Programowanie strukturalne - Wykład 3

Funkcja

Funkcje w C

Ogólna składnia funkcji

```
1 typ identyfikator (typ1 argument1, typ2 argument2)
2 {
3   /* instrukcje */
4 }
```

procedura - podprogram nie zwracający wyniku, realizujący pewną funkcjonalność

```
1 void identyfikator (typ1 argument1, typ2 argument2)
2 {
3   /* instrukcje */
4 }
```

Instrukcja return powoduje zakończenie wykonywania funkcji i zwrócenie wartości. Może być ona użyta dowolną ilość razy w kodzie funkcji.

Funkcja - podprogram zwracający wynik na podstawie przekazanych argumentów

```
typ identyfikator (typ1 argument1, typ2 argument2)
{
  /* instrukcje */
  return ...;
}
```

Funkcja czysta - funkcja, która nie ma żadnych skutków ubocznych, tzn. nie modyfikuje ani przekazanych argumentów, ani globalnego stanu programu.

Przykłady:

Funkcja bez argumentu zwracająca `int`:

```
1  int foo1()
2  {
3      return 62;
4  }
```

Funkcja z jednym argumentem zwracająca `int`

```
1  int foo2(int a)
2  {
3      return a+4;
4  }
```

Funkcja z dwoma argumentami zwracająca `float`

```
1  float foo3(int a, int b)
2  {
3      return (b+a)/3.0;
4  }
```

Procedura bez argumentu:

```
void foo4()
{
    printf("abc");
}

---

Procedura z jednym argumentem

```c
void foo5(int a)
{
    printf("%d",a*8);
}
```

Wszystkie wcześniejsze funkcje były “czyste”.
```

Czy ta funkcja jest czysta?

```c
#include <stdio.h>
#include <stdlib.h>

void foo10(int a)
{
    a+=5;
    printf("a%d\n",a);
}

int main()
{
    int a = 7;
    printf("b%d\n",a);
    foo10(a);
    printf("c%d\n",a);
    return 0;
}
```

Rekurencja

Rekurencja, zwana także rekursją (ang. recursion, z łac. recurrere, przybiec z powrotem) – odwoływanie się np. funkcji lub definicji do samej siebie.

Przykłady:

- silnia

$$0! = 1, \quad n! = (n-1)! \cdot n$$

- ciąg Fibonacciego

$$F_n := \begin{cases} 0 & \text{dla } n = 0, \\ 1 & \text{dla } n = 1, \\ F_{n-1} + F_{n-2} & \text{dla } n > 1. \end{cases}$$

```c
#include <stdio.h>
#include <stdlib.h>

int silnia(int n)
{
    if (n ==0)
    {
        return 1;
    }
    return n* silnia(n-1);
}

int main()
{
    int a = 5;
    printf("%d\n",silnia(a));
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int fib(int n)
{
    if (n ==0)
        return 0;
    if (n == 1)
        return 1;
    return fib(n-1)+fib(n-2);
}

int main()
{
    int a = 6;
    printf("%d\n",fib(a));
    return 0;
}
```

Zadanie na rekurencję

Napisz funkcję rekurencyjną, która dla otrzymanej w argumencie nieujemnej liczby całkowitej n zwraca wartość elementu o indeksie n ciągu zdefiniowanego w następujący sposób

$$a_0 = a_1 = 1$$

$$a_{3n} = a_n, n > 0$$

$$a_{3n+1} = a_{3n} - 1, n > 0$$

$$a_{3n+2} = a_{3n+1} + 1, n \geqslant 0$$

Stwórz dwa przypadki testowe dla funkcji.

Zasady dobrego programowania

- Keep it Simple Stupid (KISS) - BUZI (Bez Udziwnień Zapisu, Idioto)
- Don’t Repeat Yourself (DRY)
- Tell Don’t Ask
- You Aren’t Gonna Need It (YAGNI)
- Separation of Concerns

KISS

- kod ma być prosty i zrozumiały
- unikanie skomplikowanych zapisów
- nie oznacza upraszczania za wszelką cenę

Don’t Repeat Yourself (DRY)

- unikanie powtórzeń w kodzie – u nas to np. polimorficzność, stałe
- oddzielenie powtarzającej się części od metody
- zalety: unikanie błędów, poprawki możemy dodać w jednym miejscu

Tell Don’t Ask

- związane z hermetyzacją i podziałem obowiązków
- należy mówić obiektom jakie akcje mają wykonywać
- nie należy uzależnia wykonania akcji od stanu w jakim znajdują się obiekty

You Aren’t Gonna Need It (YAGNI)

- tworzenie tego co jest potrzebne i niezbędne
- usuwanie dodatków, które mogą się przydać
- np. usunięcie zbędnych usingów, zwolnienie zasobów, usuwanie nie używanych zmiennych, metod

Cytat: Antoine de Saint-Exupéry

Perfekcję osiąga się wtedy, gdy nie można już nic odjąć, a nie dodać.

Separation of Concerns

- elementy składowe (np. klasy i metody) powinny być rozłączne i mieć oddzielne zastosowanie
- te elementy nie powinny współdzielić odpowiedzialności

Cytat: Albert Einstein

Wszystko trzeba robić tak prosto, jak to tylko jest możliwe, ale nie prościej.

Bibliografia

- https://pl.wikibooks.org/wiki/C/Funkcje, dostęp online 10.03.2020.
- https://pl.wikipedia.org/wiki/Rekurencja, dostęp online 10.03.2020.
- https://pl.wikibooks.org/wiki/C/Instrukcje_steruj%C4%85ce, dostęp online 10.03.2020.
- Richard Reese, Wskaźniki w języku C, Wydawnictwo Helion 2014.
- https://pl.wikibooks.org/wiki/C/Wska%C5%BAniki, dostęp online 15.03.2020.
- http://wazniak.mimuw.edu.pl/index.php?title=Wst%C4%99p_do_programowania_w_j%C4%99zyku_C/Wska%C5%BAniki, dostęp online 15.03.2020.
- https://pl.wikibooks.org/wiki/C/Wska%C5%BAniki_-_wi%C4%99cej, dostęp online 15.03.2020.