Основы программирования на языке C++

Преподаватель:
Маслов Алексей Владимирович,
доцент кафедры общей физики

Проверка успеваемости и получение оценки за курс

- Экзамен в конце семестра. Экзамен проходит в виде письменного теста (16-20 вопросов) с последующим обсуждение ответов и дополнительных вопросов. Экзаменационные вопросы основаны на материале практических занятий и лекций.
- Промежуточный тест во второй половине апреля, если будет очное обучение.
- Домашние задания на практике. Наличие домашних и посещений может учитываться в случае спорной оценки на экзамене.
- Возможно выставление экзаменационной оценки заранее по результатам промежуточного теста, наличия всех выполненных домашних, посещения практических занятий.

Литература

- Стивен Прата, «Язык программирования C++»

  Stephen Prata, «C++ Primer Plus», 6th edition, (~1400 pages)

- Бьерн Страуструп, «Язык программирования C++»

  Bjarne Stroustrup, «The C++ Programming Language», 4th edition (~1300 pages)

- Stanley B. Lippman, Josee Lajoie, and Barbara E. Moo, «C++ Primer», 5th Edition (~900 pages)

- Bjarne Stroustrup, «A Tour of C++» (~180 pages)

- Brian W. Kernigan and Dennis M. Ritchie, «The C Programming Language», 2nd edition (~270 pages)

- И много других

- Поиск в интернете (google.com, yandex.ru и т.д.)

Для работы на компьютере (программирование, набор текста) удобно владеть «слепым» десятипальцевым методом набора на клавиатуре.

Существует много различных программ (платных, бесплатных, on-line и т.д.) для обучения.

Поиск «слепой набор на клавиатуре» в yandex.ru:

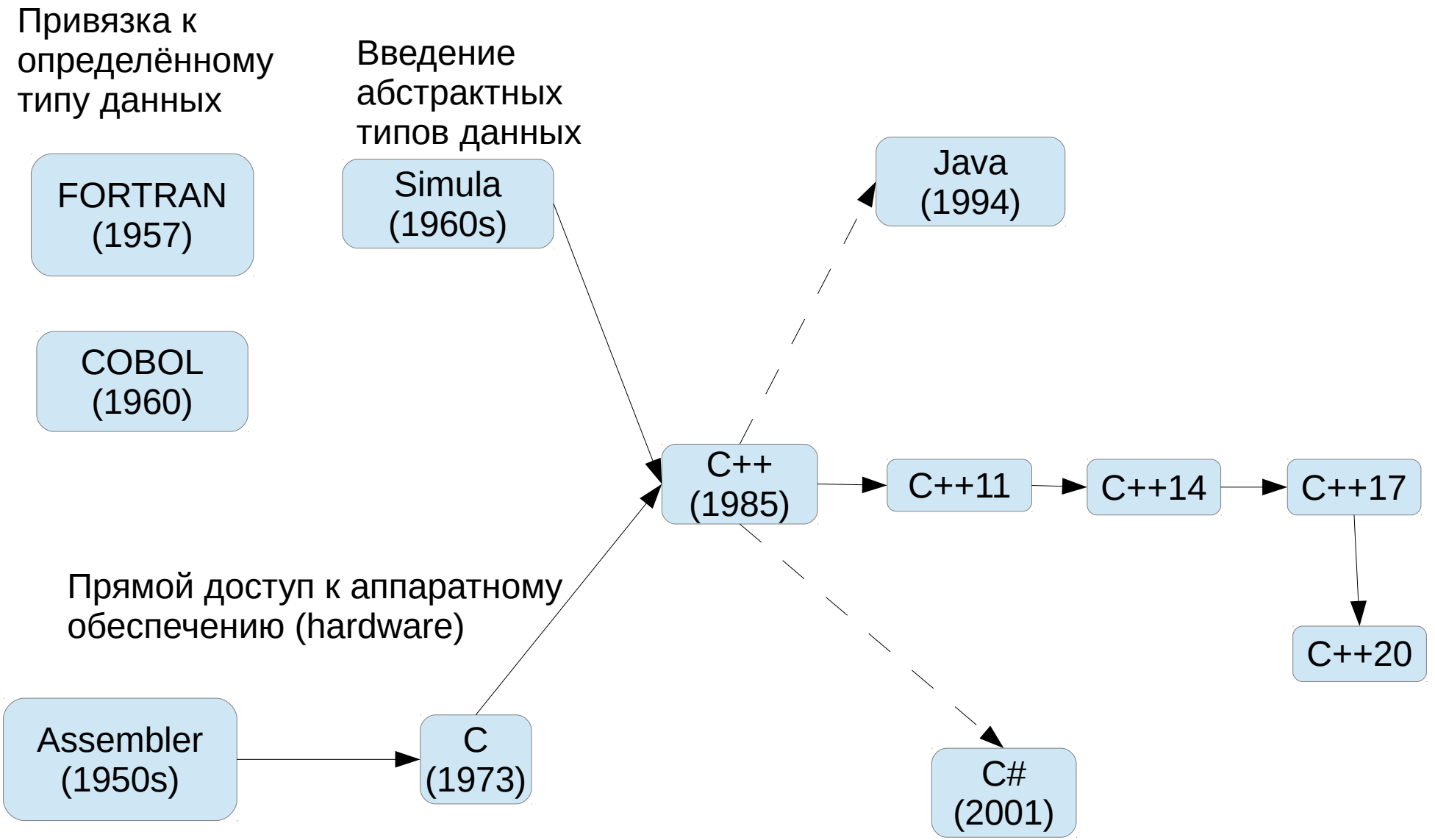
- https://typingstudy.com/
- https://www.ratatype.ru/
- http://nabiraem.ru/, клавиатурный тренажёр «СОЛО на клавиатуре
- https://staminaon.com/ru -- бесплатный клавиатурный тренажёр Stamina 6++
- http://www.rapidtyping.com/, Rapid Typing
- и многие другие ...

Минимальный набор программ для курса

- Операционная система (Linux/Windows)
- Текстовый редактор (Geany, Kwrite, Emacs, XEmacs, Vim, nano и т. д.)
  - подчёркивание синтаксиса (типы данных, имена переменных, имена функций и т.д.)
  - автоматическое форматирование (отступление)
- Компилятор (g++ – GNU C++ компилятор)

Лучше не использовать Интегрированные Среды Разработки (Integrated Development Environments), например, Microsoft Visual Studio

Языки программирования, формирование C++



Краткое описание

- Фортран (FORTRAN, FORmula TRANslator) – компилируемый императивный (использует последовательные инструкции, изменяющие данные) язык с широким спектром приложений, в основном для инженерных и научных расчётов
- Кобол (COBOL, Common Business Oriented Language) – компилируемый императивный язык, используется в коммерческих, финансовых и административных системах (например, для обработка различных транзакций); синтаксис инструкций максимально приближен к обычному английскому языку.
- Ассемблер (assembly language) – язык программирования низкого уровня, который привязан к конкретной операционной системе и архитектуре компьютера.
- Java – объектно-ориентированный язык общего назначения. Один из принципов – возможность выполнять программы под различными платформами без повторного компилирования. Программы компилируются с специальный байт-код (bytecode) и запускаются на любой архитектуре с помощью виртуальной Java машины.
- Симула (Simula) – язык программирования общего назначения, в основном для моделирования сложных систем. Считается одним из первых объектно-ориентированных языков.
- C# – объектно-ориентированный язык программирования, имеет C-подобный синтаксис, используется для разработки различных приложений под Windows.

Обзор языка C

- Структурный язык программирования (основан на использовании циклов, ветвлений, подпрограмм)
- Создан 1972-1973 в Лаборатории Белл (Bell Labs, AT&T, USA), Деннис Ритчи (Dennis Ritchie)

Цели создания:

- Разработка операционной системы (ОС) Unix велась на языке ассемблера, тесно связанного с типом процессора и памяти.
- Проблема переносимости: ОС Unix предназначалась для работы на разных платформах.
- Необходимость языка, сочетающего эффективность языка низкого уровня и возможность доступа к аппаратным средствам с универсальностью и переносимостью языка высокого уровня.
- Создавался для возможности компиляции системы UNIX на различных типах компьютеров.

Парадигмы (модели) программирования

- **Императивное программирование:** вычисления описываются в виде инструкций (операций), шаг за шагом изменяющих состояние программы (например, память в C). Для сложных программ может приводить к использованию многочисленных ветвлений (переходов, goto) к различным наборам инструкций в зависимости от результата какого-либо сравнения. Это усложняет как разработку, так и чтение программ.
- **Структурное программирование:** также оперирует инструкциями и изменяет состояние программы, но вводит такие понятия как блок (составная инструкция), операторы ветвления (if-else, switch) и цикла (for, while, do-while), то есть реализует более строгий подход к императивному программированию.
- **Процедурное программирование:** заключается в разбиение задачи на более мелкие и простые задачи (подпрограммы, процедуры, функции). Если получившаяся задача остается сложной, ее разбивают дальше. Язык C упрощает такой подход, поощряя программистов использовать отдельные программные элементы (функции).
- **Модульное программирование:** разделение кода на большие единицы (модули) по сравнению с функциями. Объединение отдельных частей программы, отвечающий за реализацию какой-либо функциональности. Разбиение программы на отдельные файлы, пакеты программ, библиотеки.

Упрощённая программа на C как иллюстрация парадигм программирования

Файл 1:

```c
#include<stdio.h>

double func(double x);

int main(void){

	printf("Hello, World!\n");

	double a = 0.5;
	double b = func(a);

	printf("func(%g)=%g\n", a, b);

	return 0;
}
```

Файл 2:

```c
#include<math.h>

double func(double x){

	if( x > 0){ // блок
	  x=sin(x);
	  return x;
	 } else {
	  return 0.0;
	 }
}
```

Императивное: последовательность инструкций

Структурное: ветвление, блок

Процедурное: разбиение на подпрограммы (функции)

Модульное: разбиение на файлы (модули)

Процедурное программирование

Данные – это информация, которую использует и обрабатывает программа.

Алгоритмы – это методы, используемые программой.

Процедурное программирование сводится к разбиению задачи на более мелкие, набор переменных, структур данных, *разработке алгоритмов* для их обработки.

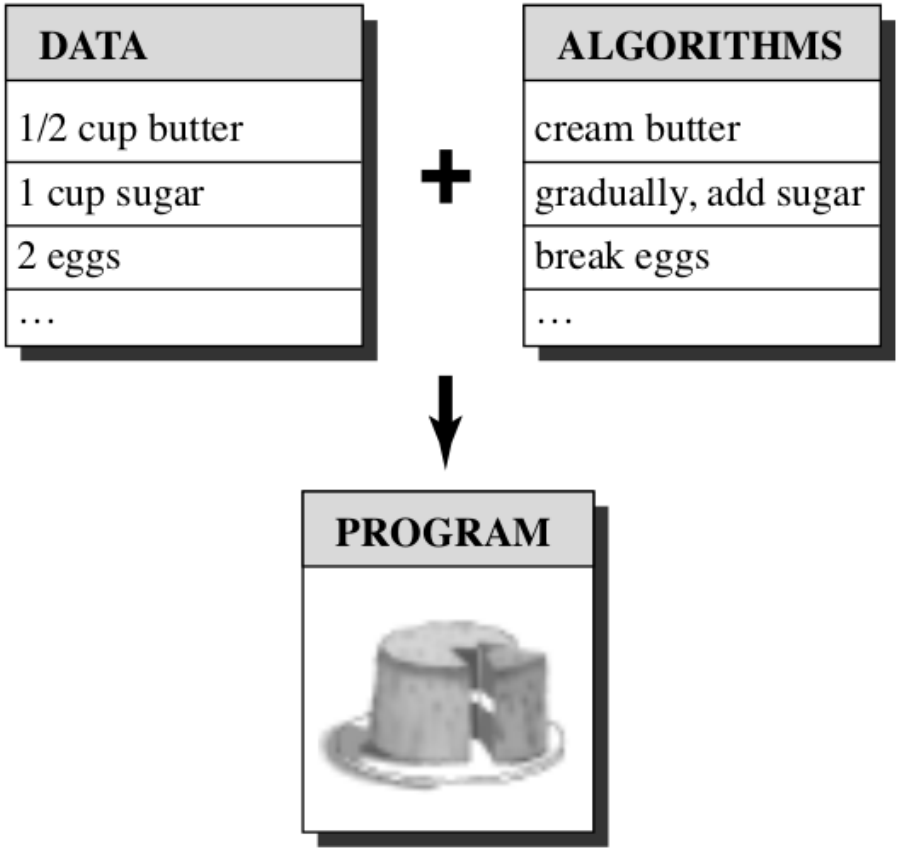


Figure 1.1 Data + algorithms = program.

Описание курса

C++ – это:

- процедурный язык, представленный C;
- объектно-ориентированный язык, представленный расширениями в форме классов (абстрактные типы данных), которые C++ добавляет к C;
- обобщенное программирование, поддерживаемое шаблонами C++.

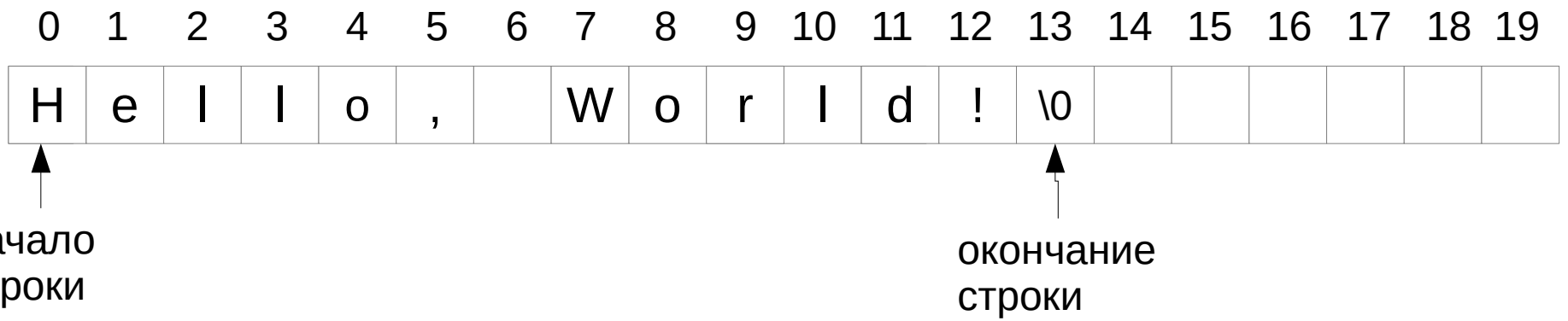
В курсе рассматриваются как элементы языка C, лежащего в основе C++, так и новые компоненты:

- рассмотрение функциональных возможностей, общих для обоих языков;
- различия между C++ и C;
- классы C++;
- шаблоны C++.

Строки в C++ как пример объектно-ориентированного программирования

C:

```
#include<stdio.h> // заголовочный файл для операций ввода и ввывода на экран
#include<string.h> // дла работы со строками
int main(void){
   char my_string_1[20];                // создаём массив символов (не строка)
   // my_string_1= "Hello, world!";     нельзя присваивать значения массивам
   sprintf(my_string, "Hello, world!");  // должен быть достаточный размер
   char my_string_2[10];                // новый массив
  // my_string_2 = my_string_1; нельзя присваивать массивы
   strcpy(my_string_2, my_string_1); // копирование my_string_1 в my_string_2
                                     // практически недетектируемая ошибка
   printf("my_string_2=\"%s\"\n", my_string_2);
   return 0;
}
```



Надо помнить об том, сколько выделено места для хранения строк в C.

Строки в C++ как пример объектно-ориентированного программирования

C++:

```
#include<string>
#include<iostream>


using namespace std;
int main(void){
  string my_string_1;     // создаём пустую строку
  my_string_1 = "Hello, World!";  // присваиваем значение, место выделяется автоматически
  cout << my_string_1; // вывод строки на экран
                       // cout — объект, стандартный поток (stream)вывода
                       // << оператор направления информации
  string my_string_2 = my_string_1; // копируем
  cout << my_string_2;
  return 0;
}
```

Строки в C++ - это уже некоторой новый тип данных (объект), для которого определены свой собственные операции.

C++ использует абстрактные понятия «потоки» (streams) для обмена информацией с текстовым терминалом (консоль).

Вся информация загружается в потоки (поток вывода, output stream, cout) либо берётся из потоков (поток ввода, input stream, cin).

Переход к C++: объектно-ориентированное программирование (ООР)

В ООП во главу угла поставлены данные, а не алгоритмы.

Создаются такие формы данных, которые могли бы выразить важные характеристики решаемой задачи.

Класс – спецификация, описывающая форму данных. Данные отображают задачу, а не «компьютерные» типы данных.

Объект – индивидуальная структура данных, созданная в соответствии с классом.

Сокрытие информации – предотвращение несанкционированного доступа к данным.

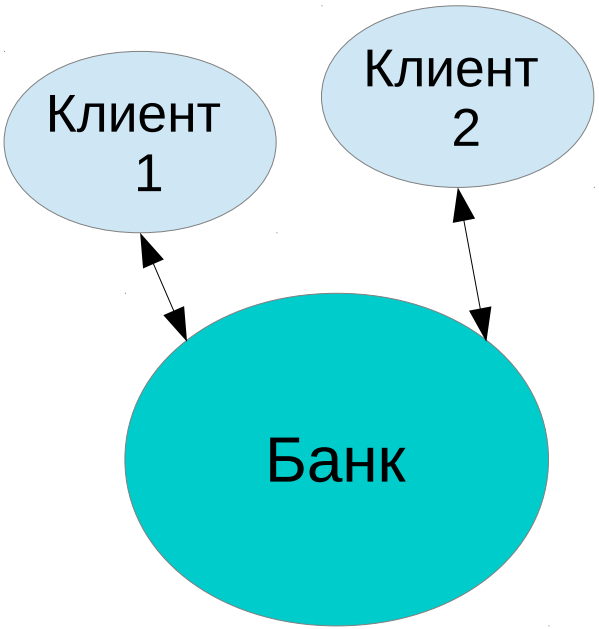
Наследование для создания новых классов на основе существующих классов.

Эффективное управление ресурсами (например, избавление от утечек памяти). Избежание ошибок (выход за пределы массива).

Классы и объекты

Класс: Клиент банка
Объект: Иванов И. И.

Методы
(функции-элементы)
определяют поведение/
правила пользования



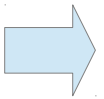
Классы и объекты

Класс: Собака
Объект: Шарик

Пример: компьютерные игры

Избежание ошибок в C

```
char str[10];
printf("Enter a string -> ");
int i=scanf("%s", str);
printf("i=%d, str=\"%s\"\n", i, str);
```



Segmentation fault

scanf() – читает строку до пробельного символа

```
char str[10];
printf("Enter a string -> ");
//  int i=scanf("%s", str);
int i=scanf("%9s", str);
printf("i=%d, str=\"%s\"\n", i, str);
```

C++ и обобщённое программирование

Пример: сортировка массива вещественных и целых чисел

Обобщённое программирование направлено на упрощение повторного использования кода и на абстрагирование общих концепций.

Акцент программирования ставится на независимость от конкретного типа данных.

Можно один раз написать функцию для обобщённого (т.е. не указанного) типа и применять её для множества существующих типов. Для этого в языке C++ предусмотрены шаблоны.

Стандарты

1998 г. C++98: Описание существующих средств языка C++ и расширений языка.

2003 г. C++03: Исправлены ошибки первого издания (опечатки, неточности и т.п.), при этом средства языка программирования не менялись.

2011 г. C++11: Добавляет к языку множество средств. Удаление противоречий, а также упрощение изучения и применения C++.

2014 г. C++14: Сделаны мелкие изменения, устранены небольшие ошибки.

2017 г. C++17: Дальнейшее совершенствование стандарта.

2020 г. C++20: Выпущен в декабре 2020

Специальные опции компилятора: g++ -std=c++11 file.cpp

Определение версии по умолчанию:

```
#include <iostream>
int main() {
  std::cout << __cplusplus << std::endl;
}
```

```
199711 for C++98
201103 for C++11
201402 for C++14
201703 for C++17
```

Основные понятия (примерный план лекций)

1. Типы данных, операторы, выражения
2. Управление (инструкции, блоки, циклы, переключатели)
3. Функции и структура программы
4. Указатели и массивы
5. Структуры
6. Интерфейс (ввод-вывод)
7. Модели памяти и пространства имён
8. Объекты и классы

Работа с встроенными (базовыми) типами данных

1. Целочисленные
    1.1. Целые числа: int, short, long, signed, unsigned и т. д.
    1.2. Символы (1 байт): char
    1.3. Логические переменные: bool

2. Вещественные числа или числа с плавающей точкой (запятой):
    2.1 float, double, long double

Имена переменных

Приветствуется назначение переменным осмысленных имен. Стоимость поездки: cost_of_trip или costOfTrip, но не x или cot.

Правила именования:

• В именах разрешено использование только алфавитных символов, цифр и символа подчеркивания «_».

• Первым символом имени не должна быть цифра.

• Символы в верхнем и нижнем регистре рассматриваются как разные.

• В качестве имени нельзя использовать ключевые слова C++: (int, long, double, for, while, return и т. д.).

Правила именования переменных

- Имена, которые начинаются с двух символов подчёркивания или с одного подчёркивания и следующей за ним буквы в верхнем регистре, зарезервированы для использования реализациями C++, т.е. с ними имеют дело компиляторы и ресурсы. Например, переменная __func__ содержит имя функции, в которой берётся значение этой переменной.
- Имена, начинающиеся с одного символа подчёркивания, зарезервированы для применения в качестве глобальных идентификаторов в реализациях.
- На длину имени не накладывается никаких ограничений, и все символы в имени являются значащими. Однако некоторые платформы могут вводить свои ограничения на длину.

Имена, назначаемые переменным

```
int  poodle;     // допустимое
int  Poodle;     // допустимое и отличающееся от poodle
int  POODLE;     // допустимое и отличающееся от двух предыдущих
Int  terrier;    // ошибка - надо int, а не Int
int  my_stars3; // допустимое
int  _Mystars3; // допустимое, но зарезервированное
int  4ever;      // недопустимое, так как начинается с цифры
int  double;     // недопустимое, так как double - ключевое слово
int  __fools;    // допустимое, но зарезервированное
int  the_very_best_variable_i_can_be_version_112; // допустимое
int  honky-tonk; // недопустимое, так как дефисы не разрешены
```

Целочисленные типы

Базовые целочисленные типы, в порядке увеличения ширины:
char, short, int, long, long long (C++11).

Каждый имеет версии со знаком (signed) и без знака (unsigned).

- целочисленный тип short имеет ширину не менее 16 битов (2 байта);
- целочисленный тип int как минимум такой же, как short. Часто 4 байта;
- целочисленный тип long имеет ширину не менее 32 битов и как минимум такой же, как int. Часто 8 байт;
- целочисленный тип long long имеет ширину не менее 64 битов и как минимум такой же, как long. Часто тоже 8 байт.

```
short  a;           // создает целочисленную переменную типа short
signed short int a; // то же, что и short int;
int  temperature;   // создает целочисленную переменную типа int
long  position;     // создает целочисленную переменную типа long

sizeof(int); // возвращает размер типа или переменной в байтах
```

В C++: 1 байт = количество бит (8 или больше), которое необходимо для набора символов на данной платформе.

Обычно, 8 бит (ASCII). Точное значение прописано в CHAR_BIT.

sizeof(char) -> 1 байт

Заголовочный файл climits (limits.h)

Определяет именованные константы:

#define INT_MAX 32767

Символическая константа	Ее представление
CHAR_BIT	Количество битов в `char`
CHAR_MAX	Максимальное значение `char`
CHAR_MIN	Минимальное значение `char`
SCHAR_MAX	Максимальное значение `signed char`
SCHAR_MIN	Минимальное значение `signed char`
UCHAR_MAX	Максимальное значение `unsigned char`
SHRT_MAX	Максимальное значение `short`
SHRT_MIN	Минимальное значение `short`
USHRT_MAX	Максимальное значение `unsigned short`
INT_MAX	Максимальное значение `int`
INT_MIN	Минимальное значение `int`
UINT_MAX	Максимальное значение `unsigned int`
LONG_MAX	Максимальное значение `long`
LONG_MIN	Минимальное значение `long`
ULONG_MAX	Максимальное значение `unsigned long`
LLONG_MAX	Максимальное значение `long long`
LLONG_MIN	Минимальное значение `long long`
ULLONG_MAX	Максимальное значение `unsigned long long`

Инициализация

```
int n_int = INT_MAX; // инициализация константой
int uncles =5;       // инициализация значением 5
int aunts = uncles;  // инициализация другой переменной
int chairs = aunts + uncles + 4; // chair=14
int wrens(432);// альтернативный синтаксис C++, wrens=432


// Для C++11 можно также
int emus{7};       // устанавливает emus в 7
int rheas = {12}; // устанавливает rheas в 12
int rocs = {};     // устанавливает rocs в 0
int eagles{};      // устанавливает eagles в 0
```

Как выбирать целочисленный тип

- int обеспечивает наиболее естественное представление целых чисел для данной системы (платформы)

- Если не нужны отрицательные значения, то можно выбирать unsigned int;

- Если выбран int на системе, где int занимает 32 бит, а затем программа используется на другой системе, где int занимает 16 бит, то программа не будет работать правильно. Тогда надо выбрать long int.

Схема представления целых чисел в памяти (зависит от архитектуры, а не от языка)

На примере 4 битов

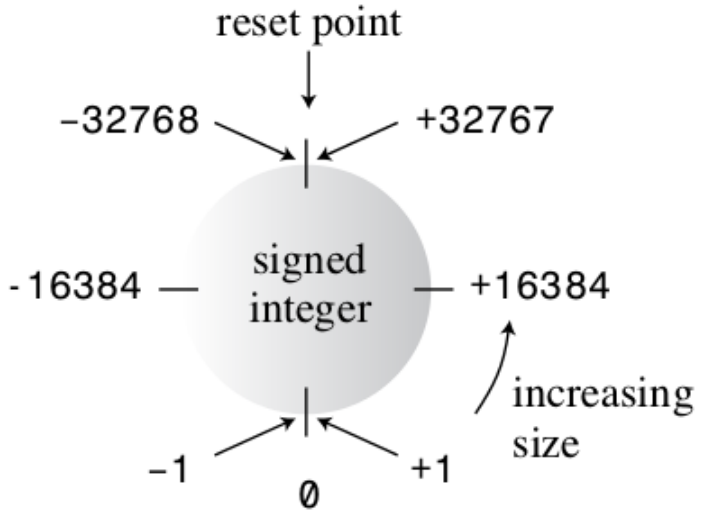
$2^4=16$

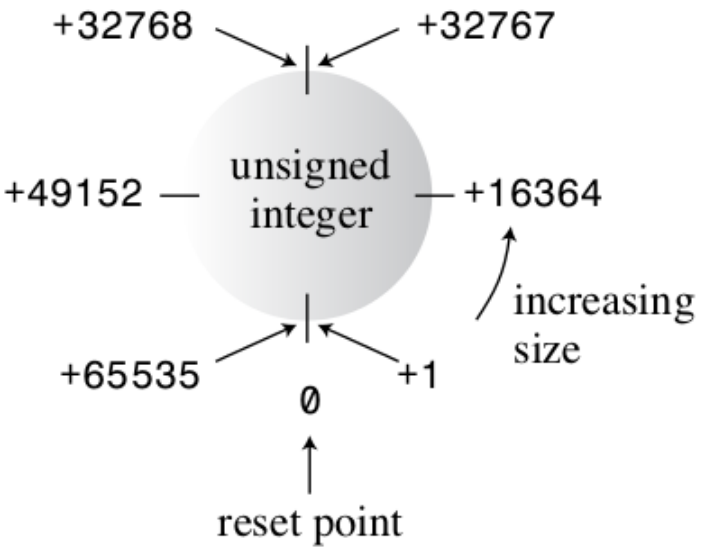
Обычно, int – 32 бита

$2^{32}=4\ 294\ 967\ 296$

Unsigned		Signed		Signed (reordered)	
0000	0	0000	0	1000	-8
0001	1	0001	1	1001	-7
0010	2	0010	2	1010	-6
0011	3	0011	3	1011	-5
0100	4	0100	4	1100	-4
0101	5	0101	5	1101	-3
0110	6	0110	6	1110	-2
0111	7	0111	7	1111	-1
1000	8	1000	-8	0000	0
1001	9	1001	-7	0001	1
1010	10	1010	-6	0010	2
1011	11	1011	-5	0011	3
1100	12	1100	-4	0100	4
1101	13	1101	-3	0101	5
1110	14	1110	-2	0110	6
1111	15	1111	-1	0111	7

Типы без знаков (unsigned), переполнение





C++ не гарантирует такое поведение для типов «signed», но такое поведение наиболее распространённое

16 bit integer

$2^{16}=65536$

$2^{15}=32768$

Целые константы
(или литералы, то есть обозначения целых чисел)

Три системы счисления:

С основанием 10: первая цифра находится в диапазоне 1-9

int a=93; // это $93 = 9*10^1 + 3*10^0$

С основанием 8: первой цифрой является 0, а вторая цифра находится в диапазоне 1-7

int a=042; // это $34 = 4*8^1 + 2*8^0$

С основанием 16: первыми двумя символами являются 0x или 0X. Для обозначения недостающих цифр используют символы (10=a, 11=b, 12=c, 13=d, 14=e, 15=f):

int a=0x42; // это $66 = 4*16 + 2*16^0$

int a=0xF; // можно 0xf, это 15

int a=0xA5; // это $165 = 10*16^1 + 5*16^0$

Определение компилятором C++ типа константы

```
cout << "Year = " << 2017  << "\n";
```

C++ хранит целочисленные константы в виде int, если только нет причины поступать по-другому. Причин может быть две:

- используется специальный суффикс, чтобы указать конкретный тип;
- значение слишком большое, чтобы его можно было уместить в int.

Суффиксы - это буквы, размещаемые в конце числовой константы для обозначения типа константы.

- Суффикс l или L в целом числе означает, что оно относится к типу long,
- Суффикс u или U – unsigned int,
- Суффикс ul, Ul, uL и UL – unsigned long,
- Суффикса для short нет.

```
long var = 11122233344455566L; // можно и без L
```

Например, в системе, с 16-битный int и 32-битный long, число 22022 сохраняется в 16 битах как int, а число 22022L – в 32 битах как long.

C++11:

Суффиксы ll и LL для типа long long,

ull, Ull, uLL и ULL для типа unsigned long long.

Зачем нужны суффиксы?

Наличие суффиксов может влиять на результат операции.

```
UINT_MAX=4,294,967,295
 INT_MAX=+2,147,483,647
 INT_MIN=-2,147,483,648
LONG_MAX=+9,223,372,036,854,775,807
```

```
int  a = 2000000000;
long b = 1000000000  + a;
long c = 1000000000L + a;

cout << "a=" << a << endl;
cout << "b=" << b << endl;
cout << "c=" << c << endl;
```

```
a=+2,000,000,000
b=-1,294,967,296
c=+3,000,000,000
```

```
int  d = 200000;
long e = 100000  * d;
long f = 100000L * d;

cout << "d=" << d << endl;
cout << "e=" << e << endl;
cout << "f=" << f << endl;
```

```
d=+200,000
e=-1,474,836,480
f=+20,000,000,000
```

```
cout << showpos;                          // печатает + или -
cout.imbue(locale("en_US.UTF-8")); // разделяет разряды
```

Зачем нужны суффиксы? (2)

Не следует использовать signed и unsigned в одном выражении.

```
if(-1 < 1 )
  cout << "true"  << endl;
else
  cout << "false" << endl;
```

true or false?

```
if(-1 < 1u)
  cout << "true"  << endl;
else
  cout << "false" << endl;
```

true or false?

```
cout << 2-5  << endl;
cout << 2-5u << endl;
int aa =2-5u;
cout << aa   << endl;
```

```
-3
4,294,967,293
-3
```

Если два операнда (аргумента) одного оператора имеют один тип, но один — signed, а другой — unsigned, то они преобразуются к типу unsigned перед осуществлением операции.

Тип char: символы

Символьные переменные хранятся как числа. Преобразование осуществляется операциями (например, cin, cout) в соответствии с таблицей ASCII.

```
char v1;          // объявление символьной переменной
char v2='D';   // объявление и инициализация
char v3=68;
char v4=0104;
char v5=0x44;
char v6=0b001000100; // двоичное число, начиная с C++14
char v7=0b1'000'100;   // двоичное число с разделением разрядов
cout << "v1=" << v1 << endl; // вывод на экран
```

Таблица ASCII (American Standard Code for Information Interchange)

Десятичный код	Восьмеричный код	Шестнадцатеричный код	Двоичный код	Символ
68	0104	0x44	01000100	D
69	0105	0x45	01000101	E
70	0106	0x46	01000110	F
71	0107	0x47	01000111	G
72	0110	0x48	01001000	H
73	0111	0x49	01001001	I

Литералы char

'A' соответствует 65, коду ASCII для символа A;
'a' соответствует 97, коду ASCII для символа a;
'5' соответствует 53, коду ASCII для цифры 5;
' ! ' соответствует 33, коду ASCII для символа !

```
char v='A';        char v=65;        char v=0x41;
cout << v;
```

Название символа	Символ ASCII	Код C++	Десятичный код ASCII	Шестнадцатеричный код ASCII
Новая строка	NL (LF)	\n	10	0xA
Горизонтальная табуляция	HT	\t	9	0x9
Вертикальная табуляция	VT	\v	11	0xB
Забой	BS	\b	8	0x8
Возврат каретки	CR	\r	13	0xD
Предупреждение	BEL	\a	7	0x7
Обратная косая черта	\	\\	92	0x5C
Знак вопроса	?	\?	63	0x3F
Одинарная кавычка	'	\'	39	0x27
Двойная кавычка	"	\"	34	0x22

Тип bool (логический тип данных, булевый)

Переменные типа bool могут принимать значения true (истина) или false (ложь). Используются для проверки условий, например, в операторе ветвления if().

```
bool  is_ready = true;
```

Литералы true и false могут быть преобразованы в тип int, причём true преобразовывается в 1, а false в 0:

```
int ans = true;          // ans присваивается 1
int promise = false; // promise присваивается 0
```

Любое числовое значение или значение указателя может быть преобразовано неявно в значение bool. Любое ненулевое значение преобразуется в true, а нулевое значение – в false:

```
bool start = -100;  // start присваивается true
bool stop = 0;         // stop присваивается false
```

Часто sizeof(bool) = sizeof(char) = 1 byte

(1 байт — минимальный размер ячейки памяти с адресом)

Квалификатор const

```
const int Months = 12;   // Months  это символическая константа для 12
```

Компилятор не позволяет в последующем изменять значение Months:

```
Month += 1; // выдаст ошибку
```

Переменные с const должны всегда инициализироваться!

```
const int var1; // неправильно, значение var1 в этот момент не определено
```

```
int var=7;
const int b=var; // можно инициализировать, но var должно иметь значение,
                 // но может откомпилироваться и без этого. Надо -Wall.
```

В C также использовали определение директивы препроцессора

```
#define Months 12
```

Числа с плавающей точкой для представление действительных (вещественных) чисел

$$123 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$$

$$123.45 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2}$$

$$123.45 = 1.2345 \cdot 10^2 = 1.2345\text{E}2 = 12.345\text{E}1 = 123.45\text{E}0 = 0.12345\text{E}3$$

Нормализованная запись: 1 <= |мантисса| <10

93900.132 // число с плавающей точкой
0.00023 // число с плавающей точкой
8.0 // тоже число с плавающей точкой
2.52e+8 // можно использовать E или e; знак + необязателен
8.33E-4 // порядок может быть отрицательным
-18.32e13 // перед записью может стоять знак + или -

Числа с плавающей запятой (floating point numbers)

Десятичные числа

$$\underset{10^3}{\square}\ \underset{10^2}{\square}\ \underset{10^1}{\square}\ \underset{10^0}{\square}\ .\ \underset{10^{-1}}{\square}\ \underset{10^{-2}}{\square}\ \underset{10^{-3}}{\square}\ \underset{10^{-4}}{\square}$$

$$123.45 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2}$$

Двоичные числа

$$\underset{2^3}{\square}\ \underset{2^2}{\square}\ \underset{2^1}{\square}\ \underset{2^0}{\square}\ .\ \underset{2^{-1}}{\square}\ \underset{2^{-2}}{\square}\ \underset{2^{-3}}{\square}\ \underset{2^{-4}}{\square}$$

$$0.1_{10} = 0.0001100110011\ldots_2 = 1 \times 2^{-4} + 1 \times 2^{-5} + 1 \times 2^{-8} \ldots$$

Нормализованная форма:

Десятичная форма
(мантисса: 1<=m<10)

$$123.45 = 1.2345 \times 10^2 = 1.2345\mathrm{E2}$$

Двоичная форма
(мантисса: 1<=m<2)

$$0.0001100110011_2 = 1.100110011 \times 2^4$$

Типы чисел с плавающей точкой: float, double, long double

Размер в битах зависит от платформы.
double должен быть не меньше чем float, а long double – не меньше double

Обычно, float – 32 бит, double – 64 бит; long double может быть 128 бит.

```
cout << 1.234 ;  // по умолчанию все постоянные - double
cout << 1.2e2f ; // f,F=float; l, L=long double
```

Файл cfloat (float.h):

```
// Минимальное количество значащих цифр
#define DBL_DIG 15 // double
#define FLT_DIG 6    // float
#define LDBL_DIG 18 // long double

// Количество бит, используемых для представления мантиссы
#define DBL_MANT_DIG 53
#define FLT_MANT_DIG 24
#define LDBL_MANT_DIG 64

// Максимальные и минимальные значения порядка
#define DBL_MAX_10_EXP +308
#define FLT_MAX_10_EXP +38
#define DBL_MIN_10_EXP -307
#define FLT_MIN_10_EXP -37
```

Арифметические операции в C++

• Операция + выполняет сложение операндов. 4 + 20 дает 24.

• Операция - вычитает второй операнд из первого. 12-3 дает 9.

• Операция * умножает операнды. 28*4 дает 112.

• Операция / выполняет деление первого операнда на второй. 100 / 5 дает 20.

Если оба операнда являются целыми числами, то результат будет равен целой доли частного. Например, 17/3 дает 5, с отброшенной дробной частью.

• Операция % находит остаток от деления первого операнда на второй.

19 % 6 равно 1, поскольку 6 входит в 19 три раза, с остатком 1.

Приоритеты: операции *, /, % имеют более высокий приоритет чем +, -

```
int a = 3 + 4 * 5; // 35 или 23 ?
int b = (3+4)*5;  //
int c = 120/4*5;  // 150 или 6 ?
```

Преобразования типов

- C++ преобразует значения во время присваивания значения одного арифметического типа переменной, относящейся к другому арифметическому типу (double x = 1/2;).
- C++ преобразует значения при комбинировании разных типов в выражениях (double x = 1.0/2;).
- C++ преобразует значения при передаче аргументов функциям.

```
double x = 4.5;
int a = x;  // из double в int
```

Тип преобразования	Возможные проблемы
Больший тип с плавающей точкой в меньший тип с плавающей точкой, например, `double` в `float`	Потеря точности (значащих цифр); исходное значение может превысить диапазон, допустимый для целевого типа, поэтому результат окажется неопределенным
Тип с плавающей точкой в целочисленный тип	Потеря дробной части; исходное значение может превысить диапазон целевого типа, поэтому результат будет неопределенным
Больший целочисленный тип в меньший целочисленный тип, например, `long` в `short`	Исходное значение может превысить диапазон, допустимый для целевого типа; обычно копируются только младшие байты.

Приведение типов

```
(имяТипа) значение // преобразует значение в тип имяТипа
имяТипа (значение) // преобразует значение в тип имяТипа

int thorn=3;
(long) thorn; // преобразует переменную thorn в тип long, работает в C
long (thorn); // преобразует переменную thorn в тип long, не работает в C

cout << int('Q'); // отображает целочисленный код для 'Q'
cout << char(65) << endl; // напечатает A
cout << (char) 65 << endl; // напечатает A
```

C++11

Указание auto вместо имени типа в инициализирующем объявлении

```
auto n = 100;       // n получает тип int
auto x = 1.5;        // x получает тип double
auto y = 1.3e12L; // y получает тип long double
```