Форматирование

Форматы ввода-вывода зависят в основном от двух факторов. Первый – специальные форматные флаги, другой – их настройка под национальные стандарты.

Класс **ios_base** содержит ряд переменных, предназначенных для определения форматов ввода-вывода. В частности, эти переменные определяют минимальную ширину поля, точность вывода вещественных чисел и заполнитель. Переменная типа **ios::fmtflags** содержит флаги конфигурации, которые определяют, нужно ли выводить знак перед положительными числами, должны ли логические значения выводиться в числовом или символьном виде и т.д.

Некоторые форматные флаги составляют группы. Определены специальные маски, упрощающие работу с группами.

Имеются три стандартные маски:

```
adjustfield = internal | left | right
basefield = dec | oct | hex
floatfield = fixed | scientific
```

Форматирование - функции

Функции определения форматов

Функция	*Описание*
setf(флаги)	Устанавливает флаги, переданные в аргументе, в качестве дополнительных форматных флагов и возвращает предыдущее состояние всех флагов
setf(флаги,маска)	Устанавливает флаги, переданные в первом аргументе, в качестве форматных флагов для группы, которая идентифицируется маской, переданной во втором аргументе, и возвращает предыдущее состояние всех флагов
unsetf(флаги)	Сбрасывает флаги, переданные в аргументе
flags()	Возвращает весь набор форматных флагов
flags(флаги)	Устанавливает флаги, переданные в аргументе, в качестве новых форматных флагов и возвращает предыдущее состояние всех флагов
copyfmt(поток)	Копирует все определения форматов из потока, переданного в аргументе

Форматирование - флаги

Флаги форматирования

Флаг	**Назначение**
skipws	при вводе пробельные литеры пропускаются
left	выводимые данные выравниваются по левому краю с дополнением символами-заполнителями по ширине поля
right	выводимые данные выравниваются по правому краю с дополнением символами-заполнителями по ширине поля (установлен по умолчанию)
internal	при выравнивании символы-заполнители вставляются между символом знака или префиксом основания системы счисления и числом
dec	целые числа выводятся по основанию 10 (установлен по умолчанию); устанавливается также манипулятором dec
oct	целые числа выводятся по основанию 8; устанавливается также манипулятором oct
hex	целые числа выводятся по основанию 16; устанавливается также манипулятором hex

Форматирование – флаги (продолжение)

Флаги форматирования

Флаг	**Назначение**
showbase	при выводе целых чисел отображается префикс, указывающий на основание системы счисления
showpoint	при выводе чисел с плавающей запятой всегда отображается десятичная точка, а хвостовые нули не отбрасываются
uppercase	шестнадцатеричные цифры от A до F, а также символ экспоненты E отображаются в верхнем регистре
showpos	при выводе положительных чисел отображается знак плюс
scientific	числа с плавающей запятой отображаются в научном формате (с экспонентой)
fixed	числа с плавающей запятой отображаются в фиксированном формате (без экспоненты)
unitbuf	при каждой операции вывода буфер потока должен очищаться
stdio	при каждой операции вывода буферы потоков **stdout** и **stderr** должны очищаться

Манипуляторы работы с флагами

Установка и сброс флагов форматирования можно осуществлять при помощи пары манипуляторов:

setiosflags(флаги) – установка передаваемых флагов

resetiosflags(маска) – сброс флагов, определяемых передаваемой маской.

Манипуляторы setiosflags() и resetiosflags() дают возможность соответственно установить или сбросить один или несколько флагов в командах записи или чтения с использованием оператора << или >>.

```
#include <iostream>
#include <iomanip>
. . .
std::cout << resetiosflags(std::ios::adjustfield) ;   // Сброс выравнивания
std::cout << setiosflags(std::ios::left) ;            // Левое выравнивание
```

Форматированный ввод-вывод логических данных

Флаг **boolalpha** определяет формат ввода и вывода логических значений – числовой или текстовый. Если флаг не установлен (значение по умолчанию), логические данные представляются в числовом виде. В этом случае *false* всегда представляется значением 0, а *true* -значением 1. При чтении логических данных в числовом представлении наличие символов, отличных от 0 и 1, считается ошибкой (для потока данных устанавливается бит failbit).

При установке флага логические данные читаются и записываются в текстовом представлении. При чтении логического значения строка должна соответствовать текстовому представлению *true* или *false*. Строки, представляющие эти значения, определяются состоянием объекта локального контекста.

Стандартный объект локального контекста "C" использует для представления логических значений строки "true" и "false". Для удобства работы с этим флагом определены специальные манипуляторы:

boolalpha	Включает текстовое представление (установка флага *ios::boolalpha*)
noboolalpha	Включает числовое представление (сброс флага *ios::boolalpha*)

Следующий фрагмент выводит переменную **b** сначала в числовом, а затем в текстовом представлении:

```
bool b;
. . .
cout << noboolalpha << b  << " " << boolalpha << b << endl;
```

Ширина поля

Функция	*Описание*
width()	Возвращает текущую ширину поля
width(val)	Задает ширину поля равной **val** и возвращает предыдущую ширину поля

При выводе функция **width()** определяет минимальную ширину поля. Определение относится только к следующему выводимому форматированному полю. При вызове без аргументов **width()** возвращает текущую ширину поля. При вызове с целочисленным аргументом функция **width()** изменяет ширину поля и возвращает ее предыдущее значение.

Ширина поля не может использоваться для сокращения вывода. То есть максимальную ширину поля задать невозможно. Вместо этого ее придется самостоятельно запрограммировать, например, записав данные в строку и ограничив вывод определенным количеством символов.

По умолчанию минимальная ширина равна 0; это означает, что размер поля может быть произвольным.

После выполнения любой операции форматированного ввода-вывода восстанавливается ширина поля по умолчанию.

Заполнитель

Функция	*Описание*
fill()	Возвращает текущий заполнитель
fill(c)	Назначает новый и возвращает предыдущий заполнитель

Функция **fill()** определяет символ, используемый для заполнения промежутков между отформатированным представлением величины и позицией, отмечающей минимальную ширину поля.

По умолчанию заполнителем является пробел.

Заполнитель остается без изменений до тех пор, пока он не будет модифицирован явно.

Выравнивание

Тип выравнивания данных внутри поля определяется тремя флагами, перечисленными в таблице. Эти флаги определяются в классе **ios_base** вместе с соответствующей маской.

Маска	Флаг	Описание
adjustfield	left	Выравнивание по левому краю
	right	Выравнивание по правому краю
	internal	Выравнивание знака по левому краю, а значения - по правому
нет		Выравнивание по правому краю (по умолчанию)

Заполнитель остается без изменений до тех пор, пока он не будет модифицирован явно.

Заполнитель – знак подчеркивания, ширина поля - 6

	-53	0.12	sss
left	**-53___**	**0.12___**	**sss___**
right	**___-53**	**___0.12**	**___sss**
internal	**-___53**	**___0.12**	**___sss**

Манипуляторы ширины, заполнения и выравнивания

Манипулятор	*Описание*
setw(val)	Устанавливает ширину поля при вводе и выводе равной val (соответствует функции width())
setfill(c)	Назначает заполнителем символ c (соответствует функции fill())
left	Устанавливает выравнивание по левому краю
right	Устанавливает выравнивание по правому краю
internal	Устанавливает выравнивание знака по левому краю, а значения - по правому

Манипуляторам **setw()** и **setfill()** должен передаваться аргумент, поэтому для их использования в программу необходимо включить заголовочный файл <iomanip>.

Пример использования манипуляторов

```
#include <iostream>
#include <iomanip>

std::cout << std::setw(8) << std::setfill('_') << -3.14
          << '  ' << 42 << std::endl;
std::cout << std::setw(8) << "sum: " << std::setw(8) << 42 << std::endl;
```

Этот фрагмент выводит следующий результат:

```
___-3.14 42
____sum: ______42
```

Использование ширины поля при вводе

Ширина поля также позволяет задать максимальное количество символов, вводимых при чтении последовательностей символов типа char*. Если значение width() отлично от 0, то из потока данных читаются не более, width() -1 символ.

Поскольку обычные С-строки не могут увеличиваться при чтении данных, при их чтении оператором >> всегда следует ограничивать максимальный размер ввода функциями width() или setw(). Пример:

```
char buffer[81]:                    // Чтение не более 80 символов
cin >> setw(sizeof(buffer)) >> buffer;
```

Функция читает не более 80 символов, хотя sizeof(buffer) возвращает 81, поскольку один символ является признаком завершения строки (он присоединяется автоматически). Обратите внимание на распространенную ошибку:

```
char* s:
cin >> setw(sizeof(s)) >> s: // ОШИБКА ВРЕМЕНИ ВЫПОЛНЕНИЯ
```

Строковые классы позволяют предотвратить подобные ошибки:

```
string buffer:
cin >> buffer: // ОК
```

Отображение знака для положительных чисел

Установка флага **ios::showpos** означает, что положительные числа должны выводиться со знаком.

Если флаг сброшен, то со знаком выводятся только отрицательные числа.

По умолчанию положительные числа выводятся без знака.

Установка и сброс флага может быть осуществлен при помощи манипуляторов **showpos** и **noshowpos** соответственно.

```
std::cout << 123.9 << std::endl;
std::cout.setf (std::ios::showpos);
std::cout << 123.9 << std::endl;
```

Этот фрагмент выводит следующий результат:

```
123.9
+123.9
```

Система счисления

Следующая группа из трех флагов управляет основанием системы счисления, используемой при вводе-выводе целых чисел. Флаги определяются в классе **ios_base** вместе с соответствующей маской.

Маска	Флаг	Описание
basefield	oct	Чтение/запись в восьмеричной системе
	dec	Чтение/запись в десятичной системе (используется по умолчанию)
	hec	Чтение/запись в шестнадцатеричной системе
нет		Запись в шестнадцатеричной системе, чтение в зависимости от

Смена основания системы счисления отражается на дальнейшем вводе-выводе всех целых чисел до следующего изменения флагов. По умолчанию используется десятичный формат.

Поддержка двоичной записи не предусмотрена, однако чтение и запись целых чисел в двоичном виде может осуществляться при помощи класса **bitset**.

Флаги системы счисления также распространяются на ввод. Данные читаются в системе, определяемой установкой одного из флагов. Если флаги не установлены, то при чтении основание системы счисления определяется по префиксу: 0x или 0X - интерпретируется как шестнадцатеричное; префикс 0 является признаком восьмеричной записи.

Во всех остальных случаях число считается десятичным.

Установка системы счисления

Существуют два основных способа переключения флагов системы счисления.

- Сброс одного флага и установка другого:

```
std::cout.unsetf (std::ios::dec);
std::cout.setf (std::ios::hex);
```

- Установка одного флага с автоматическим сбросом остальных флагов группы:

```
std::cout.setf(std::ios::hex, std::ios::basefield);
```

Для удобства программирования существуют и специальные манипуляторы oct, hex и dec:

```
int x, y, z;
std::cout << std::ios::hex << x << std::endl;
std::cout << y << ' ' << std::ios:dec << z <<
std::endl;
```

Этот фрагмент выводит **x** и **y** в 16-ой, а **z** - в 10-ой системе.

Идентификация системы счисления

Дополнительный флаг **showbase** выводит числа по стандартным правилам обозначения системы счисления числовых литералов в C/C++.

При установке флага **ios::showbase** восьмеричные числа выводятся с префиксом 0, а шестнадцатеричные числа - с префиксом 0x (или при установленном флаге **ios::uppercase** - 0X).

Установка и сброс флага **ios::showbase** может быть выполнена с использованием специальных манипуляторов **showbase** и **noshowbase**.

```
std::cout << 127 << ' ' <<255<< std::endl;
std::cout << std::hex <<127 << ' ' <<255 << std::endl;
std::cout.setf(std::ios::showbase);
std::cout << 127 << ' ' << 255 << std::endl;
std::cout.setf(std::ios::uppercase);
std::cout << 127 << ' ' << 255 << std::endl;
```

Этот фрагмент выводит следующий результат:

```
127  255
7f ff
0x7f 0xff
0X7F
0XFF
```

Вывод символов в верхнем регистре

Флаг **ios::uppercase** означает, что буквы в числовых значениях должны выводиться в верхнем регистре. Этот флаг распространяется как на целые числа, записанные в шестнадцатеричном виде, так и на вещественные числа в научной (экспоненциальной) записи.

По умолчанию символы выводятся в нижнем регистре.

Установка и сброс флага может быть осуществлен при помощи манипуляторов **uppercase** и **nouppercase** соответственно.

```
std::cout <<(std::hex) << 0xabc << ' ' << 1.2
  << ' ' << (std::scientific) << 1.2 << std::endl;
std::cout.setf (std::ios::uppercase);
std::cout << 0xabc << ' ' << 1.2 << std::endl;
Результат:
abc 1.2 1.200000e+000
ABC 1.2 1.200000E+000
```

Формат вещественных чисел

Некоторые флаги и переменные управляют выводом вещественных чисел.

Флаги, перечисленные в таблице, определяют тип записи (десятичная или научная). Эти флаги определяются в классе **ios_base** вместе с соответствующей маской.

Маска	*Флаг*	*Описание*
floatfield	fixed	Использование десятичной записи
	scientific	Использование научной записи
нет		Использование «лучшей» из этих двух записей

Можно управлять точностью представления числа при помощи функции **precision()**. При использовании научной записи функция **precision()** определяет количество десятичных разрядов в дробной части. Остаток всегда округляется. Вызов **precision()** без аргументов возвращает текущую точность. При вызове с аргументом функция **precision()** устанавливает заданную точность вывода и возвращает предыдущую точность. По умолчанию точность равна шести десятичным цифрам.

Формат вещественных чисел - 2

По умолчанию ни один из флагов **ios::fixed** и **ios::scientific** не установлен. В этом случае запись выбирается в зависимости от выводимого значения.

Для этого делается попытка вывести все значащие десятичные цифры (но не более **precision()**) с удалением начального нуля перед десятичной точкой и/или всех завершающих пробелов, а в крайнем случае - даже десятичной точки. Если **precision()** разрядов оказывается достаточно, используется десятичная запись; в противном случае - научная запись.

При помощи флага **showpoint** можно заставить поток данных выводить десятичную точку и завершающие нули до ширины **precision()** разрядов

Примеры форматирования вещественных чисел

	precision()	*421.0*	*0.0123456789*
Обычный формат	2	4.2e+02	0.012
	6	421	0.0123457
С флагом **showpoint**	2	4.2e+02	0.012
	6	421.000	0.0123457
С флагом **fixed**	2	421.00	0.01
	6	421.000000	0.012346
С флагом **scientific**	2	4.21e+02	1.23e-02
	6	4.210000e+02	1.234568e-02

Как и в случае целых значений, флаг **ios::showpos** служит для принудительного вывода знака положительных чисел.

Флаг **ios::uppercase** указывает, какая буква должна использоваться в научной записи (E или e).

Форматы вещественных чисел - манипуляторы

Флаг ios::showpoint, тип записи и точность можно задать при помощи манипуляторов, представленных в таблице:

Манипулятор	*Описание*
showpoint	Десятичная точка всегда используется при выводе (установка флага ios: :showpoint)
noshowpoint	Десятичная точка не обязательна при выводе (сброс флага ios::showpoint)
setprecision(val)	Выбор новой точности val
fixed	Использование десятичной записи.
scientific	Использование научной записи

Общие параметры форматирования

skipws - флаг автоматического игнорирования начальных пропусков при чтении данных оператором >>.

Флаг **ios::skipws** устанавливается по умолчанию; это означает, что по умолчанию некоторые операции чтения игнорируют начальные пропуски. Обычно этот флаг удобнее держать установленным. Например, вам не придется специально заботиться о чтении пробелов, разделяющих числа. С другой стороны, это означает, что вы не сможете читать пробелы оператором >>, потому что начальные пропуски всегда игнорируются.

Для управления этим флагом существуют манипуляторы **skipws** (установка) и **noskipws** (сброс).

unitbuf – флаг принудительного вывода содержимого буфера после каждой операции записи.

Флаг **ios::unitbuf** управляет буферизацией вывода. При установленном флаге **ios::unitbuf** вывод практически выполняется без буферизации – выходной буфер очищается после каждой операции записи. По умолчанию этот флаг не устанавливается. Исключение составляют потоки данных **cerr** и **wcerr**, для которых этот флаг устанавливается в исходном состоянии.

Для управления этим флагом существуют манипуляторы **unitbuf** (установка) и **nounitbuf** (сброс).

Интернационализация

Форматы ввода-вывода также адаптируются к национальным стандартам.

Функции существующие для этой цели определены в классе **ios_base**.

imbue(loc) - назначение объекта локального контекста.

getloc() - получение текущего объекта локального контекста.

С каждым потоком данных связывается некоторый *объект локального контекста*. По умолчанию исходный объект локального контекста создается как

копия глобального объекта локального контекста на момент конструирования

потока данных. В частности, объект локального контекста определяет параметры форматирования чисел (например, символ, используемый в качестве десятичной точки, или режим числового/строкового представления логических величин).

В отличие от аналогичных средств С средства интернационализации стандартной библиотеки С++ позволяют задавать локальные контексты на уровне отдельных потоков данных. Например, такая возможность позволяет выполнять чтение вещественных чисел в американском формате и последующей записи в немецком формате (в котором вместо «десятичной точки» используется запятая).

Интернационализация -2

При работе с потоками данных часто возникает задача приведения отдельных символов (в первую очередь управляющих) к кодировке потока данных. Для этого в потоках данных поддерживаются функции преобразования:

widen(c) – преобразование символа **c** типа **char** к кодировке, используемой потоком.

narrow(c,def) – преобразование символа **c** из кодировки, используемой потоком, к типу **char** (если такого символа не существует, возвращается **def**).

Следующая команда преобразует символ новой строки в кодировку, используемую потоком данных:

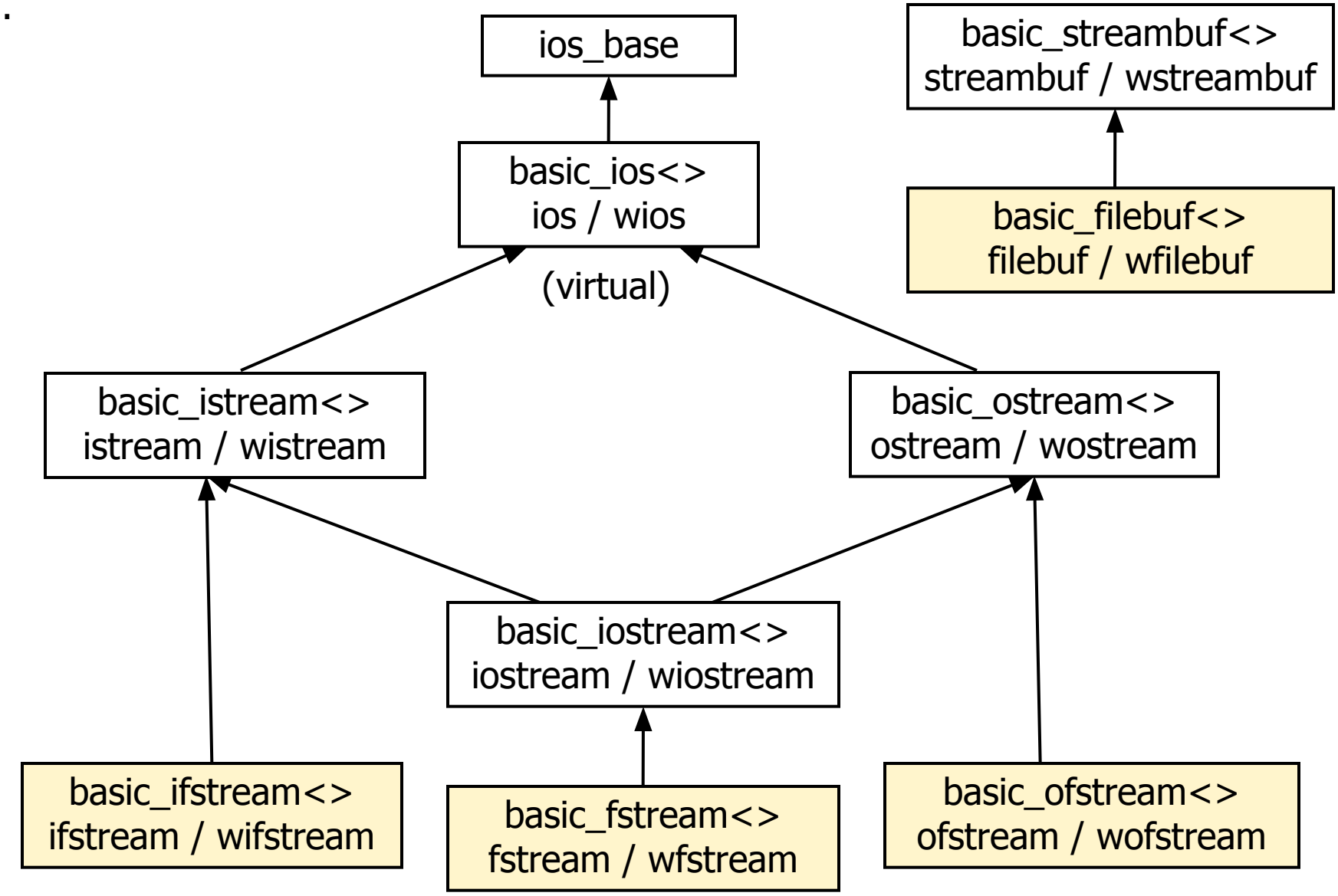
```
strm.widen('\n');
```

Доступ к файлам

Потоки данных также используются для работы с файлами. В стандартную библиотеку C++ входят четыре основных шаблона, для которых определены стандартные специализации.

- Шаблон **basic_ifstream< >** со специализациями **ifstream** и **wifstream** обеспечивает чтение файлов («файловый входной поток данных»).
- Шаблон **basic_ofstream< >** со специализациями **ofstream** и **wofstream** обеспечивает запись файлов («файловый выходной поток данных»).
- Шаблон **basic_fstream< >** со специализациями **fstream** и **wfstream** обеспечивает чтение и запись файлов.
- Шаблон **basic_filebuf< >** со специализациями **filebuf** и **wfilebuf** используется только другими классами файловых потоков данных для выполнения фактических операций чтения и записи символов.

Иерархия классов файловых потоков данных



Описание классов

```
namespace std { . . .
 template <class charT, class traits = char traits<charT> >
   class basic_ifstream;
 typedef basic_ifstream<char> ifstream;
 typedef basic_ifstream<wchar_t> wifstream;
 template <class charT, class traits = char traits<charT> >
   class basic_ofstream;
 typedef basic_ofstream<char> ofstream;
 typedef basic_ofstream<wchar_t> wofstream;
 template <class charT, class traits = char traits<charT> >
   class basic_fstream;
 typedef basic_fstream<char> fstream;
 typedef basic_fstream<wchar_t> wfstream;
 template <class charT, class traits = char traits<charT> >
   class basic_filebuf;
 typedef basic_filebuf<char> filebuf:
 typedef basic_filebuf<wchar_t> wfilebuf;
}
```

Достоинства файловых потоков

Основным достоинством потоковых классов для работы с файлами является автоматизация выполняемых операций. Файлы автоматически открываются во время конструирования и закрываются при уничтожении объекта. Естественно, что эта возможность имеется благодаря соответствующему определению конструкторов и деструкторов.

Одно важное обстоятельство, относящееся к потокам данных с поддержкой и чтения и записи, - такие потоки не должны допускать произвольного переключения между чтением и записью!! Чтобы после начала чтения из файла переключиться на запись (или наоборот), необходимо выполнить операцию позиционирования (возможно, с сохранением текущей позиции). Единственное исключение из этого правила относится к чтению с выходом за конец файла; в этой ситуации можно немедленно переходить к записи символов. Нарушение этого ограничения приводит к нежелательным побочным эффектам.

Если при конструировании файлового потока данных в аргументе передается С-строка (тип **char***), то при этом автоматически делается попытка открыть файл для чтения и/или записи. Признак успеха этой попытки отражается в состоянии потока данных. Следовательно, после конструирования следует проверить состояние потока данных.

Пример использования файловых потоков

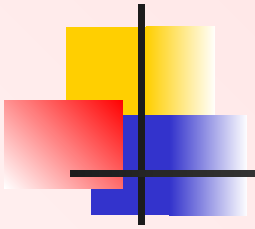
Создание файла, содержащего весь набор символов (символы от 32 до 255)

```
#include <string>      // Строки
#include <iostream>  // Ввод-вывод
#include <fstream>    // Файловый ввод-вывод
#include <iomanip>    / / setw()
#include <cstdlib>     / / exit()
using namespace std;

// Опережающие объявления
void  writeCharsetToFile (const string& filename);
void  outputFile (const string& filename);

int main ()
{
      wri teCharsetToFi le( "charset. out") ;
      outputFile("charset.out");
}
```

Пример использования файловых потоков - 2

Создание файла, содержащего весь набор символов (символы от 32 до 255)

```
void writeCharsetToFile (const string& filename)
{
// Открытие выходного файла
        ofstream file(filename.c_str());
// Файл открыт?
        if (! file) {
// NO. abort program
          cerr << "can't opeп output file \"" << filename << "\"" << endl;
          exit(EXIT_FAILURE);
        }
// Вывод текущего набора символов
        for (int i=32; i <256; i++) {
          file << "value: " << setw(3) << i << " "
               << "char: " << static_cast<char>(i) << endl;
        }
}   // Автоматическое закрытие файла
```



По поводу использования файловых потоков

В конце обеих функций открытые файлы автоматически закрываются при выходе соответствующих потоков данных из области видимости.

Деструкторы классов **ifstream** и **ofstream** закрывают файлы, если они остаются открытыми на момент уничтожения объекта.

Если файл должен использоваться за пределами области видимости, в которой он был создан, выделите объект из кучи и удалите его позднее, когда надобность в нем отпадет:

```
std::ofstream* fileptr = new std::ofstream("xyz");
delete fileptr;
```

Вместо последовательного вывода отдельных символов также можно вывести все содержимое файла одной командой, передавая указатель на потоковый буфер файла в аргументе оператора **<<**:

```
std::cout << file.rdbuf();
```

Режимы открытия файлов

Флаги управления режимами открытия файлов, определены в классе **ios_base**. Флаги относятся к типу **openmode** и группируются в битовые маски по аналогии с флагами **fmtflags**.

Флаг	*Описание*
in	Открытие файла для чтения (используется по умолчанию для ifstream)
out	Открытие файла для записи (используется по умолчанию для ofstream)
app	Запись данных производится только в конец файла
ate	Позиционирование в конец файла после открытия («at end»)
trunc	Удаление старого содержимого файла
binary	Специальные символы не заменяются

В некоторых реализациях имеются дополнительные флаги типа **nocreate** (файл должен существовать при открытии) и **noreplace** (файл не должен существовать). Однако эти флаги отсутствуют в стандарте, поэтому их использование влияет на переносимость программы.

Комментарии по флагам открытия

Флаг **binary** запрещает преобразование специальных символов или символьных последовательностей (например, конца строки или конца файла). В операционных системах типа MS-DOS или OS/2 конец логической строки в тексте обозначается двумя символами (CR и LF). При открытии файла в обычном текстовом режиме (сброшенный флаг **binary**) символы новой строки заменяются последовательностью из двух символов, и наоборот. При открытии файла в двоичном режиме (с установленным флагом **binary**) эти преобразования не выполняются.

Флаг **binary** должен использоваться всегда, когда файл не содержит чисто текстовой информации и обрабатывается как двоичные данные. Пример – копирование файла с последовательным чтением символов и их записью без модификации. Если файл обрабатывается в текстовом виде, флаг **binary** не устанавливается, потому что в этом случае символы новой строки нуждаются в специальной обработке. Флаги объединяются оператором 1. Полученный результат типа **openmode** может передаваться конструктору во втором аргументе.

Следующая команда открывает файл для присоединения текста в конце:

```
std::ofstream file("abc.out", std::ios::out | std::ios::
app);
```

Соответствие по флагам открытия C++ и C

Флаги **ios_base**	Описание	Обозначения режимов в C
in	Чтение (файл должен существовать)	**"r"**
out	Стирание и запись (файл создается при необходимости)	**"w"**
out l trunc	Стирание и запись (файл создается при необходимости)	**"w"**
out l app	Присоединение (файл создается при необходимости)	**"a"**
in l out	Чтение и запись с исходным позиционированием в начало файла	**"r+"**
in l out l trunc	Стирание, чтение и запись (файл создается при необходимости)	**"w+"**

Установленный флаг **binary** соответствует строке с присоединенным символом **b**, а установленный флаг **ate** соответствует позиционированию в конец файла немедленно после открытия. Другие комбинации, отсутствующие в таблице (например, **trunc l app**), недопустимы.

Функции обслуживания файловых потоков

Открытие файла для чтения и/или записи не зависит от класса соответствующего объекта потока данных. Класс лишь определяет режим открытия по умолчанию при отсутствии второго аргумента. Это означает, что файлы, используемые только классом **ifstream** или **ofstream**, могут открываться для чтения *и* записи. Режим открытия передается соответствующему классу потокового буфера, который открывает файл. Тем не менее операции, разрешенные для данного объекта, определяются классом потока данных.

Для открытия и закрытия файлов, принадлежащих файловым потокам данных, существуют функции :

open(имя) – открытие файла для потока в режиме по умолчанию;

open(имя, флаги) – открытие файла для потока в режиме, определяемом переданными флагами;

close() – закрытие файлового потока;

is_open() – проверка открытия файла.

Эти функции используются в основном при создании файловых потоков данных без инициализации.

Функции обслуживания – пример (начало)

```
// Заголовочные файлы для файлового ввода-вывода
#include <fstream>
#include <iostream>
using namespace std;
/* Для всех файлов. имена которых переданы в аргументах командной строки.
* - открыть. вывести содержимое и закрыть файл
*/

void main (int argc, char* argv[])
{
    ifstream file;
// Перебор аргументов командной строки
    for (int i=1; i<argc; ++i) {
// Открытие файла
        file.open(argv[i]);
```

Функции обслуживания – пример (окончание)

```
// Вывод содержимого файла в cout
    char c;
    while (file.get(c)) {
        cout.put(c);
    }

// Сброс флагов eofbit и failbit. установленных
// при обнаружении конца файла (open ничего такого не делает!!!)
    file.clear();

// Закрытие файла
    file.close();

  }   // На следующий файл из списка аргументов программы

}
```

Тест с отчетом 2

1) Каким ключевым словом обозначаются встраиваемые функции?

2) Что будет выведено на экран в результате компиляции и выполнения следующего кода?

```
#include <iostream>
using namespace std;
struct A {
   A() { cout << "A"; }
  ~A() { cout << "~A"; }
  virtual void operator()() =
0;
};
struct B : A {
  B() { cout << "B"; }
  ~B() { cout << "~B"; }
  void operator()() { cout <<
"B"; }
};
int main(void) {
  B b;              //AB
  A &a = b;         //B
  a();
  b();              //B
  return 0;         //~B~A
}
```

3) Какой тип имеет такой литерал: "C++" ?

(1) char[4]
(2) char[3]
(3) const char[4]
(4) Ничего из перечисленного
(5) const char[3]

(перечислите номера)