

Язык программирования C++. Введение.

М. А. Ложников

5 сентября 2019 г.

Ревизия: 3

Это предварительная невыверенная версия. Читайте на свой страх и риск.

1 Введение

В настоящее время язык программирования C++ кардинально отличается от языка программирования C. Правильнее считать, что это два принципиально разных языка. Тем не менее, язык C++ включает в себя многие парадигмы языка C.

Не смотря на то, что часть синтаксиса языка C++ похожа на синтаксис C, язык программирования C++ не является в строгом смысле надстройкой над языком C, то есть не любая программа, написанная на чистом C будет компилироваться компилятором C++. Во многом это связано с более широким списком ключевых слов, а также с более строгой типизацией. Тем не менее, в большинстве случаев, программы, написанные на C, компилируются компилятором C++ (возможно, с небольшими правками).

На данном этапе мы будем постепенно знакомиться с возможностями языка C++.

2 Пространства имён

Предположим, что в программе используются две различные библиотеки, каждая из которых предоставляет функцию `f()`. В таком случае эти библиотеки будет затруднительно использовать в одном файле исходного кода из-за конфликта имён. В C++ эту проблему предлагается решать при помощи пространств имён.

```
1 namespace foo {
2   void f() {
3     // реализация функции f() из пространства имён foo
4   }
5 }
6
7 namespace bar {
8   void f() {
9     // реализация функции f() из пространства имён bar
10  }
11 }
12
13 /* Без пространств имён был бы конфликт имён из-за повторного определения
14    функции f(). */
15
```

```
16 void g() {  
17     foo::f(); // вызов функции f() из пространства имён foo  
18     bar::f(); // вызов функции f() из пространства имён bar  
19 }
```

Таким образом, код (определения и реализации функций, константы) каждой библиотеки должен находиться в своём собственном пространстве имён.

2.1 Пример использования пространств имён

Listing 1: Файл foo.hpp

```
1 namespace foo {  
2     // Определение функции f()  
3     void f();  
4 }
```

Listing 2: Файл foo.cpp

```
1 namespace foo {  
2     // Реализация функции f()  
3     void f() {  
4     }  
5 }
```

Listing 3: Файл bar.hpp

```
1 namespace bar {  
2     // Определение функции f()  
3     void f();  
4 }
```

Listing 4: Файл bar.cpp

```
1 namespace bar {  
2     // Реализация функции f()  
3     void f() {  
4     }  
5 }
```

Listing 5: Пример использования пространств имён

```
1 #include <foo.hpp>  
2 #include <bar.hpp>
```

```
3
4 void g() {
5     foo::f(); // вызов функции f() из пространства имён foo
6     bar::f(); // вызов функции f() из пространства имён bar
7 }
```

2.2 Пространство имён std

В заголовочных файлах языка C++ все стандартные функции вынесены в пространство имён `std`. Рассмотрим два примера.

```
1 #include <stdio.h>
2
3 /* В C++ доступны все стандартные функции языка C. Этот код компилируется
4    компилятором C++ и работает. */
5 int main() {
6     printf("Hello world! \n");
7     return 0;
8 }
```

Предыдущий пример компилируется и работает, однако, в C++ лучше не использовать заголовочные файлы языка C напрямую. Вместо `stdio.h` следует использовать `cstdio`, вместо `stdlib.h` — `cstdlib`, вместо `math.h` — `cmath` и т.д. Дело в том, что в новых файлах все функции вынесены в пространство имён `std`. Кроме того, в заголовочных файлах языка C++ некоторые функции могут работать с аргументами различных типов данных. Например, функция `std::fabs()` из `<cmath>` может работать с любыми целочисленными и вещественными аргументами, в то время как функция `fabs()` из `<math.h>` принимает только аргументы типа `double` (подстановка целочисленного аргумента приведёт к конвертации типов).

```
1 #include <cstdio>
2
3 /* В заголовочном файле "cstdio" функции вынесены в пространство имён std. */
4 int main() {
5     std::printf("Hello world! \n");
6     return 0;
7 }
```

2.3 Директива using namespace

В тех случаях, когда лень писать `std::` можно использовать директиву `using namespace`.

```
1 #include <cstdio>
2
3 using namespace std;
4
5 /* Теперь всё, что объявлено в пространстве имён std доступно в корневом
6    пространстве имён. */
```

```
7 int main() {
8     printf("Hello world! \n");
9     return 0;
10 }
```

Никогда не следует использовать директиву `using namespace` в заголовочных файлах, поскольку она подставится при включении соответствующего заголовочного файла.

3 Ввод/вывод

Не смотря на то, что в C++ работают средства ввода/вывода языка C, в C++ есть свои собственные средства ввода/вывода.

3.1 Поточковый вывод

```
1 /* Заголовочный файл <iostream> используется для подключения средств ввода/вывода. */
2 #include <iostream>
3
4 int main() {
5     /* Для записи в стандартный поток вывода используется специальная переменная cout,
6        которая является объектом класса std::ostream (о классах на следующих занятиях).
7        Манипулятор endl служит для перевода строки, вместо него можно использовать "\n".
8        Манипулятор endl отличается от "\n" тем, что сбрасывает содержимое внутренней
9        памяти в поток вывода. */
10    std::cout << "Hello world" << std::endl;
11
12    int a = 2;
13    double b = 3.14;
14
15    /* Пример вывода переменных. */
16    std::cout << "a = " << a
17              << " b = " << b << std::endl;
18    return 0;
19 }
```

3.2 Поточковый ввод

```
1 #include <iostream>
2 /* В заголовочном файле <string> описан класс std::string для работы со строками. */
3 #include <string>
4
5 int main() {
6     int a;
7     double b;
8     // Переменная s является объектом класса std::string, то есть имеет тип std::string.
9     std::string s;
10
11    /* Для чтения из стандартного потока ввода используется специальная переменная
```

```
12     std::cin, которая является объектом класса std::istream. Обратите внимание,
13     что стрелки направлены в другую сторону. */
14     std::cin >> a;
15
16     /* Любой поток можно неявным образом преобразовывать к типу bool. В языке C++
17     переменная типа bool может принимать одно из двух значений: true (true = 1,
18     истина) или false (false = 0, ложь). Если поток приводится к типу bool
19     со значением false, значит считывание не удалось.
20     */
21     if (!std::cin) {
22         std::cout << "Can't read a!" << std::endl;
23     }
24
25     /* Оператор >>, применённый к потоку, возвращает сам поток, поэтому конструкцию можно
26     упростить. */
27     if (!(std::cin >> b)) {
28         std::cout << "Can't read b!" << std::endl;
29     }
30
31     /* Также можно читать строки. */
32     if (!(std::cin >> s)) {
33         std::cout << "Can't read s!" << std::endl;
34     }
35     return 0;
36 }
```

3.3 Поток cerr для вывода ошибок

Для вывода ошибок как правило используется специальный поток вывода `std::cerr`.

3.3.1 Пример 1

```
1 #include <iostream>
2
3 int main() {
4     int n;
5
6     if (!(std::cin >> n)) {
7         std::cerr << "Can't read n!" << std::endl;
8         return 1;
9     }
10
11     if (n < 0) {
12         std::cerr << "n < 0!" << std::endl;
13         return 2;
14     }
15
16     if (n == 0) {
17         // Выводим результат в стандартный поток вывода.
18         std::cout << "Result: 0" << std::endl;
```

```
19     return 0;
20 }
21
22 /* В языке C++ разрешено перемешивать объявления переменных и код. В некоторых
23 случаях переменные следует объявлять по мере необходимости, а не в начале блока. */
24 double sum = 0;
25
26 /* Внутри цикла for можно объявлять переменные-счётчики. Рекомендуется именно так и
27 делать, чтобы не плодить лишние сущности. */
28 for (int i = 0; i < n; i++) {
29     double number;
30
31     if (!(std::cin >> number)) {
32         std::cerr << "Can't read the next number!" << std::endl;
33         return 1;
34     }
35
36     sum += number;
37 }
38
39 sum /= n;
40
41 std::cout << "Result: " << sum << std::endl;
42 return 0;
43 }
```

3.3.2 Пример 2

```
1 #include <iostream>
2
3 int main() {
4     int n = 0;
5     double sum = 0;
6     double number;
7     /* Эта конструкция будет считывать действительные числа из стандартного потока ввода
8     до тех пор, пока они считываются. */
9     while (std::cin >> number) {
10        sum += number;
11        n++;
12    }
13
14    if (n > 0)
15        sum /= n;
16
17    std::cout << "Result: " << sum << std::endl;
18    return 0;
19 }
```

4 Ключевое слово const

В языке C++ можно явно запретить менять значения переменных. Это делается для того, чтобы избежать ошибок, связанных со случайным изменением переменной. Попытка изменить значение такой переменной приводит к ошибке компиляции.

```
1 const int a = 5;
2
3 // a = 10; // Ошибка на этапе компиляции.
4
5 /* ptr является указателем на элементы типа const int. Это означает, что элементы
6 массива нельзя менять, однако само значение указателя может быть изменено. */
7 const int* ptr;
8
9 // Сам указатель менять можно
10 ptr = (const int*) malloc(sizeof(int));
11
12 // ptr[0] = 10; // Ошибка на этапе компиляции.
13
14 /* Ошибка компиляции, функция free() принимает указатель на void (без const), значит,
15 теоретически она может менять содержимое массива, а мы передаём указатель на
16 константу. */
17 free(ptr);
18
19 int* const constPtr = (int* const) malloc(sizeof(int));
20
21 constPtr[0] = 10; // Корректно
22
23 // constPtr = NULL; // Ошибка на этапе компиляции
24
25 /* Всё в порядке */
26 free(constPtr);
27
28 const int* const constPtrToConst = (const int* const) malloc(sizeof(int));
29
30 // constPtrToConst[0] = 10; // Ошибка на этапе компиляции
31 // constPtrToConst = NULL; // Ошибка на этапе компиляции
32
33 /* Ошибка компиляции, функция free() принимает указатель на void (без const), значит,
34 теоретически она может менять содержимое массива, а мы передаём указатель на
35 константу. */
36 free(constPtrToConst);
```

В случае с указателями лучше читать написанное справа налево.

```
1 /* Синтаксис: (const) ТИП* (const) ИМЯ;
2 ИМЯ есть (константный) указатель на (константный) ТИП (соответствие в обратном
3 порядке). */
```

4.1 Преобразования типов

Указатели на неконстантный тип могут быть неявно преобразованы к указателям на константный тип. Обратное преобразование приводит к ошибке компиляции (ошибка нейтрализуется при помощи `const_cast`, но об этом позже).

5 Lvalue ссылки

В C++ для каждой переменной можно создать ссылку, которая будет как бы вторым именем для той же самой переменной. Ссылки могут быть любого типа. Разберём на примере типа `int`.

```
1 int a = 5;
2 /* Переменная refToA является ссылкой на переменную a. Ссылка обязательно должна
3    быть инициализирована некоторой переменной того же типа (или другой такой же
4    ссылкой) сразу при объявлении. */
5 int& refToA = a;
6
7 /* Синтаксис: ТИП& ИМЯ ССЫЛКИ = ИМЯ ПЕРЕМЕННОЙ; */
8
9 /* Если поменять значение ссылки, то значение переменной, на которую она ссылается
10    также поменяется. */
11 refToA = 10;
12 std::cout << "a = " << a << std::endl; // Выведет 10.
13
14 /* Ссылки бывают константными. Значения, на которые они указывают нельзя поменять
15    напрямую (запрещено компилятором). */
16 const int& constRefToA = a;
17
18 /* Неконстантная ссылка не может указывать на константу. */
19 const int b = 5;
20 const int& constRefToB = b; // Корректно.
21 // int& refToB = b; // Ошибка компиляции.
22
23 /* Ссылки можно инициализировать другими ссылками, однако неконстантную ссылку нельзя
24    инициализировать константной ссылкой, это приведёт к ошибке компиляции (ошибку
25    можно нейтрализовать при помощи const_cast). */
26 int& anotherRefToA = refToA;
27 // int& refToConstRef = constRefToA; // Ошибка компиляции.
```

5.1 Передача аргументов в функцию по ссылке

В языке C++ при передаче в функцию параметры копируются. Для того, чтобы поменять аргумент изнутри функции, можно передать его по ссылке.

```
1 #include <iostream>
2
3 void foo(int a) {
4     a = 10; // Снаружи значение переданного аргумента не поменяется.
5 }
```



```
6
7 void bar(int& b) {
8     b = 20; // А теперь поменяется.
9 }
10
11 int main() {
12     int c = 0;
13
14     foo(c);
15
16     std::cout << "c = " << c << std::endl; // Выведет 0.
17
18     bar(c);
19
20     std::cout << "c = " << c << std::endl; // Выведет 20.
21     return 0;
22 }
```

В тех случаях, когда в функцию передаётся аргумент сложного типа, то есть он долго копируется, например, строка типа `std::string`, однако этот аргумент не должен меняться снаружи, его следует передавать по константной ссылке.

```
1 #include <iostream>
2 #include <string>
3
4 void PrintLine(const std::string& str) {
5     std::cout << str << std::endl;
6 }
7
8 /* Если размер типа данных передаваемого аргумента превышает 16 байт, то его имеет смысл
9    передавать по ссылке. Для типа int это конечно же не нужно, однако как то так это
10   может выглядеть. */
11 int Sum(const int& left, const int& right) {
12     return left + right;
13 }
```

6 Выделение/освобождение памяти

Не смотря на то, что в C++ можно выделять/освобождать память при помощи функций `std::malloc()/std::free()` из `<cstdlib>`, этого делать не рекомендуется. Вместо этого следует использовать операторы `new[]/delete[]`.

```
1 /* Выделение памяти для массива элементов типа int. Память выделяется под 10 элементов.
2    Обратите внимание, что здесь указывается количество элементов, а не байт. */
3 int* arr = new int[10];
4
5 /* Синтаксис: ТИП* ПЕРЕМЕННАЯ = new ТИП[кол-во элементов]; */
6
```

```
7 for (int i = 0; i < 10; i++)
8   arr[i] = i * i;
9
10 /* Оператор delete[] освобождает память, выделенную оператором new[]. */
11 delete[] arr;
12
13 /* Синтаксис: delete[] ПЕРЕМЕННАЯ; */
```

6.1 Пример: сумма элементов массива

```
1 #include <iostream>
2
3 bool ReadArray(int* arr, int n) {
4   for (int i = 0; i < n; i++) {
5     if (std::cin >> arr[i])
6       return false;
7   }
8
9   return true;
10 }
11
12 /* Передача массива в функцию по указателю позволяет менять его содержимое внутри
13 функции. В случае если содержимое массива меняться не должно, сама возможность может
14 потенциально привести к ошибке. Поэтому считается хорошим тоном саму эту возможность
15 запретить. Это делается при помощи ключевого слова const. В данном случае arr
16 является указателем на элементы типа const int, то есть элементы массива не могут
17 меняться. */
18 int Sum(const int* arr, int n) {
19   int result = 0;
20
21   for (int i = 0; i < n; i++)
22     result += arr[i];
23
24   return result;
25 }
26
27 int main() {
28   int n;
29
30   if (!(std::cin >> n)) {
31     std::cerr << "Can't read n!" << std::endl;
32     return 1;
33   }
34
35   if (n < 0) {
36     std::cerr << "n < 0!" << std::endl;
37     return 2;
38   }
39
```

```
40 if (n == 0) {
41     std::cout << "Result: 0" << std::endl;
42     return 0;
43 }
44
45 /* Оператор new[] генерирует исключение (см. следующие занятия) в случае ошибки
46 выделения памяти. Проверку возвращаемого значения делать не нужно. */
47 int* arr = new int[n];
48
49 /* Тип данных int* можно неявно преобразовать к const int*. Всё в порядке. */
50 if (!ReadArray(arr, n)) {
51     std::cerr << "Can't read array!" << std::endl;
52     return 4;
53 }
54
55 std::cout << "Result: " << Sum(arr, n) << std::endl;
56
57 delete[] arr;
58
59 return 0;
60 }
```

Замечание 6.1 Обратите внимание, что в примере выше нет проверки того, что память выделялась. Это связано с тем, что оператор `new[]` использует механизм исключений для обработки ошибок, о котором пойдёт речь на следующих занятиях. В данном примере программа вылетит в момент вызова оператора `new[]` в случае ошибки выделения памяти. Поэтому нет смысла проверять возвращаемое значение оператора `new[]`.

7 Список задач

Задача 1 (Источник: <https://leetcode.com/problems/is-subsequence/>) Даны две последовательности целых чисел заданной длины: $\{A_i, i = 1, \dots, N\}$ и $\{B_j, j = 1, \dots, M\}$. Проверить, является ли последовательность B подпоследовательностью последовательности A .

Время работы программы: $O(N + M)$.

Задача 2 (Источник: <https://leetcode.com/problems/two-city-scheduling/>). Предположим, что в некоторой конторе работает $2N$ человек, каждого из которых необходимо отправить в командировку в один из двух городов: A или B . Программе на вход поступает число N и два вещественных массива длины $2N$: `costFlyToA` и `costFlyToB` таким образом, что элементы `costFlyToA[i]` и `costFlyToB[i]` соответствуют стоимости отправки человека с номером i в города A и B соответственно. Программа должна вывести минимально возможную стоимость отправки сотрудников таким образом, что каждый сотрудник должен поехать ровно в один из городов A или B , и в каждом городе должно побывать ровно N человек.

Время работы программы: $O(N \log N)$.

Задача 3 (Источник: <https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/>) Дана последовательность $\{a_i, i = 1, \dots, N\}$. В элементе последовательности a_i записана стоимость некоторого товара в i -ый день. Разрешается продавать и покупать товар в любой день, однако нельзя купить товар, если он уже приобретён, для покупки его нужно сначала продать. Требуется посчитать наибольшую возможную прибыль.

Время работы программы: $O(N)$.

Задача 4 (Источник: <https://leetcode.com/problems/majority-element/>) Дан непустой массив длины N . Найдите элемент, встречающийся более $N/2$ раз. Считайте, что такой элемент существует.

Время работы программы: $O(N)$.

Задача 5 (Источник: <https://leetcode.com/problems/k-closest-points-to-origin/>) Дано N точек на плоскости. Выведите K ближайших к началу координат точек (в любом порядке). Расстояние между точками определяется евклидовой метрикой.

Время работы программы: в среднем $O(N)$.

Задача 6 (Источник: <https://leetcode.com/problems/sort-array-by-parity-ii/>) Дан массив длины $2N$, половина элементов которого чётные, а вторая половина — нечётные. Отсортируйте массив таким образом, чтобы на месте чётных индексов стояли чётные элементы, а на месте нечётных — нечётные. Ответом является любой массив, удовлетворяющий требуемому условию.

Время работы программы: $O(N)$.

Задача 7 (Источник: <https://leetcode.com/problems/sort-colors/>) Дан массив длины N , элементами массива являются числа 0, 1 или 2. Отсортируйте массив таким образом, чтобы в начале располагались все нули, затем все единицы и, наконец, в конце массива лежали все двойки.

Время работы программы: $O(N)$, задача решается за один проход по массиву.

Список литературы

- [1] Бьерн Страуструп Язык программирования C++. М:Бином. 2011.
- [2] <https://en.cppreference.com>
- [3] <https://en.cppreference.com/w/cpp/language/namespace>
- [4] https://en.cppreference.com/w/cpp/language/namespace_alias
- [5] <https://en.cppreference.com/w/cpp/io/cin>
- [6] <https://en.cppreference.com/w/cpp/io/cout>
- [7] https://en.cppreference.com/w/cpp/language/constant_expression
- [8] <https://en.cppreference.com/w/cpp/language/cv>
- [9] https://en.cppreference.com/w/cpp/language/const_cast