Потоковый ввод/вывод в C++

М. А. Ложников

11 февраля 2019 г.

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

1 Потоковый вывод на экран

```
#include <iostream> // В этом заголовочном файле определены необходимые нам классы.

using namespace std; // Это для того, чтобы не писать всё время std::

int main() {
  /* Для вывода на экран в C++ используют объект cout класса std::ostream.
     Для этих целей в классе std::ostream переопределена операция <<, возвращающая
     ссылку на объект, к которому она была применена. Эта операция определена
     для стандартных типов данных. Если нужно вывести собственный тип, то нужно её
     переопределить вручную. */

  cout << "Pi is equal to " << 3.14 << endl;

  // Манипулятор std::endl служит для перевода строки. Запись выше следует трактовать как
  ((cout << "Pi is equal to ") << 3.14) << endl;
  // или, другими словами
  ((cout.operator<<("Pi is equal to")).operator<<(3.14)).operator<<(endl);
  // или ещё можно записать так
  {
    ostream& res1 = cout << "Pi is equal to ";
    ostream& res2 = res1 << 3.14;
    res2 << endl;
  }
  /* Работает это так: 1) сначала выполняется операция cout << "Pi is equal to "
     она вернёт ссылку на сам объект cout. 2) Выполнится операция << с правым
     аргументом 3.14, а в качестве левого аргумента будет возвращаемое значение
     от первой операции, то есть ссылка на cout. Иными словами, выполнится cout << 3.14.
     На третьем этапе выполнится  операция << с правым аргументом endl и левым
     аргументом - возвращаемым значением от второй операции. */
  return 0;
}
```

2 Потоковый ввод с клавиатуры

```
#include <iostream> // В этом заголовочном файле определены необходимые нам классы.
#include <string>

using namespace std; // Это для того, чтобы не писать всё время std::

int main() {
  /* Для ввода с клавиатуры в C++ используют объект cin класса std::istream.
  Для этих целей в классе std::istream переопределена операция >>, возвращающая
  ссылку на объект, к которому она была применена. Эта операция определена
  для стандартных типов данных. Если нужно вывести собственный тип, то нужно её
  переопределить вручную. */

  double a;
  int b;
  string s;
  cin >> a >> b >> s;

  // Запись выше следует трактовать как
  ((cin >> a) >> b) >> s;
  // или, другими словами
  operator>>((cin.operator>>(a)).operator>>(b), s);
  // или ещё можно записать так
  {
    istream& res1 = cin >> a;
    istream& res2 = res1 >> b;
    res2 >> s;
  }
  /* Работает это так: 1) сначала выполняется операция cin >> a
  она вернёт ссылку на сам объект cin. 2) Выполнится операция >> с правым
  аргументом b, а в качестве левого аргумента будет возвращаемое значение
  от первой операции, то есть ссылка на cin. Иными словами, выполнится cin >> b.
  На третьем этапе выполнится  операция >> с правым аргументом s и левым
  аргументом - возвращаемым значением от второй операции. */

  // Объекты типа std::istream можно неявно приводить к типу bool.
  // Эта операция возвращает false, если произошла какая либо ошибка.
  if (!(cin >> a)) {
    cout << "Can not read a!" << endl;
  }

  double sum = 0;
  b = 0;
  while (cin >> a) { // Так можно считывать до тех пор пока считывается.
    sum += a;
    b++;
  }

  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
```

```
  // С этими потоками работают как с std::cin

  if (!(strm >> pi >> e >> integer)) {
    cout << "Can not read!";
  }
  return 0;
}
```

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>`.

```
  // Конструктор класса std::ifstream принимает имя входного файла
  std::ifstream input("input.txt");

  /* Объект std::cerr класса std::ostream отвечает за поток ошибок.
     Обычно он выводится на экран так же, как и стандартный поток вывода,
     однако, это разные потоки и их можно разъединить, например,
     перенаправив в разные файлы. */
  if (!input) { // Если не получилось открыть файл
    std::cerr << "Can't open file!" << std::endl;
  }

  // Посчитаем сумму чисел в файле
  double sum = 0;
  double a;
  while (input >> a) // Так можно считывать до тех пор пока считывается.
    sum += a;

  // С выводом всё аналогично. Открываем файл output.txt.
  std::ofstream output("output.txt");
  if (!output) { // Если не получилось открыть файл
    std::cerr << "Can't open file!" << std::endl;
  }
  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:

```
  struct Time {
    int hours;
    int minutes;
```

```
    int seconds;
  };
```

Мы хотели бы, чтобы работала конструкция вроде

```
Time t = {12, 0, 30}; // 12 часов, 0 минут, 30 секунд.
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

```
#include <iostream> // Вообще здесь достаточно просто #include <ostream>
#include <iomanip>  // Для работы манипуляторов std::setw()

std::ostream& operator<<(std::ostream& out, const Time& t) {

  out << t.hours << ":" << t.minutes << ":" << t.seconds;

  return out; // Нужно, чтобы эти операции можно было записывать цепочками.
}
```

6.2 Сложный способ

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

```
#include <iostream> // Вообще здесь достаточно просто #include <ostream>
#include <iomanip>  // Для работы манипуляторов std::setw()

std::ostream& operator<<(std::ostream& out, const Time& t) {
  /* метод fill() класса std::ostream позволяет установить символ, которым
     будут заполняться пустые промежутки. Размер этих промежутков задаётся
     с помощью std::setw(). Метод fill() возвращает прежний символ-заполнитель
     или текущий, если вызвать его без аргументов. */

  // Сохраняем прежний заполнитель. Новый заполнитель - символ '0'.
  char prev = out.fill('0');

  /*  функция std::setw() позволяет установить размер рамки, в которой
      будет выведено следующее поле. Пустое пространство в рамке будет заполнено
      текущим символом-заполнителем. Выравнивание можно задать, передавая
      в поток манипуляторы std::left или std::right. Для работы нужно подключить
      <iomanip>*/

  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 IgnoreDelimeter(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
```

```
  // Метод ignore() позволяет проигнорировать несколько следующих символов.
  in.ignore(1); // Игнорируем следующий символ.

  return true;
}

std::istream& operator>>(std::istream& in, Time& t) {
  // Считываем время и пропускаем двоеточия.
  if (!(in >> t.hours))
    return in;
  if (!IgnoreDelimeter(in, ':'))
    return in;
  if (!(in >> t.minutes))
    return in;
  if (!IgnoreDelimeter(in, ':'))
    return in;

  in >> t.seconds;

  if (t.hours < 0 || t.hours >= 24 || t.minutes < 0 || t.minutes >= 60 ||
      t.seconds < 0 || t.seconds >= 60) {
    in.setstate(std::ios_base::failbit); // считали неправильное значение
  }

  return in;
}
```

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

```
#include <iostream>
#include <iomanip>  // Нужно для работы манипуляторов
#include <cmath>

int main() {
  double value = std::sqrt(2.0);

  // Манипулятор std::setprecision() говорит о том, что все выведенные
  // после него дробные числа будут выводиться с точностью 16 знаков после запятой.
  std::cout << std::setprecision(16) << value << std::endl;

  // Данный манипулятор достаточно вставить один раз.
  std::cout << "Again " << value << std::endl;

  // Можно выводить в экспоненциальном формате, вставив манипулятор std::scientific.
  // Он тоже действует до тех пор, пока его не отменят.
```

```
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.