

ПОТОКОВЫЙ ВВОД/ВЫВОД В C++

М. А. Ложников

11 февраля 2019 г.

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

1 Потокový вывод на экран

```
1 #include <iostream> // В этом заголовочном файле определены необходимые нам классы.
2
3 using namespace std; // Это для того, чтобы не писать всё время std::
4
5 int main() {
6     /* Для вывода на экран в C++ используют объект cout класса std::ostream.
7        Для этих целей в классе std::ostream переопределена операция <<, возвращающая
8        ссылку на объект, к которому она была применена. Эта операция определена
9        для стандартных типов данных. Если нужно вывести собственный тип, то нужно её
10       переопределить вручную. */
11
12     cout << "Pi is equal to " << 3.14 << endl;
13
14     // Манипулятор std::endl служит для перевода строки. Запись выше следует трактовать как
15     ((cout << "Pi is equal to ") << 3.14) << endl;
16     // или, другими словами
17     ((cout.operator<<("Pi is equal to")).operator<<(3.14)).operator<<(endl);
18     // или ещё можно записать так
19     {
20         ostream& res1 = cout << "Pi is equal to ";
21         ostream& res2 = res1 << 3.14;
22         res2 << endl;
23     }
24     /* Работает это так: 1) сначала выполняется операция cout << "Pi is equal to "
25        она вернёт ссылку на сам объект cout. 2) Выполнится операция << с правым
26        аргументом 3.14, а в качестве левого аргумента будет возвращаемое значение
27        от первой операции, то есть ссылка на cout. Иными словами, выполнится cout << 3.14.
28        На третьем этапе выполнится операция << с правым аргументом endl и левым
29        аргументом - возвращаемым значением от второй операции. */
30     return 0;
31 }
```

2 Поточковый ввод с клавиатуры

```
1  #include <iostream> // В этом заголовочном файле определены необходимые нам классы.
2  #include <string>
3
4  using namespace std; // Это для того, чтобы не писать всё время std::
5
6  int main() {
7      /* Для ввода с клавиатуры в C++ используют объект cin класса std::istream.
8      Для этих целей в классе std::istream переопределена операция >>, возвращающая
9      ссылку на объект, к которому она была применена. Эта операция определена
10     для стандартных типов данных. Если нужно вывести собственный тип, то нужно её
11     переопределить вручную. */
12
13     double a;
14     int b;
15     string s;
16     cin >> a >> b >> s;
17
18     // Запись выше следует трактовать как
19     ((cin >> a) >> b) >> s;
20     // или, другими словами
21     operator>>((cin.operator>>(a)).operator>>(b), s);
22     // или ещё можно записать так
23     {
24         istream& res1 = cin >> a;
25         istream& res2 = res1 >> b;
26         res2 >> s;
27     }
28     /* Работает это так: 1) сначала выполняется операция cin >> a
29     она вернёт ссылку на сам объект cin. 2) Выполнится операция >> с правым
30     аргументом b, а в качестве левого аргумента будет возвращаемое значение
31     от первой операции, то есть ссылка на cin. Иными словами, выполнится cin >> b.
32     На третьем этапе выполнится операция >> с правым аргументом s и левым
33     аргументом - возвращаемым значением от второй операции. */
34
35     // Объекты типа std::istream можно неявно приводить к типу bool.
36     // Эта операция возвращает false, если произошла какая либо ошибка.
37     if (!(cin >> a)) {
38         cout << "Can not read a!" << endl;
39     }
40
41     double sum = 0;
42     b = 0;
43     while (cin >> a) { // Так можно считывать до тех пор пока считывается.
44         sum += a;
45         b++;
46     }
47
48     if (b > 0)
```

```
49     sum /= b;
50     return 0;
51 }
```

3 Строковый поток `std::ostringstream`

Помимо стандартного потока вывода `std::cout` — объекта типа `std::ostream` в C++ можно создавать свои собственные потоки вывода, например, поток вывода в строку или в файл. Поток вывода в строку создаётся при помощи типа данных `std::ostringstream`, определённого в файле `<sstream>`. Класс `std::ostringstream` является производным от класса `std::ostream`.

```
1  #include <iostream>
2  #include <sstream>
3
4  using namespace std;
5
6  int main() {
7      ostringstream strm; // Объявляем строковый поток strm
8      int a = 3;
9      double pi = 3.14;
10
11     // С этими потоками работают как с std::cout
12     strm << "a = " << a << " pi = " << pi;
13
14     string writtenStr = strm.str(); // Метод str() позволяет получить записанную строку
15     cout << "written string = '" << writtenStr << "'" << endl;
16     return 0;
17 }
```

4 Строковый поток `std::istringstream`

Для считывания данных из строки можно использовать объекты класса `std::istringstream`, определённого в файле `<sstream>`.

```
1  #include <iostream>
2  #include <sstream>
3
4  using namespace std;
5
6  int main() {
7      string input = "3.14 2.7 42";
8      // Объявляем строковый поток strm. Конструктор принимает строку, откуда
9      // мы будем считывать.
10     istringstream strm(input);
11     double pi, e;
12     int integer;
13 }
```

```
14 // С этими потоками работают как с std::cin
15
16 if (!(strm >> pi >> e >> integer)) {
17     cout << "Can not read!";
18 }
19 return 0;
20 }
```

5 Файловый ввод/вывод

Для потокового ввода и вывода данных из файла используют классы `std::ifstream` и `std::ofstream` соответственно, определённые в заголовочном файле `<fstream>`.

```
1 // Конструктор класса std::ifstream принимает имя входного файла
2 std::ifstream input("input.txt");
3
4 /* Объект std::cerr класса std::ostream отвечает за поток ошибок.
5     Обычно он выводится на экран так же, как и стандартный поток вывода,
6     однако, это разные потоки и их можно развести, например,
7     перенаправив в разные файлы. */
8 if (!input) { // Если не получилось открыть файл
9     std::cerr << "Can't open file!" << std::endl;
10 }
11
12 // Посчитаем сумму чисел в файле
13 double sum = 0;
14 double a;
15 while (input >> a) // Так можно считывать до тех пор пока считывается.
16     sum += a;
17
18 // С выводом всё аналогично. Открываем файл output.txt.
19 std::ofstream output("output.txt");
20 if (!output) { // Если не получилось открыть файл
21     std::cerr << "Can't open file!" << std::endl;
22 }
23 output << "Sum is qual to " << sum << std::endl;
```

Закрывать файлы нет необходимости, они закроются при вызове деструктора, который сработает как только программа выйдет из блока, в котором были объявлены соответствующие объекты.

6 Вывод собственных типов данных

Пусть дана структура, описывающая время:

```
1 struct Time {
2     int hours;
3     int minutes;
```

```
4     int seconds;
5 };
```

Мы хотели бы, чтобы работала конструкция вроде

```
1 Time t = {12, 0, 30}; // 12 часов, 0 минут, 30 секунд.
2 std::cout << "Time: " << t << std::endl;
```

На выходе хотим получить время в формате HH:MM:SS. Для этого нужно перегрузить операцию << для типов std::ostream и Time. Возвращать эта операция должна ссылку на поток для того, чтобы можно было составлять цепочки этих операций.

6.1 Простой способ

```
1 #include <iostream> // Вообще здесь достаточно просто #include <ostream>
2 #include <iomanip> // Для работы манипуляторов std::setw()
3
4 std::ostream& operator<<(std::ostream& out, const Time& t) {
5
6     out << t.hours << ":" << t.minutes << ":" << t.seconds;
7
8     return out; // Нужно, чтобы эти операции можно было записывать цепочками.
9 }
```

6.2 Сложный способ

Отличие от предыдущего способа заключается в том, что числа от 0 до 9 выводятся с дополнительным нулём в разряде десятков.

```
1 #include <iostream> // Вообще здесь достаточно просто #include <ostream>
2 #include <iomanip> // Для работы манипуляторов std::setw()
3
4 std::ostream& operator<<(std::ostream& out, const Time& t) {
5     /* метод fill() класса std::ostream позволяет установить символ, которым
6        будут заполняться пустые промежутки. Размер этих промежутков задаётся
7        с помощью std::setw(). Метод fill() возвращает прежний символ-заполнитель
8        или текущий, если вызвать его без аргументов. */
9
10    // Сохраняем прежний заполнитель. Новый заполнитель - символ '0'.
11    char prev = out.fill('0');
12
13    /* функция std::setw() позволяет установить размер рамки, в которой
14       будет выведено следующее поле. Пустое пространство в рамке будет заполнено
15       текущим символом-заполнителем. Выравнивание можно задать, передавая
16       в поток манипуляторы std::left или std::right. Для работы нужно подключить
17       <iomanip>*/
18
19    out << std::setw(2) << t.hours << ":"
```

```
20     << std::setw(2) << t.minutes << ":"
21     << std::setw(2) << t.seconds;
22
23     out.fill(prev); // Возвращаем предыдущий заполнитель.
24
25     return out; // Нужно, чтобы эти операции можно было записывать цепочками.
26 }
```

Замечание 6.1 Данная операция будет работать с любыми потоками вывода, в том числе с потоками вывода в строку или в файл, поскольку класс `std::ostream` является базовым для всех потоков вывода.

7 Ввод собственных типов данных

Аналогично, можно написать операцию `>>` для ввода нашей структуры `Time`. Левым аргументом для неё будет объект класса `std::istream`.

7.1 Простой способ (без проверок)

```
1 #include <iostream> // Вообще здесь достаточно просто #include <istream>
2
3 std::istream& operator>>(std::istream& in, Time& t) {
4     // Считываем время и пропускаем двоеточия.
5     in >> t.hours;
6     in.ignore(1);
7
8     in >> t.minutes;
9     in.ignore(1);
10
11    in >> t.seconds;
12
13    return in;
14 }
```

7.2 Сложный способ (с проверками)

```
1 #include <iostream> // Вообще здесь достаточно просто #include <istream>
2
3 // Эта функция нам понадобится для того, чтобы пропускать двоеточия.
4 bool IgnoreDelimiter(std::istream& in, char delim) {
5     // Метод peek() позволяет посмотреть следующий символ, не вынимая его из потока.
6     if (in.peek() != delim) {
7         // После установки failbit при приведении объекта in к типу bool будет
8         // возвращаться false. Так мы сообщим, что произошла ошибка.
9         in.setstate(std::ios_base::failbit);
10        return false;
11    }
12 }
```

```
13 // Метод ignore() позволяет проигнорировать несколько следующих символов.
14 in.ignore(1); // Игнорируем следующий символ.
15
16 return true;
17 }
18
19 std::istream& operator>>(std::istream& in, Time& t) {
20 // Считываем время и пропускаем двоеточия.
21 if (!(in >> t.hours))
22     return in;
23 if (!IgnoreDelimiter(in, ':'))
24     return in;
25 if (!(in >> t.minutes))
26     return in;
27 if (!IgnoreDelimiter(in, ':'))
28     return in;
29
30 in >> t.seconds;
31
32 if (t.hours < 0 || t.hours >= 24 || t.minutes < 0 || t.minutes >= 60 ||
33     t.seconds < 0 || t.seconds >= 60) {
34     in.setstate(std::ios_base::failbit); // считали неправильное значение
35 }
36
37 return in;
38 }
```

Замечание 7.1 Данная операция будет работать с любыми потоками ввода, в том числе с потоками ввода из строки или из файла, поскольку класс `std::istream` является базовым для всех потоков ввода.

8 Манипуляторы для вывода чисел с плавающей точкой

```
1 #include <iostream>
2 #include <iomanip> // Нужно для работы манипуляторов
3 #include <cmath>
4
5 int main() {
6     double value = std::sqrt(2.0);
7
8     // Манипулятор std::setprecision() говорит о том, что все выведенные
9     // после него дробные числа будут выводиться с точностью 16 знаков после запятой.
10    std::cout << std::setprecision(16) << value << std::endl;
11
12    // Данный манипулятор достаточно вставить один раз.
13    std::cout << "Again " << value << std::endl;
14
15    // Можно выводить в экспоненциальном формате, вставив манипулятор std::scientific.
16    // Он тоже действует до тех пор, пока его не отменяют.
```

```
17
18 std::cout << std::scientific << value << std::endl;
19
20 // Чтобы вернуться к обычному формату, нужно вставить манипулятор std::fixed.
21 std::cout << std::fixed << value << std::endl;
22 return 0;
23 }
```

9 Способы повышения скорости работы потоков ввода/вывода

- Потоки вывода устроены таким образом, что они сначала накапливают данные в некотором внутреннем буфере и, как только этот буфер заполнится, сбрасывают его на экран/файл/итд. Вывод в поток манипулятора `std::endl` заставляет принудительно сбросить внутренний буфер. Для того, чтобы избежать слишком частого сброса буфера следует использовать `"\n"` вместо `std::endl`. Это может быть полезно при выводе больших массивов данных. Однако, для вывода отладочной информации лучше использовать `std::endl`.
- По умолчанию при чтении из потока `std::cin`, внутренний буфер потока `std::cout` сбрасывается. Иногда бывает полезно отключить эту синхронизацию. Для этого нужно вставить в начало программы строчку `std::cin.tie(NULL);`.
- Кроме того, можно отключить синхронизацию с функциями ввода/вывода языка C. Для этого нужно вставить в самое начало функции `main()` строчку `std::ios::sync_with_stdio(false);`. Если Вы не планируете пользоваться старыми сишными функциями, то это крайне желательно сделать для избавления от накладных расходов по синхронизации внутренних буферов старых сишных потоков и потоков C++. Однако, это нужно делать до первого использования потоков ввода/вывода, в противном случае может не сработать, зависит от реализации стандартной библиотеки.