Лекция №10
по дисциплине
«ТЕОРИЯ АЛГОРИТМОВ»

# ФУНКЦИИ

Преподаватель:
Золотоверх Д.О.

# ОПРЕДЕЛЕНИЕ

Функции – это блоки кода, выполняющие определенные операции.

Функция может определять входные параметры, позволяющие вызывающим объектам передавать ей аргументы.

При необходимости функция также может возвращать значение как выходное.

# ЗАЧЕМ

- Повторное использование кода. Функции позволяют один раз определить набор операций, но использовать их можно многократно;
- Меньше ошибок в коде;
- Упрощение, сокращение кода;
- Организация кода. Функции зачастую выполняют одну задачу, из-за чего такой код легко делить на части;
- Простая отладка кода. Если ошибка возникает внутри функции, ее легко найти.

# СВОЙСТВА

- Функцию можно вызывать из любого числа мест в программе;
- Значения, передаваемые в функцию, являются аргументами, типы которых должны быть совместимы с типами параметров в определении функции;
- Длина функции практически не ограничена.

# СИНТАКСИС

Минимальное объявление функции состоит из возвращаемого типа, имени функции и списка параметров (который может быть пустым).

Определение функции состоит из объявления, а также тела.

Если функция ничего не возвращает, ее тип – void.

Синтаксис:

```
возвращаемый_тип название_функции(
        тип_аргумента название_аргумента
) {
        тело функции;
}
```

```
#include <iostream>

void printHello(int n) {
    for (int i = 0; i < n; i++) {
        printf("Hello\n");
    }
}

int main()
{
    printHello(25);
}
```

# АРГУМЕНТЫ, ПАРАМЕТРЫ

Функция может принимать любое количество аргументов (при объявлении функции – параметры);

Параметры перечисляются через запятую;

Внутри функции аргументы функционируют как обычные переменные

Функция может вернуть только одну переменную (но можно возвращать сложные)

```
bool isPrime(int number) {
    for (
        int divider = 2;
        divider < number;
        divider++
    ){
        if (
            number % divider == 0
        ) {
            return false;
        }
    }
    return true;
}
```

# ВОЗВРАТ

Функция, после осуществления вычислений, может возвращать данные.

Для этого необходимо использовать ключевое слово return.

Когда функция достигает return, ее выполнение завершается.

Не обязательно использовать return, когда возвращаемый тип void.

Обязательно использовать return, когда любой другой возвращаемый тип.

```
bool isPrime(int number) {
    for (
        int divider = 2;
        divider < number;
        divider++
    ){
        if (
            number % divider == 0
        ) {
            return false;
        }
    }
    return true;
}
```

# ОБЛАСТЬ ВИДИМОСТИ (SCOPE) ФУНКЦИИ

Область видимости — концепция, определяющая доступность переменных

Переменные, объявленные внутри функции, а так же ее аргументы могут быть использованы только внутри функции.

«Общение с функцией» происходит (зачастую) только через ее параметры и возвращаемые значения.

Но функция видит объявленные переменные вне

```
#include <iostream>


int outside = 1337;

void testScope() {
    int inside = 42;

    printf("%d\n", outside);
    printf("%d\n", inside);
}

int main() {
    testScope();
    printf("%d\n", outside);

    // ОШИБКА
    printf("%d\n", inside);

}
```

# ПЕРЕГРУЗКА ФУНКЦИЙ (OVERLOAD)

C++ позволяет определять несколько функций с одинаковым именем для разных даннх.

Эти функции называются перегруженными функциями.

Таким образом можно написать «одну» функцию, которая принимает разные типы данных и их количество.

```
#include <iostream>

void printData(int data) {
    printf("Integer: %d\n", data);
}

void printData(float data) {
    printf("Float: %f\n", data);
}

void printData(double data) {
    printf("Double: %f\n", data);
}

void printData(char data) {
    printf("Char: %c\n", data);
}

int main() {
    printData(12);
    printData(3.32f);
    printData('c');
}
```

# АРГУМЕНТЫ ПО УМОЛЧАНИЮ

Во многих случаях функции имеют аргументы, которые используются настолько редко, что достаточно значения по умолчанию.

Можно задать аргументы по умолчанию, это позволяет указывать только важные при вызове аргументы.

Аргумент по умолчанию УКАЗЫВАЕТСЯ ПОСЛЕДНИМ

```
#include <iostream>
#include <cmath>


double round(float number, int prec=2) {
    int processed = number * pow(10, prec);
    double rounded = processed / pow(10, prec);
    return rounded;
}

int main() {
    printf(
        "Rounded number: %f\n",
        round(23.3423422f)
    );

    printf(
        "Rounded number: %f\n",
        round(23.3423422f, 3)
    );
}
```

# СПАСИБО ЗА ВНИМАНИЕ!