Podprogramy

Wstęp do Informatyki i Programowania

Maciek Gębala

17 października 2024

Notatki

Podprogramy

Jednym z pierwszych kroków aby pisać sprawnie programy jest ich strukturyzacja, czyli podział na podprogramy (funkcje i procedury).

Procedura to wyodrębniona część programu, wykonująca się na podstawie podanych parametrów i nie zwracająca wartości (nie może być traktowana jako wyrażenie).

Funkcja to wyodrębniona część programu, wykonująca się na podstawie podanych parametrów i zwracająca jakąś wartość (może być traktowana jako wyrażenie).

Zakładamy, że dobrze napisany podprogram nie zależy od wartości/zmiennych zewnętrznych (globalnych), czyli nie będących parametrami wywołania.

Notatki

Język C

Formalnie w języku C definiujemy tylko funkcje, procedury to funkcje zwracające typ `void` (pusty). Wartości funkcji nie muszą być odbierane.

Definiowanie funkcji

```
<typ wyniku> <nazwa>(lista parametrów) { <implementacja>
};
```

Zwracanie wyniku

Wynik zwracamy korzystając z kluczowego słowa `return` (które jednocześnie przerywa wykonanie funkcji).

W C nie można definiować funkcji w funkcjach.

Funkcja `int main(...)` jest zawsze w C wywoływana domyślnie jako początek programu.

Notatki

Język C

Przekazywanie parametrów

W C parametry przekazuje się przez kopię wartości - zmiany wewnątrz funkcji nie są przekazywane na zewnątrz.

Jeśli chcemy zmieniać wartość parametrów to przekazujemy w parametrach adres zmiennej i wykonujemy operacje na wskazaniach tego adresu (operatory: `&` do pobrania adresu i $*$ do wskazania adresu, zobacz `scanf`)

Więcej szczegółów będzie na wykładach o złożonych i dynamicznych typach danych.

Notatki

Przykład w języku C

gcdtest.c

```
1  #include <stdio.h>
2  void read(int *a, int *b) {
3    printf("Podaj pierwszą liczbę: ");
4    scanf("%d", a);
5    printf("Podaj drugą liczbę: ");
6    scanf("%d", b);
7  }
8  int gcd(int a, int b) {
9    int c;
10   while ( b != 0 ) {
11     if ( a < b ) {
12       c = a;
13       a = b;
14       b = c;
15     } else {
16       a = a - b;
17     }
18   }
19   return a;
20 }
```


Notatki

Przykład w języku C

gcdtest.c

```
21 void write(int a) {
22   printf("Największy wspólny dzielnik to %d\n", a);
23 }
24 int main() {
25   int a, b, c;
26   read(&a, &b);
27   c = gcd(a, b);
28   write(c);
29
30   return 0;
31 }
```


Notatki

Język Ada

W Adzie mamy osobne słowa kluczowe do definiowania procedur i funkcji.

Definiowanie procedury

`procedure <nazwa>(lista parametrów) is <implementacja>`

Definiowanie funkcji

`function <nazwa>(lista parametrów) return <typ> is <implementacja>`

Zwracanie wyniku

Wynik zwracamy korzystając z kluczowego słowa `return` (które jednocześnie przerywa wykonanie funkcji).

W można definiować procedury/funkcje wewnątrz procedur i funkcji.

Należy pamiętać, że program w Adzie jest zawsze jedną procedurą.

Notatki

Język Ada

Przekazywanie parametrów

W Adzie parametry przekazuje się przez referencję na na trzy sposoby:

1. `IN` – tryb domyślny, wewnątrz podprogramu parametry traktowane jako stałe;
2. `OUT` – tryb z możliwością przekazania wartości na zewnątrz ale traktowany jak niezainicjowany wewnątrz;
3. `IN-OUT` – pełny tryb referencji, inna nazwa zmiennej podanej w wywołaniu.

Notatki

Przykład w języku Ada

gcdtest.adb

```
1  with Ada.Text_IO; use Ada.Text_IO;
2  with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
3  procedure gcdtest is
4    procedure read(a : out Integer; b : out Integer) is
5    begin
6      Put("Podaj␣pierwszą␣liczbę:␣");
7      Get(a);
8      Put("Podaj␣drugą␣liczbę:␣");
9      Get(b);
10   end read;
```


Notatki

Przykład w języku Ada

gcdtest.adb

```
11   function gcd(x : in Integer; y : in Integer) return Integer is
12     a, b, c : Integer;
13   begin
14     a := x;
15     b := y;
16     while b /= 0 loop
17       if a < b then
18         c := a;
19         a := b;
20         b := c;
21       else
22         a := a - b;
23       end if;
24     end loop;
25     return a;
26   end gcd;
```


Notatki

Przykład w języku Ada

gcdtest.adb

```
27   procedure write(c : in Integer) is
28   begin
29     Put_Line("Największy␣wspólny␣dzielnik␣to␣" & c'Image);
30   end write;
31   a, b, c : Integer;
32 begin
33   read( a, b );
34   c := gcd( a, b );
35   write( c );
36 end gcdtest;
```


Notatki

Język Python

W Pythonie podprogramy definiujemy słowem kluczowym `def`

Definiowanie podprogramu

```
def <nazwa>(lista parametrów) :     <implementacja>
```

Zwracanie wyniku

Wynik zwracamy korzystając z kluczowego słowa `return` (które jednocześnie przerywa wykonanie funkcji). Podprogram nie ma w deklaracji, że jest funkcją.

W Pythonie można definiować podprogramy wewnątrz podprogramów.

Notatki

Język Python

Przekazywanie parametrów

Proste typy są przekazywane przez wartość a złożone przez referencję (i te są modyfikowalne na zewnątrz).

Notatki

Przykład w języku Python

gcdtest.py

```
1  def read() :
2      a = int(input("Podaj pierwszą liczbę: "))
3      b = int(input("Podaj drugą liczbę: "))
4      return a, b
5
6  def gcd( a, b ) :
7      while b != 0 :
8          if a < b :
9              a, b = b, a
10         else :
11             a = a - b
12     return a
13
14 def write( c ) :
15     print(f"Największy wspólny dzielnik to {c}")
16
17 def main():
18     a,b = read()
19     c = gcd( a, b )
20     write( c )
21
22 if __name__ == "__main__":
23     main()
```


Notatki

Przykład w języku Python (2)

gcdtest2.py

```
1  def main():
2      def read() :
3          a = int(input("Podaj pierwszą liczbę: "))
4          b = int(input("Podaj drugą liczbę: "))
5          return a, b
6
7      def gcd( a, b ) :
8          while b != 0 :
9              if a < b :
10                 a, b = b, a
11             else :
12                 a = a - b
13         return a
14
15     def write( c ) :
16         print(f"Największy wspólny dzielnik to {c}")
17
18     a,b = read()
19     c = gcd( a, b )
20     write( c )
21
22 if __name__ == "__main__":
23     main()
```


Notatki

Rekurencja

Wszystkie trzy języki pozwalają na definiowanie wywołań rekurencyjnych, czyli wywołanie podprogramu wewnątrz własnej definicji.

Należy zadbać, aby nie spowodować nieskończonego ciągu odwołań rekurencyjnych.

W praktyce unika się wywołań rekurencyjnych, zastępując je iteracjami w pętli.

Notatki

Przykład rekurencji - liczby Fibonacciego

Rekurencyjna definicja

$$
\begin{aligned}
Fibonacci(0) &= 0 \\
Fibonacci(1) &= 1 \\
Fibonacci(n) &= Fibonacci(n-1) + Fibonacci(n-2) \qquad \text{dla } n > 1
\end{aligned}
$$

Notatki

Implementacja w języku C

fib.c

```c
#include <stdio.h>
unsigned fibonacci(unsigned n) {
  if ( n <= 1 ) return n;
  return fibonacci(n-1) + fibonacci(n-2);
}
int main() {
  unsigned n, m;

  printf("Podaj liczbę: ");
  scanf("%u", &n);
  m = fibonacci(n);
  printf("Fib(%u)=%u\n", n, m);

  return 0;
}
```


Notatki

Implementacja w języku Ada

fib.adb

```ada
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
procedure fib is
  function fibonacci(n : in Natural) return Natural is
  begin
    if n<=1 then
      return n;
    end if;
    return fibonacci(n-1) + fibonacci(n-2);
  end fibonacci;
  n, m : Natural;
begin
  Put("Podaj liczbę: ");
  Get(n);
  m := fibonacci(n);
  Put_Line("Fib(" & n'Image & ")=" & m'Image);
end fib;
```


Notatki

Implementacja w języku Python

fib.py

```python
def fibonacci( n ) :
    if n <= 1 :
        return n
    return fibonacci(n-1) + fibonacci(n-2)

def main():
    n = int(input("Podaj liczbę: "))
    m = fibonacci(n)
    print(f"Fib({n})={m}")

if __name__ == "__main__":
    main()
```


Notatki