

**ЛАБОРАТОРНАЯ РАБОТА №3 ПО ПРЕДМЕТУ  
«ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ  
ПРОГРАММИРОВАНИЕ»  
ТЕМА «ПОТОКИ ВВОДА/ВЫВОДА В ЯЗЫКЕ C++»**

**ЦЕЛЬ РАБОТЫ:** изучить понятие потока, организацию ввода данных из потока и вывода в поток, контроль состояния потока и исправление ошибок, неформатированный ввод-вывод, манипуляторы потоков (стандартные и определяемые пользователем).

**ПОТОКОВЫЕ КЛАССЫ. ИЕРАРХИЯ КЛАССОВ ПОТОКОВ**

Ввод-вывод данных в языке C++ осуществляется либо с помощью функций ввода-вывода в стиле C, либо с использованием библиотеки классов языка C++. Не следует смешивать в одном коде функций языка C и конструкции языка C++. В языке C++ поток ввода-вывода представляет собой объект класса потокового ввода-вывода. Разные потоки предназначены для обработки разных типов данных. Например, класс *ifstream* представляет собой поток ввода данных, который может быть связан с файлом.

Какие преимущества дает использование потоковых классов для ввода/вывода языка C++ вместо традиционных функций языка C *printf()*, *scanf()*, *fprintf()*, *fscanf()*?

Одним из аргументов в пользу использования потоков языка C++ является **простота использования**, связанная с тем, что каждый тип сам определяет, как его объекты будут обработаны в потоке. Другой причиной является то, что можно **перегружать стандартные операторы вывода в поток (оператор вставки, <<) и ввода из потока (извлечения, >>)** для работы с объектами создаваемых классов. Это позволяет работать с собственными классами как со стандартными типами данных, что делает программирование эффективнее и избавляет от множества ошибок.

Потоковые классы имеют довольно сложную иерархическую структуру. На рисунке 1 показана взаимосвязь основных классов потокового ввода-вывода языка C++.

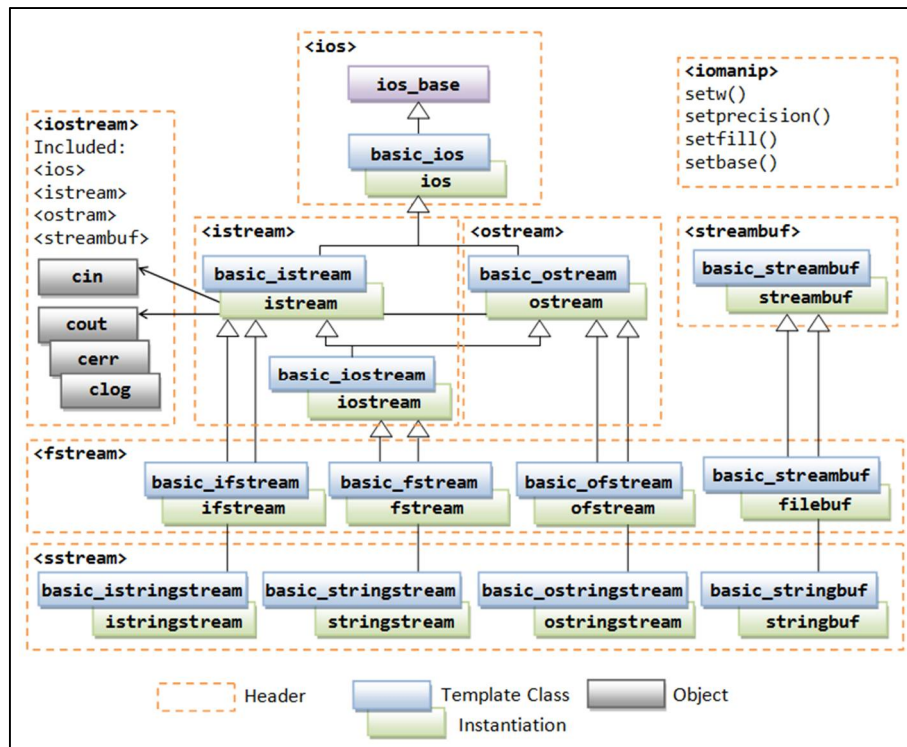


Рисунок 1 – Взаимосвязи основных классов организации потокового ввода-вывода языка C++

При работе с потоковой библиотекой ввода-вывода C++ активно используются следующие классы: *basic\_ios* – базовый потоковый класс; *basic\_istream* – класс входных потоков; *basic\_ostream* – класс выходных потоков; *basic\_iostream* – класс потоков ввода и вывода; *basic\_ifstream* – класс входных файловых потоков; *basic\_ofstream* – класс выходных файловых потоков; *basic\_fstream* – класс файловых потоков ввода и вывода.

Оператор извлечения (>>) перегружен в заголовочном файле *istream*, операция вставки (<<) перегружен в заголовочном файле *ostream*.

Объект *cout* представляет стандартный выходной поток и **по умолчанию** выводит данные на экран. Объект *cin* представляет стандартный входной поток и **по умолчанию** считывает данные с клавиатуры.

Классы, используемые для вывода данных на консоль и ввода с клавиатуры, описаны в заголовочном файле *iostream*. Классы, используемые для ввода/вывода файлов, объявлены в файле *fstream*. Манипуляторы описаны в файле *iomanip*.

Класс *ios\_base* является родительским для всех потоковых классов. Он содержит множество констант и методов, общих для операций ввода/вывода любых видов, например, флаги форматирования *showpoint* и *fixed*.

Класс *basic\_istream* содержит такие методы, как *get()*, *getline()*, *read()* и перегруженный оператор извлечения из потока (>>). Класс *basic\_ostream* содержит методы *put()*, *write()* и перегруженный оператор вставки в поток (<<). Класс *basic\_iostream* является наследником одновременно классов *basic\_ostream* и *basic\_istream*. Его производные классы могут использоваться одновременно для записи и чтения.

## МАНИПУЛЯТОРЫ. ФЛАГИ ФОРМАТИРОВАНИЯ

Для управления вводом-выводом данных в языке C++ используются флаги форматированного ввода-вывода и манипуляторы форматирования,

которые позволяют установить параметры ввода-вывода, действующие во всех последующих операторах ввода-вывода до тех пор, пока не будут отменены. Т.е. флаги форматирования позволяют включить или выключить один из параметров вывода или считывания данных. Для установки флага вывода используется метод *setf()*, а для отмены флага метод *unsetf()*. Общий вид их использования выгляди́м следующим образом:

```
cout.setf(ios::flag);
cout.unsetf(ios::flag);
```

где параметр *ios::flag* обозначает имя устанавливаемого или отменяемого флага. Если необходимо установить несколько флагов, то можно воспользоваться оператором «или» (*|*). В этом случае оператор будет иметь следующий общий вид:

```
cout.setf(ios::flag1 | ios::flag2 | ios::flag3)
```

В данном случае параметры *flag1*, *flag2*, *flag3* определяют имена устанавливаемых флагов.

Манипуляторы устанавливают в объектах *cin* и *cout* параметры **текущего** оператора ввода-вывода. Манипуляторы представляют собой специальные методы, которые включаются в выражения ввода/вывода для изменения параметров форматирования потока. Манипуляторы работают переключателями, определяющими различные форматы и способы ввода/вывода.

В таблице 1 приводится список основных флагов форматированного вывода.

Таблица 1 – Флаги форматированного вывода

Флаг	Значение
<i>boolalpha</i>	Указывает, что значения переменных типа <i>bool</i> отображаются в потоке в виде слов « <i>true</i> » или « <i>false</i> », т.е. в текстовом виде
<i>dec</i>	Указывает, что целочисленные переменные отображаются десятичной системе счисления. Установлен по умолчанию
<i>fixed</i>	Указывает, что число с плавающей запятой отображается в десятичной системе счисления. Установлен по умолчанию
<i>hex</i>	Указывает, что целочисленные переменные отображаются в шестнадцатеричной системе счисления
<i>internal</i>	Делает так, чтобы знак числа был выровнен по левому краю, а число по правому краю
<i>left</i>	Позволяет тексту, ширина которого меньше ширины вывода, появиться в потоке на одном уровне с левым полем. Установлен по умолчанию
<i>noboolalpha</i>	Указывает, что значения переменных типа <i>bool</i> отображаются в потоке как значения 1 или 0
<i>noshowbase</i>	Отключает отображение системы счисления, в которой отображается число
<i>noshowpoint</i>	Отображает только целочисленную часть чисел с плавающей запятой, дробная часть которых равна нулю
<i>noshowpos</i>	Приводит к тому, что у положительных чисел не указывается явно знак
<i>noskipws</i>	Заставляет пробелы считываться входным потоком
<i>nouppercase</i>	Указывает, что шестнадцатеричные цифры и показатель степени в научной нотации отображаются в нижнем регистре

<i>oct</i>	Указывает, что целочисленные переменные отображаются в восьмеричной системе счисления
<i>right</i>	Приводит к тому, что текст, ширина которого меньше ширины вывода, появляется в потоке на одном уровне с правым краем
<i>scientific</i>	Вызывает отображение чисел с плавающей запятой с использованием научной нотации (экспоненциальной формы)
<i>showbase</i>	Указывает систему счисления, в которой отображается число
<i>showpoint</i>	Отображает целочисленную часть числа с плавающей запятой и цифры справа от десятичной точки, даже если дробная часть равна нулю
<i>showpos</i>	Вызывает явное отображение знака «+» для положительных чисел
<i>skipws</i>	Заставляет пробелы не считываться входным потоком ( <i>skip white spaces</i> )
<i>uppercase</i>	Указывает, что шестнадцатеричные цифры и показатель степени в научной нотации отображаются в верхнем регистре

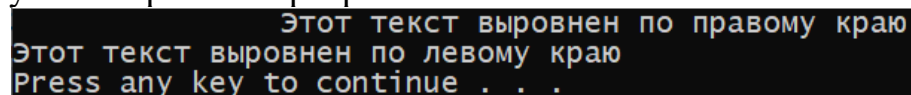
Флаги удобно использовать в тех случаях, когда следует изменить параметры всех последующих операторов ввода-вывода. Использование большего количества флагов для управления одним оператором ввода-вывода не совсем удобно. Еще одним способом форматирования является использование манипуляторов непосредственно в операторах *cin* и *cout*.

Так как флаги форматирования являются компонентами класса *ios*, то к ним обращаются посредством написания имени класса и оператора разрешения (например, *ios::skipws*). Все без исключения флаги могут быть выставлены с помощью методов *setf()* и *unsetf()*. Для доступа к этим функциям потребуется заголовочный файл *iomanip*. В следующем примере показано использование флагов форматирования.

//Пример №1. Использование флагов выравнивания

```
#include <iostream>
#include <windows.h>
using namespace std;
void Ex1() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    cout.setf(ios::right);
    cout.width(50);
    cout << "Этот текст выровнен по правому краю" << endl;
    cout.unsetf(ios::right); //вернуться к прежнему форматированию
    cout << "Этот текст выровнен по левому краю" << endl;
}
```

Результаты работы программы:



```
Этот текст выровнен по правому краю
Этот текст выровнен по левому краю
Press any key to continue . . .
```

Многие манипуляторы могут быть установлены с помощью специальных функций.

//Пример №2. Использование флагов форматирования и манипуляторов

```
#include <iostream>
#include <iomanip>
void Ex2() {
    const long double result = std::acos(-1.L);
```

```

std::cout << std::fixed << "Точность, устанавливаемая по
умолчанию(6): " << result << '\n'
    << "установка точности с помощью функции
std::setprecision(10):" << std::setprecision(10) << result << '\n'
    << "установка максимальной точности:" <<
std::setprecision(std::numeric_limits<long double>::digits10) <<
result << '\n';
double newResult = -2646693125139304345. / 842468587426513207.;
std::cout << std::setiosflags(std::ios::fixed) <<
std::setprecision(0) << newResult << '\n';
std::cout << std::showpoint << newResult << '\n';
std::cout << "Left fill:\n" << std::left << std::setfill(' ')
    << std::setw(12) << -1.23 << '\n'
    << std::setw(12) << std::hex << std::showbase << 255 <<
std::endl;
std::cout << "Internal fill:\n" << std::internal
    << std::setw(12) << std::setprecision(3) << -1.23 << '\n'
    << std::setw(12) << 255 << "\n";
std::cout << "Right fill:\n" << std::right
    << std::setw(15) << -1.23 << '\n'
    << std::setw(15) << std::hex << std::noshowbase << 255 << '\n';
}

```

Результаты работы программы:

```

Этот текст выровнен по правому краю
Этот текст выровнен по левому краю
Точность, устанавливаемая по умолчанию(6): 3.141593
установка точности с помощью функции std::setprecision(10):3.1415926536
установка максимальной точности:3.141592653589793
-3
-3.
Left fill:
-1.
0xff
Internal fill:
- 1.230
0x ff
Right fill:
-1.230
ff
Press any key to continue . . .

```

Как показывают примеры, манипуляторы бывают двух видов — с аргументами и без аргументов. Манипуляторы встраиваются непосредственно в операторы ввода-вывода. В таблице 2 приведены основные манипуляторы, определенные в файлах `<ostream>`, `<istream>`.

Таблица 2– Манипуляторы, определенные в файлах `<ostream>`, `<istream>`

Манипулятор	Назначение
<i>endl</i>	Завершает строку и сбрасывает буфер
<i>ends</i>	Завершает строку
<i>flush</i>	Очищает буфер
<i>swap</i>	Меняет местами два потоковых объекта
<i>ws</i>	Пропускает пробелы в потоке

Манипуляторы вставляются прямо в поток. Например, чтобы вывести значение переменной *var* в шестнадцатеричной форме, надо указать манипулятор *hex* в потоке *cout*:



```
cout << hex << var;
```

Манипуляторы действуют только на те данные, которые следуют за ними в потоке, а не на те, которые находятся перед ними. В таблице 3 представлены некоторые функции из заголовочного файла *<iomanip>*.

Таблица 3 – Некоторые функции для работы с манипуляторами из файла *<iomanip>*

Манипулятор	Назначение
<i>T1 resetiosflags(ios_base::fmtflags mask);</i>	Сбрасывает указанные флаги (параметр <i>mask</i> ) форматирования
<i>template &lt;class Elem&gt; T4 setfill(Elem Ch);</i>	Задаёт символ (параметр <i>Ch</i> ), который будет использоваться для заполнения пробелов на экране с выравниванием по правому краю
<i>T2 setiosflags(ios_base::fmtflags mask);</i>	Устанавливает указанные флаги форматирования (параметр <i>mask</i> )
<i>T5 setprecision(streamsize Prec);</i>	Задаёт точность для значений с плавающей запятой (параметр <i>Prec</i> )
<i>T6 setw(streamsize Wide);</i>	Задаёт ширину поля вывода в <i>Wide</i> символов для следующего элемента в потоке

Таблица 4 содержит некоторые функции для получения сведений о состоянии потока и обработки ошибок.

Таблица 4 – Функции классов *ios\_base* и *basic\_ios*

Функция	Назначение
<i>char_type fill(char_type Char);</i> <i>char_type fill() const;</i>	Задаёт и возвращает символ заполнения, который будет использоваться, когда текст не так широк, как заданная ширина вывода. По умолчанию символом заполнения является пробел
<i>streamsize precision(streamsize _Prec);</i> <i>streamsize precision() const;</i>	Возвращает и задаёт количество цифр, отображаемых в числах с плавающей запятой после запятой.
<i>streamsize width(streamsize _Wide);</i> <i>streamsize width() const;</i>	Задаёт и возвращает ширину выводимых символов в потоке (ширину поля)
<i>fmtflags setf(fmtflags _Mask);</i> <i>fmtflags setf(fmtflags _Mask, fmtflags _Unset);</i>	Устанавливает флаг форматирования <i>_Mask</i> и сбрасывает флаг форматирования <i>_Unset</i> . Возвращает предыдущие флаги форматирования
<i>void unsetf(fmtflags _Mask);</i>	Сбрасывает указанный флаг форматирования

Эти функции вызываются для потоковых объектов обычным способом — через точку. Например, чтобы установить ширину поля 14, можно написать:

```
cout.width(14);
```

Следующее выражение делает символом заполнения звездочку (как при тестировании печати):

```
cout.fill('*');
```

Можно использовать некоторые функции, чтобы манипулировать напрямую установкой флагов форматирования. Например, так можно установить выравнивание по левому краю:

```
cout.setf(ios::left);
```

Чтобы сбросить выравнивание по левому краю, необходимо написать:

```
cout.unsetf(ios::left);
```

Версия функции *setf()* с двумя параметрами (сочетания вариантов которой показаны в таблице 5) использует второй из параметров для сбрасывания всех флагов указанного типа. При этом установится флаг, указанный в качестве первого аргумента. Например, оператор

```
cout.setf(ios::left, ios::adjustfield);
```

сбрасывает все флаги, связанные с выравниванием текста, а затем устанавливает флаг *left* для выравнивания по левому краю.

Таблица 5 – Аргументы функции *setf()* с двумя параметрами

Первый аргумент: устанавливаемые флаги	Второй аргумент: сбрасываемые флаги
<i>dec, oct, hex</i>	<i>basefield</i>
<i>left, right, internal</i>	<i>adjustfield</i>
<i>scientific, fixed</i>	<i>floatfield</i>

С использованием указанных подходов можно создать способ форматированного ввода/вывода, связанного с клавиатурой, экраном, файлами.

## КЛАСС BASIC\_ISTREAM

Класс *basic\_istream* выполняет ввод, извлечение данных из потока ввода. Необходимо четко разграничивать извлечение данных из потока и связанное с ним, но противоположное действие по выводу данных в поток — вставку. Рисунок 2 показывает разницу между вводом данных и их выводом.

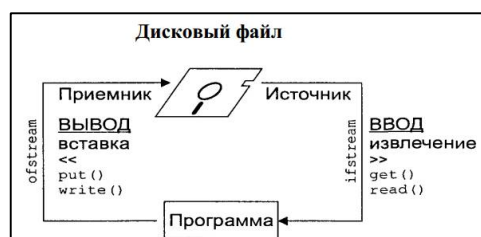


Рисунок 2 – Схема ввода и вывода данных

В таблице 6 собраны методы класса *basic\_istream*, предназначенные для реализации потокового ввода данных.

Таблица 6 – Методы класса *basic\_istream*

Функция	Назначение
---------	------------

<b>operator &gt;&gt;</b>	Считывает форматированные данные (стандартные и те, для которых он перегружен) из входного потока
<i>basic_istream</i> <Char_T, Tr> & <b>get</b> (Char_T& Ch);	Считывает один или несколько символов из входного потока в параметр <i>Ch</i>
<i>basic_istream</i> <Char_T, Tr> & <b>get</b> (Char_T* str, streamsize count);	Считывает символы в массив <i>str</i> до ограничителя '\n'. Параметр <i>count</i> определяет количество символов для чтения из потока
<i>basic_istream</i> <Char_T, Tr> & <b>get</b> (Char_T* str, streamsize count, Char_T delimiter);	Считывает символы в массив <i>str</i> до ограничителя <i>delimiter</i> . Параметр <i>count</i> определяет количество символов для чтения из потока
<i>basic_istream</i> <Char_T, Tr> & <b>getline</b> (char_type* str, streamsize count);	Считывает строку из входного потока в массив <i>str</i> . Параметр <i>count</i> определяет количество символов для чтения из потока
<i>basic_istream</i> <Char_T, Tr> & <b>getline</b> (char_type* str, streamsize count, char_type delimiter);	Считывает строку из входного потока в массив <i>str</i> до ограничителя <i>delimiter</i> . Параметр <i>count</i> определяет количество символов для чтения из потока
<i>basic_istream</i> <Char_T, Tr> & <b>putback</b> (char_type Ch);	Помещает указанный символ в обратно во входной поток
<i>basic_istream</i> <Char_T, Tr> & <b>ignore</b> (streamsize count = 1, int_type delimiter = traits_type::eof());	Вызывает пропуск ряда элементов из текущей позиции чтения. Т.е. извлекает и удаляет из входного потока до <i>count</i> числа символов или до ограничителя <i>delimiter</i> включительно (обычно '\n'). С извлеченными данными ничего не делает
<i>int_type peek</i> ();	Возвращает следующий символ для чтения. Т.е. читает один символ, оставляя его в потоке
<i>streamsize gcount</i> () const;	Возвращает количество символов, прочитанных во время последнего неформатированного ввода.
<i>basic_istream</i> <Char_T, Tr> & <b>read</b> (char_type* str, streamsize count);	Считывает заданное количество символов из потока и сохраняет их в массиве. Этот метод потенциально небезопасен, так как он полагается на вызывающий блок для проверки правильности переданных значений. Параметр <i>count</i> определяет максимальное количество символов для чтения из потока
<i>basic_istream</i> <Char_T, Tr> & <b>seekg</b> (pos_type pos);	Перемещает позицию чтения во входном потоке. Параметр <i>pos</i> определяет абсолютное положение (в байтах), на которое перемещается указатель чтения
<i>basic_istream</i> <Char_T, Tr> & <b>seekg</b> (off_type off, ios_base::seekdir way);	Перемещает позицию чтения во входном потоке. Параметр <i>off</i> определяет смещение (в байтах) для перемещения указателя чтения относительно параметра <i>way</i> . Параметр <i>way</i> может иметь одно значений перечисления <i>ios_base::seekdir</i> ( <i>ios::beg</i> , <i>ios::cur</i> , <i>ios::end</i> )
<i>pos_type tellg</i> ();	Сообщает текущую позицию (в байтах) чтения во входном потоке от начала файла

## КЛАСС BASIC\_OSTREAM

Класс *basic\_ostream* предназначен для вывода (вставки) данных в поток. Таблица 7 содержит основные методы класса *basic\_ostream*.

Таблица 7 – Функции класса *basic\_ostream*

Функция	Назначение
<b>operator &lt;&lt;</b>	Форматированная вставка данных стандартных типов и



	типов, для которых перегружен оператор вставки
<i>basic_ostream</i> <Elem, Tr>& <b>put</b> (char_type Ch);	Помещает символ в <i>Ch</i> в поток
<i>basic_ostream</i> <Elem, Tr>& <b>flush</b> ();	Очищает буфер
<i>basic_ostream</i> <Elem, Tr>& <b>write</b> (const char_type* str, streamsize count);	Помещает символы из массива <i>str</i> в поток. Параметр <i>count</i> определяет количество символов, которые нужно ввести в поток
<i>basic_ostream</i> <Elem, Tr>& <b>seekp</b> (pos_type Pos);	Устанавливает позицию в выходном потоке. Позиция указывается относительно начала файла
<i>basic_ostream</i> <Elem, Tr>& <b>seekp</b> (off_type _Off, ios_base::seekdir _Way);	Устанавливает позицию в выходном потоке. Параметр <i>_Off</i> определяет смещение относительно параметра <i>_Way</i> . Параметр <i>_Way</i> может иметь одно значений перечисления <i>ios_base::seekdir</i> ( <i>ios::beg</i> , <i>ios::cur</i> , <i>ios::end</i> )
pos_type <b>tellp</b> ();	Возвращает позицию указателя в выходном потоке

## ОБЪЕКТЫ ГЛОБАЛЬНЫХ ПОТОКОВ

Предопределенные потоковые объекты *cin* и *cout* по умолчанию связаны с клавиатурой и монитором соответственно. Объект *cin* определяет глобальный поток ввода и управляет извлечением из стандартного входного потока в виде потока байтов. Объект *cout* определяет глобальный поток вывода и управляет вставками в стандартный вывод в виде потока байтов.

Еще двумя предопределенными объектами, представляющими потоки вывода, являются *cerr* и *clog*. Объект *cerr* определяет глобальный поток вывода ошибок. Объект *cerr* управляет вставками без буфера в стандартный поток вывода ошибок в виде потока байтов. Объект *clog* определяет глобальный поток вывода ошибок с буферизацией. Объект *clog* управляет буферизованными вставками в стандартный поток вывода ошибок в виде потока байтов.

Объект *cerr* часто используется для вывода сообщений об ошибках и программной диагностики. Поток данных, отправленный в *cerr*, немедленно выводится на экран, минуя буферизацию. Этим *cerr* отличается от *cout*. К тому же этот поток не может быть перенаправлен. Объект, *clog*, похож на *cerr* в том, что также не может быть перенаправлен. Но его вывод проходит буферизацию.

## ОШИБКИ ПОТОКОВ

Рассмотрим исключительную ситуацию, когда пользователь вводит строчку «девять» вместо числа 9. При этом в потоке устанавливается состояние, свидетельствующее об ошибке. Флаги статуса ошибок потоков определяет компонент *iostate*, который сообщает об ошибках, произошедших при операциях ввода/вывода. Все эти флаги собраны в таблице 8.

Таблица 8 – Флаги статуса ошибок в потоке

Название	Значение
<i>badbit</i>	Определяет потерю целостности буфера потока
<i>eofbit</i>	Определяет конец файла при извлечении данных из потока
<i>failbit</i>	Определяет ошибку извлечения допустимого поля из потока
<i>goodbit</i>	Все биты состояния потока нулевые. Ошибок нет (флаги не установлены)

У любого потока есть набор флагов, с помощью которых можно следить за состоянием потока, т.е. определять есть ли в нем ошибки. Для

чтения и установки флагов потока могут использоваться различные функции, показанные в таблице 9.

Таблица 9 – Методы класса *basic\_ios*, предназначенные для анализа и установки флагов ошибок в потоке

Функция	Назначение
<i>bool eof() const</i>	Указывает, достигнут ли конец потока. Возвращает <i>true</i> , если достигнут конец потока, иначе возвращает <i>false</i>
<i>bool fail() const</i>	Указывает на невозможность извлечения допустимого поля из потока. Возвращает <i>true</i> , если операция завершилась неудачно, например не удалось открыть файл или передан некорректный формат ввода, иначе возвращает <i>false</i>
<i>bool good() const</i>	Указывает находится ли поток находится в состоянии без ошибок. Возвращает <i>true</i> , если поток находится в состоянии без ошибок, иначе возвращает <i>false</i>
<i>bool bad() const</i>	Указывает на потерю целостности буфера потока. Возвращает <i>true</i> , если произошла потеря целостности буфера потока, иначе возвращает <i>false</i>
<i>void clear(iostate state = goodbit, bool reraise = false)</i> <i>void clear(io_state state)</i>	Удаляет все флаги ошибок. При использовании перегрузки метода без аргумента снимает все флаги ошибок, в противном случае устанавливает указанный флаг, например <i>clear(ios::failbit)</i> . Параметр <i>reraise</i> указывает, следует ли повторно вызывать исключение

В следующем примере проверяется значение, возвращаемое функцией *good()* и сообщается об ошибке, если её значение не равно *true*, так же дается возможность пользователю повторно ввести корректные данные.

```
//Пример №3. Анализ состояния потока
#include <iostream>
using namespace std;
void Ex3() {
    int i;
    while (true) {
        cout << "Введите целое число: ";
        cin >> i;
        if (cin.good()) { //если нет ошибок
            break;
        } //выйти из цикла
        cin.clear(); //Очистить биты ошибок
        cout << "Неправильный ввод данных\n";
        cin.ignore(10, '\n'); //удалить из потока 10 символов до
знака перехода на новую строку
    }
    cout << "целое число: " << i;
}
```

Результаты работы программы:

```
Введите целое число: три
Неправильный ввод данных
Введите целое число: два
Неправильный ввод данных
Введите целое число: пять
Неправильный ввод данных
Введите целое число: 3456
целое число: 3456Press any key to continue . . .
```

Часто встречается ошибка ввода не цифр, а каких-либо символов. Это приводит к установке в потоке флага *failbit*. Но могут определяться и другие ошибки, например, при считывании данных из файла формат считанных данных может не соответствовать ожидаемому. Числа в формате с плавающей запятой (*float*, *double*, *long double*) могут анализироваться на предмет неправильного ввода так же, как и целые числа. Так же могут встречаться ошибки превышения допустимого количества символов при считывании. Обычно это происходит при передаче входного потока с ошибками. Лишние символы все еще остаются в потоке после того, как ввод уже считается завершенным. Затем они передаются в следующую операцию ввода данных, хотя для этого не предназначены. Часто в конце вводимых данных остается символ новой строки, символ табуляции. Чтобы избежать случайного извлечения из потока лишних символов, используется метод *ignore()* класса *istream*.

```
basic_istream<Char_T, Tr>& ignore(
    streamsize count = 1,
    int_type delimiter = traits_type::eof());
```

Параметр *count* определяет число элементов, которые нужно пропустить с текущей позиции чтения. Если параметр *delimiter* встречается до *count*, то метод *ignore()* прекращает пропуск символов и позволяет читать из входного потока все элементы после разделителя. Таким образом, метод *ignore()* считывает и игнорирует вплоть до *count* символов, включая указанный ограничитель *delimiter*. Например, выражение

```
cin.ignore(10, '\n');
```

приводит к считыванию до 10 символов, включая ограничитель '\n', и удалению их из входного потока.

## ВВОД ПРИ ОТСУТСТВИИ ДАННЫХ

Символы, в определенных ситуациях не несущие смысловой нагрузки, (пробелы, '\n', табуляция) могут пропускаться при вводе данных. Это может привести к некоторым нежелательным побочным эффектам. Например, пользователь вместо ввода значения может случайно нажать *Enter*. Приведенный ниже код

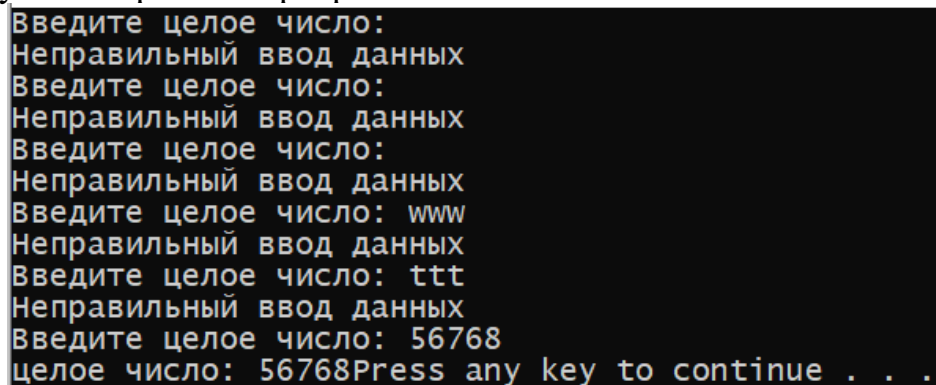
```
cin >> i;
```

после нажатия клавиши *Enter* приведет к переводу курсора на новую строку, в то время как поток все еще будет ожидать ввода данных. Дело в том, что, не увидев адекватной реакции на свои действия, пользователь может подумать, что компьютер вышел из строя. Кроме того, обычно

повторные нажатия клавиши *Enter* приводят к тому, что курсор сдвигается все ниже и ниже, пока не дойдет до конца экрана, после чего экран начинает прокручиваться вверх. Если подобная ситуация произойдет в программе, в которой используется графика, основанная на тексте, прокручивание окна приведет к неразберихе на экране. Таким образом, важно иметь возможность дать команду входному потоку игнорировать или не игнорировать разделители. Это делается с помощью флага *skipws*.

```
//Пример №4. Сбрасывание флага skipws при считывании чисел
#include <iostream>
using namespace std;
void Ex4() {
    int i;
    while (true) {
        cout << "Введите целое число: ";
        cin.unsetf(ios::skipws); //не игнорировать разделители
        cin >> i;
        if (cin.good()) { //если нет ошибок
            break;
        } //выйти из цикла
        cin.clear(); //Очистить биты ошибок
        cout << "Неправильный ввод данных\n";
        cin.ignore(10, '\n'); //Удалить разделитель строк
    }
    cout << "целое число: " << i; // целое без ошибок
}
```

Результаты работы программы:



```
Введите целое число:
Неправильный ввод данных
Введите целое число:
Неправильный ввод данных
Введите целое число:
Неправильный ввод данных
Введите целое число: www
Неправильный ввод данных
Введите целое число: ttt
Неправильный ввод данных
Введите целое число: 56768
целое число: 56768Press any key to continue . . .
```

Если теперь пользователь и нажмет *Enter*, забыв ввести данные, то будет установлен флаг *failbit* и тем самым сгенерирован признак ошибки. После этого можно попросить пользователя ввести данные повторно.

```
//Пример №5. Использование флага skipws при считывании строк
#include <iostream>
#include <sstream>
using namespace std;
int main() {
    string s1, s2, s3;
    std::istringstream ("Phrases for Anywhere")>> s1 >> s2 >> s3;
    std::cout << "Default behavior: s1=" << s1 << ",s2=" << s2 <<
    ",s3=" << s3 << '\n';
}
```

```

std::istream ("Different Phrases for")>> std::noskipws >>
s1 >> s2 >> s3;
std::cout << "noskipws behavior: s1=" << s1 << ",s2=" << s2 <<
",s3=" << s3 << '\n';
return 0;
}

```

Результаты работы программы:

```

Default behavior: s1=Phrases,s2=for,s3=Anywhere
noskipws behavior: s1=Different,s2=,s3=Anywhere
Press any key to continue . . .

```

В этом примере при извлечении первой строки «*Phrases for Anywhere*» все строки *s1*, *s2*, *s3* записываются корректно.

При чтении второй строки «*Different Phrases for*» часть строки "*Different*" будет извлечена в переменную *s1*. Для второго извлечения ничего не будет извлечено, так как установлен флаг формата *std::noskipws*, отключающий очистку ведущего *whitespace*. Из-за этого строка очищается, а затем извлечение завершается неудачей, потому что никакие символы не были введены. Когда поток определяет неудачное извлечение, флаг *std::ios\_base::failbit* устанавливается в состоянии потока, указывая на ошибку. С этого момента все попытки ввода-вывода потерпят неудачу, если состояние потока не будет очищено. Экстрактор становится неработоспособным, и он не будет работать, если состояние потока не очищено от всех его ошибок. Это означает, что извлечение в переменную *s3* ничего не делает и сохраняет значение, которое было у него при предыдущем извлечении (без *std::noskipws*), потому что поток не очистил строку.

### ПРИМЕР ПРОГРАММЫ С ПРОВЕРКОЙ ОШИБОК ПОТОКА

Рассмотрим программу, в которой ввод пользователем данных в объекты класса *Distance* проверяется на наличие ошибок. Программа будет получать от пользователя данные в футах и дюймах и выводить их на экран. Если же пользователь ошибается при вводе данных, то нужно отклонить введенные данные, объяснить ему, где он ошибся, и попросить ввести данные заново.

Программа проста, за исключением того, что метод *getDist()* направлен на поддержку обработки ошибок. Так же введено выражение, которое следит за тем, чтобы пользователь не вводил футы в формате чисел с плавающей запятой. Это важно, так как дюймы могут иметь дробную часть, а футы — нет, и пользователь может слегка запутаться.

Обычно, ожидая ввод целочисленных значений, оператор извлечения просто прерывает свою работу, увидев точку в десятичной дроби. При этом сообщение об ошибке не выдается. При необходимости узнать о совершенной ошибке можно считывать значение числа футов не в виде *int*, а в виде строки. Затем можно проверить введенную строку при помощи функции *isFeet()*, возвращающей *true*, если число футов введено корректно. Для одобрения функцией введенное число должно содержать только цифры, и его значение должно лежать в пределах от -999 до 999 (предполагаем, что более далекие расстояния измерять не будем). Если с введенными футами все в порядке, то конвертируем строку в *int* с помощью стандартной

библиотечной функции *atoi()*. Число дюймов может быть дробным. Значения дюймов должны лежать в пределах от 0 до 12.0. Кроме того, будем отслеживать наличие флагов ошибок *ios*. В общем случае будет установлен *failbit* при вводе каких-либо символов вместо числа.

```
//Пример №6. Контроль ввода данных с клавиатуры
#include <iostream>
#include <string>
#include <cstdlib> //для функций atoi(), atof()
using namespace std;
int isFeet(string);
class Distance { // Класс английских расстояний
    int feet;
    float inches;
public:
    Distance() { feet = 0; inches = 0.0; }
    Distance(int feet, float inches) { this->feet = feet; this-
>inches = inches; }
    void showDist() { cout << feet << "'-" << inches << "'"; }
    void getDist(); //метод получения длины у пользователя
};
void Distance::getDist() {
    string inStr;
    while (true) { // цикл проверки футов
        cout << "Введите количество футов: ";
        cin.unsetf(ios::skipws); // не пропускать разделители
        cin >> inStr; //получить футы как строку
        if (isFeet(inStr)) { // правильное ли введенное значение?
            cin.ignore(10, '\n'); // убрать лишние символы из
потокa ввода
            feet = atoi(inStr.c_str()); //перевести строку в целое
число
            break; //выход из цикла 'while'
        }
        cin.ignore(10, '\n');
        cout << "Футы должны быть целым числом и меньше 1000\n";
    }
    while (true) { // цикл проверки дюймов
        cout << "Введите количество дюймов: ";
        cin.unsetf(ios::skipws); // не пропускать разделители
        cin >> inches; //получить дюймы как тип float
        if (inches >= 12.0 || inches < 0.0) {
            cout << "Дюймы должны быть между 0.0 и 11.99\n";
            cin.clear(ios::failbit); //установить флаг ошибки
        }
        if (cin.good()) {
            cin.ignore(10, '\n');
            break; //Ввод корректный, выйти из 'while'
        }
        cin.clear(); //ошибка ввода; очистить статус ошибки
        cin.ignore(10, '\n');
        cout << "Неверно введены дюймы\n";
    }
}
```



```

int isFeet(string inStr) {
    int inStrLength = inStr.size();//получить длину строки
    if (inStrLength == 0 || inStrLength > 5)
        return 0;
    for (int j = 0; j < inStrLength; j++)// проверка символов строки
        if ((inStr[j] < '0' || inStr[j] > '9'))
            return 0;//строка неправильных футов
    float feetValue = atof(inStr.c_str());//перевод строки в double
    if (feetValue < -999.0 || feetValue > 999.0)
        return 0; //неправильное значение футов
    return 1;//правильное значение футов
}
void Ex6() {
    Distance distance;//создать объект Distance
    char userData;
    do {
        distance.getDist();//получить его значение
        cout << "\nРасстояние равно ";
        distance.showDist();//вывести значение полученного
расстояния
        cout << "\nЕще раз (y/n)? ";
        cin >> userData;
        cin.ignore(10, '\n');
    } while (userData != 'n');
}

```

Результаты работы программы:

```

Введите количество футов: 1000
Футы должны быть целым числом и меньше 1000
Введите количество футов: 1000.5
Футы должны быть целым числом и меньше 1000
Введите количество футов: 990
Введите количество дюймов: 23.23
Дюймы должны быть между 0.0 и 11.99
Неверно введены дюймы
Введите количество дюймов: 11.25

Расстояние равно 990'-11.25"
Еще раз (y/n)? y
Введите количество футов: 0
Введите количество дюймов: 0

Расстояние равно 0'-0"
Еще раз (y/n)? n
Press any key to continue . . .

```

В программе установлен флаг ошибки вручную. Тем самым проверено, входят ли введенные дюймы в диапазон допустимых значений. Если нет, то устанавливается флаг *failbit*. При проверке на наличие ошибок с помощью функции *cin.good()* флаг *failbit* будет обнаружен и выведется сообщение об ошибке.

### КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ:

1. Что такое поток в языке C++?
2. Какой класс является базовым для большинства потоковых классов?
3. Истинно ли утверждение “потоки делятся на входные и выходные”?
4. Что означает флаг *skipws* при его использовании с объектом *cin*?
5. Напишите сигнатуру функции *main()*, позволяющую программе использовать аргументы командной строки.

6. Что собой представляет стандартный поток ввода и вывода?
7. Какой заголовочный файл необходимо подключить для работы с потоками ввода/вывода?
8. Что представляет собой поток форматированного вывода? Какие функции обеспечивают форматированный вывод?
9. Для чего предназначена библиотеки ввода/вывода *iomanip*?
10. Напишите оператор, который позволит считать с консоли строку данных?

### **ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ:**

1. Изучить теоретические сведения, полученные на лекции и лабораторной работе, ознакомиться с соответствующими материалами литературных источников.
2. Ответить на контрольные вопросы лабораторной работы.
3. Разработать алгоритм программы по индивидуальному заданию.
4. Написать, отладить и проверить корректность работы созданной программы.

5. Написать электронный отчет по выполненной лабораторной работе.

Отчет должен быть оформлен по стандарту БГУИР ([Стандарт предприятия СТП 01-2017 "Дипломные проекты \(работы\). Общие требования"](#)) и иметь следующую структуру:

1. титульный лист
2. цель выполнения лабораторной работы
3. теоретические сведения по лабораторной работе
4. формулировка индивидуального задания
5. весь код решения индивидуального задания, разбитый на необходимые типы файлов
6. скриншоты выполнения индивидуального задания
7. выводы по лабораторной работе

**В РАМКАХ ВСЕГО КУРСА «ООП» ВСЕ ЛАБОРАТОРНЫЕ РАБОТЫ НА ЯЗЫКЕ C++ ДОЛЖНЫ ХРАНИТЬСЯ В ОДНОМ РЕШЕНИИ (SOLUTION), В КОТОРОМ ДОЛЖНЫ БЫТЬ СОЗДАНЫ ОТДЕЛЬНЫЕ ПРОЕКТЫ (PROJECTS) ДЛЯ КАЖДОЙ ЛАБОРАТОРНОЙ РАБОТЫ. ВО ВСЕХ ПРОЕКТАХ ПОЛЬЗОВАТЕЛЬ ДОЛЖЕН САМ РЕШАТЬ ВЫЙТИ ИЗ ПРОГРАММЫ ИЛИ ПРОДОЛЖИТЬ ВВОД ДАННЫХ. ВСЕ РЕШАЕМЫЕ ЗАДАЧИ ДОЛЖНЫ БЫТЬ РЕАЛИЗОВАНЫ, ИСПОЛЬЗУЯ НЕОБХОДИМЫЕ КЛАССЫ И ОБЪЕКТЫ.**

### **ВАРИАНТЫ ИНДИВИДУАЛЬНЫХ ЗАДАНИЙ К ЛАБОРАТОРНОЙ РАБОТЕ №3:**

На основе разработанной иерархии классов, реализованной в лабораторной работе №2 «ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ В ЯЗЫКЕ C++», реализовать программу для работы с данными, используя потоки ввода-вывода. Реализовать функций просмотра, добавления, удаления, редактирования, сортировки, поиска данных по необходимым параметрам. Использовать 5 флагов форматирования. Использовать функции *setf()* и *unsetf()* для установки и сбрасывания флагов. Использовать функции *eof()*, *fail()*, *good()*, *bad()*, *clear()* для проверки состояния потока.