Глава 8

Введение в систему ввода/вывода в C++

Начиная с первой главы этой книги, создавая свои программы, мы пользовались стилем ввода/вывода C++. Теперь настало время изучить его более подробно. Как и в языке **C, в** C++ имеется развитая, гибкая и достаточно полная система ввода/вывода. Важно понимать, что в C++ по-прежнему поддерживается вся система ввода/вывода C. Кроме этого в C++ включен дополнительный набор объектно-ориентированных подпрограмм ввода/вывода. Главным преимуществом системы ввода/вывода C++ является то, что она может перегружаться для создаваемых вами классов. Это отличие позволяет легко встраивать в систему ввода/вывода C++ новые создаваемые вами типы данных.

Как и в C, в системе объектно-ориентированного ввода/вывода C++ имеется незначительная разница между консольным и файловым вводом/выводом. На самом деле, консольный и файловый ввод/вывод — это просто разный взгляд на один и тот же механизм. В этой главе в примерах используется ввод/вывод на консоль (в данном случае на экран монитора), но представленная информация вполне применима и для ввода/вывода в файл (ввод/вывод в файл более детально исследуется в главе 9).

К моменту написания этой книги использовались две версии библиотеки ввода/вывода C++: старая, основанная на изначальной спецификации C++, и новая, определенная единым международным стандартом Standard C++. С точки зрения программиста для решения подавляющего большинства задач обе эти библиотеки идентичны. Так происходит потому, что новая библиотека ввода/вывода — это по существу просто обновленная и усовершенствованная версия старой библиотеки. Фактически, почти все отличия двух версий скрыты от вас, поскольку касаются не способа использования библиотек, а способа их реализации. Для программиста главное отличие заключается в том, что новая библиотека ввода/вывода C++ имеет несколько дополнительных возможностей и определяет несколько новых типов данных. Таким образом, новая библиотека ввода/вывода — это по существу просто несколько улучшенная старая. Почти все уже написанные для старой библиотеки программы при использовании новой будут компилироваться без каких бы то ни было существенных изменений. Поскольку прежняя библиотека ввода/вывода ныне считается устаревшей, данная книга описывает только новую библиотеку, как это определено стандартом Standard C++. Тем не менее, большая часть информации вполне применима и к старой библиотеке ввода/вывода.

Эта глава охватывает несколько аспектов системы ввода/вывода C++, включая форматируемый ввод/вывод, манипуляторы ввода/вывода и создание пользовательских функций ввода/вывода. Как вы увидите в дальнейшем, в системе ввода/вывода C++ имеется множество черт, характерных для системы ввода/вывода C++.

8.1. Некоторые базовые положения системы ввода/вывода C++.

Перед тем как начать обсуждение системы ввода/вывода C++, несколько общих комментариев. Система ввода/вывода C++, так же, как система ввода/вывода C, действует через *потоки (streams).* Поскольку вы программировали на C, вы уже должны знать, что такое поток ввода/вывода, однако несколько дополнительных замечаний обобщат ваши знания. Поток ввода/вывода - это логическое устройство, которое выдает и принимает

пользовательскую информацию. Поток связан с физическим устройством с помощью системы ввода/вывода C++. Поскольку все потоки ввода/вывода действуют одинаково, то, несмотря на то, что программисту приходится работать с совершенно разными по характеристикам устройствами, система ввода/вывода предоставляет для этого единый удобный интерфейс. Например, функция, которая используется для записи информации на экран монитора, вполне подойдет как для записи в файл, так и для вывода на принтер.

Как вы знаете, если программа на C начинает выполняться, открываются три потока: **stdin, stdout и stderr.** -Нечто похожее имеет место при запуске программ на C++. Когда запускается программа на C++, автоматически открываются четыре потока:

Поток	Значение	Устройство по умолчанию
cin	Стандартный ввод	Клавиатура
cout	Стандартный вывод	Экран
cerr	Стандартная ошибка	Экран
clog	Буферизуемая версия cerr	Экран

Как вы, наверное, уже догадались, потоки **cin, cout и cerr** соответствуют потокам **stdin, stdout и stderr** языка C. Потоками **cin** и **cout** вы уже пользовались. Поток **clog** — это просто буферизуемая версия потока **cerr.** В языке Standard C++ также открываются дополнительные потоки **wcin, wcout, wcerr** и wclog, предназначенные для широких (16-разрядных) символов, которые в данной книге не рассматриваются. Эти потоки обеспечивают передачу расширенных наборов символов (large character sets), что обеспечивает возможность работы с некоторыми экзотическими языками, такими как, например, китайский.

По умолчанию, стандартные потоки используются для связи с клавиатурой и экраном. Однако в среде, в которой поддерживается переопределение ввода/вывода, эти потоки могут быть перенаправлены на другие устройства.

Как отмечалось в главе 1, в C++ ввод/вывод обеспечивается подключением программе заголовочного файла **<iostream>. В** этом файле определены сложные наборы иерархий классов, поддерживающие операции ввода/вывода. Классы ввода/вывода начинаются с системы *классов-шаблонов (template classes).*

Подробно о классах-шаблонах, называемых также родовыми классами (generic classes), будет рассказано в главе 11, сейчас же ограничимся кратким комментарием. В классе-шаблоне определяется только форма класса без полного задания данных, с которыми он работает. После того как класс-шаблон определен, появляется возможность создавать отдельные экземпляры этого класса. Что касается библиотеки ввода/вывода, то Standard C++ создает две разные версии классов-шаблонов ввода/вывода: одну для 8-разрядных символов, а другую для широких (16-разрядных) символов. В данной книге рассказывается только о классах для 8-разрядных символов, поскольку именно они используются чаще всего.

Система ввода/вывода C++ строится на двух связанных, но различных иерархиях классов-шаблонов. Первая является производной от класса нижнего уровня **basic_streambuf.** Этот класс предоставляет базу для операций нижнего уровня по вводу и выводу, а также обеспечивает надлежащую поддержку всей системы ввода/вывода C++. До тех пор, пока вы

не погрузитесь в самые основы программирования ввода/вывода, непосредственно использовать класс **basic_streambuf** вам не понадобиться. Иерархия классов, с которой вам чаще всего придется иметь дело, является производной от класса **basic_ios.** Это класс ввода/вывода верхнего уровня, который обеспечивает форматирование, контроль ошибок и информацию о состоянии потока ввода/вывода. Класс **basic_ios** является базовым для нескольких производных классов, среди которых классы **basic_istream, basic_ostream и basic_iostream.** Эти классы используются соответственно для создания потоков ввода, вывода и ввода/вывода.

Как уже говорилось, библиотека ввода/вывода создает две отдельные версии иерархий классов: одну для 8-разрядных символов и другую для широких символов. В представленной ниже таблице показано соответствие имен классов-шаблонов их версиям для 8-разрядных символов (включая и те, о которых будет рассказано в главе 9).

Класс-шаблон	**Класс для 8-разрядных символов**
basic_streambuf	streambuf
basic_ios	ios
basic_istream	istream
basic_ostream	ostream
basic_iostream	iostream
basic_fstream	fstream
basic_ifstream	ifstream
basic_ofstream	ofstream

Имена классов для 8-разрядных символов будут употребляться далее на всем Протяжении книги, поскольку как раз эти имена и следует указывать в программах. Это именно те имена, которые использовались в прежней библиотеке ввода/вывода, и именно по этой причине на уровне исходного кода совместимы старая и новая библиотеки ввода/вывода.

И последнее замечание: в классе **ios** содержится множество функций и переменных — членов класса, которые контролируют или отображают основные операции потока ввода/вывода. Вы еще часто будете сталкиваться с классом **ios.** Запомните: чтобы получить доступ к этому важному классу, необходимо включить в программу заголовок **<iostream>.**

8.2. Форматируемый ввод/вывод.

До сих пор во всех примерах этой книги для вывода информации на экран использовались форматы, заданные в C++ по умолчанию. Однако информацию можно выводить в широком диапазоне форм. При этом с помощью системы ввода/вывода C++ можно форматировать данные так же, как это делала в C функция **printf().** Кроме того, можно изменять определенные параметры ввода информации.

Каждый поток ввода/вывода связан с набором флагов формата (format flags), которые управляют способом форматирования информации и представляют собой битовые маски (bitmasks). Эти маски объявлены в классе **ios** как данные перечислимого типа **fmtflags,** в котором определены следующие значения:

adjustfield	floatfield	right	skipws
basefield	hex	scientific	unitbuf

boolalpha	internal	showbase	uppercase
dec	left	showpoint	
fixed	oct	showpos	

Эти значения определены в классе **ios** и необходимы для установки или сброса флагов формата. Если вы пользуетесь устаревшим, нестандартным компилятором, может оказаться, что перечислимый тип данных **fmtflags** в нем не определен. В таких компиляторах для хранения флагов формата отводится длинное целое.

Когда при вводе информации в поток установлен флаг **skipws,** начальные невидимые символы (пробелы, табуляции и символы новой строки) отбрасываются. Когда флаг **skipws** сброшен, невидимые символы не отбрасываются.

Когда установлен флаг **left,** происходит выравнивание вывода по левому краю. Когда установлен флаг **right,** происходит выравнивание вывода по правому краю. Когда установлен флаг **internal,** для заполнения поля вывода происходит вставка пробелов между всеми цифрами и знаками числа. Если все эти флаги не установлены, то по умолчанию используется выравнивание по правому краю.3

По умолчанию числовые значения выводятся в десятичной системе счисления. Однако основание системы счисления можно поменять. Установка флага **oct** ведет к тому, что вывод будет осуществляться в восьмеричной системе счисления, а установка флага **hex** — в шестнадцатеричной. Чтобы вернуться к десятичной системе счисления, установите флаг **dec**.

Установка флага **showbase** ведет к выводу основания системы счисления. Например, шестнадцатеричное значение IF с этим флагом будет выводиться как OxIF.

По умолчанию при выводе значений в научной нотации символ "e" выводится в нижнем регистре. Кроме этого, при выводе шестнадцатеричного значения символ "x" тоже выводится в нижнем регистре. При установке флага **uppercase,** эти символы выводятся в верхнем регистре.

Установка флага **showpos** приводит к выводу знака + перед положительными значениями.

Установка флага **showpoint** ведет к появлению десятичной точки и последующих нулей при выводе любых значений с плавающей точкой.

При установке флага **scientific** числа с плавающей точкой выводятся в научной нотации. При установке флага **fixed** числа с плавающей точкой выводятся в обычной нотации. Если ни один из этих флагов не установлен, компилятор сам выбирает подходящий способ вывода.

Если установлен флаг **unitbuf,** то буфер очищается (flush) после каждой операции вставки (insertion operation).

При установленном флаге **boolalpha** значения булева типа выводятся в-виде ключевых слов **true и false.**

Одновременно на все поля, определенные с флагами **oct, dec и hex,** можно сослаться с помощью флага **basefield.** Аналогично на поля, определенные с флагами **left, right и internal,** можно сослаться с помощью флага **adjustfield.** И наконец, на поля с флагами **scientific и fixed** можно сослаться с помощью. флага **floatfield.**

Для установки флага формата пользуйтесь функцией **setf().** Эта функция является членом класса **ios.** Здесь показана ее основная форма:

fmtflags setf(fmfcflags *флаги);*

Эта функция возвращает предыдущие установки флагов формата и устанавливает новые, заданные значением *флаги.* (Значения всех остальных флагов не изменяются.) Например, для установки флага **showpos** можно воспользоваться следующей инструкцией:

поток_ввода/вывода. setf (ios: **: showpos) ;**

Здесь ***поток ввода/вывода*** **— это** тот поток, на который вы хотите повлиять. Обратите внимание на использование оператора расширения области видимости. Запомните, флаг **showpos** — это перечислимая константа внутри класса ios. Следовательно, чтобы сообщить компилятору об этом, необходимо поставить перед флагом **showpos** имя класса и оператор расширения области видимости. Если этого не сделать, константа **showpos** просто не будет распознана компилятором.

Важно понимать, что функция **setf()** является членом класса **ios и** влияет на созданные этим классом потоки ввода/вывода. Поэтому любой вызов функции **setf()** делается относительно конкретного потока. Нельзя вызвать функцию **setf()** саму по себе. Другими словами, в C++ нет понятия глобального состояния формата. Каждый поток ввода/вывода поддерживает собственную информацию о состоянии формата.

Вместо повторных вызовов функции **setf()** в одном вызове можно установить сразу несколько флагов. Для объединения необходимых флагов используйте оператор OR. Например, в следующем вызове функции **setf()** для потока **cout** устанавливаются флаги **showbase и hex:**

cout.setf(ios::showbase,| ios::hex) ;

Примеры

1. В этом примере показано, как установить несколько флагов формата.

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

int main()
{
  // вывод с использованием установок по умолчанию
  cout << 123.23 << " привет " << 100 << '\n';
  cout << 10 << ' ' << -10 << '\n';
  cout << 100.0 << '\n';

  // теперь меняем формат
  cout.unsetf(ios::dec); // требуется не для всех компиляторов
  cout.setf(ios::hex | ios::scientific);
  cout << 123.23 << " привет " << 100 << '\n';

  cout.setf(ios::showpos);
  cout << 10 << ' ' << -10 << '\n';
```

```
   cout.setf(ios::showpoint | ios::fixed);
   cout << 100.0;

   return 0;
}
```

После выполнения программы на экран выводится следующее:

```
123.23 привет 100
10 -10
100

1.232300e+02 привет 64
a fffffff6
 +100.000000
```

Обратите внимание, что флаг **showpos** влияет только на вывод десятичных значений. Он не влияет на число 10, когда оно выводится в шестнадцатеричной системе счисления. Кроме того, отметьте, что вызов функции **unsetf()** приводит к сбросу установленного по умолчанию флага dec. Этот вызов нужен не для всех компиляторов, а только для некоторых, для которых установка флага **dec** автоматически приводит к сбросу остальных флагов. Поэтому после сброса флага **dec** необходимо заново установить флаг **hex** или **oct**. Как правило, для лучшей переносимости программ лучше установить только то основание системы счисления, которое вы будете использовать и стереть остальные.

2. В следующей программе показано действие флага **uppercase**. В первую очередь устанавливаются флаги uppercase, **showbase** и **hex**. Затем выводится число 88 в шестнадцатеричной системе счисления. В этом случае символ шестнадцатеричной системы счисления "X" выводится в верхнем регистре. Далее с помощью функции **unsetf()** сбрасывается флаг uppercase и снова выводится шестнадцатеричное число 88. Теперь символ "x" оказывается в нижнем регистре.

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

int main()
{
   cout.unsetf(ios::dec);
   cout.setf(ios::uppercase | ios::showbase | ios::hex);

   cout << 88 << '\n';

   cout.unsetf(ios::uppercase);

   cout << 88 << '\n';

   return 0;
}
```

3. В следующей программе для вывода состояния флагов формата потока cout используется функция **flags()** Обратите особое внимание на функцию **showflags().** Она может вам пригодиться при разработке собственных программ.

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

void showflags();

int main()
{
   // отображение состояния флагов формата по умолчанию
   showflags();

   cout.setf(ios::oct | ios::showbase | ios::fixed);

   showflags();

   return 0;
}

// Эта функция выводит состояние флагов формата
void showflags()
{
   ios::fmtflags f;

   f = cout.flags(); // получение установок флагов формата

   if(f & ios::skipws) cout << "skipws установлен\n";
   else cout << "skipws сброшен\n";

   if(f & ios::left) cout << "left установлен\n";
   else cout << "left сброшен\n";

   if(f & ios::right) cout << "right установлен\n";
   else cout << "right сброшен\n";

   if(f & ios::internal) cout << "internal установлен\n";
   else cout << "internal сброшен\n";

   if(f & ios::dec) cout << "dec установлен\n";
   else cout << "dec сброшен\n";

   if(f & ios::oct) cout << "oct установлен\n";
   else cout << "oct сброшен\n";

   if(f & ios::hex) cout << "hex установлен\n";
   else cout << "hex сброшен\n";

   if(f & ios::showbase) cout << "showbase установлен\n";
   else cout << "showbase сброшен\n";

   if(f & ios::showpoint) cout << "showpoint установлен\n";
   else cout << "showpoint сброшен\n";

   if(f & ios::uppercase) cout << "uppercase установлен\n";
   else cout << "uppercase сброшен\n";

   if(f & ios::scientific) cout << "scientific установлен\n";
   else cout << "scientific сброшен\n";

   if(f & ios::fixed) cout << "fixed установлен\n";
   else cout << "fixed сброшен\n";

   if(f & ios::unitbuf) cout << "unitbuf установлен\n";
   else cout << "unitbuf сброшен\n";
```

```
    if(f & ios::boolalpha) cout << "boolalpha установлен\n";
    else cout << "boolalpha сброшен\n";

    cout << "\n";
};
```

В функции **showflagsO** объявляется локальная переменная f типа **fmtflags-** Если в вашем компиляторе тип данных **fintBags** не определен, объявите переменную f типа **long.** Ниже показан результат выполнения программы:

```
skipws установлен
left сброшен
right сброшен
internal сброшен
dec установлен
oct сброшен
hex сброшен
showbase сброшен
showpoint сброшен
showpos сброшен
uppercase сброшен
scientific сброшен
fixed сброшен
unitbuf сброшен
boolalpha сброшен

skipws установлен
left сброшен
right сброшен
internal сброшен
dec установлен
oct установлен
hex сброшен
showbase установлен
showpoint сброшен
showpos сброшен
uppercase сброшен
scientific сброшен
fixed установлен
unitbuf сброшен
booialpha сброшен
```

4. В следующей программе проиллюстрирована работа второй версии функции **flagsQ-** Сначала, устанавливая флаги **showpos, showbase, oct и right,** мы строим маску флагов. Затем с помощью функции **flags()** для потока **coin** маска связывается с переменной флагов. С помощью функции **showflagsO** проверяется правильность установки флагов. (Это та же функция, которая использовалась в предыдущей программе.)

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

void showflags();

int main()
{
   // отображение состояния флагов формата по умолчанию
   showflags();

   // устанавливаются флаги showpos, showbase, oct и right;
   // остальные сбрасываются
   ios::fmtflags f = ios::showpos | ios::showbase |
```

```
                ios:: oct | ios::right;

    cout.flags(f); // установка флагов

    showflags();

    return 0;
}
```

8.3. Функции *width()*, *precision()* и *fill().*

Кроме флагов формата в классе **ios** определены три функции-члена. Эти функции устанавливают следующие параметры формата: ширину поля, точность и символ заполнения. Этими функциями являются соответственно Функции **width(), precision() и fill().**

По умолчанию при выводе любого значения оно занимает столько позиций, сколько символов выводится. Однако с помощью функции **width()** можно задать минимальную ширину поля. Ниже показан прототип этой функции;

streamsize width**(streamsize** *w);*

Ширина поля задается параметром ***w,*** а функция возвращает предыдущую ширину поля- Тип данных **streamsize** определен в заголовочном файле **<iostream>** как одна из форм целого. В некоторых компиляторах при выполнении каждой операции вывода значение ширины поля возвращается к своему состоянию по умолчанию, поэтому перед каждой инструкцией вывода может понадобиться устанавливать минимальную ширину поля.

После установки минимальной ширины поля, если выводимое значение требует поле, меньшее заданной ширины, остаток поля заполняется текущим символом заполнения (по умолчанию пробелом) так, чтобы была занята вся ширина поля. Однако запомните, если размер выводимого значения превосходит минимальную ширину поля, будет занято столько символов, сколько нужно. Выводимое значение не усекается.

По умолчанию при выводе значений с плавающей точкой точность равна шести цифрам. Однако с помощью функции **precision()** это число можно изменить. Ниже показан прототип функции **precision():**

streamsize precision(streamsize **p);**

Точность (число выводимых цифр после запятой) задается параметром ***p,*** а возвращает функция прежнее значение точности.

По умолчанию, если требуется заполнить свободные поля, используются пробелы. Однако с помощью функции **fill()** можно изменить символ заполнения. Ниже показан прототип функции **fill():**

char **fill**(char ch) ;

После вызова функции **fill()** символ ***ch*** становится новым символом заполнения, а функция возвращает прежнее значение символа заполнения.

Примеры

1. В этом примере показана программа, которая иллюстрирует работу функций формата:

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

int main()
{
   cout.width(10); // установка минимальной ширины поля
   cout << "Привет" << '\n'; // по умолчанию выравнивание вправо
   cout.fill('%'); // установка символа заполнения
   cout.width(10); // установка ширины поля
   cout << "Привет" << '\n'; // по умолчанию выравнивание вправо
   cout.setf(ios::left); // выравнивание влево
   cout.width(10); // установка ширины поля
   cout << "Привет" << '\n'; // выравнивание влево

   cout.width(10); // установка ширины поля
   cout.precision(10); // установка точности в 10 цифр
   cout << 123.234567 << '\n';
   cout.width(10); // установка ширины поля
   cout.precision(6); // установка точности в 6 цифр
   cout << 123.234567 << '\n';

   return 0;
}
```

После выполнения программы на экран выводится следующее:

```
Привет
 %%%%Привет
Привет%%%%
123.234567
123.235%%%
```

Обратите внимание, что ширина поля устанавливается перед каждой инструкцией вывода.

2. В следующей программе показано, как с помощью функций установки флагов формата ввода/вывода C++ создать выровненную таблицу чисел:

```
// Создание таблицы квадратных корней и квадратов
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
   double x;

   cout.precision(4);
   cout << "      x     sqrt(x)     x^2\n\n";

   for(x = 2.0; x <= 20.0; x++) {
      cout.width(7);
      cout << x << " ";
      cout.width(7);
      cout << sqrt(x) << " ";
      cout.width(7);
      cout << x*x << "\n";
```

```
    }

    return 0;
}
```

После выполнения программы на экран выводится следующее:

x	sqrt(x)	x^2
2	1.414	4
3	1.732	9
4	2	16
5	2.236	25
6	2.449	36
7	2.646	49
8	2.828	64
9	3	81
10	3.162	100
11	3.317	121
12	3.464	144
13	3.606	169
14	3.742	196
15	3.873	225
16	4	256
17	4.123	289
18	4.'243	324
19	4.359	361
20	4.472	400

8.4. Манипуляторы ввода/вывода.

В системе ввода/вывода C++ имеется еще один способ форматирования информации. Этот способ подразумевает использование специальных функций — *манипуляторов ввода/вывода (I/O manipulators).* Как вы увидите далее, манипуляторы ввода/вывода являются, в некоторых ситуациях, более удобными, чем флаги и функции формата класса **ios.**

Манипуляторы ввода/вывода являются специальными функциями формата ввода/вывода, которые, в отличие от функций ~ членов класса **ios,** могут располагаться *внутри* инструкций ввода/вывода. Стандартные манипуляторы показаны в табл. 8.1. Как можно заметить при изучении таблицы, значительная часть манипуляторов ввода/вывода по своим действиям аналогична соответствующим функциям — членам класса **ios.** Многие манипуляторы, представленные в табл. 8.1, стали частью языка совсем недавно, после появления стандарта Standard C++, и поддерживаются только современными компиляторами.

Для доступа к манипуляторам с параметрами (таким, как функция **setw()),** необходимо включить в программу заголовок **<iomanip>. В** этом заголовке нет необходимости при использовании манипуляторов без параметров.
Как уже установлено, манипуляторы можно задавать внутри цепочки операций ввода/вывода. Например:

```
cout « oct« 100 « hex« 100;
cout « setw(10). « 100;
```

Первая инструкция сообщает потоку **cout** о необходимости вывода целых в восьмеричной системе счисления и выводит число 100 в восьмеричной системе счисления. Затем она сообщает потоку ввода/вывода о необходимости вывода целых в шестнадцатеричной системе счисления и далее осуществляется вывод числа 100 уже в шестнадцатеричном формате. Во второй инструкции устанавливается ширина поля равная 10, и затем снова выводится 100 в шестнадцатеричном формате. Обратите внимание, если используется манипулятор без аргументов (в данном примере им является манипулятор **oct),** скобки за ним не ставятся, поскольку это на самом деле адрес манипулятора, передаваемый перегруженному оператору «.

Таблица 8.1 *Манипуляторы ввода/вывода языка Standard C++*

Манипулятор	Назначение	Ввод/вывод
boolalpha	Установка флага boolaipha	Ввод/Вывод
dec	Установка флага dec	Ввод/Вывод
endl	Вывод символа новой строки и очистка потока	Вывод
ends .	Вывод значения NULL	Вывод
fixed	Установка флага fixed	Вывод
flush	Очистка потока	Вывод
hex	Установка флага hex	Ввод/Вывод
internal	Установка флага internal	Вывод
left	Установка флага left	Вывод
noboolalpha	Сброс флага boolaipha	Ввод/Вывод
noshowbase	Сброс **флага** showbase	Вывод
noshowpoint	Сброс флага showpos	Вывод
noshowpos	Сброс **флага** showpoint	Вывод
noskipws	Сброс флага skipws	Ввод
nounitbuf	Сброс флага unitbuf	Вывод
nouppercase	Сброс флага uppercase	Вывод
oct	Установка флага oct	Ввод/Вывод
resetiosflags(fmtflags f)	Сброс флагов, заданных **параметром *f***	Ввод/Вывод
right	Установка флага **right**	Вывод
scientific	Установка флага scientific	Вывод
setbase (int основание***)***	Задание *основания* системы счисления	Ввод/Вывод
setfill(int *ch)*	Задание символа заполнения *ch*	Вывод

setiosflags(fmtflags ***f)***	**Установка флагов, заданных параметром *f***	Ввод/Вывод
setprecision(int **p)**	**Задание** числа **цифр точности равным p**	Вывод
setw(int ***w)***	**Задание** ширины **поля равным *w позиций***	Вывод
showbase	**Установка** флага **showbase**	Вывод
showpoint	Установка флага **showpoint**	Вывод
showpos	Установка флага showpos	Вывод
skipws	Установка флага skipws	Ввод
unitbuf	**Установка флага** unitbuf	Вывод
uppercase	**Установка флага** uppercase	Вывод
ws	Пропуск начальных пробелов	Ввод

Запомните, что манипулятор ввода/вывода влияет только на поток, частью которого является выражение ввода/вывода, содержащего манипулятор. Манипуляторы ввода/вывода *не* влияют на все, открытые в данный момент, потоки.

Как отмечалось в предыдущем примере, главным преимуществом манипуляторов по сравнению с функциями — членами класса **ios** является то, что манипуляторы обычно удобнее, так как позволяют писать более компактные программы.

Если вы с помощью манипулятора хотите установить конкретные флаги формата, используйте функцию **setiosflags().** Этот манипулятор реализует ту же функцию, что и функция-член **setf().** Для сброса флагов формата используйте манипулятор **resetiosflags()**- Этот манипулятор эквивалентен функции **unsetf().**

Примеры

1. В этой программе представлено несколько манипуляторов ввода/вывода:

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

int main()
{
   cout << hex << 100 << endl;
   cout << oct << 10 << endl;

   cout << setfill('X') << setw(10);
   cout << 100 << " привет " << endl;

   return 0;
}
```

После выполнения программы на экран выводится следующее:

```
64
12
XXXXXXX144 привет
```

2. Здесь представлена другая версия программы, в которой на экран выводится таблица квадратов и квадратных корней чисел от 2 до 20. В этой версии вместо функций-членов и флагов формата используются манипуляторы ввода/вывода.

```
/* В этой версии для вывода таблицы квадратов и квадратных корней используются манипуляторы
 */
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main()
{
   double x;

   cout << setprecision(4);
   cout << "x      sqrt(x)     x^2\n\n";

   for(x = 2.0; x <= 20.0; x++) {
      cout << setw(7) << x << " ";
      cout << setw(7) << sqrt(x) << " ";
      cout << setw(7) << x*x << '\n';
   }

   return 0;
}
```

3. Одним из самых интересных флагов формата новой библиотеки ввода/вывода является флаг **boolalpha.** Этот флаг можно установить либо непосредственно, либо с помощью манипулятора **boolalpha.** Интересным этот флаг делает то, что он позволяет реализовать ввод и вывод значений булева типа, т. е. ключевых слов **true и false,** вместо которых раньше обычно приходилось использовать соответственно 1 для истинного значения и 0 для ложного.

```
// Использование флага формата boolalpha
#include <iostream>
using namespace std;

int main()
{
   bool b;

   cout << "Перед установкой флага формата boolalpha: ";
   b = true;
   cout << b << " ";
   b = false;
   cout << b << endl;

   cout << "После установки флага формата boolalpha: ";
   b = true;
   cout << boolalpha << b << " ";
   b = false;
   cout << b << endl;

   cout << "Введите значение булева типа: ";
   cin >> boolalpha >> b; // здесь можно ввести true или false
   cout << "Введенное значение: " << b;

   return 0;
}
```

Примерный результат работы программы:

```
Перед установкой флага формата boolalpha: 1 О
После установки флага формата boolaipha: true false
Введите значение булева типа: true Введенное значение: true
```

Как видите, после установки флага формата **boolalpha,** для обозначения вв(димых и выводимых значений булева типа используются ключевые слова **true и false.** Отметьте, что флаг формата **boolalpha** необходимо устанавливать отдельно для потока **cin** и отдельно для потока **cout.** Как и в случае с другими флагами формата, установка флага **boolalpha** для одного потока вовсе не подразумевает его автоматической установки для другого потока.

8.5. пользовательские функции вывода.

Как уже отмечалось в этой книге, одним из доводов в пользу использования операторов ввода/вывода C++, вместо аналогичных им функций ввода/вывода **C,** является возможность перегрузки операторов ввода/вывода для создаваемых вами классов. В этом разделе вы узнаете, как перегрузить оператор вывода **<<**.

В языке C++ вывод иногда называется *вставкой (insertion),* а оператор **<<** — *оператором вставки (insertion operator).* Когда вы для вывода информации перегружаете оператор **>>**, вы создаете *функцию вставки (inserter function* или *inserter).* Рациональность этим терминам дает то, что оператор вывода *вставляет (inserts)* информацию в поток. Во избежание путаницы мы будем называть функцию вставки пользовательской функцией вывода.

У всех пользовательских функций вывода следующая основная форма:

```
ostream &operator<<(ostream fistream,имя класса объект)
 {
// тело пользовательской функции вывода
return stream;
}
```

Первый параметр является ссылкой на объект типа **ostream.** Это означает, что поток ***stream*** должен быть потоком вывода. (Запомните, класс **ostream** является производным от класса ios.) Второй параметр получает выводимый объект. (Он, если для вашего приложения это нужно, тоже может быть параметром-ссылкой). Обратите внимание, что пользовательская функция вывода возвращает ссылку на поток ***stream,*** который имеет тип **ostream.** Это необходимо, если перегруженный оператор **<<** должен использоваться в ряде последовательных выражений ввода/вывода:

```
cout << ob1 << ob2 << ob3;
```

Внутри пользовательской функции вывода можно выполнить любую процедуру. То, что будет делать эта функция, полностью зависит от вас. Однако в соответствии с хорошим стилем программирования, следует ограничите задачи пользовательской функции вывода только вставкой информации поток.

Хотя это на первый взгляд может показаться странным, но пользовательская! функция вывода *не* может быть членом класса, для работы с которым она создана. И вот почему. Если оператор-функция любого типа является членом класса, то левый операнд, который неявно

передается через указатель **this,** является объектом, генерирующим вызов оператор-функции. Это подразумевает, что левый операнд является объектом этого класса. Поэтому, если перегруженная оператор-функция является членом класса, тогда левый операнд должен быть объектом этого класса. Однако, когда вы создаете пользовательскую функцию вывода, левый операнд становится потоком, а не объектом класса, а правый операнд — объектом, который нужно вывести. Именно поэтому пользовательская функция вывода не может быть функцией-членом.

То, что пользовательская функция вывода не может быть функцией-членом на первый взгляд кажется серьезным изъяном C++, поскольку подразумевает, что все данные класса, выводимые в поток через эту функцию, должны быть открытыми, нарушая тем самым ключевой принцип инкапсуляции. Однако это не так. Несмотря на то, что пользовательские функции вывода не могут быть членами класса, для работы с которым они разработаны, они могут быть дружественными классу. В подавляющем большинстве реальных ситуаций, с которыми вам придется столкнуться при программировании ввода/вывода, перегружаемая пользовательская функция вывода будет дружественной классу, для которого она создана.

Примеры

1. Для начала рассмотрим простой пример, в котором для класса **coord,** разработанного в предыдущей главе, создается пользовательская функция вывода:

```
// Использование дружественной функции вывода для объектов типа coord
#include <iostream>
using namespace std;

class coord {
   int x, y;
public:
   coord() { x = 0; y= 0; }
   coord(int i, int j) { x = i; y = j; }
   friend ostream &operator<<(ostream &stream, coord ob);
};

ostream &operator<<(ostream &stream, coord ob)
{
   stream << ob.x << ", " << ob.y << '\n';
   return stream;
}

int main()
{
   coord a(1, 1), b(10, 23);

   cout << a << b;

   return 0;
}
```

В результате выполнения программы на экран выводится следующее:

```
1,  1
10, 23
```

Пользовательская функция вывода этой программы иллюстрирует одну очень важную для создания ваших собственных функций особенность: их нужно разрабатывать, возможно,

более обобщенными. В данном конкретном случае инструкция ввода/вывода внутри функции вставляет значения x и y в поток **stream,** который и является передаваемым в функцию потоком. Как вы увидите в следующей главе, та же самая пользовательская функция вывода, которая в нашем примере используется для вывода информации на экран, может использоваться и для ее вывода в *любой* поток. Тем не менее, начи-^ нающие программисты иногда пишут пользовательскую функцию вывода для класса **coord** следующим образом:

```
ostream &operator<<(ostream Sstream, coord ob)
{
        cout<<ob.x<< “, “”<<ob.y<<’\n’;
        return stream;
}
```

В этом случае выражение жестко запрограммировано на вывод информации на стандартное устройство вывода, связанное с классом **cout.** Это ведет к тому, что другие потоки не могут воспользоваться вашей функцией. Вывод очевиден, следует делать свои функции, возможно, более обобщенными, поскольку это никогда не повредит, а иногда может оказаться полезным.

2. В следующей версии предыдущей программы, пользовательская функция вывода *не* является дружественной классу **coord.** Поскольку у пользовательской функции вывода нет доступа к закрытой части класса **coord,** переменные x и y приходиться делать открытыми.

```
// Создание не дружественной функции вывода для объектов типа coord
#include <iostream>
using namespace std;

class coord {
public:
   int x, y; // должны быть открытыми
   coord() { x = 0; y= 0; }
   coord(int i, int j) { x = i; y = j; }
};

// Пользовательская функция вывода для объектов класса coord
ostream &operator<<(ostream &stream, coord ob)
{
   stream << ob.x << ", " << ob.y << '\n';
   return stream;
}

int main()
{
   coord a(1, 1), b(10, 23);
   cout << a << b;

   return 0;
}
```

3. Возможности пользовательских функций вывода не ограничиваются выводом текстовой информации. Они могут выполнить любое действие или преобразование, необходимое для вывода информации в том виде, который требуется из-за особенностей устройства или ситуации. Например, совершенно разумно создать пользовательскую функцию вывода для отправки информации на плоттер. В этом случае, кроме собственно информации, функция передаст предназначенные для управления плоттером коды. Чтобы вы могли почувствовать вкус к такого рода функциям, в следующей программе создается класс **triangle,** в котором хранится ширина и высота прямоугольного треугольника. Пользовательская функция вывода

этого класса выводит треугольник на экран.

```
// Эта программа рисует прямоугольные треугольники
#include <iostream>
using namespace std;

class triangle {
   int height, base;
public:
   triangle(int h, int b) { height = h; base= b; }
   friend ostream &operator<<(ostream &stream, triangle ob);
};

// рисование треугольника
ostream &operator<<(ostream &stream, triangle ob)
{
   int i, j, h, k;

   i = j = ob.base - 1;
   for(h=ob.height - 1; h; h --) {
      for(k=i; k; k --)
         stream << ' ';
      stream << '*';

      if(j!=i) {
         for(k=j - i - 1; k; k --)
            stream << ' ';
         stream << '*';
      }

      i --;
      stream << '\n';
   }
   for(k=0; k <ob.base; k++) stream << '*';
   stream << '\n';

   return stream;
}

int main()
{
   triangle t1(5, 5), t2(10, 10), t3(12, 12);

   cout << t1;
   cout << endl << t2 << endl << t3;

   return 0;
}
```

Отметьте, что должным образом разработанная пользовательская функция вывода может быть целиком вставлена в "обычное" выражение ввода/вывода. После выполнения программы на экран выводится следующее:

```
    *
   * *
  *   *
 *     *
*******
```

8.6. Пользовательские функции ввода.

Точно так же, как мы перегружали оператор вывода «, можно перегрузить и оператор ввода ». В C++ оператор ввода » иногда называют *оператором извлечения (extraction operator),* а функцию, перегружающую этот оператор, — функцией извлечения *(extractor).* Смысл этих терминов в том, что При вводе информации мы извлекаем данные из потока. Во избежание путаницы мы будем называть функцию извлечения пользовательской функцией ввода.

Здесь показана основная форма пользовательской функции ввода:

```
istream &operator>> (istream &stream, имя_класса &объект)
 {
        // тело пользовательской функции ввода
        return stream;
}
```

Пользовательские функции ввода возвращают ссылку, на поток **istream,** который является потоком ввода. Первый параметр тоже является ссылкой на поток ввода. Второй параметр — это ссылка на объект, получающий вводимую информацию.

Так же, как и пользовательская функция вывода, пользовательская функция ввода не может быть функцией-членом. Хотя внутри такой функции может быть выполнена любая операция, лучше ограничить ее работу вводом информации.

Примеры

1. В этой программе к классу **coord** добавлена пользовательская функция ввода:

```
// Добавление дружественной функции ввода для объектов типа coord
#include <iostream>
using namespace std;

class coord {
   int x, y;
public:
   coord() { x = 0; y= 0; }
   coord(int i, int j) { x = i; y = j; }
   friend ostream &operator<<(ostream &stream, coord ob);
   friend istream &operator>>(istream &stream, coord &ob);
};

ostream &operator<<(ostream &stream, coord ob)
{
   stream << ob.x << ", " << ob.y << '\n';
   return stream;
}

istream &operator>>(istream &stream, coord &ob)
{
   cout << "Введите координаты: ";
   stream >> ob.x >> ob.y;
   return stream;
}

int main()
{
   coord a(1, 1), b(10, 23);
```

```
   cout << a << b;

   cin >> a;
   cout << a;

   return 0;
}
```

Обратите внимание, как пользовательская функция ввода формирует строку-приглашение для ввода данных. Хотя во многих случаях этого не требуется (или даже это нежелательно), пользовательская функция ввода показывает, как в случае необходимости почти без усложнения программы можно выдать приглашающее сообщение.

2. Здесь создается класс **inventory** (инвентарь), в котором хранится название какого-то предмета, количество выданных на руки штук и стоимость одной штуки. В программу для этого класса включены пользовательские функции ввода и вывода.

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

class inventory {
   char item[40]; // название предмета
   int onhand;    // количество предметов на руках
   double cost;   // цена предмета
public:
   inventory(char *i, int o, double c)
   {
      strcpy(item, i);
      onhand = o;
      cost = c;
   }
   friend ostream &operator<<(ostream &stream, inventory ob);
   friend istream &operator>>(istream &stream, inventory &ob);
};

ostream &operator<<(ostream &stream, inventory ob)
{
   stream << ob.item << ": " << ob.onhand;
   stream << " на руках по цене $" << ob.cost << '\n';

   return stream;
}

istream &operator>>(istream &stream, inventory &ob)
{
   cout << "Введите название предмета: ";
   stream >> ob.item;
   cout << "Введите число выданных на руки экземпляров: ";
   stream >> ob.onhand;
   cout << "Введите стоимость экземпляра: ";
   stream >> ob.cost;

   return stream;
}

int main()
{
   inventory ob("hammer", 4, 12.55);
```

```
    cout << ob;

    cin >> ob;

    cout << ob;

    return 0;
}
```