acm.mipt.ru

олимпиады по программированию на Физтехе

Раздел «Язык Си» . CoffeeFunctionPtr :

Функции

Определение функции

Часть программы можно описать отдельно, дать этой части собственное имя и исполнять как отдельную инструкцию.

Таким образом мы получим **функцию** .

Каждая функция имеет:

1. собственное имя,
2. набор аргументов, которые функции необходимо знать для вычислений,
3. значение, которое вычисляет функция в процессе своей работы,
4. набор инструкций, которые выполняет функция

Функции нужны для:

- повторного использования кода;
- разбиения задачи на подзадачи.

Простая функция (параметров нет, ничего не возвращает)

```
#include <stdio.h>

// Описание функции hi
// имя функции hi, параметров нет, ничего не возвращает
void hi() {
    // тело функции - инструкции, которая она выполняет
    printf("Hello\n");
}

// имя функции main, параметров нет, возвращает целое число
int main() {
    // ВЫЗОВ функции hi, с этого места начнутся выполняться инструкции функции
    hi();
    // дальше будут выполняться инструкции main

    return 0;
}
```

При вызове функции hi управление передается из main в hi; выполняются одна за другой инструкции функции hi (тело функции) и после окончания выполнения управление передается обратно в функцию main, сразу за местом вызова функции hi.

Выполнив программу получим на экране

```
Hello
```

Вызовем функцию hi несколько раз. Заметим, что определяем функцию только 1 раз, использовать (вызывать) можем много раз.

```
#include <stdio.h>

void hi() {
    printf("Hello\n");
}

int main() {
    // вызовем функцию несколько раз.
    // Каждый раз управление передается в hi, выполняется тело hi и
    // управление возвращается в место после вызова функции
    hi();
    hi();
    hi();

    return 0;
}
```

При запуске программы получим:

```
Hello
Hello
Hello
```

В программе можно создать несколько функций. **Каждая функция должна иметь уникальное имя.**

Параметры функции

Хотим, чтобы у нас печатался номер группы, которую мы приветствуем.

```
Hello, 878!
Hello, 816!
Hello, 611!
```

Не нужно для этого писать 3 разных функции. Нужно добавить параметр в функцию – номер группы.

```
#include <stdio.h>

// Описание функции hi
// имя функции hi, параметр group типа int, ничего не возвращает
void hi(int group) {
    // тело функции - инструкции, которая она выполняет,
    // можно использовать переменную group
    printf("Hello, %d!\n", group);
}

// имя функции main, параметров нет, возвращает целое число
int main() {
    // ВЫЗОВ функции hi с разными значениями
    // при вызове управление передается в hi
    // значение переменной group - то, что указали при вызове
    hi(878);        // group = 878
    hi(816);        // group = 816
    hi(611);        // group = 611

    return 0;
}
```

group – переменная. Видна (можно использовать) только внутри функции hi. Внутри функции main переменная group не видна.

Можно в разных функциях использовать одинаковые имена переменных. Переменные с именем group создали в функциях hi и bye (параметры). Переменная group есть в функции main (локальная переменная). Всего в программе **три разных** переменных group.

У людей тоже бывают одинаковые имена, например, Леша. Но это разные люди.

```
#include <stdio.h>

// имя функции hi, параметр group типа int, ничего не возвращает
void hi(int group) {
    // можно использовать переменную group - первая переменная с именем group
    printf("Hello, %d!\n", group);
}

// имя функции bye, параметр group типа int, ничего не возвращает
void bye(int group) {
    // можно использовать переменную group - вторая переменная с именем group
    printf("Good bye, %d!\n", group);
}

// имя функции main, параметров нет, возвращает целое число
int main() {
    // объявим переменную group - третья переменная с таким именем в программе
    int group;

    group = 878;
    hi(group);          // Hello, 878!
    bye(group);         // Good bye, 878!

    group = 816;
    hi(group);          // Hello, 816!
    bye(group);         // Good bye, 816!

    group = 611;
    hi(group);          // Hello, 611!
    bye(group);         // Good bye, 611!

    return 0;
}
```

Это разные переменные. Мы можем назвать их по-разному. Например, a, b, gr. Этот код тоже работает.

```
#include <stdio.h>

// имя функции hi, параметр a типа int, ничего не возвращает
void hi(int a) {
    // можно использовать переменную a
    printf("Hello, %d!\n", a);
}

// имя функции bye, параметр b типа int, ничего не возвращает
void bye(int b) {
    // можно использовать переменную b
    printf("Good bye, %d!\n", b);
}

// имя функции main, параметров нет, возвращает целое число
int main() {
    // объявим переменную gr
    int gr;

    gr = 878;
    hi(gr);          // Hello, 878!
    bye(gr);         // Good bye, 878!

    gr = 816;
    hi(gr);          // Hello, 816!
    bye(gr);         // Good bye, 816!

    gr = 611;
    hi(gr);          // Hello, 611!
    bye(gr);         // Good bye, 611!

    return 0;
}
```

Переменные видны (можно использовать) только в той функции, где их определили.

Параметры функции видны (можно использовать) только в этой функции.

Фукнция с несколькими параметрами

Несколько параметров функции пишут через запятую и у **каждого** параметра пишут его тип. Даже если все параметры одинакового типа.

```
void f1(int a, int b, float x) {    // OK
    // тело функции
}
void f2(int a, b, float x) {        // Ошибка - нужно указать тип b
    // тело функции
}
int main() {
    int x = 3, y = 5;
    f1(x, x+y, 3.14);               // Вызов функции f1 с параметрами a=3, b=8, x=3.14
    return 0;
}
```

Перед вызовом функции вычисляются все ее параметры.

Функция возвращает значение

Напишем функцию, которая округляет дробные числа до целых по правилам математики. Т.е 3.2 до 3, а 4.6 до 5.

Вычисленное значение функция должна вернуть с помощью оператора **return**.

Значение вызова функции trunc(3.2) - то, что вернула эта функция.

```
#include <stdio.h>

int truncate(float a) {
    int res;                // объявили локальную переменную res
    res = (int)(a + 0.5);   // вычисления
    return res;             // вернули результат
}

int main() {
    float z;
    int result;

    scanf("%f", &z);
                            // вызов функции:
                            // параметру a присвоим значение переменной z
                            // возвращаемое значение будет присвоено переменной result
    result = truncate(z);
                            // после вызова функции truncate выполняются инструкции main
                            // первая инструкция main ПОСЛЕ вызова функции
                            // присвоение в переменную result значения, которое вернула функция truncate
    printf("Число %f округлили до %d\n", z, result);
```

```
    return 0;
}
```

Аргументы в функцию передаются по значению

То есть при вызове функции создаются переменные (ее аргументы), вычисляется что им было передано при вызове и эти значения присваиваются аргументам.

После окончания вызова функции эти переменные уничтожаются.

Напишем функцию **void inc(int a)**, которая увеличивает значение a на 1.

Заметьте, что значение переменной b функции main не изменилось, потому что в функцию inc передается вычисленное значение переменной b (то есть ее копия).

```
#include <stdio.h>

void inc(int a) {       // при вызове функции создалась переменная a и инициализирована значению при вызове
    a = a + 1;          // значение переменной a увеличилось на 1
}                       // при выходе из функции переменная a уничтожается
int main() {
    int b;
    b = 3;
    inc(b);             // вызов inc(3), в функции inc создалась переменная a=3
    printf("%d\n", b);  // 3, вызов функции inc(b) не изменил значение переменной b, только a.

    return 0;
}
```

Даже если мы в функции main переменную b переименуем в a, результат будет такой же. Потому что у нас будут 2 разные переменные - одна в функции main, другая в функции inc. Теперь - с одинаковыми именами.

Как сделать, чтобы изменения переменной a в функции inc донести до переменной b в функции main? Нужно вернуть новое значение переменной a и потом присвоить результат вызова функции в b.

Для этого изменим тип возвращаемого значения с void на int, допишем в функцию return и присвоим возвращаемое значение в b:

```
#include <stdio.h>

int inc(int a) {       // при вызове функции создалась переменная a и инициализирована значению при вызове
    a = a + 1;          // значение переменной a увеличилось на 1
    return a;           // ВОЗВРАЩАЕМ измененное значение
}                       // при выходе из функции переменная a уничтожается
int main() {
    int b;
    b = 3;
    b = inc(b);         // b=inc(3) записал результат вычислений в b
    printf("%d\n", b);  // 4

    return 0;
}
```

Указатели

swap без функции

Еще один пример - код, который меняет значение переменных местами.

swap функцией - нельзя вернуть 2 значения

Указатели и операции с ними

swap с использованием указателей

```
void swap(int * px, int * py) {
   int z;
   z = *px;   // z = x
   *px = *py;   // x = y
   *py = z;   // y = z
   printf("swap: x=%d y=%d\n", *px, *py);   // 3 2
   printf("swap: %p %p\n", px, py);   // 3 2
}

int main() {
   int x, y;
   x = 2; y = 3;
   printf("read: x=%d y=%d\n", x, y);   // 2 3
```

```
    swap(&x, &y);
    printf("main: x is %d y is %d\n", x, y);   // 3 2
    printf("main: %p %p\n", &x, &y);   // 3 2
    return 0;
}
```

-- TatyanaDerbysheva - 26 Sep 2018