

3. INSTRUKCJE

3.1. Wiadomości ogólne o instrukcjach

Program napisany w ALGOLu ma za zadanie przesłanie maszynie cyfrowej niezbędnych informacji, jakie działania ma wykonać. Najmniejszą jednostką informacyjną tego rodzaju jest instrukcja (statement; czytaj: stejtmnt). Instrukcje zapisujemy kolejno jedna po drugiej, zgodnie z przewidzianą kolejnością ich wykonania przez maszynę. Ponieważ jednak wprowadzamy nieraz do programu alternatywy, program rozwidla się na przypadki, których nie można zapisać z pełnym utrzymaniem kolejnego zapisu dla kolejno wykonywanych działań. Jesteśmy wtedy zmuszeni dokonywać w programie skoków, za pomocą tzw. instrukcji skoku (go to statement; czytaj: goł tu stejtmnt). Samo rozwidlenie programu powstaje na skutek pojawienia się tzw. instrukcji warunkowej (conditional statement; czytaj: kondyszynł stejtmnt) albo wyrażenia alternatywnego w instrukcji skoku. Aby maszynie dać informację, do którego miejsca w programie ma skoczyć, niektóre instrukcje poprzedzamy tzw. etykietą (label; czytaj: lejbl).

Najczęściej spotykaną instrukcją jest instrukcja podstawienia (assignment statement; czytaj: esajnmnt stejtmnt), powodująca obliczenie wartości wyrażenia i podstawienie jej na wskazaną zmienną.

Najrzadziej, bo tylko w wyjątkowych sytuacjach, stosuje się tzw. instrukcję pustą (dummy statement; czytaj: dami stejtmnt), nie powodującą żadnego działania maszyny.

Osobny rodzaj instrukcji stanowi wywołanie procedury niefunkcyjnej (procedure statement; czytaj: prosjidżer stejtmnt), o czym będzie mowa szczegółowo w paragrafie 4.3.

Instrukcje łączymy w większe całości: instrukcje złożone (compound statements; czytaj: kompajn stejtmnts) lub bloki (blocks; czytaj: bloks).

Do powtórzenia jednej i tej samej instrukcji dla zmieniających się parametrów, służy tzw. instrukcja "dla" (for statement; czytaj: for stejtmnt).

W ALGOLu wprowadza się następujący podział:

Przez instrukcję podstawową rozumiemy jedną z następujących:

- instrukcja podstawienia,
- instrukcja skoku,
- instrukcja pustą,
- wywołanie procedury niefunkcyjnej.

Przez instrukcję bezwarunkową rozumiemy:

- instrukcję podstawową,
- instrukcję złożoną,
- blok,

Przez i n s t r u k c j ę będziemy rozumieć:

- instrukcję bezwarunkową,
- instrukcję warunkową,
- instrukcję "dla".

Przechodzimy teraz do szczegółowego omówienia powyższych rodzajów instrukcji.

3.2. Instrukcje podstawienia

Instrukcje podstawienia mają następującą budowę:

zmienna := zmienna := ... zmienna := wyrażenie.

Wyrażenie po prawej stronie ostatniego znaku := nazywamy prawą stroną instrukcji podstawienia. Pozostałą część nazywamy lewą stroną tej instrukcji. Każda instrukcja podstawienia jest wykonywana w trzech krokach:

- 1) obliczenie indeksów dla zmiennych wymienionych po lewej stronie instrukcji kolejno od lewej strony,
- 2) obliczenie wartości wyrażenia po prawej stronie,
- 3) podstawienie wartości tego wyrażenia kolejno na wszystkie zmienne wymienione po lewej stronie.

Kolejność wykonania powyższych kroków musi być ściśle przestrzegana, bo inaczej można by otrzymać dwuznaczność. Należy podkreślić, że w momencie podstawiania wartości na zmienne, indeksy zmiennych są już ustalone, na skutek uprzedniego wykonania kroku 1.

Znak := nie jest następstwem znaków : i =, lecz jednym niepodzielnym znakiem. Jest on niesymetryczny, gdyż cała instrukcja nie jest symetryczna: po lewej stronie zawsze piszemy zmienne, a obliczane wyrażenie zawsze po prawej.

Niesymetryczność instrukcji podstawiania jest związana z jej dynamicznym charakterem. Na przykład instrukcja

$$x := x + 1$$

oznacza, że w wyniku jej wykonania wartość x wzrośnie o 1. W tradycyjnym zapisie musielibyśmy kolejnym wartościom x dawać indeksy i napisać

$$x_{i+1} = x_i + 1,$$

gdyż zapis

$$x = x + 1$$

dawałby sprzeczność.

Widzimy, że znaczenie symbolu := jest inne niż znaku równości. Znak := jest symbolem podstawiania. Dynamiczny charakter instrukcji podstawiania polega na tym, że zmienne występujące w wyrażeniu po prawej stronie instrukcji, jak również zmienne występujące w indeksach po lewej stronie, przyjmują wartości obowiązujące w momencie rozpoczęcia wykonania instrukcji, natomiast zmienne wymienione kolejno po lewej stronie instrukcji przyjmują wartości dopiero w wyniku wykonania instrukcji.

W każdej nowej instrukcji w programie algolowskim obowiązują wartości zmiennych ostatnio obliczone. Nową obliczoną wartość zmiennej wymazuje jak gdyby wartość poprzednią.

W instrukcjach podstawienia mamy do czynienia z podstawianiem liczbowej wartości wyrażenia arytmetycznego na zmienne liczbowe albo z podstawianiem logicznej wartości wyrażenia booleowskiego na zmienne booleowskie. Wyrażenia desygnujące ani wyrażenia łańcuchowe w instrukcjach podstawienia nie występują.

Wszystkie zmienne wymienione po lewej stronie instrukcji podstawienia muszą być tego samego typu: albo wszystkie typu integer, albo wszystkie typu real, albo wreszcie wszystkie typu Boolean. Jeśli wszystkie te zmienne są typu integer, to wyrażenie po prawej stronie instrukcji może mieć wartość W typu real, jest ona jednak automatycznie przeliczana na wartość typu integer równą entier $(W+0.5)$, tzn. równą najbliższej liczbie całkowitej. Jeśli wszystkie te zmienne są typu real, to wyrażenie po prawej stronie może mieć wartość typu integer, jest ona jednak automatycznie interpretowana jako real i tak podstawiana na zmienne.

Jeśli zmienne wymienione po lewej stronie instrukcji podstawienia są liczbowe, to oczywiście wyrażenie po prawej stronie nie może być booleowskie, a tylko arytmetyczne, i na odwrót, jeśli wszystkie te zmienne są booleowskie, to wyrażenie po prawej stronie nie może być arytmetyczne, a tylko booleowskie.

A oto przykłady instrukcji podstawienia:

- 1) $x1 := A[1,1] := k := 2 \times k - 3$
- 2) $Beta := \exp(\alpha \times t) \times \cos(\gamma \times t)$ ✓
- 3) $RR := x > 0 \wedge x < 1$
- 4) $UKD[n-1, -1-n+5, 3 \times n] := UKD[1, n, 2 \times 1 + 3 \times n - 7] :=$
 $\text{sqrt}(1 + \text{abs}(\ln(\text{abs}(x)))) + 1$
- 5) $War := \text{if } c > 0 \text{ then } A < B \Rightarrow B < C \text{ else } B < 0$
- 6) $D[\text{if } x = 0 \text{ then } m \text{ else } m+1] := 0$
- 7) $F11 := F12 := x^3 + 2xy^3 + (\text{if } x > 1 \text{ then } x^2 - 3xy^2 \text{ else } 0)$
- 8) $A[m+n, m-n] := m := n := A[m+n, m-n] + 1$

W przykładzie 1 mamy do czynienia z podstawieniem aktualnej wartości wyrażenia $2 \times k - 3$ na trzy zmienne liczbowe: $x1$, $A[1,1]$, k . Gdyby w chwili rozpoczęcia wykonywania danej instrukcji zmienna k miała wartość 5, wyrażenie $2 \times k - 3$ miałoby wartość 7 i taką wartość otrzymałyby zmienne $x1$, $A[1,1]$, k . Zauważmy, że na skutek wykonania danej instrukcji zmienna k zmieniła swoją wartość z 5 na 7. Również, gdyby zmienne $x1$ i $A[1,1]$ miały w chwili rozpoczęcia wykonywania danej instrukcji jakieś wartości, zostałyby one zamienione na 7.

W przykładzie 2 mamy do czynienia z instrukcją podstawienia na zmienną liczbową $Beta$ aktualnej wartości wyrażenia, które w tradycyjnym języku matematycznym miałoby postać

$$e^{xt} \cos_{\beta} t. - \frac{1}{\beta}$$

Wartość tego wyrażenia jest obliczana z aktualnych wartości zmiennych α , t i γ , z tym że obliczenie zostaje wykonane z ograniczoną dokładnością (dla maszyny GIER - jak to już było powiedziane w paragrafie 1.7 - ze względną dokładnością około $3_{10}-9$).

W p r z y k ł a d z i e 3 podano instrukcję podstawienia na zmienną RR wartości wyrażenia booleowskiego $x > 0 \wedge x < 1$. Wynika stąd, że RR musi być zmienną booleowską. Gdyby w chwili rozpoczęcia wykonania danej instrukcji zmienna x miała wartość na przykład 0.2, instrukcja ta spowodowałaby nadanie zmiennej RR wartości true. Gdyby x miało wartość na przykład 5, zmienna RR otrzymałaby wartość false.

W p r z y k ł a d z i e 4 mamy do czynienia z podstawieniem aktualnej wartości wyrażenia arytmetycznego

$$\text{sqrt}(1 + \text{abs}(\ln(\text{abs}(x)))) + 1$$

na dwa wyrazy pewnej trójwymiarowej tablicy UKD. Identyfikacja tych dwu wyrazów następuje przez obliczenie ich indeksów z wartości, jakie zmienne i , n mają w chwili rozpoczęcia wykonywania danej instrukcji. Załóżmy na przykład, że wykonywanie danej instrukcji rozpoczyna się w momencie, gdy i ma wartość 1, n wartość 2, a x wartość -1. Najpierw zostają obliczone indeksy zmiennych po lewej stronie instrukcji, na skutek czego staje się ona równoznaczna z instrukcją

$$\text{UKD}[1,2,6] := \text{UKD}[1,2,1] := \text{sqrt}(1 + \text{abs}(\ln(\text{abs}(x)))) + 1$$

W drugim kroku zostaje obliczona wartość wyrażenia po prawej stronie instrukcji, przez co staje się ona równoznaczna z instrukcją

$$\text{UKD}[1,2,6] := \text{UKD}[1,2,1] := 2$$

W trzecim kroku na zmienne $\text{UKD}[1,2,6]$ i $\text{UKD}[1,2,1]$ zostaje podstawiona wartość 2.

W p r z y k ł a d z i e 5 mamy do czynienia z podstawieniem na zmienną booleowską War aktualnej wartości alternatywnego wyrażenia booleowskiego. Załóżmy na przykład, że w chwili rozpoczynania wykonania danej instrukcji zmienna A miała wartość -2, zmienna B wartość -1, zmienna C wartość -4 i zmienna c wartość 8. Wobec $c > 0$ dane alternatywne wyrażenie booleowskie staje się równoznaczne z

$$A < B \Rightarrow B < C$$

i przyjmuje wartość true \Rightarrow false czyli false. Wobec tego w wyniku wykonania danej instrukcji na zmienną War zostanie podstawiona wartość false.

W p r z y k ł a d z i e 6 mamy do czynienia z przypadkiem, gdy indeks zmiennej, na którą mamy podstawić wartość 0, jest określony alternatywnie. Załóżmy na przykład, że w momencie rozpoczynania wykonywania danej instrukcji zmienna x ma wartość 1, a zmienna m wartość 2. Wobec $x \neq 0$ alternatywne wyrażenie arytmetyczne określające indeks staje się równoznaczne z $m+1$ i ma wartość 3. Na skutek tego zostanie wykonana instrukcja

$$D[3] := 0$$

W p r z y k ł a d z i e 7 mamy znowu do czynienia z podstawieniem na dwie zmienne liczbowe F11 i F12 aktualnej wartości pewnego wyrażenia arytmetycznego. Załóżmy dla przy-

kładu, że w chwili rozpoczynania wykonywania danej instrukcji zmienna x ma wartość 2, a zmienna y wartość 3. Wobec $x > 1$ alternatywne wyrażenie arytmetyczne w nawiasach okrągłych staje się równoznaczne z $x \wedge 2 - 3 \vee y \wedge 2$ i na skutek tego zostanie wykonana instrukcja:

F11:= F12:= 39

Wyrażenie z przykładu 8 rozpatrzmy również dla konkretnych wartości zmiennych. Załóżmy na przykład, że w chwili rozpoczynania wykonania danej instrukcji zmienna m ma wartość 3, zmienna n wartość 1, a zmienna $A[4,2]$ wartość 6. Najpierw zostają obliczone indeksy zmiennych po lewej stronie danej instrukcji, przez co staje się ona równoznaczna z instrukcją

$A[4,2] := m := n := A[m+n, m-n] + 1$

W kroku drugim zostaje obliczona wartość wyrażenia arytmetycznego po prawej stronie danej instrukcji, przez co staje się ona równoznaczna z instrukcją

$A[4,2] := m := n := 7$

W kroku trzecim na zmienne $A[4,2]$, m , n zostaje podstawiona wartość 7, wymazując poprzednie wartości tych zmiennych.

3.3. Instrukcje złożone

Pisząc instrukcje kolejno jedna po drugiej, oddzielamy je znakiem; tzn. średnikiem. Jeśli pewną liczbę kolejnych instrukcji ujmemy wyrazami begin (tzn. zaczynaj; czytaj: begin) i end (tzn. koniec; czytaj: end) jak gdyby nawiasami, otrzymujemy całość zwaną instrukcją złożoną. Wyrazy begin i end są w ALGOLu uważane za pewien rodzaj nawiasów: begin jest tu nawiasem otwierającym, a end zamykającym.

Ostatnia instrukcja przed end nie musi kończyć się średnikiem, ale zakończenie jej średnikiem nie stanowi błędu.

3.4. Bloki, Deklaracje

Jeśli pewną liczbę instrukcji ujmemy w nawiasy begin i end, a między wyrazem begin i pierwszą z kolei instrukcją z danego ciągu instrukcji umieścimy tzw. d e k l a r a c j e (declarations; czytaj: deklarejszys), otrzymujemy w ten sposób całość zwaną blokiem.

Różnica między instrukcją złożoną a blokiem polega właśnie na tym, że na początku bloku są deklaracje, których nie ma na początku instrukcji złożonej.

Podkreślić tu jednak należy, że zarówno blok, jak instrukcja złożona są szczególnymi przypadkami instrukcji (patrz paragraf 3.1) i mogą być zatem zawarte wewnątrz innych bloków czy instrukcji złożonych.

Deklaracje służą w ALGOLu równolegle do dwóch celów. Po pierwsze dają one maszynie informacje co do charakteru wielkości użytych w programie. Po drugie powodują odpowiednią rezerwację miejsca dla deklarowanych wielkości w pamięci maszyny.

Deklaracje są ważne tylko w granicach bloku, w nagłówku którego zostały umieszczone. Po wyjściu z danego bloku wielkości w nim zadeklarowane stają się nieokreślone.

Deklaracje piszemy zawsze na początku bloku po jego nawiasie otwierającym begin, a przed jego pierwszą instrukcją. Każdą deklarację kończymy średnikiem. Kolejność deklaracji jest obojętna. Pomiędzy otwierającym begin a pierwszą deklaracją można umieścić tzw. komentarz (będzie o tym mowa w paragrafie 3.23). Jeśli go nie ma, nie stawiamy żadnego znaku.

Rozróżniamy cztery rodzaje deklaracji:

- d e k l a r a c j a t y p u (type declaration; czytaj: tajp deklarejszn),
- d e k l a r a c j a t a b l i c y (array declaration; czytaj: arej deklarejszn),
- d e k l a r a c j a p r z e ł ą c z n i k a (switch declaration; czytaj: slich deklarejszn),
- d e k l a r a c j a p r o c e d u r y (procedure declaration; czytaj: prosjidżer deklarejszn).

Wszystkie zmienne, tablice, przełączniki i procedury wymagają zawsze ich uprzedniego zadeklarowania, z wyjątkiem tzw. procedur standardowych, których lista była podana w paragrafie 1.8., i tzw. parametrów formalnych w procedurach, o czym będzie mowa w paragrafie 4.2. Przełączniki poznamy w paragrafie 3.6.

Deklaracja typu składa się z nazwy typu (real, integer albo Boolean, co - ale tylko w GIER-ALGOLu - można również pisać boolean) z dodaniem listy zmiennych danego typu, których nazwy oddzielamy przecinkami. Należy zapamiętać, że zmienne z indeksami nie wchodzą tu w rachubę, gdyż deklarujemy je za pomocą deklaracji tablic. Jak już była o tym mowa w paragrafie 1.12, zmienne z indeksami są zawsze komponentami jakiejś tablicy.

Przykłady deklaracji typu:

```
integer i,j,numer,alfa; real x,Y,Summa; boolean p;
```

Deklaracja tablicy rozpoczyna się wyrazem array z ewentualnym poprzedzeniem nazwą typu, a więc integer array, real array lub Boolean array, po czym wypisuje się nazwy tablic z ewentualnym dopisywaniem nawiasu kwadratowego, w którym podaje się ograniczenia indeksów dla danej tablicy. Dolne ograniczenie pisze się jako pierwsze, potem następuje dwukropek, a następnie górne ograniczenie. W nawiasie kwadratowym mamy tyle par ograniczeń, ile indeksów mają zmienne będące wyrazami tablicy, czyli ile wymiarów ma dana tablica. Pary ograniczeń są od siebie oddzielane przecinkami. Każde ograniczenie, zarówno dolne jak górne, może być dowolnym wyrażeniem arytmetycznym.

Jeśli takie wyrażenie ma wartość W real, zostaje ona automatycznie przeliczona na całkowitą równą entier(W+0.5). Jeśli w deklaracji tablicy nazwa tablicy nie jest uzupełniona nawiasami kwadratowymi obejmującymi pary ograniczeń dla indeksów, oznacza to, że ma ona ograniczenia indeksów, wspólne z następną tablicą. Na przykład deklaracja

```
array A,B,C[1:3];
```

oznacza, że wszystkie trzy tablice A, B, C są jednowymiarowe i ich wyrazy są zmiennymi o indeksach od 1 do 3. Ostatnia z tablic wymienionych w jednej deklaracji zawsze ma podane ograniczenia indeksów w nawiasach kwadratowych.

Nazwy tablic wymieniane w jednej deklaracji oddzielamy od siebie przecinkami.

Jeśli deklaracja tablic rozpoczyna się samym wyrazem array, jest to równoznaczne z deklaracją real array. Typ wymieniony przy array odnosi się do wszystkich wyrazów wszystkich tablic wymienionych w danej deklaracji. Jeśli w programie przewidujemy tablice więcej niż jednego typu, musimy napisać osobne deklaracje dla każdego typu tablic.

Jeśli górne ograniczenie jakiegokolwiek indeksu ma wartość mniejszą niż dolne, odpowiednia tablica jest nieokreślona.

Wyrażenie arytmetyczne będące w deklaracji tablicy ograniczeniem indeksu może zawierać zmienne i procedury tylko takie, które były deklarowane w bloku zewnętrznym w stosunku do tego, w którym dana deklaracja tablicy występuje. Wynika to stąd, że wartość takiego wyrażenia musi być obliczona zaraz po wejściu do bloku z daną deklaracją tablicy, a do tego potrzebne jest uprzednie ustalenie wartości zmiennych i parametrów występujących w tym wyrażeniu, co mogło nastąpić tylko w bloku zewnętrznym w stosunku do danego. Z powyższego wynika, że deklaracje tablic w bloku najbardziej zewnętrznym dla danego programu mogą mieć tylko stałe, tzn. liczbowe ograniczenia indeksów.

Jeśli deklaracja określa tablicę za dużą dla maszyny albo jeśli w którejś z deklaracji tablic górne ograniczenie ma wartość całkowitą mniejszą od wartości całkowitej ograniczenia dolnego, maszyna wydrukuje automatycznie sygnał alarmowy "array". Będzie o tym mowa w paragrafie 6.12.

A oto przykłady deklaracji tablic:

- 1) array A,B,C[-1:k+1,2:6],M,N[k-5:2xk];
- 2) Boolean array BOOLE[1:5,3x1-j:12,2x1+3xj-4:8x1+j];
- 3) integer array WS,WS1,WS2,F[1:4,1:m];

W przykładzie 1 mamy zadeklarowanych 5 tablic o wyrazach typu real (pamiętamy, że array oznacza to samo, co real array). Z tych tablic pierwsze trzy, a mianowicie tablice A, B i C są tablicami dwuwymiarowymi, ostatnie dwie natomiast, tzn. tablice M i N, są jednowymiarowe. Ograniczenia indeksów dla wyrazów tych tablic zależą od aktualnej wartości pewnej zmiennej k. Załóżmy na przykład, że w momencie wejścia do bloku, w nagłówku którego jest umieszczona dana deklaracja, zmienna k ma wartość 6. W takim przypadku dana deklaracja staje się równoznaczna z deklaracją

array A,B,C[-1:7,2:6],M,N[1:12];

Znaczy to, że wyrazy tablic A, B, C mają pierwszy indeks zmieniający się od -1 co 1 aż do 7 włącznie, a drugi zmieniający się od 2 co 1 aż do 6 włącznie (wynika stąd, że tablice A, B, C mają dla danej wartości k ogółem po 45 wyrazów). Wyrazy tablic M i N mają tylko jeden indeks, zmieniający się od 1 co 1 do 12 (tablice te mają zatem dla podanego k po 12 wyrazów).

W przykładzie 2 mamy zadeklarowaną tylko jedną tablicę o wyrazach typu Boolean, czyli o wyrazach logicznych przyjmujących wartości true albo false. Tablica jest trójwymiarowa. Ograniczenia indeksów dla jej wyrazów zależą

od aktualnych wartości zmiennych i, j . Założmy na przykład, że w momencie wejścia do bloku, w nagłówku którego znajduje się dana deklaracja, zmienna i ma wartość 2, a zmienna j wartość 1. Wobec tego dana deklaracja staje się równoznaczna z deklaracją

Boolean array BOOLE[1:5,5:12,3:17];

Znaczy to, że wyrazy tablicy BOOLE mają pierwszy indeks zmieniający się od 1 co 1 aż do 5 włącznie, drugi od 5 co 1 do 12 włącznie, a trzeci od 3 co 1 do 17 włącznie. Wynika stąd, że tablica BOOLE ma dla podanych wartości i, j ogółem 600 wyrazów.

W p r z y k ł a d z i e 3 mamy zadeklarowane 4 tablice dwuwymiarowe o wyrazach liczbowych typu integer czyli całkowitych. Wszystkie cztery tablice mają jednakowe ograniczenia dla indeksów. Ich wyrazy mają pierwszy indeks zmieniający się od 1 co 1 aż do 4, a drugi od 1 co 1 aż do aktualnej całkowitej wartości zmiennej m . Założmy dla przykładu, że w momencie wejścia do bloku, w którego nagłówku znajduje się dana deklaracja, zmienna m ma wartość 5.71. Zgodnie z podaną wyżej regułą jako górne ograniczenie drugiego indeksu będzie użyta liczba

$$\text{entier}(5.71+0.5) = \text{entier}(6.21) = 6$$

i dana deklaracja będzie równoznaczna z deklaracją

integer array WS,WS1,WS2,F[1:4,1:6];

Omówimy teraz pięć prostych przykładów bloków, zakładając, że wyniki obliczeń będą podstawiane na tzw. z m i e n i e n i e l o k a l n e, to znaczy już uprzednio zadeklarowane w jakimś bloku zewnętrznym, obejmującym blok omawiany. W przeciwnym razie wyniki obliczeń byłyby kasowane w momencie wyjścia z omawianego bloku czyli z chwilą pojawienia się zamykającego wyrazu end i cały blok kończyłby się całkowicie jałowo (metody wyprowadzania wyników poza maszynę będą omówione dopiero w rozdziale 5). Pierwsze trzy przykłady będą miały charakter czysto formalny, dwa następne będą miały geometryczną interpretację. We wszystkich jedyną zmienną nie-lokalną będzie X .

1) begin integer a, b ; real c, d ;

$a := -3$;

$b := (2 \times a)^2 - 5 \times a - 4$;

$c := a := \text{sqrt}(b+2) + a$;

$d := b_{\text{abs}}(a) / (c+1)$;

$c := d + a$;

$X := (c-d)^{\wedge}((b-7)/5/a)$

end;

- 2) begin integer m,n; integer array M[-1:3]; Boolean B1,B2;
 m:= 0; n:= 1;
 M[m]:= M[n+1]:= $n \wedge m + 3 \times n - 3$;
 M[m-n]:= M[M[0]]:= (M[2]-M[m+n-1]) \times ($3 \times m + n$)-1;
 m:= M[m+1];
 M[$4 \times n + m$] := n-m;
 n:= M[M[n]]:= $2 \times n + 1$;
 M[n]:= M[n]+n;
 B1:= M[-1] \wedge M[0];
 B2:= M[1] \wedge M[2];
 m:= if B1 then M[0] else M[-1];
 n:= if B2 then M[2] else M[1];
 B1:= n \wedge M[3];
 n:= if B1 then M[3] else n;
 X:= if m \wedge n then n else m
end;
- 3) begin real TER,gamma; Boolean KO;
 gamma:= sin(1.23586);
 TER:= exp(sqrt($n-3$));
 KO:= abs(gamma-TER) \wedge n^{-4} ;
 X:= (if KO then TER else gamma/3) \wedge 2.687
end;
- 4) begin real pi,wysokosc,promien duzy,promien.maly;
 pi:= 3.1415926; wysokosc:= 12.65;
 promien duzy:= 23.90; promien maly:= 17.35;

```

X:= pi*xwysokosc/3*(promien duzy $\sqrt{2}$ +promien duzy*promien
    maly+promien maly $\sqrt{2}$ )
end;

```

```

5) begin real a,b,c,d,e;
   a:= 5.7;  b:= 2.9;  c:= 4.2;  d:= e:= 3.3;
   a:= a $\sqrt{2}$ +b $\sqrt{2}$ ;
   a:= a-c $\sqrt{2}$ ;
   a:= a+d $\sqrt{2}$ ;
   X:= sqrt(a+e $\sqrt{2}$ )
end;

```

W p r z y k ł a d z i e 1 otrzymujemy z kolejnych instrukcji:

a=-3, b=47, c=4, a=4, d=2.2, c=6.2, X=16.

W celu śledzenia kolejno otrzymywanych wartości zmiennych wygodnie jest posługiwać się następującym harmonogramem:

a	b	c	d	X
-3				
	47			
4		4		
			2.2	
		6.2		
				16

Wychodzimy z danego bloku z wartością X=16. Wartości zmiennych a,b,c,d są wtedy skasowane.

Dla p r z y k ł a d u 2 odpowiedni harmonogram wygląda następująco:

m	n	M[-1]	M[0]	M[1]	M[2]	M[3]	B1	B2	X
0									
	1								
			1		1				
		-1		-1					
-1									

3 3

5

falsetrue

3

1

true

5

5

Wychodzimy z danego bloku z wartością $X=5$. W tym momencie wartości pozostałych zmiennych, występujących w tym bloku, jako lokalne ulegają skasowaniu.

Jak można spostrzec, blok z przykładu 2 służy do obliczenia wyrazów tablicy M , a następnie do wybrania z nich wyrazu największego.

W przykładzie 3 na zmienną γ zostaje podstawiona wartość sinusa kąta 1.23586 radianów, a na zmienną TER wartość wyrażenia, które w tradycyjnym języku matematycznym można by zapisać jako

$$e^{\sqrt{0.001}}$$

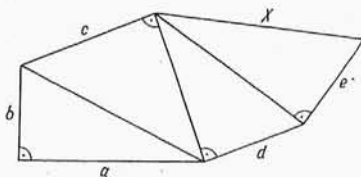
(Ściślej biorąc na obie zmienne zostają podstawione przybliżone wartości odpowiednich wyrażeń, ponieważ dokładność maszyn jest ograniczona). Nie wchodząc w to, jakie wartości dziesiętne uzyskują w ten sposób zmienne γ i TER , zauważmy jedynie, że $\sin(1.23586) < 0.99$ a $\exp(\sqrt{0.001}) > 1$. Wobec tego $\gamma - TER < -0.01$ i $abs(\gamma - TER) > 0.01$. Zatem KO otrzymuje wartość false i na zmienną X zostaje podstawiona wartość wyrażenia

$$(\gamma/3)^{1/2.687}$$

czyli w języku tradycyjnym

$$\left(\frac{1}{3} \sin 1.23586\right)^{2.687}$$

Jaką konkretnie wartość dziesiętną uzyskuje w ten sposób zmienna X , interesować się tu nie będziemy.



Rys. 1

W przykładzie 4 jako wartość zmiennej X otrzymujemy objętość stożka ściętego o wysokości 12.65, promieniu podstawy dolnej 23.90 i promieniu podstawy górnej 17.35.

W przykładzie 5 jako wartość zmiennej X otrzymujemy długość boku kratownicy, w której oznaczono kąty proste i dane boki (rys. 1).

W podanym programie wykorzystano zmienną a jako zmienną roboczą dla kwadratów długości pośrednich boków kratownicy.

3.5. Zadanie

Obliczyć końcowe wartości zmiennych nielokalnych X i Y dla następujących bloków:

1) begin real Z1,Z2,a,b; integer i,j;

i:= j:= 2;

i:= $1 \wedge j \wedge 2 - 3 \times j$;

j:= 1/j;

a:= X:= Y:= (j+3)/1;

Z1:= 1/5+a;

Z2:= Z1/1.4+0.1;

b:= (Z1-Z2)×X;

Y:= (X \wedge 2-b)/Y;

X:= sqrt(X-b+Y/10);

Y:= (b-2×Y)^($-3 \times X$);

Y:= X/Y

end;

2) begin integer k,m; integer array PAUL[1:4],KR[0:2,0:1];

m:= 3;

k:= $m \wedge 3 - 2$;

PAUL[m-2]:= KR[k/5-4,0]:= sqrt(k)-8×m;

k:= k+PAUL[m/3];

PAUL[k-m+1]:= KR[1,0]+3×k;

m:= X:= m+PAUL[4];

KR[-PAUL[2×m]+1,1]:= KR[X,X-m]:= PAUL[X]:= $k \wedge m$ +PAUL[1];

KR[KR[2,0]-KR[2,1],0]:= Y:= PAUL[k/m]:= KR[1,0]+PAUL[2];

X:= -X/Y;

KR[X,X]:= KR[0,X]:= $Y \wedge X$;

```

KR[-KR[1,1],-KR[0,1]/2]:= KR[2,1];k+m;
KR[X,1]:= KR[m,X]-X;
KR[m+Y,m+KR[0,0]]:= -KR[-Y,0]+5;
KR[1,0]:= Y^3;
Y:= (K[0,1]-K[0,0])/(PAUL[1]+PAUL[2]);
m:= (m^2+2xm-k/2)/Y;
X:= m^2x^2(-Y)+X^3x^2
end;

```

```

3) begin Boolean DAN; integer n;
n:= 5;
DAN:= .n<3;
n:= if DAN then n^2-n+1 else (n^3-4xn^2)/5-2;
DAN:= DAN => n=0;
X:= if n=1 then 0 else if DAN then n-2 else 2xn-7
end;

```

3.6. Wyrażenia desygnujące

Wyrażenia desygnujące służą do wskazywania miejsca, do którego należy skoczyć w programie. Wyrażenia desygnujące mogą być proste albo alternatywne. Po wyjaśnieniach podanych w paragrafie 2.4. wystarczy opisać budowę prostego wyrażenia desygnującego.

Proste wyrażenie desygnujące jest:

etykietą
albo
przełącznikiem
albo

dowolnym wyrażeniem desygnującym ujętym w nawiasy okrągłe.

E t y k i e t a (label; czytaj: lejbl) jest to nazwa nadawana instrukcjom (a więc m.in. instrukcjom złożonym i blokom). Nazwy takie nadajemy tylko takim instrukcjom, do których przewidujemy skoki w programie. Etykieta jest nazwą tworzoną na takich samych zasadach jak nazwy zmiennych, z tym że etykieta zawsze kończy się dwukropkiem, po którym następuje nazywana instrukcja (tzn. etykieta poprzedza mianowaną przez siebie instrukcję). W ALGOLu etykiety mogą być ponadto licz-

bami naturalnymi. W GIER-ALGOLu takie etykiety nie są dozwolone. Etykiet się nie deklaruje. Należy zapamiętać, że etykieta jest ważna tylko w najmniejszym bloku ją obejmującym. Wynika stąd, że do bloku można skoczyć tylko przez jego początek, gdyż wewnętrzne etykiety na zewnątrz bloku nie są ważne, a cały blok jest instrukcją, która może mieć swą etykietę ważną w bloku zewnętrznym. Etykieta ta dla samego nazwanego bloku jest etykietą zewnętrzną. Etykiety umieszcza się bezpośrednio przed nazywaną instrukcją.

Jeśli zachodzi potrzeba, instrukcja może być nazwana więcej niż jedną etykietą, np.:

START: WARIANT1: A: alfa:= 3xx-5x(2-a/c);

W przykładzie tym mamy do czynienia z instrukcją podstawienia, w wyniku której nastąpi podstawienie wartości na zmienną liczbową alfa. Instrukcja jest zaopatrzona w trzy etykiety: START, WARIANT 1 i A.

P r z e ł ą c z n i k (switch; czytaj: słicz) jest wyrażeniem umożliwiającym wybór miejsca, do którego należy skoczyć w programie, spośród wyrażen desygnujących podanych w deklaracji przełącznika. Przełącznik zawsze wymaga deklaracji.

Deklaracja przełącznika ma następującą budowę:

switch

nazwa przełącznika

:=

lista wyrażen desygnujących oddzielanych przecinkami

;

Oto przykłady deklaracji przełącznika:

switch Q:= pi,START,A[m];

switch KLUCZ:= S1,S2,koniec;

Sam przełącznik ma budowę zmiennej z jednym indeksem. Indeks ten może przyjmować tylko wartości naturalne niewiększe od liczby wyrażen desygnujących, podanych w deklaracji przełącznika. Oznacza on numer kolejny wyrażenia desygnującego, jakie należy wybrać z deklaracji przełącznika jako aktualny adres skoku do wykonania. Na przykład dla drugiej z podanych wyżej deklaracji przełącznik

KLUCZ[2]

oznacza, że należy skoczyć do instrukcji wskazanej przez drugie z kolei wyrażenie desygnujące w deklaracji, tzn. do instrukcji z etykietą S2. Natomiast przełącznik

KLUCZ[3]

oznaczałby, że należy skoczyć do instrukcji opatrzonej etykietą Koniec. Indeks przełącznika może być dowolnym wyrażeniem arytmetycznym, którego wartość liczymy tak jak wartości indeksów dla zmiennych.

Ponieważ w deklaracji przełącznika podana jest lista wyrażen desygnujących, które niekoniecznie muszą być etykietami, może się zdarzyć, że przełącznik powoduje tylko wybranie nowego przełącznika. Ale nawet w taki rekurencyjny sposób musimy na koniec dojść do etykiety, bo tylko etykiety

stanowią nazwy instrukcji, do których następuje skok w programie. Przełączniki same przez się nie są nazwami takich instrukcji, a służą jedynie do wyboru etykiety.

Przykład:

```
switch Q:= p1, START, KLUCZ[k], if fi ≤ 0 then S1 else KLUCZ[k+1];  
switch KLUCZ:= K1, K2, K3, K4, Koniec;
```

W deklaracjach tych dla przełącznika Q podano cztery wyrażenia desygujące, z których dwa pierwsze są etykietami, trzecie przełącznikiem KLUCZ z indeksem k, a czwarte alternatywnym wyrażeniem desygującym. Dla przełącznika KLUCZ podano pięć wyrażeń desygujących będących etykietami.

Założmy, że w danym momencie jest $k=2$ i $fi=1$. Przełącznik $Q[1]$ wskazałby etykietę p1 jako nazwę instrukcji docelowej dla skoku. Przełącznik $Q[2]$ wskazałby etykietę START. Natomiast na przykład przełącznik $Q[2+fi]$ równoznaczny z $Q[3]$ wskazałby przełącznik KLUCZ[k], czyli w danym momencie KLUCZ[2]. Z kolei przełącznik KLUCZ[2] wskazałby instrukcję o etykiecie K2.

Przełącznik $Q[4]$ wskazałby czwarte z wyrażeń desygujących w deklaracji przełącznika Q. Ponieważ $fi=1$ i tym samym $fi > 0$, wyrażenie to byłoby równoznaczne z KLUCZ[k+1] czyli w danym momencie z KLUCZ[3]. Z kolei ten przełącznik wskazałby instrukcję o etykiecie K3.

3.7. Instrukcje skoku

Instrukcje skoku mają następującą budowę:

go to

wyrażenie desygujące

Na skutek takiej instrukcji następną wykonywaną przez maszynę będzie nie instrukcja napisana po danej instrukcji skoku, lecz instrukcja wskazana przez wyrażenie desygujące.

W GIER-ALGOLU zamiast go to można również pisać goto albo go to (z jednym tylko odstępem). Go to znaczy "idź do" (czytaj: go! tu).

Należy pamiętać, o czym już była mowa w paragrafie poprzednim, że instrukcja go to nie może prowadzić z zewnątrz bloku do jego wnętrza. Natomiast może prowadzić do wnętrza instrukcji złożonej. Jeśli instrukcja skoku zawiera wyrażenie desygujące prowadzące do nieokreślonego przełącznika, to cała taka instrukcja działa jak instrukcja pusta, tzn. maszyna po prostu przechodzi do następnej z kolei instrukcji, nie wykonując żadnego skoku.

Oto przykłady instrukcji skoku:

- 1) go to A
- 2) go to KLUCZ[3×k-1]
- 3) go to if Z > 0 then A else B
- 4) go to ALT[if a+b < 1 then 2 else 3]
- 5) go to if alfa=beta then QQ else ETA[if alfa > beta then K1 else 4]

W p r z y k ł a d z i e 1 mamy do czynienia z instrukcją skoku do instrukcji opatrzonej etykietą A.

W p r z y k ł a d z i e 2 mamy do czynienia z instrukcją skoku do instrukcji wskazanej przez przełącznik o nazwie KLUCZ. Instrukcję docelową moglibyśmy ustalić tylko wtedy, gdybyśmy znali deklarację przełącznika KLUCZ i wartość zmiennej k. Wybralibyśmy wtedy z listy wyrażeń desygnujących w tej deklaracji wyrażenie $(3 \times k - 1)$ -sze z kolei. Gdyby było ono etykietą, instrukcja docelowa byłaby przez nią już określona. Gdyby było ono przełącznikiem, postępowanie prowadziłibyśmy analogicznie dalej, aż uzyskalibyśmy ostatecznie etykietę instrukcji docelowej. Załóżmy na przykład, że w momencie napotkania (przez maszynę wykonującą program) danej instrukcji skoku jest $k=2$, a przełącznik KLUCZ był uprzednio zadeklarowany jak w poprzednim paragrafie, tzn.

switch KLUCZ:= K1,K2,K3,K4,Koniec;

Dana instrukcja skoku jest w takim przypadku równoznaczna z instrukcją go to KLUCZ[5], a zatem z instrukcją skoku do instrukcji opatrzonej etykietą Koniec.

W p r z y k ł a d z i e 3 mamy do czynienia z instrukcją skoku alternatywnego. Jeśli w momencie napotkania przez maszynę danej instrukcji skoku pewna zmienna Z ma wartość dodatnią, skok nastąpi do instrukcji opatrzonej etykietą A, w przeciwnym razie do instrukcji opatrzonej etykietą B.

W p r z y k ł a d z i e 4 mamy do czynienia z instrukcją skoku zawierającą przełącznik z indeksem utworzonym przez alternatywne wyrażenie arytmetyczne. Jeżeli w danym momencie zmienne a i b mają takie wartości, że $a+b < 1$, cała instrukcja jest równoznaczna z instrukcją

go to ALT[2]

w przeciwnym zaś razie z instrukcją

go to ALT[3]

W obu przypadkach mamy do czynienia z tym samym przełącznikiem ALT, ale o indeksie zależnym od tego, czy $a+b < 1$, czy nie.

W p r z y k ł a d z i e 5 podano instrukcję skoku z podwójną alternatywą. Jeśli pewne zmienne liczbowe alfa i beta mają w danym momencie takie wartości, że $\alpha = \beta$, skok nastąpi do instrukcji opatrzonej etykietą QQ. Jeśli wymienione zmienne mają takie wartości, że $\alpha > \beta$, skok nastąpi do instrukcji wskazanej przez przełącznik ETA[K1]. Jeśli wreszcie będzie $\alpha < \beta$, skok nastąpi do instrukcji wskazanej przez przełącznik ETA[4]. Gdyby na przykład przełącznik ETA był uprzednio zadeklarowany następująco:

switch ETA:= START,Iteracja,Kontrola,Zmiana,KONIEC;

a w momencie napotkania przez maszynę danej instrukcji skoku zmienna K1 miała wartość 5, to w przypadku $\alpha > \beta$ skok nastąpiłby do instrukcji opatrzonej etykietą KONIEC, a w przypadku $\alpha < \beta$ do instrukcji opatrzonej etykietą Zmiana.

3.8. Instrukcje puste

Instrukcja pusta jest instrukcją niepisaną. Wprowadzenie pojęcia instrukcji pustej służy jedynie dla wytłumaczenia pew-