

Языки программирования

ПМИ 2 курс

Демяненко Я.М.

ЮФУ 2023

C++— компилируемый, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования, как

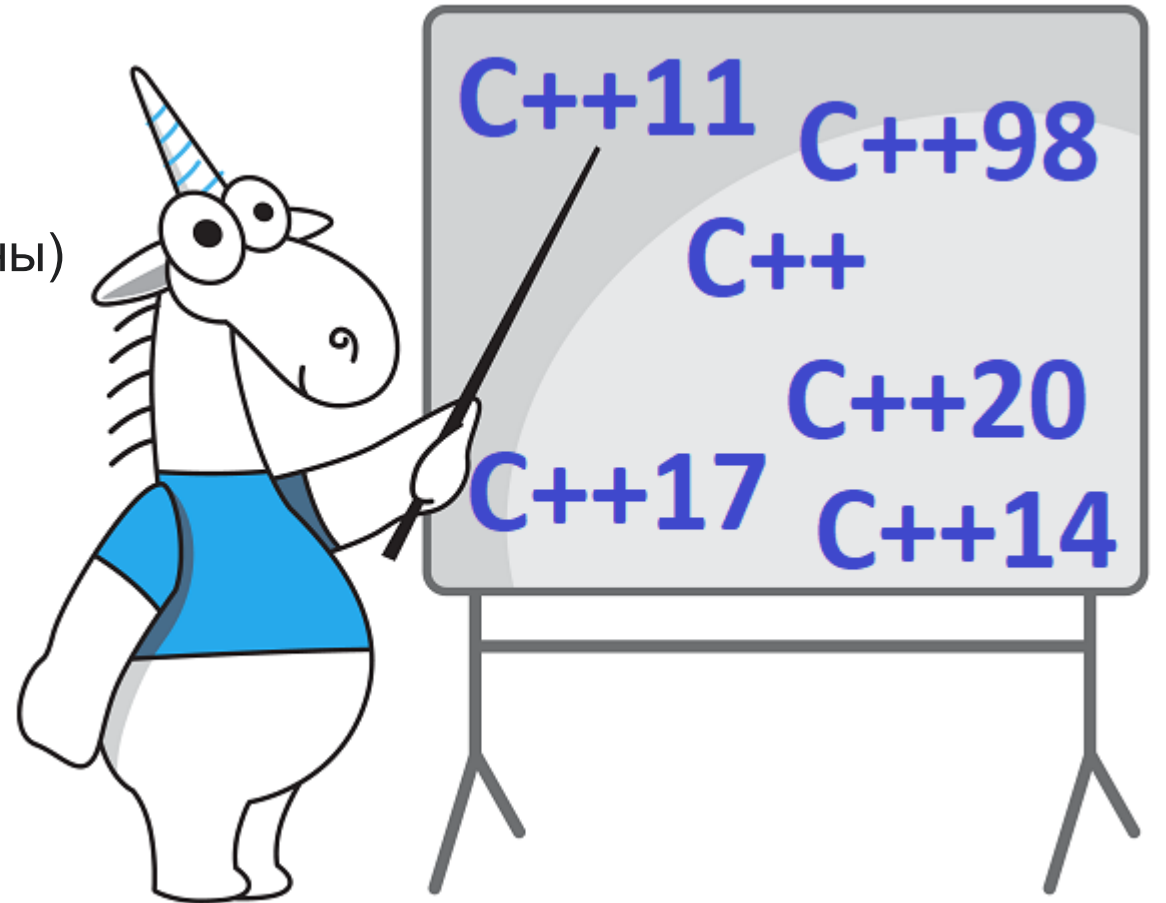
- процедурное программирование,
- объектно-ориентированное программирование,
- обобщённое программирование.

Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности.

C++ сочетает свойства как **высокоуровневых**, так и **низкоуровневых языков**.

Основные вехи развития C++

1. 1972 — создание языка C (Брайан Керниган, Деннис Ритчи)
2. 1983 — создание языка C++ (Бьярн Страуструп)
3. 1998 — принят первый стандарт языка C++, множественное/виртуальное наследование
4. 1991 — обобщённое программирование (шаблоны)
5. 2003 — следующая версия стандарта языка C++
6. 2011 — C++11
7. 2014 — C++14
8. 2017 — C++17
9. 2020 — C++20



Философия C++

- Эффективность (в ущерб безопасности: например, нет контроля выхода за пределы массива)
- Переносимость
- Мультипарадигмальность

Принципы Бьёрна Страуструпа

В книге «Дизайн и эволюция C++» (2007) Бьёрн Страуструп описывает принципы, которых он придерживался при проектировании C++ (приводятся в сокращении):

- Получить универсальный язык со статическими типами данных, эффективностью и переносимостью языка C
- Непосредственно и всесторонне поддерживать множество стилей программирования
- Дать программисту свободу выбора, даже если это даст ему возможность выбирать неправильно
- Максимально сохранить совместимость с C, тем самым, делая возможным лёгкий переход от программирования на C
- Избежать разночтений между C и C++: любая конструкция, допустимая в обоих языках, должна в каждом из них обозначать одно и то же и приводить к одному и тому же поведению программы
- Избегать особенностей, которые зависят от платформы или не являются универсальными
- «Не платить за то, что не используется» — никакое языковое средство не должно приводить к снижению производительности программ, не использующих его
- Не требовать слишком усложнённой среды программирования

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

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

```
#include <iostream>
// Заголовочный файл
int main() {
    std::cout << "Hello world!" << std::endl;
    return 0;
}
```

```
#include <iostream>
```

```
using namespace std;
```

```
// Всё содержимое пространства std становится видимым во всем файле.
```

```
using std::cout; // Либо только элемент cout становится видимым во всем файле.
```

```
int main() {
    cout << "Hello world!";
    return 0;
}
```

В C++ доступны следующие встроенные типы

int
bool
double, float
char

Символьные: char, wchar_t (char16_t и char32_t, со стандарта C++11)

Целочисленные знаковые: signed char, short int, int, long int (long long, со стандарта C++11)

Целочисленные беззнаковые: unsigned char, unsigned short int, unsigned int, unsigned long int (и unsigned long long, со стандарта C++11)

С плавающей точкой: float, double, long double

Логический: bool, имеющий значения true или false

Способы инициализации переменных

```
double d = 3.14; // старый стиль
```

```
double d {3.14}; // новый стиль
```

```
double d = {3.14}; // или так
```

Неявное преобразование типов

```
int i = 3.14; // i == 3
```

```
int i {3.14}; // Ошибка компиляции
```

Тип `char` занимает 1 байт, поэтому диапазон его значений `[-128,127]`, однако следующий код будет откомпилирован.

```
char c = 128; // c == -128
```


Основные операции

C++

Примеры

=

int a = 4;

/

7 / 3 — тип int; 7 / 3.0 — double

%

a++ / ++a

&&

i > 0 && i != 2

7&&3

||

!

==

!=

<<

(8 << 1 = 16)

>>

8 >> 1 =

&

7 & 3 = 3

|

^

~

Возможна запись $a = b = c$, означающая $b = c$; $a = c$ — множественное присваивание (справа налево)

Особое внимание стоит обратить на операцию сравнение на равенство:

`if (a = 5) ... // (1) Неверно! Но легко пропустить`

`if (5 = a) ... // (2) Неверно! И компилятор выдаст ошибку`

`if (a == 5) ... // (3) Верно! Но если спутать с (1), то легко не заметить`

`if (5 == a) ... // (4) Верно! И если спутать с (2), то легко увидеть: ошибка компиляции`

Условный оператор

```
if (условие) {  
    ...  
}  
else {  
    ...  
}
```

Допускается сокращённая форма (без else)

```
bool flag;  
...  
if (flag) {  
    cout << "true";  
} else {  
    cout << "false";  
}
```

Тернарная условная операция

```
cout << (flag ? "true" : "false");
```

Оператор выбора

```
switch (i) {  
    case 1:  
        ...  
        break;  
    case 2:  
    case 3:  
        ...  
    default:  
}
```

Операция , (запятая)

Возвращает значение своего последнего операнда, например:

```
int s = 0, i = 6;  
cout << (s = i*i, i); // Выведет 6
```

Операторы цикла

Цикл while

```
while (x > 10) {  
    x -= 1;  
}
```

Цикл do-while

```
do {  
    x--;  
} while (x > 0);
```

Цикл for

```
for (инициализация; условие_продолжения; действие_на_каждом_шаге) {  
    ...  
}
```

```
for (int i = 0; i < 10; i++) {  
    cout << i << " ";  
}  
// "0 1 2 3 4 5 6 7 8 9 "
```

```
for (;;) // бесконечный цикл
```

```
int z = int(sqrt((double)x));  
for (int i=2;i<=z; ++i) {  
    int y=x%i;  
    ...  
}
```

```
unsigned char u;  
cin>>u;  
unsigned char t;  
for (t=128; t>0; t=t>>1)  
    if (u&t)  
        cout <<"1";  
    else  
        cout <<"0";
```

Комбинированные или составные операторы присваивания

`s/=a;`

`s+=a%10;`

`r*=10;`

`s=s/a`

`s=s+a%10`

`r=*/10`

Функции

```
int abs(int x) {  
    return x > 0 ? x : -x; // После этого сразу выход из функции  
}
```

```
auto abs(int x) {  
    return x > 0 ? x : -x; // После этого сразу выход из функции  
}
```

// Функция, возвращающая void — аналог процедуры

```
void print(int i) {  
    cout << "Значение = " << i;  
}
```

Стандартные функции

Математические функции объявлены в заголовочном файле <cmath>

```
#include <cmath>
```

```
abs(x); floor(x); ceil(x); round(x); sin(x); pow(x, y); sqrt(x);
```

Накладные расходы на вызов функции

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

Накладные расходы на этапе выполнения.

1. На стек памяти кладется память, называемая записью активации
2. В запись активации копируется адрес возврата и адрес предыдущей записи активации.
3. В запись активации копируются значения фактических параметров вместо формальных параметров.
4. В записи активации оставляется память под возвращаемое значение.

Кроме того, когда возвращаемся из функции мы

1. Переходим по адресу возврата к предыдущей записи активации или переходим по адресу возврата в основную программу.
2. Передаем найденное значение в основную программу.

inline-функции

Для маленьких функций накладные расходы на вызов функции значительно превышают вычисления, производимые внутри функции.

Поэтому такую функцию можно сделать встраиваемой: при компиляции тело будет встроено на место её вызова.

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

```
int a = 3;  
int c = add(a, 5); // int c = a + 5;
```

Слово `inline` является лишь рекомендацией компилятору.

Ссылка — другое имя объекта («псевдоним»). Ссылка всегда инициализируется при объявлении

```
int i = 5;
int &pi = i; // ссылка на i
pi = 3;
cout << i; // i == 3
i = 7;
cout << pi; // pi == 7
```

Передача аргумента в функцию по ссылке

```
void calcSquareAndPerimeter(double a, double b, double &S, double &P) {
    S = a * b;
    P = 2 * (a + b);
}

int main() {
    double SS = 0.0, PP = 0.0;
    calcSquareAndPerimeter(3.0, 4.0, SS, PP);
    cout << "Perimeter: " << PP << ", Square: " << SS << "\n";
}
```

Структуры

```
struct Student {  
    string name;  
    int age;  
}; // точка с запятой!
```

```
int main() {  
    Student s {"Иванов", 19}; // Инициализация  
    cout << s.name << ' ' << s.age << endl;  
  
    Student s2 = s; // Независимая копия s  
    s2.name = "Петров";  
    cout << s.name; // "Иванов"  
}
```

Передача структуры в функцию

Желательно передавать структуру в функцию по ссылке, иначе будет передана её полная копия;
const запрещает изменение полей структуры

```
void print(const Student &s) {  
    cout << s.name << ' ' << s.age << endl;  
} 2023
```

Перегрузка функции

Объявление двух функций с одинаковыми именами означает, что имеет место перегрузка имени функции.

Перегрузка функции — возможность определить несколько функций с одним и тем же именем, если эти функции имеют разные наборы параметров (по меньшей мере, разные типы параметров).

Перегруженные функции обязательно должны иметь разные **сигнатуры**, т. е. должны отличаться своими списками спецификации параметров.

Следует помнить, что тип возвращаемого значения не участвует в формировании компилятором сигнатуры (внутреннего имени) функции.

Поэтому в качестве перегружаемых функций нельзя использовать функции, различающиеся лишь типом возвращаемых значений.

Аргументы функции по умолчанию

Аргументы по умолчанию могут быть расположены только в конце списка формальных параметров.

Функция, вычисляющая расстояние между двумя точками на плоскости.

```
double dist(double x1, double y1=0, double x2=0, double y2=0);
```

```
double dist(double x1, double y1, double x2, double y2) {  
    return sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));  
}
```

```
cout<<dist(x1,y1)+dist(x2,y2)+dist(x1,y1,x2,y2)<<endl;
```

```
cout<<dist(x1)+dist(0,y2)+dist(x1,0,0,y2)<<endl;
```

Объявления и определения

Объявление сообщает компилятору некоторое имя (идентификатор).

Определение сообщает компилятору о необходимости создания в памяти объекта с указанным именем.

Это относится как к переменным, так и к функциям.

Объявления и определения

Говорят, что переменная описана (определена), если она объявлена и под неё выделена память. Рассмотрим следующий пример, пусть в проекте есть два файла:

```
/* a.cpp */  
extern int n; // Где-то дальше определено  
void f(int); // Предварительное объявление
```

```
void main() {  
    n = 5; f(n);  
}
```

```
/* b.cpp */  
#include <iostream>  
int n; // Определение переменной
```

```
// Определение функции  
void f(int i) {  
    std::cout << i;  
}
```