Пятое занятие

Функции

Функция это

- Именованная часть программы, которая может быть многократно вызвана из другого участка программы.

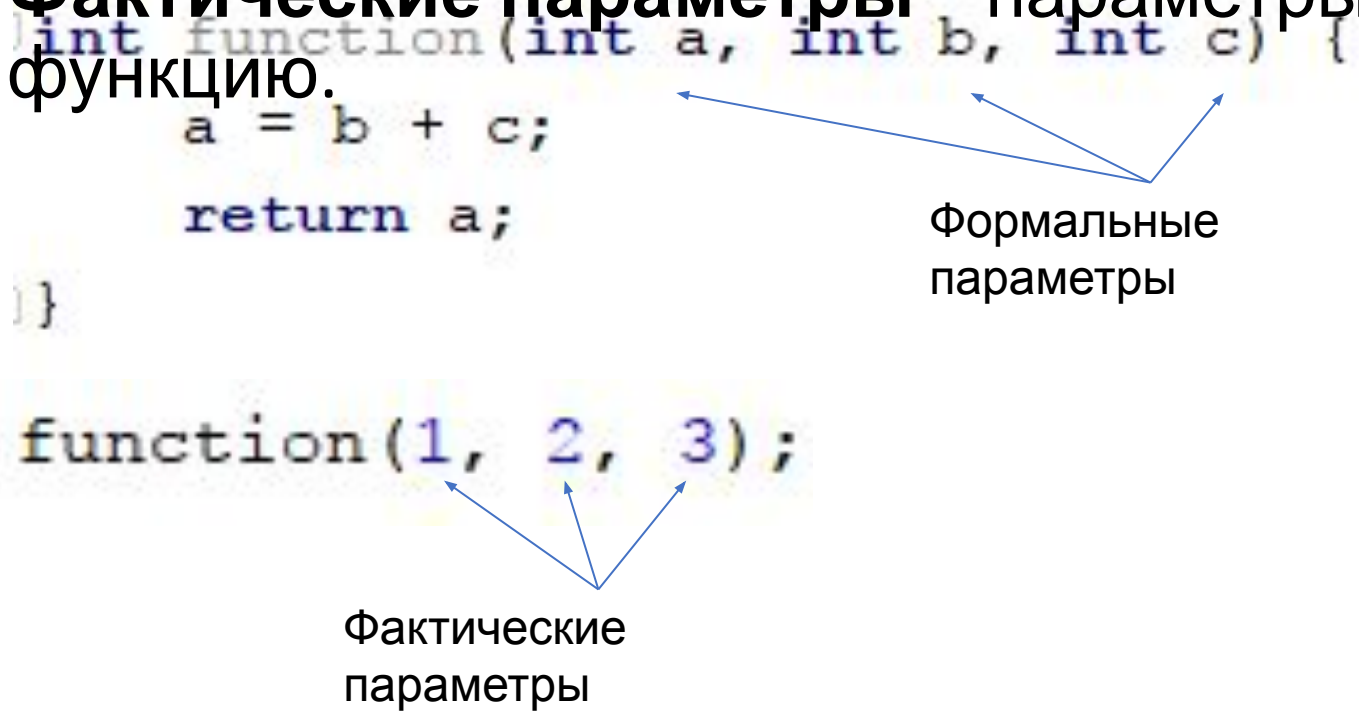
Общий вид

```
<возвращаемый тип> <имя функции>(<тип1> <арг1>, <тип1> <арг2>, ...) {
    <тело функции>
}
```

```
int function(int a, int b, int c) {
    a = b + c;
    return a;
}
```

Учим умные слова

- **Формальные параметры** – параметры описанные в функции.
- **Фактические параметры** – параметры передаваемые в функцию.



Учим умные слова

- **Сигнатура функции** – определяет правила использования функции. Обычно сигнатура представляет собой описание функции, включающее имя функции, перечень формальных параметров с их типами и тип возвращаемого значения.

```
int function(int a, int b, int c) {
    a = b + c;
    return a;
}
```

Учим умные слова

- **Семантика функции** – определяет способ реализации функции. Обычно представляет собой тело функции.

```
int function(int a, int b, int c) {
    a = b + c;
    return a;
}
```

Что происходит во время вызова функции?

- Текущий адрес выполнения кладется в стек
- Переход выполнения на новый адрес
- Выполнения тела функции
- Взятия из стека адреса и возврат к выполнению основного кода.

Возврат из функции

- Любая функция должна завершаться возвращением к месту вызова.
- Возврат происходит с помощью ключевого слова **return**.
- При возврате функция может так же вернуть один параметр в место вызова.

Возврат из функции

```
int first(int b, int c) {
    int a = b + c;
    return a;
}
```

```
void second(int b, int c) {
    int a = b + c;
}
```

```
void second(int b, int c) {
    int a = b + c;
    return;
}
```

```
void second(int b, int c) {
    if (b > c)
        return;
    int a = b + c;
}
```

Типы функций

- **Системные** – хранятся в стандартных библиотеках, и пользователю не нужно вдаваться в подробности их реализации. Достаточно знать лишь их сигнатуру. Примером системных функций, используемых ранее, являются функции printf() и scanf().

- **Собственные** – функции, написанные пользователем для решения конкретной подзадачи.

Зачем?

- Функцию можно вызвать из различных мест программы, что позволяет избежать повторения программного кода.
- Одну и ту же функцию можно использовать в разных программах.
- Функции повышают уровень модульности программы и облегчают ее проектирование.
- Использование функций облегчает чтение и понимание программы и ускоряет поиск и исправление ошибок.

Практика

- Функция определения максимального значения из двух переданных

Расположение функции

- Функция должна быть создана до функции вызывающей ее.

Правильно

```
void test() {
    printf("Hello");
}

int main() {
    test();
}
```

Не правильно

```
int main() {
    test();
}

void test() {
    printf("Hello");
}
```

Определение функции

- Что бы иметь возможность создавать функцию в любом месте, используют определение функции. Для этого нужно написать сигнатуру функции, как правило в начале файла.

```
int main() {
    test();
}

void test() {
    printf("Hello");
}
```

```
void test();

int main() {
    test();
}

void test() {
    printf("Hello");
}
```

Варианты фактических аргументов

- **По значению –** передается только значение аргумента, при изменении этого значения в вызванной функции оригинал не изменится.
- **По ссылке –** в функцию передается ссылка на переменную, при изменении значения по адресу этой ссылке, изменится и оригинал.

Практика

- Функция возведения в степень

Всего лишь набор байт

- Функции можно записывать в переменный
- Функции можно передавать как параметр

Функция как переменная

```
void test() {
    printf("I am test");
}

int main() {

    void (*fun)() = &test;
    fun();
}
```

```
void test() {
    printf("I am test");
}

void executor(void (*executable)()) {
    executable();
}

int main() {

    void (*fun)() = &test;
    executor(fun);
}
```

Практика

- Сделать так что бы предыдущая функция вызывала переданную ей функцию.

Рекурсия

- **Прямая рекурсия** – функция, которая вызывает саму себя.
- **Косвенная рекурсия** – одна или более функций вызывающих друг друга
- **Условие выхода** – условия при выполнении которого рекурсия завершает свою работу. В случае отсутствия такого условия рекурсия не закончится до тех пор пока не переполнится **стек** вашего приложения.

Пример

Прямая рекурсия

```
void recursion(int a) {
    if (a > 0) {
        printf("%d\n", a);
        recursion(--a);
    }
}

int main() {
    recursion(5);
}
```

Косвенная рекурсия

```
void pin(int count) {
    printf("pin(%d) - ", count);
    pong(count);
}

void pong(int count) {
    printf("pong(%d)\n", count);
    if (count > 0)
        pin(--count);
}

int main() {
    pin(5);
}
```

Практика

- Посчитать число Фибоначчи

Числа Фибоначчи

- Последовательность в которой первые два числа равны либо 1 и 1, либо 0 и 1, а каждое последующее число равно сумме двух предыдущих чисел.

- Пример: 1, 1, 2, 3, 5, 8, 13, 21 …