Функции в C++

Лекция #2

Пустовалова О.Г.
доцент. каф. мат.мод.
ИММиКН ЮФУ

Содержание

- Библиотека cmath
- Функции и процедуры
- Перегрузка функций
- Многофайловая компоновка
- Директивы препроцессора

Библиотека cmath

Функция	Описание
abs(a)	модуль или абсолютное значение от a
sqrt(a)	корень квадратный из a, причём a не отрицательно
pow(a, b)	возведение a в степень b
ceil(a)	округление a до наименьшего целого, но не меньше чем a
floor(a)	округление a до наибольшего целого, но не больше чем a
fmod(a, b)	вычисление остатка от a/b
exp(a)	вычисление экспоненты e^a
sin(a)	a задаётся в радианах
cos(a)	a задаётся в радианах
log(a)	натуральный логарифм a(основанием является экспонента)
log10(a)	десятичный логарифм a
asin(a)	арксинус a, где -1.0 < a < 1.0

http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/math.h.html - справка по математическим функциям

Функции и процедуры

```
#include <iostream>

// Функция, возвращающая void – процедура
void printOnly() {
      std::cout <<"Hello!"<<std::endl;
}

int main(){
      printOnly();
return 0;
}
```

В C++ процедур не бывает!

Функции и процедуры

```
#include <iostream>

// Функция, возвращающая void – процедура
void printOnly(double a, int k) {
	std::cout << k*a << std::endl;
}

int main() {
	printOnly(123, 1);
return 0;
}
```

В C++ процедур не бывает!

Функции и процедуры

```
#include <iostream>
int absoluteValue(int x) {
      return x > 0 ? x : -x;
}

int main(){
      std::cout<<absoluteValue(-5)<<std::endl;
return 0;
}
```

Функции и процедуры

```
#include <iostream>
auto absoluteValue(int x) {
return x > 0 ? x : -x;
}

int main() {
      std::cout << absoluteValue(-5) << std::endl;
return 0;
}
```

Перегрузка функций

```
#include <iostream>
void printOnly() {
	std::cout << "Hello!" << std::endl;
}
void printOnly(double a, int k) {
	std::cout << k*a << std::endl;
}


int main() {
	printOnly();
	printOnly(2,3);
return 0;
}
```

Когда не работает перегрузка функций

```
#include <iostream>
double myF(int a, int b) {
return a*b ;
}
int myF(int a, int k) {
return k*a ;
}

int main() {
std::cout << myF(5,1) << std::endl;
std::cout << myF(3, 1) << std::endl;
return 0;
}
```

Функции с параметрами по умолчанию

```
#include <iostream>
using namespace std;
double total(double salary, double tax=100.0)
{
	return salary-tax;
}

int main()
{
	setlocale(0, "");
	cout <<"Зарплата сегодня "<<total(1000) << endl;
	cout <<"Зарплата завтра  "<<total(1000,50) << endl;
	return 0;
}
```

Где можно размещать функции

```
#include <iostream>
using namespace std;
int sumABC(int a, int b = 10, int c = 100)
{
	return a + b + c;
}

int main()
{
setlocale(0, "");
cout << sumABC(1) << endl;
cout << sumABC(1, 2, 4) << endl;
return 0;
}
```

Где можно размещать функции. Прототип функции

```
#include <iostream>
using namespace std;
// прототип функции
int sumABC(int a, int b=10, int c=100);

int main()
{
setlocale(0, "");
cout << sumABC(1) << endl;
cout << sumABC(1, 2, 4) << endl;
return 0;
}
int sumABC(int a, int b , int c)
{
return a + b + c;
}
```

Где можно размещать функции. Прототип функции

```
#include <iostream>
using namespace std;
// если нет параметров по умолчанию
// прототип функции
int sumABC(int , int , int );

int main()
{
setlocale(0, "");
cout << sumABC(1, 2, 4) << endl;
return 0;
}
int sumABC(int a, int b , int c)
{
return a + b + c;
}
```

Инлайн функции

В качестве inline-функции обычно используются очень маленькие функции. Большие функции реализуются обычным способом.

Всякий раз когда вызывается inline-функция, компилятор будет заменять вызов функции фактическим кодом из функции.
Этот подход позволяет сокращать время выполнения программы.

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

inline void hello()
{
        cout << "Hello!";
}
int main(){
        hello();
}
```

```
/*

как только программа будет
скомпилирована,
вызов hello();
будет заменен на код функции

*/
```

Способы передачи параметров в функцию

- По значению (by value)
- По ссылке & (by reference)
- По указателю * (by pointer)

Передача параметров в функцию по значению

Передача параметров в функцию по значению

```
double sum3(double a, double b, int c)
{
        return a + b + c;
}.


 int main()
 {
 double a = 3; double b = 2;

// На этапе выполнения работа происходит с копиями переменных

 cout << sum3(a, b, 5) << endl;

 return 0;
 }
```

Передача параметров в функцию по значению

Фактический параметр → **Формальный параметр**

В ячейку памяти формального параметра
Передается **копия** фактического и обратно не возвращается

Передача параметров в функцию по значению

```
double sum3(double a, double b, int c)
{ return a + b + c;}.
```

Недостатки

1. Большие затраты времени на копирование значений.
2. Затраты памяти для хранения копии.

Накладно для больших массивов!

3. Невозможно из функций изменять значения аргументов!

.

Передача параметров в функцию по значению

```
#include <iostream>
using namespace std;
int copyParam(int a, int b)
{
	a = 100 * a + b;
	return a;
}
int main()
{	int a = 1;
	cout << copyParam(a,7) <<endl;//107

	cout <<"a =  "<< a << endl;//1
	return 0;
}
```

Невозможно из функций изменять значения аргументов!

Передача параметров в функцию по значению

Передача по значению

В главной программе	В функции копия	После действия функции в главной программе
a=1	После изменения *a=107*	*a=1*

Невозможно из функций изменять значения аргументов!

Передача параметров в функцию по ссылке &

Оператор взятия адреса &

```
int v = 12;
cout << "  v = " << v << endl;
cout << "  &v " << &v << endl;
```

Результат:

```
v = 12
&v = 0079F9DC
```

Оператор взятия адреса &

```
int v = 12;
int &r = v;
cout << "  &v " << &v << endl;
cout << "  &r " << &r << endl;
```

Результат:

```
&v = 008FFED8
&r = 008FFED8
```

оба имени **v** и **r** привязаны к одному и тому же адресу

Что такое ссылка (reference)

- При объявлении ссылки перед её именем ставится символ амперсанда **&**, сама же ссылка должна быть проинициализирована именем переменной, на которую она ссылается.
- Любое изменение значения содержащегося в ссылке повлечёт за собой изменение этого значения в переменной, на которую ссылается ссылка.

```
int v = 12;

int &r = v; // объявление и инициализация ссылки
cout << "  v = " << v << "  r = " << r << endl;

//изменяем значение переменной посредством изменения
//значения в ссылке
r += 12;

cout << "  v = " << v << "  r = " << r << endl;
```

Что такое ссылка (reference)

- объявляя ссылку мы задаём альтернативный идентификатор (псевдоним) для уже созданного объекта.

```
int v = 12;
int &r = v;

v = 1024;

// v = 1024  r = 1024
cout << "  v = " << v << "  r = " << r << endl;
```

- ссылка — это, по сути, указатель, который жёстко привязан к области памяти, на которую он указывает, и который автоматически разыменовывается, когда мы обращаемся по имени ссылки

Что такое ссылка (reference)

- Ссылки нельзя объявлять без привязки к переменной (то есть не инициализировав при объявлении)

```
int v = 12;
int &r;
```

Нельзя объявлять и не инициализировать!

для ссылки переменная "r" требуется инициализатор

Нужно объявлять и ИНИЦИАЛИЗИРОВАТЬ!

```
int v = 12;
int &r=v;
```

Передача параметров в функцию по ссылке &

- В случае вызова по ссылке оператор вызова дает вызываемой функции возможность прямого доступа к передаваемым данным, а также возможность изменения этих данных.

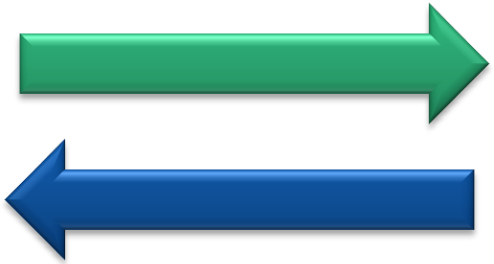
- Вызов по ссылке хорош в смысле производительности, потому что он исключает накладные расходы на копирование больших объемов данных.

```
double sum3(double &a, double &b, int &c) {
        return a + b + c;}.

int main()
{
double a = 3; double b = 2;
cout << sum3(a, b, 5) << endl;
return 0;
}
```

Передача параметров в функцию по ссылке &

Фактический параметр



Формальный параметр

Передается **адрес** фактического параметра.
По имеющемуся адресу изменяется содержимое ячеек памяти фактического параметра, происходит возврат результата

Передача параметров в функцию по ссылке &

```
#include <iostream>
using namespace std;
int myF(int &a, int b)
{
        a = 100 * a + b;
return a;
}

int main()
{

int a = 1;
int b = 7;
cout << myF(a, b) << endl;//107
cout << "a =  " << a << endl;//107
return 0;
}
```

Передача параметров в функцию по ссылке &

Передача по ссылке &

В главной программе *a=1*

В функции ссылка на *a=1*
После изменения *a=107*

После действия функции.
В главной программе *a=107*

Передача параметров в функцию по ссылке &

Передача параметра по ссылке также используется, если функция должна возвратить не один результат, а **несколько**.

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

void squareSP(double &a, double &s, double &p) {
        s=a*a;
        p = 4 * a;
}
void main() {
        double a=16;
        double s, p;
        squareSP(a,s,p);
        cout << "s =  " << s << endl;
        cout << "p =  " << p << endl;
}
```

Требуется найти площадь и периметр квадрата

Передача параметров в функцию по ссылке &

Передача параметра по ссылке также используется, если функция должна возвратить не один результат, а **несколько**.

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

void power123(double& a, double& a2, double& a3)
{
a2=a*a;
a3 = a2*a;
}

void main()
{
double a = 4,a2,a3;
power123(a, a2, a3);
cout <<a<<" "<< a2<<" "<< a3;
}
```

Требуется найти 2-ю и 3-ю степени числа

Передача параметров в функцию по ссылке &

При передаче по ссылке изменение параметра в функции ведет к изменению в главной программе.

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

double power2(double &a)
{
        return a*a;
}

void main()
{
double a=16;
cout << " Now a is equal = " << power2(a);//256
}
```

Требуется изменить число так, что

$$a \rightarrow a^2$$

Где ставить знак &

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


//double power2(double &a)
double power2(double& a)
//double power2(double & a)
{
return a*a;
}

void main()
{
double a = 16;
cout << " Now a is equal = " << power2(a);//256
}
```

в функцию передается
не сама переменная,
а ее адрес, полученный
операцией адресации **&**

Запрещение изменения параметра при передаче по ссылке

Если требуется запретить изменение параметра, передаваемого по ссылке, то нужно использовать модификатор **const**

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

double power2(const double &a, double &a2)
{
        a2 = a*a;
        return a2;
}

void main() {
        double a=16, a2;
        cout << " Now a2 is equal = " << power2(a,a2);
}
```

Требуется изменить число так, что

$$a \rightarrow a^2$$

Запрещение изменения параметра при передаче по ссылке

Попытка изменить параметр с модификатором **const** приведет к ошибке компиляции

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

double power2(const double &a, double &a2)
{
        a++; // ОШИБКА!
        a2 = a*a;
        return a2;
}

void main() {
        double a=16, a2;
        cout << " Now a2 is equal = " << power2(a,a2);
}
```

невозможно присваивать значения переменной, которая объявлена как константа

Передача параметров в функцию по ссылке &. Выводы

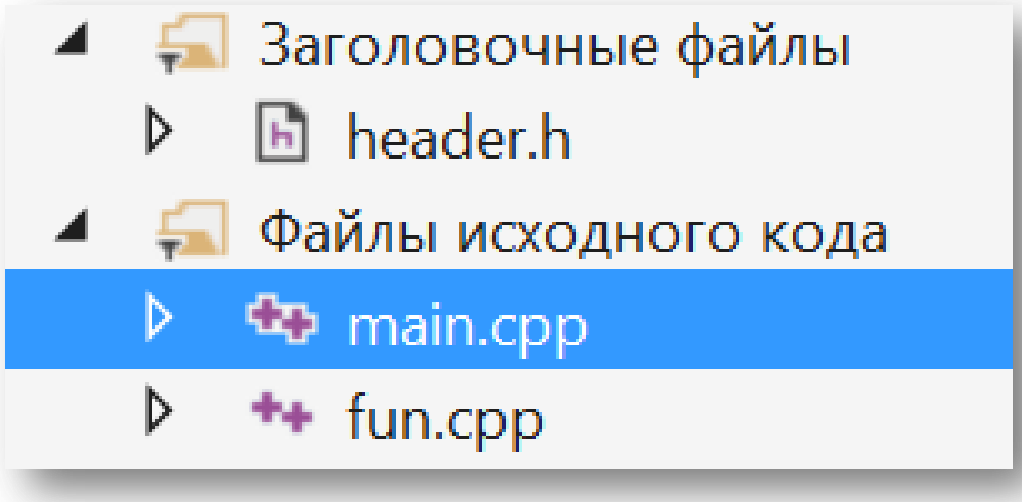
1. Экономия памяти и времени
2. Можно изменять параметры внутри функции
3. Можно возвращать несколько значений из функции
4. Можно устанавливать запрет на изменение параметров с помощью const.

Многофайловая компоновка

Многофайловая компоновка

Многофайловая компоновка

Обозреватель решений

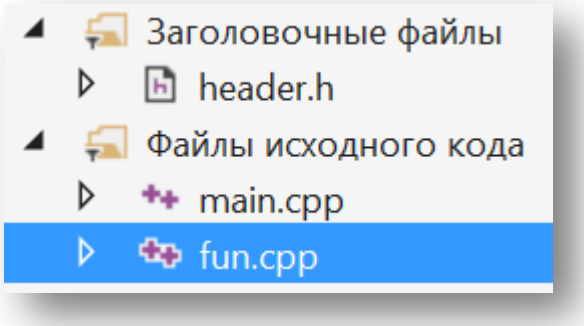


Многофайловая компоновка

Файл - fun.cpp располагать в файлах исходного кода

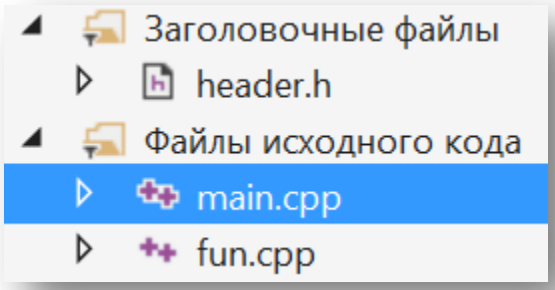
```
int myfun(int a, int b)
{
      return a + b;
}

// и все остальные функции
```



Многофайловая компоновка

Файл - main.cpp располагать в файлах исходного кода



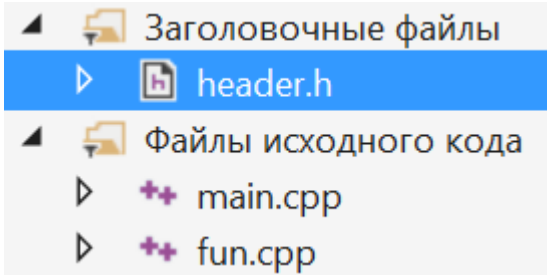

```
#include <iostream>
// файл заголовка
#include "header.h"
using namespace std;

int main()
{
	cout << myfun(2, 3) << endl;
	// и все остальные вызовы функций
	return 0;
}
```

Многофайловая компоновка

Файл - header.h располагать в заголовочных файлах

```
#pragma once
int myfun(int a, int b);
```




```
// и все остальные прототипы  функций
```

Многофайловая компоновка. Пример

Файл **header.h** содержит прототип функции с параметрами по умолчанию:

```
#pragma once
int myfun(int a, int b=10, int c=20);
```

Содержание файла **fun.cpp:**

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

Что может содержаться в заголовочном файле

1. Определения типов
2. Шаблоны типов
3. Прототипы функций
4. Определения инлайн функции
5. Описания данных
6. Определения констант
7. Перечисления
8. Описания имен
9. Команды включения файлов
10. Макроопределения
11. Комментарии

```
struct point { int x, y; };
template<class T> class V { /* ... */ }
extern int strlen(const char*);
inline char get() { return *p++; }
extern int a;
const float pi = 3.141593;
enum bool{ false, true };
class Matrix;
#include <signal.h>
#define Case break;case
/* проверка на конец файла */
```

Что нельзя размещать в заголовочном файле

1. Определений обычных функций
2. Определений данных
3. Определений составных констант

```
char get() { return *p++; }
```

```
int a;
```

```
const tb[i] = { /* ... */ };
```

Некоторые директивы препроцессора

Директивы препроцессора

- Директивы препроцессора начинаются со знака "диез" (#), который должен быть самым первым символом строки.

- Программа, которая обрабатывает эти директивы, называется ***препроцессором***

- В современных компиляторах препроцессор обычно является частью самого компилятора.

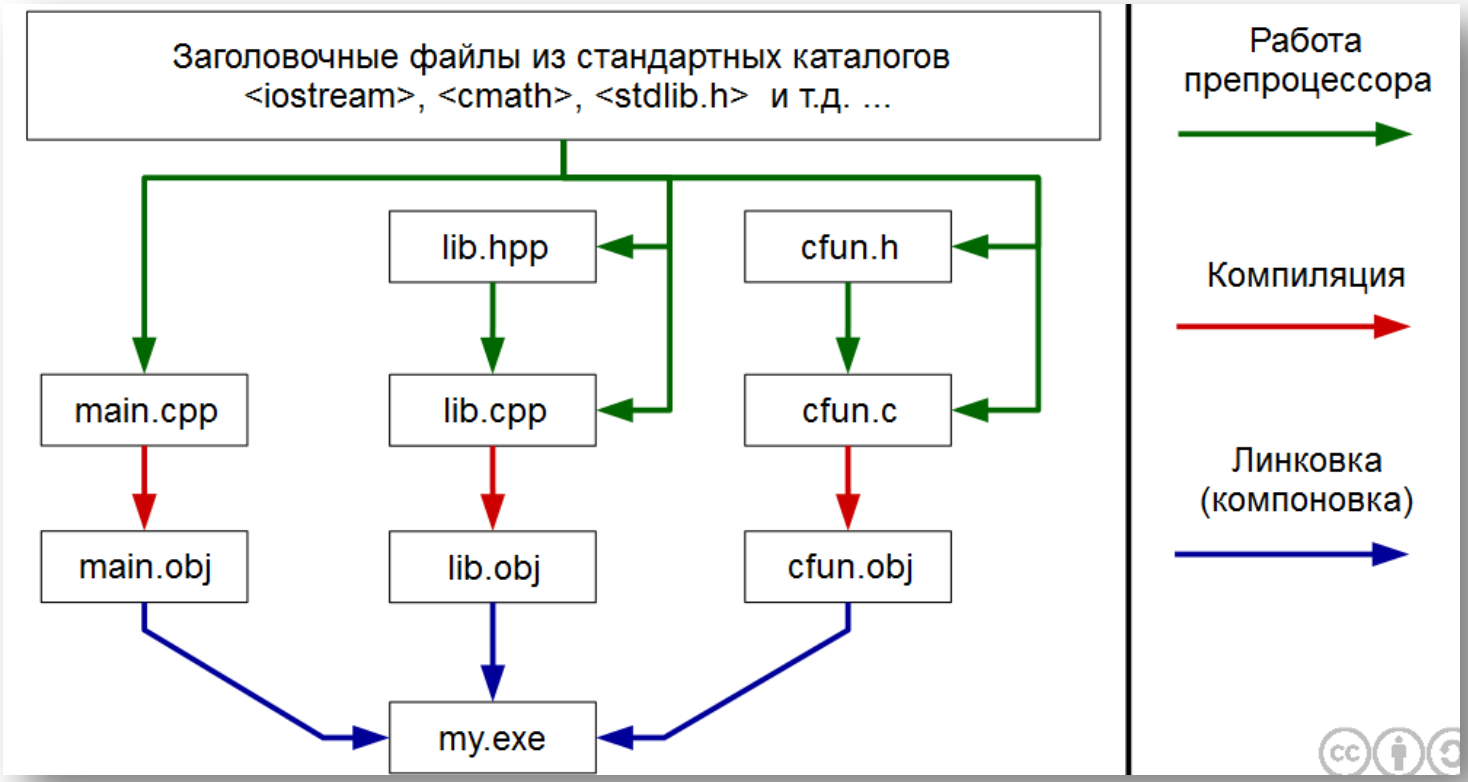
Компилятор

- **Компиля́тор** — программа или техническое средство, выполняющее *компиляцию*
- **Компиля́ция** — трансляция программы, составленной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду


Компилятор
Исходная программа
Синтаксический контроль текста программы
Генератор машинного кода
Сообщения об ошибках
Исполняемая программа

Компиляция в C++

- В C++ имеет место *независимая* компиляция: все файлы проекта компилируются независимо один от другого.
- Компиляция состоит из этапа собственно компиляции и этапа **линковки**.



Основные этапы сборки проекта в C++

1. **Препроцессорная обработка**
2. **Компиляция** каждого *.cpp-файла в объектный код (файлы *.obj или *.o).
3. **Линковка** сборка всех объектных файлов в один исполняемый (*.exe или ELF).

Замечание. inline-функции «погибают» при компиляции

Ошибки во время линковки

- Одинаковые объявления в одном пространстве имен.
- Ошибки при использовании #include "*.cpp" (грубая ошибка).
- Отсутствие описания функции.
- Отсутствие main() во всех файлах проектах.
- Несколько объявлений main().

Директивы препроцессора. #include

Директива **#include** включает в программу содержимое указанного файла. Имя файла может быть указано двумя способами:

```
// стандартный заголовочный файл
#include <iostream>

// пользовательский заголовочный файл
#include "header.h"
```

Директивы препроцессора. #pragma once

#pragma once

Совместное использование директивы **#pragma** с лексемой **once** просит компилятор включить файл заголовка только один раз, независимо от того, сколько раз она был импортирован.

Директивы препроцессора. #pragma once

```
#ifndef HEADER_H
#define HEADER_H

/* содержимое файла header.h */

#endif
```

Содержательная часть заголовочного файла **header.h** будет включена в исходный текст только один раз, сколько бы раз ни включался в текст сам этот файл.

Директивы препроцессора. #define

Директива **#define** используется для определения

- **Констант**
- **Макросов**

```
#include <iostream>

// определение константы
#define WORD "Presentation"

int main()
{
	std::cout << WORD;
	return 0;
}
```

Директивы препроцессора. #define

Директива **#define** используется для определения

- **Констант**
- **Макросов**

```
#include <iostream>

// определение макроса
#define MAX(a, b) ((a) > (b) ? (a) : (b))

int main()
{
	std::cout << MAX(2,3);
	return 0;
}
```

Директивы препроцессора. #if

Директива **#if** проверяет, является ли значение value истиной и, если это так, то выполняется код, который стоит до закрывающей директивы **#endif**.

В противном случае, код внутри **#if** не будет компилироваться, он будет удален компилятором, но это не влияет на исходный код в исходнике.

```
#if value
        // код, который выполнится, в случае, если value - истина
#elsif value1
        // этот код выполнится, в случае, если value1 - истина
#else
        // код, который выполнится в противном случае
#endif
```

Спасибо за внимание!