Числа и строки — конспект темы

Hello, friend

Для записи и чтения символов в C++ используют потоки. Стандартные потоки:

- {{`cin`}}[be_translate_cpp_cin] — входной поток, получает данные с клавиатуры.
- {{`cout`}}[be_translate_cpp_cout] — выходной поток, выводит данные на экран.
- {{`cerr`}}[be_translate_cpp_cerr] — поток вывода сообщений об ошибках.

Чтобы вывести на экран фразу “Hello, friend!”, нужно написать такой код:

```
#include <iostream>

using namespace std;

int main() {
    cout << "Hello, friend!" << endl;
}
```

Сначала подключается библиотека `iostream` — в ней определены стандартные потоки ввода и вывода:

```
#include <iostream>
```

Затем стандартное пространство имён:

```
using namespace std;
```

Затем функция `main`. Работа любой программы на C++ начинается с функции `main`, так как код не может находиться вне функции:

```
int main() {
    cout << "Hello, friend!" << endl;
```

```
}
```

Команды, принадлежащие функции, пишутся между фигурных скобок.

Символ `;` обязательно ставится в конце команды.

Текст, который нужно вывести на экран, пишется в двойных кавычках.

Операция `<<` выводит данные в поток.

Специальный объект `endl` переносит строку. “Hello, friend!” — всего одна строка, но использовать `endl` считается хорошим тоном.

В коде можно оставить комментарии. Они не влияют на работу программы и пишутся после знака `//` или между знаками `/*` и `*/`:

```
#include <iostream>

using namespace std;

/*
    Здесь начинается функция main.
    Работа любой программы начинается с запуска этой функции.
*/
int main() {
    cout << "Hello, friend!" << endl; // печатаем на экране и переводим строку
}
```

Целые числа

Чтобы вывести на экран целое число 42, нужно написать:

```
#include <iostream>

using namespace std;

int main() {
    cout << 42 << endl;
}
```

42 — **целочисленный литерал**, то есть целое число, записанное цифрами в коде программы. Бывают литералы других типов. Например, `"Hello, friend!"` — **строковый литерал**, который задаёт строку `Hello, friend!`.

Чтобы объявить числовую переменную, нужно указать тип данных `int`. Он представляет целое число:

```
#include <iostream>

using namespace std;

int main() {
    int x = 42;
    cout << x << endl;
}
```

Тип переменной задают при её объявлении и не меняют. Такая программа не скомпилируется, так как переменной целочисленного типа попытались присвоить значение строки:

```
#include <iostream>

using namespace std;

int main() {
    int x = 0;
    x = "42"; // ошибка!
}
```

Переменные типа `int` хранят в себе положительные и отрицательные числа. Переменные объявляются так:

```
#include <iostream>

using namespace std;

int main() {
    int a; // Неинициализированная значением переменная. Её значение будет случайным число
м.
    int b = 1; // идеальный вариант
    int c1 = 2, c2 = -3; // длинные списки сложно читать, лучше объявлять по одной перемен
ной в каждой строке
}
```

Над переменными и литералами можно выполнять арифметические действия:

```
#include <iostream>

using namespace std;

int main() {
    int x = 1 + 2;
    int y = x + 3;
    cout << x << endl // 3
         << y << endl // 6
         << x * 10 << endl // 30
         << y / 4 << endl; // 1
}
```

На экран будет выведено:

```
3
6
30
1
```

В C++ результат деления двух целых чисел — всегда целое число. Дробная часть отбрасывается.

Так объявленным переменным можно присвоить новые значения:

```
#include <iostream>

using namespace std;

int main() {
    int x = 2;
    int y = 3;
    x = y + 5; // значение x теперь равно 8
    x += 2; // значение x теперь равно 10
}
```

Чтобы программа принимала данные пользователя (о его количестве лет в разработке) и выводила их на экран, нужно написать такой код:

```
#include <iostream>

using namespace std;
```

```
int main() {
    cout << "How experienced are you in C++? Answer in years: ";
    int years_in_programming;
    cin >> years_in_programming;
    cout << "You are " << years_in_programming << " years in development" << endl;
}
```

Вещественные числа

Вещественные числа представляет тип данных `double`:

```
#include <iostream>

using namespace std;

int main() {
    double pi = 3.14;
}
```

Арифметические операции над вещественными числами производятся так же, как над целыми:

```
#include <iostream>

using namespace std;

int main() {
    double x = 1;
    double y = 3;
    cout << x + y << endl // 4
         << x - y << endl // -2
         << x * y << endl // 3
         << x / y << endl; // 0.333333
}
```

C++ умеет неявно преобразовывать численные типы друг к другу:

```
#include <iostream>

using namespace std;

int main() {
    int x = 42;
```

```
    double y = x; // 42.0
    double z = -(x + y + 0.8); // -84.8
    int w = z; // -84, округление в сторону нуля, дробная часть отбрасывается
}
```

Если нужен именно литерал типа `double`, пишут не 1, а 1.0.

Преобразовать `double` к `int` можно явно. Для этого используют операцию `static_cast`:

```
#include <iostream>

using namespace std;

int main() {
    int x;
    cin >> x;
    // результат выражения (x + 0.5) будет неявно приведён к double,
    // затем static_cast<int>(...) явно приведёт его к int — отбросит дробную часть
    cout << static_cast<int>(x + 0.5) << endl;
}
```

Явное приведение `int` к `double` нужно редко. Но способы есть:

- операция `static_cast<double>(/*выражение*/)`. Она попытается преобразовать выражение из скобок к типу `double`. Для целого числа такое преобразование существует, и изменение типа пройдёт успешно. Если выражение было бы, например, строковым литералом, возникла бы ошибка. `static_cast` не умеет преобразовывать строки в числа.
- прибавление к числу 0.0 или умножение его на 1.0.

Ошибки компиляции

Ошибки компиляции проявляются, когда компилятор анализирует ваш код. Если их исправить, компиляция пройдёт успешно.

Распространённые ошибки компиляции:

- Пропущена директива `#include`, которая подключает нужную библиотеку.
- Неправильно написано имя библиотеки в директиве `#include`.
- Нет строчки `using namespace std`.

- Одинарные, а не двойные кавычки для строковых литералов.
- Нет `return` в функции, возвращающей значение.
- Имя функции написано с ошибкой (`Main` вместо `main`).
- Инструкции находятся за пределами функций.
- Неверное считывание переменных оператором `>>`.
- Орфографическая ошибка в названии идентификато. а или опечатка из-за неверной раскладки клавиатуры.

Компилятор выводит сообщения об ошибках. В этих сообщениях говорится, в чём проблема и на какой строчке кода её искать.

Строки

Чтобы создать строковую переменную, нужно объявить переменную типа `string` и положите в неё значение:

```
string news = "Great news!"s;
// Переменная news содержит текст Great news!
```

Если пропустить букву `s` после кавычек, получится строковый литерал другого типа, унаследованного от C.

Почему инициализация на основе `""s`-литералов предпочтительнее:

- в них могут быть любые символы,
- такая инициализация помогает компилятору сгенерировать более быстрый код.

Код будет выглядеть так:

```
#include <string>

using namespace std;

int main() {
    string news = "Great news!"s;
    return 0;
}
```

Внутри функции `main` инструкция `return 0` завершает работу программы с кодом ошибки, равным нулю. Ненулевой код — признак ошибки.

Если просто объявить строковую переменную, не указав начальное значение, в ней будет содержаться пустая строка:

```
int age;      // В переменной age будет содержаться мусор (неопределённое значение)
string name;  // В переменной name будет пустая строка
```

Строки можно склеить в одну. Для этого используется операция `+`:

```
string great = "Great "s;
string news = "news!"s;
string design = great + news; // Получается "Great news!"
```

Чтобы просто последовательно вывести две строки, склеивать их не нужно. Просто выведите строки одну за другой:

```
cout << great << news;
```

Операция `+=` дописывает содержимое второй строки в конец первой:

```
string great = "Great "s;
string news = "news!"s;
string design;
design += great;
design += news;
// Теперь в переменной design содержится строка "Great news!"
```

Из `cin` строка считывается до первого символа-разделителя: пробела, табуляции или перевода строки. Пробелы между словами не считываются:

```
string name;
string surname;
cout << "Enter your name:"s << endl;
cin >> name >> surname;
cout << "Hello, "s << name << " "s << surname << endl;
```

Если пользователь введёт строку `John Doe`, слово `John` будет прочитано в переменную `name`, а слово `Doe` — в переменную `surname`.

Чтобы считать всю строку, а не только текущее слово, используют функцию `getline`:

```
string full_name, address;
getline(cin, full_name);
getline(cin, address);
cout << "Hello, "s << full_name << " from "s << address << endl;
```

Символы

Строка — произвольная последовательность символов алфавита. В `string` каждый элемент имеет тип `char`.

В C++ тип `char` хранит число, равное коду символа. Для представления символов часто используются символьные литералы, в которых символ заключается в одинарные кавычки:

```
char letter_a = 'A';    // Сразу понятно, что letter_a хранит код символа A
char letter_a_too = 65; // Тоже хранит код символа A, но это не очевидно

char letter_b = letter_a + 1; // Буква B имеет код, следующий за кодом символа A
```

Для доступа к отдельным символам строки определена операция индексации. Чтобы обратиться к символу строки, нужно указать его номер — индекс — внутри квадратных скобок `[]`. Индексация элементов строк и массивов в C++ начинается с нуля. Следом за последним символом строки находится служебный символ с кодом 0.

Оператор индексации можно использовать для изменения символов строки:

```
// В greeting было слово Hello
greeting[0] = 'Y'; // Теперь в greeting содержится  "Yello"
```

Индекс символа должен быть в диапазоне [0; длина строки) — включая 0, но не включая число, равное длине строки.

Операции `+` и `+=` могут приклеивать к строке не только строки, но и одиночные символы:

```
// В greeting было слово Yello
greeting += 'w';   // Внутри greeting строка "Yellow"
cout << greeting << endl;
```

Тип `char` хранит не строку, а код символа. Если сложить два значения типа `char`, произойдёт не конкатенация символов в строку, а сложение кодов символов и получится число:

```
cout << 'A' + 'B' << endl; // Выведет число 131, равное сумме 65 и 66
```

Если нужна конкатенация, создайте сначала строку из одного символа, а затем используйте операцию `+`:

```
// string(1, 'A') создаёт строку, состоящую из одного символа A
cout << string(1, 'A') + 'B' << endl; // Выведет AB

// Подобным образом можно создать строку из нескольких одинаковых символов:
cout << string(3, 'A') + 'B' << endl; // Выведет AAAB
```

В C++ под элемент типа `char` всегда выделяется 1 байт. Этого хватит на букву латинского алфавита.

Нюансы операций над строками

- **Для сложения обычных строковых литералов** (без `""s`) **нельзя использовать `+`.**

Нужно использовать `""s`-литералы, которые напрямую создают объект типа `string`:

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

int main() {
    cout << "I know "s + "C++"s << endl;
    // Следующие варианты также возможны
```

```
    // cout << "I know"s + "C++" << endl;
    // cout << "I know" + "C++"s << endl;
}
```

- **Строки и числа нельзя складывать.**

В C++ преобразование числа в строку и обратно должно выполняться явно. Для этого служат функции:

- {{`stoi`}}[be_translate_cpp_stoi];
- {{`stod`}}[be_translate_cpp_stod];
- {{`to_string`}}[be_translate_to string].

```
int x = stoi(x_str);      // string -> int
double y = stod(x_str);   // string -> double
string s = to_string(x);  // int or double -> string

int z = 42;
string t = "Hello"s + to_string(z); // В t будет значение "Hello42"
```

- **Узнать размер строки можно функцией `size`.**

Синтаксис её вызова:

```
// Метод size вызывается у строки, созданного строковым литералом "Hello"s
int length = "Hello"s.size(); // значение length равно 5

string greeting = "Hi"s;
// Метод size вызывается у строки greeting
int greeting_size = greeting.size(); // значение greeting_size равно 2
greeting = "Bye"s;
greeting_size = greeting.size(); // значение greeting_size
```

Такие функции называются «метод».

Размер строки, которую вернул метод `size`, исчисляется в элементах типа `char`:

```
cout << "Hello"s.size() << endl // 5
     << "Привет"s.size() << endl // 12 — кириллица в UTF-8
     << "👋🏾"s.size() << endl; // 8 — эмодзи
```

Если строка состоит только из латинских букв, цифр, пробела и основных знаков препинания, её размер будет равен её длине.