

dy przewidzieć skok do etykiety, która w projektowanej procedurze nie musi być precyzowana.

Obliczane kolejno niewiadome umieszczać w jednej z kolumn tablicy współczynników.

4.9. Zadanie

Zakładając, że procedura z poprzedniego zadania została już zadeklarowana w bloku zewnętrznym, napisać blok dla rozwiązania układu równań

$$2x - y - 3z = 4$$

$$x + y + z = -4$$

$$-x + y = 3$$

przyjmując również zmienne x, y, z jako nielocalne.

4.10. Procedury funkcyjne

Główna idea, dla której procedury funkcyjne zostały wprowadzone jako oddzielna klasa procedur, polega na tym, że - skoro wykonanie procedury funkcyjnej daje w wyniku jedną liczbę jako wartość funkcji, dla której skonstruowano daną procedurę - wygodnie jest wywołanie procedury traktować nie jako instrukcję, ale jako część składową wyrażenia, podobnie jak funkcje standardowe, np. $\sin(x)$ czy $\ln(x)$. Pociąga to jednak za sobą konieczność pewnych zmian w deklaracji procedury.

Skoro wywołanie procedury funkcyjnej ma stanowić część składową wyrażenia, to tym samym musi dawać pewną wartość, której typ musi być zadeklarowany. Robi się to w ten sposób, w deklaracji procedury przed wyrazem procedure umieszcza się deklarację typu, do którego należy wartość tej procedury, deklarację rozpoczyna się od real procedure, integer procedure lub Boolean procedure. W następującej po liście parametrów formalnych nie potrzeba już przewidywać osobnego parametru na wynik, ponieważ jest nim sama nazwa procedury. Natomiast w ciele procedury nazwa ta musi pojawić się co najmniej raz po lewej stronie instrukcji podstawienia, aby procedura otrzymała w wyniku jej wykonania jakąś wartość. Ponadto ciało procedury musi mieć taką konstrukcję, aby we wszystkich możliwych przypadkach wykonania procedury co najmniej jedna z takich instrukcji podstawienia z nazwą procedury po lewej stronie była wykonana. Natomiast każde pojawienie się nazwy procedury inaczej w roli zmiennej po lewej stronie instrukcji podstawienia dzie w ciele procedury powodowało wywołanie tejże procedury, o czym będzie mowa w paragrafie 4.16.

W pozostałych elementach deklaracja procedury funkcyjnej nie różni się od deklaracji procedury niefunkcyjnej.

Wywołanie procedury funkcyjnej różni się od wywołania procedury niefunkcyjnej tylko tym, że stanowi część składową wyrażenia, a nie osobną instrukcję.

Należy na zakończenie zwrócić uwagę na to, że każdą procedurę funkcyjną możemy zastąpić procedurą niefunkcyjną (ale nie na odwrót!). Wystarczy w tym celu: 1° opuścić w deklaracji procedury nazwę typu przed wyrazem procedure, 2° dodać jeden parametr formalny dla wyniku procedury (oczywiście z odpowiednią późniejszą specyfikacją) i w ciele procedury korzystać z niego - a nie z nazwy procedury - jako zmiennej po lewej stronie instrukcji podstawienia służących dla nadania wartości funkcji, dla której procedura została skonstruowana, 3° traktować wywołanie procedury jako oddzielną instrukcję, a nie jako element składowy wyrażeń.

Przykładem może być procedura funkcyjna dla obliczania silni

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$$

i jej wywołanie dla obliczenia wartości

$$\beta = \frac{m!}{n! (m-n)!}$$

Oto deklaracja procedury:

```
integer procedure silnia(k); value k; integer k;
  begin integer i,s;
    s:= 1;
    for i:= 2 step 1 until k do s:= s*i;
    silnia:= s
  end;
```

(Wprowadzenie zmiennej roboczej s jest konieczne, ponieważ nazwa "silnia" nie może tu występować jako zmienna po prawej stronie instrukcji podstawienia).

A oto fragment programu, w którym następuje obliczenie wartości β (beta) przy założeniu, że dany fragment znajduje się w bloku, w którym miały miejsce odpowiednie deklaracje dla m, n i beta:

```
.....
beta:= silnia(m)/silnia(n)/silnia(m-n);
.....
```

Gdybyśmy powyższą procedurę funkcyjną chcieli zastąpić niefunkcyjną, jej deklaracja byłaby następująca:

```
procedure silnia(k,S); value k; integer k,S;
  begin integer i;
    S:= 1;
    for i:= 2 step 1 until k do S:= S*i
  end;
```

(można tutaj obyć się bez zmiennej roboczej s , ponieważ s może występować po prawej stronie instrukcji podstawienia, gdyż nie jest nazwą procedury i nie wywołuje jej).

Fragment programu dla obliczenia β wyglądałby teraz następująco:

```
.....
silnia(m,S1);
silnia(n,S2);
silnia(m-n,S3);
beta:= S1/S2/S3;
.....
```

zakładając uprzednie poczynienie deklaracji dla $m, n, \beta, S1, S2$ i $S3$.

Jako dalszy przykład* rozpatrzmy następującą procedurę dla obliczania iloczynu skalarnego dwu wektorów (iloczynem skalarnym wektorów (a_1, a_2, \dots, a_n) i (b_1, b_2, \dots, b_n) nazywamy liczbę równą $a_1 b_1 + a_2 b_2 + \dots + a_n b_n$), w której dowcipnie wykorzystano możliwość wywoływania parametrów formalnych przez nazwę:

```
real procedure IlSkal(a,b) Rzad:(k,p);
value k; integer k,p; real a,b;
begin real s; s:= 0;
for p:= 1 step 1 until k do s:= s+a*x;
IlSkal:= s end;
```

Działanie tej procedury funkcyjnej poznamy najlepiej na przykładzie jej wywołania. Oto przykładowy fragment programu z takim wywołaniem:

```
.....
x:= 2*IlSkal(A[t,P,u],B[P],10,P)/sqrt(t2+u2);
.....
```

Przedstawiony fragment jest równoważny następującemu:

```
.....
begin integer k;
k:= 10;
begin real s; s:= 0;
```

*Przykład ten przedrukowano z publikacji: J.W. Backus i inni: Revised Report on the Algorithmic Language ALGOL-60, Regnecentralen, Copenhagen 1962, str. 36.

```

for P:= 1 step 1 until k do s:= s+A[t,P,u]xB[P];
x:= 2xs/sqrt(t/2+u/2)
end end;
.....

```

zakładając, że uprzednio były poczynione odpowiednie deklaracje dla zmiennych t, u, P, x oraz tablic A i B , oraz że zmienne t i n miały określone wartości w momencie wywołania procedury.

4.11. Zadanie

Jaką wartość otrzymalibyśmy na x w ostatnim przykładzie poprzedniego paragrafu, gdyby w deklaracji procedury `IlSkal` dodać value a, b ?

4.12. Zadanie

Procedurę `IlSkal` z ostatniego paragrafu zastąpić procedurą niefunkcyjną.

4.13. Zadanie

Ułożyć procedurę dla obliczania tangensa kąta podanego w radianach.

4.14. Zadanie

Ułożyć procedurę dla obliczania wartości dowolnej funkcji $F(x)$ metodą interpolacji kwadratowej, gdy dane są wartości tej funkcji w punktach

$$x_i = x_0 + ih, \quad h > 0, \quad h = \text{const.}, \quad i = 0, 1, \dots, n.$$

w formie tablicy, w której na i -ty wyraz podstawiona została wartość

$$y_i = F(x_0 + ih).$$

Należy skorzystać ze wzoru interpolacyjnego Lagrange'a:

$$F(x) = y_m \cdot \frac{t-1}{2}(t-2) + y_{m+1} \cdot t(t-2) + y_{m+2} \cdot t \cdot \frac{t-1}{2}$$

gdzie

$$t = \frac{x - x_m}{h}$$

Procedurę należy tak skonstruować, by

$$0 \leq t < 1,$$

gdyż wtedy błędy interpolacji są najmniejsze.

4.15. Zadanie

Ułożyć procedurę na całkowanie funkcji metodą Simpsona.
Dla całki

$$I = \int_a^b f(x) dx$$

metoda ta polega na obliczeniu wartości przybliżonej I ze wzoru

$$I = \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \dots + 2y_{n-2} + 4y_{n-1} + y_n)$$

gdzie

$$h = \frac{b-a}{n}, \quad y_i = f(a+ih) \quad \text{dla } i=0,1,2,\dots,n,$$

a liczbę n dobiera się tak, aby dokładność przybliżenia była dostateczna (sprawę dokładności metody zostawiamy tu na uboczu). Liczba n musi być parzysta.

Skorzystać z ułożonej procedury dla obliczenia całki

$$I = \int_2^3 \frac{dx}{\ln x}$$

dzieląc przedział całkowania na 20 części.

Zadanie rozwiązać w dwu wariantach:

- 1) funkcję całkowaną wprowadzić jako parametr formalny, będący nazwą procedury funkcyjnej,
- 2) funkcję całkowaną potraktować jako wyrażenie podstawiane jako parametr aktualny w miejscu parametru formalnego, będącego nazwą zmiennej.

4.16. Procedury rekurencyjne

ALGOL dopuszcza możliwość wywołania procedury w ciele tejże procedury. Procedurę nazywamy wtedy rekurencyjną.

Na przykład procedurę dla obliczania silni możemy napisać jako procedurę rekurencyjną:

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

Prześledzimy wykonanie tej procedury np. w instrukcji

```
x:= Silnia(5);
```

Wykonanie tej instrukcji jest równoważne wykonaniu następującego fragmentu programu:

```
begin integer n;  
n:= 5;  
x:= if n=0 then 1 else n×Silnia(n-1) end;
```

czyli

```
x:= 5×Silnia(4);
```

czyli

```
begin integer n;  
n:= 4;  
x:= 5×(if n=0 then 1 else n×Silnia(n-1)) end;
```

czyli

```
x:= 5×4×Silnia(3);
```

Przedłużając tę analizę dochodzimy do równoznaczności instrukcji wyjściowej z instrukcją

```
x:= 5×4×3×2×1×1;
```

Procedury rekurencyjne ułatwiają częstokroć napisanie ciała procedury, ale zazwyczaj nie są ekonomiczne, gdyż wielokrotne wywoływanie procedury z jednej strony wymaga rezerwacji znacznej liczby komórek w pamięci maszyny, a z drugiej powiększa liczbę operacji koniecznych do wykonania przez maszynę.

4.17. Zadanie

Ułożyć procedurę rekurencyjną dla obliczania całek o postaci

$$I = \int_a^b \operatorname{tg}^n(x) dx \quad /n \text{ liczba naturalna/}$$

wykorzystując wzór rekurencyjny dla $n \geq 2$

$$\int_a^b \operatorname{tg}^n x dx = \frac{\operatorname{tg}^{n-1} b - \operatorname{tg}^{n-1} a}{n-1} - \int_a^b \operatorname{tg}^{n-2} x dx ,$$

i wzory

$$\int_a^b \operatorname{tg}^0 x dx = b-a; \quad \int_a^b \operatorname{tg}^1 x dx = \ln \left| \frac{\cos a}{\cos b} \right|;$$

4.18. Zadanie

Ułożyć procedurę rekurencyjną na obliczanie współczynników binomialnych C_m^n wykorzystując wzór rekurencyjny

$$C_m^n = C_{m-1}^{n-1} + C_{m-1}^n$$

oraz wartości

$$C_m^0 = C_m^m = 1 .$$

4.19. Zadanie

Ułożyć procedurę na obliczanie wartości n -tego wielomianu Czebyszewa $T_n(x)$ w dowolnym punkcie x , wykorzystując wzór rekurencyjny

$$T_n(x) - xT_{n-1}(x) + \frac{1}{4} T_{n-2}(x) = 0$$

oraz wzory

$$T_0(x) = 2 , \quad T_1(x) = x .$$

4.20. Rekurencyjne wywoływanie procedur

Procedur rekurencyjnych nie należy mylić z rekurencyjnym wywoływaniem procedur. W pierwszym przypadku rekurencja zachodzi w deklaracji procedury, w drugim w jej wywoływaniu. W niniejszym paragrafie poznamy rekurencyjne wywoływanie procedur. Polega ono na tym, że co najmniej jednym z parametrów aktualnych w wywoływaniu procedury jest wywołanie tejże procedury.

Przykład:

Mając procedurę dla obliczania sumy szeregu

$$S = \sum_{i=m}^n a_i$$

w postaci

```
real procedure SUMA(a,i,m,n);
  value m,n; real a; integer i,m,n;
  begin real s; s:= 0;
  for i:= m step 1 until n do s:= s+a;
  SUMA:= s end;
```

możemy obliczyć np. sumę podwójną

$$P = \sum_{j=1}^{13} \sum_{k=2}^{17} (j^2 + j \times k + k^2) / \text{sqrt}(j^2 + 1)$$

Do tego celu wystarczy jedna tylko instrukcja:

```
P:= SUMA(SUMA((j^2+j*k+k^2)/sqrt(j^2+1),k,2,17),j,1,13);
```

Prześledźmy wykonanie tej instrukcji.

Wywołanie zewnętrzne procedury SUMA następuje w taki sposób, że dana instrukcja jest równoważna blokowi:

```
begin integer m,n; real SUMA;
m:= 1; n:= 13;
begin real s; s:= 0;
for j:= m step 1 until n do
  s:= s+SUMA((j^2+j*k+k^2)/sqrt(j^2+1),k,2,17);
SUMA:= s end;
P:= SUMA end;
```

Natomiast wywołanie wewnętrzne procedury SUMA sprawia, że powyższy blok jest równoważny następującemu:


```

begin integer m,n; real SUMA;
m:= 1; n:= 13;
begin real s; s:= 0;
for j:= m step 1 until n do
    begin integer m,n; real SUMA;
    m:= 2; n:= 17;
    begin real s; s:= 0;
    for k:= m step 1 until n do
        s:= s+(j2+j×k+k2)/sqrt(j2+1);
    SUMA:= s
    end;
    s:= s+SUMA
    end;
SUMA:= s end;
P:= SUMA end;

```

Aby zrozumieć działanie takiego bloku wygodnie jest za-
uważyć, że w bloku można zmieniać nazwy zmiennych lokalnych
bez wpływu na działanie całości. W ten sposób z łatwością po-
zbywamy się konkurencji zmiennych o tych samych nazwach,
o której była mowa w paragrafie 3.15:

```

begin integer m,n; real SUMA;
m:= 1; n:= 13;
begin real s; s:= 0;
for j:= m step 1 until n do
    begin integer p,q; real suma;
    p:= 2; q:= 17;
    begin real t; t:= 0;
    for k:= p step 1 until q do
        t:= t+(j2+j×k+k2)/sqrt(j2+1);
    suma:= t

```

```

end;
s:= s+suma
end;
SUMA:= s end; ,
P:= SUMA end;

```

Sposób powyższy możemy stosować we wszystkich skomplikowanych przypadkach konkurencji parametrów o tych samych nazwach.

4.21. Zadanie

Wykorzystując procedurę na całkowanie numeryczne podaną w zadaniu 4.15 obliczyć całkę:

$$I = \int_1^5 dx \int_{-\sqrt{4+x^2}}^{\sqrt{x}} \sin \sqrt{x^2+y^2} dy$$

przyjmując, że krok całkowania numerycznego jest około 0.1.

4.22. Zadanie

Wykorzystując rekurencyjne wywoływanie procedur zaprojektować obliczenie

$$x = \sqrt{3t+2} \sqrt{3t+2} \sqrt{3t+2} \sqrt{2t^2+3t+2}$$

zakładając, że zmienne x i t są dla projektowanego bloku nielokalne.

4.23. Zadanie

Jakie obliczenie realizuje program

```

begin real procedure U(t); value t; real t;
U:= 1/(1+t);
y:= aXU(bXU(cXU(dXU(eXU(x)))) end;

```

w którym zmienne a, b, c, d, e, x, y są nielokalne?

Jak można inaczej ułożyć program, aby uniknąć rekurencyjnego wywoływania procedury?

4.24. Trudności wynikające z nielokalności parametrów w procedurach

Gdy w ciele procedury pojawiają się parametry nielocalne, należy bardzo uważać na konsekwencje mogące stąd wypływać. Jako przykład rozpatrzmy procedurę o następującej deklaracji:

```
real procedure PRZEMYTNIK(a); value a; real a;
begin PRZEMYTNIK:= a/2+a+1;
X:= 10
end PRZEMYTNIKA;
```

w której została użyta zmienna nielokalna X. Jeśli w programie użyjemy tej procedury np. w instrukcji

```
P:= PRZEMYTNIK(A)+2
```

spowodujemy zmianę X w sposób "przemysłowy", gdyż w danej instrukcji zmienna X bezpośrednio w ogóle nie występuje. Należy zatem bacznie uważać, czy procedury zawierają parametry nielocalne, i w przypadku, gdy zawierają, specjalnie śledzić ich działanie.

Parametry nielocalne w procedurach mogą spowodować jeszcze jeden efekt: mogą "zdynamizować" wyrażenia, tzn. wprowadzić zależność wartości wyrażenia od kolejności wykonywania działań nawet w tych sytuacjach, gdy normalnie w przypadku niewystępowania procedur z parametrami nielocalnymi, wyniki otrzymujemy jednakowe. Na przykład wywołanie podanej wyżej procedury PRZEMYTNIK w instrukcji

```
Psikus:= PRZEMYTNIK(A)*X;
```

dałoby inną wartość zmiennej Psikus aniżeli w instrukcji

```
Psikus:= X*PRZEMYTNIK(A);
```

pomimo przemienności mnożenia w innych przypadkach, tzn. gdy nie użyto procedury o parametrach nielocalnych. Załóżmy więc w chwili wywoływania powyższych instrukcji mieliśmy następujące wartości zmiennych

```
X = 5;      A = 2;
```

W tej sytuacji wywołanie PRZEMYTNIK(A) zmienia wartość na

```
X = 10
```

Ponieważ maszyna odczytuje wyrażenia od lewej strony prawej, w instrukcji

```
Psikus:= PRZEMYTNIK(A)*X
```

będzie wykonane działanie

```
Psikus:= PRZEMYTNIK(2)*10
```

dające wartość Psikus=70, natomiast w instrukcji

Psikus:= X*PRZEMYTNIK(A)

działanie

Psikus:= 5*PRZEMYTNIK(2)

dające wartość Psikus=35, gdyż w tym ostatnim przypadku maszyna pobiera wartość zmiennej X przed wywołaniem procedury PRZEMYTNIK.

Z powyższych przykładów widać, jak należy uważać w przypadku procedur zawierających parametry nielokalne.

W stosunku do procedur mają zastosowanie również reguły dotyczące przestrzegania lokalności zmiennych, etykiet, przełączników, analogicznie do podanych w paragrafie 3.15. Najłatwiej jest rozszyfrować kolizje parametrów o tych samych nazwach, stosując metodę zaproponowaną w zakończeniu paragrafu 4.13.

4.25. Zadanie

Obliczyć końcową wartość zmiennej nielokalnej x w programie:

begin integer i; procedure p; x:= x+1; i:= 4; x:= 1; p;

begin integer i; i:= 2; p end end;