Параметры по умолчанию

Пример

```
#include <iostream>
using namespace std;

int main() {
    int a = 5, b = 7;
    cout << max_num(a, b);
    return 0;
}
int max_num(int a, int b) {
    if (a > b)
        return a;
    return b;
}
```

Что будет выведено на экран?

Пример

```
#include <iostream>
using namespace std;

int main() {
    int a = 5, b = 7;
    cout << max_num(a, b);
    return 0;
}
int max_num(int a, int b) {
    if (a > b)
        return a;
    return b;
}
```

Что будет выведено на экран?

Возникнет ошибка компиляции.

Пример

```
#include <iostream>
using namespace std;

int main() {
    int a = 5, b = 7;
    cout << max_num(a, b);
    return 0;
}
int max_num(int a, int b) {
    if (a > b)
        return a;
    return b;
}
```

Что будет выведено на экран?

Возникнет ошибка компиляции.

Почему?

Проблема

Компилятор читает код последовательно. В момент, когда в функции `main` мы пытаемся обратиться к функции `max_num`, компилятор ещё не знает о существовании этой функции.

Как можно решить данную проблему?

Меняем порядок функций

```
#include <iostream>
using namespace std;

int max_num(int a, int b) {
    if (a > b)
        return a;
    return b;
}

int main() {
    int a = 5, b = 7;
    cout << max_num(a, b);
    return 0;
}
```

Меняем порядок функций

```
#include <iostream>
using namespace std;

int max_num(int a, int b) {
    if (a > b)
        return a;
    return b;
}

int main() {
    int a = 5, b = 7;
    cout << max_num(a, b);
    return 0;
}
```

Работающий вариант.
Но что делать, если функций в программе много?

Меняем порядок функций

```
#include <iostream>
using namespace std;

int max_num(int a, int b) {
    if (a > b)
        return a;
    return b;
}

int main() {
    int a = 5, b = 7;
    cout << max_num(a, b);
    return 0;
}
```

Работающий вариант.
Но что делать, если функций в программе много?
А что, если функции нельзя расположить в правильном порядке?

Прототипы функций

Ещё одним решением проблемы является **предварительное определение** функций.

Чтобы сообщить компилятору, что та или иная функция существует, используется **прототип функции**.

Прототип функции состоит из типа возвращаемого значения, имени и набора параметров функции (т.е. из заголовка функции) и завершается точкой с запятой.

В краткой форме прототипа можно не указывать имена параметров, а только их типы

Меняем порядок функций

```
#include <iostream>
using namespace std;

int max_num(int a, int b);

int main() {
    int a = 5, b = 7;
    cout << max_num(a, b);
    return 0;
}

void max_num(int a, int b) {
    if (a > b)
        return a;
    return b;
}
```

Заранее объявляем функцию, после чего уже далее в коде определяем реализацию этой функции.

Меняем порядок функций

```
#include <iostream>
using namespace std;

int max_num(int, int);

int main() {
    int a = 5, b = 7;
    cout << max_num(a, b);
    return 0;
}

void max_num(int a, int b) {
    if (a > b)
        return a;
    return b;
}
```

Имена параметров можно не указывать.

Меняем порядок функций

```
#include <iostream>
using namespace std;

int max_num(int, int);

int main() {
    int a = 5, b = 7;
    cout << max_num(a, b);
    return 0;
}

void max_num(int a, int b) {
    if (a > b)
        return a;
    return b;
}
```

Что произойдет, если у прототипа не будет реализации?

Задача

Хотим написать кастомизированный инкремент
с возможностью изменять шаг

```
#include <iostream>
using namespace std;

void inc(int* a, int b) {
    *a += b;
}

int main() {
    int a = 5;
    inc(&a, 1);
    cout << a << endl;
    inc(a, 10);
    cout << a << endl;
    return 0;
}
```

В качестве дополнительного аргумента **b** будем передавать сам шаг, на который нужно увеличить переменную

Задача

Хотим написать кастомизированный инкремент
с возможностью изменять шаг

```
#include <iostream>
using namespace std;

void inc(int* a, int b) {
    *a += b;
}

int main() {
    int a = 5;
    inc(&a, 1);
    cout << a << endl;
    inc(a, 10);
    cout << a << endl;
    return 0;
}
```

В качестве дополнительного аргумента **b** будем передавать сам шаг, на который нужно увеличить переменную

Какое самое частое увеличение числа мы используем?

Можно ли по умолчанию добавлять 1 и не передавать b в функцию?

Можно ли по умолчанию добавлять 1 и не передавать b в функцию?

Можно!

Параметры по умолчанию

Параметры по умолчанию

Параметр по умолчанию - это параметр функции, который имеет определенное (по умолчанию) значение.
Если пользователь **не передает** в функцию значение этого параметра, то используется значение по умолчанию.
Если же **передается** значение, то именно это значение и используется в функции вместо значения по умолчанию.

Пример

```
#include <iostream>
using namespace std;

void inc(int* a, int b = 1) {
    *a += b;
}

int main() {
    int a = 5;
    inc(&a);
    cout << a << endl;
    inc(&a, 10);
    cout << a << endl;
    return 0;
}
```

Значение по умолчанию указывается в заголовке функции.

Пример

```
#include <iostream>
using namespace std;

void inc(int* a, int b = 1) {
    *a += b;
}

int main() {
    int a = 5;
    inc(&a);
    cout << a << endl;
    inc(&a, 10);
    cout << a << endl;
    return 0;
}
```

Значение по умолчанию указывается в заголовке функции.

Все значения по умолчанию указываются после всех обычных параметров.

Пример

```
#include <iostream>
using namespace std;

void fillArray(int a[], int n, int value = 0) {
    for (int i = 0; i < n; i++)
        a[i] = value;
}

int main() {
    int a[100], b[100];
    fillArray(a, 10);
    fillArray(b, 10, 1000);
    return 0;
}
```

Что делает эта функция?

Пример

```
#include <iostream>
using namespace std;

void fillArray(int a[], int n, int value = 0) {
    for (int i = 0; i < n; i++)
        a[i] = value;
}

int main() {
    int a[100], b[100];
    fillArray(a, 10);
    fillArray(b, 10, 1000);
    return 0;
}
```

Что делает эта функция?

Заполняет массив одинаковыми значениями. По умолчанию заполняет массив нулями.

Прототип и параметры по умолчанию

Прототип и параметры по умолчанию

```
#include <iostream>
using namespace std;

void inc(int* a, int b = 1);

int main() {
    int a = 5;
    inc(&a, 2);
    cout << a;
    return 0;
}

void inc(int* a, int b) {
    *a += b;
}
```

Прототип и параметры по умолчанию

```
#include <iostream>
using namespace std;

void inc(int* a, int b = 1);

int main() {
    int a = 5;
    inc(&a, 2);
    cout << a;
    return 0;
}

void inc(int* a, int b) {
    *a += b;
}
```

Значения по умолчанию указываются только в прототипе функции.

Прототип и параметры по умолчанию

```
#include <iostream>
using namespace std;

void inc(int*, int = 1);

int main() {
    int a = 5;
    inc(&a, 2);
    cout << a;
    return 0;
}

void inc(int* a, int b) {
    *a += b;
}
```

Значения по умолчанию можно указывать также без имен параметров

Перегрузки функций

Вспомним функцию, которую разбирали на предыдущем занятии:

Обмен

```
#include <iostream>
using namespace std;

void swap(int* a, int* b) {
    int c = *a;
    *a = *b;
    *b = c;
}

int main() {
    int a = 5, b = 6;
    swap(&a, &b);
    cout << a << " " << b; //Будет выведено 6 5
    return 0;
}
```

Обмен

```
#include <iostream>
using namespace std;

void swap(int* a, int* b) {
    int c = *a;
    *a = *b;
    *b = c;
}

int main() {
    int a = 5, b = 6;
    swap(&a, &b);
    cout << a << " " << b; //Будет выведено 6 5
    return 0;
}
```

Что произойдет, если в эту функцию передать 2 вещественные переменные?

Обмен

```
#include <iostream>
using namespace std;

void swap(int* a, int* b) {
    int c = *a;
    *a = *b;
    *b = c;
}

int main() {
    int a = 5, b = 6;
    swap(&a, &b);
    cout << a << " " << b; //Будет выведено 6 5
    return 0;
}
```

Что нужно сделать, чтобы функция работала как с целыми, так и с вещественными переменными?

Обмен

```
#include <iostream>
using namespace std;

void swapInt(int* a, int* b) {
    int c = *a;
    *a = *b;
    *b = c;
}

void swapDouble(double* a, double* b) {
    double c = *a;
    *a = *b;
    *b = c;
}
```

Обмен

```
#include <iostream>
using namespace std;

void swapInt(int* a, int* b) {
    int c = *a;
    *a = *b;
    *b = c;
}

void swapDouble(double* a, double* b) {
    double c = *a;
    *a = *b;
    *b = c;
}
```

Одна и та же функция, но с разными названиями в зависимости от типа данных. Работает, но выглядит грустно.

Обмен

```
#include <iostream>
using namespace std;

void swap(int* a, int* b) {
    int c = *a;
    *a = *b;
    *b = c;
}

void swap(double* a, double* b) {
    double c = *a;
    *a = *b;
    *b = c;
}
```

Можно определить несколько функций с одним и тем же названием!

Параметры по умолчанию являются частным случаем перегрузок функций.

Чем должны отличаться перегружаемые функции?

Перегрузки

```
#include <iostream>
using namespace std;

void f(int a, int b) {
    // ...
}

void f(int a, double b) {
    // ...
}
```

Будут ли конфликтовать эти функции между собой?

Перегрузки

```
#include <iostream>
using namespace std;

void f(int a, int b) {
    // ...
}

void f(int a, double b) {
    // ...
}
```

Будут ли конфликтовать эти функции между собой?

Нет.

Перегрузки

```
#include <iostream>
using namespace std;

void f(int a, int b) {
    // ...
}

int f(int a, double b) {
    // ...
}
```

Будут ли конфликтовать эти функции между собой?

Перегрузки

```
#include <iostream>
using namespace std;

void f(int a, int b) {
    // ...
}

int f(int a, double b) {
    // ...
}
```

Будут ли конфликтовать эти функции между собой?

Нет.

Перегрузки

```
#include <iostream>
using namespace std;

void f(int a, int b) {
    // ...
}

int f(int a, double b) {
    // ...
}
```

Будут ли конфликтовать эти функции между собой?

Перегрузки

```
#include <iostream>
using namespace std;

void f(int a, int b) {
    // ...
}

int f(int a, double b) {
    // ...
}
```

Будут ли конфликтовать эти функции между собой?

Нет.

Перегрузки

```
#include <iostream>
using namespace std;

void f(int a, int b) {
    // ...
}

int f(int a, int b) {
    // ...
}
```

Будут ли конфликтовать эти функции между собой?

Перегрузки

```
#include <iostream>
using namespace std;

void f(int a, int b) {
    // ...
}

int f(int a, int b) {
    // ...
}
```

Будут ли конфликтовать эти функции между собой?

Да.

Перегрузки функций

Для корректной перегрузки функции необходимо, чтобы **отличался набор параметров** функции - отличаться могут как типы данных, так и количество параметров.
Тип возвращаемого значения **не влияет** на возможность перегрузить функцию.

Перегрузки

```
#include <iostream>
using namespace std;

void f(int a, int b, int c = 0) {
    // ...
}

void f(int a, int b, double c = 1.0) {
    // ...
}
```

Будут ли конфликтовать эти функции между собой?

Перегрузки

```
#include <iostream>
using namespace std;

void f(int a, int b, int c = 0) {
    // ...
}

void f(int a, int b, double c = 1.0) {
    // ...
}
```

Будут ли конфликтовать эти функции между собой?

И да, и нет

Перегрузки

```
#include <iostream>
using namespace std;

void f(int a, int b, int c = 0) {
    // ...
}

void f(int a, int b, double c = 1.0) {
    // ...
}
```

Если не вызывать функцию только с 2 параметрами, то ошибки компиляции не будет.

Но если запустить функцию от двух параметров, то компилятор не сможет определить, какую версию функции использовать.