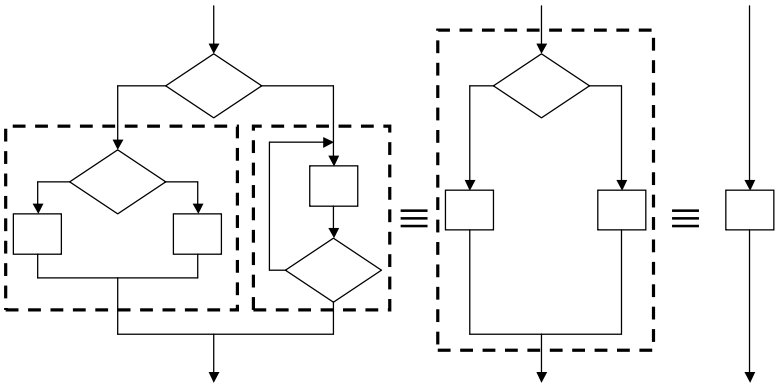
Algorytmy i struktury danych

Funkcje i procedury
Zasięg zmiennych
Rekurencja

Witold Marańda
maranda@dmcs.p.lodz.pl

1

Modularyzacja programu



Algorytmy strukturalne można redukować, zastępując złożone fragmenty blokami o wyższym stopniu funkcjonalności.

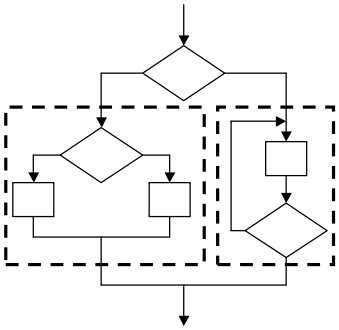
Podział na moduły ułatwia testowanie całego programu, gdyż można zapewnić oddzielnie poprawność modułów, a następnie całej, zredukowanej sieci.

2

Funkcje i procedury

Funkcje i procedury umożliwiają modułową realizację programu

- Dekompozycja problemu na prostsze części
- Większa czytelność
- Unikanie powtórzeń

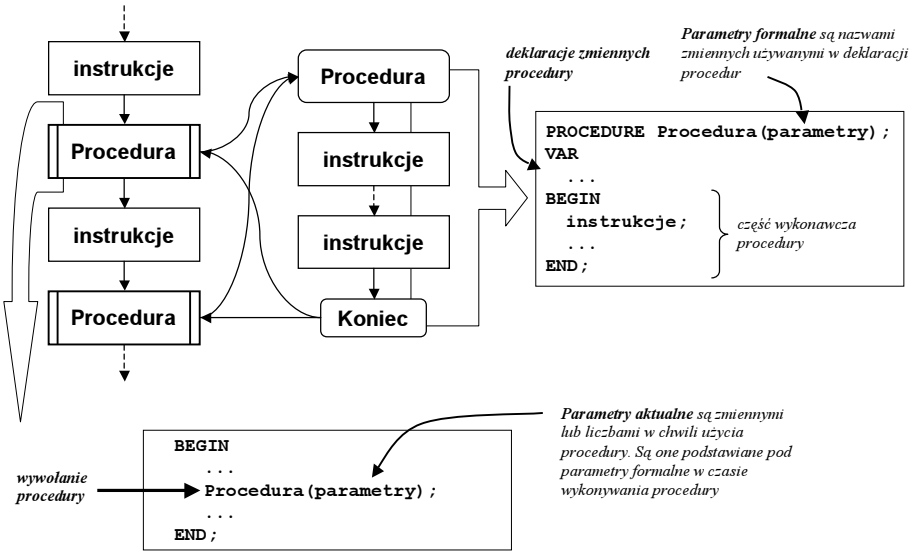


Funkcja - moduł programowy, który może pobierać wiele argumentów (danych wejściowych), zwracający tylko jedną wartość jako wynik swojego działania

Procedura - moduł programowy, który może pobierać wiele argumentów (danych wejściowych), ale nie zwraca żadnego wyniku i może być używana jak normalna instrukcja języka.

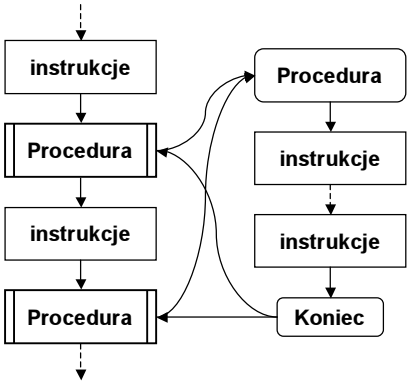
3

Procedury



4

Procedury



Przykład:

```
Ramka();                          PROCEDURE Ramka();
FOR i:=1 to N DO                  BEGIN
  Writeln(x[i]);                    writeln('==========');
Ramka();                          END;
```

5

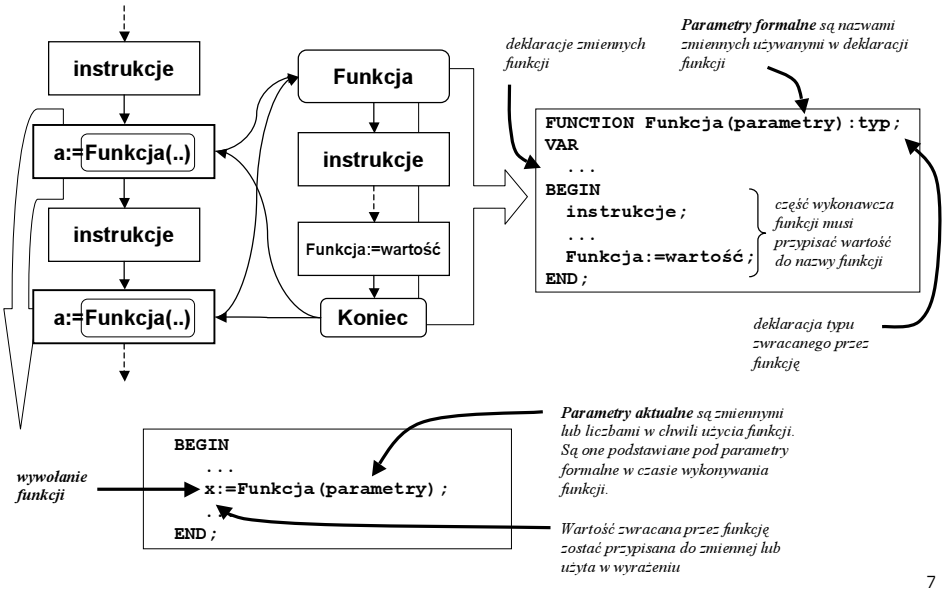
Procedury

Przykłady:

```
PROCEDURE Ramka();
BEGIN
  writeln("==========");
END;

PROCEDURE Tablica(tab:array[1..N] of REAL);
VAR i:INTEGER;
BEGIN
  FOR i:=1 TO N DO
     writeln('[',i,']=',tab[i])
END;

VAR x,y : ARRAY[1..N] of REAL;
...
  Ramka();
  Tablica(x);
  Ramka();
  Tablica(y)
  Ramka();
...
```

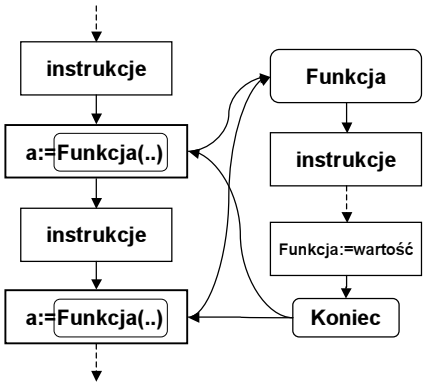
6

Funkcje



7

Funkcje



Przykład:

```
FOR k:=1 TO 10 DO
BEGIN
  s:=Silnia(k);
  writeln(s);
END;
```

```
FUNCTION Silnia(n:BYTE):INTEGER;
VAR i:BYTE;
BEGIN
  Silnia:=1;
  FOR i:=2 TO n DO
    silnia:=Silnia*i;
END;
```

8

Procedury, funkcje - przykład

```
PROGRAM Tablice;

CONST N=3;
TYPE Tablica=ARRAY[1..N] OF REAL;

PROCEDURE Wypelnij(var tab:Tablica;x:REAL);
VAR k:INTEGER;
BEGIN
        FOR k:=1 TO N DO
                tab[k]:=x;
END;

PROCEDURE Wprowadz(var tab:Tablica);
VAR k:INTEGER;
BEGIN
        FOR k:=1 TO N DO
                Read(tab[k]);
END;

PROCEDURE Wydrukuj(tab:Tablica);
VAR k:INTEGER;
BEGIN
        FOR k:=1 TO N DO
                Writeln('[',k,']=',tab[k]);
END;
```

```
FUNCTION Srednia(tab:Tablica):REAL;
VAR k:INTEGER;
BEGIN
        Srednia:=0;
        FOR k:=1 TO N DO
                Srednia:=Srednia+tab[k];
        Srednia:=Srednia/k
END;

FUNCTION Najwiekszy(tab:Tablica):REAL;
VAR k:INTEGER;
    max:REAL;
BEGIN
        Max:=tab[1];
        FOR k:=2 TO N DO
                IF tab[k]>Max THEN
                        Max:=tab[k];
        Najwiekszy:=Max;
END;

VAR a,b:Tablica;
BEGIN
        Wypelnij(b,1);
        Wprowadz(a);
        Wydrukuj(a);
        Wydrukuj(b);
        Writeln(Srednia(a));
        Writeln(Najwiekszy(a));
END.
```

Przekazywanie parametrów

Przekazywanie parametrów możliwe jest na dwa sposoby:

- Przez wartość - procedura (funkcja) otrzymuje kopię zmiennych i nie może zmienić wartości oryginalnej zmiennej
- Przez zmienną (wskaźnik) - procedura (funkcja) otrzymuje oryginalną zmienną (wskazanie na zmienną) i może zmienić wartość oryginalnej zmiennej
  (co oznacza się słowem kluczowym **VAR** przed parametrem formalnym)

Przykłady:

```
VAR
   x:REAL;

PROCEDURE F(x:REAL)
BEGIN
   x:=1;
END

x:=0;
F(x);
writeln(x);
```

wydrukowanie wartości **0**

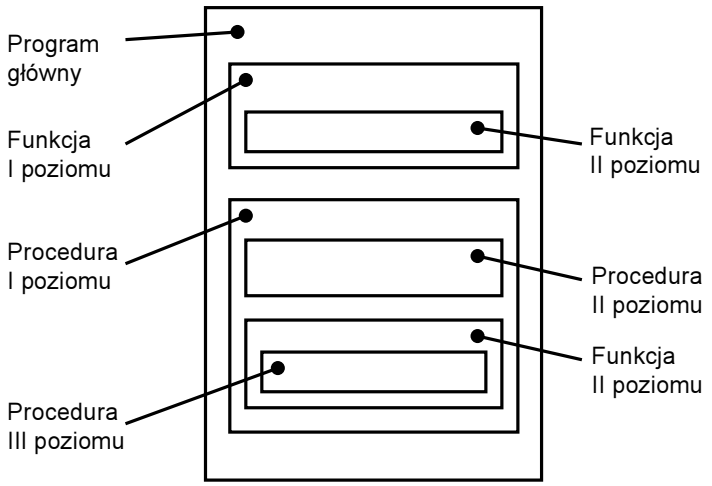
```
VAR
   x:REAL;

PROCEDURE F(VAR x:REAL)
BEGIN
   x:=1;
END

x:=0;
F(x);
writeln(x);
```

wydrukowanie wartości **1**

10

PASCAL - struktura blokowa programu

Język PASCAL ma strukturę blokową, tzn. dopuszcza zagnieżdżanie funkcji i procedur



11

Zasięg zmiennych

Język PASCAL ma klarowne reguły zasięgu zmiennych, związaną z jego strukturą blokową

```
VAR
   X : real;

   VAR
      Y : real;

   X – zmienna OK (globalna)
   Y – zmienna OK (lokalna)

   X – zmienna OK (globalna)
   Y – ???

      X – zmienna OK (globalna)
      Y – ???

X – zmienna OK (globalna/lokalna)
Y – ???
```

Zasięg zmiennej (identyfikatora) oznacza fragment programu, w którym można używać tę zmienną i oznacza ona tę samą zmienną

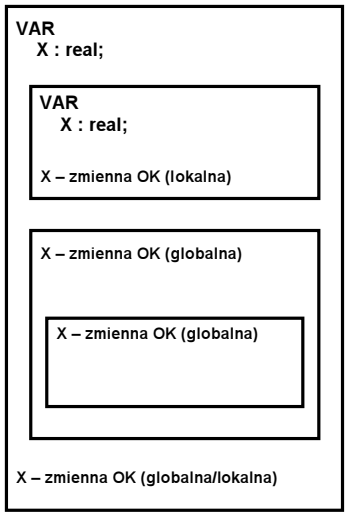
Zasięg zmiennej w PASCALU obejmuje blok, w którym tę zmienną zadeklarowano oraz wszystkie bloki w nim zagnieżdżone.

Zmienne globalne to zmienne zadeklarowane w bloku głównym programu, gdyż są widoczne we wszystkich blokach.

Zmienne lokalne to zmienne zadeklarowane w danym bloku, ale nie widoczne na zewnątrz tego bloku.

12

Przesłanianie zmiennych



Jeśli zmienna lokalna ma taką samą nazwę jak zmienna w bloku zewnętrznym, to zmienna zewnętrzna jest **przesłonięta** przez zmienną lokalną.

Na zewnątrz tego bloku nadal widoczna jest zmienna zewnętrzna.

13

Zasięg zmiennych a przekazywanie parametrów

Procedury i funkcje w mogą komunikować się z programem wywołujących:

- **poprzez parametry** – gdy trzeba przekazywać różne zmienne lub wartości, poprawia to również czytelność procedur i funkcji, gdyż posiadają one jasno zapisany interfejs komunikacji.
- **poprzez zmienne globalne** – gdy dotyczy to zmiennych szczególnie ważnych w obrębie całego programu lub gdy dotyczy to dużych rozmiarowo struktur danych, ale ogólnie zmniejsza czytelność programu.

Poprawna organizacja wymiany informacji pomiędzy blokami programu jest kwestią kompromisu i stylu programowania.

14

Zasięgi - przykład

```
PROGRAM scope(input,output);
VAR
    a,b,c : INTEGER;

PROCEDURE aa(VAR a,b: INTEGER);
BEGIN
    a:=2;
    b:=b-1;
END;

PROCEDURE bb(VAR c: INTEGER);
BEGIN
    c:=c+a;
END;

BEGIN
    a:=1;
    aa(b,a);
    aa(c,b);
    bb(b);
    writeln(a, b, c);
END.
```

Jakie liczby wydrukuje ten program ???

15

Rekurencja

Rekurencja (inaczej rekursja - *ang. recursion*) oznacza odwoływanie się funkcji lub procedury do samej siebie

```
PROCEDURE Procedura(parametry);
BEGIN
 ...
 Procedura(parametry); {wywołanie rekurencyjne}
 ...
END;
```

Przykład rekurencji: obliczanie silni według wzoru:

n!=n(n-1)!*

W celu obliczenia *n!*, należy najpierw obliczyć *(n-1)!*

```
FUNCTION Silnia(n:BYTE):LONGINT;
BEGIN
        IF n<2 THEN
                Silnia:=1
        ELSE
                Silnia:=n*Silnia(n-1)
END;
```

16

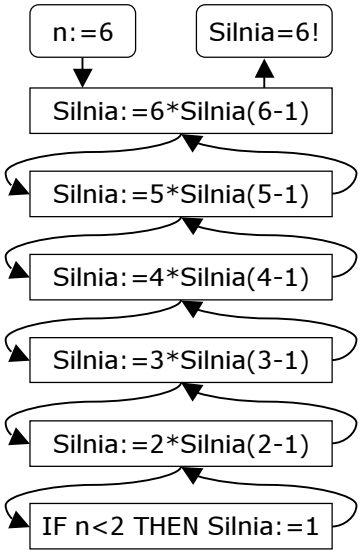
Rekurencja – przykład

Podczas obliczania silni liczby 6, wywołana zostaje funkcja silnia z parametrem 6.

Wewnątrz tej funkcji następuje rekurencyjne wywołanie tej samej funkcji
(a więc samej siebie), ale z parametrem *n-1* czyli 5.

W kolejnym kroku znów wywoływana jest silnia dla n-1, aż do momentu gdy n<2. Wówczas spełniony zostaje warunek i najbardziej zagnieżdżona funkcja silnia zwraca wynik 1.

Następnie wszystkie wcześniej wywołane funkcje zwracają sobie po kolei wyniki, a na końcu funkcja silnia zwraca do programu głównego wynik dla n=6.



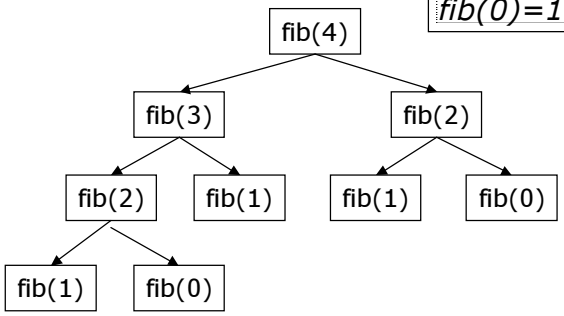
17

Rekurencja - przykład

Ciąg Fibonacciego

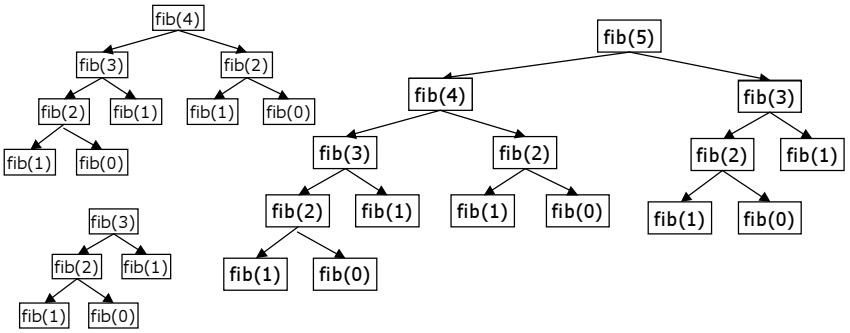
1, 1, 2, 3, 5, 8, 13 ,21 ,34 ,55 ,...

fib(n) := fib(n-2) + fib(n-1)
fib(1)=1
fib(0)=1



18

Rekurencja - przykład

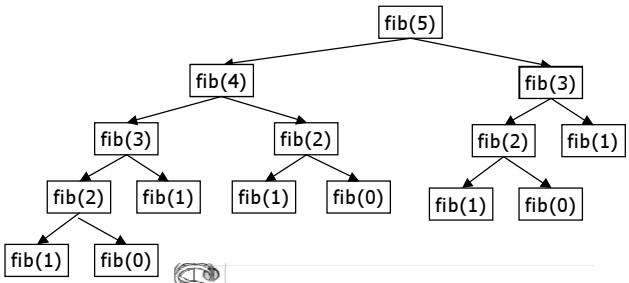



```
FUNCTION Fibonacci(n:INTEGER):INTEGER;
BEGIN
        IF n<2 THEN
                Fibonacci:=1
        ELSE
                Fibonacci:=Fibonacci(n-2)+Fibonacci(n-1)
END;
```

19

Ciąg Fibonacciego – nienajlepszy przykład wykorzystania rekurencji?

Obliczanie liczb ze pomocą powyższej funkcji powoduje wywołanie procedury Fibonacci wielokrotnie obliczającej te same liczby Fibonacciego, co jest bardzo nieefektywne.




```
FUNCTION
Fibonacci(n:INTEGER):LONGINT;
VAR f0,f1:LONGINT;
BEGIN
   f0:=1;
   f1:=1;
   FOR k:=1 TO n DO
   BEGIN
         Fibonacci:=f0+f1;
         f0:=f1;
         f1:=Fibonacci;
   END
END;
```

Rekurencja nie jest właściwym rozwiązaniem problemu (nawet jeśli problem jest łatwy do rekurencyjnego zdefiniowania) gdy liczba wywołań jest nierozsądnie duża lub dochodzi do powtórzeń obliczeń.

20