

Содержание

- функции** 3
 - Прототип или описание функции** 3
 - Параметры** 4
 - Передача параметров 4
 - Переменное количество параметров 5
 - Возвращаемые значения** 5
 - Рекурсия** 6
 - Типы функций** 6
 - Тип функции-указатель 6

функции



Процедура - такая ф-я, которая имеет побочный эффект (те затрагивает части программы за пределами себя) и, как правило, ничего не возвращает.

```
// определение функции
возвращаемый_тип имя_функции(параметры) {
    выполняемые_инструкции;
}

// вызов функции
имя_функции(параметры);
```

Прототип или описание функции

Функция должна быть определена до ее вызова. Большинство компиляторов не скомпилирует данный код:

```
#include <stdio.h>

int main(void){
    hello(); // вызов до определения - ошибка
    return 0;
}

void hello(){
    printf("Hello!\n");
}
```

Для определения функции после ее вызова существуют **прототипы** функций:

```
#include <stdio.h>

// прототип
void hello(void);

int main(void){
    hello();
    return 0;
}

// определение
void hello(){
    printf("Hello!\n");
}
```

Таким образом компилятор, на момент вызова ф-ии, уже будет знать о ней.

Параметры

```
// функция принимает массив символов
void print_message(char message[]){
    printf("%s\n", message);
}

// прототип выглядел бы так
void print_message(char[]);
```

если параметров несколько, они перечисляются через запятую

параметры ф-ии можно изменять¹⁾ в теле функции

параметры могут иметь те же модификаторы, что и обычные переменные

Передача параметров

Существует два основных способа передачи параметров в функции: **по значению** и **по указателю**.

При **передаче по значению** значением будет копия аргумента, а функция работает с этой копией. Это означает, что изменения, внесенные в параметры внутри функции, не отразятся на аргументах, переданных в функцию.

```
void modify_value(int value) {
    value = 100; // Это изменение не затронет оригинальный аргумент
}

int main(void) {
    int original = 42;
    modify_value(original);
    printf("Original: %d\n", original); // Вывод: Original: 42
    return 0;
}
```

При **передаче по указателю** передается адрес переменной, что позволяет функции изменять оригинальный аргумент. Этот подход часто используется для работы с массивами, строками и большими структурами данных, чтобы избежать лишнего расхода памяти при копировании.

```
void modify_value(int *value) {
    *value = 100; // Изменяет оригинальный аргумент
}

int main(void) {
    int original = 42;
```

```
modify_value(&original);
printf("Original: %d\n", original); // Вывод: Original: 100
return 0;
}
```

Использование указателей также позволяет передавать массивы в функции, так как имя массива автоматически преобразуется в указатель на его первый элемент:

```
void print_array(int array[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");
}

int main(void) {
    int numbers[] = {1, 2, 3, 4, 5};
    print_array(numbers, 5); // Передается адрес первого элемента массива
    return 0;
}
```

Переменное количество параметров

[stdarg.h](#)

Возвращаемые значения

Функция может иметь результат - некоторое значение, которое оно возвращает во внешний код.

```
void sum1(int x, int y){
    printf("%d + %d = %d \n", x, y, x + y);
}

// аналогично sum1
void sum2(int x, int y){
    printf("%d + %d = %d \n", x, y, x + y);
    return;
}

// возвращает int
int sum3(int x, int y){
    return x + y;
}

int result = sum(x, y);
```

В тех функциях, которые не возвращают никакого значения в качестве возвращаемого типа используется `void`. Если такой ф-ии явно не указать пустой `return;` - компилятор сам подставит его.

При вызове `return` функция завершается, даже если после `return` есть еще код

Рекурсия

функция может вызывать саму себя прямо в теле

```
int factorial(int n) {
    if (n == 1) {
        return 1;
    }
    return n * factorial(n - 1);
}
```

Типы функций

Используя определение функции - ее возвращаемый тип и типы параметров мы можем описать тип функций. Для этого применяется оператор `typedef`:

`typedef возвращаемый_тип (название)(типы_параметров);`

```
#include <stdio.h>

typedef void (message)(void);

void hello() { printf("Hello, World \n"); }
void goodbye() { printf("Good Bye, World \n"); }

int main(void) {
    message* mes1 = &hello; // указатель указывает на функцию hello
    message* mes2 = &goodbye; // указатель указывает на функцию goodbye

    mes1(); // Hello, World
    mes2(); // Good Bye, World

    return 0;
}
```

Тип функции-указатель

Тип функции можно определить как указатель. Например:

```
typedef int (*binary_op)(int, int);
```

Но стоит понимать, что в этом случае `binary_op` уже будет представлять указатель:

```
#include <stdio.h>

typedef int (*binary_op)(int, int);

int sum(int x, int y) { return x + y; }

int main(void) {
    binary_op op1 =  $\Sigma$  // op1 уже изначально представляет указатель
    printf("result = %d \n", op1(10,5)); // result=15

    return 0;
}
```

1)
При компиляции функции для ее параметров выделяются отдельные участки памяти.
Передается только значение

From:
<https://wiki.radi0.cc/> - radi0wiki

Permanent link:
https://wiki.radi0.cc/c:c_ultimate_guide:functions

Last update: **2025/11/09 12:07**

