

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 wyrażen 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 prowadzilibyśmy 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 $\text{alfa} = \text{beta}$, skok nastąpi do instrukcji opatrzonej etykietą QQ. Jeśli wymienione zmienne mają takie wartości, że $\text{alfa} > \text{beta}$, skok nastąpi do instrukcji wskazanej przez przełącznik ETA[K1]. Jeśli wreszcie będzie $\text{alfa} < \text{beta}$, 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 $\text{alfa} > \text{beta}$ skok nastąpiłby do instrukcji opatrzonej etykietą KONIEC, a w przypadku $\text{alfa} < \text{beta}$ do instrukcji opatrzonej etykietą Zmiana.

3.8. Instrukcje puste

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

nych wyjątków od reguł. Na przykład według reguł algolowskich pomiędzy ostatnią instrukcją w instrukcji złożonej czy bloku a wyrazem end nie dajemy średnika. Chcemy jednak od tej reguły zrobić wyjątek i średnik postawić. Można to jednak zrobić również bez łamania reguły, jeśli sobie wyobraźmy, że po tym nadprogramowym średniku następuje instrukcja pusta (już bez średnika), a dopiero po niej wyraz end.

Innym przykładem zastosowania instrukcji pustej jest następujący: Przez pomyłkę zapisaliśmy po jakiejś instrukcji dwa średniki. Chcemy wiedzieć, czy to nie będzie miało wpływu na przebieg programu. Otóż na skutek dopuszczenia w ALGOLu instrukcji pustych, możemy sobie wyobrazić, że między naszymi średnikami jest instrukcja pusta, a zatem program jest poprawny. Natomiast maszyna na skutek instrukcji pustej nie wykonuje oczywiście żadnej czynności.

Instrukcje puste mogą ponadto służyć do wyjątkowego umieszczenia etykiety. Była mowa o tym w paragrafie 3.6, że etykietę stawiamy tylko przed instrukcją. Widzimy jednak, że etykietę można również postawić np. przed wyrazem end, jak np. w następującym przykładzie:

.....

a := a + 3 * b;

ZOFIA: end;

ponieważ możemy sobie wtedy wytłumaczyć to wstawieniem instrukcji pustej pomiędzy etykietą i wyrazem end. Etykieta daje wtedy nazwę instrukcji pustej.

3.9. Instrukcje warunkowe

Instrukcja warunkowa może mieć trojakią postać:

- 1) warunek "jeśli"
 instrukcja bezwarunkowa

albo

- 2) warunek "jeśli"
 instrukcja bezwarunkowa bez średnika na końcu

else
dowolna instrukcja

albo

- 3) warunek "jeśli"
 instrukcja "dla"

Wykonanie instrukcji warunkowej zależy od warunku "jeśli". Jeżeli warunek ten jest spełniony, czyli zawarte w nim wyrażenie boolowskie ma wartość true, to cała instrukcja warunkowa staje się równoznaczna z instrukcją następującą po warunku "jeśli". Jeśli warunek ten nie jest spełniony, czyli zawarte w nim wyrażenie boolowskie ma wartość false, to wykonanie instrukcji warunkowej zależy od jej postaci. W przypadku postaci 1 lub 3 następuje wtedy po prostu przejście do następnej z kolei instrukcji, czyli cała dana instrukcja warunkowa staje się równoznaczna z instrukcją pustą. Natomiast

w przypadku postaci 2 następuje wykonanie instrukcji napisanej po else (która z kolei może być również instrukcją warunkową), czyli cała dana instrukcja warunkowa staje się wtedy równoznaczna z instrukcją wymienioną po wyrazie else.

Należy zapamiętać, że instrukcja napisana przed else nie może kończyć się średnikiem. Dodanie instrukcji pustej nic by tu nie pomogło, gdyż mielibyśmy wtedy po warunku "jeśli" dwie instrukcje zamiast wymaganej jednej. Wobec tego musieliśmy obie ująć w nawiasy begin i end, aby w ten sposób utworzyć jedną instrukcję złożoną, i po end w dalszym ciągu nie byłoby średnika.

Podane dla postaci 1 i 2 instrukcji warunkowej ograniczenie, że instrukcja po warunku "jeśli" musi być bezwarunkowa, podyktowane jest koniecznością uniknięcia dwuznaczności. Ograniczenie to nie jest istotne, ponieważ dowolna instrukcja przez wzięcie jej w nawiasy begin i end staje się instrukcją złożoną, która należy do bezwarunkowych, jak to było podane w paragrafie 3.1.

Podobnie ograniczenie, że po warunku "jeśli" i po else można podać tylko jedną instrukcję, nie jest istotne, ponieważ przez wzięcie w nawiasy begin i end z całego ciągu instrukcji czynimy jedną instrukcję złożoną.

Instrukcja "dla" wymieniona w postaci 3 instrukcji warunkowej będzie omówiona w paragrafie 3.18.

A oto przykłady instrukcji warunkowych:

- 1) if $a > b$ then $X := X \uparrow 2 + X + 1$
- 2) if $A < B \wedge B < C$ then $A := \text{sqrt}(z) - x$ else go to ST[k]
- 3) if $a < 0 \leq b < 0$ then Q1: begin $X := X + 2$; if $X < 10$ then go to Q1 end
else if $a \leq b < -5$ then begin Q2: $X := X - 2$; if $X > -10$ then go to Q2
else $X := \text{abs}(X)$ end else go to KONIEC

W p r z y k ł a d z i e 1 wykonanie instrukcji $X := X \uparrow 2 + X + 1$ zależy od tego, czy w momencie napotkania przez maszynę danej instrukcji warunkowej zmienna liczbowa a ma wartość większą niż b , czy nie. Jeśli tak, to cała instrukcja warunkowa staje się równoznaczna z instrukcją $X := X \uparrow 2 + X + 1$, jeśli nie, to cała instrukcja warunkowa staje się równoznaczna z instrukcją pustą, tzn. przechodzimy od razu do instrukcji następnej.

W p r z y k ł a d z i e 2 podano instrukcję warunkową, która w przypadku, gdy wyrażenie boolowskie $A < B \wedge B < C$ ma wartość true, jest równoznaczna z instrukcją

$A := \text{sqrt}(z) - x$

a w przypadku, gdy wymienione wyrażenie boolowskie ma wartość false, z instrukcją

go to ST[k]

Zauważmy, że pierwsza z tych instrukcji została zaopatrzona w etykietę A. W ALGOLu jest bowiem dopuszczalny skok do wnętrza instrukcji warunkowej.

Założmy dla przykładu, że w momencie napotkania przez wykonującą program maszynę danej instrukcji warunkowej mamy następujące wartości zmiennych: $A=B=C=1$, $k=3$. Wobec tego, że

obie relacje $A < B$ i $B < C$ mają wartości false, całe wyrażenie boolowskie ma wartość false i dana instrukcja warunkowa staje się równoznaczna z instrukcją skoku

go to ST[3]

Przełącznik ST musiał być oczywiście uprzednio zadeklarowany.

W przykładzie 3 instrukcja warunkowa ma następującą budowę:

Warunkiem "jeśli" jest:

if $a < 0 = b < 0$ then

Instrukcją bezwarunkową następującą po warunku "jeśli" jest instrukcja złożona:

Q1: begin $X := X + 2$; if $X < 10$ then go to Q1 end

Instrukcją po else jest nowa instrukcja warunkowa

if $a < b < -5$ then begin Q2: $X := X - 2$; if $X > -10$ then go to Q2

else $X := \text{abs}(X)$ end else go to KONIEC

Rozpatrzmy kolejno wymienione trzy części danej instrukcji warunkowej.

Warunek "jeśli" jest równoważnością dwu relacji i ma zatem wartość true, gdy obie relacje $a < 0$ i $b < 0$ mają wartość true albo obie false.

Instrukcja bezwarunkowa po warunku "jeśli" jest opatrzona etykietą Q1 i jest instrukcją złożoną z dwu instrukcji: instrukcji podstawienia

$X := X + 2$

i instrukcji warunkowej

if $X < 10$ then go to Q1

która w przypadku $X < 10$ jest równoznaczna z instrukcją skoku do instrukcji opatrzonej etykietą Q1, czyli skoku do początku tejże instrukcji złożonej, a w przypadku $X \geq 10$ jest równoznaczna z instrukcją pustą. Widzimy, że taka instrukcja złożona doprowadza do wartości X większej, a co najmniej równej 10, ponieważ w przypadku mniejszej wartości X następuje zawsze ponowne powiększenie tej wartości o 2.

Instrukcja warunkowa po else ma następującą budowę:
Warunek "jeśli"

if $a < b < -5$ then

Instrukcja bezwarunkowa po warunku "jeśli":

begin Q2: $X := X - 2$; if $X > -10$ then go to Q2 else $X := \text{abs}(X)$ end

Instrukcja po else:

go to KONIEC

Instrukcja po warunku "jeśli" jest tu instrukcją złożoną z dwu instrukcji: instrukcji podstawienia

Q2: $X := X - 2$

opatrzonej etykietą Q2 i jeszcze jednej instrukcji warunkowej

if $X > -10$ then go to Q2 else $X := \text{abs}(X)$

która w przypadku, gdy $X > -10$, jest równoznaczna z instrukcją skoku do instrukcji opatrzonej etykietą Q2, czyli do poprzedniej instrukcji, a w przypadku, gdy $X \leq -10$, z instrukcją podstawienia $X := \text{abs}(X)$. Widzimy, że taka instrukcja złożona doprowadza zawsze do wartości X większej, a co najmniej równej 10, ponieważ najpierw przez powtarzanie instrukcji podstawienia $X := X - 2$ sprowadza wartość zmiennej X poniżej -10 albo co najwyżej do -10 , a potem zmienia jej znak na dodatni poprzez instrukcję podstawienia $X := \text{abs}(X)$.

Ujmując wszystkie możliwe przypadki dla wyjściowej instrukcji warunkowej w przykładzie 3, widzimy, że może być ona równoznaczna z następującymi instrukcjami:

```

a<0, b<0 albo a<0, b>0:      Q1: begin X:= X+2;

                               if X<10 then go to Q1 end

a<0, b>0 albo a>0, b<0 1  a×b<-5:  begin Q2: X:= X-2; if X>-10

                               then go to Q2 else

                               X:= abs(X) end

a<0, b>0 albo a>0, b<0 1  a×b>-5:  go to KONIEC

```

Załóżmy na przykład, że w momencie napotkania przez maszynę danej instrukcji warunkowej zmienne a i b miały odpowiednio wartości -3 i $+4$, a zmienna X wartość 0 . Wobec tego, że relacja $a < 0$ ma wtedy wartość true, a relacja $b < 0$ wartość false, równoważność $a < 0 \equiv b < 0$ ma wartość false i tym samym cała wyjściowa instrukcja warunkowa staje się równoznaczna z instrukcją

```

if a×b<-5 then begin Q2: X:= X-2; if X>-10 then go to Q2

else X:= abs(X) end else go to KONIEC

```

Ponieważ w danym przypadku jest $a \times b < -5$, ta z kolei instrukcja warunkowa staje się równoznaczna z instrukcją złożoną:

```

begin Q2: X:= X-2; if X>-10 then go to Q2 else X:= abs(X) end

```

Po wykonaniu instrukcji $X := X - 2$ mamy $X = -2$ i wobec tego następna instrukcja złożona

```

if X>-10 then go to Q2 else X:= abs(X)

```

staje się równoznaczna z instrukcją go to Q2. Powracamy zatem do instrukcji $X := X - 2$ i otrzymujemy $X = -4$ a z następnej instrukcji znowu go to Q2. Po jeszcze trzykrotnym powtórzeniu instrukcji $X := X - 2$ otrzymujemy wreszcie $X = -10$ i następna instrukcja warunkowa staje się tym razem równoznaczna z instrukcją

```

X:= abs(X)

```

skąd otrzymujemy $X = 10$. Na tym kończy się wykonanie całej rozpatrywanej instrukcji i maszyna przechodzi do instrukcji następnej.

3.10. Zadanie

Następujące instrukcje bezwarunkowe napisać w postaci instrukcji warunkowych:

- 1) $XX := (\text{if } c > 0 \text{ then } 2xz \text{ else } -4) + 5xz + 6$
- 2) $\text{go to if } a^2 + b^2 > 15 \text{ then ITER}[2xk+j] \text{ else K13K}$
- 3) $\text{go to RAK}[\text{if } a < b \vee a > 0 \text{ then } 3 \text{ else } m+n]$
- 4) $\text{BoB} := \text{if } wrm < -1 \text{ then } A = B \wedge C \text{ else } B = A \vee C$

3.11. Przykład: Sumowanie szeregu

Jako przykład ułożymy program na zsumowanie szeregu.

$$\frac{1}{1.4} - \frac{1}{3.4^3} + \frac{1}{5.4^5} - \frac{1}{7.4^7} + \dots$$

zakładając, że suma S tego szeregu jest dla danego bloku nielokalną. Sumowanie przerwiemy, gdy następny kolejny wyraz szeregu będzie co do wartości bezwzględnej mniejszy niż 10^{-8} , tzn. - jak wiadomo z teorii szeregów - różnica między szukaną sumą szeregu a sumą uwzględnionych wyrazów tego szeregu będzie co do wartości bezwzględnej mniejsza niż 10^{-8} .

Oto przykład takiego programu:

```
begin integer n; real a;
n:= 1; S:= 0;
Q: a:= 1/n/4n;
if a<10-8 then go to KONIEC;
if n-1=4*((n-1):4) then S:= S+a else S:= S-a;
n:= n+2;
go to Q;
KONIEC: end;
```

3.12. Zadanie

- a) Ułożyć program na obliczenie pierwiastka równania

$$x^3 - 2x + 5 = 0$$

metodą kolejnych przybliżeń Newtona, zakładając, że pierwszym przybliżeniem szukanego pierwiastka $x_0 = -2$. Jak wiadomo, kolejne przybliżenie szukanego pierwiastka równania

$$f(x) = 0$$

liczy się metodą Newtona ze wzoru

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

co w danym przypadku daje wzór

$$x_{n+1} = x_n - \frac{x_n^3 - 2x_n + 5}{3x_n^2 - 2}$$

Liczenie należy przerwać, gdy różnica między dwoma kolejnymi przybliżeniami szukanego pierwiastka będzie mniejsza niż 10^{-8} . Zmienną oznaczającą szukany pierwiastek równania przyjąć jako nielokalną dla danego programu.

b) Rozwiązać poprzednie zadanie posługując się metodą Reguła Falsi i przyjmując jako przybliżenia wyjściowe $x_1 = -3$ i $x_2 = -2$. Jak wiadomo, Reguła Falsi dla równania

$$f(x) = 0$$

i takich x_1 i x_2 , że $f(x_1) \cdot f(x_2) < 0$

polega na obliczeniu nowego przybliżenia x_3 szukanego pierwiastka ze wzoru

$$x_3 = \frac{x_2 \cdot f(x_1) - x_1 \cdot f(x_2)}{f(x_1) - f(x_2)}$$

Gdy $f(x_1) \cdot f(x_3) < 0$, dalsze obliczenia prowadzi się dla punktów x_1 i x_3 , gdy natomiast $f(x_2) \cdot f(x_3) < 0$, to dla punktów x_2 i x_3 .

3.13. Zadanie

Jakie błędy popełniono w następującym programie, w którym tylko X jest zmienną nielokalną:

```
begin real a,b; integer k;
a:=1; k:=0.1;
b:=(3a-4)/k;
a:X:=if a>0 then 2 else k+c;
a:=k:=X/2;
```

```

if a>X then go to a;
X:= if b<15 then 5 else go to a
end

```

3.14. Zadanie

Podać wykorzystywane kolejno etykiety i końcowe wartości zmiennych nielokalnych a i b dla programu:

```

begin integer i,j;
switch Kad:= A,AA,AAA;
switch IT:= Kad[i-2×j+2],B,BB;
i:= j:= 1;
B: a:= if i>j then i else j;
AA: b:= if i>j then i+2 else j+2;
i:= i+1;
if i<3 then go to IT[j] else go to Kad[if i+j<3 then i+j else j];
A: j:= (j+1)+1;
go to if j>10 then BB else if i>3 then B else Kad[i];
AAA: j:= j+3;
a:= j-b; if j<5 then go to IT[j-i+1];
BB: end;

```

3.15. Lokalność zmiennych, tablic, etykiet i przełączników

Zmienne, tablice i przełączniki są zawsze deklarowane, jak to już było powiedziane w paragrafie 3.4. Wobec tego jasne jest, że można z nich korzystać tylko w granicach bloku, w nagłówku którego dana deklaracja została podana. Etykiet natomiast nie deklaruje się, ale mimo to – jak o tym już była mowa w paragrafie 3.6 – są one ważne tylko w granicach najmniejszego bloku je obejmującego.

Powyższe stwierdzenia nie są wystarczające, ponieważ w programach może dochodzić do konfliktowych sytuacji. Takie sytuacje omówimy w niniejszym paragrafie.

Pierwszą sytuacją konfliktową, najczęściej spotykaną, jest zawieranie się jednego bloku w drugim. Możliwość konfliktów zostaje usunięta przez wprowadzenie następujących jednoznacznych reguł:

- jeżeli jakaś wielkość jest deklarowana w bloku zewnętrznym a nie jest deklarowana w bloku wewnętrznym, to można z niej korzystać w granicach całego bloku zewnętrznego, tzn. łącznie z wnętrzem bloku wewnętrznego, w sensie deklaracji z bloku zewnętrznego; taką wielkość dla bloku wewnętrznego nazwaliśmy *n i e l o k a l n ą*, ponieważ została ona wprowadzona na podstawie deklaracji zewnętrznej, a więc *n i e l o k a l n e j*; ta sama wielkość będzie dla bloku zewnętrznego *l o k a l n ą*, ponieważ w nim właśnie była deklarowana;

- jeżeli jakaś wielkość jest deklarowana w bloku zewnętrznym, a następnie wielkość o tej samej nazwie jest deklarowana również w bloku wewnętrznym, to deklaracja zewnętrzna jest ważna tylko w granicach bloku zewnętrznego z wyłączeniem bloku wewnętrznego; wewnątrz bloku wewnętrznego obowiązuje natomiast deklaracja wewnętrzna; w danej sytuacji wielkość zadeklarowaną w bloku wewnętrznym traktuje się jako różną od zadeklarowanej w bloku zewnętrznym, pomimo że mają identyczne nazwy; np. przy wejściu do bloku wewnętrznego wartość zmiennej zadeklarowanej w jego nagłówku jest na razie nieokreślona, pomimo że zmienna o tej samej nazwie na zewnątrz bloku wewnętrznego mogła już mieć jakąś aktualną wartość; podobnie przy wyjściu z bloku wewnętrznego obowiązuje dla takiej zmiennej nie ostatnia wartość z bloku wewnętrznego, lecz ostatnia otrzymana przed ostatnim wejściem do bloku wewnętrznego; tożsamość nazw powoduje jedynie niemożliwość korzystania w bloku wewnętrznym z wielkości zadeklarowanej na zewnątrz, natomiast nie utożsamia samych wielkości, ponieważ odróżniają się one umiejscowieniem deklaracji.

P r z y k ł a d:

```

begin integer a,b,c,z;

z:= 3;

a:= z/z;

b:= ai(2xz);

c:= b-z+1;

begin integer y,z;

y:= b+c;

Q: z:= a-2xy+1;   a:= a-y;

if z>0 then go to Q;

b:= b-z+5

end;

x:= a+b+z

end;

```

W przykładzie tym deklaracja zmiennych a, b, c rozciąga się na cały podany program, deklaracja zmiennej y tylko na blok wewnętrzny. Zmienna x jest dla podanego programu nie-lokalna. Mamy natomiast dwie różne zmienne o wspólnej nazwie z : jedna z nich jest zadeklarowana w bloku zewnętrznym, ale ta deklaracja ze względu na wspólność nazw nie rozciąga się na blok wewnętrzny, natomiast druga jest zadeklarowana w bloku wewnętrznym i tylko w nim może być użyta. Prześledzimy wykonanie programu. Najpierw $z=3$. Potem z następnych trzech instrukcji otrzymujemy kolejno $a=27$, $b=4$ i $c=2$. Wchodzimy teraz do bloku wewnętrznego. Wartość z jest nieokreślona, ponieważ jest to nowa zmienna. Obliczamy, że $y=6$, a następnie $z=16$ i $a=21$. Ponieważ $16 > 0$, następna instrukcja warunkowa staje się równoznaczna z go to Q i powracamy do instrukcji z etykietą Q . Otrzymujemy teraz $z=10$ i $a=15$. Powracamy jeszcze raz i otrzymujemy $z=4$ i $a=9$. Powracamy jeszcze raz i otrzymujemy $z=-2$ i $a=3$. Ponieważ teraz $z < 0$, następna instrukcja warunkowa staje się równoznaczna z instrukcją pustą. Obliczamy dalej, że $b=11$ i wychodzimy z bloku wewnętrznego. Zmienne a i b zachowują swe ostatnie wartości z bloku wewnętrznego, ponieważ były dla niego nie-lokalne, wartość zmiennej y jako lokalnej dla bloku wewnętrznego staje się nieokreślona, natomiast zmienna z ma wartość ostatnią przed wejściem do bloku wewnętrznego, tzn. $z=3$ (ostatnia wartość z z bloku wewnętrznego, tj. $z=-2$ przepada). Wobec tego z ostatniej instrukcji obliczamy, że $x=17$ i z tą wartością wychodzimy z danego programu. Z tą chwilą wartości wszystkich innych zmiennych poza x , jako wartości zmiennych lokalnych dla danego programu, stają się nieokreślone.

Analogiczne reguły obowiązują dla etykiet w przypadku zawierania się jednego bloku w drugim:

- jeżeli etykieta mianuje instrukcję bloku zewnętrznego na zewnątrz bloku wewnętrznego, a w bloku wewnętrznym nie jest użyta etykieta o tej samej nazwie, to z bloku wewnętrznego możemy wyskoczyć pod daną etykietę, pomimo że jest ona na zewnątrz tego bloku wewnętrznego;

- jeżeli etykieta mianuje instrukcję bloku zewnętrznego na zewnątrz bloku wewnętrznego, a w bloku wewnętrznym jest użyta etykieta o tej samej nazwie, to w bloku wewnętrznym każda instrukcja skoku do etykiety o danej nazwie spowoduje skok wewnątrz tego bloku, natomiast każda instrukcja skoku do etykiety o danej nazwie pojawiająca się na zewnątrz bloku wewnętrznego spowoduje skok do instrukcji leżącej również na zewnątrz tego bloku wewnętrznego.

Widzimy zatem, że etykietę traktujemy tak, jakby była zadeklarowana w nagłówku najmniejszego bloku ją obejmującego.

P r z y k ł a d:

```
begin integer a,b,c,z;
```

```
z:= 3;
```

```
a:= z/z;
```

```
b:= a;(2xz);
```

```
E: c:= b-z+1;
```

```
begin integer y,z;
```

```

y:= b+c;

Q: z:= a-2xy+1;   a:= a-y;

  if z>0 then go to Q;

b:= b-z+5;

  if b<25 then go to E

end;

Q: x:= a+b+z;

  if x<40 then begin z:= z+5;   go to Q end

end;

```

Przykład ten powstał z poprzedniego przez dodanie dwu etykiet i dwu instrukcji warunkowych. Prześledzimy jego wykonanie.

Zaczynamy - jak poprzednio - od $z=3$ i obliczamy $a=27$, $b=4$ i $c=2$. Wchodzimy do bloku wewnętrznego i obliczamy $y=6$, $z=16$ i $a=21$. Ponieważ $16 > 0$ następna instrukcja warunkowa staje się równoznaczna z go to Q. Są w programie dwie etykiety Q. Ponieważ instrukcja skoku była w bloku wewnętrznym, skok następuje do etykiety Q również w bloku wewnętrznym. Obliczamy $z=10$ i $a=15$, powracamy i obliczamy $z=4$ i $a=9$, powracamy jeszcze raz i obliczamy $z=-2$ i $a=3$. Teraz instrukcja warunkowa staje się równoznaczna z pustą. Dalej obliczamy $b=11$ i następna z kolei instrukcja warunkowa staje się równoznaczna z instrukcją go to E. Ponieważ istnieje tylko jedna instrukcja o takiej etykiecie, nie ma kolizji etykiet takiej, jaka była w przypadku dwu etykiet Q i skok następuje do instrukcji leżącej na zewnątrz bloku wewnętrznego. Obliczamy dalej, że $c=9$ (obowiązuje teraz wartość $z=3$) i wchodzimy ponownie do bloku wewnętrznego. Otrzymujemy $y=20$ a następnie $z=-36$ i $a=-17$. Następna instrukcja warunkowa jest obecnie równoznaczna z pustą, a z jeszcze następnej dostajemy $b=52$. Wobec tego i następna instrukcja warunkowa staje się równoznaczna z pustą, po czym wychodzimy z bloku wewnętrznego. Obliczamy teraz, że $x=38$, wobec czego kolejna instrukcja warunkowa jest równoznaczna z instrukcją złożoną z dwu instrukcji:

```
z:= z+5;
```

```
go to Q
```

Z pierwszej otrzymujemy $z=8$. Druga jest instrukcją skoku. Mamy znowu kolizję dwu etykiet Q. Ponieważ instrukcja skoku pojawiła się na zewnątrz bloku wewnętrznego, skok nastąpi do instrukcji leżącej również na zewnątrz bloku wewnętrznego tzn. do instrukcji

```
Q: x:= a+b+z;
```

Otrzymujemy teraz $x=43$, wobec czego kolejna instrukcja warunkowa staje się równoznaczna z pustą. Wychodzimy z danego programu z wartością $x=43$.

Z lokalności etykiet wynika również, że etykiety będące wartościami wyrażen desygnujących w deklaracji przełącznika nie mogą leżeć w bloku wewnętrznym w stosunku do bloku, w którego nagłówku jest deklaracja tego przełącznika, zgodnie z zasadą, że etykieta nie jest dostępna z zewnątrz najmniejszego bloku ją obejmującego.

Natomiast wywołanie przełącznika może nastąpić w bloku wewnętrznym w stosunku do bloku, w nagłówku którego znajduje się deklaracja danego przełącznika, zgodnie z zasadą, że z bloku można wyskoczyć do etykiety leżącej na zewnątrz. Ewentualne konflikty stąd wynikające rozwiązujemy zgodnie z podanymi wyżej regułami.

3.16. Zadanie

Obliczyć końcową wartość zmiennej nielokalnej Z w następującym programie:

```
begin real A,B,C;   integer n;

  switch W:= W1,S[n+3],W3;   switch S:= S1,S2,S3,STOP;

  A:= 5;   B:= -3;   n:= 0;

W1: C:= A+B;

S1: n:= n+C;

S2: if A>3 then begin A:= A+B;   go to W[n-1] end;

W3: B:= B-C;

S3: C:= B-2*C;

begin integer n;   real C;

  C:= A+B+1;   n:= entier(C)+1;   Z:= A+B+C+n;

  if C>0 then go to W[n] end;

  A:= B:= C:= 0;

STOP: end;
```

Podać wykorzystane kolejno etykiety.

3.17. Deklaracje z "own"

W chwili wyjścia z jakiegokolwiek bloku, wartości zmiennych i tablic zadeklarowanych w nagłówku tego bloku stają się nieokreślone. Jeżeli jednak deklarację poprzedzimy wyrazem own (czytaj: ołn; znaczy "własny"), to przy ponownym wejściu do tego bloku wielkości podane w takiej deklaracji będą miały te wartości, jakie miały w chwili ostatniego wyjścia z danego bloku. Natomiast wiel-

kości zadeklarowane bez own przy ponownym wejściu do danego bloku pozostają nieokreślone. Wyraz own powoduje zachowanie ostatnich wartości z danego bloku dla późniejszego ich ewentualnego wykorzystania przy ponownym wejściu do tego bloku.

W GIER-ALGOLu wyrazem own można poprzedzać tylko deklaracje zmiennych, a więc deklaracje typu. W przeciwieństwie do ALGOLu GIER-ALGOL nie dopuszcza own w deklaracjach tablic. Wynika to z faktu, że deklaracje own array wymagają rezerwowania w pamięci maszyny bardzo wielu miejsc. Deklaracje own array mogą być zatem dopuszczane w maszynach o znacznie większej pamięci niż w maszynie GIER.

A oto przykłady deklaracji z own:

- 1) integer i,j,n; own real x; real y,z,a,b,c;
- 2) real k,skok,ro; own boolean alfa,c;

3.18. Instrukcje "dla"

Instrukcja "dla" służy dla powtórzenia jednej i tej samej instrukcji (która w szczególności może być instrukcją złożoną, blokiem czy nową instrukcją "dla") dla różnych wartości parametru zwanego z m i e n n ą k o n t r o l o w a n ą. Listę kolejnych wartości zmiennej kontrolowanej nazywamy listą elementów "dla" ("dla" po angielsku znaczy "for"; czytaj: for).

Instrukcja "dla" ma następującą budowę:

```

for
nazwa zmiennej kontrolowanej
:=
lista elementów "dla"
do
instrukcja
```

Wyraz do (czytaj: do) znaczy: wykonaj.

Elementami listy "dla" (oddzielanymi między sobą przecinkami) mogą być następujące trzy rodzaje wyrażeń:

- 1) wyrażenie arytmetyczne
- 2) wyrażenie arytmetyczne
step
 wyrażenie arytmetyczne
until
 wyrażenie arytmetyczne
- 3) wyrażenie arytmetyczne
while
 wyrażenie boolowskie

Wyraz step (czytaj: step) znaczy: krok. Wyraz until (czytaj: until) znaczy: aż do. Wyraz while (czytaj: łajł) znaczy: podczas gdy.

Wyrażenie pierwszego typu, tzn. wyrażenie arytmetyczne daje jedną wartość dla zmiennej kontrolowanej.

Wyrażenie drugiego typu może dawać wiele wartości dla zmiennej kontrolowanej. Sposób działania tego wyrażenia w instrukcji "dla" można by opisać za pomocą następującego programu, w którym zakładamy, że elementem listy "dla" jest wyrażenie

A step B until C;

(co po polsku znaczy "od A co B aż do C") zmienną kontrolowaną zmienna V, a instrukcją do wykonania INSTRUKCJA:

V:= A;

Powtorz: if (V-C)×sign(B)≤0 then begin INSTRUKCJA;

V:= V+B;

go to Powtorz

end;

Jeżeli zatem wyrażenie A ma wartość spełniającą warunek

$(A-C) \times \text{sign}(B) \leq 0$

to instrukcja INSTRUKCJA zostaje najpierw wykonana dla V=A (w przeciwnym razie przechodzimy do następnego elementu z listy "dla", a gdy następnego już nie ma, do następnej instrukcji po danej instrukcji "dla").

Następnie wartość zmiennej V stale wzrasta o B i dla tak otrzymywanych kolejnych wartości V wykonywana jest ponownie INSTRUKCJA. Dopiero gdy wartość V przekroczy wartość C, warunek

$(V-C) \times \text{sign}(B) \leq 0$

przestaje być spełniany i przechodzimy do następnego elementu listy "dla", a gdy takiego już nie ma, do instrukcji następującej po danej instrukcji "dla".

Jak widzimy z powyższego, zmienna V nie musi przybierać wartości C, gdyż zależy nam jedynie na sygnalizacji, kiedy ta wartość zostanie przekroczona, a nie osiągnięta. Fakt ten wykorzystujemy zawsze w przypadkach, gdy - na skutek zaokrągleń - wartości zmiennej V ulegają zniekształceniom. Podwyższamy wtedy sztucznie wartość C, tak aby to powiększenie objęło błędy zaokrągleń, a nie przekroczyło wartości kroku B (dla B < 0 mamy tu oczywiście zamiast powiększenia - zmniejszenie). Gdybyśmy takiego zabiegu nie dokonali, błędy zaokrągleń mogłyby zmienić liczbę powtórzeń danej INSTRUKCJI.

Wyrażenie trzeciego typu może również dawać wiele wartości dla zmiennej kontrolowanej. Sposób działania tego wyrażenia w instrukcji "dla" można by opisać za pomocą następującego programu, w którym zakładamy, że elementem listy "dla" jest wyrażenie

ARYTM while BOOL,

zmienną kontrolowaną V a instrukcją do wykonania INSTRUKCJA:

Powtorz: V:= ARYTM;

if BOOL then begin INSTRUKCJA; go to Powtorz end;

Wykonanie INSTRUKCJI dla $V=ARYTM$ jest zatem uzależnione od tego, czy wyrażenie $BOOL$ ma wartość true czy false. Wartości wyrażenia arytmetycznego $ARYTM$, jak też wyrażenia boolowskiego $BOOL$, ulegają na ogół przy powtarzaniu zmianom na skutek wykonywania INSTRUKCJI i związanych z tym zmian wartości zmiennych.

Jeżeli lista elementów "dla" zostanie wyczerpana, wartość zmiennej kontrolowanej staje się nieokreślona. Jeśli natomiast nastąpiło wyjście z instrukcji "dla" na skutek pojawienia się instrukcji skoku w czasie wykonywania powtórzeń, zmiennej kontrolowanej zachowuje swą ostatnią wartość.

Natomiast nie wolno z zewnątrz instrukcji "dla" skoczyć poprzez instrukcję skoku do wnętrza instrukcji "dla".

Przykłady instrukcji "dla":

- 1) for $k:=1,3,4,7,15$ do $X[k]:=k+1$
- 2) for $i:=1$ step 1 until n do $C[i]:=A[i]+B[i]$
- 3) for $u:=17, u-2$ while $v>0$ do $v:=v-u$
- 4) for $i:=1$ step 1 until n do
 for $j:=1$ step 1 until m do
 $C[i,j]:=A[i,j]+B[i,j]$

W p r z y k ł a d z i e 1 instrukcja $X[k]:=k+1$ jest wykonana najpierw dla $k=1$. Otrzymujemy $X[1]=2$. Następnie powtarzamy instrukcję dla $k=3$. Otrzymujemy $X[3]=4$. Z kolei obliczamy, że $X[4]=5$, $X[7]=8$ i $X[15]=16$. Z pomocą jednej instrukcji "dla" obliczyliśmy tutaj kilka wyrazów jakiejś tablicy X .

W p r z y k ł a d z i e 2 jedną instrukcją "dla" wykonujemy dodanie dwu tablic jednowymiarowych o indeksach od 1 do n .

Dla wykonania instrukcji "dla" w p r z y k ł a d z i e 3 konieczna jest znajomość ostatniej wartości zmiennej v . Zauważmy, że ostatnio było $v=40$. Lista elementów "dla" dla zmiennej kontrolowanej u zawiera 2 elementy: wyrażenie arytmetyczne redukujące się do liczby 17 i wyrażenie $u-2$ while $v>0$. Instrukcja $v:=v-u$ jest zatem najpierw wykonana dla $u=17$. Otrzymujemy $v=23$. Ponieważ $23>0$, powtarzamy wykonanie tej instrukcji dla $u:=u-2$, tzn. dla $u=15$. Otrzymujemy $v=23-15$, czyli $v=8$. Ponieważ $8>0$, powtarzamy postępowanie dla $u:=u-2$, czyli $u=13$. Otrzymujemy $v=8-13$, czyli $v=-5$. Ponieważ -5 nie jest większe od zera, wychodzimy z instrukcji "dla". W tym momencie wartość zmiennej kontrolowanej u jest nieokreślona, ponieważ wyczerpaliśmy listę elementów "dla", natomiast zmienna v ma wartość -5 .

W p r z y k ł a d z i e 4 instrukcją do wykonania w danej instrukcji "dla" jest druga instrukcja "dla". Za pomocą takiej podwójnej instrukcji "dla" zostaje wykonane dodawanie dwu tablic dwuwymiarowych. Instrukcja "dla" wewnętrzna powoduje tu dodanie wyrazów leżących w ustalonym i-tym wierszu, począwszy od wyrazów leżących w kolumnach 1, a skończywszy na wyrazach z kolumn m -tych. Instrukcja "dla" zewnętrzna powoduje przesuwanie dodawania na następny wiersz, począwszy od 1 aż do n -go.

Instrukcja "dla" może wchodzić w skład instrukcji warunkowej. Należy wtedy pamiętać, że dopuszczalna jest tylko konstrukcja

```
warunek "jeśli"
instrukcja "dla"
```

natomiast nie wolno używać konstrukcji

```
warunek "jeśli"
instrukcja "dla"
else
instrukcja
```

która mogłaby dawać dwuznaczność (patrz paragraf 3.9). Przy takiej regule będziemy mogli jednoznacznie rozumieć na przykład instrukcję

```
if B then for q:= ... do if B2 then I1 else I2
```

Mamy tu na początku warunek "jeśli"

```
if B then
```

Po nim następuje instrukcja "dla", w której po do musi być jakaś instrukcja. Gdyby nią była instrukcja

```
if B2 then I1
```

całość przybrałaby zabronioną formę

```
warunek "jeśli"
instrukcja "dla"
else
I2
```

Wobec tego po do następuje tu instrukcja

```
if B2 then I1 else I2
```

co można by zaakcentować np. nawiasami begin i end:

```
if B then
```

```
for := ... do
```

```
begin if B2 then I1 else I2 end
```

Wspomniane ograniczenie nie jest kłopotliwe, ponieważ mamy zawsze możliwość przez wzięcie w nawiasy begin i end z instrukcji "dla" uczynić instrukcję złożoną, a zatem instrukcję bezwarunkową.

W instrukcjach "dla" zmienna kontrolowana może być indeksowana. Wartość indeksów ustala się przed wykonaniem obliczeń dla listy elementów "dla" i nie mają na nią wpływu późniejsze zmiany wartości zmiennych występujących w indeksach.

3.19. Zadanie

Ułożyć program na mnożenie macierzy A i B, jeżeli w wyniku ma być otrzymana macierz C, a w bloku zewnętrznym macierze były zadeklarowane następująco:

```
array A[1:1,1:j], B[1:j,1:k], C[1:1,1:k];
```


3.20. Zadanie

Ułożyć fragment programu, który kolejnym wyrazom jednomiarowej tablicy A nadałby wartości:

1.668 , 4.608 , 6.951 , 8.002 , 10.168 ,
12.376 , 14.589 , 18.207 , 20.113 , 22.724 .

3.21. Zadanie

Obliczyć kolejne wartości zmiennej nielokalnej t w programie:

```
begin real a,b,c; integer i,j,k;
a:= 0; b:= -2; c:= 3; t:= 5;
for i:= a+b,b+c,c:b-2xb do t:= t+i;
for j:= t,1,2 step 1 until 7 do t:= t-3xj;
for k:= t+100 step 3 until 22 do t:= t+2xk;
for i:= 2,i+2 while t>3,t-1,1 step -1 until -5 do t:= t-i;
for j:= 1 step 1 until 4 do
for k:= j step 1 until 4 do t:= t+j-k;
for i:= -1+t,t-i step if t>i then t else i until
if t>0 then t+10 else t+i do t:= t-i
end;
```

3.22. Zakończenie instrukcji

Każda instrukcja musi kończyć się albo średnikiem, albo wyrazem end, albo wyrazem else. Wynika stąd, że między ostatnią instrukcją bloku lub instrukcji złożonej a wyrazem end średnik nie jest potrzebny. Możliwość jego umieszczenia wynika - jak to zauważyliśmy w paragrafie 3.8. - z możliwości dodania instrukcji pustej. Tego nie możemy jednak zrobić przed else (patrz paragraf 3.9). Wyraz else nie może być za tym poprzedzony bezpośrednio średnikiem.

Z powyższego wynika, że każdy ciąg end end end ... musi kończyć się średnikiem albo else, ponieważ każde end jest zakończeniem jakiejś instrukcji.

Z powyższego wynika również, że cały program musi kończyć się średnikiem, ponieważ stanowi zawsze blok.

3.23. Komentarze

ALGOL dopuszcza wpłatanie w program tzw. komentarzy, które służą do objaśnienia programu, natomiast przez translator są ignorowane. Dopuszczalne są następujące trzy rodzaje komentarzy:

1)

```
; comment
dowolny ciąg znaków nie zawierający średnika
;
```

2)

```
begin comment
dowolny ciąg znaków nie zawierający średnika
;
```

3)

```
end
dowolny ciąg znaków nie zawierający ani średnika,
ani end, ani else
```

Komentarz typu 1 jest równoważny pojedynczemu średnikowi.

Komentarz typu 2 jest równoważny pojedynczemu wyrazowi begin (bez średnika).

Komentarz typu 3 jest równoważny pojedynczemu wyrazowi end (bez średnika).

Dla przykładu napiszemy program z zadania 3.11 wraz z kilkoma komentarzami:

```
begin comment Sumowanie szeregu; integer n; real a;
```

```
n:= 1; S:= 0; wyzerowanie sumy S przed rozpoczęciem jej liczenia;
```

```
Q: a:= 1/n/4n;
```

```
if a<10-8 then go to KONIEC;
```

```
if n-1=4×((n-1):4) then S:= S+a else S:= S-a; dodanie do sumy
```

```
następnego wyrazu;
```

```
n:= n+2;
```

```
go to Q;
```

```
KONIEC: end Sumowania szeregu;
```

3.24. Zadanie

Napisać program z poprzedniego paragrafu, zastępując fragment rozpoczynający się od etykiety Q a kończący się go to Q, instrukcją "dla".