędu (taśmy, czytnika, pamięci bębnowej lub maszyny).

Bezpośrednio po uruchomieniu systemu jego zegar wskazuje 0 godzin, 0 minut i 0 sekund. Zegar można ustawić dowolnie pisząc zlecenie postaci "pt G M S" - p. 11.5. Stan zegara jest uaktualniany co sekundę.

10.2 Pisanie zleceń

Pisanie tekstu zlecenia należy zawsze poprzedzić naciśnięciem klawisza ZO monitora; jeżeli system nie drukuje w tym momencie sygnału, to zapala lampki PISZ, sygnalizując w ten sposób gotowość do czytania znaków pisanych na monitorze przez operatora.

Do pisania tekstu zlecenia można używać tylko małych liter

łacińskich, cyfr i odstępów, a na zakończenie zlecenia należy napisać jeden z dopuszczalnych symboli końca, którymi są znak pusty, podwójny odstęp, nowa linia, wysunięcie papieru, kropka, przecinek, minus, znak mnożenia, nawias łańcuchowy otwierający i znak podkreślania. Pozostałe znaki są traktowane jako błędne: napisanie któregokolwiek z nich powoduje skasowanie całego napisanego dotąd tekstu zlecenia i wydrukowanie sygnału złożonego ze znaku zapytania i zmiany wiersza. Tak samo są sygnalizowane zlecenia napisane do końca, ale błędne lub w danym stanie maszyny niewykonalne (p. 10.4).

Pojedyncze odstępy służą do oddzielania kolejnych części zlecenia. Wszystkie odstępy napisane na początku zlecenia są pomijane.

Zlecenie puste, tzn. złożone tylko z symbolu końca (różnego od podwójnego odstępu) oznacza rezygnację operatora ze zgłoszonego zamiaru pisania zlecenia; system gasi lampki PISZ i na tym kończy wykonywanie zlecenia pustego.

Zlecenie niepuste składa się z nagłówka i wykazu parametrów; wykaz parametrów może być pusty. Nagłówek zlecenia jest co najwyżej czteroznakowym ciągiem liter lub cyfr zaczynającym się od litery. Niepusty wykaz parametrów oddziela się od nagłówka pojedynczym odstępem; tak samo oddziela się kolejne parametry, a po ostatnim parametrze należy napisać symbol końca zlecenia.

Wykonanie zlecenia niepustego powoduje m. in. wydrukowanie jakiegoś sygnału (p. 11.1 i 11.5). Każdy sygnał zaczyna się od odstępu i kończy się nową linią w położeniu małych liter (potrzebnym do pisania zleceń); jedynie sygnał czasu po uruchomieniu translatora lub programu kończy się odstępem i minusem. Oto przykład zarejestrowanej na monitorze zestawu D konwersacji operatora z systemem obsługi:

```
load ta1     ta1 LOADED
rbt  p 41 1,
 END
w    STOP
del xpr   D=78497
dump xbis    xbis DUMPED
load most    most NOT IN LIBRARY
```

Biorąc pod uwagę informacje podane w p. 11.1, 11.3, 11.5 i 11.6 można na podstawie przytoczonego tekstu jednoznacznie

odtworzyć czynności wykonane przez operatora i maszynę.

Czytanie zleceń i drukowanie sygnałów nie zakłóca przebiegu innych czynności wykonywanych przez maszynę, pod oczywistym warunkiem, że treścią zlecenia nie jest żądanie "zakłócenia" - np. przerwania pracy programu.

10.3. Obsługa urządzeń zewnętrznych

Każde z urządzeń zewnętrznych maszyny ma klawisz oznaczony symbolem ZO (w drukarce wierszowej - napisem PRZYDZIEL). Naciśnięcie tego klawisza powoduje natychmiastową reakcję systemu; jego działanie zależy od rodzaju urządzenia i stanu tego urządzenia. Reakcję na ZO monitora opisano w p. 10.1 i 10.2. W innych urządzeniach - z wyjątkiem pamięci bębnowej - kolejne naciskanie ZO oznacza kolejno zezwolenie i zakaz używania urządzenia: jeśli obowiązywał zakaz, to ZO oznacza zezwolenie, a w przeciwnym razie naciśnięcie ZO oznacza zakaz. Otrzymując zezwolenie używania urządzenia system zapala lampkę P1 na pulpicie tego urządzenia i gasi ją w przypadku zakazu. Tak więc po przygotowaniu urządzenia do pracy (np. po założeniu taśmy do czytnika) trzeba nacisnąć ZO na pulpicie tego urządzenia. Jeśli urządzenie przestało być potrzebne (np. operator zdejmuje taśmę z czytnika) albo chwilowo nie powinno być używane przez system (np. zmiana taśmy w perforatorze lub papieru w drukarce), to należy nacisnąć ZO ponownie.

System wprowadza automatycznie zakaz używania urządzenia po wykryciu jego błędnej pracy i sygnalizuje błąd na monitorze (sygnały podajemy dalej). Bezpośrednio po uruchomieniu systemu (p. 10.1) obowiązuje zakaz używania wszystkich urządzeń, z wyjątkiem pamięci bębnowej.

Jeżeli wykonanie zlecenia operatora wymaga użycia urządzenia, którego użycie jest zakazane (lampka P1 jest zgaszona), to system czeka z wykonaniem zlecenia do chwili naciśnięcia odpowiedniego klawisza ZO albo do chwili napisania zlecenia "stop" (lub naciśnięcia klawisza oznaczonego napisem PRZERWANIE PROGRAMU). Sygnał czekania drukowany na monitorze zawiera nazwę urządzenia i słowo WAIT oznaczające czekanie. Wykazy tych sygnałów dla zestawów M i D są następujące:

Zestaw M	Zestaw D
READER WAIT	READER 1 WAIT
PUNCH WAIT	READER 2 WAIT
	PUNCH 1 WAIT
	PUNCH 2 WAIT
	PRINTER WAIT

READER oznacza czytnik, PUNCH - perforator, a PRINTER - drukarkę.

Jeżeli lampka P1 urządzenia pali się i system wykryje błąd w tym urządzeniu, to drukuje sygnał zawierający nazwę urządzenia i słowo ERROR (błąd). Wykazy sygnałów są następujące:

Zestaw M	Zestaw D
READER ERROR	READER 1 ERROR
PUNCH ERROR	READER 2 ERROR
	PUNCH 1 ERROR
	PUNCH 2 ERROR
	PRINTER ERROR

Bliższe informacje o rodzaju błędu są wyświetlane na pulpicie urządzenia; na przykład lampka nr 3 na pulpicie czytnika oznacza wykrycie na taśmie z danymi lub z programem w ALGOLu znaku o parzystej liczbie dziurek, lampka nr 5 - na ogół niewłaściwą obsługę urządzenia (blokada w czasie pracy).

Sygnały błędów pamięci bębnowej są bardziej szczegółowe i mają postać DEn, gdzie n (=0,3,4,5,6,21) jest numerem błędu (jest to numer wskaźnika, który nie powinien się zapalić). Po wykryciu błędu pamięci bębnowej system czeka (nie sygnalizując czekania) na naciśnięcie ZO pamięci, przy czym nie wykonuje zleceń; zlecenie może być przeczytane, ale jego analiza i wykonanie będą odroczone do czasu naciśnięcia ZO. Po naciśnięciu tego klawisza system powtarza operację bębnową, w czasie której został wykryty błąd i wykonuje prace uprzednio odroczone.

Pojawienie się sygnału błędu pamięci bębnowej wymaga zwykle interwencji konserwatora, któremu należy przekazać informację o numerze błędu. Jedynym wyjątkiem jest sygnał postaci

DE21, AB1-AB2

oznaczający, że system ma wykonać operację zapisu do obszaru bębnowego o adresach AB1 do AB2 i obszar ten jest zablokowany; należy w tym przypadku odblokować ten obszar (zwolnić odpowied-

nie klawisze blokady) i nacisnąć ZO pamięci bębnowej, co spowoduje wykonanie zapisu. Jeżeli regulamin ośrodka obliczeniowego nie zezwala na wykonanie zapisu do wskazanego obszaru, to należy naciśnięcie ZO pamięci bębnowej poprzedzić naciśnięciem klawisza PRZERWANIE PROGRAMU; operacja zapisu nie będzie wykonana, a możliwość konwersacji z systemem zostanie przywrócona. Poza opisanymi przypadkami naciskanie ZO pamięci bębnowej jest pomijane przez system.

Błąd monitora jest sygnalizowany miganiem lampek PISZ i czekaniem na naciśnięcie ZO monitora. W praktyce prawie zawsze potrzebna będzie interwencja konserwatora, jeżeli tylko błąd nie był wynikiem blokady monitora (klawisz BLM monitora powinien być zwolniony).

10.4. Stany maszyny i wykonalność zleceń

Pracując pod kontrolą systemu MASON lub MASON (D) maszyna znajduje się w jednym i tylko jednym z wymienionych dalej stanów; stanom maszyny nadano nazwy wymienione w tabeli poniżej.

Nazwa stanu	Czynności maszyny
STOP	Czekanie na pracę.
LOADING	Wprowadzanie programu (z taśmy binarnej lub z biblioteki bębnowej) do pamięci operacyjnej.
RUNNING	Praca translatora lub programu.
DUMPING	Wyprowadzanie zawartości zarezerwowanej pamięci operacyjnej (na taśmę binarną lub do pamięci bębnowej).
TESTING	Porównywanie taśmy binarnej z zawartością pamięci (operacyjnej lub bębnowej).
TAKING	Wprowadzanie do biblioteki bębnowej taśmy programu specjalnego - np. translatora D AIGOLu 1204.

Po uruchomieniu systemu (p. 10.1) maszyna znajduje się w stanie STOP. Przejście do każdego z pozostałych stanów następuje na żądanie operatora. Ponowne przejście do stanu STOP jest automatyczne - po wykonaniu zleconej pracy, ale może również nastąpić wcześniej na żądanie operatora (por. "stop", p. 11.5).

W razie wątpliwości, w jakim stanie znajduje się maszyna, należy napisać zlecenie "now"; powoduje to wydrukowanie nazwy aktualnego stanu maszyny. Dodatkowo, jeśli system czeka w tym czasie na przygotowanie urządzenia zewnętrznego, to drukuje odpowiedni sygnał czekania (p. 10.3).

Każde zlecenie jest wykonalne tylko w określonym stanie (jednym lub kilku) maszyny. Odpowiednie informacje podano w p. 11.5; na ogół są to informacje oczywiste: np. zlecenia uruchomienia programu są wykonalne tylko w stanie STOP, a zlecenie "now" - w każdym stanie.

Przy przejściu ze stanu STOP do stanu RUNNING i przy przejściu odwrotnym system drukuje stan zegara (rejestracja czasu rozpoczęcia i zakończenia tłumaczenia lub wykonywania programu); w systemie MASON (D) drukowanie stanu zegara odbywa się warunkowo (p. 11.5, "on", "off" i "pt").

10.5. (D) Podział pamięci bębnowej

Pamięć bębnowa zestawu D, który ma pracować pod kontrolą systemu MASON (D), dzieli się na cztery rozłączne obszary, które dla skrócenia opisów oznaczymy tutaj symbolami odpowiednio ROB, BIBL, SYST, X. Wielkość i położenie obszarów SYST i X ustala się w momencie zapisywania systemu w pamięci bębnowej (p. 10.6). Obszary ROB i BIBL wypełniają pozostałą część pamięci bębnowej. Szczegóły opisujemy poniżej.

ROB jest obszarem roboczym, zaczynającym się zawsze od komórki o adresie zero i dostępnym bez ograniczeń dla wszystkich programów pracujących pod kontrolą systemu. Zapisy wykonane w tym obszarze mogą być chronione za pomocą mechanizmu blokady pamięci bębnowej.

BIBL jest obszarem bibliotecznym, zawierającym programy identyfikowane za pomocą nazw (p. 10.7). Zmiana zapisu w tym obszarze może być wykonana tylko na żądanie operatora; w szczególności żaden program pracujący pod kontrolą systemu nie może zmienić zawartości obszaru BIBL. Bezpośrednio po zapisaniu systemu na bębnie obszar BIBL jest pusty.

SYST jest obszarem zawierającym zapis systemu MASON (D); SYST zawiera ok. 4000 komórek bębnowych. Obszar ten jest niedostępny dla programów pracujących pod kontrolą systemu.

X jest ostatnim obszarem (komórki o największych adresach), który nie jest używany pod kontrolą systemu MASON (D). Obszar ten może być pusty w ośrodku obliczeniowym stosującym tylko system MASON (D), a w innych ośrodkach może zawierać np. zapisy innych systemów.

Łączna wielkość obszarów ROB i BIBL po zapisaniu systemu na

bębnie jest ustalona, przy czym każde zwiększenie BIBL powoduje takie same zmniejszenie ROB; zmniejszenie BIBL, wykonalne tylko przy pomocy zlecenia "del" (p. 11.5), powoduje takie same zwiększenie ROB.

Translator D ALGOLu 1204 jest tzw. programem specjalnym, co oznacza m. in., że można tego translatora używać tylko po uprzednim umieszczeniu jego zapisu w obszarze BIBL (zlecenie "take"); wymaga to zajęcia ok. 13000 komórek pamięci bębnowej.

10.6. (D) Zapisanie systemu MASON (D) na bębnie

Przed zapisaniem systemu na bębnie należy ustalić wielkość obszaru bębnowego, który ma być administrowany przez system (suma obszarów ROB, BIBL, SYST - p. 10.5). Do zapisania systemu służy taśma oznaczona symbolem MASON (D); pierwsza część taśmy zawiera program wprowadzający i jest czytana przez program stały maszyny. Program wprowadzający uruchamia się przez naciśnięcie klawisza START. Po poprawnym przeczytaniu całej taśmy, na monitorze jest drukowany sygnał

DRUM LIMIT:

i zapalają się lampki PISZ; należy napisać na monitorze co najwyżej 6-cyfrową dziesiętną liczbę całkowitą oznaczającą wielkość obszaru przydzielanego systemowi i znak pusty (blank). Zamiast znaku pustego można napisać literę "k"; w tym przypadku napisana liczba będzie potraktowana tak, jak liczba 1024 razy większa. Zamiast "256k" można również napisać jeden znak pusty (256k jest pojemnością największej pamięci bębnowej w zestawie D).

Po kilku sekundach pracy maszyny jest drukowany sygnał

STARTER PUNCHING (p) OR CHECKING (c) ?

i zapalają się lampki PISZ. Napisanie litery "p" powoduje wyprodukowanie krótkiej taśmy, którą należy oznaczyć symbolem STARTER; jest to program służący do uruchomienia systemu (p. 10.1). Poprawność taśmy można sprawdzić zakładając ją do czytnika i pisząc literę "c" w odpowiedzi na pytanie "pc?" wydrukowane przez maszynę. Sygnał ERROR oznacza błąd (taśmy, czytnika, perforatora lub maszyny).

Czynności produkowania i sprawdzania taśmy można powtarzać dowolnie wiele razy, pisząc odpowiednio literę "p" lub "c" w odpowiedzi na "pc?"; napisanie innego znaku powoduje uruchomienie

programu STARTER znajdującego się w pamięci maszyny (p. 10.1).

Zatrzymanie się maszyny w czasie wykonywania opisanych czynności oznacza błąd (operatora, maszyny lub jej urządzeń); wszystkie czynności trzeba powtórzyć od początku.

10.7. (D) Biblioteka systemu

Na zlecenie "take" system czyta i umieszcza w bibliotece bębnowej taśmę translatora D oznaczoną symbolem ALGOL 1204 (D). Jeżeli wiadomo, że translator D znajduje się w bibliotece, to w celu wprowadzenia tego translatora do pamięci operacyjnej wystarczy napisać zlecenie postaci "load NP", gdzie NP jest nazwą używanego wydania translatora D. Pierwsze wydanie ma nazwę "ta1", a czwarte – "ta4". Nazwa jest podana na taśmie translatora; system drukuje m. in. tę nazwę po umieszczeniu translatora w bibliotece.

Na przykładzie translatora D opisaliśmy tutaj umieszczanie w bibliotece tzw. programów specjalnych. Są to na ogół translatory lub inne programy napisane w kodzie maszynowym, których tutaj nie rozważamy. Jedną z własności programów specjalnych jest to, że ich nazwa biblioteczna jest z góry ustalona (podana na taśmie programu).

Aby dołączyć do biblioteki dowolny inny program, należy go umieścić w pamięci operacyjnej (np. przetłumaczyć program w ALGOLu) i napisać zlecenie postaci "dump NP", gdzie NP jest nazwą, którą ma być opatrzony zapis biblioteczny programu (p. 11.5).

Wykaz nazw programów umieszczonych w bibliotece, wraz z informacjami dodatkowymi, system drukuje na zlecenie "list".

System zmniejsza obszar biblioteczny (zwiększając jednocześnie obszar roboczy) na zlecenie postaci "del NP", gdzie NP jest nazwą dowolnego programu umieszczonego w bibliotece. Wykonanie tego zlecenia powoduje usunięcie wskazanego programu z biblioteki wraz ze wszystkimi programami dołączonymi po nim i wydrukowanie informacji o aktualnej wielkości obszaru roboczego. Zawartość pamięci operacyjnej nie zmienia się. Zachowane są również wszystkie informacje potrzebne do dalszego działania programu (na zlecenie "go").

ROZDZIAŁ 11. Wykazy

11.0. Kody znaków

Kod znaku w ALGOLu 1204 jest liczbą przyporządkowaną znakowi wyznaczoną przez kod znaku na taśmie i sposób czytania taśmy przez czytnik. Znaki małych liter (ML) i znaki dużych liter (DL) są pomijane przez czytnik. W elektronice czytnika pozostaje jedynie informacja o tym, który z tych znaków był ostatnio czytany. Oznaczmy przez

t8 t7 t6 t5 t4 . t3 t2 t1

jeden rządek taśmy, przy czym ti=1, jeśli na i-tej ścieżce taśmy jest dziurka, a w przeciwnym razie ti=0 ($1 \leq i \leq 8$); kropką oznaczyliśmy położenie dziurki prowadzącej na taśmie. Kod znaku jest liczbą równą

$$(((((K\times 2+t7)\times 2+t6)\times 2+t4)\times 2+t3)\times 2+t2)\times 2+t1$$

gdzie

$$K = \begin{cases} 0, & \text{jeśli ostatnio był przeczytany znak ML} \\ 1, & \text{jeśli ostatnio był przeczytany znak DL} \end{cases}$$

Znaki sterujące (niewidoczne w maszynopisie) oznaczamy następującymi symbolami:

sp	odstęp
NL	nowa linia
LF	wysuw papieru
DL	znak dużych liter
ML	znak małych liter
kor	znak korekty
CR	powrót karetki
LEU	zmiana czytnika
LOA	wyłączenie perforatorów
LO1	włączenie perforatora nr 1
LO2	włączenie perforatora nr 2
NIC	znak bez interpretacji w maszynie OPTIMA
NS	wyłączenie drukowania

SP włączenie pomijania znaków
SPE wyłączenie pomijania znaków
STOP zatrzymanie czytnika maszyny OPTIMA
TAB tabulator
WS włączenie drukowania

W podanej tabeli kolejne kolumny oznaczone symbolami KOD, OP, DR, DRC zawierają odpowiednio kod znaku, znak drukowany na maszynie OPTIMA, na drukarce wierszowej i na nietypowej drukarce wierszowej (p. 12.2). Puste miejsce w kolumnie DR lub DRC oznacza, że znak o odpowiednim kodzie jest pomijany przy wyprowadzaniu na drukarkę.

KOD	OP	DR	DRC	KOD	OP	DR	DRC
0	sp	sp	sp	1	1	1	1
2	2	2	2	3	3	3	3
4	4	4	4	5	5	5	5
6	6	6	6	7	7	7	7
8	8	8	8	9	9	9	9
10	‘	'	)	11	STOP		
12	NS			13	SPE		
14	LEU			15	NL	NL	NL
16	0	0	0	17	␣	0	0
18	s	S	S	19	t	T	T
20	u	U	U	21	v	V	V
22	w	W	W	23	x	X	X
24	y	Y	Y	25	z	Z	Z
26	×	*	*	27	.	.	.
28	WS			29	_		
30	TAB			31	LOA		
32	-	-	-	33	j	J	J
34	k	K	K	35	l	L	L
36	m	M	M	37	n	N	N
38	o	O	O	39	p	P	P
40	q	Q	Q	41	r	R	R
42	CR			43	>	>	Ą
44	LO1			45	.=	=	Ż
46	NIC			47	LF	NL	NL
48	,	,	,	49	a	A	A
50	b	B	B	51	c	C	C

KOD	OP	DR	DRC	KOD	OP	DR	DRC
52	d	D	D	53	e	E	E
54	f	F	F	55	g	G	G
56	h	H	H	57	i	I	I
58	ML			59	<	<	Ł
60	DL			61	SP		
62	LO2			63	kor		
64	sp	sp	sp	65	ω	@	E
66	[	[	(	67	]	]	)
68	?	?		69	+	+	+
70	=	=	=	71	:	:	:
72	(	(	(	73	/	/	/
74	'	"	)	75	STOP		
76	NS			77	SPE		
78	LEU			79	NL	NL	NL
80	)	)	)	81	÷	%	Z
82	S	S	S	83	T	T	T
84	U	U	U	85	V	V	V
86	W	W	W	87	X	X	X
88	Y	Y	Y	89	Z	Z	Z
90	;	;	:	91	¬	#	S
92	WS			93	|		
94	TAB			95	LOA		
96	V	!	C	97	J	J	J
98	K	K	K	99	L	L	L
100	M	M	M	101	N	N	N
102	O	O	O	103	P	P	P
104	Q	Q	Q	105	R	R	R
106	CR			107	≡	$	Ę
108	LO1			109	↑	↑	Ż
110	NIC			111	LF	NL	NL
112	Λ	&	N	113	A	A	A
114	B	B	B	115	C	C	C
116	D	D	D	117	E	E	E
118	F	F	F	119	G	G	G
120	H	H	H	121	I	I	I
122	ML			123	⊃	£	Ł
124	DL			125	SP		
126	LO2			127	kor		

11.1. Zlecenia dla translatorów ALGOLu 1204

Zlecenia dla translatorów są wykonalne tylko w stanie STOP, a ich wykonanie powoduje uruchomienie translatora (przejście ze stanu STOP do stanu RUNNING - p. 10.4). Elementy podanego dalej wykazu zawierają kolejno tekst zlecenia zakończony kropką (która jest jednym z dopuszczalnych symboli końca zlecenia), interpretację zlecenia, a następnie najczęściej drukowane sygnały maszyny wraz z wyjaśnieniami. Często pojawiające się sygnały END i SORRY oznaczają odpowiednio "zakończono wykonywanie" i "nie można wykonać"; dokładna interpretacja tych sygnałów zależy od kontekstu i jest tutaj omówiona.

Napisanie zlecenia uruchamiającego translator A lub B może spowodować wydrukowanie sygnału LOAD ALGOL oznaczającego, że zlecenie można wykonać dopiero po ponownym wprowadzeniu translatora, gdyż jego aktualny zapis w pamięci operacyjnej częściowo zniszczono w czasie czynności wykonywanych poprzednio. Translator D w analogicznej sytuacji nie drukuje żadnego sygnału, ale wprowadza potrzebną część zapisu z biblioteki bębnowej i przystępuje do wykonania zlecenia.

Sygnał TRANSLATE PROGRAM oznacza niewykonalność zlecenia uruchomienia programu przetłumaczonego (z powodu braku zapisu tego programu w pamięci operacyjnej albo z powodu połączenia taśmy wyprodukowanej przez translator B z niewłaściwym wydaniem podprogramów pomocniczych PP).

Niezbędne uwagi dodatkowe podajemy w nawiasach. Symbol (D) występujący po kropce kończącej tekst zlecenia oznacza, że to zlecenie wykonuje tylko translator D.

Oto zapowiedziany wykaz zleceń w porządku odpowiadającym kolejności wykonywania typowych czynności operatorskich.

t. Tłumaczenie programu.

SORRY Programu nie można przetłumaczyć (p. 8.1).

pP rR Program zawiera P słów maszynowych przekładu i może użyć co najwyżej R komórek roboczych w pamięci operacyjnej..
(Jeżeli do tłumaczenia użyto translatora A lub D, to w pamięci operacyjnej znajduje się program gotowy do wykonania - por. "w", "w0", "w1" i "w13". Jeżeli użyto translatora B, to należy wyprodukować taśmę binarną

przekładu - por. "pbt". Jej wprowadzenie do pamięci - zleceniem "load" - należy poprzedzić wprowadzeniem podprogramów pomocniczych PP).

tp. Tłumaczenie procedury (p. 1.9).
SORRY Procedury nie można przetłumaczyć (p. 8.1).
pP Przekład procedury zawiera P słów maszynowych. (por. "outp" i "comp").

rc0. Czytanie wykazu poprawek (p. 8.9) z monitora.
STOP Przeczytano symbol końca wykazu poprawek. Jeżeli trzeba do tego wykazu dołączyć dalsze poprawki, to należy napisać zlecenie "go" (p. 11.5).
E Wykryto błąd w aktualnie czytanej poprawce. Ta błędna poprawka została skasowana. Translator czyta poprawkę od początku, zachowując przeczytaną dotąd część wykazu.
(Znak pusty napisany na monitorze w czasie pisania poprawki jest traktowany jako błąd, a zatem może służyć do skasowania poprawki, jeśli tylko nie napisano jeszcze symbolu końca tej poprawki).

rc1. Czytanie wykazu poprawek (p. 8.9) z taśmy.
STOP Patrz "rc0".
SORRY Czytany wykaz jest błędny.

ct. Tłumaczenie programu z uwzględnieniem zapamiętanego uprzednio wykazu poprawek.
SORRY Nie wprowadzono wykazu poprawek lub nie można przetłumaczyć tekstu skorygowanego.
pP rR Patrz "t".

ctp. Tłumaczenie procedury z uwzględnieniem zapamiętanego uprzednio wykazu poprawek.
SORRY Patrz "ct".
pP Patrz "tp".

pbt. Produkowanie taśmy binarnej przekładu programu za pomocą translatora B lub D.
SORRY W pamięci nie ma przekładu programu.
END Zakończono produkowanie taśmy.

(Wyprodukowana taśma nie zawiera podprogramów pomocniczych, por. "rbt" i "t").

cbt. Sprawdzenie poprawności taśmy wyprodukowanej na zlecenie "pbt".

SORRY Taśma błędna lub w pamięci nie ma przekładu programu.

END Na taśmie nie wykryto błędu.

rbt. (D) Wprowadzenie taśmy binarnej przekładu programu do pamięci

SORRY Taśma błędna albo wyprodukowano ją za pomocą wydania translatora D różnego od wydania aktualnie znajdującego się w bibliotece bębnowej.

END Zakończono wprowadzanie taśmy. Pamięć operacyjna zawiera program gotowy do wykonania - taki sam, jak po przetłumaczeniu programu - por. "t" i "pbt".

outp. Produkowanie taśmy binarnej przekładu procedury, czyli segmentu (por. "tp" i "ctp").

SORRY W pamięci nie ma przekładu procedury.

END. Zakończono produkowanie taśmy.

comp. Sprawdzenie poprawności taśmy wyprodukowanej na zlecenie "outp".

SORRY Taśma błędna lub w pamięci nie ma przekładu procedury.

END Błędu na taśmie nie wykryto.

w. Wykonanie instrukcji

setinput(1); setoutput(1)

i uruchomienie programu od początku.

END Program wykonano do końca - do ostatniego end.
(Inne sygnały podano w p. 11.3).

w0. Wykonanie instrukcji

setinput(0); setoutput(0)

i uruchomienie programu od początku.
(Sygnały jak dla "w").

w1. Wykonanie instrukcji

setinput(1); setoutput(0)

i uruchomienie programu od początku.
(Sygnały jak dla "w").

w13. (D) Wykonanie instrukcji

setinput(1); setoutput(3)

i uruchomienie programu od początku.
(Sygnały jak dla "w").

end. (D) Przejście do ostatniego end programu.
END Zlecenie wykonano.
(Zlecenie to stosuje się, jeżeli program został zatrzymany inaczej, niż przez dojście do ostatniego end; przejście do ostatniego end powoduje wyprowadzenie na drukarkę ostatniego wiersza wyników przeznaczonych na drukarkę, jeżeli tylko wyprowadzenie tego wiersza nie nastąpiło wcześniej).

ditr. Drukowanie śladu retroaktywnego.
END Ślad wydrukowano.
SORRY W programie nie ma możliwości drukowania śladu.

tmon. Przejście do wykonania programu

begin setinput(1); setoutput(0);
L: outchar(inchar); go to L end

bez niszczenia zapisu programu lub translatora w pamięci operacyjnej.
STOP Operator przerwał działanie programu.

mont. Przejście do wykonania programu

begin setinput(0); setoutput(1);
L: outchar(inchar); go to L end

bez niszczenia zapisu programu lub translatora w pamięci operacyjnej.
STOP Operator przerwał działanie programu.

11.2. Słowne opisy błędów

Przekroczenia ograniczeń ilościowych translatora, niektóre błędy semantyczne i niektóre grube błędy gramatyczne w pierwszym etapie tłumaczenia są opisywane za pomocą krótkich zwrotów zaczerpniętych z terminologii angielskiej. Niżej podajemy wykaz tych zwrotów. Symbol ⟨I⟩ oznacza nazwę, a symbol ⟨S⟩ oznacza nazwę lub symbol podstawowy użyty w programie.

DECLIST OVERFLOW

Zbyt wiele opisów lub specyfikacji (informacje o opisach zajmują więcej, niż 2048 komórek pamięci).

IDLIST OVERFLOW

Zbyt wiele różnych nazw (nazwy zajmują więcej, niż 1024 komórek pamięci).

⟨I⟩ IN BOUND PAIR LIST

Wykaz par granicznych w opisie tablic zawiera lokalną nazwę ⟨I⟩.

NUMBER LIST OVERFLOW

Zbyt wiele różnych stałych (więcej, niż 510 różnych stałych całkowitych lub więcej, niż 512 różnych stałych rzeczywistych).

⟨I⟩ OUT OF SCOPE

Nazwy ⟨I⟩ opisanej w programie użyto poza jej obszarem działania.

⟨I⟩ := OUT OF PROCEDURE BODY

Podstawienie pod nazwę procedury ⟨I⟩ występuje poza treścią tej procedury.

⟨I⟩ REPEATED

a. Nazwa ⟨I⟩ opisana powtórnie na tym samym poziomie oznaczeń.
b. Nazwa ⟨I⟩ powtórzona w wykazie parametrów formalnych.
c. Nazwa ⟨I⟩ powtórzona w zbiorze wartości.
d. Parametr formalny ⟨I⟩ powtórnie specyfikowany.

PROCEDURE LEVEL OVERFLOW

Stopień procedury większy od 3 (p. 1.0).

PROCEDURE OVERFLOW .

Za długi segment (dłuższy od ok. 4200 słów maszynowych); jeśli jest to jedyny sygnał wydrukowany w czasie tłumaczenia, to

procedura jest poprawna i można jej użyć w programie w zwykły sposób (nie jako segment), jeśli tylko przekład tego programu mieści się w pamięci.

SPACE OVERFLOW

Brak miejsca w pamięci operacyjnej maszyny.

STRING LIST OVERFLOW

Zbyt wiele łańcuchów (kolejny czytany łańcuch jest różny od wszystkich dotąd przeczytanych łańcuchów zajmujących łącznie więcej, niż 1024 komórek pamięci).

⟨I⟩ UNDECLARED

Użyta nazwa ⟨I⟩ nie jest w programie opisana, ani nie jest nazwą standardową.

⟨S⟩ UNEXPECTED

a. Symbol ⟨S⟩ użyty w niedopuszczalnym kontekście.
b. Nazwa ⟨S⟩ wymieniona w zbiorze wartości lub w zbiorze specyfikacji nie występuje w wykazie parametrów formalnych.
c. Parametr formalny ⟨S⟩ o specyfikacji label, switch, string, procedure, real procedure, integer procedure, Boolean procedure umieszczony w zbiorze wartości.

VALIST OVERFLOW

Przetłumaczenie programu wymaga wprowadzenia zbyt wielu zmiennych dodatkowych.

11.3. Sygnały zatrzymania translatorów i programów

Po wykonaniu zleconej pracy program lub translator przechodzi do stanu STOP (p. 10.4). Przejście do stanu STOP następuje również po wykryciu błędu w czasie działania programu. We wszystkich przypadkach zatrzymania jest drukowany tzw. sygnał zatrzymania zawierający informację o tym, dlaczego nastąpiło przejście do stanu STOP.

W stanie STOP są wykonalne zlecenia standardowe "go" i "skip". W ALGOLu 1204 w wielu przypadkach wykonanie tych zleceń powoduje uruchomienie programu lub translatora i natychmiastowe zatrzymanie z powtórzeniem sygnału zatrzymania. W podanym dalej wykazie przyjmujemy następującą umowę:

a. jeżeli zlecenie "go" lub "skip" ma po wydrukowaniu sygnału zatrzymania jakąś interpretację różną od wyżej wymienionej,

to omawiamy tę interpretację po omówieniu sygnału zatrzymania,
b. w przeciwnym razie interpretacji zleceń "go" lub "skip" nie omawiamy.

W podanym niżej wykazie sygnałów zatrzymania nie omawiamy sygnałów drukowanych po zakończeniu tłumaczenia poprawnego programu lub opisu procedury ze średnikiem; sygnały te omówiono w p. 11.1.

Oto zapowiedziany wykaz sygnałów:

ADDRESS OVERFLOW

a. Sytuacja nie określona, np. skok do wnętrza instrukcji "dla".
b. Błędne działanie programu wskutek niewłaściwie użytych procedur specjalnych.

Zlecenia "go" i "skip" nie mają określonej interpretacji. Z punktu widzenia programowania w kodzie wewnętrznym maszyny omawiany sygnał oznacza tzw. "przekroczenie adresu".

ARC

Błąd przy obliczaniu wartości funkcji arcsin lub arccos. Zlecenie "skip" powoduje nadanie funkcji wartości .0 i kontynuowanie obliczeń.

BOUND PAIR

a. W wykazie par granicznych granica dolna jest większa od granicy górnej.
b. W opisie tablic z mianem own wykaz par granicznych zawiera więcej niż 10 par granicznych.

DRUM PARAMETER

Nie można wykonać instrukcji procedury todrum lub fromdrum (p. 3.21).

DRUM SEGMENT

W czasie odczytywania segmentu z bębna wykryto błąd w zapisie tego segmentu.

END

a. Program wykonano do końca.
b. Wykonano zlecenie "pbt", "cbt", "rbt", "outp", "comp" lub "ditr".

EXP

Parametr aktualny funkcji exp ma wartość zbyt dużą. Zlecenie "skip" powoduje nadanie funkcji wartości .0 i kontynuowanie ob-

liczeń.

EXPONENTIATION

Wynik potęgowania nie jest określony (w sensie [1], rozdz.9).

FORMAT

Błędne S w wykonywanej instrukcji format(S). Zlecenie "go" powoduje wykonanie na monitorze instrukcji print(S) i ponowne zatrzymanie, a zlecenie "skip" powoduje wykonanie instrukcji

format('?-1.123␣456␣789␣+123')

i kontynuację obliczeń.

IO PARAMETER

Błędna wartość parametru aktualnego procedury wejścia lub wyjścia, np. setinput(-1) lub outchar(150).

LN

Parametr aktualny funkcji ln nie jest dodatni. Zlecenie "skip" powoduje nadanie funkcji wartości .0 i kontynuowanie obliczeń.

LOAD ALGOL

Żądanie ponownego wprowadzenia translatora.

NUMBER

Błąd w czytanej liczbie. Zlecenie "skip" powoduje rozpoczęcie procesu czytania liczby od początku.

PARAMETER LIST

a. Liczba parametrów aktualnych nie jest równa liczbie parametrów formalnych.
b. Typ lub rodzaj parametru aktualnego nie jest zgodny ze specyfikacją odpowiedniego parametru formalnego.

RI CONVERSION

Niewykonalna zmiana z typu real na typ integer.

REAL OVERFLOW

Za duży wynik typu real albo dzielenie przez zero (dzielenie całkowite przez zero nie jest sygnalizowane). Skutki zlecenia "go" nie są określone. Zlecenie "skip" powoduje kontynuację obliczeń z nie określonym wynikiem działania.

SPACE OVERFLOW

Brak miejsca w pamięci (zbyt wielka tablica albo zbyt wiele

jednocześnie czynnych procedur).

SORRY

a. Zleconej pracy translator nie może wykonać.
b. Błąd na taśmie w czasie wykonywania zlecenia "cbt", "rbt" lub "comp".

SQRT

Parametr aktualny funkcji sqrt jest ujemny. Zlecenie "skip" powoduje nadanie funkcji wartości .0 i kontynuowanie obliczeń.

STOP

a. Wykonano instrukcję stop.
b. W stanie RUNNING wykonano zlecenie "stop" lub naciśnięto klawisz PRZERWANIE PROGRAMU.
c. Zakończono czytanie wykazu poprawek.

Zlecenie "go" powoduje ponowne uruchomienie od miejsca zatrzymania tak, jak gdyby zatrzymania nie było. Zlecenie "skip" nie ma określonej interpretacji.

STRING

Czytany łańcuch nie mieści się w tablicy.

SUBSCRIPT

a. Liczba wskaźników przy zmiennej nie jest równa liczbie par granicznych w opisie odpowiedniej tablicy.
b. Wartość wskaźnika nie należy do przedziału określonego odpowiednią parą graniczną.
c. Skok przy nie określonym przełączeniu.

TRANSLATE PROGRAM

Żądanie przetłumaczenia programu.

TRIG

Błąd przy obliczaniu wartości funkcji sin, cos lub tan. Zlecenie "skip" powoduje nadanie funkcji wartości .0 i kontynuowanie obliczeń.

WRONG INSTRUCTION

Interpretacja taka, jak sygnału ADDRESS OVERFLOW. Z punktu widzenia programowania w kodzie wewnętrznym maszyny sygnał ten oznacza tzw. "nielegalny rozkaz" w programie.

?

Odmowa przyjęcia zlecenia.

11.4. Standardowe znaczenie klawiszy nr 0,1,2,3 i 4 w ALGOLu 1204

Podane dalej informacje odnoszą się do przypadku, kiedy wymieniony klawisz jest wciśnięty w czasie działania translatora lub programu.

Klawisz 0 : Wyjściem aktualnym translatora lub programu jest perforator (p. 3.10, 8.1 i 8.8).

Klawisz 1 : Kopiowanie (z ewentualnym korygowaniem) taśmy tłumaczonego programu lub opisu procedury (p. 8.7).

Klawisz 2 : Drukowanie wykazu punktów orientacyjnych tłumaczonego programu lub procedury (p. 8.8).

Klawisz 3 : Uwzględnienie śladu etykiet (p. 8.6).

Klawisz 4 : Uwzględnienie śladu procedur niestandardowych (p. 8.6).

11.5. Zlecenia standardowe systemów MASON i MASON (D)

Podany dalej wykaz zawiera wszystkie zlecenia standardowe dla systemów MASON i MASON (D). Na początku wykazu podano zlecenia najczęściej stosowane przy eksploatacji translatorów ALGOLu 1204. Zlecenia wymienione na końcu są przydatne przede wszystkim w programowaniu w kodzie maszynowym, ale mogą również służyć do specjalnego sterowania pracą programów w ALGOLu 1204; odpowiednie informacje podano w p. 11.6.

W celu skrócenia opisów zleceń będziemy stosować następujące oznaczenia:

a - zawartość akumulatora,
w - zawartość wydłużenia akumulatora,
l - zawartość licznika rozkazów, połączona z zawartością rejestrów U, Z, Nd w taki sposób, jak w śladach podprogramów [5],
A,B - co najwyżej 5-cyfrowe adresy ósemkowe nie większe od 34737,
k[A] - zawartość komórki o adresie A,
t - stan zegara systemu w sekundach,
GG,MM,SS - co najwyżej 2-cyfrowe liczby dziesiętne nie większe od 60 wyrażające aktualny lub żądany stan zegara systemu w godzinach, minutach i sekundach,
P - co najwyżej 8-cyfrowy ciąg cyfr ósemkowych oznaczający 24-bitowe słowo maszynowe,

M - cyfra ósemkowa oznaczająca część P i B słowa maszynowego (p. 6.1),

R - co najwyżej 3-cyfrowa liczba ósemkowa oznaczająca część OR słowa maszynowego (p. 6.1),

N - co najwyżej 5-cyfrowa liczba ósemkowa, nie większa od 37777, oznaczająca część AR słowa maszynowego (p. 6.1),

NP - nazwa programu w bibliotece bębnowej (p. 10.7), tj. co najwyżej 4-znakowy ciąg małych liter lub cyfr zaczynający się od litery,

AB - co najwyżej 6-cyfrowa liczba całkowita oznaczająca dziesiętny adres bębnowy (początku jakiegoś obszaru bębnowego).

W wykazie stosujemy następujący układ informacji:

⟨postać zlecenia⟩. ⟨krótki opis słowny zlecenia⟩.
1. ⟨stany maszyny, w których zlecenie jest wykonalne⟩.
2. ⟨interpretacja zlecenia - słownie lub symbolicznie⟩.
3. ⟨sygnały drukowane po zakończeniu wykonywania zlecenia i ich interpretacja⟩.
(ew. uwagi dodatkowe).

Symbol (D) występujący niekiedy po kropce w pierwszym wierszu informacji oznacza, że zlecenie jest standardowe tylko w systemie MASON (D). Pozostałe zlecenia są standardowe w obu systemach obsługi.

Zwrot "zarezerwowana pamięć operacyjna" oznacza obszary pamięci określone wartościami k[21] i k[22], czyli zawartościami komórek o adresach dziesiętnych 17 i 18 (p. 7.1 i 11.6).

Dalej podajemy zapowiedziany wykaz zleceń.

load. Wprowadzanie taśmy binarnej.
1. STOP.
2. Przejście do stanu LOADING, a następnie

a:=k[16]; w:=k[17]; l:=k[20].

3. LOADED: Zlecenie wykonano.
LOAD ERROR: Operator przerwał wykonywanie zlecenia albo wykryto błąd na taśmie.
(UWAGA: Zlecenia "load" nie stosuje się do wprowadzania taśm binarnych przekładu wyprodukowanych przez translator D na zlecenie "pbt" - p. 11.1, "rbt").

load NP. (D) Szukanie w bibliotece bębnowej programu o nazwie NP i wprowadzenie go do pamięci operacyjnej.

1. STOP.
2. Patrz "load".
3. NP LOADED: Program NP znaleziono i wprowadzono.
   LOAD ERROR: Operator przerwał wykonywanie zlecenia.
   NP'NOT IN LIBRARY: Programu NP nie ma w bibliotece.
   LIBRARY DAMAGED: Biblioteka uszkodzona, programu NP nie znaleziono.

load AB. (D) Wprowadzenie do pamięci operacyjnej programu zapisanego w obszarze roboczym bębna (p. 10.5) poczynając od komórki o adresie AB.

1. STOP.
2. Patrz "load".
3. AB LOADED: Program wprowadzono.
   LOAD ERROR: Operator przerwał wykonywanie zlecenia lub wykryto błąd w zapisie.

dump. Wyprowadzenie zarezerwowanej pamięci operacyjnej na taśmę binarną.

1. STOP, przy czym określona jest wartość 1 (por. "dl").
2. k[16]:=a; k[17]:=w; k[20]:=1; (por. "load"), a następnie przejście do stanu DUMPING.
3. DUMPED: Zlecenie wykonano.
   DUMP ERROR: Operator przerwał wykonywanie zlecenia albo wartości k[21] i k[22] nie określają rezerwacji pamięci.
   (UWAGA: Nie należy wyprowadzać pamięci w przypadku obliczeń programem używającym segmentów bębnowych - p. 1.11 - ponieważ utrwalone na tej taśmie informacje o stanie pamięci bębnowej mogą zdezaktualizować się w wyniku działań operatora i innych użytkowników maszyny).

dump NP. (D) Dołączenie zawartości zarezerwowanej pamięci operacyjnej do biblioteki bębnowej i nadanie zapisowi bibliotecznemu nazwy NP.

1. Patrz "dump".
2. Patrz "dump".
3. NP DUMPED: Zlecenie wykonano.
   DUMP ERROR: Patrz "dump".

NP IN LIBRARY: Zlecenia nie można wykonać, gdyż biblioteka zawiera już program o nazwie NP.
DRUM OVERFLOW: Dostępna pamięć bębnowa jest za mała.
(UWAGA: Wykonanie zlecenia powoduje zmniejszenie obszaru roboczego bębna o liczbę komórek potrzebnych do rozszerzenia biblioteki; por. także "dump" - uwaga końcowa).

dump AB. (D) Zapisanie w obszarze roboczym bębna, poczyhając od komórki o adresie AB, zawartości zarezerwowanej pamięci operacyjnej.

1. Patrz "dump".
2. Patrz "dump".
3. AB DUMPED: Zlecenie wykonano.
   DUMP ERROR: Patrz "dump".
   DRUM OVERFLOW: Zlecenia nie można wykonać, gdyż zapis programu nie mieści się w obszarze o adresie AB (trzeba zmniejszyć AB lub zwiększyć obszar roboczy - por. "del").
   (UWAGA: Patrz "dump" - uwaga końcowa).

test. Porównanie taśmy binarnej z zawartością pamięci operacyjnej.

1. STOP.
2. Przejście do stanu TESTING.
3. TESTED: Zlecenie wykonano.
   TEST ERROR: Operator przerwał wykonywanie zlecenia albo wykryto niezgodność taśmy z zawartością pamięci.

test NP. (D) Porównanie zapisu programu bibliotecznego o nazwie NP z taśmą binarną założoną do czytnika.

1. STOP.
2. Przejście do stanu TESTING.
3. NP TESTED: Zlecenie wykonano.
   TEST ERROR: Operator przerwał wykonywanie zlecenia albo wykryto niezgodność taśmy z zapisem bębnowym.
   NP NOT IN LIBRARY: Programu NP nie ma w bibliotece.
   LIBRARY DAMAGED: Biblioteka uszkodzona, programu NP nie znaleziono.

take. (D) Dołączenie do biblioteki bębnowej programu specjalnego (p. 10.7) z taśmy założonej do czytnika i nadanie zapisowi bibliotecznemu nazwy podanej na taśmie.

1. STOP.

2. Przejście do stanu TAKING.
3. NP TAKEN: Zlecenie wykonano.
   TAKE ERROR: Operator przerwał wykonywanie zlecenia albo wykryto błąd na czytanej taśmie.
   NP IN LIBRARY: Patrz "dump NP".
   DRUM OVERFLOW: Patrz "dump NP".
   (UWAGA: Programem specjalnym jest np. translator D ALGOLu 1204, którego pierwsze wydanie ma nazwę biblioteczną ta1; programów specjalnych nie można pisać w ALGOLu 1204).

take NP. (D) Sprawdzenie identyczności nazwy NP z nazwą podaną na taśmie i wykonanie zlecenia "take".
1,2,3. Patrz "take".

drum. (D) Drukowanie aktualnej wielkości obszaru roboczego bębna (w słowach maszynowych).
1. Wykonalne w dowolnym stanie.
2. Drukowanie na monitorze sygnału postaci D=⟨liczba⟩.
3. Nowa linia.

list. (D) Drukowanie wykazu nazw programów umieszczonych w bibliotece bębnowej, w porządku dołączania ich do biblioteki.
1. Patrz "dump".
2. Drukowanie wykazu postaci

   NPi Wi (i=1,2,...,L)

   gdzie L jest liczbą programów w bibliotece, NPi jest nazwą i-tego programu, a Wi jest wielkością obszaru roboczego w momencie bezpośrednio poprzedzającym dołączenie NPi do biblioteki (nazwy programów specjalnych są poprzedzone znakiem mnożenia).
3. Sygnał jak po wykonaniu zlecenia "drum"; jeśli wartość 1 nie jest określona, to sygnał jak po "dl".
   (UWAGA: Wykonywanie zlecenia można przerwać naciskając ZO monitora lub klawisz PRZERWANIE PROGRAMU).

del NP. (D) Powiększenie obszaru roboczego bębna (zmniejszenie biblioteki).
1. Patrz "dump".
2. Usunięcie z biblioteki programu o nazwie NP oraz wszystkich programów następnych (tzn. dołączonych do biblioteki później)

i odpowiednie (por. "list") zwiększenie obszaru roboczego bębna.
3. NP NOT IN LIBRARY: Programu NP nie ma w bibliotece.
LIBRARY DAMAGED: Biblioteka uszkodzona, programu NP nie znaleziono.
(Inne sygnały - jak po "list").

go. Uruchomienie zatrzymanego programu od miejsca zatrzymania.
1. STOP.
2. Przejście do stanu RUNNING.
3. Sygnały zatrzymania (p. 11.3).

skip. Uruchomienie zatrzymanego programu z pominięciem jednego rozkazu.
1. Patrz "dump".
2. l:=l+1; przejście do stanu RUNNING.
3. Sygnały zatrzymania (p. 11.3).

stop. Zatrzymanie pracy maszyny.
1. Wykonalne w dowolnym stanie różnym od STOP.
2. Przejście do stanu STOP.
3. STOP: Wykonano w stanie RUNNING.
LOAD ERROR: Wykonano w stanie LOADING.
DUMP ERROR: Wykonano w stanie DUMPING.
TEST ERROR: Wykonano w stanie TESTING.
TAKE ERROR: Wykonano w stanie TAKING.
(UWAGA: Naciśnięcie klawisza PRZERWANIE PROGRAMU działa tak, jak zlecenie "stop").

now. Drukowanie nazwy stanu maszyny.
1. Wykonalne w dowolnym stanie.
2. Badanie stanu maszyny.
3. Drukowanie nazwy stanu i ew. sygnału czekania na ZO urządzenia (jeżeli system czeka na przygotowanie urządzenia - p. 10.3).

dt. Drukowanie stanu zegara systemu.
1. Wykonalne w dowolnym stanie.
2. Wyrażenie stanu zegara t w godzinach GG, minutach MM i sekundach SS.
3. GG.MM SS i nowa linia.

pt GG MM SS. Ustawienie zegara systemu.
1. Wykonalne w dowolnym stanie.
2. t:=3600×GG+60×MM+SS.
3. Nowa linia.
   (W systemie D zlecenie "pt" powoduje dodatkowo wykonanie zlecenia "on").

on. (D) Włączenie rejestracji czasu na monitorze.
1. Wykonalne w dowolnym stanie.
2. Ustawienie systemu w taki sposób, że w momencie przejścia ze stanu STOP do stanu RUNNING i przy przejściu odwrotnym na monitorze jest drukowany stan zegara systemu.
3. Nowa linia.

off. (D) Wyłączenie rejestracji czasu na monitorze.
1. Wykonalne w dowolnym stanie.
2. Anulowanie skutków zlecenia "on".
3. Nowa linia.
   (Po uruchomieniu systemu D rejestracja jest wyłączona. Zlecenia "on" i "off" nie mają żadnego wpływu na uaktualnianie stanu zegara, ani uaktualnianie wartości zmiennej standardowej time w ALGOLu 1204).

d A. Drukowanie zawartości komórki.
1. STOP, RUNNING.
2. Drukowanie k[A] w postaci ósemkowej.
3. Nowa linia.

dd A B. Drukowanie zawartości bloku komórek.
1. STOP, RUNNING.
2. Dla I=A,A+1,...,B drukowanie nowej linii, I, k[I] w postaci ósemkowej.
3. Nowa linia.
   (Naciśnięcie ZO monitora przerywa wykonywanie tego zlecenia po wydrukowaniu najbliższej nowej linii).

di A. Drukowanie zawartości komórki w postaci rozkazu.
1. STOP, RUNNING.
2. Drukowanie k[A] w postaci M R N z odstępami.
3. Nowa linia.

ddi A B. Drukowanie zawartości bloku komórek w postaci rozkazów.
1. STOP, RUNNING.
2. Dla I=A,A+1,...,B drukowanie nowej linii, I, k[I] w postaci M R N z odstępami.
3. Nowa linia.
   (Naciśnięcie ZO monitora przerywa wykonywanie tego zlecenia po wydrukowaniu najbliższej nowej linii).

da. Drukowanie zawartości akumulatora.
1. STOP, RUNNING.
2. Drukowanie akumulatora w postaci ósemkowej.
3. Nowa linia.
   (por. "dl").

dw. Drukowanie zawartości wydłużenia akumulatora.
1. STOP, RUNNING.
2. Drukowanie wydłużenia w postaci ósemkowej.
3. Nowa linia.
   (por."dl").

dl. Drukowanie zawartości licznika rozkazów.
1. STOP, RUNNING.
2. Drukowanie 1 w postaci ósemkowej.
3. Nowa linia.
   (Jeżeli zlecenie zostanie rozpoznane w czasie wykonywania przez system jakiejś operacji zleconej przez program, to odpowiedź na to zlecenie ma postać WAIT P, gdzie P jest wartością 1 bezpośrednio po wykonaniu operacji).

p A P. Podstawienie do komórki.
1. STOP, RUNNING.
2. k[A]:=P.
3. Nowa linia.

pi A M R N. Podstawienie rozkazu do komórki.
1. STOP, RUNNING.
2. k[A]:=(M×128+R)×16384+N.
3. Nowa linia.

pa P. Podstawienie do akumulatora.
1. STOP, RUNNING.
2. a:=P.
3. Nowa linia.

pw P. Podstawienie do wydłużenia akumulatora.
1. STOP, RUNNING.
2. w:=P.
3. Nowa linia.

pl P. Podstawienie do licznika rozkazów.
1. STOP, RUNNING.
2. l:=P.
3. Nowa linia.

goto A. Uruchomienie programu od wskazanego rozkazu.
1. STOP.
2. Wykonanie zleceń: "pl A" i "go".
3. Sygnały zatrzymania (p. 11.3).

W ALGOLu 1204 komórki o adresach ósemkowych 21 i 22 (określające rezerwację pamięci operacyjnej) wykorzystano w następujący sposób:

Ostatnio wykonywana czynność	Zawartość zarezerwowanej pamięci
Wprowadzenie translatora A	Translator A
Wprowadzenie translatora B	Translator B
Wprowadzenie podprogramów PP	Podprogramy PP
Wprowadzenie translatora D	Nie określona
Przetłumaczenie programu za pomocą translatora A lub D	Przekład programu wraz z podprogramami PP - czyli program gotowy do wykonania
Wprowadzenie podprogramów PP i taśmy przekładu wyprodukowanej przez translator B	Jak wyżej
Rezerwacja komórek roboczych w czasie działania programu w ALGOLu 1204	Przekład programu, podprogramy PP i zarezerwowane komórki robocze

Z informacji podanych w tabeli i interpretacji zleceń "load", "dump" i "test" wynika już sposób wykonania każdej z wymienionych niżej czynności:

c1. produkowanie kompletnej (tzn. wraz z podprogramami) taśmy binarnej programu,

c2. produkowanie i sprawdzanie kopii taśm translatorów A i B oraz podprogramów PP i sprawdzanie poprawności taśm omawianych w c1,

c3. przerwanie obliczeń na czas dłuższy i ich późniejszą kontynuację (niekoniecznie na tym samym egzemplarzu maszyny).

W pewnych przypadkach bardzo istotna jest informacja o tym, czy wykonanie zlecenia powoduje zmianę zawartości pamięci operacyjnej (przykładem moze być konieczność zwiększenia obszaru roboczego bębna w momencie, kiedy w pamięci operacyjnej znajduje się program i ewentualnie nie wydrukowane jeszcze wyniki). Dla uniknięcia nieporozumień wyjaśniamy, że

1° wykonanie zlecenia "dump" (z parametrem lub bez parametru) zmienia tylko zawartość komórek 16, 17 i 20 (ósemkowo), gdzie zapamiętuje się wartości a,w,l,

2° wykonanie zleceń

"test" (bez parametru)
"drum", "list", "del",
"stop", "now",
"dt", "pt", "on", "off",
"d", "dd", "di", "ddi", "da", "dw", "dl"

nie zmienia zawartości pamięci operacyjnej,

3° jeżeli NP nie jest programem specjalnym (np. translatorem D), to po wykonaniu zlecenia "test NP" zawartość pamięci operacyjnej jest taka, jak po wykonaniu zlecenia "load NP".

UWAGA

Jeżeli operator miał zamiar napisać np. zlecenie "dump", a wskutek omyłki napisał np. "fump" (zlecenie niestandardowe), to spowoduje to uruchomienie programu. W ALGOLu 1204 natychmiastową reakcją będzie odmowa przyjęcia zlecenia "fump", tzn. sygnał składający się ze znaku zapytania i zmiany wiersza (na zestawie M - także dwa mało różniące się stany zegara). Skutkiem omyłki będzie wynikająca z uruchomienia translatora zmiana wartości a,w,l.

Jeśli omyłka opisanego rodzaju zostanie zauważona przed napisaniem symbolu końca zlecenia, to można to zlecenie skasować pisząc np. literę E (p. 10.2); w tym przypadku nie zmieni się zawartość żadnego rejestru.

11.6. Początkowy obszar pamięci operacyjnej w ALGOLu 1204

Na końcu poprzedniego punktu opisano zlecenia umożliwiające wykonywanie operacji na komórkach zajętych przez program lub translator (drukowanie i zmiana zawartości). Niektóre takie operacje mogą być użyteczne przy uruchamianiu i sprawdzaniu programów w ALGOLu 1204. Aby umożliwić wykorzystanie najważniejszych z tych możliwości, podajemy adresy ósemkowe komórek początkowego obszaru pamięci wraz z informacjami o możliwych manipulacjach na tych komórkach.

Komórki 21 i 22 zawierają informacje (które system wydrukuje na zlecenie "dd 21 22") o aktualnie zarezerwowanej pamięci operacyjnej. Pierwszy zarezerwowany obszar pamięci zawiera komórki o adresach 1,2,3,...,k[21]. Jeżeli k[22]=0, to drugi obszar jest pusty; w przeciwnym razie drugi obszar zawiera komórki o adresach k[22],k[22]+1,...,34737 (dziesiętnie: 14815). Zawartości tych komórek decydują o sposobie wykonania zlecenia "dump" (z parametrem lub bez parametru).

Pozostałe komórki początkowego obszaru pamięci opisujemy według następującego schematu:

Adres ósemkowy komórki.
1. Zawartość.
2. Przykład manipulacji operatora.
3. Równoważna operacja programu w ALGOLu 1204.
   (UWAGI).

Komórka 23.
1. Wartość zmiennej standardowej time.
2. p 23 0.
3. time:=0 albo zerk(time).

Komórka 31 (w czasie tłumaczenia programu zawartości zmieniać nie wolno).
1. Numer wejścia aktualnego.
2. p 31 1.
3. setinput(1).
   (Do komórki 31 można podstawić tylko 0, 1 lub 2).

Komórka 32 (w czasie tłumaczenia programu zawartości zmieniać nie wolno).

1. Numer wyjścia aktualnego.
2. p 32 3.
3. setoutput(3).
   (Do komórki 32 można podstawiać tylko 0, 1, 2 lub 3).

Komórka 33, 34 i słowo długie w komórkach 35-36.

1. Wartości zmiennych standardowych lastchar, lastinteger i lastreal.
2. dd 33 36 - wynik w postaci ósemkowej.
3. print(lastchar,lastinteger,lastreal).

Komórka 41.

1. Długość bufora czytnika (por. wyjaśnienie na końcu).
2. p 41 1.
3. begin integer a; a:=33; usak(1); pska(+a) end
   (Do komórki 41 wolno podstawiać tylko wartości 1,2,3,4,5,6, 7 i 10 - ósemkowo).

Komórka 44 (tylko translator D).

1. Długość bufora czytnika 2.
2. p 44 10.
3. begin integer a; a:=36; usak(8); pska(+a) end
   (Por. komórka 41).

Komórka 45 (tylko translator D).

1. Wartość zmiennej standardowej drumplace.
2. p 45 0.
3. drumplace:=0 lub zerk(drumplace).

Komórka 46 (tylko translator D).

1. Liczba ograniczająca dopuszczalne wartości zmiennej drumplace.
2. d 46 - wynik w postaci ósemkowej.
3. begin integer a; usak(+38); pska(a); print(a) end

Podane informacje są ważne dla wszystkich translatorów ALGOLu 1204, z wyjątkiem informacji o komórkach 44, 45 i 46, ważnych tylko dla translatora D.

"Długość bufora czytnika" (por. komórki 41 i 44) jest liczbą znaków czytanych jako jeden blok. Podprogramy czytania znaków współpracują z systemem obsługi w ten sposób, że jeden bufor czytnika na ogół znajduje się w stadium "napełniania znakami",

a jednocześnie z drugiego są czerpane znaki, co daje na ogół jednoczesną pracę czytnika i programu lub translatora.

Po wprowadzeniu translatora długość bufora czytnika jest równa 8. Zmiana długości na 1 może być przydatna, jeżeli czytana taśma zawiera przypadkowe znaki o parzystej liczbie dziurek; zignorowanie błędu czytnika (tzn. naciśnięcie ZO po sygnale READER ERROR) powoduje pominięcie błędnego znaku. Jeśli bufor jest dłuższy od 1, to w opisanym przypadku będzie pominięty cały blok zawierający błędny znak.

Zmiana długości bufora może być również potrzebna, jeśli na końcu czytanej taśmy znajduje się mniej niż 16 odstępów. Trzeba jednak pamiętać, że im krótszy bufor, tym wolniej jest czytana taśma.

ROZDZIAŁ 12. Wersje specjalne ALGOLu 1204

12.0. Uwagi ogólne

W tym rozdziale omawiamy krótko translatory A-DISP, B-DISP, podprogramy pomocnicze PP-DISP, system obsługi MASON (DISP) oraz translator D-COAN. Wersję DISP ALGOLu 1204 wykonano na zlecenie Instytutu Podstawowych Problemów Techniki PAN w Warszawie, a translator D-COAN - na zlecenie Centrum Obliczeniowego PAN w Warszawie.

Omawiane wersje specjalne znajdują się w dyspozycji wpomnianych instytucji i nie są rozprowadzane przez producenta maszyny ODRA 1204 (WZE ELWRO).

12.1. Wersja DISP ALGOLu 1204

W Instytucie Podstawowych Problemów Techniki PAN skonstruowano bardzo użyteczne urządzenie zewnętrzne maszyny, które nazwiemy tutaj wyjściem kineskopowym. Po podłączeniu tego urządzenia do zestawu minimalnego (urządzenie nr 7 w kanale 1) można z niego korzystać w programach napisanych w ALGOLu 1204 stosując wersję DISP systemu obsługi, translatorów A i B i podprogramów PP; wersje te są rozszerzeniem odpowiednich programów dla zestawu M (p. 9.0). System MASON (DISP) może być również używany do nadzorowania pracy zwykłych translatorów A i B, ponieważ zajmuje on tyle samo (1536) komórek pamięci operacyjnej, co systemy MASON i MASON (D).

Na ekranie kineskopu maszyna może wyświetlić punkt - reprezentowany przez plamkę świetlną. W ciągu jednej sekundy plamka ta może zmienić swoje położenie ok. 6000 razy w sposób określony w programie. Położenie punktu określa się za pomocą pary współrzędnych całkowitych z przedziału ⟨0, 511⟩; jako osie współrzędnych przyjmuje się odpowiednio dolny i lewy brzeg ekranu. W pamięci maszyny punkt o współrzędnych X,Y jest reprezentowany przez liczbę całkowitą równą (4096×Y+X)×4, czyli 16384×Y+4×X, gdzie wartości X,Y muszą być liczbami całkowitymi z przedziału ⟨0, 511⟩.

Do wyświetlania N-elementowego zbioru punktów, z których

pierwszy jest reprezentowany przez zmienną całkowitą ze wskaźnikami P[i,j,...], a następne punkty - przez zmienne leksykograficznie następne, służy instrukcja postaci

```
display(N,P[i,j,...])
```

gdzie N jest wyrażeniem arytmetycznym o wartości dodatniej (po ewentualnym zaokrągleniu); jeśli tak nie jest, albo liczba elementów tablicy P, od elementu P[i,j,...] do elementu ostatniego, jest mniejsza od N, to program zatrzymuje się (sygnał SUBSCRIPT).

Szybkość wyświetlania jest równa maksymalnej szybkości urządzenie. Wykonanie instrukcji wyświetlania trwa tak długo, jak długo trwa praca urządzenia (system obsługi czeka na zakończenie wyświetlania). Po zakończeniu wyświetlania można natychmiast ten sam zbiór punktów wyświetlić ponownie.

W praktyce często jest potrzebne wyświetlanie wielokrotne, na przykład w celu sfotografowania wykresu, albo w celu wizualnej oceny rozwiązania (np. równania różniczkowego zwyczajnego). Nie ma specjalnej procedury wyświetlania wielokrotnego, można to bowiem łatwo zaprogramować korzystając ze zwykłych możliwości ALGOLu 1204. Ilustrują to dwa przykłady, w których zakładamy, że potrzebny zbiór punktów jest reprezentowany przez zmienne całkowite P[1],P[2],...,P[N].

PRZYKŁAD 1.

Wyświetlanie zbioru w ciągu T sekund.

```
  time:=0;
E:display(N,P[1]);
  if time<T then go to E
```

PRZYKŁAD 2

Wyświetlanie zbioru do chwili zmiany stanu klawisza nr K.

```
  B:=key(K);
E:display(N,P[1]);
  if key(K)≡B then go to E
```

Jeżeli trzeba dodatkowo obliczyć, ile razy była wykonana instrukcja wyświetlania, to można zadanie wykonać tak:

```
  B:=key(K);
  for I:=0,I+1 while key(K)≡B do display(N,P[1])
```

Procedura display jest - w sensie ALGOLu 1204 - procedurą specjalną (p. 3.0). Nazwa "display" jest standardowa tylko w translatorach A-DISP i B-DISP.

System MASON (DISP) z punktu widzenia operatora różni się od zwykłego systemu MASON tylko dodatkowymi sygnałami (DISPLAY ERROR, DISPLAY WAIT) związanymi z dodatkowym urządzeniem oraz zbiorem zleceń standardowych; różnica w zbiorach zleceń polega na tym, że wersja DISP systemu wykonuje dodatkowo zlecenia "on" i "off" o interpretacji takiej, jak w systemie MASON (D) - p. 11.5. Skutkiem ubocznym uruchomienia systemu jest wykonanie zlecenia "off", a skutkiem ubocznym ustawienia zegara systemu (pt G M S) jest wykonanie zlecenia "on".

12.2. Translator D-COAN

Translator D-COAN pracuje pod kontrolą systemu MASON (D) i jest wersją translatora D dostosowaną do nietypowej drukarki wierszowej pracującej w zestawie D zainstalowanym w Centrum Obliczeniowym PAN w Warszawie.

Jeżeli zestaw D nie zawiera drukarki wierszowej, to wszystkie skutki używania translatora D-COAN na tym zestawie są identyczne ze skutkami używania translatora D.

Prace cytowane

[1] Stefan Paszkowski, Język ALGOL 60. Warszawa (wyd. 1 - 1965, wyd. 2 - 1968, wyd. 3 - 1970, wyd. 4 - 1971 Państwowe Wydawnictwo Naukowe.

[2] Jan Madey, Niejednoznaczności ALGOLu 60 i metody ich usunięcia w GIER-ALGOLu 4. Warszawa 1971. Wydawnictwa Uniwersytetu Warszawskiego.

[3] Krystyna Jerzykiewicz, Jerzy Szczepkowicz, ALGOL 1204. Wrocław 1970. WZE-ELWRO.

[4] Krystyna Jerzykiewicz, Jerzy Szczepkowicz, ALGOL 1204 (D). Wrocław 1971. WZE-ELWRO.

[5] ODRA 1204 - Opis funkcjonalny. Wrocław 1968. WZE-ELWRO.

Skorowidz nazw, sygnałów i symboli

Wykaz alfabetyczny nazw standardowych, nagłówków zleceń, nagłówków poprawek, symboli elementów składniowych oraz sygnałów systemów, translatorów i programów zawiera m. in. informacje potrzebne do interpretacji sygnałów związanych z omawianym w książce systemem programowania. Każda pozycja wykazu zawiera kolejno:

1° nazwę lub inny symbol - np. "arctan",
2° znaczenie tej nazwy lub symbolu - np. "nazwa standardowa",
3° numer tej strony książki, na której podano najbardziej istotne informacje o nazwie lub symbolu - np. "39".

Znaczenia symboli elementów składniowych podano w nawiasach. Ponieważ skorowidz jest jedynym miejscem książki, w którym wymieniono wszystkie takie symbole, więc nie podano przy nich numeru strony, z wyjątkiem niektórych symboli omawianych na str. 108.

abs nazwa standardowa 39
absa nazwa standardowa 92
ACPA (actual parameter, parametr aktualny)
ACPALIST (actual parameter list, wykaz parametrów aktualnych)
ADDOP (adding operator, operator typu dodawania)
ADDRESS OVERFLOW sygnał zatrzymania 140
ADEC (array declaration, opis tablic)
and sygnał translatora 109, 14
ARAID (arithmetic array identifier, nazwa tablicy arytmetycznej)
ARALIST (arithmetic array list, wykaz tablic arytmetycznych)
ARASEG (arithmetic array segment, segment tablic arytmetycznych)
ARAST (arithmetic assignment statement, arytmetyczna instrukcja podstawienia) 108
arccos nazwa standardowa 39
arcsin nazwa standardowa 39
arctan nazwa standardowa 39
ARC sygnał zatrzymania 140
AREX (arithmetic expression, wyrażenie arytmetyczne)
ARFAC (arithmetic factor, czynnik arytmetyczny)

ARFUDES (arithmetic function designator, nazwa wartości funkcji arytmetycznej)
ARFUID (arithmetic function identifier, nazwa funkcji arytmetycznej)
ARLEPA (arithmetic left part, arytmetyczna lewa strona) 108
ARPRIM (arithmetic primary, pierwotne wyrażenie arytmetyczne)
ARTER (arithmetic term, składnik arytmetyczny)
ARVA (arithmetic variable, zmienna arytmetyczna)
BAST (basic statement, instrukcja podstawowa)
BLOCK (block, blok)
blockbegin (block begin, begin rozpoczynające blok) 108
BLOCK HEAD (block head, nagłówek bloku)
BOAID (Boolean array identifier, nazwa tablicy logicznej)
BOALIST (Boolean array list, wykaz tablic logicznych)
BOASEG (Boolean array segment, segment tablic logicznych)
BOAST (Boolean assignment statement, logiczna instrukcja podstawienia) 108
BOEX (Boolean expression, wyrażenie logiczne)
BOFAC (Boolean factor, czynnik logiczny) 108
BOFUDES (Boolean function designator, nazwa wartości funkcji logicznej)
BOFUID (Boolean function identifier, nazwa funkcji logicznej)
BOLEPA (Boolean left part, logiczna lewa strona) 108
BOPRIM (Boolean primary, pierwotne wyrażenie logiczne)
BOSUM (Boolean sum, suma logiczna) 108
BOTER (Boolean term, składnik logiczny) 108
BOUND PAIR (bound pair, para graniczna)
BOUND PAIR sygnał zatrzymania 140
BOUND PAIR LIST (bound pair list, wykaz par granicznych)
BOVA (Boolean variable, zmienna logiczna)
cbt zlecenie dla translatora 136
comp zlecenie dla translatora 136
COMST (compound statement, instrukcja złożona)
CONST (conditional statement, instrukcja warunkowa)
copy nazwa standardowa 54
cos nazwa standardowa 39
ct zlecenie dla translatora 135
ctp zlecenie dla translatora 135
czkl nazwa standardowa 91

d początek nagłówka poprawki 115
d zlecenie dla systemu 149
D= sygnał systemu 147
da zlecenie dla systemu 150
dd zlecenie dla systemu 149
ddąk nazwa standardowa 92
ddi zlecenie dla systemu 150
DE sygnał systemu 126
DEC (declaration, opis)
deca nazwa standardowa 92
DECLIST OVERFLOW opis błędu 138
DEEX (designational expression, wyrażenie desygnujące)
del zlecenie dla systemu 147
di zlecenie dla systemu 149
ditr zlecenie dla translatora 137
div sygnał translatora 109, 14
dl zlecenie dla systemu 150
dlak nazwa standardowa 91
dokj nazwa standardowa 91
drum zlecenie dla systemu 147
DRUM LIMIT: sygnał systemu 129
DRUM OVERFLOW sygnał systemu 146
DRUM PARAMETER sygnał zatrzymania 140
drumplace nazwa standardowa 53
DRUM SEGMENT sygnał zatrzymania 140
dsak nazwa standardowa 92
dt zlecenie dla systemu 148
dump zlecenie dla systemu 145
DUMPED sygnał systemu 145
DUMP ERROR sygnał systemu 145
DUMPING nazwa stanu 127
DUMST (dummy statement, instrukcja pusta)
dw zlecenie dla systemu 150
dzak nazwa standardowa 92
end zlecenie dla translatora 137
END sygnał zatrzymania 140
enta nazwa standardowa 92
entier nazwa standardowa 39
eqv sygnał translatora 109, 14

ERROR sygnał systemu 129
exch nazwa standardowa 54
exp nazwa standardowa 39
EXP sygnał zatrzymania 140
EXPONENTIATION sygnał zatrzymania 141
format nazwa standardowa 47
FORMAT sygnał zatrzymania 141
FORCL (for clause, warunek "dla")
FORLIST (for list, wykaz "dla")
FORST (for statement, instrukcja "dla")
fromdrum nazwa standardowa 55
ge sygnał translatora 109, 14
go zlecenie dla systemu 148
goto zlecenie dla systemu 151
GOTOST (go to statement, instrukcja skoku)
gt sygnał translatora 109, 14
i sygnał programu 40
i początek nagłówka poprawki 115
IDLIST OVERFLOW opis błędu 138
IFCL (if clause, warunek "jeśli")
IFST (if statement, instrukcja "jeśli")
imp sygnał translatora 109, 14
IMP (implication, implikacja)
IN BOUND PAIR LIST opis błędu 138
inchar nazwa standardowa 42
ininteger nazwa standardowa 42
IN LIBRARY sygnał systemu 146
inreal nazwa standardowa 42
instring nazwa standardowa 43
IO PARAMETER sygnał zatrzymania 141
key nazwa standardowa 40
LAB (label, etykieta)
lastchar nazwa standardowa 53
lastinteger nazwa standardowa 53
lastreal nazwa standardowa 53
le sygnał translatora 109, 14
LEPA (left part, lewa strona)
LIBRARY DAMAGED sygnał systemu 145
line nazwa standardowa 45

list zlecenie dla systemu 147
ln nazwa standardowa 39
LN sygnał zatrzymania 141
lnd nazwa standardowa 93
lnk nazwa standardowa 93
load zlecenie dla systemu 144
LOAD ALGOL sygnał zatrzymania 141
LOADED sygnał systemu 144
LOAD ERROR sygnał systemu 144
LOADING nazwa stanu 127
LOGVALUE (logical value, wartość logiczna)
lt sygnał translatora 109, 14
mina nazwa standardowa 92
mlak nazwa standardowa 91
mlka nazwa standardowa 91
moa1 nazwa standardowa 94
moa2 nazwa standardowa 94
mok1 nazwa standardowa 94
mok2 nazwa standardowa 94
mont zlecenie dla translatora 137
mow1 nazwa standardowa 94
mow2 nazwa standardowa 94
msak nazwa standardowa 92
MULOP (multiplying operator, operator typu mnożenia)
mzak nazwa standardowa 92
ne sygnał translatora 109, 14
not sygnał translatora 109, 14
NOT IN LIBRARY sygnał systemu 145
now zlecenie dla systemu 148
NUMBER (number, liczba)
NUMBER sygnał zatrzymania 141
NUMBER LIST OVERFLOW opis błędu 138
odak nazwa standardowa 92
odkj nazwa standardowa 91
olak nazwa standardowa 91
off zlecenie dla systemu 149
on zlecenie dla systemu 149
or sygnał translatora 109, 14
osak nazwa standardowa 91

oska nazwa standardowa 91
outchar nazwa standardowa 44
OUT OF PROCEDURE BODY opis błędu 138
OUT OF SCOPE opis błędu 138
outp zlecenie dla translatora 136
outstring nazwa standardowa 45
ozak nazwa standardowa 92
p zlecenie dla systemu 150
pa zlecenie dla systemu 151
pad nazwa standardowa 93
pak nazwa standardowa 93
pakw nazwa standardowa 91
PARAMETER LIST sygnał zatrzymania 141
pc? sygnał systemu 129
pbt zlecenie dla translatora 135
pck nazwa standardowa 93
pgw nazwa standardowa 94
pi zlecenie dla systemu 150
pkza nazwa standardowa 91
pl zlecenie dla systemu 151
plak nazwa standardowa 94
pnk nazwa standardowa 93
print nazwa standardowa 46
PRINTER ERROR sygnał systemu 126
PRINTER WAIT sygnał systemu 126
PROCEDURE LEVEL OVERFLOW opis błędu 138
PROCEDURE OVERFLOW opis błędu 138
PRODEC (procedure declaration, opis procedury)
PROGRAM (program, program)
PROHEAD (procedure heading, nagłówek procedury)
PROID (procedure identifier, nazwa procedury)
PROST (procedure statement, instrukcja procedury)
psak nazwa standardowa 94
pska nazwa standardowa 91
pswa nazwa standardowa 91
pt zlecenie dla systemu 149
PUNCH ERROR sygnał systemu 126
PUNCH 1 ERROR sygnał systemu 126
PUNCH 2 ERROR sygnał systemu 126

PUNCH WAIT sygnał systemu 126
PUNCH 1 WAIT sygnał systemu 126
PUNCH 2 WAIT sygnał systemu 126
pw zlecenie dla systemu 151
pzka nazwa standardowa 92
r sygnał programu 40
r początek nagłówka poprawki 115
rbt zlecenie dla translatora 136
rc0 zlecenie dla translatora 135
rc1 zlecenie dla translatora 135
rdak nazwa standardowa 92
read nazwa standardowa 41
READER ERROR sygnał systemu 126
READER 1 ERROR sygnał systemu 126
READER 2 ERROR sygnał systemu 126
READER WAIT sygnał systemu 126
READER 1 WAIT sygnał systemu 126
READER 2 WAIT sygnał systemu 126
REAL OVERFLOW sygnał zatrzymania 141
ref nazwa standardowa 40
REL (relation, relacja)
RELOP (relational operator, operator relacji)
REPEATED opis błędu 138
RI CONVERSION sygnał zatrzymania 141
rsak nazwa standardowa 91
rska nazwa standardowa 91
RUNNING nazwa stanu 127
rzak nazwa standardowa 92
s sygnał programu 40
sdak nazwa standardowa 92
setinput nazwa standardowa 41
setoutput nazwa standardowa 44
sign nazwa standardowa 39
SIMAREX (simple arithmetic expression, proste wyrażenie arytmetyczne)
SIMARVA (simple arithmetic variable, prosta zmienna arytmetyczna)
SIMBOEX (simple Boolean expression, proste wyrażenie logiczne)
SIMBOVA (simple Boolean variable, prosta zmienna logiczna)

SIMDEEX (simple designational expression, proste wyrażenie desygnujące)
sin nazwa standardowa 39
skb nazwa standardowa 90
skd nazwa standardowa 90
skip zlecenie dla systemu 148
skn nazwa standardowa 90
sku nazwa standardowa 90
skw nazwa standardowa 90
skz nazwa standardowa 90
snd nazwa standardowa 90
snu nazwa standardowa 90
snz nazwa standardowa 90
SORRY sygnał zatrzymania 142
space nazwa standardowa 45
SPACE OVERFLOW opis błędu 139
SPACE OVERFLOW sygnał zatrzymania 141
sqrt nazwa standardowa 39
SQRT sygnał zatrzymania 142
ssak nazwa standardowa 91
sska nazwa standardowa 91
ST (statement, instrukcja)
STARTER PUNCHING (p) OR CHECKING (c) ? sygnał systemu 129
STLIST (statement list, wykaz instrukcji) 108
stnd nazwa standardowa 92
stop nazwa standardowa 52
stop zlecenie dla systemu 148
STOP nazwa stanu 127
STOP sygnał zatrzymania 142
STRING (string, łańcuch)
STRING sygnał zatrzymania 142
STRING LIST OVERFLOW opis błędu 139
SUARVA (subscripted arithmetic variable, zmienna arytmetyczna ze wskaźnikami)
SUBOVA (subscripted Boolean variable, zmienna logiczna ze wskaźnikami)
SUBSCRIPT sygnał zatrzymania 142
SULIST (subscript list, wykaz wskaźników)
SWIDEC (switch declaration, opis przełącznika)

SWIDES (switch designator, przełączenie)
SWIID (switch identifier, nazwa przełącznika)
SWILIST (switch list, wykaz przełączeń)
szak nazwa standardowa 92
t zlecenie dla translatora 134
take zlecenie dla systemu 147
TAKEN sygnał systemu 147
TAKE ERROR sygnał systemu 147
TAKING nazwa stanu 127
tan nazwa standardowa 39
test zlecenie dla systemu 146
TESTED sygnał systemu 146
TEST ERROR sygnał systemu 146
TESTING nazwa stanu 127
time nazwa standardowa 53
tmon zlecenie dla translatora 137
todrum nazwa standardowa 55
tp zlecenie dla translatora 135
TRANSLATE PROGRAM sygnał zatrzymania 142
TRIG sygnał zatrzymania 142
u początek nagłówka poprawki 115
uazk nazwa standardowa 91
UNDECLARED opis błędu 139
UNEXPECTED opis błędu 139
UNCONST (unconditional statement, instrukcja bezwarunkowa)
usak nazwa standardowa 91
usaw nazwa standardowa 91
uswk nazwa standardowa 91
uzak nazwa standardowa 92
VALIST OVERFLOW opis błędu 139
w zlecenie dla translatora 136
w początek nagłówka poprawki 115
wait nazwa standardowa 52
wait sygnał programu 52
WAIT sygnał systemu 150
WRONG INSTRUCTION sygnał zatrzymania 142
w0 zlecenie dla translatora 136
w1 zlecenie dla translatora 137
w13 zlecenie dla translatora 137

z początek nagłówka poprawki 115
żeaw nazwa standardowa 92
zerk nazwa standardowa 91
zmak nazwa standardowa 91
zmaw nazwa standardowa 91