C++

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

Кафедра ИВТ и ПМ

2018

План

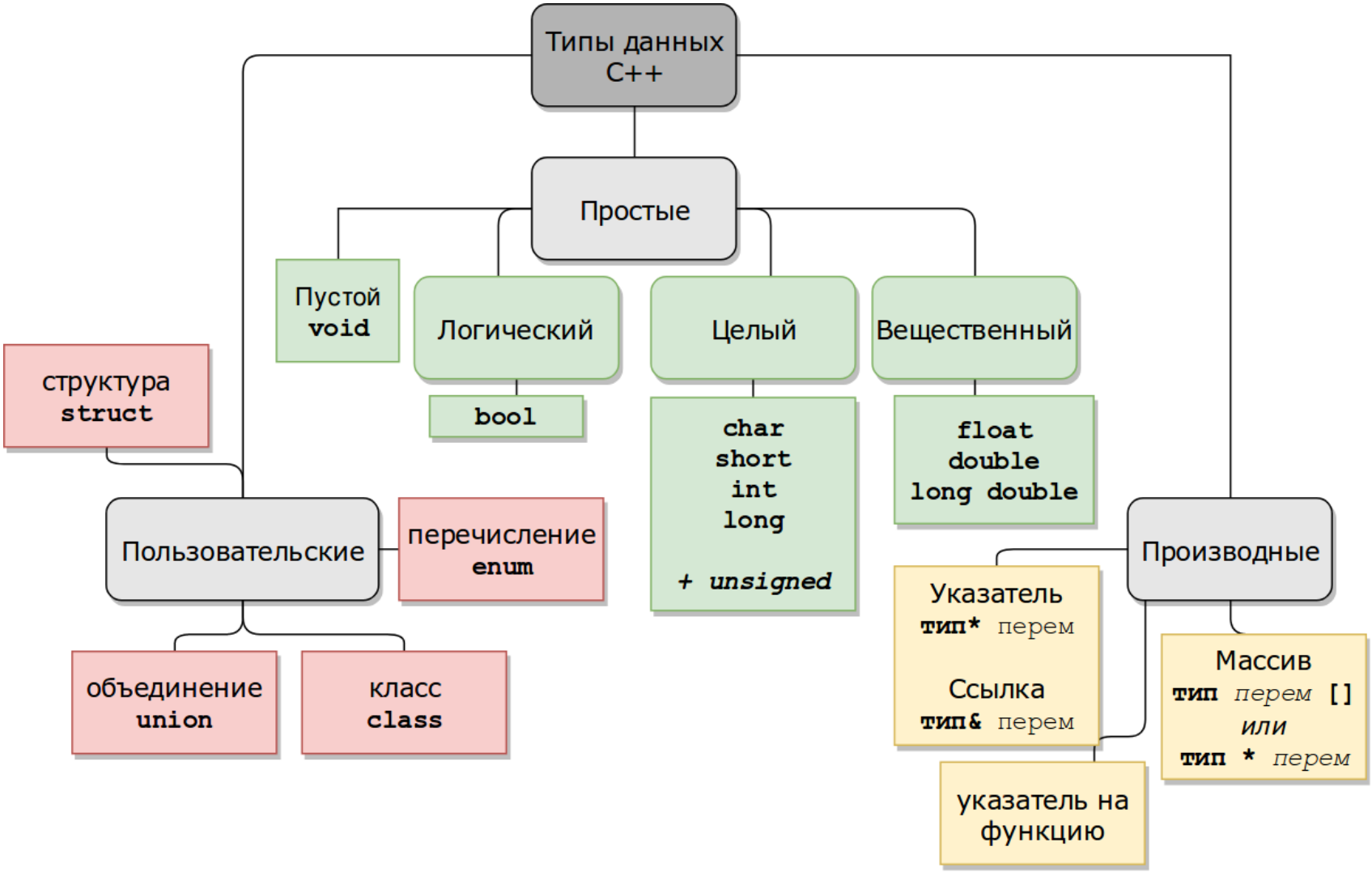
IDE и компиляторы

- Qt Creator (с компилятором MinGW)
  Кроссплатформенный, лаконичный, свободный, устанавливается вместе с фреимворком Qt
- Visual Studio
- CodeBlocks
- JetBrains CLion
  Кроссплатформеный, нет бесплатной версии

Outline

Outline

Типы данных


Типы данных C++
Простые
Пустой void
Логический
bool
Целый
char
short
int
long
+ unsigned
Вещественный
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
- Структура
- Перечисление

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

```
// 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

- Цикл со счётчиком

```
for (действие до цикла;
     условие;
     действия в конце итерации) {

  оператор1
  оператор2
  ...
  операторN
}
```

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

Управляющие операторы

- Цикл с предусловием

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

Управляющие операторы

- Цикл с постусловием

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

Управляющие операторы

- Совместный цикл (нововведение C++11)

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

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;
```

Outline

Функции

Общий вид определения (definition) функции.

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

```
// Пример
float foo( int x ) { return rand()/x; }

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

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.

Outline

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-функция мало чем отличается от 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.

Ссылки на функции

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

Стандартная библиотека шаблонов

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 масси

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) -> элемент с индексом
-

  -> элемент с индексом
- empty() -> true если пуст
- size() -> размер
- clear() очищает вектор
- pop_back() -> последний элемент; элемент удаляется из вектора
- push_back(значение) добавляет значение в конец вектора
- Resize(n, нач_значение)
- front()
- back()

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

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

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