

C++

Обзор. Стандартная библиотека. Нововведения стандартов C++11 и C++14 Черновик

Кафедра ИВТ и ПМ

2021

План

Основы C++

Типы данных

- Указатели и ссылки

- Массивы

- Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

Определение типа

Ссылки на правосторонние значения

Лямбда-функции

Создание программ с GUI

Outline

Основы C++

Типы данных

- Указатели и ссылки

- Массивы

- Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

Определение типа

Ссылки на правосторонние значения

Лямбда-функции









Создание программ с GUI

*Си позволяет легко
выстрелить себе в ногу; с
С++ это сделать сложнее,
но, когда вы это делаете,
вы отстреливаете себе
ногу целиком.*

*Ограничение возможностей
языка с целью
предотвращения
программистских ошибок в
лучшем случае опасно.*

Б. Страуструп^а

^аСоздатель языка С++

Aug 2021	Aug 2020	Change	Programming Language		Ratings	Change
1	1			C	12.57%	-4.41%
2	3	▲		Python	11.86%	+2.17%
3	2	▼		Java	10.43%	-4.00%
4	4			C++	7.36%	+0.52%
5	5			C#	5.14%	+0.46%
6	6			Visual Basic	4.67%	+0.01%
7	7			JavaScript	2.95%	+0.07%
8	9	▲		PHP	2.19%	-0.05%
9	14	▲▲		Assembly language	2.03%	+0.99%
10	10			SQL	1.47%	+0.02%

О языке

- ▶ Общего назначения
- ▶ Компилируемый
- ▶ Статическая типизация
- ▶ Объектно-ориентированный¹
- ▶ Без сборщика мусора

¹поддерживаются и другие парадигмы программирования

О языке

- ▶ Большая часть кода Microsoft Windows написана на C++
- ▶ Часть кода Apple OS X – C++ код
- ▶ Adobe Photoshop
- ▶ MySQL Server
- ▶ Autodesk Maya
- ▶ Mozilla Firefox
- ▶ Многие библиотеки для Python
- ▶ ...

IDE

- ▶ **Qt Creator**

Кроссплатформенный, лаконичный, свободный, устанавливается вместе с фреймворком [Qt](#)²

- ▶ **Visual Studio**

- ▶ [youtube.com/watch?v=kpgG9imQ2fU](https://www.youtube.com/watch?v=kpgG9imQ2fU) - Самые вкусные возможности Visual Studio 2019
- ▶ [youtube.com/watch?v=JhxC-K-Eehg](https://www.youtube.com/watch?v=JhxC-K-Eehg)

- ▶ **JetBrains CLion**

Кроссплатформенный, есть версия для студентов, нет бесплатной версии

- ▶ [youtube.com/watch?v=Srnw1dl1iAA](https://www.youtube.com/watch?v=Srnw1dl1iAA)

- ▶ jupyter.org/try - C++ (компилятор Clang) в Jupyter. подходит для экспериментов: работает как интерпретатор

²путь к папке с установкой должен содержать только латиницу (без пробелов)

Стандартная библиотека C++ содержит многое, что необходимо для хранения и обработки данных (динамический массив, список, и т.д.), для работы с файлами, сетью, потоками и др. Модули для создания приложений с GUI в состав библиотеки не входят.

В отличие от Python вместе с компилятором C++ не поставляется средств для автоматической установки дополнительных библиотек. Библиотеки необходимо скачивать вручную, компилировать (при необходимости) и устанавливать в систему или размещать в каталогах проекта



Набор библиотек **boost** поставляется отдельно и представляет больший набор возможностей чем стандартная библиотека. Boost содержит в том числе математические модули, например посвященные линейной алгебре, работе с графами и для статистической обработки данных.

Структура программы

Далее рассматривается шаблон простого приложения на C++.

Эти шаблоны могут немного отличаться в зависимости от используемой среды программирования и типа проекта, который создаётся.

Приведённый на следующем слайде шаблон был создан в Qt Creator: создать проект ... > проект без Qt > приложение на языке C++

Структура программы

```
// подключение модулей. Имя модуля (заголовочного файла) в угловых скобках
// он в известных компилятору местах (например модуль стандартной библиотечки)

#include <iostream> // модуль для ввода\вывода (в консоль)

// подключение заголовочного файла расположенного в том же каталоге
// где и основной файл исходных кодов. вместо угловых скобок - кавычки
#include "my_file.h"

// стандартная библиотека содержится в пространстве имён std
// чтобы каждый раз не использовать std:: при обращении к содержимому
// этой библиотеки сделаем содержимое std доступным непосредственно
using namespace std;

// переменные, константы, типы и функции можно объявлять здесь

// основная программа:
int main(int argc, char* argv[])
//допускается и такой заголовок: int main()
{
    // здесь тоже можно объявлять переменные, константы и типы
    cout << "Hello< World!" << endl;
    return 0;
}
```

Структура программы

Пояснения

- ▶ `#include` - директива компилятора помещающая содержимое указанного файла исходных кодов в текущий файл
- ▶ `main` - функция вызываемая при запуске программы
`int main(int argc, char* argv[])`
 - ▶ `int` - возвращаемый функцией тип данных
 - ▶ для каждого параметра функции тоже указывается тип данных
 - ▶ `argc` - число аргументов командной строки
 - ▶ `char* argv[]` - массив из аргументов (первый аргумент - полное имя исполняемого файла)
- ▶ `{ }` - операторные скобки³
- ▶ `return 0.` по договорённости программа должна вернуть 0 если она завершилась без сбоев. Этот код возврата может использоваться другими программами, которые вызывают данную.

Далее на слайдах не будет приводится заголовок функции `main` и иногда подключение модулей для сокращения кода

³объединяют несколько операторов в блок команд. В Python для этих же целей служат отступы

Прошлые темы

- ▶ Что такое литерал?
- ▶ Что такое идентификатор?
- ▶ Что такое объявление?
- ▶ Что такое определение?
- ▶ Что такое тип данных?

Outline

Основы C++

Типы данных

- Указатели и ссылки

- Массивы

- Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

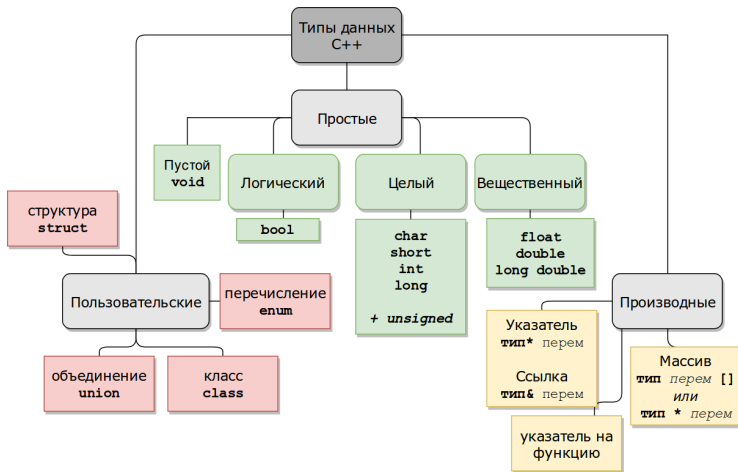
- Определение типа

- Ссылки на правосторонние значения

- Лямбда-функции

Создание программ с GUI

Типы данных



Подробнее о простых типах: ru.cppreference.com/w/cpp/language/types

Типы данных

Data Type	Size (bytes)	Size (bits)	Value Range
unsigned char	1	8	0 to 255
signed char	1	8	-128 to 127
char	1	8	either
unsigned short	2	16	0 to 65,535
short	2	16	-32,768 to 32,767
unsigned int	4	32	0 to 4,294,967,295
int	4	32	-2,147,483,648 to 2,147,483,647
unsigned long	8	64	0 to 18,446,744,073,709,551,616
long	8	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long long	8	64	0 to 18,446,744,073,709,551,616
long long	8	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4	32	3.4E +/- 38 (7 digits)
double	8	64	1.7E +/- 308 (15 digits)
long double	8	64	1.7E +/- 308 (15 digits)
bool	1	8	false or true

Для определения размера переменной или типа используется оператор `sizeof`. Он возвращает размер в байтах. Например `sizeof(double)`

<https://en.cppreference.com/w/cpp/language/sizeof>

Объявление переменных и констант

- ▶ В C++ нет специального раздела программы для определения или объявления переменных
- ▶ При объявлении:
 - ▶ сначала указывается тип данных ⁴
 - ▶ потом идентификатор переменной
 - ▶ наконец присваивается значение (если необходимо)

```
int n; // можно не задавать значение
float x = -47.039; // можно задавать...
float y, z;
const short N = 24; // константе задавать значение обязательно
char str1[] = "qwerty"; // строка как массив символов
// string - тоже строка, но лучше.
string str2 = "qwerty"; // нужно подключить модуль string
n = N;
N = n; // Ошибка! Константу поменять нельзя
a = 42; // Ошибка! Переменная a не объявлена
```

⁴ В отличие от Python переменная не может менять свой тип данных

Вывод данных

Пример

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

cout << "Hello, World!";

// endl - вывод символа конца строки и очистка буфера вывода
cout << "Hello, World!" << endl;

// Вывод переменной
float x;
cout << x << endl;

// Вывод данных нужно подписывать
cout << "x = " << x << endl;
```

Вывод данных

- ▶ `cout << "qwerty" << 3.14 << 42;`

На экране появится: `qwerty3.1442`

- ▶ `cout` - объект предназначенный для вывода на стандартный вывод
- ▶ `<<` - оператор вывода данных данных.
Левый операнд - объект `cout`;
Правый операнд - выводимые данные.
- ▶ `cout` объявлен в заголовочном файле `iostream`, пространстве имён `std`;

Подробнее: mycpp.ru/cpp/book/c20.html

Вывод данных

```
#include <iostream>          // std::cout, std::fixed
#include <iomanip>             // std::setprecision

...
// Установка формата вывода:
// (без использования экспоненциальной формы)
// установка 2 знаков после запятой
cout << fixed << setprecision(2);

// Вывод строки и переменной одновременно
cout << "X = " << x << endl;
```

Ввод данных

`cin` - объект предназначенный для чтения данных с клавиатуры.

`>>` - оператор чтения данных с клавиатуры.

Левый операнд - объект `cin`;

Правый операнд - переменная.

```
cin >> x;
```

`cin` объявлен в заголовочном файле `iostream`, пространстве имён `std`;

Ввод данных

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

float x;
cout << "Введите число ";
cin >> x;
```

Типы данных

Производные типы

- ▶ Указатель (pointer)
- ▶ Ссылка
- ▶ Массив⁵
- ▶ Структура
- ▶ Класс
- ▶ Перечисление

⁵рекомендуется использовать классы стандартной библиотеки вместо массивов

Типы данных

Указатели и ссылки

Указатель (pointer) – переменная, диапазон значений которой состоит из адресов ячеек памяти или специального значения — нулевого адреса.

При объявлении указателя после типа данных, на который он должен указывать, ставится *

```
// объявление указателя на тип int  
int * ip;
```

```
// объявление указателя на тип float  
// здесь сразу в записывается адрес  
// nullptr - это пустой указатель,  
// таким образом указатель fp в данный момент  
// ни на что не указывает  
float *fp = nullptr;
```

Типы данных

Указатели и ссылки

Основные операции используемые при работе с указателями

- ▶ взятие адреса. оператор `&`
используется при записи адреса переменной в указатель
- ▶ разыменование. оператор `*`
доступ к значению, адрес которого записан в указателе

```
// объявление указателя на тип int
```

```
int * ip;
```

```
int i = 42;
```

```
// в указатель можно записать адрес переменной  
// для этого используется оператор взятия адреса &  
ip = &i;
```

```
// теперь можно обращаться к переменной i через указатель  
// чтобы обратиться не к адресу, который записан в указателе  
// а к значению, на которое он указывает нужно использовать  
// оператор разыменования *
```

```
*ip = 8; // переменная i теперь содержит 8
```

Типы данных

Указатели и ссылки

```
// объявление указателя на тип int
int * ip;

int i = 42;

ip = &i;

*ip = 8; // переменная i теперь содержит 8
int *ip2;

// конечно можно записывать в один указатель другой
// если типы данных, на которые они ссылаются совпадают
ip2 = ip;
// *ip2 = 8
// *ip = 8
// i = 8

*ip2 = 100;
// *ip = 100
// i = 100
```

Типы данных

Указатели и ссылки

Ссылки похожи на указатели, только с разницей

- ▶ Ссылка не может менять своё значение
- ▶ Следовательно при объявлении ссылки она обязательно инициализируется
- ▶ При обращении к значению по ссылке оператор `*` не требуется

Про ссылку можно думать как про другое имя для объекта

```
int i = 42;  
// при объявлении ссылки используется &  
// здесь не стоит путать с оператором взятия адреса,  
// хотя для их обозначения используется один и тот же символ  
int &il = i;  
  
// оператор разыменования не требуется  
int n = il;  
il = 100;  
// i = 100; n = 42
```

Типы данных

Массивы

```
// массив из 128 целых чисел
```

```
int a[128];
```

```
// обращение к элементу по его индексу
```

```
a[0] = 42; // нумерация с нуля
```

```
// рекомендуется хранить размер массива в переменной или
```

```
unsigned const n = 128;
```

```
float b[n];
```

```
n[n-1] = 36.6; // последний элемент массива
```

Пример заполнения массива в цикле приведён на слайде 38

Типы данных

Динамические массивы

```
const unsigned n = 128;  
// для работы с динамическими массивами используются указатели  
// запишем в указатель адрес памяти выделенной для 128 элементов  
// оператор new выделяет память в куче  
int *a = new int[n];  
  
// обращение к элементам такое же как и для статического массива  
a[0] = 42;  
int x = a[2];  
  
// после окончания работы  
// нужно освободить память, которую он занимает  
delete[] a;
```

В большинстве случаев использование класса `vector` из стандартной библиотеки (см. C++. part2.pdf) предпочтительнее использования динамических массивов. `vector` предоставляет удобный для программиста способ работы с динамическими массивами.

Типы данных

Перечисления

Типы данных

Составные типы данных

Для представления составных типов данных в C++ используются

- ▶ Структуры `struct`
- ▶ Объединения⁶ `union`
- ▶ Классы `class`

⁶Объединение позволяет хранить один набор данных, но обращаться к нему как к различным типам

Типы данных

struct

```
// Определение нового типа данных
```

```
struct Point{  
    float x, y;  
};
```

```
Point p; // объявление переменной типа Point
```

```
// обращение к полям
```

```
p.x = 10;  
float a = p.y;
```

```
// можно задавать значения полей при объявлении
```

```
Point p1 = {10, 2};  
a = p1.x; // a = 10
```

Outline

Основы C++

Типы данных

Указатели и ссылки

Массивы

Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

Определение типа

Ссылки на правосторонние значения

Лямбда-функции

Создание программ с GUI

Управляющие операторы

Условный оператор

```
if ( условие )  
    оператор1  
else  
    оператор2
```

Если требуется выполнить несколько операторов вместо оператор1 или оператор2 то используются операторные скобки { }

Управляющие операторы

Условный оператор. Пример.

Определение максимального из двух чисел.

```
float x, y, max;
```

```
// ...
```

```
cout << "Определение максимального из двух чисел: "
```

```
cout << x << " и " << y << endl;
```

```
if ( x > y )
```

```
    max = x;
```

```
else
```

```
    max = y;
```

```
cout << "Максимальное число: " << max;
```

Управляющие операторы

Цикл со счётчиком

```
for (действие до цикла;  
      условие;  
      действия в конце итерации) {  
  
    оператор1  
    оператор2  
    ...  
    операторN  
  
}
```

Тело цикла выполняется пока ***условие*** истинно

Управляющие операторы

Цикл со счётчиком. Примеры.

Печать чисел от 0 до 10

```
for (int i = 0; i<11; i++) {  
    cout << i << endl; }
```

Заполнение массива случайными целыми числами

```
const int N = 10;  
int a[N];  
for (int i = 0; i<N; i++) {  
    a[i] = rand(); }
```

ru.cppreference.com/w/cpp/numeric/random/rand

Управляющие операторы

Цикл со счётчиком. Примеры.

Печать элементов массива

```
const int N = 10;  
int a[N]  
  
cout << "Набор чисел: ";  
for (int i = 0; i < N; i++) {  
    cout << a[i] << " "; }
```

Управляющие операторы

Цикл с предусловием

```
while (условие) {  
    Тело цикла;  
}
```


Управляющие операторы

Цикл с предусловием. Пример.

Печать строки посимвольно

```
char s[] = "Print Me!";  
  
unsigned i = 0;  
  
while (s[i]!=0){  
    cout << s[i];  
    i++;}
```

В C++ каждая строка заканчивается символом с нулевым кодом.

Управляющие операторы

Цикл с постусловием

```
do {  
    Тело цикла;  
}  
while (Условие)
```

Управляющие операторы

Цикл с постусловием. Пример.

Контроль входных данных

```
float x;  
do {  
    cout << "Введите положительное число > " << endl;  
    cin >> x;  
}  
while ( x <= 0);
```

Управляющие операторы

Совместный цикл

```
for (type item : set) {  
    // тело цикла  
    //использование item  
}
```

В начале каждой итерации цикла в переменную `item` будет записано значение из последовательности `set`.

`set` - массив или любым другим типом имеющим итератор (например `list`), т.е. тип должен допускать перебор элементов.

Управляющие операторы

Совместный цикл. Примеры

```
int my_array[5] = {1, 2, 3, 4, 5};  
for(int x : my_array)  
    cout << x << " ";
```

```
// в X записывается только значение.  
// Этот цикл ничего не изменит в vec1  
for (auto x: vec1) x *= 2;
```

```
// а ЭТОТ изменит  
for (auto& x: vec1) x *= 2;
```

auto используется вместо указания типа, см. определение типа.

Outline

Основы C++

Типы данных

- Указатели и ссылки

- Массивы

- Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

- Определение типа

- Ссылки на правосторонние значения

- Лямбда-функции

Создание программ с GUI

Функции

Общий вид определения (definition) функции.

```
возвр.тип func_name( тип параметр, ... ) // заголовок функции
{
    // тело функции
}
```

Функции

```
// Возврат значения из функции  
float foo( int x ) {  
    return rand()/x; }
```

```
// Функция не возвращающая ничего  
void bar( int x) {  
    cout << x*rand() << endl; }
```


Функции

```
void print_array(int *a, unsigned n){
    for (int i = 0; i<n; i++)
        cout << a[i] << " ";
}

int main(){
    const unsigned M = 4;
    int b1[M] = {1, 2, 3, 4};
    int b2[M] = {10, 20, 30, 40};

    cout << "Набор чисел #1:" << endl;
    print_array(b1,M);
    cout << endl;

    cout << "Набор чисел #2:" << endl;
    print_array(b2,M);
    cout << endl;
}
```

Принцип единственной ответственности?

Параметры-ссылки и параметры-значения

Функции. Параметры-ссылки и параметры-значения

Для фактического параметра переданного "**по значению**" внутри функции создаётся локальная копия. Изменение этой копии (формального параметра) не влияет на фактический параметр.

```
int a = 42;
```

```
// x - формальный параметр-переменная  
void foo ( int x ) { x = 123; }
```

```
foo( a ); // a - фактический параметр  
cout << a; // 42  
// переменная a не изменилась
```

Функции. Параметры-ссылки и параметры-значения

Для фактического параметра переданного в функцию "***по ссылке***", на самом деле передаётся его *адрес*. Значит изменения формального параметра внутри функции означают изменения фактического параметра.

```
int a = 42;
```

```
// x - формальный параметр-ссылка  
void foo ( int &x ) { x = 123; }
```

```
foo( a ); // a - фактический параметр  
cout << a; // 123  
// переменная a изменилась
```

Значения параметров по умолчанию

Функции. Значения параметров по умолчанию

Когда параметр необходим, но функция часто вызывается с определённым его значением, то можно задать для него значение по умолчанию.

```
void foo( int x = 42 ) {cout << x;}
```

```
foo( 123 ); // 123
```

```
foo()      // 42
```

Формальные параметры со значением по умолчанию должны быть последними.

Функции. Значения параметров по умолчанию

- ▶ Используйте для аргументов, значения которых часто принимают одно и то же значение
- ▶ Приводите эти аргументы в последнюю очередь
- ▶ Не используйте неожиданных значений по умолчанию

Перегрузка Overloading

Функции. Перегрузка

Функциям выполняющие одинаковую работу с разными по типу наборами данных можно давать одинаковые имена. Компилятор определит по набору фактических параметров, какая функция должна быть вызвана.

```
void foo(int x){ cout << "1";}
```

```
void foo(float x){ cout << "2";}
```

```
void foo(int x, int y){ cout << "3";}
```

```
foo(20);    // 1
```

```
foo(20.0);  // 2
```

```
foo(1, 2);  // 3
```

```
foo(1, 2.0) // 3
```

Функции. Перегрузка

- ▶ Функциям выполняющим одинаковую работу с разными данными можно давать одинаковые имена
- ▶ Перегруженные функции должны отличаться по типу и количеству параметров
- ▶ Перегруженные функции не отличаются по типу возвращаемого значения
- ▶ При компиляции перегруженным функциям даются разные имена.
- ▶ Какая из перегруженных функций будет вызвана также определяется на этапе компиляции

Переменное число параметров функции variadic arguments

- ▶ Документация:
en.cppreference.com/w/cpp/language/variadic_arguments
- ▶ Описание и примеры:
ravesli.com/urok-111-ellipsis-pochemu-ego-ne-sleduet-ispolzovat/

Функции

Выводы

- ▶ Функции делают возможным алгоритмическую декомпозицию
- ▶ Функции делают возможным повторное использование кода

Функции

Выводы

- ▶ Для того чтобы пользоваться функцией не нужно обладать минимальными знаниями о её внутреннем устройстве
- ▶ Легче повторно использовать функцию служащую одной цели
- ▶ Следует стремиться к чистоте функций
- ▶ Стоит избегать использования глобальных переменных в функциях
- ▶ Параметры, которые дорого копировать следует передавать по ссылке
- ▶ Параметры, переданные по ссылке, но не изменяющиеся в теле функции нужно делать константными.

Ссылки на функции

Outline

Основы C++

Типы данных

Указатели и ссылки

Массивы

Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

Определение типа

Ссылки на правосторонние значения

Лямбда-функции

Создание программ с GUI

Модули

- ▶ В C++ полноценные модули добавили в стандарте C++20.⁷
- ▶ На данный момент (осень 2021) поддержка модулей не полностью реализована в компиляторах⁸
- ▶ В C++ исходный код можно приводить в отдельных файлах, которые подключать в основной
- ▶ Под модулем в C++ можно понимать отдельный файл исходных кодов
- ▶ Подключение модуля = включение содержимого модуля в другой файл исходных кодов
- ▶ Обычно *объявления* переменных, типов данных, функций приводится в заголовочном файле (header file) который имеет расширение .h (или .hpp)
- ▶ *Определения* же приводятся в файлах с расширением .cpp

⁷Стандарт C++20: обзор новых возможностей C++. Часть 1 «Модули и краткая история C++»

⁸en.cppreference.com/w/cpp/compiler_support

Модули

- ▶ Можно говорить, что модуль состоит из двух файлов: заголовочного (см. [wikipedia](#)) и `.cpp` файла. Имена этих файлов (с точностью до расширения) рекомендуется выбирать одинаковыми
например `my_module.h` и `my_module.cpp`
- ▶ Подключать рекомендуется только заголовочные файлы
- ▶ При этом с каждым заголовочным файлом должна быть защита от повторного подключения (см. [Include guard](#))

Модули

- ▶ В C++ исходный код можно приводить в отдельных файлах, которые подключать в основной
- ▶ Обычно *объявления* переменных, типов данных, функций приводится в заголовочном файле (header file) который имеет расширение `.h` (или `.hpp`)
- ▶ *Определения* же приводятся в файлах с расширением `.cpp`
- ▶ Можно говорить, что модуль состоит из двух файлов: заголовочного (см. [wikipedia](#)) и `cpp` файла. Имена этих файлов (с точностью до расширения) рекомендуется выбирать одинаковыми
например `my_module.h` и `my_module.cpp`
- ▶ Подключать рекомендуется только заголовочные файлы. Сначала из стандартной библиотеки, потом собственные.
- ▶ Имена `cpp` файлов передаются компилятору
- ▶ При этом с каждым заголовочным файле должна быть защита от повторного подключения (см. [Include guard](#))

Модули

Пример

my_module.h

```
#ifndef MY_MODULE_H
#define MY_MODULE_H
// глобальная переменная
extern int A;

struct Point{
    float x, y; };

// объявление функции
float distance(const Point &p1,
               const Point &p2);
#endif // MY_MODULE_H
```

main.cpp

```
#include <iostream>
#include "my_module.h"
using namespace std;
int main(){
    Point p1 = {0, 0};
    Point p2 = {3,4};
    float d = distance(p1,p2);
    cout << d;}
```

my_module.cpp

```
#include <math.h>
#include "my_module.h"

// глобальная переменная
int A = 42;

float distance(const Point &p1,
               const Point &p2){
    return pow( pow(p1.x - p2.x, 2)
               + pow(p1.y - p2.y, 2), 0.5 )
}
```

Outline

Основы C++

Типы данных

- Указатели и ссылки

- Массивы

- Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

- Компиляция в командной строке

- Пространства имён

- Синонимы

- static

- Обработка исключительных ситуаций

- Нововведения C++11,14...

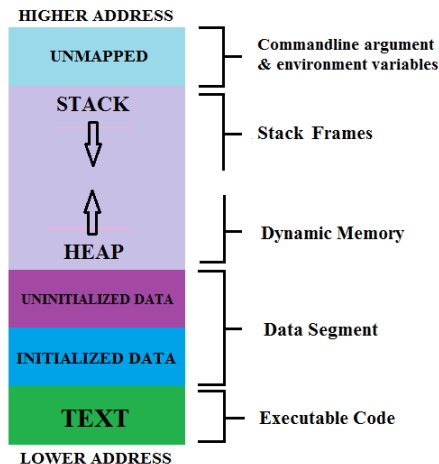
 - Определение типа

 - Ссылки на правосторонние значения

 - Лямбда-функции

- Создание программ с GUI

Устройство памяти программы



- ▶ **Стек (stack):** локальные переменные, адреса вызова функций
- ▶ **Куча (heap):** динамические (созданные во время работы программы, оператором new) переменные
- ▶ **Сегмент данных (data segment):** глобальные и статические переменные
- ▶ **Скомпилированный код программы (executable code)**

Операторы управления динамической памятью

- ▶ **new** выделяет память в куче (Heap), адрес выделенной памяти записывается в указатель.
- ▶ **delete** освобождает память.

Если память была выделена динамически (с помощью оператора `new`), то она обязательно должна быть освобождена вызовом `delete` во избежание *утечки памяти*.

см. примеры использования на слайде 29

Операторы управления динамической памятью

Пример

```
struct Point{
    float x,y;
};

// выделенную в функции память можно использовать и вне это фу
Point* random_point(){
    Point* p = new Point;
    // оператор -> используется вместо . при работе с указател
    // на структуру
    p->x = float(rand()) / RAND_MAX;
    p->y = float(rand()) / RAND_MAX;
    return p; // возвращается указатель на выделенную память
}

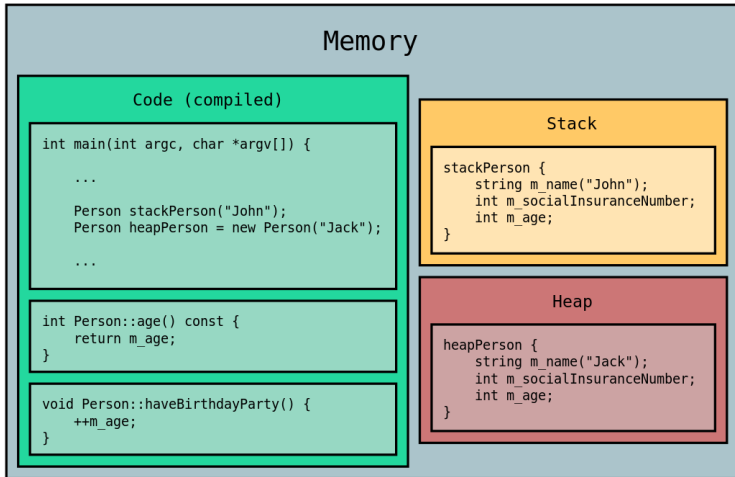
int main(){
    Point *p = random_point();
    delete p;
}
```


Устройство памяти программы

```
class Person {  
  
public:  
    Person(string const& m_name);  
  
    // imagine other methods here  
  
private:  
    string m_name;  
    int m_socialInsuranceNumber;  
    int m_age;  
};  
  
// ...  
  
Person stackPerson("John");  
Person* heapPerson = new Person("Jack");
```

Устройство памяти программы

Пример: стек и куча



Outline

Основы C++

Типы данных

- Указатели и ссылки

- Массивы

- Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

Определение типа

Ссылки на правосторонние значения

Лямбда-функции

Создание программ с GUI

Компиляция

```
g++ main.cpp -o my_prog
```

Будет создан исполняемый файл с именем my_prog

gcc должен быть доступен непосредственно (добавлен в переменную PATH). Иначе потребуются указать полный путь к компилятору, например:

```
C:\Qt\Tools\mingw530\_32\bin\g++.exe
```

Далее будут приведены примеры команд для ОС семейства linux - Debian. Стоит отметить, что в linux для указания пути используется прямой слеш /, а в windows обратный \

Компиляция

Несколько файлов

Если исходный код расположен в нескольких файлах, то для компиляции в параметры `g++` должны быть переданы имена всех `cpp` файлов

Например код содержится в файлах:

`main.cpp`,
`module.cpp`,
`module.h`.

`module.h` подключается в `module.cpp` и `main.cpp`.

```
g++ main.cpp module.cpp -o my_progr
```

Компиляция

Создание статической библиотеки

- ▶ Файлы модулей можно скомпилировать отдельно в статическую библиотеку
- ▶ Потом, при компиляции основной программы указать путь к полученной статической библиотеке
- ▶ Код статической библиотеки будет добавлен к программе и помещён внутрь исполняемого файла
- ▶ Такой подход может сократить время компиляции, если в библиотеку изменения вносятся относительно редко, поэтому нет необходимости компилировать её повторно, вместе с остальными файлами

Компиляция

Создание статической библиотеки

- ▶ Создание объектного файла из исходного кода
`g++ -c module.cpp -o lib/my_lib.o`
будет создан объектный файл `module.o` в заранее созданной папке `lib`
- ▶ Создание статической библиотеки из объектного файла
`ar rvs lib/my_lib.a module.o`
Здесь полученный файл библиотеки `my_lib.a` сохраняется в папку `lib`
- ▶ Компиляция программы с использованием статической библиотеки⁹
`g++ main.cpp lib/my_lib.a -o my_prog`

⁹стоит отметить, что заголовочный файл `module.h` по прежнему необходим, не смотря на то, что он не указывается нигде при компиляции

Компиляция

Создание shared library

- ▶ Код из статической библиотеки полностью включается в исполняемый файл программы
- ▶ Если код такой библиотеки был обновлён всю программу придётся перекомпилировать
- ▶ К тому же одну и ту же библиотеку могут использовать несколько программ
- ▶ Тогда нужно использовать общего пользования (shared library)
- ▶ Таки библиотеки используются как отдельные файлы вместе с исполняемым файлом программы
- ▶ в linux эти файлы имеют расширение so, в Windows - dll
- ▶ gernotklingler.com/blog/creating-using-shared-libraries-different-compilers-different-operating-systems/

Outline

Основы C++

Типы данных

- Указатели и ссылки

- Массивы

- Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

- Определение типа

- Ссылки на правосторонние значения

- Лямбда-функции

Создание программ с GUI

Пространства имён

Пространство имён (namespace) — некоторое множество, под которым подразумевается абстрактное хранилище или окружение, созданное для логической группировки уникальных идентификаторов (то есть имён).

Пространство имён

В пространство имён обычно объединяют несколько связанных между собой функций, классов, типов данных.

Как правило на практике пространство имён это именованная область кода, например в модуле.

Пространства имён

Пример пространства имён

```
namespace my_functions {  
void foo() {...}
```

```
void bar() {...}
```

```
void baz() {...}
```

```
}
```

```
int main(){
```

```
// при использовании идентификатора указывается его пространство
```

```
my_functions::foo();
```

```
foo(); // ошибка: идентификатор foo не найден
```

```
}
```

Пространства имён

Пример

Приведём пример со слайда 68 только использовав пространства имён для модуля

my_module.h

```
...
extern int A;
namespace points{
    struct Point{
        float x, y;};
    float distance(const Point &p1,
                   const Point &p2);
}
#endif // MY_MODULE_H
```

main.cpp

```
...
int main(){
    A = 100;
    points::Point p1 = {0, 0};
    using points::Point;
    Point p2 = {3,4}; // теперь можно так
    using namespace points; // для примера подключим всё
    float d = distance(p1,p2);
    cout << d; }
```

my_module.cpp

```
...
int A = 42;

namespace points {
    float distance(const Point &p1,
                   const Point &p2){
        return pow( pow(p1.x - p2.x, 2)
                    + pow(p1.y - p2.y, 2), 0.5 );
    } }
```

Пространства имён

- ▶ Пространства имён можно дополнять определяя его в разных файлах
- ▶ Что на самом деле и происходит при объявлении пространства имён в заголовочном и cpp файле
- ▶ Но можно не ограничиваться только одним модулем (парой файлов: h и cpp). Одно пространство имён может объединять и несколько модулей.
- ▶ Так устроена стандартная библиотека C++. В каждом из файлов (iostream, fstream, string, ...) определена часть пространства имён std.

Пространства имён

Дополнительно

- ▶ Поиск имён в иерархии пространстве имён
- ▶ Безымянное пространство имён
stepik.org/lesson/53370/step/10?unit=31458

Outline

Основы C++

Типы данных

- Указатели и ссылки

- Массивы

- Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

- Определение типа

- Ссылки на правосторонние значения

- Лямбда-функции

Создание программ с GUI

Синонимы

- ▶ Если имя какого-то пространства имен кажется нам слишком длинным, можно ввести для него короткий синоним:

```
namespace новое_имя = старое_имя;
```

- ▶ То же самое можно сделать и для любого имени:

```
using новое_имя = старое_имя;
```

```
using uint = unsigned int;
```

```
// объявление переменной  
uint x;
```

- ▶ Никогда не следует пользоваться директивой `using` в заголовочных файлах, потому что это приведет к тому, что эта директива будет действовать во всех тех файлах, в которые этот файл будет включаться.

Outline

Основы C++

Типы данных

Указатели и ссылки

Массивы

Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

Определение типа

Ссылки на правосторонние значения

Лямбда-функции

Создание программ с GUI

static внутри функции

```
void foo(){  
    static int x;  
}
```

Локальная переменная объявленная с использованием static:

- ▶ Хранится как глобальная переменная.
- ▶
- ▶ Но доступна только локально.
- ▶ Инициализируется один раз – при первом входе в область видимости.

См. пример с подсчётом числа вызовов функции.

Outline

Основы C++

Типы данных

Указатели и ссылки

Массивы

Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

Определение типа

Ссылки на правосторонние значения

Лямбда-функции

Создание программ с GUI

Обработка исключений

Обработка исключительных ситуаций (exception handling) – механизм языков программирования, предназначенный для описания реакции программы на ошибки времени выполнения и другие возможные проблемы (исключения), которые могут возникнуть при выполнении программы и приводят к невозможности (бессмысленности) дальнейшей отработки программой её базового алгоритма.

Исключительные ситуации

- ▶ Не выполнено предусловие
например функция ожидает в параметре положительное вещественное число, но передано отрицательное
- ▶ Невозможно создать объект (завершить выполнение конструктора)
- ▶ Ошибки типа "индекс вне диапазона"
- ▶ Невозможно получить ресурс
например нет доступа к файлу (файл удалён или не хватает прав доступа)

Исключительные ситуации

Использование кодов возврата

Исключительные ситуации можно обрабатывать используя коды возврата из функции:

```
int foo(float x){
    if (x < 0) // проверка предусловий
        return 1; // если возникла искл.ситуация возвратим 1

    // do something
    return 0; // если всё хорошо, возвращаем 0
}

//...
int main(){
    float x;
    // ...
    int res = foo(x);
    if ( res != 0 ){
        cout << "Ошибка!";
    }
    // ...
}
```

Исключительные ситуации

Использование кодов возврата

Использование кодов возврата не всегда возможно или оправдано

- ▶ Если функция должна возвращать другие данные тогда нужно либо менять возвращаемый тип либо предусмотреть другой способ сообщения об исключительной ситуации внутри функции - например через параметр

Исключительные ситуации

Использование кодов возврата

Использование кодов возврата не всегда возможно или оправдано

- ▶ Если обработать ошибку нужно не в той функции которая вызывает другую

```
int foo(float x){  
    // тут может возникнуть исключение  
}  
void bar(){  
    //...  
    foo();  
    //...  
}  
void baz(){  
    //...  
    bar();  
    // обработка искл. ситуации должна быть здесь  
    //...  
}
```

Обработка исключительных ситуаций

Общая структура кода

```
try {  
    // это защищенный блок кода  
    // ... тут может возникнуть исключение ...  
    // ... в любом месте ...  
    // ... любого вида ...  
}  
catch (тип переменная) {  
    // обработчик исключения  
    // код обрабатывающий исключение  
}  
catch (тип2 переменная) {  
    // обработка остальных исключений  
}  
catch (тип3 переменная) {  
    // обработка остальных исключений  
}  
catch (...) {  
    // Поймать все исключения  
}  
// остальной код
```

Обработка исключительных ситуаций

Пример

```
void foo(float x){
    if ( /* проверка предусловий */
        // если предусловия не выполнены
        throw 1;
    // ...
    if ( /* ещё какая-нибудь проверка */ )
        // если всё плохо...
        throw 2;
    //...
}

int main(){
    float x;
    // ...
    try{
        foo(x);
    }
    catch (int e){
        switch (e) {
            case 1: // справляемся с исключительной ситуацией 1
                break;
            case 2: // справляемся с исключительной ситуацией 2
                break;
        }
    }
}
```

Пример

```
struct Exception{
    int code;
    string message;};

const Exception InvalidArgument = {-1, "Invalid Argument"};
const Exception EmptyArray = {-2, "Empty Array"};

float bmi(float m, float h){
    if (m>0 && h>0) { return m/h/h; }
    else { throw InvalidArgument; }

float print_array(float *a, unsigned n){
    if (a != nullptr && n != 0){
        \\...
    else throw EmptyArray;

int main(){
    try {    bmi(0,2);
            print_array(nullptr, 3);
    } catch (Exception e) {
        if (e.code == InvalidArgument.code) {
            //...
        } else if (e.code == EmptyArray.code) {
            //...
        }
    }
}
```

fail-fast

- ▶ Forgive! подход: приложение продолжает выполняться и старается минимизировать последствия ошибки.
- ▶ Fail Fast! подход: приложение немедленно прекращает работу и сообщает об ошибке.
- ▶ Поход Fail Fast предпочтительнее: желательно, чтобы ошибка автоматически выявлялась на этапе компиляции или, как можно проще и быстрее, в процессе выполнения.

std::exception

en.cppreference.com/w/cpp/error/exception

Outline

Основы C++

Типы данных

Указатели и ссылки

Массивы

Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

Определение типа

Ссылки на правосторонние значения

Лямбда-функции

Создание программ с GUI

Стандарты языка

1. 1983 г. появление языка.
2. C++89/99 (C++ версии 2.0)
3. C++98
4. C++03
5. C++11
6. C++14 (небольшие изменения)
7. C++17
8. C++20 (модули и др.)

Outline

Основы C++

Типы данных

Указатели и ссылки

Массивы

Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

Определение типа

Ссылки на правосторонние значения

Лямбда-функции

Создание программ с GUI

Определение типа во время компиляции

Указание **auto** вместо типа заставляет компилятор самостоятельно подставить тип ориентируясь на задаваемое значение.

```
auto x = 20;           // int
auto y = 3.14159;      // float
auto z = "gues type";  // char*
auto a; // ошибка! Не задано значение!
```

Определение типа во время компиляции

decltype объявляет тип, беря тип другой переменной или выражения.

```
int my_v;  
decltype(my_v) v = 100; // v имеет тип int
```

Информация о типе

```
#include <typeinfo>
auto y = 123.8;
cout << typeid(x).name() << endl; // печатает т

typeid(x) == typeid(xx); // типы можно сравнива
```

cplusplus.com: type_info

Outline

Основы C++

Типы данных

Указатели и ссылки

Массивы

Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

Определение типа

Ссылки на правосторонние значения

Лямбда-функции

Создание программ с GUI

rvalue и lvalue - правосторонние и левосторонние значений

Выражения, которым можно присваивать, называются **lvalue** (left value, т. е. слева от знака равенства). Остальные выражения называются **rvalue**.

Ссылки на rvalue и rvalue

rvalue references – ссылки на правосторонние значения.

Синтаксис

Тип &&

Outline

Основы C++

Типы данных

Указатели и ссылки

Массивы

Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

Определение типа

Ссылки на правосторонние значения

Лямбда-функции

Создание программ с GUI

Лямбда-функции

`[захват](параметры) mutable исключения
атрибуты -> возвращаемый_тип {тело}`

Захват - глобальные переменные используемые функцией (по умолчанию не доступны),

параметры - параметры функции; описываются как для любой функции,

mutable - указывается, если нужно поменять захваченные переменные,

исключения - которые может генерировать функция,

атрибуты - те же что и для обычных функций.

Лямбда-функции. Примеры

Возведение аргумента в квадрат

```
[](auto x) {return x*x;}
```

Сумма двух аргументов

```
[](auto x, auto y) {return x + y;}
```

Лямбда-функции. Примеры

Возведение аргумента в квадрат

```
[](auto x) {return x*x;}
```

Сумма двух аргументов

```
[](auto x, auto y) {return x + y;}
```

Вывод в консоль числа и его квадрата

```
[](float x) {cout << x << " " << x*x << endl;}
```

Тело лямбда-функции описывается также как и обычной функции

```
[](int x) { if (x % 2) cout << "н"; else cout << "ч
```

Лямбда-функции. Примеры

Использование захвата.

= - захватить все переменные.

- захватить переменную по ссылке.

Чтобы изменять переменную захваченную по ссылке нужно добавить ***mutable*** к определению функции.

```
float k = 1.2;
```

```
float t = 20;
```

```
[k](float x) {return k*x;}
```

```
[k,&c](float x) mutable {if (k*x > 0) c = 0; else c
```

Лямбда-функции. Примеры

Когда использовать лямбда функции?

Когда не требуется объявлять функцию заранее.

Функция очень короткая.

Функция нужна один раз.

Функцию лучше всего описать там, где она должна использоваться.

youtube.com/watch?v=5s4CFEMARtU – Яндекс Практикум:
Обзор основных нововведений стандарта C++20

Статья по вебинару

[Хабр: Стандарт C++20: обзор новых возможностей C++. Часть 1](#)
[«Модули и краткая история C++»](#)

Outline

Основы C++

Типы данных

Указатели и ссылки

Массивы

Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

Определение типа

Ссылки на правосторонние значения

Лямбда-функции

Создание программ с GUI

Создание программ с GUI

Для создания приложений с GUI используются сторонние фреймворки, не входящие в стандартную библиотеку C++.

Некоторые из них:

Для Windows

- ▶ Windows Presentation Foundation (WPF)¹⁰

Кроссплатформенные

- ▶ Qt
- ▶ GTK+
- ▶ wxWidgets

¹⁰ входит в состав .NET Framework

Outline

Основы C++

Типы данных

Указатели и ссылки

Массивы

Составные типы данных

Операторы

Функции

Модули

Операторы управления динамической памятью

Компиляция в командной строке

Пространства имён

Синонимы

static

Обработка исключительных ситуаций

Нововведения C++11,14...

Определение типа

Ссылки на правосторонние значения

Лямбда-функции

Создание программ с GUI

Ссылки и литература

1. **Stepik: Программирование на языке С++**
2. **Б. Страуструп Язык программирования С++.** 2013. 350 страниц. Учебник по языку. Шаблоны. ООП. Проектирование.
3. **MSDN: Справочник по языку С++**
4. **Эффективный и современный С++: 42 рекомендации по использованию С++ 11 и С++14.** 2016. 300 страниц. Просмотреть. Изучить. Использовать как справочник. Неформальный стиль. Много примеров. Хорошее знание С++.
5. **stackoverflow.com – система вопросов и ответов**

Ссылки и литература

Документация по языку:

- ▶ ru.cppreference.com – информация по языку и стандартной библиотеке C++. Есть примеры.
- ▶ Zeal - (zealdocs.org) – офлайн документация по языкам программирования и фреймворкам
при первом запуске программы требуется скачать
необходимую документацию

Дополнительно:

- ▶ habr.com/company/pvs-studio/ Блог компании PVS-Studio.
Примеры ошибок в C++ (и не только) коде найденных
статическим анализатором кода PVS-Studio.

Ссылки и литература

Ссылка на слайды
github.com/VetrovSV/OOP