JAN MADEY

ALGOL 60
GIER ALGOL III

WYDAWNICTWA
UNIWERSYTETU
WARSZAWSKIEGO
1965

U N I W E R S Y T E T W A R S Z A W S K I
Zakład Obliczeń Numerycznych

Jan Madey

»ALGOL 60«
»GIER ALGOL III«

W Y D A W N I C T W A U N I W E R S Y T E T U W A R S Z A W S K I E G O
1 9 6 5

W czasie opracowywania niniejszego skryptu dołączono do translatora GIER ALGOLu cztery nowe procedury standardowe: to buf, from buf, to car, from car, o których pisze Andrzej Salwicki w broszurce pt. „Współpraca z pamięcią pomocniczą maszyny GIER", Wydawnictwa Uniwersytetu Warszawskiego, Warszawa 1965.

Wyd. I, nakł. 1000 egz. Odbito w powielarni własnej w czerwcu 1965 r.
Druk. „Gryf", zam. 931..

PRZEDMOWA

Skrypt niniejszy ma na celu:

- przekazanie czytelnikom podstawowych informacji o języku algorytmicznym ALGOL 60 i pewnych informacji o programowaniu w języku GIER ALGOL III;
- ułatwienie studiowania publikacji, w których język ALGOL 60 i GIER ALGOL III są zdefiniowane. Należą do nich przede wszystkim pozycje /1/ i /2/ wymienione w Bibliografii.

Przy opracowywaniu skryptu wykorzystywane były również pozostałe pozycje Bibliografii.

Rozdziały 1-26 /część I skryptu/ poświęcone są zasadniczo ALGOLowi. Ewentualne różnice w stosowaniu danej konstrukcji ALGOLu w GIER ALGOLu sygnalizowane są jednak w odpowiednich rozdziałach tej części. W drugiej części skryptu /rozdziały 27-32/ omawiamy tylko GIER ALGOL.

Każdy rozdział dzieli się na paragrafy. Paragraf zatytułowany WSTĘPNE INFORMACJE zawiera szkicowy opis wprowadzanego pojęcia. Opis ten uzupełniany jest w następnych paragrafach. Ścisłą definicję składni wprowadzanej konstrukcji języka zapisujemy przy pomocy formuł metajęzykowych, których użycie objaśniamy poniżej. Tekst ilustrowany jest często przykładami. Jeżeli kilka przykładów występuje w jednym wierszu, wówczas oddziela je wyraźny odstęp.

Składnia języków ALGOL 60 i GIER ALGOL III opisana jest w publikacjach /1/ i /2/ przy pomocy formuł metajęzykowych. Interpretację tych formuł najwygodniej można wyjaśnić na przykładach.

< mart > ::= 0|A|124|6

Ciąg znaków ujęty w nawiasy < > oznacza zmienną metajęzykową, której wartościami są ciągi określonych symboli. Znak ::= odczytujemy: "równy z definicji", zaś znak | odczytujemy: "lub". Są to tzw. łączniki metajęzykowe. Każdy symbol w formule, który nie jest zmienną ani łącznikiem, oznacza samego siebie /np. 0 oznacza cyfrę zero, A oznacza literę A itd./.

Zatem podana wyżej formuła określa zbiór możliwych wartości zmiennej < mart > . Mogą nimi być:

0 lub A lub 124 lub 6

Formuła:

< L > ::= a|b|< mart > c

określa zbiór wartości zmiennej < L > /przy pomocy symboli a,b,c oraz uprzednio zdefiniowanej zmiennej <mart> /.Wartościami tej zmiennej mogą być np.:

a lub Ac lub 124c

Ponieważ odstępy, przejścia do nowego wiersza i tym podobne cechy typograficzne mogą być dowolnie stosowane i nie mają określonego znaczenia, zatem wartościami zmiennej <L> mogą być również:

6 c lub 0 c lub 124 c

Formuła:

< war > ::= < mart >|< war > < mart >

podaje rekurencyjną regułę tworzenia wartości zmiennej <war> . Wartością tą może być któraś z wartości zmiennej <mart> . Dodatkowo, jeżeli mamy już daną pewną wartość dopuszczalną na <war> , to inną wartość tej zmiennej możemy otrzymać przez umieszczenie za tamtą jednej z wartości zmiennej mart . Zatem wartościami zmiennej <war> są skończone ciągi następujących po sobie, w dowolnym porząd-

ku znaków 0,A,124,6, /z ewentualnymi odstępami między nimi/, np.: 0 lub 0A lub 0 A12412466 A000 lub AAAA
Formuła:

< WAR > ::= (|[< war >|< WAR > ; |< WAR > < L >

określa rekurencyjną regułę tworzenia wartości zmiennej <WAR>. Może nią być nawias okrągły- (lub nawias kwadratowy- [z następującą po nim wartością zmiennej < war > . Poza tym umieszczając za dopuszczalną wartością zmiennej <WAR> symbol ; lub którąś z wartości zmiennej <L>, otrzymujemy inną wartość zmiennej <WAR>, np.:

(lub [0 lub [AAA lub [A12466;;0
lub (;;;;aa;b;0cAc6c;

W metajęzyku opisującym składnię ALGOLu /i GIER ALGOLu/ dobieramy nazwy zmiennych metajęzykowych tak, aby ich znaczenie potoczne odpowiadało możliwie blisko znaczeniu danego pojęcia w ALGOLu /i GIER ALGOLu/. Dodatkowo w skrypcie tym, jeżeli opisujemy uproszczoną postać, to stosujemy polskie nazwy zmiennych metajęzykowych, np.

< instrukcja skoku > ::= go to < etykieta >

Natomiast gdy podajemy ostateczną definicję składni danej konstrukcji języka, to zmienne metajęzykowe zapisujemy w terminologii angielskiej, umieszczając przed formułą słowniczek, np.:

label - etykieta
identifier - nazwa
unsigned integer - liczba całkowita bez znaku

< label > ::= < identifier >|< unsigned integer >

Formułę zapisaną przy użyciu terminów angielskich, ale różniącą się od formuły podanej w publikacji /1/ lub /2/ opatrujemy znakiem (ϰ) na prawym marginesie, zaś w odpowiednim rozdziale wyjaśniamy przyczyny tej różnicy.

Terminologia polska w większości przypadków oparta jest na zaproponowanej przez Paszkowskiego /3/.

Na zakończenie chciałbym wyrazić serdeczne podziękowanie prof. dr St. Turskiemu, z którego inicjatywy powstał niniejszy skrypt. Składam również gorące podziękowanie dr A. Kiełbasińskiemu i mgr B. Kiełbasińskiej, których uwagi i propozycje znalazły wielokrotne odbicie w tekście, mgr W. Pankiewiczowi, którego energiczna współpraca umożliwiła opracowanie skryptu w stosunkowo krótkim okresie czasu, pani H. Złomaniec za sprawdzenie odpowiedzi do ćwiczeń i korektę maszynopisu, oraz pani J. Stanisławskiej za staranne przepisanie rękopisu.

1. WIADOMOŚCI WSTĘPNE

1 - 1. Programowanie maszyn cyfrowych

Współczesne elektronowe maszyny cyfrowe charakteryzują się, mimo dużej różnorodności typów, podobną strukturą logiczną i zasadą działania. Podstawowymi częściami składowymi maszyny są:

- urządzenie arytmometryczne /arytmometr/,
- urządzenie pamięciowe /pamięć/,
- urządzenie sterowania,
- urządzenie wejścia - wyjścia.

Przed rozpoczęciem obliczeń musi być umieszczony w pamięci maszyny p r o g r a m , czyli ciąg i n s t r u k c j i , z których każda powoduje wykonanie przez maszynę określonej o p e r a c j i . Instrukcje występujące w programie dotyczyć mogą jedynie tych operacji, które maszyna jest w stanie wykonać. Operacje te i odpowiadające im instrukcje nazwiemy e l e m e n t a r n y m i .

Operacje elementarne można podzielić na m a t e m a t y c z n e i a d m i n i s t r a c y j n e . Pierwsze z nich polegają na wykonaniu przez maszynę prostego działania arytmetycznego lub logicznego. Operacje administracyjne sterują kolejnością wykonywanych przez maszynę instrukcji, lub powodują przesyłanie i n f o r m a c j i tzn. instrukcji lub liczb, z pewnych miejsc maszyny do innych. W szczególności operacje te mogą przekazywać jednostki informacji z urządzeń wejścia - wyjścia do innych urządzeń /lub odwrotnie/. Umożliwia to wprowadzenie progra-

mu i danych do maszyny oraz wyprowadzenie wyników na zewnątrz maszyny.

Aby wykonać obliczenia na maszynie trzeba:

1. Zanalizować rozważany problem, wybrać metodę numeryczną rozwiązywania i ułożyć plan obliczeń.
2. Z a p r o g r a m o w a ć rozwiązanie problemu, tzn. przekształcić algorytm metody w skończony ciąg instrukcji elementarnych, z których każda musi być zapisana jako grupa określonych symboli /czyli w k o d z i e maszyny/.
3. Program napisany w kodzie doprowadzić do postaci, która umożliwi przesłanie go, poprzez urządzenie wejścia - wyjścia, do pamięci /np. wyperforowanie taśmy papierowej lub kart dziurkowanych/.

1 - 2. Język algorytmiczny ALGOL

1 - 2.1. Autokody

Wykonanie czynności 2. opisanej w poprzednim paragrafie /tzn. napisanie programu w j ę z y k u w e w n ę t r z n y m maszyny/, jest procesem skomplikowanym. Poza tym, program taki użyteczny jest jedynie dla maszyny, w której języku wewnętrznym został zakodowany. W celu uniknięcia tych trudności zostały opracowane a u t o k o d y /j ę z y k i z e w n ę t r z n e maszyn cyfrowych/. W autokodzie procesy obliczeniowe zapisywane są zazwyczaj przy pomocy instrukcji, mających postać zbliżoną do przyjętej w matematyce. Program napisany w autokodzie jest bardziej zwięzły i przejrzysty od odpowiadającego mu programu w języku wewnętrznym, ponieważ zmniejszona jest do minimum ilość instrukcji administracyjnych. Aby dany autokod mógł być użyty jako język zewnętrzny, musi być uprzednio opracowany program, który każdą z instrukcji autokodu automatycznie przekształci w odpowiedni ciąg instrukcji elementarnych. Program taki nazywamy t r a n s l a t o r e m lub c o m p i l e r e m .

Od autokodu można żądać dodatkowego warunku - niezależności od konkretnej maszyny. Wówczas algorytm zapisany w autokodzie może być realizowany na każdej maszynie wyposażonej w compiler z tego autokodu na jej język wewnętrzny.

1 - 2.2. Poziomy ALGOLu

W latach 1958-1959 został opracowany międzynarodowy autokod, spełniający postulat niezależności. Nazwano go ALGOL, od początkowych liter angielskich słów ALGO - rithmic Language /język algorytmiczny/.

W ALGOLu rozróżnia się 3 poziomy języka:

1. Język wzorcowy, będący językiem definicji, ze ściśle ustalonym zbiorem symboli, używanych do zapisu algorytmów /ew. tekstów/.
2. Język publikacyjny, który dopuszcza zmiany symboli języka wzorcowego /związane np. z wygodą druku/, przy zachowaniu wzajemnej jednoznaczności między zbiorami tych symboli.
3. Dowolna konkretna reprezentacja, będąca odmianą ALGOLu związaną z konkretną maszyną. Reprezentacja ta musi zawierać reguły jednoznacznego przekładu z języka wzorcowego oraz publikacyjnego. Jednocześnie powinny być dołączone do niej instrukcje umożliwiające korzystanie z urządzeń wejścia - wyjścia i pamięci danej maszyny.

Tego typu odmianą jest GIER ALGOL III, opracowany dla duńskiej maszyny matematycznej GIER.

1 - 2.3. Struktura ALGOLu

Poszczególne poziomy ALGOLu muszą posiadać tę samą strukturę /pomimo ew. różnic między zbiorami symboli opisujących obiekty definiowane w języku/.

ALGOL służy do formułowania procesów obliczeniowych. Naturalne jest więc istnienie takich elementów języka, jak l i c z b y , z m i e n n e , f u n k c j e . Z nich

budowane są zgodnie z pewnymi zasadami wyrażenia arytmetyczne lub logiczne, których wartości mogą być nadawane zmiennym, przy pomocy instrukcji podstawienia. Instrukcje te, wraz z szeregiem innych /niearytmetycznych/, tworzą jednostki operacyjne języka. Mogą być one oznaczone przy pomocy etykiet. W ALGOLu występują również jednostki nieoperycyjne, zwane deklaracjami. Podają one informacje o istnieniu i własnościach obiektów występujących w programie.

1 - 3. Przykład programu w ALGOLu

Zapiszemy w ALGOLu algorytm realizujący następujący problem:
Wyznaczyć wartość e^w ze wzoru

$$e^w = 1 + w/1! + w^2/2! + \ldots + w^k/k! + \ldots$$

Zauważmy, że /k+1/-szy składnik szeregu /k>0/ otrzymamy przez pomnożenie poprzedniego składnika przez ułamek w/k. Sumowanie przerywamy, gdy kolejny składnik jest mniejszy od pewnej liczby oznaczonej przez ε /epsilon/.

```
        begin integer k;
        real skladnik, suma, epsilon, w, e do w;
           k:= 0;
           skladnik:= suma:= 1;
POWT:      k:= k+1;
           skladnik:= skladnik × w/k;
           suma:= suma + skladnik;
           if abs(skladnik) ≥ epsilon then go to POWT;
           e do w:= suma
        end
```

Realizacja programu na maszynie będzie możliwa dopiero po uzupełnieniu go instrukcją wczytującą wartości zmiennych epsilon i w, oraz instrukcją wyprowadzającą zmienną

e do w /tzn. jej wartość, będąca szukaną wartością e^w/.

Jak wiadomo /por. 1 - 2.2/ instrukcje te nie mogą występować w języku wzorcowym ani publikacyjnym, gdyż poziomy te muszą być niezależne od typu maszyny.

2. SYMBOLE PODSTAWOWE W ALGOLu

Zbiór symboli podstawowych języka wzorcowego utworzony jest z czterech podzbiorów:

liter, cyfr, wartości logicznych, ograniczników.

Nie zalecamy dokładnego studiowania następnego paragrafu /2 - 1/ przy pierwszym czytaniu skryptu.

2 - 1. Opis w metajęzyku

basic symbol	- symbol podstawowy
letter	- litera
digit	- cyfra
logical value	- wartość logiczna
delimiter	- ogranicznik

< basic symbol > ::= < letter >|< digit >|< logical value >|
< delimiter >

2 - 1.1. Litery

< letter > ::= 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

2 - 1.2. Cyfry

< digit > ::= 0|1|2|3|4|5|6|7|8|9

2 - 1.3. Wartości logiczne

true	- prawda
false	- fałsz

< logical value > ::= true | false

2 - 1.4. Ograniczniki

operator	- operator
separator	- przerywnik
bracket	- nawias
declarator	- miano
specificator	- specyfikator

< delimiter > ::= < operator >|< separator >|< bracket >|
< declarator >|< specificator >

2 - 1.4.1. Operatory

arithmetic operator	- operator arytmetyczny
relational operator	- operator relacji
logical operator	- operator logiczny
sequential operator	- operator następstwa
go to	- skocz do
if	- jeśli
then	- to
else	- w przeciwnym razie
for	- dla
do	- wykonaj

< operator > ::= < arithmetic operator >|< relational operator >|
< logical operator >|< sequential operator >

< arithmetic operator > ::= +|-|×|/|÷|↑

< relational operator > ::= <|≤|=|≥|>|≠

< logical operator > ::= ≡|⊃|∨|∧|¬

< sequential operator > ::= go to|if|then|else|for|do

2 - 1.4.2. Przerywniki

step - krok
until - aż do
while - podczas gdy
comment - komentarz

< separator > ::= ,|.|₁₀|:|;|:=|␣|step|until|while|comment

2 - 1.4.3. Nawiasy

begin - początek
end - koniec

< bracket > ::= (|)|[|]|‘|’|begin|end

2 - 1.4.4. Miana

own - własne
Boolean - boolowskie
integer - całkowite
real - rzeczywiste
array - tablica
switch - przełącznik
procedure - procedura

< declarator > ::= own|Boolean|integer|real|array|switch|procedure

2 - 1.4.5. Specyfikatory

string - łańcuch
label - etykieta
value - wartość

< specificator > ::= string|label|value

2 - 2. Uwagi

1. Znaczenie wielu symboli podstawowych jest oczywiste. Dokładne wyjaśnienie wszystkich symboli podamy równolegle z wprowadzaniem konstrukcji ALGOLu, w których dane symbole występują.
2. Każde podkreślone słowo jest interpretowane jako pojedynczy symbol podstawowy.
3. Ze względów technicznych, w dalszej części skryptu zamiast symboli języka wzorcowego wymienionych w 2-2.1.p.4, będą występowały odpowiednie symbole GIER ALGOLu.

2 - 2.1. GIER ALGOL

Zbiór symboli podstawowych GIER ALGOLu różni się w sposób następujący od zbioru symboli języka wzorcowego:

1. Występują cztery dodatkowe litery: æ , Æ , ø , Ø.
2. Obok Boolean może występować równoważny mu symbol boolean. Obok go to mogą występować równoważne mu symbole goto oraz go to.
3. Symbole różniące się:

Język wzorcowy	↑	¬	␣	'	`	÷	⊃
GIER ALGOL	^	-,	_	{	}	:	=>

4. Wszystkie symbole podstawowe w GIER ALGOLu należy pisać dokładnie według definicji /poza wyjątkami podanymi w p. 2/.

3. LICZBY

Liczby zapisujemy w układzie dziesiętnym, w sposób zbliżony do powszechnie używanego.

3 - 1. Wstępne informacje

1. Liczby ujemne poprzedza się znakiem minus /-/, liczby dodatnie zapisuje się bez znaku, lub z plusem /+/.
2. Część całkowitą oddzielamy od części ułamkowej kropką /nie przecinkiem/.
3. W przypadku liczb z przedziału otwartego /-1,1/, zero przed kropką dziesiętną może być opuszczone.
4. Poszczególne grupy cyfr mogą być oddzielone odstępami, wolno również dodawać nieznaczące zera.

3 - 1.1. Przykłady

0	+327	-0.845	475 566
-2	.345	03.00	-.128

3 - 2. Mnożnik skalujący /cecha/

Po liczbie zapisanej dziesiętnie może następować symbol podstawowy ₁₀ po którym musi być umieszczana liczba całkowita. Tak utworzony ciąg znaków jest liczbą w ALGOLu, której wartość równa jest wartości części zapisanej dziesiętnie przemnożonej przez odpowiednią potęgę dziesięciu /patrz przykłady/.

3 - 2.1. Przykłady

Zapis liczby w ALGOLu	Zapis konwencjonalny
$2_{10}4$	$2 \cdot 10^4 = 20000$
$8.9100_{10}+2$	$8{,}910 \cdot 10^2 = 891$
$-.820_{10}-3$	$-0{,}820 \cdot 10^{-3} = -0{,}00082$
$1_{10}+3$	$1 \cdot 10^3 = 1000$
$_{10}3$	$1 \cdot 10^3 = 1000$
$+0.3450_{10}+05$	$0{,}345 \cdot 10^5 = 34500$

3 - 3. Typy liczb

Liczby całkowite /definicja, patrz 3-4/ są typu integer. Wszystkie pozostałe liczby są typu real./Podkreślone słowa integer i real są symbolami podstawowymi języka - por. 2-1.4.4./

3 - 3.1. Przykłady

1. Liczby typu integer

0	+1	-327	0034
84122	100	+62781	-222 222

2. Liczby typu real

0.0	+1.34	-.826	-0.002
$3.6_{10}2$	$-1.0_{10}-2$	$.0048_{10}-080$	$_{10}-5$
$+2_{10}4$	$_{10}+5$	$1_{10}1$	$+30_{10}0$

3 - 4. Opis w metajęzyku

unsigned	- bez znaku
integer	- liczba całkowita
decimal fraction	- ułamek dziesiętny
exponent part	- mnożnik skalujący /cecha/
decimal number	- liczba dziesiętna
number	- liczba

< unsigned integer > ::= < digit > | < unsigned integer > < digit >

np. 3 001 26387 300 04

< integer > ::= < unsigned integer >|+ < unsigned integer >|
- < unsigned integer >

np. 3 +26 -684 -0001 +04

< decimal fraction > ::= . < unsigned integer >

np. .684 .001 .2 .3000 .0

< exponent part > ::= ₁₀ < integer >

np. ₁₀20 ₁₀+15 ₁₀-02 ₁₀0 ₁₀+101

< decimal number > ::= < unsigned integer >|< decimal fraction >|
< unsigned integer > < decimal fraction >

np. 0345 .38 25.25 0.1840 0.00

< unsigned number > ::= < decimal number >|< exponent part >|
< decimal number > < exponent part >

np. .38 000 ₁₀15 $3_{10}+3$ $9.34_{10}-12$

< number > ::= < unsigned number >|+ < unsigned number >|
- < unsigned number >

3 - 5. Uwagi

1. W definicji liczby oprócz cyfr występują symbole podstawowe ₁₀ . + - /por. 2-1.4.1,2/.
2. Umieszczenie mnożnika skalującego nie jest konieczne. Tę samą liczbę można przedstawić na wiele sposobów, np. 384 0384 $384_{10}0$ $+3.84_{10}2$ $0.00384_{10}+5$ $38400_{10}-2$
3. Zgodnie z definicją, liczba z mnożnikiem skalującym jest typu <u>real</u> /bo nie jest liczbą całkowitą w ALGOLu/.

3 - 5.1. GIER ALGOL

W GIER ALGOLu nałożone są na liczby następujące ograniczenia:

1. Liczby typu <u>integer</u> muszą spełniać nierówności:

$$-536\,870\,912 \leq x \leq 536\,870\,911$$

2. Liczby typu real muszą spełniać nierówności:

$$7.458_{10}-155 \leq |x| \leq 1.341_{10}154$$

3. Liczby typu real przedstawiane są z dokładnością względną rzędu $3_{10}-9$ /tzn. mają 8-9 cyfr znaczących/.
4. Gdy $|x|<7.458_{10}-155$,wówczas przyjmowane jest x = 0.
   Gdy $|x|>1.341_{10}154$,wówczas realizacja programu jest zatrzymana, maszyna podaje odpowiedni sygnał przez urządzenie wejścia-wyjścia.

3 - 6. Ćwiczenia

1. Wskazać i uzasadnić, które z następujących ciągów symboli nie przedstawiają liczby w ALGOLu:

   1) 3,1429
   2) $-5_{10}+3$
   3) $-2._{10}4$
   4) $3 \times {}_{10}5$
   5) $-.3_{10}-5$
   6) $4_{10}-n$
   7) 384.121 00 $2_{10}-23$
   8) 1/7
   9) 1.234.567
   10) 24.81 08

2. Napisać liczby o tych samych wartościach, co niżej podane, ale bez mnożnika skalującego:

   1) $+ 7.293_{10}8$
   2) $98.12_{10}+2$
   3) ${}_{10}+3$
   4) $-.1834_{10}-5$
   5) $-{}_{10}-6$
   6) $-4.8_{10}3$

3. Zapisać liczby mające te same wartości, co niżej podane, przy użyciu minimalnej ilości symboli podstawowych:

   1) + 17000
   2) 1 000
   3) - 0.00134
   4) + 1.00024
   5) - 0.0020041298
   6) 170

4. NAZWY

W konwencjonalnym zapisie matematycznym stosuje się dla oznaczenia pewnych wielkości /zmiennych, funkcji,współczynników, elementów macierzy itp./ litery różnych alfabetów, ewentualnie kombinacje liter z innymi znakami, np. x, Z, a_{36} , sin, x', $\bar{y}$, A_{ijk}. Zapisując algorytmy w ALGOLu, musimy również powiązać takie wielkości z pewnymi symbolami, w celu ich identyfikacji. Dlatego też należy ściśle określić, jakie symbole mogą być używane jako nazwy / identyfikatory/. Z punktu widzenia języka, jest znacznym uproszczeniem utożsamianie danej wielkości z jej nazwą.

4 - 1. Wstępne informacje

1. Nazwa jest to dowolnej długości ciąg liter i cyfr, zaczynający się od litery /patrz 2-1.1,2./.
2. Pomiędzy litery i cyfry mogą być wstawiane odstępy. Dwie nazwy, różniące się tylko odstępami, są identyczne.
3. Pewna ilość nazw powinna być zarezerwowana w każdej dowolnej reprezentacji, do oznaczania tzw. procedur i funkcji standardowych /będą one omówione później/.

4 - 1.1. Przykłady

a	alfa i beta	X1Y1Z122abc01
A3	NEWTON	KURS ALGOLu
Test20	input output 3	napiszN2c7 i 5

4 - 2. Opis w metajęzyku

identifier - nazwa

< identifier > ::= < letter >|< identifier > < letter >|
< identifier > < digit >

4 - 3. Uwagi

1. W nazwach języka wzorcowego mogą występować małe i duże litery alfabetu angielskiego /zgodnie z 2-1.1./.
2. Ta sama nazwa nie może być użyta do równoczesnego oznaczenia dwu różnych wielkości.

4 - 3.1. GIER ALGOL

1. W nazwach mogą występować dołączone do alfabetu angielskiego cztery litery duńskie æ,Æ,ø,Ø.
   np. Æ121 abcø A2Ø
2. 39 nazw jest zastrzeżonych dla funkcji, procedur i zmiennych standardowych. Wykaz ich podany jest w części skryptu, poświęconej GIER ALGOLowi.

4 - 4. Ćwiczenia

4. Wskazać i uzasadnić, które z poniżej podanych ciągów symboli nie mogą być nazwami:

   1) ppp3
   2) 4P2
   3) begin
   4) begin
   5) a 29 ∨ 3
   6) ∨ 7
   7) Start value
   8) B.B.
   9) epsilon
   10) + suma
   11) ab
   12) a×b
   13) Q12(3)
   14) Ile masz lat

15) π

16) ALGOL

17) GIER - ALGOL

18) aB134ccc8 236 7BBa

19) alfa i beta

20) q

5. ZMIENNE

Pojęcie zmiennej jest analogiczne do pojęcia znanego w matematyce.

5 - 1. Wstępne informacje

1. Przez zmienną obecnie będziemy rozumieli wielkość, na którą podstawiana jest wartość liczbowa.
2. Zmienną w ALGOLu może być zmienna prosta lub zmienna indeksowana /ze wskaźnikami/. Definicję zmiennej indeksowanej omówimy w rozdziale 12.
3. Zmiennej przyporządkowuje się nazwę - pojęcie zdefiniowane w poprzednim rozdziale.

5 - 2. Typy zmiennych

Ponieważ na zmienne podstawiane są liczby typu real lub typu integer, zmienne również są typu real lub typu integer. Wartością zmiennej typu integer jest liczba całkowita. Wartością zmiennej typu real jest liczba, która nie jest całkowita /por. 3-3/.

5 - 2.1. Opis w metajęzyku

type - typ

< type > ::= integer|real (æ)

W rozdziale 14 wprowadzony zostanie nowy typ zmiennych, który uzupełni definicję pojęcia < type >

5 - 3. Opis w metajęzyku

simple variable - zmienna prosta
subscripted variable - zmienna indeksowana

< variable identifier > ::= < identifier >

< simple variable > ::= < variable identifier >

< variable > ::= < simple variable >|< subscripted variable >

5 - 4. Uwagi

Zmienna używana jest w wyrażeniach arytmetycznych/rozdział 6 i dalsze/, do formowania nowych wartości. Wartość zmiennej może być zmieniona przez użycie instrukcji podstawienia /rozdział 9/.

5 - 4.1. Ścisła definicja zmiennej

Zmienna jest to nazwa dana pojedynczej wartości.Przez wartość rozumiemy obecnie liczbę - pojęcie to będzie później rozszerzone.

6. INFORMACJE O STOSOWANIU OPERATORÓW ARYTMETYCZNYCH

W punkcie 2-1.4.1. zdefiniowaliśmy operatory arytmetyczne. Znaczenie poszczególnych symboli użytych w tej definicji, jest następujące:

+ operator dodawania
- operator odejmowania
× operator mnożenia
/ operator dzielenia
: operator całkowitego dzielenia /wynikiem a:b jest część całkowita ilorazu dwu liczb całkowitych a/b/.
↑ operator potęgowania

6 - 1. Wstępne informacje

1. Łącząc /według pewnych reguł/ operatorami arytmetycznymi liczby oraz zmienne, otrzymujemy najprostsze przykłady wyrażeń arytmetycznych.

2. Rola nawiasów okrągłych w wyrażeniach arytmetycznych omówiona jest w paragrafie 6-2.

3. Znak mnożenia nie może być opuszczony /np. zamiast zwykłego zapisu algebraicznego 3y, w ALGOLu należy zapisać 3×y/.

4. Dwa operatory nie mogą występować bezpośrednio obok siebie /np. 4↑-2 jest zapisem błędnym - należy go zastąpić przez 4↑(-2) /.

6 - 1.1. Przykłady

3 + beta	(7 + 18 × c) × 8.02₁₀-4
c - a + 4 × d	a ↑ (b ↑ c) : 84 + a
AL × 3₁₀3	1 - 2
4 : integer + real	(-3.2₁₀-18 + 82 × c)×(+ 4₁₀5)
a/b + 3 × c - d ↑ 18	a - B + c × D / e : F ↑ g
a ↑ 3 + b ↑ 4 + c ↑ C	(84/84 + 3↑(-8) × b)/a - 7

6 - 2. Wartość wyrażenia arytmetycznego

Wyrażenie arytmetyczne służy do wyznaczenia wartości liczbowej. Wartość ta obliczana jest przez wykonanie wskazanych operacji arytmetycznych na liczbach i aktualnych wartościach liczbowych zmiennych, według pewnych reguł /analogicznych do przyjętych w arytmetyce/.

Zasadniczo operacje wykonywane są kolejno od lewej strony, do prawej, przy czym:

1. O ile występują różne rodzaje operatorów wówczas respektowane są następujące reguły pierwszeństwa:
   1/ ↑
   2/ × / :
   3/ + -
2. Jeżeli po operatorze arytmetycznym umieszczony jest nawias (, wówczas operacja wskazana tym operatorem wykonywana jest dopiero po obliczeniu wartości wyrażenia ujętego w nawiasy ( ) .
3. Działania a↑b, a/b oraz a:b nie są określone dla dowolnych liczb a, b /patrz punkty 8-3.3,4,5/.

6 - 2.1. Przykłady

Po lewej stronie podane są przykłady wyrażeń arytmetycznych w ALGOLu, po prawej stronie ich odpowiedniki w zapisie algebraicznym. Jeżeli w ALGOLu podane są postacie

równoważne, wówczas odpowiednik algebraiczny występuje tylko raz.

a / (b × c) a / b / c a / c / b (a / b)/c (a / c)/b	$\frac{a}{bc}$
a × b / c ↑ d / e × f ((((a × b)/(c ↑ d))/e) × f)	$\frac{ab}{ec^d}$
a ↑ b ↑ c (a ↑ b) ↑ c	$(a^b)^c$
a ↑ (b ↑ c)	a^{b^c}
a ↑ b + c	$a^b + c$
a ↑ (b + c)	a^{b+c}
a ↑ (-c)	a^{-c}
a + b + b ↑ 2/2 + b ↑ 3/3	$a + b + \frac{b^2}{2} + \frac{b^3}{3}$
a / b + c / d	$\frac{a}{b} + \frac{c}{d}$
(a + b)/(c + d)	$\frac{a + b}{c + d}$
a + b / c + d	$a + \frac{b}{c} + d$
a × z ↑ 2 + b × z + c	$az^2 + bz + c$

6 - 3. Uwagi

1. Każda liczba i zmienna jest wyrażeniem arytmetycznym.
2. Pojęcie wyrażenia arytmetycznego zostanie omówione szczegółowo i zdefiniowane w dalszych rozdziałach.

6 - 4. Ćwiczenia

5. Napisać następujące wyrażenia w ALGOLu, opuszczając zbędne nawiasy:

1) $A+\frac{s-t}{v^2}$ 2) $(x+y)^3$ 3) $x+y^3$ 4) $v(-z)$ 5) $\frac{p^q}{r^{s+t}}$

6) a^{b+c^d} 7) $1+x^{-2}+ay^{-1+c}$ 8) $(\frac{a}{10000} + 0{,}000005)c$

6. Wskazać i uzasadnić, które z następujących ciągów symboli nie mogą być wyrażeniami arytmetycznymi w ALGOLu:

1) $2_{10}6 + 4.3 + Q2$
2) $2 \times {}_{10}6/4.3$
3) $.84_{10}(7 + n)/4$
4) p/qrs + tu - v
5) PQ ↑ + 4.3
6) -(+(-(+(-v))))

7. Zakładając, że zmienne a, b, c mają wartości aktualne odpowiednio 1, 2, 3, znaleźć wartości następujących wyrażeń arytmetycznych:

1) a + b/3 + c ↑ 2
2) a/b/c × 2
3) (a - b × c) ↑2 ↑ b
4) (1 - b) ↑ (-c) ↑ (a + 1)
5) 3 : b
6) 2 : b
7) 1 : b
8) + a - b × (c + a ↑ (b - c))

7. FUNKCJE STANDARDOWE. NAZEWNIKI FUNKCYJNE

7 - 1. Funkcje standardowe

W rozdziale 4 wspominaliśmy, że pewne nazwy powinny być zarezerwowane dla funkcji standardowych.

7 - 1.1. Wstępne informacje

W języku wzorcowym zaleca się, by lista zarezerwowanych nazw zawierała następujące nazwy:

abs(W) dla oznaczenia modułu wartości W

sign(W) dla oznaczenia znaku wartości W, tzn. +1 dla W>0, 0 dla W=0, -1 dla W<0

sqrt(W) dla oznaczenia pierwiastka kwadratowego z wartości W

sin(W) dla oznaczenia sinusa wartości W

cos(W) dla oznaczenia cosinusa wartości W

arctan(W) dla oznaczenia wartości głównej arcus tangens W, tzn. | arctan(W) | $\leq \pi/2$

ln(W) dla oznaczenia logarytmu naturalnego wartości W

exp(W) dla oznaczenia e do potęgi wartość W

entier(W) dla oznaczenia największej liczby całkowitej, nieprzekraczającej W.

W oznacza wyrażenie arytmetyczne, jednak w kilku miejscach było utożsamiane z wartością wyrażenia /np. przy zapisie W > 0/.

7 - 1.2. Uwagi

Język wzorcowy jedynie zaleca umieszczenie w konkretnej reprezentacji wymienionych powyżej funkcji standardowych. Dokładniejsze omówienie tych funkcji jest możliwe tylko w przypadku konkretnej reprezentacji.

7 - 1.2.1. GIER ALGOL

1. W GIER ALGOLu występują wszystkie, z listy podanej w 7-1.1, funkcje standardowe.
2. Argument funkcji /wyrażenie arytmetyczne/ musi być umieszczony w nawiasach okrągłych.
3. Funkcja sqrt(W) określona jest dla $W \geq 0$, zaś ln(W) dla $W > 0$, oraz ln(0) oznacza liczbę $-9.35_{10}49$.
4. Wartości funkcji standardowych wyznaczane są z określonym przybliżeniem. Niech F (x) oznacza jedną z następujących funkcji /argumentu rzeczywistego x/: sinus, cosinus, arcus tangens, logarytm i funkcja wykładnicza o podstawie naturalnej, pierwiastek kwadratowy, zaś f(x) odpowiadnią funkcję standardową GIER ALGOLu. Wtedy dla każdej liczby x /z przedziału określoności danej funkcji i z zakresu liczb rzeczywistych GIER ALGOLu/, zachodzi następująca nierówność:

$$|F(x) - f(x)| \leq 4.5_{10}-9 \times \max(|f(x)|, |F'(x) \times x|)$$

gdzie F' oznacza pochodną funkcji F.

7 - 1.3. Przykłady

abs(5 × c - 7)	sin(a + k × pi)	cos(alfa/2)
sign(delta)	ln(a ↑ 2 + b ↑ 2)	sqrt((33 + B)/7)
exp(-3)	arctan(c ↑ 3 : 2)	entier(a×b/c/10)

7 - 2. Nazewniki funkcyjne

Wprowadzone w poprzednim paragrafie funkcje standardowe, są najprostszymi przykładami tzw. nazewników funkcyjnych. Pojęcie to będzie omówione dokładniej w części skryptu, poświęconej procedurom.

7 - 2.1. Wstępne informacje

1. Nazewnik funkcyjny składa się z nazwy /np.sin, exp, ln/ oraz parametrów aktualnych ujętych w okrągłe nawiasy. W przypadku funkcji standardowych parametrami aktualnymi są wyrażenia arytmetyczne /np. 3.1&3, x1, 2 × alfa - b/.

2. Nazewnik funkcyjny posiada wartość liczbową. W przypadku funkcji standardowych, wartość ta jest wyznaczana zgodnie z opisem w punkcie 7-1.1.

7 - 2.2. Uwagi

Do czasu podania dalszych informacji o nazewniku funkcyjnym należy rozumieć pod tym pojęciem dowolną funkcję standardową.

8. PROSTE WYRAŻENIA ARYTMETYCZNE

W rozdziale 6 podane były przykłady szczególnej postaci wyrażeń arytmetycznych, tzw. prostych wyrażeń arytmetycznych.

8 - 1. Wstępne informacje

1. Przy opisie prostego wyrażenia arytmetycznego, istotną rolę odgrywa pierwotne wyrażenie arytmetyczne. Pierwotnym wyrażeniem arytmetycznym jest liczba bez znaku, zmienna, nazewnik funkcyjny lub dowolne wyrażenie arytmetyczne, ujęte w okrągłe nawiasy,

   np. b1 3.14 ln(a ↑ 2 - 1) (c × d / e - 5)

2. Proste wyrażenie arytmetyczne składa się z kilku pierwotnych wyrażeń arytmetycznych, połączonych /według pewnych zasad/ operatorami arytmetycznymi.
3. Każde pojedyncze wyrażenie pierwotne jest również prostym wyrażeniem arytmetycznym.
4. Wartość prostego wyrażenia arytmetycznego wyznaczana jest zgodnie z opisem w paragrafie 6-2.
5. Przez wyrażenie arytmetyczne będziemy obecnie rozumieli proste wyrażenie arytmetyczne. W ogólnej postaci wyrażenie arytmetyczne składa się z kilku prostych wyrażeń arytmetycznych, wraz z regułą wyboru jednego z nich. Wartością wyrażenia arytmetycznego jest wartość tego wybranego prostego wyrażenia /dokładna definicja wyra-

żeń arytmetycznych podana jest w rozdziale 16/.

6. Ponieważ argumentami funkcji standardowych są wyrażenia arytmetyczne, zaś same funkcje mogą występować w wyrażeniach arytmetycznych /jako nazewniki funkcyjne/, więc poprawne są na przykład następujące wyrażenia:

ln(ln(x)) abs(cos(x) - 0.5) sqrt(sin(a ↑ 2) ↑ 3)

8 - 1.1. Przykłady

+88	cos(cos(cos(alfa+1)+2)+3)×4/d
(-2.17₁₀3)	((A×arctan(y)+Z)↑(3))×(A+B)
a+sin(a)	(1)
U2 × U3	YE+WE-a12×sum/7.39₁₀8↑(a-b)
cos(y+z×3×ln(v))	(a+b+sqrt(abs(b-a)))↑2-2.2

8 - 2. Opis w metajęzyku

adding operator - operator typu dodawania
multiplying operator - operator typu mnożenia
primary - wyrażenie pierwotne
function designator - nazewnik funkcyjny
arithmetic expression - wyrażenie arytmetyczne
factor - czynnik
term - składnik
simple arithmetic expression - proste wyrażenie arytmetyczne

< adding operator > ::= +|-

< multiplying operator > ::= ×|/|:

< primary > ::= < unsigned integer >|< variable >|
< function designator >|(< arithmetic expression >)

np. $4_{10}+6$ 3.2 a1b2 ln(3+pi/8)
(a+b) (delta) $(0.3_{10}-05+sqrt(exp(3×d)+1))$

< factor > ::= < primary >|< factor > ↑ < primary >

np. omega (a+b/c) 3↑(1/3) x↑y

z↑(a+b/c↑d) a↑b↑c↑(d-f×g↑h/e)↑p

< term > ::= < factor >|< term > < multiplying operator > < factor >

np. 4.7 alfa 2 1/3 a↑b×c
4:a sin(x)/ln(z) a↑3×3$_{10}$-2/(a+b)

< simple arithmetic expression > ::= <term>|<adding operator> <term>|
<simple arithmetic expression> <adding operator> <term>

< arithmetic expression > ::= < simple arithmetic expression > (æ)

8 - 3. Typy prostych wyrażeń arytmetycznych

Wartością prostego wyrażenia arytmetycznego jest liczba typu real lub integer. Należy zatem określić, od czego zależy typ wyrażenia. Wystarczy w tym celu rozpatrzyć typ funkcji standardowych oraz typ wyrażenia, postaci:

< argument > < operator arytmetyczny > < argument >

8 - 3.1. Funkcje standardowe

Funkcje sign(W) i entier(W) są typu integer, pozostałe są typu real - dla dowolnego argumentu.

8 - 3.2. A + B, A - B, A × B

Jeżeli obydwa argumenty są typu integer, wówczas wynik jest typu integer, w przeciwnym razie real.

8 - 3.3. A/B

Wyrażenie określone jest tylko dla dzielnika różnego od zera. Wynik jest zawsze typu real.

8 - 3.4. A : B

Wyrażenie określone jest tylko dla obu argumentów typu integer. Ścisłą definicję operatora całkowitego dzielenia możemy zapisać równością:

$$a : b = \text{sign}(a/b) \times \text{entier}(\text{abs}(a/b))$$

8 - 3.5. A ↑ B

Połóżmy dla uproszczenia: i - liczba typu integer, r - liczba typu real, b - liczba dowolnego typu, wówczas:

	WYKŁADNIK	PODSTAWA	WYNIK	TYP WYNIKU
b ↑ i	i > 0	dowolna	b×b×...×b (i razy)	zgodny z typem b
	i = 0	b ≠ 0	1	
		b = 0	nieokreślony	
	i < 0	b ≠ 0	1/(b×b×...×b) (-i razy)	real
		b = 0	nieokreślony	
b ↑ r	dowolny	b > 0	exp(r × ln(b))	real
	r > 0	b = 0	0.0	real
	r ≤ 0		nieokreślony	
	dowolny	b < 0	nieokreślony	

300 + 5 = 305 typ integer

3.2 + 5 = 305.0 typ real

3.0 - 2 = 1.0 typ real

5 × 3 = 15 typ integer

5.0 × 3 = 15.0 typ real

6 / 2 = 3.0 typ real

(-5) : 3 = -1 typ integer

5.1 : 3 nieokreślone

2.0 ↑ 3 = 8.0 typ real

2000 ↑ 0 = 1 typ integer

2.3/0 = 1.0 typ real

0/0 nieokreślone

1/(-3) = 1.0 typ real

0.2/(-5) nieokreślone

3/0.0 = 1.0 typ real

0/2 = 0 typ integer

0/2.0 = 0.0 typ real

0.0/2 = 0.0 typ real

(-3)/2 = 9 typ integer

(-3)/2.0 nieokreślone

9. INSTRUKCJE PODSTAWIENIA

Zmienne występujące w wyrażeniach arytmetycznych muszą posiadać aktualne wartości. Wartości te mogą być nadawane zmiennym przy pomocy instrukcji podstawienia.

9 - 1. Wstępne informacje

1. Najprostsza postać instrukcji podstawienia ma następującą budowę:

   < instrukcja podstawienia > ::= <zmiennna>:=<wyrażenie arytmetyczne>

   np. ALFA := 32
   a := b × 3 + sqrt(c) : d ↑ e
   n := n + 1

2. Znak := jest symbolem podstawowym /por. 2-4.1.2/ i oznacza operację podstawienia.
3. Sposób wykonania instrukcji: obliczana jest wartość wyrażenia arytmetycznego i podstawiana na zmienną występującą po lewej stronie.
4. W postaci bardziej ogólnej na lewo od wyrażenia arytmetycznego znajduje się tzw. wykaz lewych stron. Wykaz lewych stron utworzony jest przez kilka /conajmniej jedną/ zmiennych, z następującym po każdej symbolem :=

   np. a :=
   b1:= epsilon := alfa :=
   A := B := C := delta := alfa i beta :=

9 - 1.1. Przykłady

```
a := 3
b1:= epsilon := alfa := a ↑ 2 + sin(b1 × ln(As)/10)
A := B := C := delta := alfa i beta := (x - 3) : (y - 2)
```

9 - 2. Typy

Zmienne występujące w wykazie lewych stron muszą być wszystkie albo typu real, albo typu integer. Jeżeli zmienne są typu integer, a wyrażenie typu real, wówczas wartość wyrażenia jest zaokrąglona. Równoważne jest to podstawieniu na zmienne wartości:

entier (W + 0.5)

gdzie W oznacza wyrażenie arytmetyczne, występujące po prawej stronie.

9 - 2.1. Przykłady

Załóżmy, że zmienna n jest typu integer. Wówczas

n := 2.4 jest równoważne podstawieniu	n := 2
n := 2.7	n := 3
n :=-2.3	n :=-2
n := 0.5	n := 1
n := 0.0	n := 0
n :=-0.5	n := 0

9 - 3. Opis w metajęzyku

left part	- lewa strona
left part list	- wykaz lewych stron
assignment statement	- instrukcja podstawienia

< left part > ::= < variable > := (æ)

< left part list > ::= < left part > | < left part list > < left part >

< assignment statement > ::=

< left part list > < arithmetic expression > (æ)

9 - 4. Uwagi

1. Pojęcie instrukcji podstawienia zostanie w dalszych paragrafach rozszerzone. Wiązać się z tym będą modyfikacje sposobu jej wykonania /podanego w p.3 paragrafu 9-1/.
2. Zwróćmy uwagę na dynamiczny charakter instrukcji podstawienia. Dzięki temu poprawny jest np. następujący zapis: n := n + 1, co należy czytać: nowa wartość zmiennej n staje się równa poprzedniej wartości zmiennej n, zwiększonej o jeden. Symbol := należy rozumieć: "staje się" w odróżnieniu od matematycznego = /równa się/, oznaczającego jedynie relację między wielkościami po lewej i prawej stronie, a nie efektywne podstawienie.

9 - 5. Ćwiczenia

8. Znaleźć błędy w następujących instrukcjach podstawienia:
   1) xy1 := xy2 := (x - a)b/c + 1 ↑(D - E) (F - G)
   2) p := -q := 3 ↑ 2 - p
   3) a := b := c := b + 1 + sqrt(-3)/2 + 14
   4) a + b := c - 4 ↑ d + 2a × b
   5) 5 := 13 - 8 × q ↑ (a + b) : 2 - 1

10. INFORMACJE O PROCEDURACH WEJŚCIA - WYJŚCIA W GIER ALGOLU

W dalszej części materiału podawane będą przykłady programów w ALGOLu. Język wzorcowy nie zawiera procedur wejścia-wyjścia, tzn. instrukcji, umożliwiających wprowadzanie danych i wyprowadzanie wyników przez urządzenia wejścia-wyjścia maszyny cyfrowej. Dlatego też programy będą zapisywane w GIER ALGOLu.

Różnice między programem w ALGOLu i GIER ALGOLu polegać będą jedynie na umieszczeniu w programie GIER ALGOLowych procedur wejścia-wyjścia. Omówimy je teraz bardzo pobieżnie, biorąc dla przykładu procedurę wejścia input i procedurę wyjścia output. Dokładna definicja tych oraz pozostałych procedur GIER ALGOLu podana będzie w dalszej części skryptu.

10 - 1. Procedura input

Zakładamy, że dane, które należy wprowadzić do programu mają postać układu liczb oddzielonych przecinkami, wyperforowanego na taśmie papierowej /przy pomocy specjalnego urządzenia/.

10 - 1.1. Uproszczony opis w metajęzyku

< parametr wejścia > ::= < zmienna >

< wykaz parametrów wejścia > ::= < parametr wejścia >|

< wykaz parametrów wejścia >,< parametr wejścia >

np. Alfa a,b,c A1,B2

< instrukcja input > ::= input(< wykaz parametrów wejścia >)

np. input(Alfa) input(a,b,c) input(A1,B2)

10 - 1.2. Działanie

Działanie procedury input objaśnimy na przykładzie. Załóżmy, że na taśmię wyperforowany jest następujący układ liczb:

-1, 0, -13.5, 27, 0.333

wówczas instrukcja

input (a, b, c, d, e)

równoważna jest pięciu instrukcjom podstawienia:

a := -1, b := 0, c := -13.5, d := 27, e := 0.333

10 - 2. Procedura output.

Procedura output powoduje wyprowadzenie wyników w postaci wyperforowanej taśmy papierowej. Liczby wydziurkowane na taśmie można następnie odczytać za pomocą specjalnego urządzenia.

10 - 2.1. Wzorzec

Przy wyprowadzaniu liczby konieczne jest dostarczenie informacji o żądanym sposobie zapisu tej liczby, tzn. ilość cyfr, ewentualne dodanie znaku, ewentualne dodanie mnożnika skalującego itd. Służy do tego tzw. wzorzec /layout/. Ilość możliwych sposobów wyperforowania liczby /tzn. ilość możliwych wzorców/ jest bardzo duża. W paragrafie tym omówimy tylko jeden przykładowy wzorzec, postaci:

$\{+d.dddd_{10}+dd\}$

Zawarte są w nim następujące informacje:

1. Nawiasy ⟨ ⟩ oznaczają, że jest to wzorzec wyperforowania liczby.
2. Liczba dodatnia będzie poprzedzona plusem, liczba ujemna minusem; żądanie to jest wyrażone przez umieszczenie znaku + po symbolu ⟨ .
3. Liczba będzie wyperforowana z jedną cyfrą przed kropką pozycyjną i czterema po kropce; żądanie to wyraża we wzorcu ciąg d.dddd. O ile ilość cyfr znaczących jest większa - nastąpi zaokrąglenie.
4. Jeżeli wartość absolutna liczby jest mniejsza od 1 lub większa od 9.9999, to zostanie dobrany automatycznie mnożnik skalujący z liczbą całkowitą z przedziału domkniętego /-99, +99/ taką, by cyfry znaczące były wydrukowane w postaci omówionej w punkcie 3; żądanie to wyrażone jest pozostałą częścią wzorca, tzn. ciągiem $_{10}$+ dd .

10 - 2.1.1. Przykłady

Liczba	liczba zapisana w/g wzorca
	⟨+d.dddd$_n$+dd⟩
1	+1.0000
-3.15	-3.1500
4.856	+4.8560
32	+3.2000 $_n$+1
485.22	+4.8522 $_n$+2
-0.0012	-1.2000 $_n$-3
36.6666	+3.6667 $_n$+1
234567.891	+2.3457 $_n$+5
-.2$_n$-31	-2.0000$_n$-32
84.2$_n$15	+8.4200$_n$+16
.111111111	+1.1111 $_n$-1

10 – 2.2. Uproszczony opis w metajęzyku

< parametr wyjścia > ::= < wyrażenie arytmetyczne >

np. 3.14 alfa 2 × 6 - b ↑ 3

< wykaz parametrów wyjścia > ::= < parametr wyjścia > |
< wykaz parametrów wyjścia >,< parametr wyjścia >

np. a,b,3.14 alfa,2 × 6 - b ↑ 3,18,b

< wzorzec > ::= {+d.dddd$_{10}$+dd}

< instrukcja output > ::=
output(< wzorzec >,< wykaz parametrów wyjścia >)

np. output({+d.dddd$_{10}$+dd}, a, b, 3.14)
output({+d.dddd$_{10}$+dd}, alfa, 2×6 - b/3, 18, b)
output({+d.dddd$_{10}$+dd}, 1)

10 – 2.3. Działanie

Instrukcja output powoduje wyperforowanie na taśmie w/g podanego wzorca wartości wyrażeń arytmetycznych wymienionych w wykazie parametrów wyjścia. Np.:

output({+d.dddd$_{10}$+dd}, alfa, 2×6 - b/3, 18, b)

przy założeniu, że alfa = -3, b = 1.5 spowoduje wydrukowanie następujących liczb

-3.0000 +8.6250 +1.8000 $_{10}$+1 +1.5000

11. PROGRAM. DEKLARACJE

Zanalizujemy obecnie program na obliczanie części całkowitej stosunku objętości kuli /o żądanym promieniu R/ do objętości stożka /o wysokości h i promieniu podstawy r/.

```
begin
real r,h,R,Vkuli,Vstozka,pi;
integer Vk do Vs;
      input(r,h,R);
      pi := 3.1416;
      Vkuli := 4 × pi × R ↑ 3 / 3;
      Vstozka := pi × h × r ↑ 2 / 3;
      Vk do Vs := entier(Vkuli/Vstozka);
      output(‹+d.dddd₁₀+dd›,Vk do Vs)
end
```

11 - 1. Program

11 - 1.1. Wstępne informacje

1. Program w ALGOLu zaczyna się symbolem podstawowym begin i kończy symbolem podstawowym end /por. 2-1.4.3/.
2. Prawie każdy program ma postać bloku, tzn. po symbolu begin występują deklaracje. Znaczenie ich omówimy w następnym paragrafie. W powyżej podanym przykładzie deklaracje oddzielone są średnikiem od pierwszej instrukcji programu input (r, h, R) .

3. Po deklaracjach występują instrukcje programu. Realizowane są one w kolejności zapisania.
4. Instrukcje oddzielane są średnikiem /symbol podstawowy, por. 2-1.4.2/.
5. Po ostatniej instrukcji przed symbolem end nie ma średnika.

11 - 2. Deklaracje

Nazwy występujące w instrukcjach programu nie zawierają w sobie informacji o rodzaju i typie wielkości, które reprezentują. Do prawidłowego realizowania programu informacje te są konieczne, więc umieszcza się je na początku programu /po symbolu begin/ w formie tzw. deklaracji.

11 - 2.1. Deklaracje zmiennych prostych

< wykaz zmiennych > ::= < zmienna prosta >|
< wykaz zmiennych >,< zmienna prosta >

np. AL c a, b, K, eps

< deklaracja zmiennych prostych > ::= < typ > < wykaz zmiennych >

np. real AL real a, b, K, eps
integer c

Deklaracja tej postaci podaje informację, że nazwy wymienione w wykazie zmiennych oznaczają w programie zmienne proste typu odpowiednio real lub integer. W przykładzie podanym na początku tego rozdziału deklaracje są następujące:
real r, h, R, Vkuli, Vstożka, pi
/w programie nazwy te będą oznaczały zmienne proste typu real/,
integer Vk do Vs
/w programie będzie jedna zmienna typu integer o nazwie Vk do Vs/.

11 - 2.2. Deklaracje nazewników funkcyjnych

Funkcje standardowe nie są w programie deklarowane. Pozostałe nazewniki funkcyjne należy deklarować. Zostanie to omówione w części skryptu poświęconej procedurom.

11 - 3. Uwagi

1. Deklaracje w programie oddzielane są /podobnie jak instrukcje/ średnikami.
2. W przypadku realizacji w dowolnej maszynie programu tłumaczącego z ALGOLu na język wewnętrzny maszyny /tzw.compilera/, deklaracja powoduje rezerwowanie odpowiedniej ilości miejsc w pamięci maszyny.
3. Przy wprowadzaniu nowych wielkości w ALGOLu identyfikowanych nazwami, będzie podawany od razu sposób deklarowania tych wielkości w programie.

11 - 4. Ćwiczenia

9. Znaleźć końcowe wartości wszystkich zmiennych w niżej podanych programach:

1/
```
begin real a,b,p,q;
   a:= b:= 7;
   p:= a + 3 × b - 2.3₁₀-1;
   q:= p +(a+3)/(-b-13);
   a:= p:= q - b × 0.2
end
```

2/
```
begin real r1,ra,rb;
integer n,i,j;
   n:= 5;
   r1:= n/(n+15);
   rb:= n+6/(6×r1+0.5);
   i:= n:= n-2;
```

```
  j:= rb-i;
  ra:= (j-1)×r1×(rb-4);
  r1:= ra+rb+n+i+j+8×r1;
  rb:= (r1-rb×n+j-ra) ↑ (rb-j) + ra;
  j:= n:= 1 + n : (j-2);
  i:= n + ra
end
```

12. TABLICE. ZMIENNE INDEKSOWANE

W wielu zagadnieniach matematycznych wygodne jest posługiwanie się zmiennymi ze wskaźnikami. Na przykład współczynniki wielomianu oznacza się często a_0, a_1, ..., a_n, elementy macierzy prostokątnej b_{11}, b_{12},..., b_{pq}. Dzięki temu wiele wzorów i algorytmów matematycznych można zapisać w prostszej postaci.

W ALGOLu istnieje też możliwość posługiwania się zmiennymi ze wskaźnikami. W tym celu wprowadzone zostało pojęcie zmiennej indeksowanej. Wskaźniki zapisuje się nie u dołu, lecz w nawiasach kwadratowych, np.

```
a[0], a[1],....,a[n]
b[1,1],b[1,2],....,b[1,q],....,b[p,q]
```

Występowanie zmiennych indeksowanych w programie sygnalizuje deklaracja tablicy jedno- lub wielowymiarowej, ponieważ zmienne indeksowane interpretowane są jako elementy tablicy.

12 - 1. Wstępne informacje

1. Tablica jest zbiorem zmiennych, zwanych zmiennymi indeksowanymi.
2. Na zmienne indeksowane można podstawić wartości liczbowe, podobnie jak na zmienne proste.
3. Tablice identyfikowane są przy pomocy nazw.
4. Zmienne indeksowane mają postać:

< zmienna indeksowana > ::= < nazwa tablicy > [< wskaźnik lub kilka wskaźników oddzielonych przecinkami >]

(gdzie wskaźnik jest liczbą całkowitą)

np. A[1] b[4,7] AL26c[3,3,-2]

5. Ilość wskaźników w zmiennej indeksowanej jest równa wymiarowi odpowiedniej tablicy, a zakres każdego wskaźnika jest wyznaczony przez odpowiednią parę graniczną w deklaracji tablicy.
6. Tablice muszą być w programie deklarowane, co jest równoznaczne z podaniem informacji o ilości i typie zmiennych indeksowanych występujących w programie.

12 - 2. Deklaracje tablic

Przed dokładnym opisaniem w metajęzyku deklaracji tablicy, podamy i objaśnimy najprostszą jej postać:

< deklaracja tablicy > ::= < typ > array < nazwa tablicy >[< para p:q lub kilka par tej postaci oddzielonych przecinkami >]

(gdzie p i q są liczbami całkowitymi)

np. real array A1[3:7]
integer array Beta[2:7,-3:2,0:8]

12 - 2.1. Uwagi

1. Typ oznacza typ zmiennych indeksowanych /elementów danej tablicy/.
2. array jest symbolem podstawowym /por. 2-1.4.4/.
3. Nazwa tablicy służy do identyfikacji wszystkich zmiennych indeksowanych - elementów danej tablicy.
4. Ilość par p:q oznacza ilość wskaźników zmiennej indeksowanej, a więc wymiar tablicy.
5. W każdej parze p:q ,p oznacza dolną, zaś q górną granicę między którymi zmienia się odpowiedni wskaźnik zmien-

nej indeksowanej. Musi być zawsze spełniona nierówność.
$p \leq q$.

12 - 2.2. Opis w metajęzyku

lower bound	- dolna granica
upper bound	- górna granica
bound pair	- para graniczna
bound pair list	- wykaz par granicznych
array	- tablica
array segment	- segment tablicy
array list	- wykaz tablic

< lower bound > ::= < arithmetic expression >

< upper bound > ::= < arithmetic expression >

< bound pair > ::= < lower bound > : < upper bound >

np. 3 : 10. -2 : n a×b:3+7×d/e+f

< bound pair list > ::= <bound pair>|<bound pair list>,<bound pair>

np. 3 : 10 3:10, -2:n, a×b:3+7×d/e+f

< array identifier > ::= < identifier >

< array segment > ::= < array identifier >[< bound pair list >]|

< array identifier >,< array segment >

np. ALFA[3:10]
B 12[3:10,-2:n,a×b:3+7×d/e+f]
c1,c2,c3[-5:5]

< array list > ::= < array segment >|< array list >,< array segment >

np. c1,c2,c3[-5:5]
c1,c2,c3[-5:5], ALFA[3:10], D1,D2[0:n,3×k:n2]

< array declaration > ::= <u>array</u> <array list>|<type> <u>array</u> <array list> (*)

12 - 2.2.1. Przykłady

```
array a,b,c[7:n,2:m],s[-2:10]
integer array INTEGER[a1:b1,a2:b2]
real array A1[1:p],A2[1:q],AS1,AS2[1:p,1:q]
array S4S5[a/c-6:k:82/AT]
```

12 - 2.2.2. Objaśnienia

Deklaracja tablicy powyżej zdefiniowana jest dużo ogólniejsza od postaci podanej w paragrafie 12-2.

1. Granice wskaźników są typu integer nie muszą jednak być dane liczbami całkowitymi; wartość ich może być określona wyrażeniami arytmetycznymi. O ile wyrażenie jest typu real to jego wartość jest zaokrąglana do najbliższej liczby całkowitej. Poza tym, tablica jest zdefiniowana tylko wtedy, gdy wartości wszystkich górnych granic są nie mniejsze od odpowiednich wartości dolnych granic. Na przykład:

   real array A[n/2 + 3.5 : n × m]

   przy n = 4.6 i m = 4 równoważne jest deklaracji:

   real array A[6 : 18]

   natomiast przy n = 4.6 i m = 1 tablica nie jest zdefiniowana.
2. Tablice tego samego typu można deklarować w jednej deklaracji, np. real array P1[-5:10],P2[n:m],As[0:k,-7:12] zamiast trzech oddzielnych deklaracji dla tablic P1, P2 i As.
3. Gdy w jednej deklaracji występuje kilka tablic o tych samych wymiarach i zakresach indeksów /czyli o tych samych wykazach par granicznych/, wówczas nie jest konieczne umieszczanie wykazu par granicznych za każdą naz-

wą. Wystarczy utworzyć segment tablic, tzn. podać listę nazw tablic rozdzielonych przecinkami i po ostatniej nazwie umieścić wykaz par granicznych /odnoszący się do wszystkich nazw/. Na przykład:

integer array ap1, ap2, ap3[p:q] zamiast
integer array ap1[p:q], ap2[p:q], ap3[p:q]

4. Określenie typu real można przed symbolem array opuścić i tak array M[1:3] jest równoważne real array M[1:3].

12 - 3. Zmienne indeksowane

Jak było już wspomniane, zmienne indeksowane są to elementy tablic. Korzystanie z tych zmiennych w programie musi być poprzedzone deklaracją odpowiedniej tablicy.

12 - 3.1. Opis w metajęzyku

subscript expression - wyrażenie wskaźnikowe
subscript list - wykaz wskaźników
subscripted variable - zmienna indeksowana /ze wskaźnikami/

< subscript expression > ::= < arithmetic expression >

< subscript list > ::= < subscript expression > |
< subscript list >,< subscript expression >

np. 4 × c 2 3,4,m alfa,2,H1,H2,H3

< subscripted variable > ::= < array identifier >[< subscript list >]

np. AL[3] epsilon[1,2,3,4] X[sin(n×pi/2),83]

12 - 3.2. Objaśnienia

1. Każdy wskaźnik jest wyrażeniem arytmetycznym. O ile wartość tego wyrażenia jest typu real, wówczas zostaje zaokrąglona do najbliższej liczby całkowitej.
2. Zmienna indeksowana jest zdefiniowana tylko dla takich

wskaźników, które zawarte są między odpowiednimi granicami wskaźników w deklaracji tablicy.

3. Typ zmiennych indeksowanych określony jest przez typ tablicy /w deklaracji tablicy/.

12 - 4. Uwagi

Po deklaracji tablic w programie umieszcza się średnik /tak jak i po innych deklaracjach, por. 11-3.p.1/.

12 - 4.1. Zmienne indeksowane w wyrażeniach arytmetycznych

Zgodnie z definicją podaną w paragrafie 5-3, zmienna w ALGOLu jest to zmienna prosta lub indeksowana. Zatem w prostym wyrażeniu arytmetycznym /a tym samym i w wyrażeniu arytmetycznym/ może występować zmienna indeksowana /por. 8-3/. Przed wykonaniem działania ze zmienną indeksowaną, wyznaczane są jej wskaźniki, tzn. obliczane są i ewentualnie zaokrąglane wartości wyrażeń wskaźnikowych.

12 - 4.1.1. Przykłady

```
3 × A1[4,5] + beta                    Y[1,ap[3,4],5]

sin(3 × cp2[7] × pi) + K[a / 2, b / 4]

A1[A2[A3[a×b, b/c, 3, c[1]]]]

(5 + cos(sqrt(X[sin(n × pi/2), Q[3,n,4]]) + 2))/7
```

12 - 4.2. Zmienne indeksowane w instrukcji podstawienia

Zmienne indeksowane mogą występować również w wykazie lewych stron instrukcji podstawienia. Na przykład:

```
a := b := c[7] := alfa := 3 + sin(x) × D[x,y]
```

Podamy obecnie pełny opis realizacji instrukcji podstawienia /por. 9-4.1/. Proces ten przebiega w następujący sposób:

1. Wyznaczane są wskaźniki wszystkich zmiennych indeksowanych, występujących w wykazie lewych stron, w kolejności od lewej do prawej strony.
2. Wyznaczana jest wartość wyrażenia znajdującego się po prawej stronie instrukcji podstawienia.
3. Wartość wyrażenia jest podstawiana na wszystkie zmienne z wykazu lewych stron.

12 - 4.3. Wczytywanie tablic

Rozszerzymy sposób korzystania z instrukcji input przez uogólnienie, podanej w 10-1.1, definicji parametru wejścia. Obecnie:

< parametr wejścia > ::= < zmienna >|< nazwa tablicy >

O ile w wykazie parametrów wejścia instrukcji input występuje nazwa tablicy, wówczas będzie wczytany i podstawiony na zmienne indeksowane odpowiedni zestaw liczb. Na przykład, użycie następującej instrukcji:

input(a,A[3],B)

gdzie B jest nazwą pięcioelementowej tablicy, spowoduje wczytanie siedmiu liczb i podstawienie ich kolejno na zmienne a, A[3], B[1], B[2], B[3], B[4], B[5].

Jeżeli wczytujemy tablicę dwuwymiarową, wówczas wartości wczytane z taśmy podstawiane są kolejno na elementy pierwszego wiersza, dalej drugiego, potem trzeciego itd. W przypadku tablicy o większej ilości wymiarów realizacja jest analogiczna. Opiszemy ją dokładnie przy pełnym omawianiu procedury input.

12 - 5. Ćwiczenia

10. Wskazać błędy w deklaracjach programu:

```
begin real a[1], a[2], a[3];
array A,B[2 : -2];
integer array Ko12[0:10,5]
integer s,s1,s2,s3;
.......
end
```

13. ETYKIETY. INSTRUKCJE SKOKU. PRZEŁĄCZNIKI

Instrukcje są wykonywane w programie zasadniczo w kolejności ich zapisania /por. 11-1.1/. Możliwa jest jednak zmiana porządku wykonywanych operacji, a mianowicie przez użycie instrukcji skoku, wskazującej explicite następną instrukcję, która ma być realizowana.

Niektóre instrukcje trzeba więc w jakiś sposób oznaczać - służą do tego etykiety.

13 - 1. Etykiety

13 - 1.1. Wstępne informacje

1. Etykietą może być nazwa lub liczba całkowita bez znaku,

np.	Nowe dane	KONIEC	E1	powt
	368	011	0	24500

2. Etykieta z następującym po niej dwukropkiem /symbol podstawowy, por. 2-1.4.1/ może być umieszczona przed dowolną instrukcją, np.:

```
Nowe dane : input(a,b,c)

E1        : alfa := alfa + 1

24500     : c[8] := ab := (sin(x)↑2 × c[7]) : 16
```

3. Etykiety nie mogą być w programie deklarowane.

13 - 1.2. Opis w metajęzyku

label - etykieta

< label > ::= < identifier >|< unsigned number >

13 - 2. Instrukcje skoku

Najprostsza postać instrukcji skoku jest następująca:

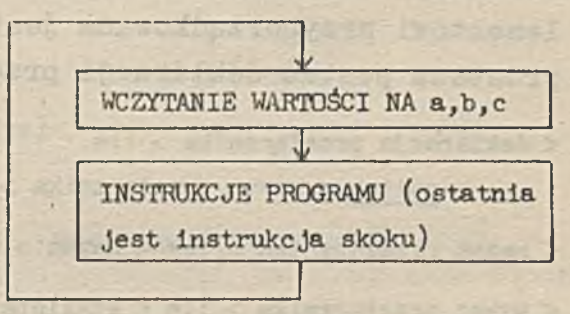
< instrukcja skoku > ::= go to < etykieta >

np. go to Nowe dane
go to E1
go to 24500

go to jest symbolem podstawowym /por. 2-1.4.1 i 2-2/. Następną instrukcją wykonywaną po instrukcji skoku będzie ta, która jest poprzedzona etykietą, wskazaną w instrukcji skoku.

13 - 2.1. Przykład

```
begin real a,b,c;
Nowe dane: input(a,b,c);
           . . . . .
           . . . . .
           go to Nowe dane
end
```



Po wykonaniu instrukcji programu poprzedzających instrukcję skoku, nastąpi powrót do instrukcji przy etykiecie Nowe dane, czyli wczytanie nowych wartości liczbowych na zmienne a, b, c. Dalej ponowna realizacja programu, powrót

do wczytania a, b, c itd. Program w ten sposób zapisany będzie pracował w pętli, którą przerwać może interwencja z zewnątrz /np. zatrzymanie maszyny, brak nowej taśmy z danymi/.

13 - 3. Przełączniki

Określona w poprzednim paragrafie instrukcja skoku wyznacza jednoznacznie następną do wykonania instrukcję. W wielu przypadkach wygodna byłaby możliwość stosowania takiej postaci instrukcji skoku, aby kierowała ona, w zależności od przebiegu obliczeń, do różnych instrukcji/jako następnych do realizowania/. Możliwość tę daje w ALGOLu konstrukcja zwana przełącznikiem.

13 - 3.1. Wstępne informacje

1. Przełącznik jest uporządkowanym zbiorem etykiet.
2. Przełącznik musi być w programie zadeklarowany.

13 - 3.2. Deklaracje przełączników

W deklaracji przełącznika podajemy jego nazwę oraz elementy /etykiety/. Kolejnemu, k-temu od lewej strony elementowi przyporządkowana jest liczba naturalna k. Najprostsza postać deklaracji przełącznika jest następująca:

< deklaracja przełącznika > ::=
switch < nazwa przełącznika > := < wykaz przełącznika >

< nazwa przełącznika > ::= < nazwa >

< wykaz przełącznika > ::= < etykieta >|
< wykaz przełącznika >,< etykieta >

np. KONIEC E1,256,Nowe dane,A,a

switch jest symbolem podstawowym /por. 2-1.4.4/.

13 - 3.2.1. Przykłady

```
switch s:= s1,s2,s3

switch A121 := E1,362,001,powt,A lub B

switch SWITCH := A,B,a,b,5,s1,s2
```

13 - 3.3. Zastosowanie w instrukcji skoku

Oprócz podanej w paragrafie 13-2, instrukcja skoku może mieć jeszcze następującą postać:

< instrukcja skoku > ::= go to < nazewnik przełącznika >

< nazewnik przełącznika > ::=

< nazwa przełącznika >[< wyrażenie arytmetyczne >]

13 - 3.3.1. Przykłady

```
go to s[3]

go to A121[n + 1]

go to SWITCH[(alfa/2-1) + beta]
```

13 - 3.3.2. Realizacja

Wszystkie etykiety wymienione w deklaracji danego przełącznika powinny znajdować się w programie. Przypuśćmy, że jest ich p. Wówczas każdej z nich przyporządkowana jest liczba naturalna k, $1 \leq k \leq p$ /zgodnie z uwagą w punkcie 13-3.2/. Obliczana jest wartość wyrażenia arytmetycznego w nazewniku przełącznika i zaokrąglana do naj-

bliższej liczby całkowitej q. Jeżeli $1 \leq q \leq p$, wówczas następną instrukcją wykonywaną w programie, po instrukcji skoku, jest ta przed którą znajduje się etykieta q-ta z kolei w wykazie przełącznika. Jeśli natomiast $q < 1$ lub $q > p$, wówczas instrukcja skoku jest opuszczona i jako następna wykonywana jest instrukcja zapisana po tej instrukcji skoku.

13 - 3.4. Przykład

Zanalizujemy następujący program:

```
begin integer i;
real a,b,c;
switch s := EL,EM,EN;
        i := 0;
        a := b := c := 0;
A:      i := i + 1;
C:      go to S[i];
        go to B;
EL:     a := a + 0.5;
EM:     b := b + 1;
EN:     c := c + 1.5;
        go to A;
B:      output(⟨+d.dddd10+dd⟩,a × b × c)
end
```

Deklaracje programu kończą się średnikiem po deklaracji przełącznika /czyli po etykiecie EN z wykazu przełącznika/. Instrukcja oznaczona etykietą A zwiększa wartość i o 1, zatem przy pierwszym przebiegu i staje się równe 1. Dlatego też instrukcja skoku oznaczona etykietą C, kieruje nas do instrukcji po etykiecie EL /etykieta ta jest pierwsza na liście przełącznika/. Następują więc kolejne podstawienia wartości a:=0.5, b:=1, c:=1.5 i powracamy do instrukcji przy etykiecie A.

Teraz na zmienną i zostaje podstawiona wartość 2,więc następna instrukcja kieruje nas do etykiety EM. Wartość a zostaje ta sama, zaś b:=2, c:=3 i znów powracamy do instrukcji oznaczonej etykietą A. Zmienna i przyjmuje wartość 3, po czym wykonany jest skok do EN, czyli wartość c zostaje zwiększona do 4.5 i znów powracamy do A. Przy tym przebiegu programu zmienna i ma wartość 4, a więc instrukcja C: go to s [4] jest opuszczona /ponieważ na liście przełącznika są 3 etykiety/.

Przechodzimy więc do następnej instrukcji, która kieruje nas do procedury output. Obliczona jest wartość wyrażenia a × b × c dla aktualnych wartości tych zmiennych /0.5,2,4.5/ i zostaje ostatecznie wyperforowana liczba +4.5000 .

13 - 4. Uwagi

1. Etykiety będące liczbami całkowitymi mogą mieć dodawane nieznaczące zera, tzn. 00217 oznacza tę samą etykietę co 217.
2. Dokładny opis w metajęzyku instrukcji skoku i deklaracji przełącznika podany jest w rozdziale 17.

13 - 4.1. GIER ALGOL

W GIER ALGOLu /podobnie jak w wielu innych reprezentacjach/ etykiety mogą być tylko nazwami.

13 - 5. Ćwiczenia

11. Znaleźć końcowe wartości wszystkich zmiennych występujących w programie:

```
begin real p,q,SUM;
integer n;
switch S := loop, END;
      n := 1;
```

```
        p := 0.5;
        q := 1;
        SUM := 0;
loop:   SUM := SUM + q/n;
        q :=  q × p;
END:    n :=  n + 1;
        go to S[n - 1/2/q + 3 ×(p-q)];
        SUM := SUM × n
    end
```

14. RELACJE. ZMIENNE LOGICZNE

W ALGOLu występuje 6 operatorów relacji: $< \leq = \geq > \neq$ /por. 2-1.4.1/.

14 - 1. Wstępne informacje

1. Za pomocą operatorów relacji można porównywać proste wyrażenia arytmetyczne, tworząc relacje:

a < b	(a mniejsze od b)		
	5 < 4	c ↑ 2 < exp(d) × 2	1 < -1.5
a ≤ b	(a mniejsze lub równe b)		
	1 ≤ 1	(a1/bet) ≤ A[3]	c ≤ 3.14
a = b	(a równe b)		
	4 = 5	abs(c) = ₁₀3 + c/d	x = 0
a ≥ b	(a większe lub równe b)		
	2 ≥ 1	c : d ≥ sin(x) + 3	1₁₀4 ≥ 10 ↑ 4
a > b	(a większe od b)		
	0 > 1.3 - 1.4	A[7] > 13 × alfa1	a1 > 3 × b1
a ≠ b	(a różne od b)		
	32.000 ≠ 2 ↑ 5	d ≠ A1/B1	x ≠ y

2. W zależności od aktualnych wartości zmiennych, relacje mogą być prawdziwe lub fałszywe. Mówimy więc, że relacje mogą przyjmować dwie wartości logiczne: true lub false /prawda lub fałsz/. true i false są symbolami podstawowymi /por. 2-1.3/.
3. Celowe jest więc wprowadzenie nowego typu zmiennych tzw.

zmiennych logicznych. Wartościami zmiennych logicznych są wartości logiczne.

4. Zmienne logiczne deklarujemy w programie, jako zmienne typu Boolean /por. 2-1.4.4 i 2-2/. Zmiennymi tego typu mogą być zarówno zmienne proste jak i indeksowane.

14 - 2. Opis w metajęzyku

relational operator	- operator relacji
relation	- relacja
logical value	- wartość logiczna
true	- prawda
false	- fałsz

< relational operator > ::= <|≤|=|>|≥|≠

< relation > ::= < simple arithmetic expression >
< relational operator > < simple arithmetic expression >

< logical value > ::= true | false

14 - 3. Deklaracje typu

W punkcie 11-2.1. podany był sposób deklarowania zmiennych prostych. Po wprowadzeniu zmiennych logicznych możemy podać pełny opis w metajęzyku deklaracji typu

type list	- wykaz typu
type	- typ
type declaration	- deklaracja typu

< type list > ::= <simple variable>|<simple variable>,<type list>

np. alfa a1, b1, c, delta, Eps

< type > ::= real | integer | Boolean

(uzupełniona została obecnie definicja pojęcia < type >, por. 5-2.1)

< type declaration > ::= < type > < type list > (*)

np. real a,b,c
integer i
Boolean r, log, n

Deklaracja typu informuje, że nazwy wymienione w wykazie reprezentują zmienne proste danego typu. Zmienne zadeklarowane jako rzeczywiste /real/ mogą przybierać wartości liczbowe dodatnie, ujemne lub zero. Zmienne zadeklarowane jako całkowite /integer/ mogą przybierać wartości liczbowe całkowite dodatnie, całkowite ujemne lub zero. Zmienne zadeklarowane jako logiczne /Boolean/ mogą przybierać tylko wartości true lub false.

14 - 4. Uzupełnienie do deklaracji tablic

Ponieważ zmienne indeksowane mogą być też zmiennymi logicznymi, należy odpowiednią tablicę zadeklarować jako tablicę logiczną. Na przykład deklaracja

Boolean array AB [1:15]

zawiera informację, że w programie wystąpi 15 zmiennych boolowskich indeksowanych AB [1], AB [2],..., AB[15].

Deklaracja ta jest poprawna według definicji podanej w 12-2.2, dzięki rozszerzeniu pojęcia < type >.

14 - 5. Uwagi

W ALGOLu występuje symbol podstawowy own, który może być umieszczany przed <type> w deklaracjach. Znaczenie tego symbolu omówimy w rozdziale 19. Symbol own jest stosunkowo rzadko stosowany w programach.

Pełna definicja deklaracji typu i deklaracji tablic wygląda następująco:

< type declaration > ::= <type> < type list>|own <type> <type list>

< array declaration > ::= array < array list >|
< type > array < array list >|
own < type > array < array list >

np.
```
own real a
integer b,c
own Boolean b1,b2,c1,c2,bool
```

```
real array A,B,C,d[1:10]
own integer array I[-3:3]
```

Dla skrócenia zapisu, w publikacji /1/ wprowadzono dodatkową definicję, typu lokalnego lub własnego:

< local or own type > ::= < type > | own < type >

Przy jej użyciu, podane powyżej definicje przybiorą następującą postać:

< type declaration > ::= < local or own type > < type list >

< array declaration > ::= array < array list > |
< local or own type > array < array list >

15. PROSTE WYRAŻENIA BOOLOWSKIE

Obok podanych już operatorów arytmetycznych i relacji, występują w ALGOLu następujące operatory logiczne /por. 2-1.4.1, 2-2.1/:

≡ operator równoważności,
=> operator implikacji,
∨ operator alternatywy,
∧ operator koniunkcji,
¬ operator negacji.

15 - 1. Informacje o stosowaniu operatorów logicznych

Łącząc /według pewnych reguł/ wartości logiczne, zmienne logiczne oraz relacje, otrzymujemy najprostsze przykłady wyrażeń boolowskich.

15 - 1.1. Przykłady

```
b1 ≡ b2
A => c<d
Bool[7] ≡ ¬,b1 ∧ b2 ∨ a=3
B1 ≡ ¬,b1 ∧ b2 ∨ a=3

3>d ∨ a=b => false
X ∧ ¬,(cc ∨ dd)
¬,false ∧ s
3>7 ∨ 7>3 ¬,a=b/2
```

15 - 1.2. Wartość wyrażenia boolowskiego

Znaczenie operatorów logicznych wyjaśniamy poniżej. Tabelka zawiera wartości logiczne wyrażeń boolowskich, w zależności od dwu zmiennych typu Boolean: b1 i b2.

b1	true	true	false	false
b2	true	false	true	false
-,b1	false	false	true	true
b1 ∧ b2	true	false	false	false
b1 ∨ b2	true	true	true	false
b1 => b2	true	false	true	true
b1 = b2	true	false	false	true

Wartość wyrażenia boolowskiego uzyskiwana jest przez wykonanie wskazanych operacji logicznych na elementach posiadających aktualne wartości logiczne. Zasadniczo operacje wykonywane są od strony lewej do prawej, przy czym:

1. o ile występują różne rodzaje operatorów, wówczas respektowane są następujące reguły pierwszeństwa:
   1/ operatory artymetyczne /z zachowaniem ich własnych reguł pierwszeństwa, por. 6-2/.
   2/ operatory relacji /równorzędne między sobą/.
   3/ operatory logiczne z następującymi regułami pierwszeństwa:

   1) -,
   2) ∧
   3) ∨
   4) =>
   5) =

2. mogą występować nawiasy okrągłe - znaczenie ich jest analogiczne, jak w wyrażeniu arytmetycznym /por.6 p.2/.

15 - 2. Proste wyrażenia boolowskie

W poprzednim paragrafie podane były przykłady szczególnej postaci wyrażeń boolowskich, tzw. prostych wyrażeń boolowskich. Przy opisie prostego wyrażenia boolowskiego, analogicznie jak przy opisie prostego wyrażenia arytmetycznego, wprowadzone jest pojęcie wyrażenia pierwotnego. Może nim być wartość logiczna, zmienna logiczna, relacja, nazewnik funkcyjny przyjmujący wartość logiczną /nazewniki takie można wprowadzać w ALGOLu - wrócimy do tego w rozdziałach poświęconych procedurom/, lub dowolne wyrażenie boolowskie, ujęte w okrągłe nawiasy. Łącząc operatorami logicznymi pierwotne wyrażenia boolowskie, otrzymamy proste wyrażenie boolowskie. Wartość jego /true lub false/ wyznaczana jest zgodnie z opisem w 15-1.2. Ogólną postać wyrażenia boolowskiego zdefiniujemy w następnym rozdziale - obecnie poprzestaniemy na informacji, że każde proste wyrażenie boolowskie jest wyrażeniem boolowskim.

15 - 3. Opis w metajęzyku

Boolean primary	- pierwotne wyrażenie boolowskie
Boolean expression	- wyrażenie boolowskie
Boolean secondary	- wtórne wyrażenie boolowskie
Boolean factor	- czynnik boolowski
Boolean term	- składnik boolowski
implication	- implikacja
simple Boolean	- proste wyrażenie boolowskie

<Boolean primary> ::= <logical value>|<variable>|<function designator>|<relation>|(<Boolean expression>)

np. true b1 B[exp(x)+1] a>7

sin(x)/2 = (a-b)/c+$_{10}$-3 (b1 ∨ b2 => b3 ∧ -,A[17])

<Boolean secondary> ::= <Boolean primary>|-,<Boolean primary>

np. b1 (b3 ∧ b4 => b5) -,B[2] -,a>7

< Boolean factor > ::= < Boolean secondary >|
< Boolean factor > ∧ < Boolean secondary >

np. false -,B[2] -,(b1 ∧ -,b2) b1
true ∧ false b1 ∧ b2 b1 ∧ -,B[2] -,b3 ∧ -,b1
-,(a>7 = c<5 ∧ X ∨ d/2=e) ∧ (B[exp(x)] => false)
-,b3 ∧ (a=0 = alfa<5) ∧ -,b2 ∧ (A => c<d)

<Boolean term> ::= <Boolean factor>|<Boolean term> ∨ <Boolean factor>

np. b1 b3 ∧ b4 -,B[2] ∨ b1 ∧ b2
-,b2 ∧ a<b ∨ -,B[3] ∧ c<d ∨ aa ∨ bb

<implication> ::= <Boolean term>|<implication> => <Boolean term>

np. b1
a+b>-5 ∧ z-d<q12 => -,B[2] ∨ b1 ∧ b2

<simple Boolean> ::= <implication>|<simple Boolean> = <implication>

<Boolean expression> ::= <simple Boolean> (æ)

15 - 4. Rozszerzenie definicji instrukcji podstawienia

Przy pomocy instrukcji podstawienia nadawane są aktualne wartości zmiennym. Ponieważ zmienne mogą posiadać wartości liczbowe lub logiczne, naturalne wydaje się następujące rozszerzenie definicji instrukcji podstawienia /por. 9-3/:

<assignment statement> ::= <left part list> <arithmetic expression>|
<left part list> <Boolean expression>

Sposób realizacji został opisany dostatecznie ogólnie w punkcie 12-4.2, przy czym, jeżeli wyrażenie po prawej stronie jest wyrażeniem boolowskim, wówczas zmienne w wykazie lewych stron /proste lub indeksowane/ muszą być typu Boolean.

15 - 4.1. Przykłady

```
b1:= true
b1:= b2:= a>7
B:= p:= AL3[a/2-7]:= q:= a-b≠3 ∨ p ∧ -,q => AL3[2]
V:= Q > Y ∧ Z ∨ P < T
```

15 - 5. Uwagi

Każde pierwotne wyrażenie boolowskie jest prostym wyrażeniem boolowskim.

15 - 6. Ćwiczenia

12. Wyznaczyć wartość zmiennej logicznej boole, kolejno dla a = 2 i b = 5, a = 7 i b = 3, a = -1 i b = 10.

    1) boole := a > b ∧ b ≠ 10
    2) boole := a = b ∨ a - 4 = b
    3) boole := a > 0 ∨ (b > 0 => a ≠ b)
    4) boole := -,((a > 0 => (b ≠ 3)) = ((a < b) ∧ a = 0))

13. Znaleźć końcowe wartości wszystkich zmiennych w następującym programie:

```
begin real ra,rb;
integer ia;
Boolean ba,bb; ra:= 7.5; ia:= 5;
  rb:= 3 × ra - 2 × ia;
  ba:= rb > ia ∧ ia > ra;
  ra:= 2 × (ra - ia) - 1;
  ba:= -,ra > ia ∨ ba;
  bb:= (ba = rb > ia) ∧ ra < rb;
  ba:= -,(ba ∨ bb)
end
```

16. WYRAŻENIA ARYTMETYCZNE. WYRAŻENIA BOOLOWSKIE

Wiemy już, że proste wyrażenie arytmetyczne /boolowskie/ jest wyrażeniem arytmetycznym /boolowskim/. Obecnie podamy pełną definicję wyrażenia arytmetycznego i wyrażenia boolowskiego.

16 - 1. Warunek "jeśli"

16 - 1.1. Opis w metajęzyku

if clause - warunek "jeśli"
if - jeśli
then - to

< if clause > ::= if < Boolean expression > then

(if, then - symbole podstawowe; por. 2-1.4.1)

16 - 1.2. Przykłady

if a = b then (jesli relacja a = b jest prawdziwa, to)

if (b1 ∧ b2) ∧ ¬b3 ∨ c > 0 then

if false then

if a + b > -5 ∧ z - d > q/2 ≡ X = -2 ⇒ b5 then

16 - 1.3. Zastosowanie

Warunek "jeśli" występuje w definicji wyrażeń ALGOL-owych oraz w tzw. instrukcjach warunkowych.

16 - 2. Wyrażenia arytmetyczne

Wyrażenie arytmetyczne w ogólnej postaci składa się z kilku prostych wyrażeń arytmetycznych, rozdzielonych w specjalny sposób wyrażeniami boolowskimi. W dalszej części skryptu stosować będziemy czasami skróty PWA, PWB, WA, WB, oznaczające odpowiednio proste wyrażenie arytmetyczne, proste wyrażenie boolowskie, wyrażenie arytmetyczne, wyrażenie boolowskie.

16 - 2.1. Wstępne informacje

1. W definicji WA wstępuje warunek "jeśli" oraz symbol podstawowy else /por. 2-1.4.1/.
2. Wartość WA równa się wartości jednego z PWA. Wyrażenia boolowskie występujące w WA służą do wyznaczenia tego PWA.

16 - 2.2. Opis w metajęzyku

< arithmetic expression > ::= <simple arithmetic expression>|
<if clause><simple arithmetic expression>else<arithmetic expression>

/por. 8-2/.

16 - 2.3. Przykłady

a/2 + b/2 - sin(a × b) + sqrt(c/d)

```
if a > b then 4 else 5

if a > b then a/2 - b/2 else a/ 2 + b/2

if b1 then sin(x) else cos(x)

if p = 0 then (if q = 0 then 1 else q) else p
```

/Ostatnia konstrukcja jest prawidłowa,gdyż WA ujęte w nawiasy jest PWA/.

```
if alfa/2 = 4 ∧ -,b2 then (if b1 then (if b3 then 3 else 4)
        else sqrt(c)) else if b1 ∧ b2 ∨ b3 then 0 else 327/3

if a < 0 then U+V else if a+b>17 then U/V else if c = 0
        then V/U else 456
```

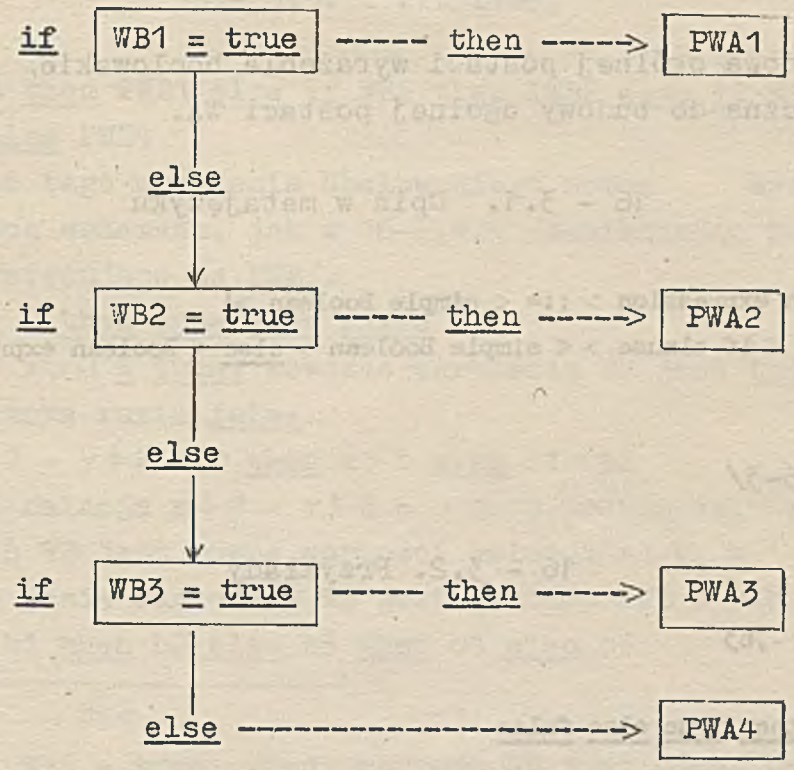
16 - 2.4. Wyznaczanie wartości liczbowej

Wyznaczane są wartości wyrażeń boolowskich w warunkach "jeśli" kolejno od strony lewej do prawej, aż do momentu znalezienia WB o wartości true. Wówczas wartość WA równa jest wartości pierwszego PWA /najdłuższego na tej pozycji/ następującego po tym WB. Jeżeli wszystkie WB są fałszywe, wówczas wartość WA równa jest wartości PWA występującego po ostatnim else.

16 - 2.4.1. Przykłady

1. if WB1 then PWA1 else if WB2 then PWA2 else if WB3 then PWA3 else PWA4

Wyznaczenie wartości tego wyrażenia arytmetycznego możemy przedstawić w formie następującego schematu:



2. if a>b then 4 else 5

Jeżeli a > b = true, wówczas wartością WA jest 4, w przeciwnym razie 5.

3. if b1 then 4×exp(a) else if b2 then 0 else 1

Jeżeli b1 = true, wówczas wartość WA jest równa 4 × exp (a), w przeciwnym razie, wartość WA wynosi 0, gdy b2 = true lub 1, gdy b2 = false.

4. if p=0 then (if q=0 then 1 else q) else p

Następujące zestawienie może ułatwić zrozumienie tego przykładu:

aktualne wartości p i q	wartość WA
p = 0, q = 0	1
p = 0, q = 3	3
p = 2, q = 0	2
p = 3, q = 7	3

16 - 3. Wyrażenia boolowskie

Budowa ogólnej postaci wyrażenia boolowskiej jest analogiczna do budowy ogólnej postaci WA.

16 - 3.1. Opis w metajęzyku

< Boolean expression > ::= < simple Boolean >|
< if clause > < simple Boolean > else < Boolean expression >.

/por. 15-3/

16 - 3.2. Przykłady

b1 ∨ b2 ∨ ¬b3

false

if x > 0 then true else false

if b1 ∧ 3>a ∨ b2 then b1 else if b3 then c = d else b3

if if if b1 then b2 else c = 0 then a>3 else b3 then b4 else b5

Ostatni przykład dla łatwiejszego zrozumienia możemy zapisać z użyciem nawiasów:

if(if(if b1 then b2 else c = 0) then a>3 else b3) then b4 else b5

16 - 3.3. Wyznaczanie wartości logicznej

Wyrażenia boolowskie służą do wyznaczania wartości logicznej. Zasady wyznaczania tej wartości są całkowicie analogiczne do zasad wyznaczania wartości liczbowej, w przypadku wyrażenia arytmetycznego.

16 - 3.3.1. Przykłady

1. if WB1 then PWB1 else if WB2 then PWB2 else if WB3 then PWB3 else PWB4
   Wartość tego wyrażenia boolowskiego można wyznaczyć z pomocą schematu, jak w 16-2.4.1 /zamieniając każde PWA tam występujące na PWB/.
2. if x > 0 then true else false
   Jeżeli x > 0 = true, wówczas wartością WB jest true, w przeciwnym razie false.
3. if x ↑ 2 - y ↑ 2 = 0 then a > b else b1 ∧ b2
   Jeżeli relacja x ↑ 2 - y ↑ 2 = 0 jest prawdziwa, wówczas wartość WB jest równa wartości relacji a > b, w przeciwnym razie wartością WB jest wartość PWB b1 ∧ b2.
4. if if b1 then b2 else b3 then b4 else b5
   (WB1)
   Jeżeli WB1 = true, wtedy wartość WB jest równa wartości zmiennej b4, w przeciwnym razie wartość WB jest równa wartości zmiennej b5.

16 - 4. Uwagi

W wielu zdefiniowanych uprzednio pojęciach występowało wyrażenie arytmetyczne lub boolowskie. Podamy przykłady tych pojęć z ogólną postacią WA lub WB.

1. Proste wyrażenie arytmetyczne:

```
sqrt (if b1 then a + b else a)

4 + (if a = 0 then a + 1 else b)
```

2. Zmienna indeksowana:

```
A[4, 3 × beta, if alfa > 15 then sqrt(alfa) else 0]
```

3. Instrukcja podstawienia:

```
b1:= if b2 then b1 ∨ ¬,b3 = a↑7 else b2 ∧ b5
a:= A1[3]:= if M then entier(c/2) else ln(5)
```

4. Nazewnik przełącznika:

```
S[if ¬,b1 ∧ b2 then a × 1.3 else if b3 then L[2] else Q]
```

5. Warunek "jeśli":

```
if if b1 then b2 else b3 then
```

16 - 5. Ćwiczenia

14. Znaleźć błędy w następującym programie:

```
    begin real a,b,c,d,f,e;
    integer A,B;
    Boolean p,q,r;

    . . . .
E1:   A:= e:= f/2/(if p ∧ ¬,q then 1 else a/3);
E2:   a:= if a > 0 then b + c else d;
E3:   b:= if q then if p then a else b else if p => q then c else d;
E4:   c:= a + if b = c then a else e - f;
E5:   q:= p ∨ if p ∨ q then r ∧ p else q = p ¬,r;
E6:   p:= if p then ¬,p else if q then ¬,q else if if r then ¬,r

          else a > b then p ∧ c < d else B = 0;
      . . . .
    end
```

15. Znaleźć wartości wyrażenia:

if k < 1 then s > w else h ≤ c

dla następujących zestawów wartości zmiennych:

	k	s	w	h	c
1)	-1	2	2	4	3
2)	2	2	2	4	3
3)	1	4	5	2	2

16. Znaleźć wartość wyrażenia

if if if a then b else c then d else f then g else h < k

dla następujących zestawów wartości zmiennych:

	a	b	c	d	f	g	h	k
1)	true	true	true	false	false	false	5	7
2)	false	true	false	false	true	false	5	4
3)	false	false	false	false	false	false	5	4

17. WYRAŻENIA MIANUJĄCE

Wyrażenie mianujące służy do wyznaczania etykiety.

17 - 1. Wstępne informacje

1. Budowa wyrażenia mianującego jest analogiczna do budowy wyrażenia arytmetycznego lub boolowskiego. Wyrażenie mianujące składa się z kilku prostych wyrażeń mianujących rozdzielonych warunkami "jeśli" i symbolami else.
2. Proste wyrażenie mianujące jest to etykieta lub nazewnik przełącznika, lub wyrażenie mianujące ujęte w okrągłe nawiasy.
3. Reguły wyznaczania etykiety są całkowicie analogiczne do reguł wyznaczania wartości liczbowej wyrażenia arytmetycznego.

17 - 2. Opis w metajęzyku

switch identifier	- nazwa przełącznika
switch designator	- nazewnik przełącznika
subscript expression	- wyrażenie wskaźnikowe
simple designational expression	- proste wyrażenie mianujące
designational expression	- wyrażenie mianujące

< switch identifier > ::= < identifier >

< subscript expression > ::= < arithmetic expression >

< switch designator > ::= <switch identifier>[<subscript expression>]

< simple designational expression > ::= <label>|<switch designator>|
(<designational expression>)

< designational expression > ::= <simple designational expression>|

<if clause><simple designational expression>else<designational expression>

17 - 3. Przykłady

Proste wyrażenia mianujące:

L1
1947
SW[n-1 + 7 × b]
(L5)
(if c > 0 then s[5] else s[if d = e then as else hs ↑ 2])
Ar34[if x = y then sqrt(7+c) else K]

Wyrażenia mianujące:

if b1 then L1 else L2

if b2 ∧ b3 then 178 else s[3]

if a = b then STOP else if c > 7 then L1 else s[if b1 then 3 else 4]

17 - 4. Wyznaczanie wartości - etykiety

Wyrażenia boolowskie występujące w wyrażeniu mianującym określają wybór /por. 16-2.4/ pewnego prostego wyrażenia mianującego. Jeśli jest to nazewnik przełącznika, wówczas następuje wyznaczenie etykiety w powiązaniu z odpo-

wiednią deklaracją przełącznika. Każdej etykiecie w dekla - racji przełącznika przyporządkowana jest liczba naturalna k, $1 \leq k \leq p$, gdzie p jest ilością etykiet. Obliczona jest wartość wyrażenia wskaźnikowego i zaokrąglona do najbliż - szej liczby całkowitej q. O ile $1 \leq q \leq p$, wówczas nazew - nik przełącznika wyznacza q-tą z kolei etykietę z wykazu przełącznika. Jeżeli $q < 1$ lub $q > p$, wyrażenie mianujące nie określa żadnej etykiety /por. 13-3.3.2/.

17 - 5. Pełna definicja instrukcji skoku

17 - 5.1. Opis w metajęzyku

go to statement - instrukcja skoku

< go to statement > ::= go to < designational expression >

17 - 5.2. Przykłady

go to L1

go to 845

go to s[3]

go to if a=b then STOP else if c>7 then L1 else s[if b1 then 3 else 4]

17 - 5.3. Uwagi

Jeżeli wyrażenie mianujące nie wyznacza żadnej etykiety, wówczas skok nie następuje i realizuje się kolejna po instrukcji skoku instrukcja programu.

17 - 6. Pełna definicja deklaracji przełącznika

17 - 6.1. Opis w metajęzyku

switch list - wykaz przełącznika
switch declaration - deklaracja przełącznika

< switch list > ::= < designational expression >|
< switch list > , < designational expression >

< switch declaration > ::=switch <switch identifier> := <switch list>

17 - 6.2. Przykłady

switch WKR:= LEWA,PRAWA

switch a12:= L1,if 3>d then L5 else Ab,WKR[b]

17 - 6.3. Uwagi

Elementami wykazu przełącznika w ogólnym przypadku są wyrażenia mianujące /por. 13-2.2/. Wyrażenia te wyznaczają etykiety zgodnie z opisem podanym w paragrafie 17-4.

17 - 7. Wyrażenia w ALGOLu

Możemy obecnie rozszerzyć definicję wartości podaną w paragrafie 5-1. Wartością w ALGOLu jest uporządkowany zbiór liczb /w szczególności jedna liczba/, zbiór wartości logicznych /w szczególności jedna wartość logiczna/ lub etykieta. Wyrażenia w ALGOLu służą do wyznaczania liczby /wyrażenie arytmetyczne/, wartości logicznej /wyrażenie boolowskie/ lub etykiety /wyrażenie mianujące/.

17 - 7.1. Opis w metajęzyku

expression - wyrażenie

< expression > ::= < arithmetic expression >|< Boolean expression >|
< designational expression >

18. INSTRUKCJE WARUNKOWE

Wszystkie instrukcje w ALGOLu można podzielić na 3 grupy: instrukcje bezwarunkowe, instrukcje warunkowe oraz tzw. instrukcje "dla". Do momentu podania dokładnych definicji, przez instrukcję bezwarunkową będziemy rozumieli dowolną instrukcję, która nie jest instrukcją "dla" /omówimy ją w rozdziale 21/, ani instrukcją warunkową. Przykładami instrukcji bezwarunkowej są instrukcje skoku i podstawienia.

18 - 1. Wstępne informacje

1. Instrukcja warunkowa ma złożoną budowę - w jej skład wchodzą warunki "jeśli" /jeden lub więcej/, a zatem wyrażenia boolowskie.
2. W zależności od aktualnych wartości tych wyrażeń, instrukcja warunkowa powoduje wykonanie lub opuszczenie pewnych instrukcji.
3. Szczególną postacią instrukcji warunkowej jest instrukcja "jeśli".

18 - 2. Instrukcja "jeśli"

< instrukcja "jeśli" >::= <warunek "jeśli"> <instrukcja bezwarunkowa>

Jeżeli wartością WB w warunku "jeśli" jest true, wówczas instrukcja bezwarunkowa zostaje wykonana, w przeciwnym razie - opuszczona.

18 - 2.1. Przykłady

```
if b1 then go to FINIS
if a > 2 then A:= 3 × (a-2) + A/2
if (a > b ∨ c < d) ∧ ¬B[3] then b1:= B[1]:= false

if if b1 then b2 else b3 then go to E123
if if if a = 0 then b = 0 else c < 2 then B[1] else B[2] then n:= n+1
```

Dwa ostatnie przykłady mają postać skomplikowaną, ponieważ wyrażenia boolowskie w warunku "jeśli" są znacznie rozbudowane.

18 - 2.2. Uwagi

1. Przykład programu w paragrafie 1-3 powinien być obecnie całkowicie zrozumiały
2. Po warunku "jeśli" następuje instrukcja bezwarunkowa. Instrukcje bezwarunkowe nie zaczynają się symbolem if, zatem bezpośrednio po then nigdy nie można umieścić if .

18 - 3. Instrukcje warunkowe

```
< instrukcja warunkowa >::= < instrukcja "jeśli" >|
        < instrukcja "jeśli" > else < instrukcja >|
            < etykieta >:< instrukcja warunkowa >
```

Jeżeli instrukcja warunkowa ma budowę:

if WB then I1 else I

/gdzie I1 oznacza instrukcję bezwarunkową/, wówczas gdy WB = true wykonywana jest I1, zaś gdy WB = false, wykonywana jest I.

18 - 3.1. Przykłady

```
if a ≥ 0 then b:= sqrt(a) else b:= sqrt(-a)

if b1 then go to E else b1:= true

C: if c = ln(a) then b:= 0 else if b2 then go to BB2

if B[1] then p:= q else if B[2] then p:= q+1 else p:= q+2
```

18 - 3.2. Objaśnienia

Rozważmy konstrukcję ALGOLową, składającą się z instrukcji warunkowej i następującej po niej instrukcji IN. Mamy wówczas dwie możliwości:

if WB1 then I1; IN

lub

if WB1 then I1 else I; IN

Jeżeli instrukcja I jest warunkową, wówczas rozpisując ją otrzymamy:

if WB1 then I1 else if WB2 then I2; IN

lub

if WB1 then I1 else if WB2 then I2 else I'; IN

Postępując dalej analogicznie, wnioskujemy, że każda instrukcja warunkowa musi zakończyć się jedną z dwu wersji:

1. . . . if WB then Ip; IN

lub

2. . . . if WB then Ip else Iq; IN

gdzie wszystkie poprzedzające instrukcje oraz Ip i Iq są bezwarunkowe.

Wykonywanie instrukcji warunkowej odbywa się według następujących zasad:

- wyznaczane są wartości wyrażeń boolowskich, od strony lewej do prawej, aż do znalezienia WB = true. Wykonywana jest następująca bezpośrednio po tym wyrażeniu instrukcja bezwarunkowa. Jeżeli nie wyznacza ona sama następ-

nej instrukcji, wówczas wykonywana jest po niej instrukcja IN.

- Jeżeli wszystkie WB są fałszywe, wtedy w wersji pierwszej wykonywana jest od razu instrukcja IN, zaś w wersji drugiej wykonywana jest instrukcja Iq i jeśli nie wyznacza ona sama następnej, wówczas po niej IN.

18 - 4. Uwagi

1. Dowolne instrukcje, a więc i bezwarunkowe występujące w instrukcji warunkowej, mogą być poprzedzone etykietą lub ciągiem etykiet /i odpowiednio dwukropków/.
2. Wykonanie jakiejkolwiek z instrukcji bezwarunkowych w instrukcji warunkowej powoduje, że następną realizowaną instrukcją jest albo wyznaczona przez tę bezwarunkową, albo następująca po instrukcji warunkowej /wynika to z p. 18-3.2/.
   Jeżeli więc mamy dane instrukcje:
   if a = 0 then E1: c:= 2 else E2: go to A;
   a:= a + 1;
   A: b:= (b↑ 2-3) /(a+0.5)
   wówczas po instrukcji skoku go to E1 zostaną wykonane instrukcje
   c:= 2; a:= a + 1; b:= (b↑ 2-3) /(a+0.5)
   zaś po instrukcji skoku go to E2 zostanie wykonana instrukcja skoku do etykiety A, czyli
   b:= (b↑ 2-3) /(a+0.5)
3. Po instrukcjach bezwarunkowych wewnątrz instrukcji warunkowej nie wolno umieszczać średnika.
4. Instrukcji warunkowej:
   if a≥0 then b:= sqrt (a) else b:= sqrt (-a)
   równoważna jest następująca instrukcja podstawienia:
   b:= if a≥ 0 then sqrt (a) else sqrt (-a)

17. Wykonując kolejno instrukcje, znaleźć końcowe wartości wszystkich zmiennych występujących w programie:

```
1/      begin real a,b;
          a:= 7;
    E:    b:= if a > 10 then 15 + a else 13 - a;
          a:= 17 - b;
          if a > b then go to E
        end

2/      begin real a,b;
        Boolean p;
          a:= 3;
          p:= true;
    A:    b:= a - 2;
          if a/(2-1/a)>0 ∧ b>-2 then a:= 1/a else
          if p then go to Z else go to end;
    Z:    p:= false;
          a:= b + 2 × a;
          go to A;
  end:    p:= a ≥ b
        end
```

/Uwaga: nie należy mylić etykiety end z symbolem podstawowym end/

18. Napisać program, który wyznacza n! dla n naturalnego z przedziału 1÷10. Wartość n wczytujemy z taśmy, wynik perforujemy na taśmie. Jeżeli wczytana wartość nie leży w przedziale 1÷10, należy wyperforować zero.

19. INSTRUKCJE ZŁOŻONE I BLOKI

Programy podawane dotąd w przykładach miały stosunkowo prostą budowę. Wprowadzimy obecnie pojęcia instrukcji złożonej oraz bloku - ułatwią one zapisywanie w ALGOLu bardziej skomplikowanych algorytmów obliczeniowych. Instrukcje złożone i bloki są instrukcjami bezwarunkowymi /por. początek poprzedniego rozdziału/.

19 - 1. Wstępne informacje

1. Instrukcja złożona ma postać ciągu instrukcji ujętego w nawiasy begin i end.
2. Poszczególne instrukcje wewnątrz instrukcji złożonej, oddzielone są średnikami.
3. Po ostatniej instrukcji przed symbolem end średnika nie umieszcza się.
4. Blok, jest to instrukcja złożona, w której po symbolu begin umieszczone są deklaracje rozdzielone średnikami /por. 11-1.1/.
5. Instrukcje wewnątrz instrukcji złożonej lub bloku, mogą być również instrukcjami złożonymi, ewentualnie blokami.
6. Symbol begin można poprzedzić etykietą i dwukropkiem /lub kilkoma etykietami z dwukropkiem po każdej z nich/.
7. Oznaczmy symbolicznie: D - deklaracja, I - instrukcja, E - etykieta. Wówczas instrukcja złożona ma postać:
   E: E: ... begin I; I; ... I; I end
   zaś blok ma postać:
   E: E: ... begin D; D; ... D; I; I; ... I; I end

Instrukcje złożone:

```
1.  begin
      x:= 0;
      if A > B then go to E1 else
      if q then go to E2;
C:    y:= y/2+7
    end
```

```
2.  A:  begin
          p:= q:= true;
          if p1 ∨ s=0 then
    B:    begin
            p1:= false;
            s:= 1;
            go to KONIEC
          end else
          go to nowe dane
        end
```

Bloki:

1. Dowolny z podanych dotychczas przykładowych programów był blokiem.

2.
```
    begin Boolean p; real a;
      p:= true;
L:    begin real b;
        b:= 7;
        if p then a:= b+5 else go to S
      end;
      p:= a=10;
      go to L;
S:    a:= a-3
    end
```

3.
```
    begin real a,b,c;
    integer w;
      w:= 0;  a:= 5;   b:= 3;   c:= 1;
A:    if a < b then
      begin
        w:= w + 1;
        go to if abs(a) > abs(c) then C else D
      end;
```

```
B:  a:= a - b;
    c:= (a - 2)/c;
    if c < a then go to A;
    go to if b > a then E else B;

C:  if b > c then

D:  begin
      c:= b - a;
      go to B
    end;

E:  b:= c + a - b
  end
```

19 - 2. Lokalność nazw i etykiet

Z pojęciem bloku wiąże się bardzo istotne pojęcie lokalności obiektów oznaczanych nazwami oraz etykiet. Ponieważ w ALGOLu utożsamiamy obiekt oznaczony nazwą z tą nazwą, zatem pojęcie lokalności nazwy jest równoważne pojęciu lokalności obiektu, reprezentowanego przez tę nazwę.

1. Etykiety są lokalne dla najmniejszego bloku, wewnątrz którego znajdują się instrukcje, poprzedzone danymi etykietami. Inne obiekty są lokalne dla tego bloku, w którym reprezentujące je nazwy zostały zadeklarowane.
2. Lokalność oznacza, że obiekt nie istnieje na zewnątrz bloku, w którym jest lokalny.
3. Nazwy i etykiety występujące w pewnym bloku i nie będące w nim lokalnymi, nazywamy nielokalnymi. Reprezentują one wewnątrz bloku te same obiekty, co na bezpośrednio wyższym poziomie /tzn. w najmniejszym bloku, zawierającym wewnątrz omawiany przez nas blok/. Oczywiście, jeżeli dla tego wyższego poziomu są również nielokalne ,

wówczas reprezentują te same obiekty również na poziomie wyższym dla tego poziomu itd.

Rozważmy przykład bloku:

```
E1:    begin Boolean p; real a;
         a:= 0;
         . . . . .
E2:      begin integer i,j;
E3:        p:= true;
E4:        i:= a/2 - 16;
           . . . . . .
E5:        begin array A[1:3]; real b;
             A[1]:= A[2]:= A[3]:= if p then j else a + i;
             if p then go to E4;
             . . . . .
E6:        end;
           j:= j + 1;
           if a > 3 then go to E3;
           . . . . . .
E7:    end;
       a:= a + (if p then 3.14 else -1.1);
       . . . . .
E8:  end
```

W bloku /od E1 do E8/ zadeklarowane są zmienne: logiczna p i rzeczywista a. Są one lokalne dla tego bloku i mogą być użyte w dowolnej instrukcji wewnątrz bloku /z wyjątkiem przypadku, kiedy są "zasłonięte" - będzie o tym mowa w punkcie 4/. Jedną z tych instrukcji jest blok poprzedzony etykietą E2, który wprowadza nowe dwie zmienne całkowite i, j. Są one lokalne dla tego bloku, czyli nie mogą występować w instrukcjach przed symbolem begin poprzedzonym E2 i po symbolu end poprzedzonym E7. Zmienne p oraz a są dla niego nielokalne. Wewnątrz tego bloku jest jeszcze jeden blok, deklarujący tablicę A oraz zmienną b. Ostatecznie

więc zmienne indeksowane tej tablicy i zmienna b nie mogą występować na zewnątrz bloku E5 - E6, zmienne i oraz j są wspólne dla obydwu bloków wewnętrznych, zaś zmienne p i a są wspólne dla wszystkich trzech bloków. Etykieta E6 jest lokalna dla bloku najmniejszego /czyli na zewnątrz jego nie może być instrukcji go to E6/. Pozostałe etykiety są dla tego bloku nielokalne. Etykiety E3, E4, E5, E7 są lokalne dla bloku pośredniego, zaś E2 i E8 lokalne dla bloku najbardziej zewnętrznego. Etykieta E1 jest nielokalna dla wszystkich trzech bloków - będzie lokalna dla najmniejszego bloku, obejmującego bloki E1-E8.

Uwaga:

W powyższym przykładzie etykiety E6, E7 i E8 poprzedzają symbol end. Nie jest to sprzeczne z podanymi dotąd informacjami, że etykieta i dwukropek mogą poprzedzać tylko instrukcję /lub etykietę/, ponieważ interpretujemy to w ten sposób, że pomiędzy dwukropkiem a symbolem end występuje tzw. instrukcja pusta, którą szczegółowo opiszemy w rozdziale 20.

Rozważmy teraz przykład bloku, w którym nazwy w instrukcjach używane są nieprawidłowo:

```
        begin real a;
E1:       input(b);
          begin real b;
E2:         a:= 3.5;
E3:         b:= (a/2 + 1)/7
          end;
          a:= a + 1;
E4:       go to E3
        end
```

Powyższy program jest błędny gdyż:

- Instrukcja oznaczona etykietą E1 zawiera nazwę zmiennej b, która w tym miejscu programu "nie istnieje" /gdyż jest lokalna dla bloku wewnętrznego/.

- Instrukcja oznaczona etykietą E4 zawiera nazwę etykiety E3, która też jest lokalna dla bloku wewnętrznego, więc w tym miejscu programu nie istnieje.

4. ALGOL dopuszcza występowanie w jednym programie tych samych nazw, oznaczających różne obiekty. Mogą to być nazwy deklarowane w dwu /lub więcej/ rozłączonych blokach,co nie powoduje żadnych komplikacji w interpretacji programu, np.

```
        begin
        . . . . .
E1:     begin real a;
          . . . . .
E2:     end;
        . . . . .
E3:     begin integer a;
          . . . . .
        end;
        . . . . .
        end
```

Wewnątrz bloku E1-E2 nazwa a oznacza zmienną typu real, zaś wewnątrz bloku E3-E4 zmienną całkowitą. Na zewnątrz tych bloków /przy założeniu, że opuszczone instrukcje nie są blokami deklarującymi nazwę a/ żaden obiekt reprezentowany nazwą a nie istnieje. ALGOL pozwala również na oznaczenie tą samą nazwą dwu /lub więcej/ różnych obiektów,deklarowanych w dwu /lub kilku/ blokach zawartych kolejno jeden w drugim. Czyli nazwa lokalna dla bloku wewnętrznego może występować na zewnątrz tego bloku i reprezentować jakiś inny obiekt. Wówczas jednak obiekt reprezentowany przez tę nazwę na zewnątrz mniejszego bloku, jest wewnątrz tego bloku niedostępny /"zasłonięty"/, np.

```
begin real x,y;
  x:= 1; y:= 2;
  begin real x,z;
    z:= y + 1; x:= 3;
```

```
    output(⟨+d.dddd₁₀+dd⟩,x,y,z)
  end;
  output(⟨+d.dddd₁₀+dd⟩,x,y)
end
```

W programie tym, w bloku wewnętrznym zmienna x z bloku zewnętrznego jest "zasłonięta", zatem zestaw wydrukowanych liczb jest następujący:

+ 3.0000 +2.0000 +3.0000 +1.0000 +2.0000

Wszystkie uwagi tego punktu odnoszą się również do etykiet, przy czym przez etykiety zadeklarowane w danym bloku, należy rozumieć etykiety lokalne dla tego bloku.

5. Przeanalizujemy jeszcze następujący przykład bloku:

```
E1:    begin real A,B,C;
         . . . . .
E2:  P: A:= B + 2 × C;
         . . . . .
E3:      begin real A,D;

E4:  Q:    A:= 2 × B + C;

E5:        D:= 2 + B + A;
           . . . . .
E6:  P:    C:= 2 × A - D;
           . . . . .
E7:        go to P;
           . . . . .
E8:        go to R;
           . . . . .
E9:      end;
         . . . . .
E10: R:  go to P;
         . . . . .

E11:   end
```

W bloku /od E1 do E11/ zadeklarowane są nazwy A, B, C, które tym samym są dla niego lokalne. Blok ten zawiera jako jedną instrukcję, blok wewnętrzny /od E3 do E9/,który wprowadza nowe, lokalne zmienne A oraz D. Zmienna zewnętrzna A jest więc w mniejszym bloku "zasłonięta" i nie ma nic wspólnego ze zmienną lokalną A. Natomiast zmienne B i C są wspólne dla obu bloków. Przy użyciu tych zmiennych podstawiona jest wartość na lokalną zmienną A /przy etykiecie E4/. Lokalne zmienne A i D występują w wyrażeniu arytmetycznym w instrukcji podstawienia na zmienną nielokalną C /przy E6/. Etykiety P i Q przy E6 i E4 są lokalne dla bloku wewnętrznego, zatem są dostępne tylko z wewnątrz tego bloku /natomiast etykieta P przy E2 z wnętrza bloku jest niedostępna/. Instrukcja skoku przy E7 skieruje więc do instrukcji przy E6. Natomiast instrukcja skoku przy E8 spowoduje wyjście z bloku wewnętrznego, gdyż etykieta R nie jest dla tego bloku lokalna. W momencie tym zmienne lokalne A i D przestają "istnieć". Instrukcja skoku przy E10 skieruje do instrukcji przy E2, ponieważ etykieta P przy E6 jest lokalna dla wewnętrznego bloku, a tym samym niedostępna z E10.

19 - 3. Uwagi

1. Z paragrafu 19-1 i poprzednich wynika, że pierwszym symbolem, który następuje po dowolnej instrukcji programu, musi być jeden z trzech niżej podanych:

   ; else end

   /wyjątek stanowi możliwość umieszczania komentarzy po symbolu end, o czym będzie mowa w rozdziale 20/.
2. Wprowadzone w poprzednim paragrafie pojęcie lokalności jest szczególnie użyteczne przy opracowywaniu dużego programu. Można bowiem pisać oddzielnie niezależne jego części, a później połączyć je w całość, bez konieczności uprzedniej zmiany wszystkich nazw, które mogłyby z sobą ewentualnie kolidować. Ponieważ w konkretnej

reprezentacji deklaracja pociąga za sobą rezerwację odpowiedniej ilości miejsc pamięci, podział programu na rozłączne bloki może spowodować bardziej ekonomiczną gospodarkę pamięcią maszyny.

3. Ponieważ deklaracja związana jest z rezerwacją miejsc w pamięci maszyny, więc musimy przestrzegać następującej zasady: jeżeli w deklaracji tablicy wykaz jej par granicznych zależy od pewnych zmiennych, wówczas deklaracja tablicy musi być umieszczona w takim bloku, by zmienne te przyjęły bezpośrednio przed wejściem do tego bloku określone wartości. W szczególności więc, jeżeli deklarujemy tablicę w najbardziej zewnętrznym bloku programu, wyrażenia w wykazie par granicznych mogą zależeć tylko od stałych.
Na przykład z dwu konstrukcji podanych poniżej, występująca po lewej stronie jest nieprawidłowa i powinna być zastąpiona konstrukcją po prawej stronie:

```
begin integer n;
array A[1:n,1:n+1];
  input(A);
. . . . .
end
```

```
begin integer n;
  input(n);
  begin array A[1:n,1:n+1];
    input(A);
    . . . . .
  end
end
```

4. Do bloku wejść można tylko poprzez początkowe begin /ze względu na lokalność etykiet/, ale wyjść dowolnie tzn. przy pomocy instrukcji skoku, lub poprzez końcowe end. Natomiast w instrukcji złożonej etykiety nie są lokalne, zatem do wnętrza instrukcji złożonej można wejść instrukcją skoku do dowolnej etykiety, poprzedzającej którąś z instrukcji w instrukcji złożonej.
5. Wyjaśnimy obecnie w skrócie znaczenie symbolu own /por. 14-5/. Deklaracja typu oraz deklaracja tablicy może być poprzedzona mianem own. Zmienne zadeklarowane w ten

sposób nazywamy własnymi. Zmienne własne po wyjściu z bloku, dla którego są własnymi, nie tracą swojej wartości /w odróżnieniu od pozostałych zmiennych, które po wyjściu z bloku, w którym je deklarowano przestają"istnieć"/. Zatem przy powtórnym wejściu do bloku, zmienne własne mają wartości równe tym, które miały po ostatnim wyjściu z danego bloku. Jednakże z nazw własnych nie można korzystać na zewnątrz bloku, dla którego są własnymi.

W GIER ALGOLu miano own może wystąpić jedynie przy deklaracji typu.

19 - 4. Ćwiczenia

19. Znaleźć końcowe wartości wszystkich zmiennych w blokach podanych w przykładzie drugim i trzecim, p. 19-1.1.
20. Znaleźć końcowe wartości zmiennych zadeklarowanych w bloku zewnętrznym:

```
begin real W,S,B,C;
  W:= 8;  S:= 3;
  B:= 2 × W - S;  C:= B - W;
  begin real P,W;
    W:= B - 2 × C;
    P:= C/2 - B;
AA: W:= P - 2 × W;
    C:= C + 1;
    if W > 1 then go to AA;
    S:= W - P + S
  end;
  W:= W - C + S
end
```

20. INSTRUKCJE PUSTE. KOMENTARZE

W rozdziale tym wprowadzamy dwie nowe konstrukcje języka. Nie mają one bezpośredniego wpływu na przebieg obliczeń, ale mogą ułatwić programowanie oraz zrozumienie i kontrolę algorytmu.

20 - 1. Instrukcje puste

Instrukcja pusta nie powoduje wykonania żadnej operacji. Składa się ona z pustego ciągu symboli.

20 - 1.1. Przykłady

1. begin real a; a:= 5; end

Po instrukcji bezpośrednio przed symbolem end nie umieszcza się średnika. Instrukcją tą jest w podanym przykładzie instrukcja pusta. Powyższy blok ma więc postać:

```
begin  < deklaracja typu >;
       < instrukcja podstawienia >;
       < instrukcja pusta >
end
```

2. begin integer x;; x:= 0 end

Instrukcja pusta występuje między średnikami po deklaracji.

3. begin real a,b;

```
   E:    input (a) ; if a< 0 then go to KONIEC;
```

```
        . . . .
        go to E;
    KONIEC:
    end
```

Instrukcja pusta może być poprzedzona etykietą i dwukropkiem /por. 13-1.1.p 2/. W przykładzie tym instrukcja pusta występuje po dwukropku za etykietą KONIEC.

20 - 1.2. Opis w metajęzyku

empty - puste
dummy statement - instrukcja pusta

< empty > ::=

/tzn. pusty ciąg symboli/

< dummy statement > ::= < empty >

20 - 1.3. Uwagi

Instrukcja pusta służy najczęściej do umieszczania etykiety /analogicznie, jak w trzecim przykładzie paragrafu 20-1.1/.

20 - 2. Komentarze

Bardzo często wygodne jest umieszczanie w programie tekstów objaśniających, tzw. komentarzy. W ALGOLu może się to odbywać według następujących zasad:

W tabelce symbol umieszczony w kolumnie L może być zastąpiony odpowiadającą mu w tej samej linii konstrukcją w kolumnie P, bez żadnego wpływu na realizację programu.

L	P
;	; comment < dowolny ciąg symboli, bez ; >;
begin	begin comment < dowolny ciąg symboli, bez ; >;
end	end < dowolny ciąg symboli bez end lub ; lub else >

Konstrukcja w trzecim wierszu tabelki umożliwia umieszczanie tekstów pomiędzy end i end lub end i; lub end i else.

20 - 2.1. Przykłady

1.
```
a:= a + 4;
comment start nowego cyklu;
V:  alfa:= alfa + 1;
```

2.
```
begin comment program na obliczenie wyznacznika macierzy metoda
      eliminacji Gaussa;
```

3.
```
begin real a;
  . . . . .
  begin
    a:= a/3;
    . . . . .
S:  begin real x,y,z;
      . . . . .
      x:= x + y + z
    end bloku S oraz
  end instrukcji zlozonej;
  output(⟨+d.dddd10+dd⟩, a)
end PROGRAMU
```

1. comment jest symbolem podstawowym, por. 2-1.4.2.
2. Przyczyną częstych błędów programu jest opuszczanie niezbędnych średników. Opuszczenie średnika po end/poza przypadkami, kiedy go tam można lub trzeba nie umieszczać/ spowoduje, że następująca po tym symbolu instrukcja traktowana będzie jako komentarz.
   Na przykład konstrukcja:

   ```
   end; S: a:= b/2 + a; if a < 10 then go to S
   ```

   zapisana błędnie /bez średnika/:

   ```
   end  S: a:= b/2 + a; if a < 10 then go to S
   ```

   równoważna jest konstrukcji:

   ```
   end; if a < 10 then go to S
   ```

3. Należy pamiętać, że możemy ułatwić zrozumienie i kontrolę programu przez wybieranie odpowiednich nazw na etykiety i zmienne oraz odpowiednio stosowanie odstępów i przejść do nowej linii.

21. INSTRUKCJE "DLA"

Instrukcja "dla" powoduje n-krotne /n jest liczbą całkowitą nieujemną/ wykonanie pewnej instrukcji, będącej elementem składowym instrukcji "dla". Ten sam efekt można uzyskać przy pomocy innych konstrukcji ALGOLu /np. przy pomocy instrukcji skoku/, jednakże instrukcja "dla" pozwala bardziej prosto i elegancko zaprogramować pewne algorytmy.

21 - 1. Wstępne informacje

1. W definicji instrukcji "dla" występuje 5 nowych symboli podstawowych: for, step, until, while, do /por. 2-1.4.1,2/
2. Postać instrukcji jest następująca:

   < instrukcja "dla" > ::= for < zmienna > := < wykaz "dla" >
   do < instrukcja >

3. Na zmienną po symbolu for /zwaną zmienną kontrolowaną/ podstawiane są wartości wyrażeń arytmetycznych, zdefiniowanych w wykazie "dla". Po każdorazowym podstawieniu wykonywana jest instrukcja następująca po symbolu do, zwana instrukcją kontrolowaną.
4. Wykaz "dla", którego budowę opiszemy w następnym paragrafie, służy do wyznaczenia pewnego ciągu wyrażeń arytmetycznych. Instrukcja kontrolowana jest wykonywana tyle razy, ile wyrażeń arytmetycznych wystąpi w tym ciągu. Jeżeli wykaz nie wyznacza żadnego wyrażenia, instrukcja kontrolowana nie zostaje wykonana ani razu.

5. Instrukcją kontrolowaną może być również instrukcja "dla".

21 - 2. Wykaz "dla"

Wykaz "dla" zawiera jeden element lub kilka elementów rozdzielonych przecinkami. Każdy element może być wyrażeniem arytmetycznym lub konstrukcją postaci step-until lub konstrukcją postaci while. Rozważymy początkowo instrukcje "dla", w których w wykazie "dla" wszystkie elementy są tej samej postaci.

21 - 2.1. Element - wyrażenie arytmetyczne

Element tej postaci jest wyrażeniem arytmetycznym. Na zmienną kontrolowaną podstawiane są kolejno, od lewej strona do prawej, wartości elementów i każdorazowo wykonywana jest instrukcja kontrolowana.

21 - 2.1.1. Przykłady

1. `for p:= 3 do s:= p ↑ 2`

Instrukcję tę możemy zastąpić dwoma podstawieniami:

```
p:= 3;   s:= p ↑ 2
```

2. `for i:= -3, a, 8/2 + 1 do a:= a + i`

Instrukcja równoważna jest następującemu ciągowi instrukcji podstawienia:

```
i:= -3;  a:= a + i;  i:= a;  a:= a + i;  i:= 8/2 + 1;  a:= a + i
```

Jeżeli np. wartością początkową a było 5, wówczas wartością końcową a jest 69.

3. for i:= 1,2,3,4,5,6,7,8,9,10,11,12,13,14 do

```
A[i]:= B[i] + C[i]
```

Instrukcja spowoduje dodanie odpowiednich elementów dwóch tablic.

4. for i:= 1,2,3,4,5,6,7,8,9,10 do

```
for j:= 1,2,3,4,5,6,7 do   A[i,j]:= B[i,j] + C[i,j]
```

Instrukcja spowoduje dodanie dwóch macierzy prostokątnych o wymiarach 10 × 7 /dodawanie odbywać się będzie kolejno, wiersz po wierszu/.

5.
```
for x:= x1,x2,x3,x4 do
begin
  i:= n;   M:= a[n];
E: i:= i - 1;  M:= M × x + a[i];
  if i > 0 then go to E;
  output(‡+d.dddd_n+dd‡,M)
end
```

Instrukcja spowoduje wyznaczenie i wyprowadzenie na zewnątrz wartości wielomianu

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

dla $x = x_1, x_2, x_3, x_4$.

Problemy rozwiązane w trzech ostatnich przykładach będą zapisane w dalszych punktach, za pomocą innych postaci elementów wykazu "dla".

21 - 2.1.2. Uwagi

Zauważmy ogólnie, że gdy wykaz "dla" zawiera n elementów, wówczas instrukcja:

for z := E1, E2, ... , En do I

równoważna jest następującemu ciągowi instrukcji:

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

gdzie z oznacza zmienną kontrolowaną, I - instrukcję kontrolowaną, zaś E1, ..., En - n elementów wykazu"dla".

21 - 2.2. Element postaci step - until

Element tej postaci ma następującą budowę:

WA1 step WA2 until WA3 gdzie

WA1, WA2, WA3 są wyrażeniami arytmetycznymi.

Załóżmy, że wykaz "dla" składa się tylko z jednego elementu tego typu. Wówczas instrukcję:

```
for z := WA1 step WA2 until WA3 do I
```

możemy odczytać: "dla z zmieniającego się od WA1 z krokiem WA2 aż do WA3 wykonuj instrukcję I".

Instrukcję tę opisuje następująca konstrukcja:

```
      z:= WA1;
POWT: if sign(WA2) × (z-WA3) ≤ 0 then
      begin
        I;  z:= z + WA2;  go to POWT
      end
```

Zauważmy, że jeden element tego typu w wykazie "dla" może powodować wielokrotne wykonywanie instrukcji I. Jeżeli wykaz składa się z kilku takich elementów /rozdzielonych przecinkami/, wówczas realizacja instrukcji odbywa się zgodnie z uwagą podaną w punkcie 21-2.1.2. Zauważmy również, że w przypadku, gdy sign (WA2) × (WA1-WA3) > 0 w momencie, gdy przechodzimy do wykonania instrukcji "dla", element step-until nie określa żadnego wyrażenia i instrukcja kontrolowana I nie zostaje ani razu wykonana.

1. `for i:= 1 step 1 until 14 do A[i]:= B[i] + C[i]`

Instrukcję tę możemy zastąpić następującymi instrukcjami:

```
        i:= 1;
POWT:   if i-14 ≤ 0 then
        begin
          A[i]:= B[i] +C[i];  i:= i + 1;
          go to POWT
        end
```

2.
```
for i:= 1 step 1 until 10 do
for j:= 1 step 1 until  7 do
    A[i,j]:= B[i,j] + C[i,j]
```
/por. 21-2.1.1. przykład 4/

3.
```
for x:= x1,x2,x3,x4 do
begin
  M:= 0;
  for i:= n step -1 until 0 do
  M:= M × x + a[i];
  output(‡+d.dddd₁₀+dd‡,M)
end
```
/por. 21-2.1.1. przykład 5/

4.
```
for i:= 0 step 0.4 until 1, -2 step 0.9 until 0 do
a:= a × i + b/2 - c
```

Instrukcja ta równoważna jest instrukcji:

```
for i:= 0, 0.4, 0.8, -2, -1.1, -0.2 do  a:= a × i + b/2 - c
```

5.
```
for i:= if a > 0 then 2 else 3 step i/2 + 5 until b+j do
    j:= i - 5
```

6.
```
for z:= 3 step 1 until 2 do
p:= p/2/z
```

Ponieważ sign (1) × (3 - 2) > 0, instrukcja kontro - lowana nie zostanie wykonana ani razu.

21 - 2.3. Element postaci while

Element tej postaci ma następującą budowę:

WA while WB gdzie

WA oznacza wyrażenie arytmetyczne, a WB - boolowskie. Zakładając analogicznie, jak w 21-2.2, że wykaz "dla"składa się z jednego elementu tej postaci, instrukcję:

for z:= WA while WB do I

należy odczytać:

"dla zmiennej z przyjmującej wartość wyrażenia WA dopóki prawdziwe jest WB, wykonuj instrukcję I".

Instrukcję tę możemy opisać następująco:

```
      z:= WA;
POWT: if WB then
      begin
        I; go to POWT
      end
```

Wykaz może składać się z kilku elementów - realizacja instrukcji odbywa się według zasad podanych w 21-2.1.2.

21 - 2.3.1. Przykłady

1.
```
for x:= 3 while z > 0 do
begin
  z:= z - 1.5;
  a:= x/2 + a + z
end
```

2.
```
for a:= a/2 while a < 22 ∧ x < 0 do  x:= x + a
```

3.
```
for x:= x1, x2, x3, x4 do
begin
  M:= a[n];  i:= n;
```

```
    for i:= i - 1 while i ≥ 0 do M:= M × x + a[i];
    output(⟨+d.dddd10+dd⟩,M)
  end
```

4.
```
  for i:= a/2 while b > 0 do
  begin
    a:= a - 1;
    b:= b - c × i
  end
```

21 - 3. Opis w metajęzyku

for list element	- element wykazu "dla"
for list	- wykaz "dla"
for clause	- warunek "dla"
for statement	- instrukcja "dla"
statement	- instrukcja

< for list element > ::= < arithmetic expression >|
< arithmetic expression > step < arithmetic expression > until
< arithmetic expression >|
< arithmetic expression > while < Boolean expression >

< for list > ::= <for list element>|<for list>,<for list element>

< for clause > ::= for < variable > := < for list > do

< for statement > ::= < for clause > < statement >|
< label > : < for statement >

21 - 4. Uwagi

1. W postaci ogólnej wykazu "dla" elementy wykazu mogą być różnej postaci w jednym wykazie, np.
for i:= -7, 2 step 1 until 6, 8, 0 while a=b, 16 do I
Wówczas zmiennej kontrolowanej nadaje się wartości otrzymane przez kolejny przegląd, od lewej, do prawej, elementów wykazu "dla" w/g reguł podanych w 21-2 i wykonuje instrukcje I. Zatem instrukcja ta jest równoważ-

na następującemu ciągowi instrukcji:

```
for i:= -7 do I;
for i:= 2 step 1 until 6 do I;
for i:= 8 do I;
for i:= 0 while a = b do I;
for i:= 16 do I
```

2. Po wyjściu z instrukcji "dla" poprzez instrukcję skoku /zakładając, że instrukcja kontrolowana zawiera instrukcję skoku/ wartość zmiennej kontrolowanej będzie równa tej, którą miała ona bezpośrednio przed instrukcją skoku. Jeżeli natomiast wyjście z instrukcji "dla" nastąpiło wskutek wyczerpania wykazu "dla", zmienna kontrolowana po wyjściu jest nieokreślona.
3. Wynik działania instrukcji skoku z zewnątrz instrukcji "dla" kierującej do etykiety wewnątrz instrukcji "dla" nie jest określony.

21 - 4.1. GIER ALGOL

1. W GIER ALGOLu III zmienna kontrolowana jest określona również po wyjściu z instrukcji kontrolowanej wskutek wyczerpania wykazu "dla". Jeżeli ostatni element wykazu "dla" był wyrażeniem arytmetycznym, wówczas zmienna kontrolowana ma po zakończeniu instrukcji wartość tego wyrażenia. Jeżeli ostatni element wykazu jest elementem postaci step-until lub postaci while, wówczas wartością zmiennej kontrolowanej po zakończeniu instrukcji, jest wartość pierwszego WA nie spełniającego testu /występującego w warunku "jeśli"/.
2. Jeżeli zmienną kontrolowaną jest zmienna indeksowana, wówczas wartość jej wskaźników obliczana jest tylko raz, przy pierwszym podstawieniu.

21 - 5. Ćwiczenia

21. Znaleźć kolejne wartości, które będą przyjmowały zmienne i oraz Q, podczas realizacji następującej instruk-

cji "dla" /przy wartościach początkowych: Q = 0, b= 0, p= true/:

```
for i:= -4, 3, i-1 while Q > -40, 0, 400 step 10 until 430,
        2 while b < 2↑3 ∨ -,p do
begin
  b:= b × 2 + 1;
  p:= -,p;
  Q:= Q + i - b
end
```

22. Znaleźć wartości zmiennej s po każdej instrukcji "dla", w programie:

```
begin real p,q,r,s;
integer k,m;
  p:= 1;  q:= 2;  r:= 3;  s:= 0;
  for k:= p + q, q - p, r × p - q do  s:= s + k;
  for k:= q step r until 7 × q + 1 do  s:= s - k;
  for k:= 2,s, 2 step 2 until 6 do  s:= s + 2 × k;
  for k:= s + 45, k + 2 while s < 0 do  s:= s - k;
  for k:= 1 step 1 until 5 do
      for m:= 3 step -1 until 0 do  s:= s + k + m
end
```

23. Napisać program na wyznaczenie iloczynu skalarnego wektorów V1 [1:p] i V2 [1:p] , wartość na p i elementy wektorów wczytujemy z taśmy, wynik perforujemy na taśmie. /Należy pamiętać o uwadze 3 w paragrafie 19-3/.

22. ŚCISŁE DEFINICJE I KLASYFIKACJA WPROWADZONYCH INSTRUKCJI

Uściślimy i uzupełnimy obecnie wprowadzone w rozdziałach 18–20 definicje różnych instrukcji. Ponieważ chcemy podać pełną klasyfikację, w opisach metajęzykowych wystąpi nie znana nam jeszcze instrukcja procedury – opiszemy ją dokładnie w następnych rozdziałach.

22 – 1. Instrukcje podstawowe

Instrukcjami podstawowymi są instrukcje: podstawienia, skoku, pusta oraz instrukcja procedury. Instrukcja podstawowa poprzedzona etykietą /i dwukropkiem/ jest nadal instrukcją podstawową.

22 – 1.1. Opis w metajęzyku

unlabelled basic statement – instrukcja podstawowa bez etykiety
procedure statement – instrukcja procedury
basic statement – instrukcja podstawowa

< unlabelled basic statement > ::= < assignment statement >|
< go to statement >|< dummy statement >|
< procedure statement >

< basic statement > ::= < unlabelled basic statement >|
< label > : < basic statement >

22 - 2. Rodzaje instrukcji. Program

Informacje o rodzajach instrukcji podaliśmy na początku rozdziału 18.

Program jest blokiem lub instrukcją złożoną, przy czym nie może być zawarty wewnątrz innej instrukcji i nie może korzystać z instrukcji, nie zawartych w nim.

22 - 2.1. Opis w metajęzyku

unconditional statement	- instrukcja bezwarunkowa
conditional statement	- instrukcja warunkowa
compound statement	- instrukcja złożona
block	- blok
program	- program

< statement > ::= < for statement >|< conditional statement >|
< unconditional statement >

< uncoditional statement > ::= < basic statement >|< block >|
< compound statement >

< program > ::= < block >|< compound statement >

22 - 2.2. Przykłady

Wszystkie programy uprzednio podawane w przykładach miały postać bloku. Następująca instrukcja złożona jest programem:

```
begin
  output(‡+d.dddd₁₀+dd‡,(3 + exp(1.5))↑2-7)
end
```

22 - 3. Instrukcje warunkowe

Instrukcje te opisaliśmy obszernie w rozdziale 18. Obecnie, gdy poznaliśmy definicję instrukcji "dla" możemy podać jeszcze jedną postać instrukcji warunkowej:

<instrukcja warunkowa>::= < warunek "jeśli" > < instrukcja "dla" >

np. if b1 then for s:= 0 step 0.1 until 1.2 do A:= A × s

W związku z tym, do objaśnień w p.18-3.2. należy wnieść uzupełnienie - instrukcja Ip w wersji 1 oraz instrukcja Iq w wersji 2 może być również instrukcją "dla" /natomiast instrukcja Ip w wersji 2 może być tylko instrukcją bezwarunkową/.

22 - 3.1. Opis w metajęzyku

if statement - instrukcja "jeśli"

< if statement > ::= < if clause > < uncoditional statement >

< conditional statement > ::= < if statement >|

< if statement > else < statement >|

< if clause > < for statement >|

< label > : < conditional statement >

22 - 4. Instrukcje złożone i bloki

Pojęcia te opisaliśmy szczegółowo w rozdziale 19. Obecnie zdefiniujemy je tylko w metajęzyku.

22 - 4.1. Opis w metajęzyku

compound tail - zakończenie instrukcji złożonej

block head - początek bloku

declaration - deklaracja
unlabelled compound - instrukcja złożona bez etykiety
unlabelled block - blok bez etykiety

<compound tail> ::= <statement> end|<statement> ; <compound tail>

np. a:=3 end go to P; r:=3; q:=false end

<block head> ::= begin <declaration>|<block head> ; <declaration>

np. begin real a begin real a; integer b
begin real a; integer b,c; Boolean B1,B2

< unlabelled compound > ::= begin < compound tail >

< unlabelled block > ::= < unlabelled compound >|
< label > : < compound statement >

< block > ::= < unlabelled block >|< label > : < block >

23. PROCEDURY - WPROWADZENIE

Rozpoczynamy obecnie opisywanie jednego z najtrudniejszych pojęć ALGOLu - procedury. Celem tego rozdziału jest podanie informacji, które mogą pomóc w zrozumieniu dalszej części skryptu - z tego też powodu struktura rozdziału różni się od struktury rozdziałów poprzednich.

Załóżmy, że programujemy w ALGOLu algorytm, który przewiduje wykonanie dodawań dwu wektorów, w różnych miejscach programu.

```
      begin array A,A1,B,B1,C[1:10]; integer i,...;
        . . . . .
E1:     for i:= 1 step 1 until 10 do C[i]:= A[i] + B[i];
        . . . . .
E2:     for i:= 1 step 1 until 10 do A[i]:= A1[i] + B1[i];
        . . . . .
E3:     for i:= 1 step 1 until 10 do C[i]:= C[i] + A[i];
        . . . . .
        begin array K,L,M[1:15],S[1:10];
          . . . . .
E4:       for i:= 1 step 1 until 10 do S[i]:= C[i] + B[i];
          . . . . .
E5:       for i:= 1 step 1 until 15 do K[i]:= L[i] + M[i];
          . . . . .
E6:       for i:= 1 step 1 until 10 do C[i]:= S[i] + M[i];
          . . . . .
        end;
```

```
. . . . .
E7:  for i:= 1 step 1 until 10 do A1[i]:= B1[i] + C[i];
. . . . .
end programu
```

Posługując się omówionymi dotychczas konstrukcjami ALGOLu, musimy wypisać explicite 7 instrukcji "dla", różniących się jedynie nazwami wektorów, występujących w nich i ewentualnie wymiarami tych wektorów.

Program możemy w sposób istotny uprościć, korzystając z konstrukcji ALGOLu, zwanej procedurą.

Idea procedur jest następująca:

1. Deklarujemy procedurę w odpowiednim bloku. W deklaracji umieszczamy algorytm zapisany w ALGOLu lub w innym języku, zwany treścią procedury, łącząc go w pewien sposób z nazwą deklarowanej procedury.
2. We wszystkich miejscach programu, w których treść procedury ma być wykonana, wywołujemy procedurę przy pomocy instrukcji procedury. Instrukcja procedury składa się z nazwy procedury i umieszczonej po niej w nawiasach listy konkretnych elementów, na których algorytm ma być wykonany /elementy te nazywamy parametrami aktualnymi/.
3. W treści procedury specjalną rolę odgrywają tzw. parametry formalne. Są to nazwy służące do zapisania ogólnej postaci algorytmu, które przy wywołaniu procedury zastępowane są parametrami aktualnymi.

Pomińmy obecnie sposób zadeklarowania procedury na dodawanie dwu wektorów, zakładając jedynie, że nazwą procedury jest DODWEK, zaś w nawiasach po niej musimy umieścić wymiar wektorów i 3 nazwy wektorów, z których pierwszy ma być sumą dwu pozostałych. Wektory te i ich wymiar są parametrami aktualnymi - zastąpią one odpowiadające im w treści procedury parametry formalne. Przy użyciu takiej procedury program podany na początku rozdziału możemy zastąpić programem o następującej postaci:

```
    begin array A,A1,B,B1,C[1:10]; integer i,...;

        < deklaracja procedury DODWEK >;
          . . . . .
E1:     DODWEK(10,C,A,B);
          . . . . .
E2:     DODWEK(10,A,A1,B1);
          . . . . .
E3:     DODWEK(10,C,C,A);
          . . . . .

          begin array K,L,M[1:15],S[1:10];
            . . . . .
E4:         DODWEK(10,S,C,B);
            . . . . .
E5:         DODWEK(15,K,L,M);
            . . . . .
E6:         DODWEK(10,C,S,M);
            . . . . .
          end;

          . . . . .
E7:     DODWEK(10,A1,B1,C);
          . . . . .
    end programu
```

W przypadku, gdyby treść procedury była bardziej rozbudowanym algorytmem /np. wyznaczanie wyznacznika macierzy, całkowanie funkcji, znajdowanie pierwiastków wielomianu itd./, uproszczenie programu byłoby bardziej widoczne.

Z pojęciem procedur zetknęliśmy się już w rozdziałach 7 i 10, przy omawianiu funkcji standardowych i procedur wejścia-wyjścia. Są one przykładami procedur GIER ALGOLu, z tym jednak, że nie trzeba ich w programie deklarować. Deklaracje funkcji standardowych i procedur wejścia-wyjścia są umieszczone na stałe w translatorze /programie tłumaczącym z konkretnej reprezentacji na język wewnętrzny maszyny/. Z informacji podanych o tych procedurach można wysnuć następujące wnioski:

1. Treść procedury niekiedy nie może być wyrażona tylko w ALGOLu. Na przykład w treści procedury input muszą występować instrukcje języka wewnętrznego maszyny, które powodują uruchomienie czytnika i przesłanie liczb wyperforowanych na taśmie papierowej do urządzenia pamięci.
2. Procedury mogą być dwojakiego rodzaju:
   - nazwa procedury z listą parametrów aktualnych służy tylko do uruchomienia treści procedury /np. input output/
   - nazwa procedury z listą parametrów aktualnych jest nazewnikiem funkcyjnym, tzn. spełnia rolę podaną powyżej, ale dodatkowo przyjmuje po wywołaniu procedury wartość logiczną lub liczbową /np. dowolna z funkcji standardowych/.

Ten drugi rodzaj procedur nazywać będziemy procedurami funkcyjnymi. O nazewnikach funkcyjnych, definiowanych przez procedury funkcyjne, wspominaliśmy omawiając wyrażenia arytmetyczne i boolowskie. Procedury funkcyjne wywoływać możemy instrukcją procedury lub nazewnikiem funkcyjnym.

24. DEKLARACJE PROCEDUR

Procedury omówimy w trzech etapach, poświęcając uwagę kolejno: deklarowaniu procedur, wywoływaniu procedur oraz specyficznym cechom procedur.

24 - 1. Wstępne informacje

1. Deklarację procedury umieszczamy na początku bloku, oddzielając ją średnikiem od pozostałych deklaracji bloku lub od instrukcji.
2. Zakresem działania procedury jest blok, na początku którego została zadeklarowana. Oznacza to, że wywołanie tej procedury /instrukcją lub nazewnikiem/ może mieć miejsce tylko wewnątrz tego bloku.
3. Deklaracja procedury ma następującą budowę:

< deklaracja procedury > ::=
    procedure < nagłówek procedury > < treść procedury > |
    < typ > procedure < nagłówek procedury > < treść procedury >

/procedure - symbol podstawowy, por. 2-1.4/

4. Symbol procedure poprzedza się deklaratorem typu tylko w deklaracjach procedur funkcyjnych /por. zakończenie poprzedniego rozdziału/.

5. Sama deklaracja procedury nie powoduje wykonania żadnych operacji.

24 - 2. Treść procedury

Najważniejszą częścią deklaracji procedury jest treść procedury, będąca instrukcją lub kodem. Przez kod rozumiemy algorytm zapisany częściowo lub całkowicie nie w ALGOLu. W języku wzorcowym nie daje się żadnych reguł odnośnie języka kodu, przyjmując, że jest to zagadnienie konkretnej reprezentacji. Zatem w rozdziałach poświęconych procedurom będziemy zakładać, że treść procedury jest instrukcją ALGOLuwą. Najczęściej instrukcja ta jest instrukcją złożoną lub blokiem.

24 - 2.1. Nazwy w treści procedury

Nazwy występujące w treści procedury dzielą się na parametry formalne, nazwy lokalne i nazwy nielokalne.

1. Parametry formalne są to nazwy wymienione w nagłówku procedury. Zgodnie z tym, co podawaliśmy w poprzednim rozdziale, służą one do zapisywania ogólnej postaci instrukcji i wyrażeń w treści procedury. Obiekty reprezentowane tymi nazwami nie istnieją. Przy wywoływaniu procedury, parametry formalne zastępuje się parametrami aktualnymi /lub podstawia się na nie wartości parametrów aktualnych/.
2. Nazwy lokalne są to nazwy zadeklarowane w treści procedury. Dodatkowo, ponieważ treść procedury działa zawsze jak blok /nawet, gdy nie ma jego budowy/, etykiety poprzedzające instrukcje w treści procedury, lub samą treść procedury, są również lokalne. Obiekty reprezentowane tymi nazwami nie istnieją na zewnątrz treści procedury - zatem nazwy lokalne spełniają rolę pomocniczą.
3. Nazwy, które nie są parametrami formalnymi i nie są lokalne, są nielokalne. Muszą być one zadeklarowane na

początku tego samego bloku co procedura, lub na poziomie wyższym, tzn. w bloku zawierającym blok z zadeklarowaną procedurą. Przy wywoływaniu procedury oznaczają one te same obiekty w treści procedury, co na początku bloku, w którym procedurę deklarujemy.

24 - 3. Nagłówek procedury

Nagłówek procedury ma następującą budowę:

< nagłówek procedury > ::= < nazwa procedury > < zbiór parametrów formalnych >;< zbiór wartości > < zbiór specyfikacji >

Nagłówek procedury jest więc rozdzielony średnikiem na dwie części. Omówimy je oddzielnie.

24 - 3.1. Pierwsza część nagłówka

Parametrem formalnym jest nazwa. Zbiór parametrów formalnych zawiera ujęty w okrągłe nawiasy ciąg parametrów formalnych, oddzielonych od siebie /gdy parametrów jest więcej niż jeden/ przecinkami. W przypadku procedur nie mających parametrów formalnych, część pierwsza nagłówka składa się tylko z nazwy procedury /rozpatrzymy to w rozdziale 26/. Oznaczając dla skrótu f1, f2, ..., fn - parametry formalne i N-nazwa procedury, pierwszą część nagłówka procedury możemy przedstawić następująco:

N (f1, f2,...,fn)

Przy wywoływaniu procedury, każdy z parametrów formalnych fi zastępujemy odpowiednim parametrem aktualnym ai. Pierwsza część nagłówka procedury podaje więc informacje niezbędne przy wywoływaniu danej procedury - jaka jest nazwa tej procedury oraz ile /i w jakim porządku/ parametrów aktualnych należy umieścić w instrukcji procedury.

24 - 3.2. Druga część nagłówka

Część ta składa się ze zbioru wartości i zbioru specyfikacji. Omówimy je w odwrotnej kolejności.

24 - 3.2.1. Zbiór specyfikacji

Ponieważ parametry formalne będą zastępowane aktualnymi, nie należy ich deklarować. Z drugiej jednak strony mogą być potrzebne informacje o rodzaju i typie obiektów, które parametry formalne reprezentują w treści procedury /w przypadku konkretnej reprezentacji informacje te mogą być niezbędne do poprawnego przetłumaczenia programu na język wewnętrzny/. W tym celu w nagłówku procedury może być umieszczony zbiór specyfikacji. Specyfikacje mają postać zbliżoną do deklaracji, tzn. po odpowiednim symbolu podstawowym /np. real, array, Boolean/ wymienione są nazwy tych parametrów formalnych, które repezentują dany obiekt. Zachodzą jednak pewne istotne różnice między specyfikacjami a deklaracjami:

1. Specyfikacje nie lokalizują nazw ani /w przypadku konkretnej reprezentacji/ nie rezerwują miejsc w pamięci - dlatego też nie podaje się pełnych informacji o obiekcie, ale jedynie wymienia nazwę /dotyczy to tablic, przełączników i procedur/.
2. Etykiety będące parametrami formalnymi możemy specyfikować /umieszczając ich wykaz za symbolem podstawowym label/.
3. W specyfikacjach nie może wystąpić symbol own /w deklaracjach możemy go umieszczać przed deklaracją typu lub deklaracją tablic, por. 14-5/.
4. Parametrem formalnym może być również nazwa reprezentująca tzw. łańcuch /string/, o którym będziemy mówili w rozdziale 26. Łańcuchy specyfikujemy umieszczając za symbolem podstawowym string /por. 2-1.4.5/ odpowiedni wykaz nazw.

Do cech wspólnych deklaracji i specyfikacji zaliczyć możemy:

1. Kolejność deklaracji na początku bloku i specyfikacji w nagłówku, jest dowolna.
2. Ta sama nazwa nie może być zadeklarowana dwa lub więcej razy na początku jednego bloku i również żaden parametr formalny nie może pojawić się więcej niż raz w zbiorze specyfikacji.

24 - 3.2.2. Zbiór wartości

Zbiór specyfikacji może być poprzedzony tzw. zbiorem wartości. Zbiór wartości składa się z symbolu podstawowego value /por. 2-1.4.5/, po którym umieszczamy jeden lub więcej parametrów formalnych. Od tego, czy dany parametr jest w zbiorze wartości, zależy sposób wykonywania instrukcji procedury oraz nazewnika funkcyjnego - omówimy do dokładnie w następnym rozdziale.

Parametry formalne wymienione w zbiorze wartości muszą być wymienione również w zbiorze specyfikacji. Pozostałe parametry formalne mogą być nie specyfikowane.

Nie wszystkie parametry formalne wolno wymienić w zbiorze wartości - mogą to być jedynie parametry specyfikowane jako integer, real, Boolean, array, label.

24 - 4. Wartość nazewnika funkcyjnego

Procedury funkcyjne określają wartość odpowiedniego nazewnika funkcyjnego. Oznacza to, że ich nazwa z listą parametrów aktualnych posiada wartość liczbową lub logiczną i tym samym może występować w wyrażeniach arytmetycznych lub boolowskich.

Na to, by deklaracja procedury funkcyjnej mogła definiować wartość nazewnika funkcyjnego, w treści procedury musi występować co najmniej jedna instrukcja podstawienia, z nazwą procedury po jej lewej stronie. Ostatnia wartość

nadana taką instrukcją podstawienia jest używana jako wartość nazewnika funkcyjnego przy obliczaniu wartości wyrażenia, w którym ten nazewnik występuje.

24 - 5. Przykłady

Podamy obecnie i zanalizujemy kilka przykładów deklaracji procedur.

```
procedure DODWEK(n,P,Q,R);
value n;
integer n;
array P,Q,R;

  for i:= 1 step 1 until n do
  P[i]:= Q[i] + R[i]
```

Treścią procedury jest instrukcja "dla". Nazwy n, P, Q, R są parametrami formalnymi, wymienionymi w nawiasach okrągłych po nazwie procedury /DODWEK/.

Zbiór wartości nie jest pusty - wymieniony w nim jest parametr n. W zbiorze specyfikacji podane są informacje o wszystkich parametrach formalnych.

Zwróćmy uwagę, że średniki umieszczone są po:
- zbiorze parametrów formalnych,
- zbiorze wartości,
- każdej specyfikacji ze zbioru specyfikacji.

Zmienna prosta i nie jest parametrem formalnym ani zmienną lokalną, musi więc być zadeklarowana w programie, zgodnie z p.2 w 24-2.1.

Większą niezależność procedury uzyskać możemy po zlokalizowaniu tej zmiennej i w treści procedury:

```
procedure DODWEK(n,P,Q,R);
value n; integer n; array P,Q,R;
begin integer i;
```

```
    for i:= 1 step 1 until n do  P[i]:= Q[i] + R[i]
end
```

2. W deklaracji procedury mogą być umieszczane komentarze zgodnie z zasadami podanymi w paragrafie 20-2.

```
procedure Absmax(a, n, m, y, i, k);
comment procedura wyznacza najwiekszy co do modulu element macierzy
        a o wymiarach n × m, wartosc jego podstawia na zmienna y,

        natomiast wskazniki tego elementu na i oraz k;
array a; integer n,m,i,k; real y;
begin integer p,q;
  y:= 0;
  for p:= 1 step 1 until n do
  for q:= 1 step 1 until m do
  if abs(a[p,q]) ≥ y then
  begin
    y:= abs(a[p,q]);
    i:= p; k:= q
  end
end Absmax
```

3.
```
procedure triangle(a,b,c,S);
real a,b,c,S;
comment procedura wyznacza pole S trojkata o bokach a, b, c. Jeze-
    li trojkat o tych bokach nie moze istniec, wowczas na S podsta-
    wiona jest wartosc -1;
begin real p;
  p:= (a + b + c)/2;  p:= p × (p-a) × (p-b) × (p-c);
  S:= if p < 0 then -1 else sqrt(p)
end triangle
```

4. Procedurę z poprzedniego przykładu zadeklarujemy jako procedurę funkcyjną /zmieniając dla odróżnienia nazwę procedury/.

```
real procedure Triangle (a,b,c);
real a,b,c;
begin real p;
  p:= (a + b + c)/2;  p:= p × (p-a) × (p-b) × (p-c);
  Triangle:= if p < 0 then -1 else sqrt(p)
end Triangle
```

5.
```
real procedure SLAD(A,n);
comment procedura wyznacza slad macierzy kwadratowej A rzedu n;
array A;  integer n;
begin real s; integer k;
  s:= 0;
  for k:= 1 step 1 until n do s:= s + A[k,k];
  SLAD:= s
end procedury SLAD
```

24 - 6. Opis w metajęzyku

formal parameter	- parametr formalny
letter string	- łańcuch liter /ciąg liter/
parameter delimiter	- ogranicznik parametrów
formal parameter list	- wykaz parametrów formalnych
formal parameter part	- zbiór parametrów formalnych
identifier list	- wykaz nazw
value part	- zbiór wartości
specifier	- specyfikacja
specification part	- zbiór specyfikacji
procedure identifier	- nazwa procedury
procedure heading	- nagłówek procedury
procedure body	- treść procedury
code	- kod
procedure declaration	- deklaracja procedury

< formal parameter > ::= < identifier >

< letter string > ::= < letter > | < letter string > < letter >

< parameter delimiter > ::= , |) < letter string >:(

< formal parameter list > ::= <formal parameter> |
<formal parameter list> <parameter delimiter> <formal parameter>

< formal parameter part > ::= < empty > | (< formal parameter list >)

< identifier list > ::= <identifier>|<identifier list>,<identifier>

< value part > ::= value < identifier list >; |< empty >

< specifier > ::= string | label | switch | array | procedure |
< type >|< type > array |< type > procedure

< specification part > ::= <empty>|<specifier> <identifier list>; |
<specification part> <specifier> <identifier list>;

< procedure identifier > ::= < identifier >

< procedure heading >::=<procedure identifier> <formal parameter part>;
<value part> <specification part>

< procedure body > ::= < statement >|< code >

< procedure declaration > ::=
procedure < procedure heading > < procedure body >|
< type > procedure < procedure heading > < procedure body >

26 - 6.1. Objaśnienia

1. Parametry formalne oddzielane są ogranicznikiem parametrów. W poprzednio podanych przykładach jako ogranicznika używaliśmy tylko przecinka. Z definicji w metajęzyku wynika, że przecinek możemy zastąpić równoważną mu konstrukcją:) < ciąg liter > : (

   Zatem pierwsze części nagłówków procedur podanych w paragrafie 24-5 mogą mieć również inną postać:

```
DODWEK(n)wektor:(P) jest suma wektorow:(Q,R)
        zamiast DODWEK(n,P,Q,R)

Absmax(a)wymiary:(n,m)wynik:(y)wskazniki:(i,k)
      zamiast Absmax(a,n,m,y,i,k)
triangle(a,b,c)Pole:(S)
      zamiast triangle(a,b,c,S)

SLAD(A)wymiar macierzy:(n)
      zamiast SLAD(A,n)
```

Własność ta ma charakter pomocniczy - analogicznie, jak konstrukcja komentarza omówiona w rozdziale 20.

2. Zgodnie z tym, co było powiedziane w poprzednich paragrafach, zbiór parametrów formalnych, zbiór wartości oraz zbiór specyfikacji, mogą być zbiorami pustymi - zależy to od konkretnych procedur.

24 - 7. Uwagi

1. Mamy jeszcze jedno rozszerzenie instrukcji podstawienia. Przy jej pomocy nadawane są wartości nie tylko zmiennym ale i nazwom procedur funkcyjnych. Przy czym to ostatnie podstawienie może wystąpić tylko w treści procedury.
2. Zdefiniowujemy obecnie ściśle, co rozumiemy w ALGOLu pod słowem deklaracja.

declaration - deklaracja

< declaration > ::= < type declaration >|< array declaration >|

< switch declaration >|< procedure declaration >

24 - 7.1. GIER ALGOL

1. W GIER ALGOLu /podobnie, jak i w większości konkretnych reprezentacji ALGOLu/ wszystkie parametry formalne muszą być specyfikowane.
2. W zbiorze wartości nie mogą występować parametry formalne specyfikowane jako array oraz label.

24 - 8. Ćwiczenia

24. Wskazać w przykładach 2, 3, 4 i 5 paragrafu 24-5 nagłówek procedury i jego części składowe:
nazwę procedury, zbiór parametrów formalnych, średnik

oddzielający część pierwszą nagłówka od drugiej, zbiór wartości, zbiór specyfikacji oraz ograniczniki parametrów. Poza tym stwierdzić, które z nazw w treści procedury są lokalne, nielokalne lub są parametrami formalnymi.

25. WYWOŁYWANIE PROCEDUR

Operacje opisane w deklaracji procedury wykonywane są w czasie wywoływania procedury. Procedurę wywołujemy albo przez umieszczenie w programie instrukcji procedury, albo - w przypadku procedur funkcyjnych - przez umieszczanie w odpowiednim wyrażeniu /arytmetycznym lub boolowskim/ nazewnika funkcyjnego.

25 - 1. Wstępne informacje

1. Procedura może być wywołana w bloku, na początku którego została zadeklarowana.
2. Instrukcja procedury oraz nazewnik funkcyjny mają tę samą budowę. Jest to nazwa procedury z następującym po niej zbiorem parametrów aktualnych /por. wywoływanie procedury DODWEK, rozdz. 23/.
3. Parametrem aktualnym może być wyrażenie /w szczególnym przypadku: zmienna, etykieta lub liczba/, nazwa /tablicy, procedury lub przełącznika/ oraz łańcuch.
4. Zbiór parametrów aktualnych, jest to ciąg parametrów aktualnych rozdzielonych przecinkami /lub dowolnymi innymi ogranicznikami parametrów/, ujęty w nawiasy okrągłe. Na przykład:

   (b,3,ALFA,c+7) (sin(x)\2,1,1,A[3]+2,B[i])

5. Ilość parametrów aktualnych musi być zgodna z ilością parametrów formalnych, podanych w nagłówku deklaracji

procedury. W szczególności więc zbiór parametrów aktualnych może być zbiorem pustym.

6. Rodzaj i typ każdego parametru aktualnego musi być zgodny z rodzajem i typem odpowiadającego mu parametru formalnego.
7. Wykonanie instrukcji procedury /lub nazewnika funkcyjnego/, równoważne jest wykonaniu odpowiednio zmodyfikowanej treści procedury.

25 - 2. Opis realizacji wywołania procedury

Wywołanie procedury równoważne jest wykonaniu następujących czynności:

1. Treść procedury zostaje zmodyfikowana według poniższych zasad:
   - Parametry formalne nie wymienione w zbiorze wartości /mówimy krótko: wołane przez nazwę/ są zastąpione we wszystkich instrukcjach i wyrażeniach odpowiadającymi im parametrami aktualnymi, ujętymi w nawiasy wszędzie tam, gdzie jest to możliwe ze względu na składnię.
   - Na parametry formalne wymienione w zbiorze wartości /krócej: wołane przez wartość/ podstawiane są bezpośrednio przed treścią procedury wartości odpowiadających im parametrów aktualnych. Nazwy tych parametrów nie są w treści zastępowane. Podstawianie wartości odbywa się tak, jak gdyby był utworzony fikcyjny blok /zawierający wewnątrz treść procedury/, w którym odpowiednie parametry formalne wołane przez wartość są zadeklarowane. W bloku tym występują tylko wspomniane powyżej instrukcje podstawienia wartości bezpośrednio przed treścią procedury.
2. Tak zmodyfikowana treść procedury zostaje wstawiona na miejsce instrukcji procedury i wykonana. Jeżeli procedura została wywołana nazewnikiem funkcyjnym, przyjmujemy, że zmodyfikowana treść zostaje wykonana w trakcie obliczania wartości odpowiedniego wyrażenia. W momencie

wyjścia z treści procedury wyliczona w niej wartość nazewnika funkcyjnego zostaje użyta do kontynuacji obliczania wartości tego wyrażenia.

Z podanego powyżej opisu nie wynika poprawność i jednoznaczność konstrukcji ALGOLu pojawiających się w programie w czasie wywoływania procedury. Dlatego też opis ten wymaga kilku wyjaśnień - podamy je w dalszych punktach paragrafu po prostych przykładach.

25 - 2.1. Przykłady

Rozpatrzymy dwukrotne wywoływanie w programie procedury ABSMAX, która różni się od podanej w przykładzie 2 paragrafu 24-5 tym, że parametr m wołamy przez wartość. Dodaliśmy to dla lepszego zilustrowania zmian dokonywanych na treści procedury w czasie wywoływania procedury. Modyfikując treść procedury nie będziemy ujmowali parametrów aktualnych zastępujących parametry formalne wzywane przez nazwę w nawiasy, ponieważ w przypadku tej procedury jest to zbyteczne /nie prowadzi do błędnej interpretacji/.

```
begin array A[1:20,1:7]; integer array Index[1:2]; real max1,max2;
integer wiersz,kolumna,M,N,i,k,a,b,c;
procedure ABSMAX(a,n,m,y,i,k);
value m; integer n,m,i,k;
array a; real y;
begin integer p,q;
  y:= 0;
  for p:= 1 step 1 until n do
  for q:= 1 step 1 until m do
  if abs(a[p,q]) ≥ y then
  begin
    y:= abs(a[p,q]); i:= p;  k:= q
  end
end procedury ABSMAX;
. . . . . .
```

```
E1:  ABSMAX(A,20,7) wyniki:(max1,Index[1],Index[2]);
. . . . . .
begin array Te[1:M,1:N-1];
  if a × b ≤ M ∧ c ≤ N/2 then
E2:
  ABSMAX(te,a×b,2×c-1,max2,i,k);
  . . . . .
  end bloku;
  . . . . .
end programu
```

Podamy teraz zmodyfikowaną treść procedury, która będzie wstawiona na miejsce instrukcji procedury po etykiecie E1 oraz po etykiecie E2.

```
E1:  begin integer m;
       m:= 7;
       begin integer p,q;
         max1:= 0;
         for p:= 1 step 1 until 20 do
         for q:= 1 step 1 until m  do
         if abs(A[p,q]) ≥ max1 then
         begin
           max1:= abs(A[p,q]);  Index[1]:= p;  Index[2]:= q
         end
       end tresci procedury
     end fikcyjnego bloku
E2:    begin integer m;
         m:= 2 × c - 1;
         begin integer p,q;
           max2:= 0;
           for p:= 1 step 1 until a × b do
           for q:= 1 step 1 until m do
           if abs(Te[p,q]) ≥ max2 then
           begin
             max2:= abs(A[p,q]);  i:= p;  k:= q
           end
         end
       end
```

Konstrukcje treści procedury muszą być po dokonaniu modyfikacji opisanych na początku tego paragrafu, poprawnymi konstrukcjami ALGOLu. W szczególności więc należy przestrzegać następujących zasad:

1. Jeżeli parametr formalny wołany przez nazwę pojawia się po lewej stronie dowolnej instrukcji podstawienia występującej w treści procedury, wówczas odpowiadający mu parametr aktualny może być tylko zmienną. Wynika to stąd, że w instrukcji podstawienia po lewej stronie może występować tylko zmienna, a parametry formalne wołane przez nazwę zastępowane są przy modyfikacji treści procedury, odpowiadającymi im parametrami aktualnymi.

2. Parametrowi formalnemu wołanemu przez wartość może odpowiadać jedynie parametr aktualny posiadający wartość, tzn. będący wyrażeniem lub nazwą tablicy /wartością nazwy tablicy jest uporządkowany zbiór wartości zmiennych indeksowanych będących elementami tej tablicy/.

3. Jeżeli parametrem formalnym jest nazwa tablicy, wówczas sposób użycia jej w treści procedury określa na ogół wymiar i granice wskaźników tablicy /informacji tych nie podaje specyfikacja/. Parametr aktualny, odpowiadający temu parametrowi, musi być tablicą o tym samym wymiarze i takich samych /lub szerszych/ granicach wskaźników.

4. Ograniczenia odnośnie parametru aktualnego będącego łańcuchem podamy w następnym rozdziale, przy omawianiu tego pojęcia.

W czasie wywoływania procedury może zaistnieć sytuacja, że ta sama nazwa reprezentuje różne obiekty. Zdarzyć się mogą bowiem następujące przypadki:

1. Parametry formalne wołane przez nazwę zastępowane są parametrami aktualnymi, zawierającymi nazwy identyczne z nazwami lokalnymi dla treści procedury. Na przykład, gdy nazwy p, q wystąpią w instrukcji procedury ABSMAX:

```
ABSMAX(K,(p+q):7+1,3,u,v,w)
```

2. Parametry formalne wołane przez nazwę zastępowane są parametrami aktualnymi, zawierającymi nazwy oznaczające w treści procedury parametry formalne wołane przez wartość. Na przykład, gdy nazwa m wystąpi w instrukcji procedury ABSMAX:

```
ABSMAX(Alfa,m,3,u,v,w)
```

3. Nazwy nielokalne dla treści procedury są takie same, jak nazwy lokalne w bloku, w którym procedura jest wywoływana /a który jest blokiem wewnętrznym w stosunku do bloku, w którym procedura jest deklarowana/. Na przykład, należy ustalić do której z etykiet A nastąpi skok przy wywoływaniu procedury TEST w następującym programie:

```
      begin procedure TEST(q);
      Boolean q; if q then go to A;
      comment trescia procedury jest instrukcja warunkowa;
          . . . . .
A:    begin integer k;
          . . . . .
        TEST(true);
A:        . . . . .
      end bloku wprowadzajacego nowa etykiete lokalna A;
      . . . . .
    end programu
```

Sprzeczności, do których mogłyby prowadzić wyżej podane sytuacje, wyeliminowane są w ALGOLu przez odpowiednie zamiany nazw. W przypadkach podanych w punkcie 1 i 2 zamieniane są nazwy lokalne dla treści procedury, lub nazwy parametrów formalnych. Zatem instrukcja:

```
ABSMAX(K,(p+q):7+1,3,u,v,w)
```

równoważna jest instrukcji:

```
begin integer m;
  m:= 3;
  begin integer p1,q1;
  comment zmienione zostaly nazwy lokalne;
      a:= 0;

      for p1:= 1 step 1 until (p+q):7+1 do
      for q1:= 1 step 1 until m do
      if abs(K[p1,q1]) > u then
      begin
        u:= abs(K[p1,q1]);  v:= p1;  w:= q1

      end
    end
  end
```

zaś instrukcja

```
ABSMAX(Alfa,m,3,u,v,w)
```

równoważna jest instrukcji:

```
begin integer m1;
comment zamieniona zostala nazwa parametru formalnego wolanego

    przez wartosc;
  m1:= 3;
  . . . . .
end
```

Natomiast w przypadku podanym w punkcie 3 zamieniana jest nazwa lokalna dla bloku, w którym wołamy procedurę. Zatem program z procedurą TEST równoważny jest następującemu programowi:

```
begin procedure TEST(q);

  Boolean q;  if q then go to A;
    . . . . .
A:  begin integer k;
      . . . . .
      if true then go to A;
A1:   . . . . .
    end;
    . . . . .
  end
```

Procedury wprowadzają więc wyjątek w regułach o "zasłanianiu", podanych w paragrafie 19-2. - są one "silniejsze". Wywołanie ich w bloku wewnętrznym powoduje "odsłonięcie" nazw nielokalnych dla treści procedury "zasłoniętych" w tym bloku. Oczywiście użycie w bloku wewnętrznym nazwy takiej samej, jak nazwa procedury dla oznaczenia innego, lokalnego obiektu powoduje "zasłonięcie" tej procedury dla tego bloku.

25 - 3. Znaczenie zbioru wartości

Sposób modyfikacji treści procedury w czasie wywoływania procedury zależy od tego, czy dany parametr wołany jest przez wartość czy też przez nazwę. Rozpatrzmy dwie procedury:

```
procedure EX1(a,b,c);
value a,c;  real a,b;  integer c;
begin integer k;
  a:= (a+1) × sqrt(abs(a)) + 1;
  b:= 0;
```

```
  for k:= 1 step 1 until c do   b:=(b+a)/k
end
```

oraz

```
procedure EX2(a,b,c);
real a,b; integer c;
begin integer k;

  a:= (a+1) × sqrt(abs(a)) + 1;
  b:= 0;
  for k:= 1 step 1 until c do
  b:= (b+a)/k
end
```

Procedury różnią się tylko tym, że parametry a, c wołane są w EX1 przez wartość, zaś w EX2 przez nazwę.

1. EX1 (p+q,B,7) jest poprawną instrukcją, natomiast EX2 (p+q,B,7) nie jest poprawną instrukcją, gdyż konstrukcja otrzymana w wyniku modyfikacji treści

$$p + q := (p + q + 1) \times \mathrm{sqrt}\,(\mathrm{abs}(p + q))$$

jest nieprawidłowa w ALGOLu /por. 25-2.2 p.1/.
Wniosek 1: parametr formalny wymieniony w zbiorze wartości nagłówka procedury może służyć w treści procedury jako zmienna robocza, niezależnie od tego, czy odpowiedni parametr aktualny będzie zmienną czy wyrażeniem.

2. Niech zmienna A ma wartość zero bezpośrednio przed wywołaniem procedury EX1 (A,B,C) i procedury EX2 (A,B,C) Wartość zmiennej A nie zostanie podczas wywołania procedury EX1 zmieniona, zaś po instrukcji EX2(A,B,C) wartością A będzie 1.
Wniosek 2: wołanie przez wartość chroni zmienną, będącą parametrem aktualnym przed zmianą wartości w trakcie wywoływania procedury. Wynika to z faktu, że parametr formalny wołany przez wartość jest lokalny dla fikcyjnego

bloku /co praktycznie biorąc oznacza lokalność dla treści procedury/ i odpowiedni parametr aktualny służy jedynie do obliczenia nadawanej mu początkowej wartości. Możemy stąd wysnuć dalszy wniosek - jeżeli pewne parametry aktualne mają wyprowadzić z treści procedury obliczone w niej wielkości /np. parametry odpowiadające w procedurze ABSMAX parametrom formalnym y, i, k/ wówczas parametry formalne, które będą przez te parametry aktualne zastępowane, nie mogą być wołane przez wartość.

3. Przy wywołaniu procedury EX1 instrukcją

EX1(A,B,3/2-7+4×82)

wartość wyrażenia arytmetycznego podstawionego na miejsce parametru c obliczana jest tylko raz /w fikcyjnym bloku/. Natomiast w instrukcji procedury EX2(A,B,3/2-7+4×82) wartość tego wyrażenia obliczana będzie 331 razy /o jeden raz więcej niż ilość powtórzeń instrukcji kontrolowanej w instrukcji "dla"/.
Wniosek 3: w przypadku wywoływania procedury z wyrażeniami jako parametrami aktualnymi, może okazać się dużo bardziej ekonomiczne /z punktu widzenia czasu/ wołanie parametrów formalnych przez wartość.

4. Istnieją przykłady prccedur, gdzie jedynie pozornie opłaca się wołać niektóre parametry przez wartość. Rozważmy procedurę:

```
procedure Innerproduct(a,b,k,p,y);
value k; integer k,p; real a,b,y;
begin real s;
  s:= 0;
  for p:= 1 step 1 until k do  s:= s + a × b;
  y:= s
end Innerproduct
```

Dzięki temu, że w deklaracji procedury Innerproduct parametry formalne a, b wołane są przez nazwę, możliwe jest użycie tej procedury do przemnożenia skalarnego

trzeciej kolumny macierzy Q przez jej drugi wiersz, przy pomocy instrukcji:

```
Innerproduct(Q[i,3],Q[2,i],16,i,iloczyn skalarny)
```

Widzimy z powyższych przykładów, że konstrukcje "wołanie parametru przez nazwę" i "wołanie przez wartość" mają na ogół zupełnie odmienne zastosowanie. W tych przypadkach, w których obydwie konstrukcje byłyby dopuszczalne, należy wybierać ten sposób wołania parametru, który daje bądź większą ekonomię czasu pracy maszyny bądź ekonomię rezerwacji miejsca w pamięci maszyny - zależnie od konkretnej sytuacji.

25 - 4. Opis w metajęzyku

actual parameter - parametr aktualny
actual parameter list - wykaz parametrów aktualnych
actual parameter part - zbiór parametrów aktualnych
procedure statement - instrukcja procedury
function designator - nazewnik funkcyjny

<actual parameter>::= <string>|<expression>|<array identifier>|
<switch identifier>|<procedure identifier>

<actual parameter list>::= <actual parameter>|
<actual parameter list> <parameter delimiter> <actual parameter>

<actual parameter part>::= <empty>|(<actual parameter list>)

<procedure statement>::= <procedure identifier> <actual parameter part>

<function designator>::= <procedure identifier> <actual parameter part>

25 - 4.1. Objaśnienia

Instrukcja procedury oraz nazewnik funkcyjny mają taką samą budowę syntaktyczną. Procedurę niefunkcyjną można wywoływać tylko instrukcją procedury, natomiast procedurę funkcyjną można wywoływać zarówno instrukcją procedury /tzn. umieszczając tę instrukcję w programie, w dowolnym

miejscu, gdzie instrukcja ALGOLu może występować/ jak i nazewnikiem funkcyjnym /umieszczając go w wyrażeniu arytmetycznym lub boolowskim/. Na przykład procedura funkcyjna Triangle /por. 24-5/ pojawiająca się w programie następująco:

```
. . . . .
Triangle(2,3,4);
. . . . .
```

jest wywoływana instrukcją procedury, zaś pojawiająca się w programie w konstrukcji:

```
. . . . .
P:= 5.7 + 2 × Triangle(2,3,4);
. . . . .
```

jest wywoływana nazewnikiem funkcyjnym.

Jeżeli procedurę funkcyjną wywołujemy instrukcją procedury, to nie możemy wykorzystać w dalszej części programu wartości podstawianej na nazwę procedury w trakcie takiego wywołania procedury. W wyrażeniu, w którym chcielibyśmy z wartości procedury skorzystać musi pojawić się odpowiedni nazewnik funkcyjny, czyli procedura zostanie ponownie wywołana.

25 - 5. Przykłady

Podamy deklarację procedury POLOWIENIE, która służy do wyznaczania przybliżonej wartości pierwiastka funkcji w zadanym przedziale.

```
procedure POLOWIENIE(f,x,a,b,eps,BRAK ZERA);

value a,b,eps;

real f,x,a,b,eps;

label BRAK ZERA;

comment Parametrami procedury sa:
```

f - funkcja rzeczywista zmiennej x. Przy wolaniu procedury pramaterem aktualnym powinno byc wyrazenie arytmetyczne zalezne od x,

x - zmienna niezalezna. Na wyjsciu /po wykonaniu instrukcji procedury/ wartoscia x jest znalezione przyblizenie pierwiastka,

a,b - krance przedzialu w ktorym szukamy pierwiastka,

eps - zadana dokladnosc, tzn. wyznaczone przyblizenie rozni sie od pierwiastka nie wiecej niz o eps,

BRAK ZERA - etykieta, do ktorej nastepuje skok, jezeli nie mozna wyznaczyc ta procedura pierwiastka w < a,b >;

```
begin real fa,fb;
  x:= a;  fa:= f;  if fa = 0 then go to KONIEC;
  x:= b;  fb:= f;  if fb = 0 then go to KONIEC;
  if fa × fb > 0 then go to BRAK ZERA;
  for x:= (a+b)/2 while abs(b-a) > eps do
  if fa × f < 0 then
  begin
    b:= x;  fb:= f
  end
  else
  begin
    a:= x;  fa:= f
  end;

KONIEC:
end POLOWIENIE
```

Procedurę tę zastosujemy w programie na znalezienie pierwiastków wielomianu

$$x^4 + 1{,}7x^3 - 1{,}6x^2 - 1{,}7x + 0{,}6$$

w przedziałach $[-5/2,-3/2]$ oraz $[-1/2, 1/2]$

```
  begin real z;
  < deklaracja procedury POLOWIENIE >;
    POLOWIENIE(z↑4+1.7×z↑3-1.6×z↑2-1.7×z+0.6,z,-2.5,-1.5,₁₀-8,L1);
    output(⟨+d.dddd₁₀+dd⟩,-2.5,-1.5,z);
    comment jezeli w przedziale znaleziony zostal pierwiastek, to
            wyperforowane sa trzy liczby: krance przedzialu i war-
            tosc pierwiastka;
L1: POLOWIENIE(z↑4+1.7×z↑3-1.6×z↑2-1.7×z+0.6,z,-0.5,0.5,₁₀-8,L2);
    output(⟨+d.dddd₁₀+dd⟩,-0.5,0.5,z);
L2:
  end programu
```

25 - 6. Ćwiczenia

25. Jaka liczba będzie wyperforowana na taśmię w wyniku zrealizowania następującego programu /należy przeprowadzić dokładną analizę wywołania procedury f oraz g/:

```
begin real r,Re,s,Se;
real procedure f(a);
value a; real a;
begin
  r:= r↑2;   f:= a × r
end f;
real procedure g(a);
real a;
begin
  s:= s↑2;  g:= a × s
end g;
comment w procedurze g parametr formalny a jest
wzywany przez nazwe, zas w f przez wartosc;
r:= 2;
Re:= f(r↑2);
s:= 2;
Se:= g(s↑2);
```

```
output(‡+d.dddd₁₀+dd‡,Re-Se)
end programu
```

26. Napisać deklarację procedury na wyznaczanie średniej arytmetycznej /A/, geometrycznej /G/ oraz harmonicznej /H/ danych n liczb.

```
/ A = (x1 + x2 + ... + xn)/n
  G = (x1 × x2 × ... × xn)↑(1/n)
  H = n/(1/x1 + 1/x2 + ... + 1/xn) /
```

26. PROCEDURY - UZUPEŁNIENIE

26 - 1. Procedury bez parametrów

Z metajęzykowego opisu deklaracji procedur /24-6/ wynika, że zbiór parametrów formalnych może być zbiorem pustym. W treści takich procedur muszą występować więc nazwy nielokalne. Nazwy te należy w programie zadeklarować, na równym lub wyższym poziomie w stosunku do deklaracji procedury /por. 24-2.1 p.3/. Wywołując procedurę bez parametrów instrukcją procedury lub nazewnikiem funkcyjnym w odpowiednich konstrukcjach programu umieszczamy jedynie nazwę procedury.

26 - 1.1. Przykłady

1.
```
begin integer p; real a,b,c,d,e;
  procedure Nota;
  p:= entier(ln((a/4 + b/4) × abs(c)));
  procedure Alp;
  begin integer i;
    d:= 0;
    for i:= 1 step 1 until p do
    d:= d + e/2; d:= d/p
  end;
  . . . . .
  Nota;
  d:= a × p + b;
```

```
    Alp;
    a:= b:= d↑(1/3);
    if a > p then Nota else Alp;
    . . . . .
end
```

2.
```
begin real x,y,S,p,q;
real procedure R;
  R:= sqrt(x↑2 + y↑2);
  . . . . .
  S:= p + q + R;
  comment procedura R wywolywana jest w wyrazeniu arytmetycznym,
  po prawej stronie instrukcji podstawienia;
  . . . . .
end
```

3. Instrukcja:

```
if b1 then a else c:= a
```

jest prawidłową instrukcją ALGOLu, przy założeniu, że nazwa a jest nazwą procedury funkcyjnej bez parametrów. W instrukcji tej procedura a będzie wywoływana jednym z dwu sposobów: albo instrukcją procedury /po symbolu then/albo nazewnikiem funkcyjnym /w instrukcji podstawienia, po symbolu else/.

26 - 2. Procedury rekurencyjne

Procedurami rekurencyjnymi nazywamy takie procedury, które wywołują same siebie w treście procedury,bezpośrednio lub przy pomocy innej procedury. Użycie procedur rekurencyjnych pozwala często na bardzo zwięzły i elegancki zapis ich treści, ale stosowanie ich jest na ogół nieekonomiczne /poza nielicznymi wyjątkami/.

26 - 2.1. Przykłady

Podamy przykłady procedur rekurencyjnych obliczania silni liczby naturalnej n /n!/. Procedury te służą jedynie do ilustracji zagadnienia - stosunkowo prostu można zapisać procedurę wyznaczania silni, bez użycia rekurencyjności:

1.
```
integer procedure SILNIA(n);
value n; integer n;
SILNIA:= if n = 0 then 1 else n × SILNIA(n-1)
```

2.
```
integer procedure S1(n);
S1:= if n = 0 then 1 else S2(n);
integer procedure S2(k);
S2:= k × S1(k-1)
```

/w deklaracjach nie specyfikujemy parametrów formalnych - jest to dozwolone w ALGOLu, ale nie w GIER ALGOLu/.

26 - 2.2. Uwagi

Przy opracowywaniu procedur funkcyjnych należy zwrócić uwagę, że pojawienie się w treści procedury nazwy procedury inaczej, niż po lewej stronie instrukcji podstawienia, oznaczać będzie wywołanie tej procedury.

26 - 3. Rekurencyjne wywoływanie procedur

Rekurencyjne wywoływanie procedur polega na tym, że jedynym z parametrów aktualnych jest dana procedura, lub wyrażenie zawierające tę procedurę. Najprostszymi przykładami rekurencyjnego wywoływania procedur są następujące konstrukcje:

```
sin(sin(x))                    sqrt(1+b×sqrt(a))
abs(A[7] + ln(abs(b-sqrt(abs(c)) × ln(d/2))) × abs(d))
```

Rekurencyjne wywoływanie procedur ma szczególne zastosowanie w przypadkach obliczania całek wielokrotnych /przy użyciu procedury funkcyjnej wyznaczającej całkę oznaczoną/, sum wielokrotnych itd.

26 - 3.1. Przykłady

Załóżmy, że dysponujemy procedurą funkcyjną obliczania całki oznaczonej w przedziale a, b z funkcji rzeczywistej f zmiennej niezależnej x. Niech nagłówek tej procedury będzie następujący:

```
real procedure CALKA(a,b,f,x);
value a,b;  real a,b,f,x;
```

Wówczas z procedury tej można skorzystać do obliczania całki $\int_0^1 dx \int_1^2 (x^2 + y \sin 2y)\,dy$ w następujący sposób:

```
CALKA(0,1,CALKA(1,2,x↑2+y×sin(2×y),y),x)
```

26 - 4. Uboczne działanie procedur

Rozpatrzmy następujący program:

```
begin real k,W,x,y;
real procedure PODSTEP(z);
value z;  real z;
  begin

    PODSTEP:= z + (z-2)/2;
    W:= z + 1

  end procedury;
  k:= 2;  W:= 1;
  x:= PODSTEP(k) × W;
  k:= 2;  W:= 1;
```

```
    y:= W × PODSTEP(k)
end programu
```

Po wykonaniu programu zmienna x ma wartość 6, zaś zmienna y ma wartość 2. Wynika to stąd, że przy wywoływaniu procedury PODSTEP w instrukcji podstawienia na zmienną x, wartość zmiennej nielokalnej W zostaje zmieniona, "za plecami użytkownika".

Zmiany wartości obiektów o nazwach nielokalnych dla treści procedury, w czasie wywoływania procedury, nazywamy ubocznym działaniem procedur. Należy być szczególnie ostrożnym przy stosowaniu procedur o nielokalnych nazwach w treści procedury, nieprzemyślane ich użycie może łatwo doprowadzić do nieprzewidzianych przez programistę skutków.

26 - 5. Łańcuchy

26 - 5.1. Wstępne informacje

1. Łańcuchem nazywamy dowolny ciąg symboli podstawowych, ujęty w nawiasy ‘ ’ z tym, że jeżeli w ciągu tym występuje lewy nawias ‘ , to musi mu odpowiadać prawy nawias ’ i odwrotnie.
2. Łańcuchy mają zasadnicze zastosowanie w procedurach wyjścia w konkretnych reprezentacjach języka ALGOL 60. Konkretne przykłady łańcuchów omówimy więc w następnych rozdziałach.
3. Jeżeli łańcuch jest parametrem aktualnym procedury, której treść jest wyrażona w ALGOLu, wówczas w treści może on być użyty tylko jako parametr aktualny do wywołania innej procedury. Ostatecznie więc łańcuch ten musi być użyty jako parametr aktualny jakiejś procedury, której treścią jest kod.

26 - 5.2. Opis w metajęzyku

proper string - prawidłowy łańcuch
open string - otwarty łańcuch
string - łańcuch

< proper string > ::= < dowolny ciąg symboli podstawowych niezawierający ‘ani’ > | < empty >

< open string > ::= < proper string > | ‘< open string >’ |
< open string > < open string >

< string > ::= ‘< open string >’

26 - 5.3. Przykłady

‘’ ‘ab:/.,’ “al,,,,, kon”

“...;,,XV∧V‘aaa13’‘,,’” abcdefghijklm’

26 - 5.4. Uwagi

1. Symbol podstawowy , /por. 2-1.4.2. oraz 2-2.1.p.4/ oznacza odstęp i może być używany tylko w łańcuchach.

26 - 5.4.1. GIER ALGOL

Nawiasy ‘ ’ zastąpione są w GIER ALGOLu nawiasami ≮ ≯. Spotkaliśmy się z nimi po raz pierwszy w rozdziale 10, przy omawianiu wzorca w procedurze output.

26 - 6. Uwagi

Na zakończenie omawiania procedur podamy jeszcze dwie uwagi, które mogą pomóc w wyjaśnieniu ewentualnych niejasności.

26 - 6.1. Parametry formalne i nazwy nielokalne

Rola parametrów formalnych i nazw nielokalnych w treści procedury jest analogiczna w pewnym sensie do roli ujawnionych i nieujawnionych argumentów funkcji. W matematyce rozpatrując zależność funkcyjną

$$y = ax^2 + bx + c$$

funkcję y traktujemy często jako funkcję tylko zmiennej x, zapisując

$$y = w(x) = ax^2 + bx + c$$

Wartość funkcji zależy oczywiście również od zmiennych a,b,c, ale w wielu zagadnieniach wystarczy ujawnienie zależności y od zmiennej x.

Jeżeli jednak musimy podkreślić zależność zmiennej y od innych zmiennych, wówczas zapisujemy np.

$$y = u(x,a) \quad \text{lub} \quad y = v(x,a,b,c) \quad \text{itd.}$$

Ujawnienie danej zmiennej niezależnej pozwala na uproszczenie zapisu wyrażeń, np.

$$w(A + 1) \qquad u(z,5) \qquad v(x^2,1,2,a + b + c)$$

26 - 6.2. Użycie bloków i procedur

Zacytujemy teraz sugestie Petera Naura odnośnie używania bloków i procedur, podane w publikacji /4/:

"Ważnym krokiem w planowaniu programu ALGOLowego jest podzielenie programowanego algorytmu na części, które mogą być łatwo zapisane w postaci bloków lub procedur. Aby było to możliwe, programista musi dobrze rozumieć własności tych konstrukcji ALGOLu. Następujące uwagi mogą być użyteczne jako wprowadzenie.

Bloki są użyteczne dla opisania takich części programu, które tworzą w pewnym sensie zamknięte całości. W

szczególności blok jest niezbędny, jeżeli w algorytmie konieczne jest użycie tablicy, której wymiary zależą od wyniku poprzednich obliczeń. Tablica taka musi być lokalna dla bloku. Dodatkowo, dowolny inny obiekt /zmienna prosta, etykieta, przełącznik, procedura/, który jest wykorzystywany tylko do obliczeń wewnątrz bloku /czyli po wykonaniu tych obliczeń nie jest nam potrzebny/, może być zlokalizowany dla tego bloku. Jest to szczególnie użyteczne, gdy różne bloki programu pisane są przez różnych programistów. Korzystając z bloków, programiści powinni uzgodnić jedynie nielokalne nazwy bloków. Natomiast każdy programista może dowolnie wybierać nazwy, reprezentujące robocze obiekty wewnątrz bloku.

Procedury mają trzy ważne zastosowania:

1. Uogólnienie zastosowania bloków,
2. Skrócenie zapisu wprowadzanych doraźnie funkcji,
3. Przekazywanie opracowanych algorytmów między programistami różnych ośrodków.

Wyjaśnimy to w kilku zdaniach:

1. Dowolny blok może być przekształcony w procedurę, przez dodanie nagłówka. Nagłówek przyporządkowuje nazwę blokowi i na ogół określa niektóre lub wszystkie nielokalne nazwy bloku, jako parametry formalne procedury. Jeżeli rozważany blok napisany jest specjalnie dla danego programu, takie przekształcenie bloku w procedurę zaleca się jedynie wtedy, gdy blok używany jest dwa lub więcej raz z różnymi nielokalnymi nazwami, co odpowiada dwu lub więcej krotnemu wywołaniu procedury. Wynika to z oczywistego faktu, że wywołanie procedury jest procesem bardziej pracochłonnym, niż zwykłe wejście do odpowiedniego bloku.
2. Program może być często skrócony przez użycie odpowiednich nazewników funkcyjnych. Analogicznie, jak w punkcie 1 powyżej, będzie to ekonomiczne tylko wtedy, gdy odpowiadające doraźnie wprowadzane funkcje mają być używane w programie więcej niż jeden raz.

3. Należy przypuszczać, że w najbliższej przyszłości wszystkie ważniejsze metody analizy numerycznej będą wyrażane i publikowane w postaci procedur ALGOLowych /por. Comm.ACM - the Algorithms section oraz BIT - ALGOL Programming section/. Ponieważ procedury te będą przypuszczalnie więcej niż przeciętne pod względem wydajności, zaleca się usilnie używanie ich gdziekolwiek jest to możliwe."

26 - 7. Ćwiczenia

27. Podać liczby, które będą wyperforowane w wyniku zrealizowania następujących trzech programów:

I.

```
begin real a;
procedure P(x,y);
real x,y;
  a:= x + y;
  a:= 6;
  P(1,2);
  output(⟨+d.dddd₁₀+dd⟩,a)
end
```

II.

```
begin real a;
procedure P(x,y);
real x,y;
begin real a;
  a:= x + y
end;
a:= 6;
P(1,2);
output(⟨+d.dddd₁₀+dd⟩,a)
end
```

III.

```
begin real a;
procedure P(x,y);
real x,y;
  a:= x + y;
  a:= 6;
  begin real a;
    a:= 1;
    P(1,2);
    output(⟨+d.dddd₁₀+dd⟩,a)
    end;
    output(⟨+d.dddd₁₀+dd⟩,a);
    P(1,2);
    output(⟨+d.dddd₁₀+dd⟩,a)
  end
```

28. Kiedy w AOLGOLu równość

a + b = b + a

może być fałszywa /skorzystać z informacji podanych w paragrafach 26-1 oraz 26-4/.

27. INFORMACJE O MASZYNIE GIER I URZADZENIACH POMOCNICZYCH

GIER ALGOL III jest konkretną reprezentacją ALGOLu 60 opracowaną dla duńskiej maszyny cyfrowej GIER. Na wstępie omawiania GIER ALGOLu podajemy więc kilka informacji o maszynie GIER.

27 - 1. Ogólna charakterystyka

Wiadomości podane w tym paragrafie przeznaczone są dla czytelników, którzy znają podstawowe pojęcia, związane z maszynami cyfrowymi. Informacje te nie są niezbędne dla osób programujących w GIER ALGOLu - dotyczą one tylko programów pisanych w języku wewnętrznym maszyny.

1. GIER jest tranzystorową maszyną binarną, równoległą i jednoadresową.
2. Arytmometr realizuje działania arytmetyczne stało i zmiennoprzecinkowe.
3. Słowo o stałej długości 42 bity /co odpowiada w przybliżeniu 12 cyfrom dziesiętnym lub 7 znakom alfanumerycznym /służy do przechowania jednej instrukcji długiej lub dwu instrukcji krótkich, albo jednej liczby stało lub zmiennoprzecinkowej.
4. Liczba stałoprzecinkowa ze znakiem zapisana jest 40 bitami, co odpowiada prawie 12 cyfrom dziesiętnym. Liczba zmiennoprzecinkowa zawiera mantysę ze znakiem zapisana 30 bitami /prawie 9 cyfr dziesiętnych/ oraz cechę ze

znakiem zapisaną 10 bitami /prawie 3 cyfry dziesiętne/.

5. Czas trwania operacji w mikrosekundach /do każdego z nich należy dodać 27μsek. na automatyczną modyfikację adresu/:

Dodawanie :	22 /st.p./	,	66 /zm.p./
Mnożenie :	155 /st.p./	,	140 /zm.p./
Dzielenie :	244 /st.p./	,	190 /zm.p./
Inne operacje:	2 do 27		

27 - 2. Poziomy pamięci

1. Jednostka centralna GIER wyposażona jest w dwa poziomy pamięci:
   - Ferrytowa pamięć operacyjna /skrót: PO/ o pojemności 1024 miejsc /komórek/ i czasie dostępu 10 mikrosekund/słowo.
   /Miejsce służy do przechowania jednego słowa 42 bitowego/.
   - Bęben magnetyczny o pojemności 12800 miejsc /320 ścieżek po 40 miejsc każda/ i czasie dostępu 20 milesekund/ścieżka.
   Gospodarkę miejscami tych pamięci w GIER ALGOLu III oraz sposób korzystania z bębna, jako dodatkowej pamięci, omówimy dokładnie w jednym z dalszych rozdziałów.
2. Standardowe wyposażenie maszyny może być uzupełnione dodatkowymi poziomami pamięci. Maszyna GIER zainstalowana w Zakładzie Obliczeń Numerycznych Uniwersytetu Warszawskiego posiada dodatkowo:
   - Bufor /pamięć ferrytowa/ o pojemności 4096 miejsc i czasie dostępu 6-13 mikrosekund/słowo.
   - Karuzelę Facita /pamięć na taśmie magnetycznej/ o pojemności 64 szpule x 16 bloków x 512 miejsc /w sumie dla każdego zestawu szpul 524 288 miejsc/ i o czasie dostępu 0,1-2,7 sekundy/blok.

Korzystanie z tych poziomów pamięci w GIER ALGOLu III jest stosunkowo skomplikowane - wymaga dołączania do programu bloków pisanych w języku wewnętrznym maszyny. W skrypcie tym nie będziemy omawiać szczegółowo sposobów użycia języka wewnętrznego w programach ALGOLowych. Tym samym więc nie będą podane informacje wystarczające do zapoznania się ze sposobami korzystania w GIER ALGOLu z bufora i karuzeli.

27 - 3. Urządzenia wejścia - wyjścia

1. Wprowadzanie informacji /tzn. programu i danych/ do pamięci maszyny może odbywać się dwojako:
   - Poprzez czytnik, wczytujący informacje uprzednio wydziurkowane na 8-mio kanałowej taśmie papierowej. Szybkość czytnika - do 2000 znaków na sekundę. Taśmę perforujemy przy pomocy flexowritera /urządzenie zbliżone do dalekopisu/.
   - Poprzez elektryczną maszynę do pisania, sprzężoną z jednostką centralną /nazywać ją będziemy: monitor/.

   Wyprowadzanie wyników z maszyny również może odbywać się dwoma sposobami:
   - Poprzez perforator, dziurkujący informacje na 8-mio kanałowej taśmie papierowej /którą następnie odczytujemy na flexowriterze/. Szybkość perforatora - do 150 znaków na sekundę.
   - Poprzez monitor. Szybkość monitora 8-12 znaków na sekundę.

   Zasadniczo zaleca się korzystanie z czytnika i perforatora, ponieważ urządzenia te są dużo szybsze od monitora. Flexowriter omówimy szczegółowo w następnym paragrafie.
2. Na stoliku operatora znajduje się tzw. klucz KB, który możemy ustawić ręcznie w pozycji: włączony lub wyłączony. Klucz KB wykorzystany jest w GIER ALGOLu III - od jego położenia zależy wartość logiczna standardowej procedury funkcyjnej kbon.

27 - 4. Flexowriter

Flexowriter jest elektryczną maszyną do pisania, na której można dodatkowo perforować 8-mio kanałową taśmę papierową, odczytywać ją oraz repreforować /sporządzać kopię taśmy/. Układy dziurek pojawiających się na taśmie mogą odpowiadać symbolom sterującym, symbolom typograficznym oraz symbolom drukowanym na papierze.

1. Symbole sterujące:
   STOP CODE, END CODE, PUNCH OFF, PUNCH ON, TAPE FEED.
   Omówimy je w dalszej części materiału.
2. Symbole typograficzne:
   SPACE /odstęp/
   TAB /kilka odstępów, w zależności od ustawionych uprzednio konikow tabulatora/
   CAR RET /powrót karetki, tzn. obrót wałka i maksymalne przesunięcie go w prawo - drukowanie rozpocznie się od nowej linii i od lewego marginesu/.
   LOWER CASE/dolne położenie czcionek/
   UPPER CASE/górne położenie czcionek/
3. Symbole drukowane:
   litery /por. 2-1.1 oraz 2-2.1 p. 1/, cyfry oraz 20 innych znaków /nawiasy, operatory, itd./, których wykaz zamieścimy w 27-4.1. Symbole drukowane połączone są w pary. Każdej parze odpowiada jeden klawisz flexowritera i jeden układ dziurek na taśmie. Wybór symbolu z pary zależy od położenia /dolnego lub górnego/ czcionek. Znaki pojawiające się na taśmie po LOWER CASE, aż do pierwszego UPPER CASE będą interpretowane jako wyperforowane w dolnym położeniu czcionek i analogicznie, po UPPER CASE, aż do pierwszego LOWER CASE, jako wyperforowane w górnym położeniu czcionek.

27 - 4.1. Reprezentacja liczbowa

Każdemu układowi /w jednym wierszu/dziurek na taśmie odpowiada pewna wartość liczbowa. Zależność pozycji

w linii od wartości ilustruje nam poniższy szkic:

Pozycja	8	7	6	5	4	3	2	1
Wartość dziesiętna:	64	32	16	0	8	4	2	1

Zatem dziurce na k-tej pozycji odpowiada wartość w, według wzoru:

w:= if k<5 then 2↑(k-1) else if k=5 then 0 else 2↑(k-2)

Jeżeli w linii jest kilka dziurek, to wartość układu równa jest sumie wartości, odpowiadających każdej z dziurek. Na przykład:

układ	o oo. oo	ma wartość 32 + 0 + 8 + 2 + 1 = 43
układ	o .	ma wartość 64
układ	oooo.o	ma wartość 32 + 16 + 0 + 8 + 4= 60

Podamy obecnie w tabeli wykaz wszystkich /istotnych w GIER ALGOLu/ znaków, które możemy wyperforować przy pomocy flexowritera. W pierwszej kolumnie /W/ zapisana jest wartość dziesiętna odpowiadająca danemu układowi, zaś w drugiej i trzeciej odpowiedni symbol, w dolnym /LC/ oraz górnym /UC/ położeniu czcionek.

W	LC	UC	W	LC	UC
0	SPACE		10	nieużywane	
1	1	v	11	STOP CODE	
2	2	×	12	END CODE	
3	3	/	13	nieużywane	
4	4	=	14	_	|
5	5	;	15	nieużywane	
6	6	[	16	0	∧
7	7	]	17	<	>
8	8	(	18	s	S
9	9	)	19	t	T

W	LC	UC	W	LC	UC
20	u	U	42	nieużywane	
21	v	V	43	ø	Ø
22	w	W	44	PUNCH ON	
23	x	X	45	nieużywane	
24	y	Y	46	nieużywane	
25	z	Z	47	nieużywane	
26	nieużywane		48	æ	Æ
27	,	»	49	a	A
28	CLEAR CODE		50	b	B
29	nieużywane		51	c	C
30	TAB		52	d	D
31	PUNCH OFF		53	e	E
32	-	+	54	f	F
33	j	J	55	g	G
34	k	K	56	h	H
35	l	L	57	i	I
36	m	M	58	LOWER CASE	
37	n	N	59	.	:
38	o	O	60	UPPER CASE	
39	p	P	61	SUM CODE	
40	q	Q	62	nieużywane	
41	r	R	63	TAPE FEED	
			64	CAR RET	

27 - 4.2. Uwagi

1. Klawisz ze znakami drukowanymi _| nie powoduje przesunięcia karetki. Użycie tego klawisza omówimy w następnym rozdziale.
2. Symbole CLEAR CODE, SUM CODE nie były wymienione w żadnej z trzech grup. Mają one znaczenie pomocnicze, przy tzw. mechaniźmie sum kontrolnych. Mechanizmu tego nie będziemy w skrypcie omawiać.

3. Flexowriter perforuje każdy symbol nieparzystą ilością dziurek. Jeżeli z rozwinięcia dwójkowego wartości danego symbolu wynika, że powinna być parzysta liczba dziurek /np. y: 24 = 16 + 8, 3: 3 = 2 + 1/, wówczas doperforowywana jest automatycznie dziurka na pozycji 5-tej /o wartości zero/.
Na przykład:
END CODE : 12 = 8 + 4 + 0
q : 40 = 32 + 8 + 0
TAPE FEED : 63 = 32 + 16 + 8 + 4 + 2 + 1 + 0
Pozwala to na przeprowadzenie kontroli przy wczytywaniu informacji z taśmy, do pamięci maszyny GIER.

27 - 5. Cwiczenia

29. Podać układy dziurek, które będą odpowiadały następującym 7-miu symbolom:

7 (t PUNCH OFF Æ | UPPER CASE

/znaleźć w tabeli ich wartość dziesiętną, rozłożyć na sumę potęg dwójki, pamiętając o ewentualnym dodaniu zera/.

28. RÓŻNICE MIĘDZY ALGOLEM 60 A GIER ALGOLEM III

Wprowadzając w poprzednich rozdziałach konstrukcje ALGOLu 60, podawaliśmy ewentualne różnice między budową lub sposobem użycia danej konstrukcji w języku wzorcowym, a w GIER ALGOLu. Różnic tych nie będziemy ponownie wymieniać - wskażemy jedynie paragrafy, w których była o nich mowa, dodając niekiedy uzupełnienia.

28 - 1. Symbole podstawowe

1. Należy ponownie przeczytać punkt 2-2.1.
2. W podanym w poprzednim rozdziale wykazie znaków, które znajdują się na klawiaturze flexowritera, nie ma wielu z symboli podstawowych GIER ALGOLu. Symbole te, powstające w wyniku złożenia kilku dostępnych znaków będziemy nazywali złożonymi. Symbole złożone /które wymieniliśmy w 2-2.1 p. 2-p.4/ otrzymujemy w większości przypadków przy pomocy znaku _ /podkreślenie/ lub | / jego odpowiednika w UPPER CASE/, połączonego z innymi znakami /por. uwaga 1 punktu 27-4.2/. Na przykład:
   - symbol podstawowy if otrzymujemy za pomocą czterech znaków flexowritera: podkreślenie, litera i, podkreślenie, litera f.
   - symbol podstawowy ↑ otrzymujemy za pomocą dwu znaków, obu w UPPER CASE: | ^
   - symbol podstawowy := otrzymujemy za pomocą dwukropka i znaku równości.

28 - 2. Liczby

1. Należy ponownie przeczytać punkt 3-5.1.
2. Jeżeli w trakcie realizacji programu następuje zaokrąglenie liczby typu real do typu integer /np. w instrukcji podstawienia, w której po lewej stronie jest zmienna całkowita, zaś po prawej wyrażenie typu real/, wówczas otrzymany wynik zależy od przedziału, w którym znajduje się zaokrąglana wartość rzeczywista r. Zależność ta jest następująca:
   - gdy abs(r) < 2 ↑ 29 = 536 870 912
   to wynik jest poprawny,
   - gdy 2 ↑ 29 ≤ abs(r) ≤ 2 ↑ 39
   to uzyskana liczba całkowita ma za mało cyfr znaczących,
   - gdy 2 ↑ 39 < abs(r) < 2 ↑ 512
   to wynik jest całkowicie błędny.

28 - 3. Zarezerwowane nazwy

Zarezerwowaną nazwą nazywamy taką nazwę, która może być użyta w programie do standardowego celu, bez uprzedniej deklaracji tej nazwy w programie. Zarezerwowane nazwy możemy podzielić na 4 grupy:

1. Funkcje standardowe:
   abs, arctan, cos, entier, exp, ln, sign, sin, sqrt /por. 7-1.1/.
2. Procedury wejścia:
   char, inchar, inone, input, kb on, lyn, setchar, typechar, typein.
3. Procedury wyjścia:
   outchar, outclear, outcopy, outcr, outputt, outsp, outsum, outtext, write, writechar, writecopy, writecr, writetext.
4. Zmienna standardowa i pozostałe procedury standardowe:

drumplace, from drum, gier, gierdrum, gierproc,pack, split, to drum.

28 - 4. Etykiety

Etykiety mogą być tylko nazwami.

28 - 5. Instrukcja "dla"

Należy ponownie przeczytać 21-4.1.

28 - 6. Miano own

Należy ponownie przeczytać 19-3 p. 5.

28 - 7. Deklaracje procedur

Należy ponownie przeczytać 24-7.1.

28 - 8. Ogólne ograniczenia

GIER ALGOL, podobnie jak każda inna konkretna reprezentacja, narzuca szereg ograniczeń na program, wynikających ze skończonych wymiarów /a zatem pojemności/ poszczególnych urządzeń maszyny. Niektóre z tych ograniczeń omówiliśmy już /np. zakres liczb/, niektóre omówimy /np. maksymalna ilość zmiennych deklarowanych w bloku/. Nie będziemy jednakże wspominali o tych limitach, których przekroczenie jest bardzo mało prawdopodobne /np. gdy nazwa składa się ze zbyt dużej ilości znaków/.

28 - 9. Uwagi

W dalszych rozdziałach będziemy tylko sporadycznie korzystali z formuł metajęzykowych. Nazwy zmiennych metajęzykowych podawać będziemy na ogół w terminologii pol-

skiej, bez względu na to, czy dana formuła definiuje całkowicie wprowadzoną konstrukcję, czy tylko częściowo.

29. STANDARDOWE PROCEDURY WEJŚCIA

Standardowe procedury wejścia służą do wprowadzania danych do pamięci maszyny. Każdorazowe wywołanie procedury wejścia w programie /instrukcją procedury lub nazewnikiem funkcyjnym/ powoduje uruchomienie odpowiedniego urządzenia wejścia i na ogół podstawienie wprowadzonej wartości na wskazaną nazwę /zmiennej, tablicy lub procedury/.

29 - 1. Wstępne informacje

1. Procedur: char i setchar /por. 28-3. p. 2/ nie będziemy tu omawiać. Mają one stosunkowo małe zastosowanie.
2. Procedury wejścia wprowadzające dane poprzez czytnik, z wyjątkiem procedury lyn /tzn. input, inone, inchar/, podlegają tzw. uniwersalnemu mechanizmowi wejścia. Mechanizm ten spełnia rolę "filtru", który powoduje pominięcie części wprowadzanych na taśmie symboli, a części z nich nadaje specjalne znaczenie.
3. Procedura kbon jest jedyną z procedur wejścia, która nie odnosi się ani do czytnika, ani do monitora /por. 27-3. p. 2/.

29 - 2. Uniwersalny mechanizm wejścia

29 - 2.1. Symbole "ślepe"

Niektóre z pojawiających się układów dziurek na taśmie nie stanowią żadnej informacji dla uniwersalnego me-

chanizmu wejścia. Będziemy o odpowiadających im symbolach mówili, że są ignorowane podczas wczytywania /są "ślepe"/. Poszczególne z procedur wejścia mogą mieć swoje własne "ślepe" symbole, generalnie jednak, następujące 4 układy dziurek są ignorowane:

.	Pusta taśma
oooo.ooo	TAPE FEED
ooooo.ooo	Wszystkie dziurki
ooo o.ooo	Dziurki wszędzie poza 5-tą pozycją.

29 - 2.2. Błąd parzystości

Pojawienie się na taśmie układu o przystej ilości dziurek, powoduje zatrzymanie wczytywania taśmy, a tym samym zatrzymanie realizacji programu /wyjątek stanowią wspomniane powyżej symbole "ślepe" - pusta taśma oraz wszystkie dziurki/.

29 - 2.3. PUNCH OFF - PUNCH ON

Wszystkie znaki pomiędzy PUNCH OFF i pierwszym następującym po nim PUNCH ON, włączając obydwa te symbole, są ignorowane podczas wczytywania taśmy. /PUNCH ON nie poprzedzony PUNCH OFF nie posiada specjalnego znaczenia/.

29 - 2.4. END CODE

Pojawienie się symbolu END CODE powoduje zatrzymanie wczytywania taśmy i wydrukowanie na monitorze informacji:

pause

Realizacja programu jest zatrzymana do momentu naciśnięcia dowolnego klawisza na monitorze. Naciśnięcie klawisza powoduje kontynuację wprowadzania danych.

29 - 2.5. LOWER CASE I UPPER CASE

Symbole te służą jedynie do wprowadzania odpowiedniego /dolnego lub górnego/ położenia czcionek. Wiąże się z tym interpretacja następujących po nich symboli drukowanych - por. 27-4. /Symbole typograficzne i kontrolne nie zależą od położenia czcionek/.

29 - 2.6. Pozostałe symbole

Symbole nie wymienione w poprzednich punktach, nie są traktowane w specjalny sposób przez uniwersalny mechanizm wejścia. Dzielą się one na 3 grupy: symbole informacyjne, symbole "ślepe", oraz symbole oddzielające /które wskazują, że zostało zakończone wczytywanie pewnego odcinka informacji, np. liczby/.

Podział związany jest z konkretną procedurą wejścia.

29 - 3. Procedura input

Wstępne informacje o tej procedurze podaliśmy już w paragrafie 10-1 oraz 12-4.3. Procedura input może być wywoływana z dowolną ilością parametrów aktualnych. /Takiej własności nie mogą mieć procedury z treścią wyrażoną w ALGOLu/.

Wywołanie procedury input /przez umieszczenie w programie instrukcji tej procedury/ powoduje uruchomienie czytnika i podstawienie wartości liczb wyperforowanych na taśmie papierowej na zmienne i tablice zmiennych /typu real lub integer/. W tym celu parametrami aktualnymi powinny być odpowiednie zmienne /proste lub indeksowane/ lub nazwy tablic. Zasady tego podstawiania wyjaśnimy poniżej.

29 - 3.1. Przygotowywanie danych

Taśma z danymi musi być przygotowywana według następujących zasad:

1. < symbol informacyjny > ::= < cyfra >|+|-|.|₁₀

   < symbol ślepy > ::= SPACE | _

   < symbol oddzielający > ::= < litera > |CARRET|STOP CODE|
   PUNCH ON|TAB|∨|∧|×|/|,|:|;|=|<|>|[|]|(|)||

2. Ciąg symboli informacyjnych /przemieszanych ewentualnie z symbolami ślepymi/ powinien tworzyć liczbę, zbudowaną zgodnie z regułami ALGOLu /por. rozdział 3/.
3. Liczby powinny być rozdzielone jednym lub więcej symbolami oddzielającymi /przemieszanymi ewentualnie z symbolami ślepymi/.

29 - 3.1.1. Przykłady

1. W lewej kolumnie podajemy ciąg znaków, który może pojawić się na taśmie, w prawej zaś odpowiadającą mu liczbę:

24 856,	24856
+344;	344
i= +16.4p	16.4
S [25]	25
podstaw: [w=8.21 ₁₀ 3]	8210
pi:= 3.14/	3.14

2. Następujący ciąg znaków:
   dane: wymiar = 8, elementy: 1,4,16,3 oraz 2i2i2 $\leq$ 4 |
   równoważny jest ciągowi:
   8,1,4,16,3,2,2,2,4,

29 - 3.2. Kolejność podstawiania

Podstawianie wartości wczytanych z taśmy odbywa się kolejno, od strony lewej do prawej, na każdy parametr wymieniony w wykazie parametrów aktualnych. Jeżeli jednym z parametrów jest nazwa /np. T/ tablicy n wymiarowej, o dolnych i górnych granicach wskaźników odpowiednio D1,D2,...,Dn,G1,G2,...Gn, wówczas podstawiane są wartości

na wszystkie zmienne indeksowane tej tablicy. Instrukcja input (T) jest w tym przypadku równoważna następującej instrukcji "dla":

```
for i1:= D1 step 1 until G1 do
for i2:= D2 step 1 until G2 do
. . . . . .
for in:= Dn step 1 until Gn do input (T[i1,i2,...,in])
```

/i1,i2,...,in są roboczymi, wewnętrznymi zmiennymi/.

29 - 3.2.1. Przykłady

Niech będą dane tablice **B[1:3,1:2,1:2], A[1:10]** oraz taśma z wyperforowanym na niej następującym układ znaków /kończących się średnikiem/:

Dane nr. 3, element A = 4.8, tablica B
ma elementy: 1,2,3,4,5,6,7,8,9,10,11,12;

Wówczas instrukcja:

input(p, A[p], B)

równoważna jest następującym 14-tu instrukcjom podstawienia:

```
p:= 3; A[p]:=4.8; B[1,1,1]:= 1; B[1,1,2]:= 2;
B[1,2,1]:= 3; B[1,2,2]:= 4; B[2,1,1]:= 5; ...; B[3,2,2]:= 12
```

29 - 3.3. Uwagi

Określimy pomocniczą zmienną metajęzykową **< input ditto >** Wartością jej jest minus lub ciąg minusów /ewentualnie przemieszanych z symbolami "ślepymi"/, np.:

\- --- ---=-=-= -

Między symbolami oddzielającymi, może znajdować się oprócz liczby, również **< input ditto >** . Znaczenie tej konstrukcji objaśniamy na przykładzie.

Niech na taśmie założonej do czytnika, będzie wyperforowany następujący układ znaków /zakończony przecinkiem/:

Start: -3, -, 17.2; ---alfa = 0,

wówczas w wyniku wykonania instrukcji:

input(a,b,c,d,e)

pominięte zostaną te zmienne, którym na taśmie odpowiada < input ditto > , tzn. b,d. Ich wartości zostają nie zmienione przez instrukcję podstawienia. Ostatecznie instrukcja input będzie równoważna ciągowi instrukcji podstawienia:

a:= -3; c:= 17.2; e:= 0

29 - 4. Procedura inone

1. inone jest procedurą funkcyjną typu real, bez parametrów formalnych.
2. Każdorazowe wywołanie procedury inone powoduje uruchomienie czytnika, wczytanie jednej liczby z taśmy i podstawienie wartości tej liczby na nazwę inone.
3. Dane na taśmie muszą być przygotowane identycznie jak dla procedury input.
4. W przypadku pojawienia się na taśmie w czytniku < input ditto > , wartością nazewnika funkcyjnego inone jest zero. Jest to jednak cechą obecnej wersji translatora GIER ALGOL III.

Dlatego też należy raczej opierać się na ogólnej definicji nazewnika funkcyjnego inone w przypadku < input ditto > która mówi, że wartość procedury jest nieokreślona.

29 - 4.1. Przykłady

1. a:= inone
2. if inone> 7 then go to L
3. A [1, inone,inone] := (inone/3) ↑ (inone + 7)

Jeżeli założymy, że na taśmie znajduje się układ sym-

boli A:4,1; 6.2, -3.6/
wówczas instrukcja podana w tym przykładzie równoważna jest następującej instrukcji podstawienia:

A[1,4,1]:= (6.2/3)↑(-3.6+7)

29 - 5. Procedura typein

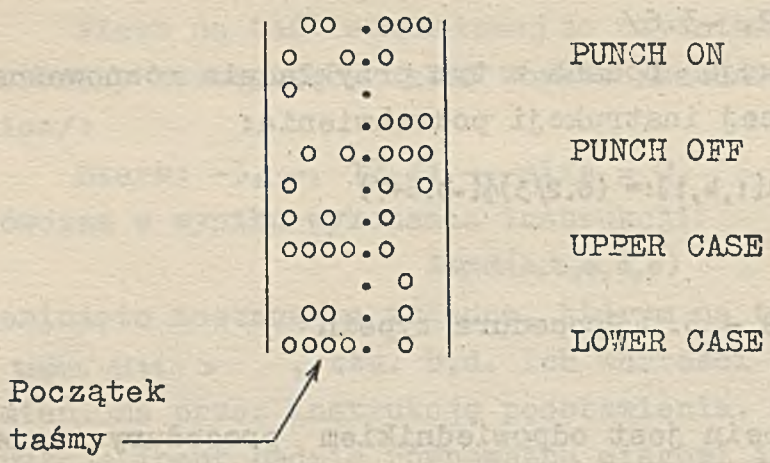
Procedura typein jest odpowiednikiem procedury inone z tą tylko różnicą, że symbole są wprowadzane /pisane/ z monitora, zamiast z czytnika /z taśmy/.

29 - 6. Procedura inchar

1. inchar jest procedurą funkcyjną typu integer, bez parametrów formalnych.
2. Każdorazowe wywołanie procedury inchar powoduje uruchomienie czytnika i wczytanie jednego symbolu właściwego, tzn. takiego, któremu uniwersalny mechanizm wejścia nie nadaje specjalnego znaczenia.
3. Wartość wczytanego symbolu właściwego podstawiana jest na nazwę inchar, przy czym
   - wartości symboli właściwych w dolnym położeniu czcionek, podane są bezpośrednio w tabeli, p. 27-4.1,
   - wartość symbolu w górnym położeniu czcionek równa jest wartości jego odpowiednika w dolnym położeniu, zwiększonej o 128.

29 - 6.1. Przykłady

Na taśmie mamy następujący ciąg symboli:



Odczytując ten odcinek taśmy na flexowriterze, otrzymalibyśmy wydruk: s2MN] +X

Ponieważ przed nawiasem kwadratowym na taśmie jest symbol PUNCH OFF, a po znaku + PUNCH ON, symbole zawarte między nimi będą zignorowane. Dodatkowo LOWER CASE i UPPER CASE służą jedynie do zmiany położenia czcionek.
Instrukcja:

A[inchar] := (inchar+inchar)/2 - inchar ↑ 2 + 3 × inchar

równoważna będzie w tym przypadku instrukcji:

A[18] := (2 + 164)/2 - 165 ↑ 2 + 3 × 151

29 - 7. Procedura typechar

Procedura typechar jest odpowiednikiem procedury inchar, z tą tylko różnicą, że symbole są wprowadzane z monitora, zamiast z czytnika.

29 - 7.1. Przykłady

Procedurę typechar stosuje się bardzo często do sterowania programem:

1. if typechar = 0 then go to NOWE DANE else goto KONIEC
2. if typechar = 51 then input (A) else
   for i:= 1 step 1 until q do A [i] := typein

/w zależności od tego, czy operator naciśnie na monitorze literę c, czy inny symbol, tablica A będzie wprowadzona z czytnika lub z monitora/.

29 - 8. Procedura lyn

1. lyn jest procedurą funkcyjną typu integer, bez parametrów formalnych.
2. Każdorazowe wywołanie procedury lyn powoduje uruchomienie czytnika i wczytanie jednego symbolu z taśmy.
3. Procedura lyn nie podlega uniwersalnemu mechanizmowi wejścia, zatem symbole specjalne dla tego mechanizmu tracą swoje własności.
4. Każdy wczytany symbol ma wartość wynikającą z odpowiadającego mu układu dziurek na taśmie, wartość ta zostaje podstawiona na nazwę lyn.

29 - 8.1. Przykłady

Rozważmy ten sam odcinek taśmy, co w przykładach 29-6.1. Wówczas instrukcja:

for i:= 1 step 1 until 11 do P[i] := lyn

równoważna jest następującym instrukcjom podstawienia:

```
P[1]:= 58;   P[2]:= 18;   P[3]:= 2;    P[4]:= 60;
P[5]:= 36;   P[6]:= 37;   P[7]:= 31;   P[8]:= 7;
P[9]:= 32;   P[10]:= 44;  P[11]:= 23;
```

29 - 8.2. Uwagi

Procedura lyn pozwala wczytywać taśmy, wyperforowane w innym kodzie. Wartość nazewnika funkcyjnego lyn musi się zawierać między 0 a $2^7 - 1 = 127$ /wszystkie dziurki/.

29 - 9. Procedura kb on

1. kb on jest procedurą funkcyjną typu Boolean, bez parametrów formalnych.
2. Wartością kb on jest true, gdy klucz KB jest włączony, false, gdy KB jest wyłączony /por. 27-3 p. 2/.

29 - 9.1. Przykłady

1. pierwsza seria:= kb on
2. if kbon $\vee$ i = 0 then go to KONIEC
3. if kbon then output ({+d.dddd$_{10}$+dd},k)

29 - 9.2. Uwagi

Nazwę kb on można pisać również bez odstępu, kbon.

30. STANDARDOWE PROCEDURY WYJŚCIA

Standardowe procedury wyjścia służą do wyprowadzania wyników lub tekstów. Każdorazowe wywołanie w programie procedury wyjścia powoduje wyprowadzenie za pomocą odpowiedniego urządzenia wyjścia określonego ciągu symboli, tzn. wydrukowanie tych symboli na monitorze lub wydziurkowanie ich na taśmie w perforatorze.

30 - 1. Wstępne informacje

1. W GIER ALGOLu III wprowadzono 13 standardowych procedur wyjścia - nazwy ich wymieniliśmy w p.3.paragrafu 28-3.
2. Nie będziemy tu omawiać procedur outchar i outsum. Mają one zastosowanie przy pewnym mechaniźmie kontroli, którego opis pomijamy w tym skrypcie.
3. Przed omówieniem najważniejszej procedury wyjścia - output /i jej odpowiednika - write, wyprowadzającego wyniki na monitorze/, opiszemy kilka prostych procedur, mających pomocniczy charakter.

30 - 2. Procedura outsp

1. < instrukcja outsp > ::= outsp(< wyrażenie arytmetyczne >)

2. Wywołanie procedury outsp instrukcją procedury powoduje wykonanie następujących operacji:

- obliczenie WA /wartości wyrażenia arytmetycznego, będącego parametrem aktualnym/,
- zaokrąglenie WA do najbliższej liczby całkowitej k /k:= entier (WA + 0.5) /,
- jeżeli k> 0 wówczas wyperforowanie na taśmie k symboli SPACE /odstęp/, w przeciwnym razie nie jest perforowany żaden symbol.

30 - 2.1. Przykłady

```
outsp(1)                    outsp(A[7] + 1 - a × b)
outsp(n)                    outsp(if b1 then c/2 else inone)
```

30 - 3. Procedura outcr

1. < instrukcja outcr > ::= outcr
2. outcr jest procedurą bez parametrów formalnych, której wywołanie instrukcją procedury powoduje wyperforowanie na taśmie symbolu CAR RET /powrót karetki - por. 27-4, p.2/.

30 - 3. Procedura writecr

Procedura writecr jest odpowiednikiem procedury outcr, z tym, że symbol CAR RET jest wyprowadzony na monitor /tzn. nastąpi na monitorze obrót wałka i przesunięcie karetki na prawo/.

30 - 5. Procedura outchar

1. < instrukcja outchar > ::= outchar(< wyrażenie arytmetyczne >)
2. Wywołanie procedury outchar instrukcją procedury powoduje wykonanie następujących czynności:
   - obliczenie WA /wartości wyrażenia arytmetycznego będącego parametrem aktualnym/,

- zaokrąglenie WA do najbliższej liczby całkowitej k,
- wyperforowanie na taśmie układu dziurek, którego wartością jest liczba całkowita k1 z przedziału [0,127], przystająca do k modulo 128 /tzn. różnica k-k1 dzieli się bez reszty przez 128/.

3. Ponieważ symbole GIER ALGOLu mają wartości z przedziału 0 - 64 /por. 27-4.1./, zatem większości otrzymanych procedurą outchar układów dziurek, może nie odpowiadać żaden symbol GIER ALGOLu. Przy odczytywaniu na flexo - writerze taśmy z takim układem dziurek, nie otrzymamy żadnego wydruku.
4. Zgodnie z p.3 paragrafu 27-4 symbole drukowane połączone są w pary. Każdej parze odpowiada jeden układ dziurek. Wybór symbolu z pary zależy od położenia czcionek. Gdy chcemy więc otrzymać ciąg symboli w górnym /dolnym/ położeniu, należy go poprzedzić symbolem UPPER CASE /LOWER CASE/ uzyskiwanym np. instrukcją outchar (60) /outchar (58)/.

30 - 5.1. Przykłady

1. Procedura outchar służy bardzo często do wyprowadzania symboli sterujących lub typograficznych, np.
   outchar (11) /STOP CODE/ outchar(12)/END CODE/
   outchar (30) /TAB/
2. Jeżeli chcemy otrzymać taśmę z wyperforowanym ciągiem

   B1AVC12

   to możemy to osiągnąć wywołując procedurę outchar 11 - to krotnie:

```
outchar(60); outchar(50); outchar(58); outchar(1); outchar(60);
outchar(49); outchar(1); outchar(51); outchar(58); outchar(1);
outchar(2)
```

30 - 6. Procedura writechar

1. Procedura writechar jest odpowiednikiem procedury outchar, z tym, że odpowiedni symbol jest drukowany na monitorze.
2. Jeżeli nie istnieje symbol o wartości k1, wówczas nie zostaje wydrukowany żaden znak.

30 - 6.1. Przykłady

Gdy chcemy odczytać na monitorze odcinek taśmy, możemy zastosować następujące instrukcje:

```
L: writechar (lyn) ;
   go to L
```

30 - 7. Procedura output

Pewne wiadomości o tej standardowej procedurze podaliśmy już w paragrafie 10-2.

30 - 7.1. Wstępne informacje

1. < instrukcja output >::= output(< wzorzec >,< wykaz parametrów wyjścia >)

2. < wzorzec > zawiera informacje o żądanej postaci wyprowadzanej wartości liczbowej - omówimy go dokładnie w jednym z dalszych punktów paragrafu /przykładowy wzorzec opisaliśmy w p. 10-2.1/.
3. < wykaz parametrów wyjścia > składa się z dowolnej ilości parametrów wyjścia rozdzielonych przecinkami, gdzie < parametr wyjścia > ::= < wyrażenie arytmetyczne >|< instrukcja wyjścia >
4. Wykonanie instrukcji output polega na przeglądzie kolejnych parametrów wyjścia /od strony lewej/ i wykonywaniu odpowiednich operacji, wyjaśnionych niżej:
   - jeżeli kolejnym parametrem wyjścia jest wyrażenie arytmetyczne, to obliczana jest jego wartość, a następ-

nie wartość ta zostaje wyperforowana na taśmie w postaci określonej przez wzorzec, będący pierwszym parametrem aktualnym wywołanej procedury;
- jeżeli kolejnym parametrem wyjścia jest instrukcja wyjścia /dowolna z 13-tu istniejących/, wówczas zostanie wywołana tą instrukcją odpowiednia procedura wyjścia.

30 - 7.1.1. Przykłady

Niech zmienne a,b,c mają odpowiednio wartości -64,22, 136 wtedy instrukcja:

```
output(⟨+d.dddd₁₀+dd⟩,outcr,b/2-7,outchar(60),outchar(3),outchar(3),
    outchar(58),c,outcr,sqrt(abs(a))+3,outchar(27),c/8,outsp(20),a)
```

spowoduje wyperforowanie taśmy, która po odczytaniu na flexowriterze da następujący wydruk:

```
 +4.7700 ₁₀+2//+1.3600 ₁₀+2
 +1.1000 ₁₀+1,+1.7000 ₁₀+1                    -6.4000 ₁₀+1
```

30 - 7.2. Wzorzec

Z definicji podanej w publikacji /2/ wynika możliwość konstruowania bardzo różnorodnych wzorców. Dla uproszczenia określimy początkowo wzorzec mniej ogólnie. Wzorzec jest to ujęty w nawiasy ⟨ ⟩ ciąg symboli, zbudowany według niżej podanych zasad, z następujących znaków:

\+ - ± d . ₁₀

1. Pierwszym symbolem występującym po nawiasie musi być jeden ze znaków: +,- lub ±. Oznaczając go przez z oraz przez D ciąg liter d /składający się co najmniej z jednego elementu/, możemy symbolicznie zapisać 6 możliwych postaci wzorca, następująco:

‹zD›	np. ‹+ddd›	‹±d›
‹zD.D›	np. ‹-d.dddd›	‹+ddddd.dd›
‹z.D›	np. ‹±.dd›	‹-.dddd›
‹zD$_n$zD›	np. ‹+dd$_n$-dd›	‹+d$_n$+d›
‹zD.D$_n$zD›	np. ‹-ddd.dd$_n$-d›	‹±d.ddd$_n$-dd›
‹z.D$_n$zD›	np. ‹±.d$_n$+d›	‹-.ddd$_n$+dddd›

2. Ilość liter d w ciągach

D lub D.D lub .D

/poprzedzających mnożnik skalujący/ określa ilość cyfr znaczących wyprowadzanej liczby. Gdy liczba ma więcej cyfr znaczących niż jest liter d, zostanie ona zaokrąglona.

3. Jeżeli nie możemy przewidzieć rzędu wyprowadzanych liczb /lub jest on bardzo duży, ew. bardzo mały/, należy umieścić we wzorcu mnożnik skalujący

$_n$zD

Liczba będzie wówczas perforowana z cyframi znaczącymi w żądanej postaci /omówionej powyżej/, zaś by jej wartość nie uległa zmianie, zostanie automatycznie dodany odpowiedni mnożnik skalujący. Ilość liter d w ciągu zD określa przedział, z którego będzie wybierana liczba całkowita do mnożnika.

4. Symbol z określa jakim znakiem będzie poprzedzona perforowana wartość bezwzględna liczby L. Jeżeli symbolem jest odpowiednio:

+, to przy $L < 0$ moduł tej liczby będzie poprzedzony minusem, przy $L \geq 0$ plusem.

-, to przy $L < 0$ moduł L będzie poprzedzony minusem, przy $L \geq 0$ odstępem /zamiast plusa/.

W obu tych przypadkach symbol + lub - lub SPACE będzie występować bezpośrednio przed pierwszą cyfrą liczby, lub kropką pozycyjną /będzie więc maksymalnie przesunięty na prawo/.

$\pm$, to przy L < 0 moduł L będzie poprzedzony minusem, przy L $\geq$ 0 plusem.
Symbol /+lub-/ nie będzie w tym przypadku przesuwany na prawo. Objaśnimy to na przykładach.

5. Nieznaczące zera przed kropką pozycyjną są opuszczane.
6. Podane poniżej przykłady ilustrują częściowo opisane /i pewne nieopisane tu/ szczegóły dotyczące związku między wzorcem a wydrukiem liczby.

30 - 7.2.1. Przykłady

	Liczba	Liczba zapisana w/g wzorców		
1.		{-ddd.dd}	{+ddd.dd}	{$\pm$ddd.dd}
	48	48.00	+ 48.00	+ 48.00
	-1	-1.00	-1.00	- 1.00
	333.487	333.49	+333.49	+333.49
	-0.007	-.01	-.01	- .01
	0.003	.00	+.00	+ .00

2.		{-.ddd$_n$+dd}	{+d.d$_n$-dd}	{$\pm$ddd$_n$+dd}
	48	.480 $_n$+2	+4.8 $_n$ 1	+480$_n$- 1
	-1	-.100 $_n$+1	-1.0	-100$_n$- 2
	333.487	.333 $_n$+3	+3.3 $_n$ 2	+333
	-0.007	-.700 $_n$-2	-7.0 $_n$-3	-700$_n$- 5
	0.003	.300 $_n$-2	+3.0 $_n$-3	+300$_n$- 5

30 - 7.2.2. Ograniczenia

1. Maksymalną dopuszczalną ilością liter d w ciągu .D lub D.D lub D /występującym przed kropką pozycyjną i mnożnikiem skalującym/ jest 15.
2. Maksymalną dopuszczalną ilością liter d w mnożniku skalującym jest 7.

Jeżeli moduł liczby, która ma być drukowana, jest większy niż przewiduje to wzorzec, wówczas wydrukowanie tej liczby zajmie więcej pozycji niż wynikałoby to ze wzorca /będzie bowiem dodany mnożnik skalujący, lub w istniejącym mnożniku będzie zwiększona ilość cyfr/. Na przykład:

Liczba	Liczba zapisana w/g wzorców	
	{+dddddd.dd}	{-d.dd$_{n}$-d}
4867521$_{n}$9	+486752.10$_{n}$10	4.87 $_{n}$15

Pełna definicja wzorca dopuszcza jeszcze następujące możliwości, które omówimy skrótowo:

1. Pierwsza z liter d ciągu D.D lub D /występującego przed kropką pozycyjną i mnożnikiem skalującym/ może być zastąpiona literą n. Wówczas będzie perforowane zero przed kropką pozycyjną wszędzie tam, gdzie jest to możliwe. Na przykład:

	{-n.dd}	{-d.dd}
0.2	0.20	20
	{-nd}	{-dd}
0	0	/nic nie jest drukowane/

2. Znak może być opuszczony, tzn. bezpośrednio po nawiasie { może wystąpić . lub d lub n. O liczbie zakłada się wtedy, że jest nieujemną i nie jest perforowany przed nią żaden znak.
   Jeżeli wyprowadzana liczba będzie jednak ujemna, to moduł jej zostanie poprzedzony minusem, zajmującym dodatkową, nie przewidzianą we wzorcu pozycję /jest to drugi przypadek tzw. drukowania alarmowego, por.30-7.2.3/.
3. Końcowe /poza pierwszą/ litery d z ciągu D.D lub .D lub D /występującego przed kropką pozycyjną i mnożnikiem

skalującym/ mogą być zastąpione przez cyfry 0 /przy czym, jeżeli którąś z liter d zastąpimy zerem, to następujące po niej w ciągu również należy zastąpić/. Zera mają znaczenie pomocnicze - rezerwują tylko miejsca we wzorcu, a nie określają ilości cyfr znaczących. Jeżeli we wzorcu z mnożnikiem skalującym jest m zer, to liczba całkowita w mnożniku będzie zawsze dobierana tak, by była podzielona przez m+1. Np.:

	⟨±dd.0ₙ-dd⟩	⟨nd.d00⟩	⟨nd.ddd⟩
0,001	+10 ₙ-4	0.001	0.001
0,012	+ 1.2 ₙ-2	0.012	0.012
0,123	+12 ₙ-2	0.123	0.123
1,234	+ 1.2	1.23	1.234

4. Między symbole w dowolnej z par: dd lub nd lub d0 może być wstawiony jeden symbol odstępu ␣ /lub równoważny mu SPACE/. Wtedy pomiędzy odpowiednimi cyframi będzie wyperforowany 1 symbol odstępu. Na przykład:

	⟨n␣ddd ddd.d 000⟩
4856.5251	4 856.5 251
66678.2656	66 678.2 66
3.3ₙ7	3 300 000.0

5. W związku z wyżej podanymi rozszerzeniami definicji wzorca, należy następująco uzupełnić ograniczenia, podane w p. 30-7.2.2.
- ogólna ilość liter n i d w części wzorca nie będącej mnożnikiem skalującym nie może przekraczać 15.
- ogólna ilość liter n, d oraz zer, występujących po lewej stronie kropki pozycyjnej, nie może przekraczać 15.
- ogólna ilość liter d oraz zer, występujących po prawej stronie kropki pozycyjnej, nie może przekraczać 15.
- nie możemy dawać symbolu odstępu w takim miejscu wzorca, gdzie poprzedza go więcej niż 20 symboli typu .,n,dP.

30 - 7.3. Wyrażenia wyznaczające wzorzec

W pełnej definicji instrukcji input, pierwszym parametrem aktualnym jest tzw. wyrażenie wyznaczające wzorzec. Konstrukcja ta ma budowę analogiczną do budowy wyrażeń ALGOLu.

<wzorzec ogólny>::= <wzorzec>|(<wyrażenie wyznaczające wzorzec>)

<wyrażenie wyznaczające wzorzec>::= <wzorzec ogólny>|
<warunek "jeśli"> <wzorzec ogólny>else<wyrażenie wyznaczające wzorzec>

Wartością tego wyrażenia jest zawsze wzorzec.

30 - 7.3.1. Przykłady

1. output((⫷dddd⫸),a)
2. output(if i > q then ⫷nd⫸ else ⫷d⫸,i)
3. output(if kbon then ⫷-n.dddddd⫸ else if b1 then ⫷d.dd$_{10}$-d⫸ else
⫷d.dd$_{10}$-dd⫸,A,outcr,B)

30 - 7.4. Uwagi

Procedura output pozostawia zawsze urządzenie wyjścia w położeniu LOWER CASE, a prawidłowe wykonanie tej procedury zakłada, że również w momencie rozpoczynania perforacji wyprowadzanej liczby, urządzenie wyjścia jest w dolnym położeniu czcionek. Należy zatem uważać, by nie spowodować wadliwego wykonania procedury output, tam, gdzie instrukcje tej procedury są przemieszane z instrukcjami outchar /lub writechar/. Na przykład w wyniku instrukcji

output(⫷-nd⫸,+5,outchar(60),-4)

druga z wyprowadzanych liczb będzie wyperforowana nieprawidłowo.

30 - 8. Procedura write.

Procedura write jest odpowiednikiem procedury output, z tym, że wyprowadzone liczby będą drukowane na monitorze. Procedury wyjścia, które występują jako parametry aktualne instrukcji write, będą wyprowadzały informacje poprzez odpowiadające im urządzenia wyjścia. Na przykład instrukcja:

write(‹nd›,i,output(‹-dd.d›,a),writecr,j)

spowoduje wydrukowanie wartości zmiennej j, pod wartością zmiennej i oraz wyperforowanie na taśmie wartości zmiennej a.

30 - 9. Procedura outtext

Procedura outtext służy do wyprowadzania ciągów symboli podstawowych /perforowania ich na taśmie/.

30 - 9.1. Wyrażenia wyznaczające teksty

Analogicznie do wprowadzonych uprzednio pojęć: wzorzec, wzorzec ogólny, wyrażenie wyznaczające wzorzec, określimy konstrukcje: tekst, tekst ogólny, wyrażenie wyznaczające tekst.

1. Tekst, jest to dowolny ciąg symboli podstawowych, za wyjątkiem ‹ oraz › , ujęty w nawiasy: ‹< >

   np. ‹<a = ›
   ‹< Tablicowanie funkcji f(x) dla
   x:= 0 step 1 until 20 ›

2. <tekst ogólny>::= <tekst>|(<wyrażenie wyznaczajace tekst>)

   <wyrażenie wyznaczające tekst>::= <tekst ogólny>|
   <warunek "jeśli"> <tekst ogólny>else<wyrażenie wyznaczające tekst>

   Wartością tego wyrażenia jest zawsze tekst.

30 - 9.1.1. Przykłady

1. (‹< WYNIKI ›)

2. if b1 then ‹<alfa = › else ‹<beta = ›

3. if inone > 3 then (‹<koniec pierwszego cyklu›)
   else ‹<powrot do nowych danych›

30 - 9.2. Instrukcja outtext

1. < parametr outtext > ::= < wyrażenie wyznaczające tekst > |
< procedura wyjścia >

< wykaz parametrów outtext > ::= < parametr outtext > |
< wykaz parametrów outtext >,< parametr outtext >

< instrukcja outtext > ::= outtext(<wykaz parametrów outtext>)

2. Wywołanie procedury outtext instrukcją procedury powoduje /w kolejności wskazanej w wykazie parametrów outtext/:
   - wyperforowanie na taśmie symboli zawartych między nawiasami tekstów, będących parametrami aktualnymi,
   - wywołanie procedur wyjścia, których instrukcje są parametrami aktualnymi.
3. Ilość parametrów aktualnych w instrukcji procedury outtext jest dowolna.

30 - 9.3. Przykłady

1. outtext(‡<x = ‡)
2. outtext(‡<WARTOSC KONCOWA:‡,output(‡nd‡,1),outcr,‡<skok:‡,
   output(‡.d‡,h))

Zakładając, że i = 34, h = 0.7, w wyniku wykonania tej instrukcji otrzymamy na taśmie ciąg symboli:
WARTOSC KONCOWA: 34
skok: .7

30 - 9.4. Uwagi

Uwaga 30-7.4. dotyczy również procedury outtext /po zastąpieniu w tej uwadze nazwy output nazwą outtext/.

30 - 10. Procedura writetext

Procedura writetext jest odpowiednikiem procedury outtext, z tym, że wyprowadzane symbole tekstów będą drukowane na monitorze /symbole wyprowadzane przez instrukcje wyjścia będące parametrami aktualnymi, będą drukowane lub perforowane, w zależności od tego, jakie urządzenie wyjścia jest z tymi instrukcjami powiązane/.

30 - 11. Procedura outcopy

1. <instrukcja outcopy>::= outcopy(<wyrażenie wyznaczające tekst>)

2. Ciąg symboli podstawowych zawartych między nawiasami tekstu, wyznaczonego przez wyrażenie będące parametrem aktualnym instrukcji, powinien składać się z jednego albo z dwu znaków.
3. Wywoływanie procedury outcopy instrukcją procedury powoduje kopiowanie znaków z taśmy wejściowej /założonej na czytnik/ na taśmę wyjściową /poprzez perforator/.
4. Jeżeli w tekście, będącym wartością parametru aktualnego występuje jeden symbol, kopiowanie rozpocznie się od aktualnego miejsca na taśmie, aż do tego symbolu. Jeżeli w tekście są dwa symbole, zostanie skopiowany odcinek taśmy wejściowej, zawarty między odpowiadającymi im układami dziurek.
5. Procedury outcopy jest procedurą wejścia-wyjścia. Podlega ona uniwersalnemu mechanizmowi wejścia /por. 29-2/.

30 - 11.1. Przykłady

Niech na taśmie wejściowej będzie wyperforowany ciąg znaków:

Taśma danych: [Numer problemu:]

Wtedy instrukcja:

outcopy(‡<[]‡)

spowoduje wyperforowanie taśmy z ciągiem symboli:

Numer problemu:

30 - 12. Procedura writecopy

Procedura writecopy jest odpowiednikiem procedury outcopy, z tym, że symbole z taśmy wejściowej nie są kopiowane na taśmę wyjściową, ale drukowane na monitorze.

30 - 13. Uwagi

1. Należy rozróżniać nawiasy stosowane dla wzorca: ≠ ≠
   tekstu: ≠< ≠
2. Omawiając procedury wspominaliśmy kilkakrotnie o konstrukcji ALGOLu, zwanej łańcuchem /string/.
   W GIER ALGOLu występują dwa rodzaje łańcuchów: wzorzec i tekst. Wykorzystywane są one w instrukcjach procedury output, write, outtext oraz writetext. Z informacji podanych w paragrafie 26-5 wynika, że procedury specyfikujące pewien parametr formalny jako łańcuch, mogą w treści korzystać z tych procedur standardowych, których parametrami są łańcuchy.
   Podamy obecnie przykład takiej procedury. Przed tym należy rozszerzyć następująco definicje podane w p. 30-7.3 oraz w p. 30-9.1.

<wzorzec ogólny>::= <wzorzec>|(<wyrażenie wyznaczające wzorzec>)|
<parametr formalny>

<tekst ogólny>::= <tekst>|(<wyrażenie wyznaczające tekst>)|
<parametr formalny>

```
procedure DRUK TABLICY(A) o wymiarach:(n,m) wedlug:(wzorzec, tekst);
value n,m;
integer n,m; array A; string wzorzec, tekst;
comment Elementy tablicy A[1:n,1:m] beda wyperforowane na tasmie
        wedlug zadanego wzorca. Po kazdym wierszu wystapi symbol
        CAR RET, elementy wiersza rozdzielone beda przecinkiem i
```

```
        jednym odstepem. Na poczatku tasmy wyperforowany zostanie
        zadany tekst, na koncu symbole: END CODE oraz STOP CODE;
begin integer i,j;
  outcr; outcr; outcr; outchar(58);
  outtext(tekst);
  outcr; outcr;
  for i:= 1 step 1 until n do
  begin
    outcr;
    for j:= 1 step 1 until m do
    output(wzorzec,A[i,j],outchar(27),outsp(1))
  end;
  outchar(12);
  outchar(11)
end DRUK TABLICY
```

Przykłady wywołania tej procedury

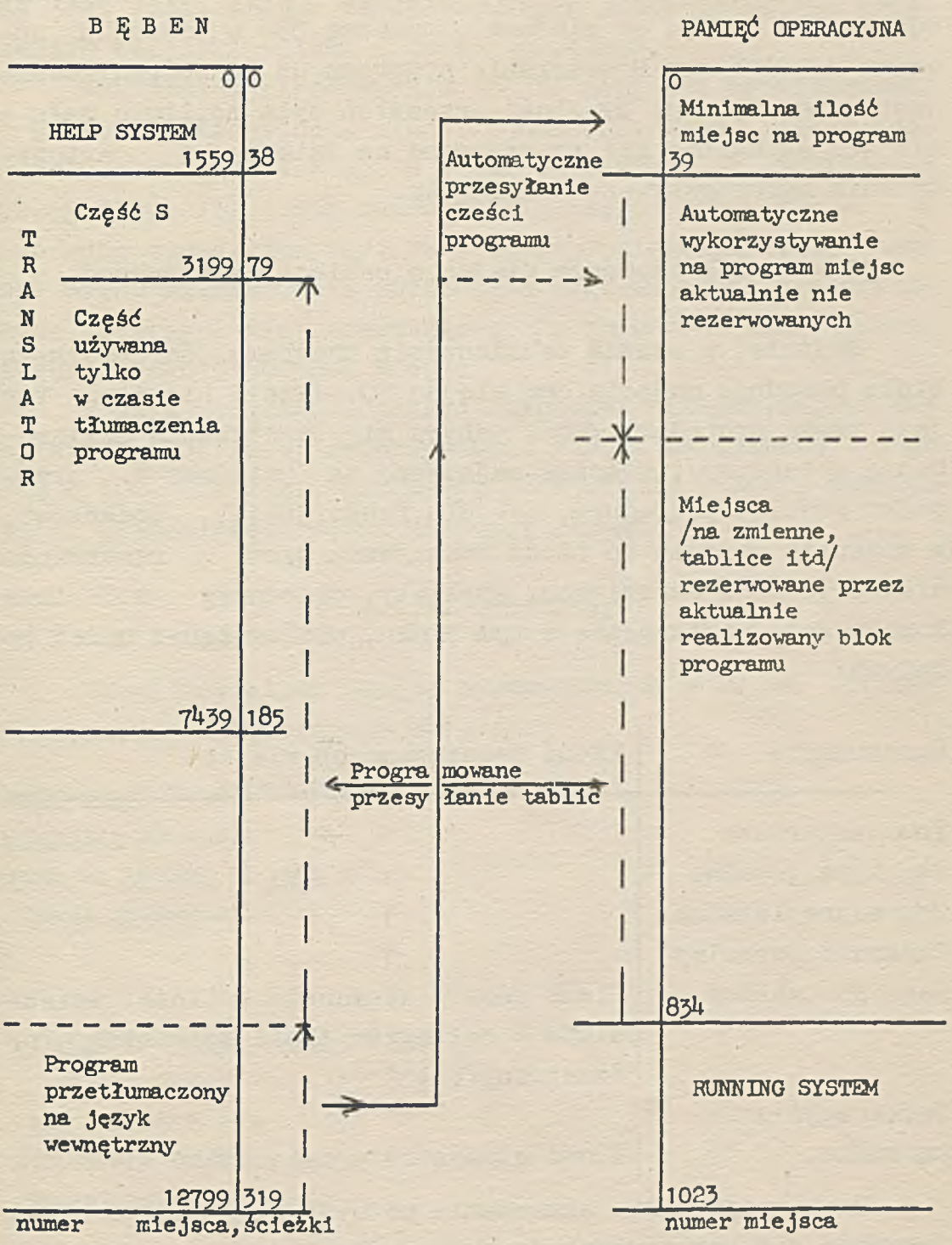
```
DRUK TABLICY (Q,10,10,‹ndddd›,‹<Macierz parametrow:›)
DRUK TABLICY (Pak,a,b,‹-n.dddd₁₀-d›,‹<elementy pak:›)
DRUK TABLICY (a12,cXd-7,set,‹±n.ddd00₁₀+d›,‹<Wyniki kontrolne
      testu IM, i-ty wiersz jest szukanym wektorem M i ›)
```

31. WYKORZYSTYWANIE PAMIĘCI OPERACYJNEJ I BĘBNOWEJ PRZY REALIZACJI PROGRAMÓW GIER ALGOLU

W trakcie realizacji programu przetłumaczonego z GIER ALGOLu na język wewnętrzny, pamięci jednostki centralnej GIER /por. 27-2/ współpracują ze sobą automatycznie. Ponadto, o ile w programie wywoływane są procedury standardowe to drum i from drum, wówczas odbywają się też wskazane przez programistę przerzuty danych z PO na bęben i z bębna do PO /PO - pamięć operacyjna/. Współpracę tę ilustruje poniższy diagram. Objaśnimy go dokładnie dalej w tekście - należy tylko pamiętać, bęben ma 12800 miejsc /320 ścieżek/, zaś PO - 1024 miejsca.

Program zapisany w GIER ALGOLu zostaje po przetłumaczeniu umieszczony na bębnie, począwszy od ostatniego miejsca /o numerze 12799/. Na bębnie zapisany jest poza tym system administracyjny maszyny, HELP, /ścieżki 0 - 38/, oraz translator GIER ALGOLu, składający się z dwu części. Część S /ścieżki 39-79/ zawiera informacje potrzebne do pracy programu /m.in. opisane są tam wszystkie procedury standardowe/. Pozostała część translatora /ścieżki 80-185/ wykorzystywana jest jedynie w czasie tłumaczenia programu. Zatem na miejsca zajmowane przez nią można przesłać tablice z PO /omówimy to w paragrafie 31-2/.

W pamięci operacyjnej miejsca 835-1023 zajęte są przez RUNNING SYSTEM /system administracyjny GIER ALGOLu/, który organizuje przesyłanie części przetłumaczonego programu z bębna do PO /w celu ich realizacji/ oraz rezerwowanie miejsc na zmienne deklarowane w danym bloku programu. Ilość rezerwowanych miejsc zmienia się w czasie pracy programu



Uwaga: strzałki na diagramie wskazują kierunek przesyłania lub zapisywania informacji.

/omawiamy to w paragrafie 31-1/, przy czym rezerwowanie odbywa się zawsze od miejsca o numerze 834 w kierunku numerów malejących. Przesyłanie programu do PO zorganizowane jest w ten sposób, by ilość przesyłań była możliwie mała - miejsca aktualnie nie rezerwowane na zmienne są automatycznie wykorzystywane na program.

31 - 1. Rezerwowanie miejsc w pamięci operacyjnej

Wejście, w czasie realizowania programu, do dowolnego bloku powoduje zajęcie dwu miejsc PO. Jeżeli blokiem tym jest treść procedury /tzn. odbywa się wywoływanie deklarowanej procedury/, wówczas zajmowane są 3 miejsca w przypadku zwykłej procedury, 4 - dla funkcyjnej. Dodatkowo, w momencie wejścia do bloku dokonywana jest rezerwacja miejsc pamięci na zmienne, etykiety, procedury i inne konstrukcje deklarowane w tym bloku, według zasad niżej podanych:

Konstrukcja	Ilość rezerwowanych miejsc
Zmienna prosta	1
Etykieta dowolna	1
Procedura lokalna	1
Parametr formalny	1
Segment tablicy	Ilość nazw w segmencie + ilość wskaźników + całkowita ilość zmiennych indeksowanych + 1
Deklaracja przełącznika	Ilość elementów przełącznika + 1

Rezerwacja ulega skasowaniu po wyjściu z danego bloku /poprzez end lub instrukcję skoku/. Ponieważ program zbudowany jest na ogół z kilku bloków /każde wywołanie procedury deklarowanej, równoważne jest ze wstawieniem bloku/, więc ilość rezerwowanych miejsc zmienia się w czasie pracy programu. W każdym momencie ilość zajętych w PO miejsc rów-

na jest sumie miejsc zarezerwowanych przez aktualnie działające wszystkie bloki i treści procedur.

Maksymalna ilość miejsc, które mogą być zarezerwowane w pamięci operacyjnej, wynosi ponad 700. Wtedy jednak program przesyłany jest do PO w małych dawkach. W niektórych przypadkach /np. duża ilość pętli, częste korzystanie z procedur standardowych/, realizacja takiego programu może być nawet kilkakrotnie wolniejsza od realizacji analogicznego programu, ale rezerwującego na zmienne i inne konstrukcje około 500-600 miejsc.

W celu wyzyskania pełnej szybkości maszyny, zaleca się ograniczenie ilości rezerwowanych miejsc na zmienne do około 500, a w razie potrzeby, korzystanie z bębna jako pamięci pomocniczej /patrz następny paragraf/.

31 - 1.1. Przykłady

Określimy ilość miejsc rezerwowanych w PO w czasie realizacji następującego programu:

```
begin array A1, A2[1:10,1:10], B[1:15,1:15];
procedure S(a,n,s);
array a; integer n; real s;
  begin integer k;

    s:= 0;
    for k:= 1 step 1 until n do
    s:= s + a[k,k]
  end procedury S;
L:input(A1, A2, B);
  begin real a1,a2,b;

    S(A1,10,a1);
    S(A2,10,a2);
    S(B,15,b);
    output(⊀-n.ddd₁₀+d⊁,outcr,a1,outsp(2),a2,outsp(2),b)
```

```
   end bloku wewnetrznego;
   if kbon then go to L
end programu
```

	Ilość rezerwowanych miejsc
Wejście do bloku zewnętrznego:	2
Tablice, pierwszy segment: 2+2+200+1=	205
Tablice, drugi segment: 1+2+225+1=	229
Procedura:	1
Parametry formalne:	3
Etykieta:	1
- przed wejściem do bloku wewnętrznego:	441
Wejście do bloku wewnętrznego:	2
Zmienne proste:	3
- przed wywołaniem procedury:	446
Wejście do treści procedury:	3
Zmienna prosta /lokalna dla treści/:	1
- w czasie wykonywania instrukcji procedury:	450
Wyjście z treści procedury:	-4
- po pierwszym wywołaniu procedury:	446

/Następne wywołanie procedury powodują analogiczne zmiany w ilości rezerwowanych miejsc/.

- po trzecim wywołaniu procedury:	446
Wyjście z bloku wewnętrznego:	-5
- w czasie wykonywania instrukcji "jeśli"	441

Maksymalna ilość rezerwowanych miejsc wynosi więc 450 /podczas każdorazowego wywoływania procedury S/.

31 - 2. Współpraca z bębnem jako pamięcią pomocniczą

Bęben może być wykorzystywany w programie jako pamięć pomocnicza dla przechowywania danych. Przesyłanie tablic z danymi odbywa się za pomocą standardowych procedur to

drum /na bęben/, from drum /z bębna/ oraz zmiennej standardowej drumplace /miejsce na bębnie/.

31 - 2.1. Wstępne informacje

1. Wartością zmiennej drumplace jest numer miejsca na bębnie.
2. Zmienną tę można wyobrazić sobie jako strzałkę, wskazującą w momencie rozpoczęcia realizacji programu pierwsze wolne miejsce na bębnie /stąd: długość przetłumaczonego programu = 12799 - drumplace/.
3. Każdorazowe zapisanie tablicy na bębnie lub odczytanie jej z bębna do PO /wywołanie procedury to drum lub odpowiednio from drum/, powoduje zmniejszenie wartości drumplace o ilość przesyłanych słów. Interpretować to możemy jako automatyczne przesunięcie strzałki w górę /w kierunku malejących numerów miejsc/.
4. Aby tablica, uprzednio zapisana na bębnie od miejsca np. Adres mogła być odczytana, wartość zmiennej drumplace musi wskazywać /tuż przed wywołaniem procedury from drum/ miejsce Adres. Należy ją więc "nastawić" na to miejsce instrukcją podstawienia, np.

   drumplace:= Adres
5. to drum jest procedurą z jednym parametrem, będącym nazwą tablicy.
6. Wywołanie procedury to drum powoduje przesłanie z PO na bęben /skopiowanie/ wartości zmiennych indeksowanych tablicy, której nazwa jest parametrem aktualnym procedury.
7. Przesłana tablica będzie zapisana na bębnie od miejsca wskazanego wartością drumplace /po przesłaniu tablicy drumplace zmieni swoją wartość, por. p.3/. Rozłożenie elementów przesyłanej tablicy na bębnie objaśnimy na przykładach. Niech wartością drumplace będzie 10000, a tablica ma wymiary A [1:2,1:3] . Wtedy po wykonaniu instrukcji procedury

   to drum (A)

otrzymamy:

```
                                        9993
/nowa wartość drumplace/ ------------> 9994
                                        9995  A [1,1]
                                        9996  A [1,2]
                                        9997  A [1,3]
                                        9998  A [2,1]
                                        9999  A [2,2]
/wartość drumplace przed przesyłaniem/--> 10000  A [2,3]
```

8. Jeżeli przesyłana tablica jest więcej niż dwuwymiarowa, rozłożenie elementów dokonywane jest w analogiczny sposób /ostatni na miejscu wskazanym strzałką drumplace i następnie kolejno, do góry/.
9. Procedura from drum działa analogicznie jak to drum, z tym, że na elementy tablicy /zadeklarowanej w PO/,której nazwa jest parametrem aktualnym procedury, zostają podstawione wartości uprzednio zapisane w bębnie.
10. Podstawienie odbywa się od końca, tzn. jeżeli chcemy, by tablicy A /przykład z p.7/ nadane były jej poprzednie wartości, strzałkę drumplace należy uprzednio nastawić na 10000:

```
drumplace := 10000;
from drum (A)
```

Jeżeli zaś chcemy, by wartościami zmiennych indeksowanych tablicy B [1:3] były wartości zmiennych pierwszego wiersza tablicy A, wówczas należy wykonać instrukcje:

```
drumplace := 9997;
from drum (B)
```

Równoważne jest to podstawieniom:

```
B[3]:= A[1,3]; B[2]:= A[1,2]; B[1]:= A[1,1]
```

31 - 2.2. Procedury to drum, from drum

Uzupełniamy obecnie podany wyżej opis tych procedur standardowych.

1. Procedury to drum i from drum są procedurami funkcyjnymi typu integer.
2. Wywołanie którejś z tych procedur powoduje /oprócz efektów opisanych w 31-2.1/ podstawienie na odpowiedni nazewnik funkcyjny wartości równej ilości przesyłanych elementów tablicy, ze znakiem minus.
3. Procedury to drum oraz from drum wywoływane są często nazewnikiem funkcyjnym, np.

zmiana := to drum (A1)×3

31 - 2.3. Przykłady

```
begin integer i,aplace;
array Ast[1:100];

  aplace:= drumplace;
  for i:= 1 step 1 until 10 do

  begin
    input(Ast);
    to drum(Ast)
  end

      przeslania na beben 10-ciu tablic. K-ta tablica zapisana jest
      od miejsca: aplace - (K-1) × 100;


  drumplace:= if inone > 1 then aplace -400 else aplace;
  from drum(Ast);

      comment jezeli wczytana z tasmy liczba jest wieksza od jednosci,
      wtedy na Ast podstawiane beda wartosci 5-tej tablicy przeslanej
      na beben, w przeciwnym razie - pierwszej;
      . . . . .
    end
```

1. Z diagramu podanego na początku rozdziału wynika, ile miejsc na bębnie możemy użyć do przysyłania tablic w programie. Jeżeli ilość ta jest niewystarczająca, należy zapisać w inny sposób translator - omawiamy to w paragrafie 31-4.
2. W poprzednich punktach paragrafu opisaliśmy sposób korzystania z bębna w obecnej wersji GIER ALGOLu III. Autorzy GIER ALGOLu zastrzegli jednak, że rozmieszczenie elementów tablic na bębnie może być w różnych wersjach translatora różnie realizowane. Jednakże w każdej z tych wersji, w wyniku wykonania następującego ciągu instrukcji:

```
Miejsce A := drumplace;
to drum (A) ;
drumplace := Miejsce A;
from drum (A)
```

wartości zmiennych indeksowanych tablicy A nie zostaną zmienione.

31 - 3. Przykłady

Podamy przykład procedury GIER ALGOLu III, korzystającej z bębna:

```
procedure MULT1(p,q,r,Aplace,Bplace,Cplace);
value p,q,r,Aplace,Bplace,Cplace;
integer p,q,r,Aplace,Bplace,Cplace;

comment MULT1 mnozy macierz A[1:p,1:q] zapisana na bebnie od miejsca
Aplace przez macierz B[1:q,1:r] zapisana na bebnie od miejsca Bplace.
Wynik C[1:p,1:r] umieszczony bedzie rowniez na bebnie, od miejsca
Cplace. W PO w czasie wywolywania procedury znajdowac sie bedzie ma-
cierz B oraz wiersz macierzy A i wiersz C. Zatem q × r + q + r ≤ 650,
co w przypadku macierzy kwadratowych oznacza, ze rzad < 25;
```

```
begin integer i,j,k; real s;
array P[1:q,1:r],WA[1:q],WB[1:r];
  drumplace:= Eplace;
  from drum(P);
  comment przeslanie B, beben -> PO;
  for i:= 1 step 1 until p do
  begin
    drumplace:= Aplace - (i-1) × q;
    from drum (WA);
    comment przeslanie i-tego wiersza macierzy A, beben -> PO;
    for j:= 1 step 1 until r do
    begin
      s:= 0;
      for k:= 1 step 1 until q do
      s:= s + WA[k] × P[k,j];
      WB[j]:= s
    end obliczone zostaly elementy i-tego wiersza macierzy C;
    drumplace:= Cplace-(i-1) × r;
    to drum (WB);
    comment przeslanie i-tego wiersza macierzy B, PO-> beben;
  end petli i
end procedury MULT1
```

31 - 4. Uwagi

Translator GIER ALGOL III może być zapisany na bębnie na kilka sposobów - najczęściej stosowane podaliśmy na diagramie /początek rozdziału/. Wspomnimy krótko o dwu innych możliwościach:

1. W przypadku programów, które nie mieszczą się na 134 ścieżkach bębna /są zbyt obszerne/, lub które wymagają więcej miejsca dla procedur to drum i from drum, translator zapisany jest w tzw. "skrócony" sposób, prawie od

początku bębna /niszcząc HELP SYSTEM/. Pozwala to zaoszczędzić 36 ścieżek, czyli 1440 miejsc.

2. Jeżeli program jest na tyle obszarny, że wymaga w czasie tłumaczenia więcej niż 134+36 ścieżek, to należy użyć translatora, tzw. "z taśmy". Wówczas na bębnie zapisana jest mniej niż połowa translatora - reszta wprowadzana jest sukcesywnie z taśmy. Szybkość tłumaczenia programu ulega wyraźnemu zmniejszeniu, ale na czas trwania translacji zyskuje się dla tłumaczonego programu 78 ścieżek bębna.

Informacja o tym, że program wymaga więcej miejsca na bębnie, podawana jest automatycznie przez translator w czasie tłumaczenia lub realizacji programu - omawiamy to w następnym rozdziale.

32. TŁUMACZENIE PROGRAMU I JEGO REALIZACJA

W rozdziale tym podamy końcowe informacje o programowaniu w GIER ALGOLu. Czytelnikom pragnącym zapoznać się bardziej szczegółowo z GIER ALGOLem poleca się przestudiowanie publikacji /2/.

32 - 1. Wprowadzanie programu

Program, uprzednio wyperforowany na 8-mio kanałowej taśmie papierowej, wprowadza się do pamięci maszyny za pomocą czytnika. W czasie wczytywania programu prowadzona jest następująca kontrola:

1. Kontrola parzystości /analogicznie, jak w przypadku uniwersalnego mechanizmu wejścia, por. 29-2.2/.
2. Symbolom PUNCH OFF i PUNCH ON nadane jest specjalne znaczenie. W zależności od klawisza naciśniętego na monitorze, ciąg symboli, zawartych między symbolem PUNCH OFF i pierwszym następującym po nim PUNCH ON będzie włączony do programu, lub pominięty /w przypadku wczytywania danych procedurą podlegającą uniwersalnemu mechanizmowi wejścia, ciąg taki jest zawsze ignorowany, por. 29-2.3/.
3. W czasie wczytywania informacji z taśmy, prowadzona jest kontrola poprawności programu pod względem syntaktycznym - wychwytywane są błędy tzw. przebiegu 1. Omówimy to dokładnie w następnym paragrafie.
4. Każdy program GIER ALGOLu musi być zakończony średni-

kiem - wczytywanie programu kontynuowane jest aż do momentu natrafienia na średnik po ostatnim end.

32 - 2. Tłumaczenie programu

Tłumaczenie programu z GIER ALGOLu na język wewnętrzny przebiega w 8-miu fazach. W każdej fazie /zwanej przebiegiem/ prowadzona jest kontrola, mająca na celu wykrycie błędów syntaktycznych określonego typu. Analiza formalnej poprawności programu prowadzona jest do końca, tzn. wykrycie jakiegoś błędu nie przerywa dalszych poszukiwań błędów. Znalezienie błędu sygnalizowane jest informacją:

line < numer linii > < krótki opis błędu >

drukowaną na czerwono na monitorze. Informacja o pierwszym błędzie wykrytym w danym przebiegu, poprzedzana jest dodatkowo numerem przebiegu i kropką /drukowane są na czarno/. Na przykład:

```
3. line 26 - delimiter
   line 31   operand
   line 37   termination
5. line 3  - declar.
   line 28   value
```

Numer linii odpowiada numerowi wiersza programu, w którym pojawił się błąd, przy czym:

- linia z pierwszym symbolem begin programu ma numer 0
- jeżeli błędna konstrukcja pojawia się blisko początku lub końca wiersza, numer tego wiersza może być zmieniony o jeden,
- przy informacji o pewnym typie błędu wykrywanym w przebiegu 5, numer linii może wskazywać całkiem inny wiersz /opiszemy to poniżej/,
- przy błędach wykrywanych w przebiegu 7 i 8 wskazywana jest zawsze linia 0.

32 - 2.1. Opisy błędów

W następnych punktach wymienimy wszystkie informacje sygnalizujące błąd /wypisywane na monitorze/, oraz podamy krótkie objaśnienia odnośnie rodzaju wskazanego błędu.

32 - 2.1.1. Przebiegi 1 - 8

program too big

Program zbyt obszerny, nie mieści się na wolnych miejscach bębna. Wskazówka: zastosować, o ile to możliwe, wersję "z taśmy" oraz "skróconą" translatora /por. 31-4/.

32 - 2.1.2. Przebieg 1

character

Układ dziurek, który pojawił się na taśmie, nie odpowiada żadnemu symbolowi GIER ALGOLu.

compound

Po ciągu znaków reprezentujących kilka pierwszych znaków symbolu złożonego /por. 28-1, p. 2/, nie pojawił się na taśmie kolejny znak symbolu, ale jakiś inny układ dziurek. Na przykład: intger until)<improper>

Po konstrukcji) < ciąg liter > nie następuje:(Na przykład: procedure P (a) b; outsp (2) outsp (3)

comment

Symbol comment nie jest poprzedzony ani symbolem begin ani ;. Na przykład: a:=7 comment zle;

string

Symbol złożony ⊀ nie jest umieszczony ani na początku wzorca, ani na początku tekstu /por. 30-7.2,9.1/.

32 - 2.1.3. Przebieg 2

too many identifiers

W programie występuje za dużo nazw lub nazwy są za długie. Wskazówka: zredukować ilość różnych nazw korzystając ze struktury bloków.

32 - 2.1.4. Przebieg 3

-delimiter

Brak ogranicznika między argumentami /nazwami, liczbami, wartościami logicznymi, łańcuchami, złożonymi wyrażeniami w nawiasach/. Na przykład: 7.3 sin (5) 4 true r.77 a ≮ < string ≯

operand

1. Argument występuje w nieprawidłowej konstrukcji. Na przykład: 7:= begin true
2. Argument został pominięty. Na przykład: a:= [i]

delimiter

1. Ogranicznik występuje w nieprawidłowej konstrukcji. Na przykład: begin r/i:= if go to if for
2. Operator dwuargumentowy nie jest poprzedzony argumentem. Na przykład: i:= × a;

-operand

Opuszczony został argument na końcu konstrukcji. Na przykład: r:=r/;

termination

Nieprawidłowe użycie nawiasów. Na przykład:
r [i) begin r:=a+b, p (i,r;

number

1. Konstrukcja, której pierwsze symbole odpowiadają składni liczby, nie jest zakończona prawidłowo. Na przykład: 20.; $17._{10}-3$
2. Moduł liczby przekracza dopuszczalny zakres. Na przykład: $7_{10}170$

stack

Konstrukcja nawiasowa: begin begin ... begin zawiera za dużo symboli begin

32 - 2.1.5. Przebieg 4

stack

Zbyt duża ilość w jednym bloku nazw różnego typu i rodzaju, elementów przełącznika oraz wywoływań procedur.

32 - 2.1.6. Przebieg 5

+declar

Ta sama nazwa deklarowana jest dwukrotnie w jednym bloku lub pojawia się dwa razy w wykazie parametrów formalnych /etykiety lokalne dla danego bloku interpretujemy jako zadeklarowane w tym bloku/.

+specif.

Ta sama nazwa jest dwukrotnie specyfikowana w jednym nagłówku deklaracji procedury.

-declar.

Nazwa została użyta w miejscu, w którym nie jest znana /tzn. nazwa ta nie była zadeklarowana, lub miejsce użycia nazwy jest poza zasięgiem jej deklaracji/. Numer linii poprzedzający ten opis błędu, może wskazywać inny wiersz w dwu następujących przypadkach:

1. Nazwa pojawia się w parametrze aktualnym. Wtedy będzie wskazany wiersz, w którym występuje lewy nawias instrukcji procedury lub nazewnika funkcyjnego.
2. Nazwa jest elementem przełącznika. Wtedy wskazany będzie wiersz zawierający begin tego bloku, w którym dany przełącznik jest zadeklarowany.

-specific.

Opuszczona została specyfikacja parametru formalnego.

-formal

Specyfikowana jest nazwa, która nie była wymieniona w wykazie parametrów formalnych.

value

Wymieniony został w zbiorze wartości parametr formalny, który zgodnie z jego specyfikacją, nie może być wołany przez wartość.

stack

Za duża ilość nazw deklarowanych jednocześnie.

32 - 2.1.7. Przebieg 6

subscript 704

Ilość wskaźników zmiennej indeksowanej nie zgadza się z wymiarami podanymi w deklaracji odpowiedniej tablicy.

proc. call or ident. 790

Nazwa bezpośrednio poprzedzająca lewy nawias, (,nie może być użyta w celu wywołania procedury albo ze względu na nieodpowiednią ilość parametrów po nawiasie, albo ze względu na błędną konstrukcję.

proc. call or ident. 840

Nazwa procedury pojawia się w sposób nie zgodny z deklaracją tej procedury.

type <numer błędu>

Numer błędu określa bardziej szczegółowo jakiego rodzaju błąd został wykryty. Objaśniamy to poniżej. W tabeli opis:<nar> ::= <nieodpowiedni argument> oznacza argument, który jest nieodpowiedniego typu lub rodzaju w stosunku do konstrukcji, w której został zastosowany. Nazwy użyte w przykładach oznaczają obiekty opisane poniższymi deklaracjami:

```
integer i; real r; Boolean b; array a1[1:10],a2[2:4,4:6];
switch s:= L,L2; procedure p0;; procedure p1(f); real f;;
```

Numer błędu	Błędne konstrukcje	Przykłady
576	+<nar>|-<nar>|x<nar>|/<nar>|/\<nar>	+s /L
582	:<nar>	:r
585	<zmienna boolowska>:=<nar>	b:=r
590	< <nar>|≤ <nar>|= <nar>|≥ <nar>|> <nar>|≠ <nar>	=b
593	∧ <nar>|∨ <nar>|≡ <nar>|-,<nar>	∨L
596	<nar> <operator dwuargumentowy> <nar>	i∨a1 b=s
599	<zmienna rzeczywista>:=<nar>	r=:=s
604	<zmienna całkowita>:=<nar>	i:=p0
616	abs(<nar>) |arctan(<nar>) |cos(<nar>) | entier(<nar>) |exp(<nar>) |ln(<nar>) | sign(<nar>) |sin(<nar>) |sqrt(<nar>) | outcopy(<nar>) |outsp(<nar>) |outchar(<nar>) | writecopy(<nar>) |writechar(<nar>)	cos(a2) ln(r=i) sqrt(s) outsp(b)
630	go to <nar>|switch sw:=<nar>	go to b
633	; <nar>;	;r;
640	<nar>[	i[
649	from drum(<nar>) |to drum(<nar>)	to drum(r)
657	then <nar>	then s
677	<nar> else <nar>|<nar> else if...then <nar>	L else b
686	:=<nar>:=	r:=b:=b∨b
690	<nar> step|<nar> until|for...<nar> | for...<nar>do	i=r step
697	<nar>]	p0]
714	<nar>:=	p0:=
725	<nar> then|while <nar> do	if r then
728	for <nar>:=	for a1:=
732	<nieodpowiedni wskaźnik>	a1[L]
735	<nar>: (W deklaracji tablicy)	array q[b:1]

32 - 2.1.8. Przebieg 7

number

Wartość wyrażenia arytmetycznego, w którym argumentami są tylko liczby, przekracza dopuszczalny zakres liczb. Na przykład: 1/0 $7_{10}\ 35 \times 9.2_{10}\ 135$

32 - 2.1.9. Przebieg 8

stack

Za duża ilość etykiet, konstrukcji: for ... do for ... do for ... do ... for ... do oraz konstrukcji: if ... then ... else if ... then ... else ... if ... then ... else

32 - 2.2. Uwagi

1. Istnieje szereg błędów formalnych języka, które nie są wykrywane w czasie tłumaczenia.
2. Program poprawny pod względem formalnym może zawierać błędy merytoryczne, które spowodują bądź przedwczesne przerwanie realizacji programu /omówimy to w następnym paragrafie/, bądź inną realizację programu, niż to przewidywał programista.
3. W GIER ALGOLu III istnieje możliwość wyprowadzenia z pamięci maszyny przetłumaczonego na język wewnętrzny programu. Jest to szczególnie przydatne w przypadkach programów często eksploatowanych.

32 - 3. Sytuacja run

Jeżeli w trakcie tłumaczenia translator nie wykryje żadnego błędu formalnego programu, na monitorze drukowana jest informacja

run

Oznacza ona, że program jest przetłumaczony z GIER ALGOLu

na język wewnętrzny maszyny i gotowy do realizacji. Realizacja programu rozpocznie się po naciśnięciu klawisza na monitorze. Wybierając odpowiedni klawisz, możemy w tym momencie dokonać zmian w programie, dotyczących urządzeń, przez które będą wyprowadzane teksty i wyniki programu. Istnieją następujące cztery możliwości:

- wyprowadzanie będzie zgodne z rodzajem instrukcji wyjścia użytych w programie /instrukcje write-, spowodują drukowanie na monitorze, instrukcje out-, na perforatorze/,
- wyprowadzanie odbywać się będzie tylko na monitor /równoważne jest to zamianie wszystkich instrukcji out- programu, na write-/,
- wyprowadzanie odbywać się będzie tylko na perforator /równoważne jest to zamianie wszystkich instrukcji write- programu, na out-/,
- wyprowadzanie odbywać się będzie równocześnie na monitor i perforator.

32 - 4. Realizacja programu

Normalna /tzn. bez awarii technicznych lub ingerencji z zewnątrz /realizacja programu zakończona jest zawsze wydrukowaniem pewnej informacji na monitorze. Po całkowitym wykonaniu programu /tzn. po przejściu przez końcowe end/, drukowaną informacją jest

end .

Każde inne zakończenie normalnej pracy programu spowodowane jest błędami merytorycznymi/ np. nieodpowiednie argumenty funkcji standardowych, rezerwowanie zbyt dużej ilości miejsc w pamięci itd./. Wymienimy poniżej, jakiego typu błędy wykrywane w czasie realizacji programu, powodują przerwanie jego pracy:

Alac Ilość miejsc aktualnie rezerwowanych w PO przez konstrukcje ALGOLu, przekracza dopuszczalny limit /por. 31-1/.

array — Na początku aktualnie realizowanego bloku pojawia się deklaracja tablicy opisująca tablice o zbyt dużej ilości elementów lub w którejś z par granicznych granica dolna wskaźnika jest większa od odpowiadającej jej granicy górnej.

drum alas — Procedura standardowa to drum wywoływana jest z wartością drumplace będącą poza dopuszczalnym zakresem /por. 31-2./ lub from drum wywoływana jest z drumplace, o wartości leżącej na zewnątrz przedziału domkniętego [ilość elementów przesyłanej tablicy, 12799] .

exp — Zakres liczb rzeczywistych przekroczony jest przez wartość nazewnika funkcyjnego exp (W) lub przez wartość wyrażenia a↑b, gdzie b jest typu real.

index — Wskaźnik zmiennej indeksowanej ma wartość leżącą poza zakresem odpowiedniej pary granicznej podanej w deklaracji odpowiedniej tablicy.

ln — Funkcja standardowa ln wywoływana jest z ujemnym argumentem, albo w wyrażeniu arytmetycznym występuje a↑b, gdzie a<0 oraz b jest typu real.

spill — Wynik operacji arytmetycznych przekracza zakres liczb w GIER ALGOLu. Ponieważ operacja potęgowania z wykładnikiem typu integer wykonywana jest najpierw dla modułu wykładnika, zatem np. a↑(-1053) spowoduje sygnał spill, mimo, że końcowy rezultat jest równy zeru.

sqrt — Funkcja standardowa sprt wywoływana jest z ujemnym argumentem.

32 - 4.1. Uwagi

1. Po każdorazowym zakończeniu realizacji programu/tzn. po wydrukowaniu dowolnej z w/w informacji/ można wrócić do sytuacji run, bez ponownego tłumaczenia programu.

2. W dowolnym momencie pracy programu można zatrzymać jego realizację i powrócić do sytuacji run.

32 - 5. Oszacowanie czasu realizacji programu

Niżej podana tabela umożliwi przybliżone oszacowanie czasu trwania realizacji programu. Należy jednak pamiętać, że na czas pracy programu GIER ALGOLu istotny wpływ ma ilość miejsc rezerwowanych w pamięci operacyjnej maszyny /por. 31-1/.

OPERACJA	PRZYKŁAD	CZAS WYKONYWANIA W MILISEKUNDACH
Dodawanie, odejmowanie	a + b	0.12
Mnożenie	a × b	0.18
Dzielenie	a / b	0.21
Podnoszenie do kwadratu	a ↑ 2	0.18
Podnoszenie do sześcianu	a ↑ 3	0.4
Potęgowanie, wykładnik typu integer	a ↑ i	
abs(wykładnika) = 1		3.8
10		5.5
100		8
1000		10
10000		12
100000		14
1000000		16
Potęgowanie, wykładnik typu real	a ↑ r	12
Zmienne indeksowane		
1 wskaźnik	A[i]	0.9
2 wskaźniki	B[i,j]	1.2
3 wskaźniki	C[i,j,k]	1.5
Element rodzaju step-until ze stałym krokiem i prostą górną granicą; każda pętla	step 1 until n	0.6
Blok ze zmiennymi prostymi	begin real a; end	1.4
Blok z deklaracją tablicy	begin array a[1:10]; end	3.0
Odwołanie sie do parametrów formalnych wołanych przez nazwę		
Parametrem aktualnym jest:		
zmienna, liczba		0.4
wyrażenie		3.2
nazwa tablicy		0.0
nazwa przełącznika		0.0
nazwa procedury		0.0

OPERACJA	PRZYKŁAD	CZAS WYKONYWANIA W MILISEKUNDACH
Instrukcja podstawienia		
a:= 0		0.05
a:= b		0.1
Instrukcja skoku		
prosta, wewnątrz danego bloku	go to A; A:	0.8
do nazewnika przełącznika	go to S[i]	2.1
Warunek "jeśli" z prostą relacją	if a > b then	0.3
Wywołanie deklarowanej procedury /której treść jest instrukcją pustą/		
bez parametrów	P;	3.8
1 parametr	Q(a);	4.7
2 parametry	R(a,b);	5.2
3 parametry	S(a,b,c);	5.5
Wywołanie procedur standardowych		
abs	abs(x)	0.17
arctan	arctan(x)	6.6
cos	cos(x)	6.0
entier	entier(x)	0.4
exp	exp(x)	5.8
ln	ln(x)	5.6
sign	sign(x)	3.2
sin	sin(x)	5.8
sqrt	sqrt(x)	6.2
kbon	b:= kbon	3.5

32 - 5.1. Uwagi

Z przeglądu powyżej podanej tabeli wynika, że można w sposób istotny zmniejszyć czas rozwiązywania zagadnienia przez maszynę, ograniczając posługiwanie się zmiennymi indeksowanymi oraz procedurami. Zaprogramujmy przykładowo rozwiązanie następującego problemu: Znaleźć iloczyny skalarne A×A, A×B, A×C, gdzie A,B,C są wektorami n elementowymi.

```
begin integer k,n;
  n:= inone;
```

```
  begin array A,B,C[1:n]; real aa,ab,ac;
    input(A,B,C); aa:= ab:= ac:= 0;
    for k:= 1 step 1 until n do
    begin
      aa:= aa+A[k]/2;
      ab:= ab+A[k]×B[k];
      ac:= ac+A[k]×C[k]
    end petli;
    output(‡-n.dddddd₁₀-d‡,outcr,aa,outsp(5),ab,outsp(5),ac)
  end bloku z tablicami
end programu
```

Dokonując następującej zmiany w programie:

```
. . . . .
begin array A,B,C[1:n];  real aa,ab,ac,zysk;
. . . . .
  for k:= 1 step 1 until n do
  begin
    zysk:= A[k];
    aa:= aa+zysk/2;
    ab:= ab+zysk×B[k];
    ac:= ac+zysk×C[k]
  end petli;
  . . . . .
```

zaoszczędzimy $2 \times 0.9 \times n$ milisekundy /zmienna indeksowana/ tracąc $0.1 \times n$ milisekundy /instrukcja podstawienia/ oraz jedno miejsce w PO na dodatkową zmienną, zysk.

ODPOWIEDZI DO CWICZEN

.1) przecinek zamiast kropki 2) -2. nie jest <decimal number> 4),6),8) nie mogą wystepować symbole × n / 9) dwie kropki

2.1) 729 300 000 2) 9812 3) 1000 4) -0.000 001 834 5) -0.000 001 6) [illegible] -4800

3.1) $17_{10}3$ 2) $_{10}3$ 3) $-134_{10}-5$ 4) 1.00024 5) -0.00200 41298 6) 170

4.2),4),5),6),8),10),12),13),15),17) nie spełniają definicji nazwy

5.1) A+(s-t)/v↑2 2) (x+y)↑3 3) x+y↑3 4) v×(-z) 5) p↑q/r↑(s+t) 6) a↑(b+c↑d) 7) 1+x↑(-2)+a×y↑(-1+c) 8) $(a/_{10}4+5_{10}-5)\times c$

6.3) po symbolu $_{10}$ musi następować <integer> 5) dwa operatory nie mogą ze sobą sąsiadować

7.1) 32/3 2) 1/3 3) 625 4) 1 5) 1 6) 1 7) 0 8) -7

8.1) ciąg symboli po prawej stronie nie jest wyrażeniem arytmetycznym w ALGOLu - brak operatorów między) i b oraz E) i (F 2),4), 5) po lewej stronie symbolu := może występować tylko zmienna 3) argumentem funkcji sqrt może być tylko liczba nieujemna

9.1) a=25.87 b=7 p=25.87 q=27.27 2) r1=25 ra=4 rb=12 n=2 i=6 j=2

10. zmienne indeksowane a[1],a[2],a[3] należy zadeklarować jako elementy tablicy, tzn. array a[1:3],dolna granica wskaźnika nie może być większa od górnej,[0:10,5] nie jest [<bound pair list>], przed deklaracją integer brak średnika

11. p=0.5 q=0.25 SUM=5 n=4

12.1) false true false 2) false true false 3) true true true 4) true false true

13. ra=4 rb=12.5 ia=5 ba=false bb=true

14. w instrukcji przy etykiecie: E1 zmienne po lewej stronie są różnych typów (A,e); E3 po warunku "jeśli" występuje if; E4 wyrażenie po prawej stronie znaku + nie jest < term >; E5 wyrażenie po prawej stronie znaku V nie jest <Boolean factor> oraz wyrażenie występujące po else nie jest <Boolean expression>

15.1) false 2) false 3) true

16.1) true 2) false 3) false

17.1) a=-9 b=26 2) a=13/15 b=-17/15 p=true

18.przykładowe rozwiązanie:

```
begin integer n,s;
  input(n); s:=n;
  if n<1 ∨ n>10 then go to B;
A:if n>2 then n:=n-1 else go to C;
  s:=s×n; go to A;
B:s:=0;
C:output(⟨+d.dddd₁₀+dd⟩,s)
end
```

19.2) p=false a=9 b=7 (po wyjściu z bloku wewnętrznego b ma wartość nieokreśloną) 3) a=-4 b=-8.5 c=-1.5 w=2

20.W=-8 S=-9 B=13 C=7

21.i:-4,3,2,1,0,0,400,410,420,430,2,2,2 Q:0,-5,-5,-10,-24,-55, -118,155,310,219,-374,-2419,-6512

22.5,-35,-69,15,25,39,57,79,105

23.

```
begin integer i,p;
real iloczyn;
comment w bloku tym nie wolno deklarowac wektorow;
  input(p);
  begin array V1,V2[1:p];
    iloczyn:=0;
    for i:=1 step 1 until p do
    iloczyn:=iloczyn + V1[i]×V2[i]
  end bloku z deklaracja wektorow;
  output(⟨+d.dddd₁₀+dd⟩,iloczyn)
end
```

25.$-4.8000_{10}+1$

26.

```
procedure SREDNIE(A,G,H,X,n,Brak H);
comment jezeli srednia H nie istnieje, to nastepuje skok do
        etykiety Brak H;
value n;
integer n; array X;
real A,G,H; label Brak H;
begin integer i;
Boolean Nie zero;
  A:=H:=0;
  G:=1;
  Nie zero:=true;
  for i:=1 step 1 until n do
  begin
    A:=A+X[i];
    G:=G×X[i];
    if X[i]=0 then
    Nie zero:=false;
    if Nie zero then H:=H+1/X[i]
  end;
```

```
    A:=A/n;
    G:=sign(G)×abs(G)↑(1/n);
    if -,Nie zero ∨ H=0 then go to Brak H;
    H:=n/H
  end
```

27. I) +3.0000 II) +6.0000 III) +1.0000 +3.0000 +3.0000

28. gdy nazwa a (lub b) jest nazwą procedury funkcyjnej bez parametrów , w której treści zmieniana jest wartość zmiennej b (lub odpowiednio a).

29.

7:	.ooo	(:	o.	t: o . oo
PUNCH OFF:	o o.ooo	Æ:	ooo .	: o.oo
UPPER CASE:	oooo.o			

SKOROWIDZ ZMIENNYCH METAJĘZYKOWYCH ALGOLu 60

BIBLIOGRAFIA

/1/ J.W.Backus i inni - Revised Report on the Algorithmic Language ALGOL /red. P. Naur/, Regnecentralen, Copenhagen /1962/;

/2/ H.Christensen i inni - A Manual of GIER ALGOL III /red. P.Naur/, Regnecentralen, Copenhagen /1962/;

/3/ J.W.Backus i inni - Język algorytmiczny ALGOL - 60 /przekład S.Paszkowskiego/, Centrum Obliczeniowe PAN Warszawa /1962/;

/4/ P.Naur - A Course of ALGOL 60 Programming,Regnecentralen, Copenhagen /1962/;

/5/ Ch.Andersen - ALGOL DASK-ALGOL GIER-ALGOL,Regnecentralen Akademisk Forlag /1963/;

/6/ Ch.Andersen - An Introduction to ALGOL 60, Addison - Wesley Publishing Company, London /1964/;

7/ M.I.Agiejew - Osnowy ałgoritmiczeskowo jazyka ALGOL-60, Wyczislitielnyj Centr AN SSSR, Moskwa /1964/;

/8/ Ch.Gram i inni - Opgaver til ALGOL Kursus /GIER ALGOL/ Regnecentralen /1964/;

/9/ H.Vilstrup - An ALGOL Training Manual, Regnecentralen Copenhagen /1963/.

SPIS TREŚCI

Cena zł 15.–