Семинар по „Увод в програмирането“

Функции

1. Какво представляват функциите и за какво се използват?

Функция в програмирането представлява парче код, което изпълнява някакво действие. Една функция може да има параметри и може да бъде извикана от друго парче код многократно. Има два типа функции-такива, които **връщат** стойност и такива които **не връщат** стойност(void). До тук ни е известна известна функцията **main()**.

Функии използваме, когато искаме повтарящ се код да го обобщим на едно място с цел да спестим място и нашият код да изглежда по-добре. Предимство на функциите е, че ако имаме някаква грешка, можем да си спестим многократно оправянето и на различени места в кода, ако тя се намира на едно място, във функция. Като цяло функциите ни предоставят една абстракция, която ни позволява да поддържаме кода по-лесно.

2. Синтаксис

```
<return type>  <function name> (parameter1, parameter2)
{
        //function body
}
```

- На мястото на **<return type>** пишем типа на връща на функцията. Той може да е тип данни(int, double, char…) или без връщане на резултат, т.е. void.
- Следващото е **<function name>.** Идентификатор или име на функцията. Така кръщаваме функцията.
- На мястото на скобите **(parameter1, parameter2, …)** описваме какви параметри ще изисква функцията, за да бъде изпълнена.
- В края на тялото на функцията задължително трябва да върнем разултат с оператор **return**, освен ако типа на връщане не е **void**. Иначе ще получим грешка при компилиране или още **undefined behaviour**.

Пример:

```cpp
#include <iostream>

int sum(int a, int b)
{
    return a + b;
}

int main()
{
    int a = 5;
    int b = 2;
    int result = sum(a, b);

    std::cout << result;
    return 0;
}
```

3. Параметри

1) Особености на параметрите

Всеки параметър от своя страна трябва да съдържа тип и име:

parameter1 := **<type> <parameter name>**

Параметрите във функцията не са задължителни. Има значение подредбата на параметрите на функцията за компилатора. Ако при извикване на функция се подадат параметри от тип, различен от този в дефиницията, се получава грешка.

2) Стойности по подразбиране

В дефиницията на дадена функция част от параметрите, а може и всички, имат стойности по подразбиране.

Така изглежда функция със стойности по подразбиране:

```cpp
int sum(int a = 0, int b = 0)
{
    return a + b;
}
```

А това е разликата, когато извикваме функцията със и без параметри:

```cpp
#include <iostream>

int sum(int a = 0, int b = 0)
{
    return a + b;
}

int main()
{
    int a = 5;
    int b = 2;
    int result1 = sum(a, b);
    int result2 = sum();

    std::cout << result1 << "\n";
    std::cout << result2;
    return 0;
}
```

Съответно, когато извикваме функцията без параметри за стойност се взимат тези по подразбиране(a = 0 и b = 0). Резултата на конзолата е следния:

```
7
0
D:\C++\WorkSpace\Debug\WorkSpace.exe (process 8808) exited with code 0.
Press any key to close this window . . .
```

Но има една важна особеност при функциите със стойности по подразбиране!

```cpp
//Valid
int sum(int a, int b);
//Invalid
int sum(int a = 0, int b);

//Valid

int sum(int a = 0, int b = 0)
```

В списъка от параметри на дадена функция след първото срещане на параметър със стойност по подразбиране трябва всички останали параметри да имат също стойност по подразбиране. Иначе се получава компилационна грешка.

4. Function Overloading

Overloading-ът на дадефа функция е друга функция със същото име, но с различни параметри. По време на изпълнение, програмата сама решава коя точно дефиниция на дадената функция да извика според броя и типа на променливите.

```
//First definition
int sum(int a, int b);
//Second definition
double sum(double a , double b);
```