C++

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

Черновик

Кафедра ИВТ и ПМ

2018

План

Outline

О языке

- Общего назначения
- Компилируемый
- Статическая типизация
- Объектно-ориентированный[1]

[1] поддерживаются и другие парадигмы программирования

IDE и компиляторы

- Qt Creator (с компилятором MinGW)
  Кроссплатформенный, лаконичный, свободный, устанавливается вместе с фреймворком Qt[2]
- Visual Studio
- JetBrains CLion
  Кроссплатформеный, есть версия для студентов, нет бесплатной версии

[2] при установке избегать путей с кириллицей

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)[3]

Кроссплатформенные

- **Qt**
- GTK+
- wxWidgets

[3] входит в состав .NET Framework

Библиотеки

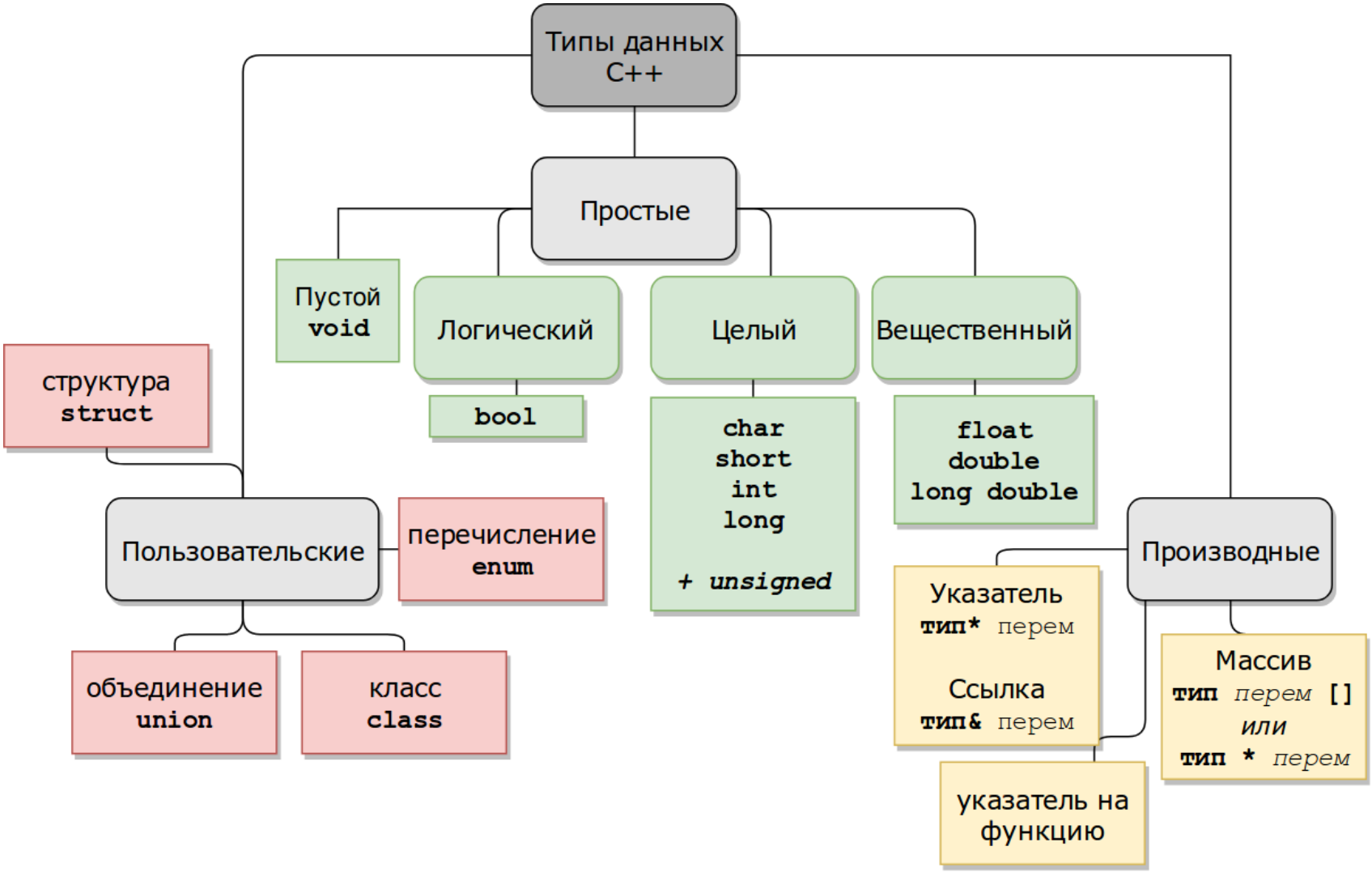
Стандартная библиотека C++ содержит классы для хранения данных (динамический массив, список, и т.д.), для работы с файлами, потоками и др.

Библиотеки

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.

Outline

Типы данных


Типы данных C++
Простые
Пустой void
Логический
Целый
Вещественный
bool
char
short
int
long
+ unsigned
float
double
long double
структура struct
Пользовательские
перечисление enum
объединение union
класс class
Производные
Указатель тип* перем
Ссылка тип& перем
указатель на функцию
Массив тип перем [] или тип * перем

Объявление переменных и констант

```
int n;                  // можно не задавать значение
float x = -47.039;      // можно задавать...

// но константе задавать значение обязательно
const unsigned N = 24;

n = N;
N = n; // Ошибка! Константу поменять нельзя
```

Типы данных

Производные типы

- Указатель (pointer)
- Ссылка
- Массив[4]
- Структура
- Класс
- Перечисление

[4] рекомендуется использовать классы стандартной библиотеки вместо массивов

Вывод данных

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**;

Вывод данных

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

cout << "Hello, World!";

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

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

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 перменной 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 объявлен в заголовочном файле **iostream**, пространстве имён **std**;

Ввод данных

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

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

Outline

Управляющие операторы

- Условный оператор

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

Управляющие операторы

Условный оператор. Пример.

Поиск 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 числами

```
const int N = 10;
int a[N];
for (int i = 0; i<N; i++) {
    a[i] = 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[] = "Prnt 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 (нововведение C++11)

```
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

Функции

Общий вид определения (definition) функции.

```
возвр.тип func_name( тип параметр, ... ) // заголовок ф
{
        // 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;
}
```

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
```

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 значению по умолчанию должны быть последними.

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

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

Функции

- Для того чтобы пользоваться функцией не нужно обладать минимальными знаниями о её внутреннем устройстве
- Легче повторно использовать функцию служащую одной цели
- Следует стремится 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

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 использование идентификатора указывается его пространство

my_functions::foo();

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

}
```

Outline

Обработка исключительных ситуаций

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

Outline

Файлы

```
#include <fstream>
using namespace std;
...
// создать объект для записи в файл
// и открыть текстовый файл для записи
ofstream f("myfile");
// запись в файл
// здесь все данные будут записаны слитно. так лучше не делать
f << "qwerty";
f << 123;
f << 3.14;
f << endl; // записать символ перехода на новую строку
f << 42.5;
f.close();
```

Содержимое созданного файла:

```
qwerty1233.14
42.5
```

https://en.cppreference.com/w/cpp/io/basic

Файлы

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

// создать экземпляр класса ifstream (для чтения файлов)
ifstream f1;
// открыть текстовый файл
f1.open("myfile");
if (f1.is_open()){
string s;
f1 >> s; // s = "qwerty1233.14"
...
f1 >> s; // s = "42.5"
float number = stof(s); // строка -> число
f1.close();
}
```

https://en.cppreference.com/w/cpp/io/basic_ifstream

Outline

Контейнеры

Некоторые контейнеры

- list - двусвязный список
- vector - динамический массив
- map - ассоциативный массив (словарь)
- stack - стэк
- queue - очередь
- pair - пара

Классы контейнеров объявлены в заголовочных файлах с соответствующими именами. Например класс list объявлен в заголовочном файле list.

```
# include <list>
```

vector

vector имитирует динамический массив.

```
#include <vector>
using std::vector;

// пустой вектор типа int
vector<int> myVector;
// зарезервировали память под 10 элементов
myVector.reserve(10);
```

vector

```
typedef vector<float> vectorf;  // лучше создать синоним
unsigned n = 128;
vector<float> v; // можно не указывать размер
vector<float> v2(n); // а можно указывать

vectorf v3(128, 0);  // легко инициализировать нулём
vectorf v4 = {1,2,3,4};  // легко инициализировать массивом

v4.resize(10, 9);

//  cout << v3 << endl; // Так печатать нельзя :(
// вывод значений на экран
for (auto i=0; i<v4.size(); i++)
    cout << v4[i] << " ";
cout << endl;
```

vector. методы

методы и операторы класса vector

- at(индекс) - возвращает элемент по индексу
- индекс возвращает элемент по индексу
- empty() - возвращает true если вектор пуст
- size() - возвращает размер вектора
- clear() - очищает вектор
- pop_back() возвращает последний элемент; элемент удаляется из вектора
- push_back(значение) добавляет значение в конец вектора
- Resize(n, нач_значение) -изменяет размер вектора
- front() - возвращает первый элемент
- back() - возвращает последний элемент

Outline

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. ... 2020 г.

Outline

Outline

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!
```

Рекомендуется использовать auto везде, где не требуется строгого задания типа. Например если необходим тип unsigned, но auto выводит int.

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 тип


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

cplusplus.com: type_info

Outline

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

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=k*x;}
```

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.

Outline

Операторы управления динамической памятью

new
delete

Ссылки и литература

1. **Stepik: Программирование на языке C++**
2. **Б. Страуструп Язык программирования C++.** 2013. 350 страниц. Учебник по языку. Шаблоны. ООП. Проектирование.
3. Эффективный и современный C++: 42 рекомендации по использованию C++ 11 и C++14.
   2016. 300 страниц. Просмотреть. Изучить. Использовать как справочник. Неформальный стиль. Много примеров. Хорошее знание C++.
4. ru.cppreference.com - информация по языку и стандартной библиотеке C++
5. www.stackowerflow.com - система вопросов и ответов

Ссылки и литература

Ссылка на слайды
github.com/VetrovSV/OOP