КОНСТРУИРОВАНИЕ ПРОГРАММ И ЯЗЫКИ ПРОГРАМИРОВАНИЯ

Лекция № 22.2 — Библиотека ввода-вывода

Преподаватель: Поденок Леонид Петрович, 505а-5

+375 17 293 8039 (505а-5)
+375 17 320 7402 (ОИПИ НАНБ)
prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by

Кафедра ЭВМ, 2021

Оглавление

Библиотека ввода/вывода

Общие замечания

Библиотека **iostream** — это объектно-ориентированная библиотека, которая обеспечивает функции ввода и вывода с использованием потоков.

Поток — это абстракция, представляющая устройство, на котором выполняются операции ввода и вывода. В основном поток может быть представлен как источник или место назначения символов неопределенной длины.

Потоки обычно связаны с физическим источником или местом назначения символов, таким как файл на диске, клавиатура или консоль, поэтому символы, полученные или записанные в/из абстракции, называемой потоком, физически вводятся/выводятся на физическое устройство.

Например, файловые потоки — это объекты C++ для управления файлами и взаимодействия с ними. Когда файловый поток используется для открытия файла, любая операция ввода или вывода, выполняемая в этом потоке, физически отражается в файле.

Для работы с потоками ввода/вывода C++ предоставляет стандартную библиотеку **iostream**, которая содержит следующие элементы:

- базовые шаблоны классов;
- создание экземпляров шаблона класса;
- стандартные объекты;
- типы;
- манипуляторы.

Базовые шаблоны классов

В основе библиотеки **iostream** лежит иерархия шаблонов классов. Шаблоны классов обеспечивают большую часть функциональных возможностей библиотеки независимо от типа.

Это набор шаблонов классов, каждый из которых имеет два параметра шаблона:

- **char type (charT)**, который определяет тип элементов, которые будут обрабатываться;
- **traits**, который предоставляет дополнительные характеристики, специфичные для конкретного типа элементов.

Шаблоны классов в этой иерархии классов имеют то же имя, что и их экземпляры типа **char**, но с префиксом **basic_**. Например, шаблон класса, из которого создается экземпляр **istream**, называется **basic_istream**, тот, из которого вызывается **fstream**, называется **basic_fstream** и так далее.

Единственным исключением является **ios_base**, который является обычным классом.

Создание экземпляров шаблона класса

Библиотека включает в себя два стандартных набора экземпляров всей иерархии шаблонов класса **iostream** — один является байт-ориентированным для управления элементами типа **char**, а другой — широко-ориентированным для управления элементами типа **wchar_t**.

Байт-ориентированное (тип **char**) создание экземпляров, вероятно, является наиболее известной частью библиотеки **iostream**. Такие классы, как **ios**, **istream** и **ofstream**, являются байт-ориентироваными. На диаграмме вверху этой страницы показаны имена и отношения байт-ориентированных классов.

Классы с широкой ориентацией (**wchar_t**) следуют тем же соглашениям об именах, что и с узкой, но для имени каждого класса и объекта используется префикс **w**, образующий, например, **wios**, **wistream** и **wofstream**.

Стандартные объекты

Как часть библиотеки **iostream**, заголовок **<iostream>** объявляет определенные объекты, которые используются для выполнения операций ввода и вывода на устройствах стандартного ввода и вывода.

Они разделены на два набора — узкоориентированные объекты, которыми являются **cin**, **cout**, **cerr** и **clog**, и их широко ориентированные аналоги, объявленные как **wcin**, **wcout**, **wcerr** и **wclog**.

Типы

Классы **iostream** практически не используют фундаментальные типы в прототипах своих членов. Обычно они используют определенные типы, которые зависят от свойств, используемых при их создании.

Для экземпляров **char** и **wchar_t** для представления позиций, смещений и размеров по умолчанию используются типы **streampos**, **streamoff** и **streamsize**, соответственно.

Манипуляторы

Манипуляторы — это глобальные функции, предназначенные для использования вместе с операторами вставки (**<<**) и извлечения (**>>**), выполняемыми для объектов потока **iostream**.

Обычно они изменяют свойства и настройки форматирования потоков.

endl, **hex** и **scientific** — вот некоторые примеры манипуляторов.

Манипуляторы потока

Манипуляторы — это функции, специально разработанные для использования в сочетании с операторами вставки (**<<**) и извлечения (**>>**) при работе с объектами потока, например:

```
cout << boolalpha;
```

Манипуляторы являются обычными функциями, и их также можно вызывать как любую другую функцию с использованием объекта потока в качестве аргумента, например:

```
boolalpha (cout);
```

Манипуляторы используются для изменения параметров форматирования потоков, а также для вставки или извлечения определенных специальных символов.

Основные форматные флаги

Эти манипуляторы можно использовать как для входных, так и для выходных потоков, хотя многие из них действуют только тогда, когда применяются только к выходным или только к входным потокам.

Независимые флаги (включение/выключение)

boolalpha/noboolalpha — буквенные или цифровые логические значения
showbase/noshowbase — префиксы основания счисления
showpoint/noshowpoint — десятичная точка
showpos/noshowpos — знак плюс
skipws/noskipws — пропускать пробельный материал при каждом извлечении
unitbuf/nounitbuf — сбрасывать/не сбрасывать буфер после вставки
uppercase/nouppercase — верхний регистр для базы и экспоненты

Флаги основания счисления
`dec` — десятичное
`hex` — шестнадцатеричное
`oct` — восьмеричное

Флаги форматирования чисел с плавающей запятой
`fixed` — формат с фиксированной точкой
`scientific` — «научный» формат

Флаги выравнивания
`internal` — выравнивание путем вставки символов во внутреннюю позицию.
`left` — выравнивание влево
`right` —выравнивание вправо

Манипуляторы ввода
`ws` — пропустить пробельный материал

Манипуляторы вывода
`endl` — вставить перевод строки и сбросить буфер потока
`ends` — вставить нулевой символ
`flush` — сбросить буфер потока

Параметризованные манипуляторы

Эти функции при использовании в качестве манипуляторов принимают параметры. Они требуют явного включения заголовочного файла **<iomanip>**.

setiosflags/resetiosflags — установить/ сбросить флаги формата
setbase — установить флаг основания счисления
setfill — установить символ заполнения
setprecision — установить точность представления
setw — установить ширину поля

std::internal/std::left/std::right — манипуляторы флагов выравнивания (поле adjustfield)

```
ios_base& right(ios_base& str);
ios_base& left(ios_base& str);
ios_base& internal(ios_base& str);
```

Когда для параметра **adjustfield** установлено значение **right/left**, вывод в поток **str** дополняется до ширины (ширины) поля путем вставки символов заполнения (**fill**) в начале/конце поля, выравнивая вывод по правой/левой границе.

Когда для параметра **adjustfield** установлено значение **internal**, выходные данные дополняются до ширины (ширины) поля путем вставки символов заполнения (**fill**) в указанной внутренней точке, которая для числовых значений находится между знаком и/или числовым основанием и величиной числа. Для нечислового вывода значение **internal** эквивалентно значению **right**.

Пример использования манипуляторов для поля adjustfield

```
#include <iostream>     // std::cout, std::internal, std::left, std::right

int main () {

  int n = -123;

  std::cout.width(6); std::cout << std::internal << n << '\n';
  std::cout.width(6); std::cout << std::left << n << '\n';
  std::cout.width(6); std::cout << std::right << n << '\n';

  return 0;
}
```

Вывод

```
-  123
-123
  -123
```

Примеры использования манипуляторов

```
#include <iostream>     // std::cout, std::showbase, std::noshowbase
                        // std::uppercase, std::scientific

int main () {

  int    n = 20;
  double q = 3.141592;
  std::cout << std::hex << std::showbase   << n << '\n';
  std::cout << std::hex << std::scientific << std::showbase   << q << '\n';
  std::cout << std::uppercase;
  std::cout << std::hex << std::showbase   << n << '\n';
  std::cout << std::hex << std::scientific << q << '\n';
  std::cout << std::hex << std::noshowbase << n << std::endl;
  return 0;
}
```

Вывод

```
0x14
3.141592e+00
0X14
3.141592E+00
14
```

Пример skipws

```
#include <iostream>     // std::cout, std::skipws, std::noskipws
#include <sstream>      // std::istringstream

int main () {

  char a, b, c;

  std::istringstream iss ("  123");
  iss >> std::skipws >> a >> b >> c;
  std::cout << a << b << c << '\n';

  iss.seekg(0);
  iss >> std::noskipws >> a >> b >> c;
  std::cout << a << b << c << '\n';
  return 0;
}
```

```
123
  1
```

std::setiosflags/std::resetiosflags — установить/сбросить флаги формата

```
#include <iomanip>

setiosflags(ios_base::fmtflags mask);
```

Устанавливает/сбрасывает флаги формата, указанные в маске параметра.

Ведет себя так, как если бы был вызван член **ios_base::setf()/unsetf()** с маской **mask** в качестве аргумента в потоке, в который он вставляется/извлекается как манипулятор (он может вставляться/извлекаться как для входных потоков, так и для выходных).

Этот манипулятор объявлен в заголовке **<iomanip>**.

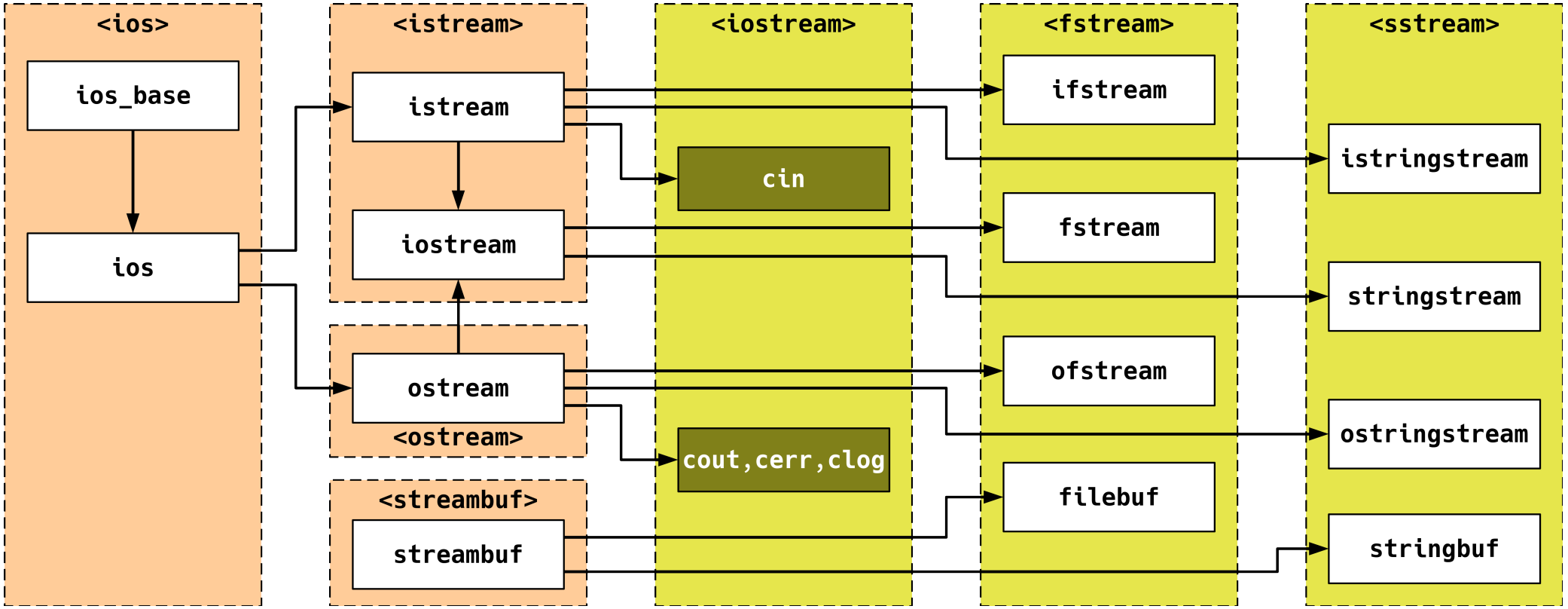
Для получения дополнительной информации о конкретных флагах, которые могут быть изменены с помощью этой функции-манипулятора, следует смотреть **ios_base::fmtflags**.

```
#include <iostream>     // std::cout, std::hex, std::endl
#include <iomanip>      // std::setiosflags

int main () {
  std::cout << std::hex;
  std::cout << std::setiosflags(std::ios::showbase | std::ios::uppercase);
  std::cout << 100 << " -- " ;
  std::cout << std::resetiosflags(std::ios::showbase) << 100 << std::endl;
}
```

0X64 — 64

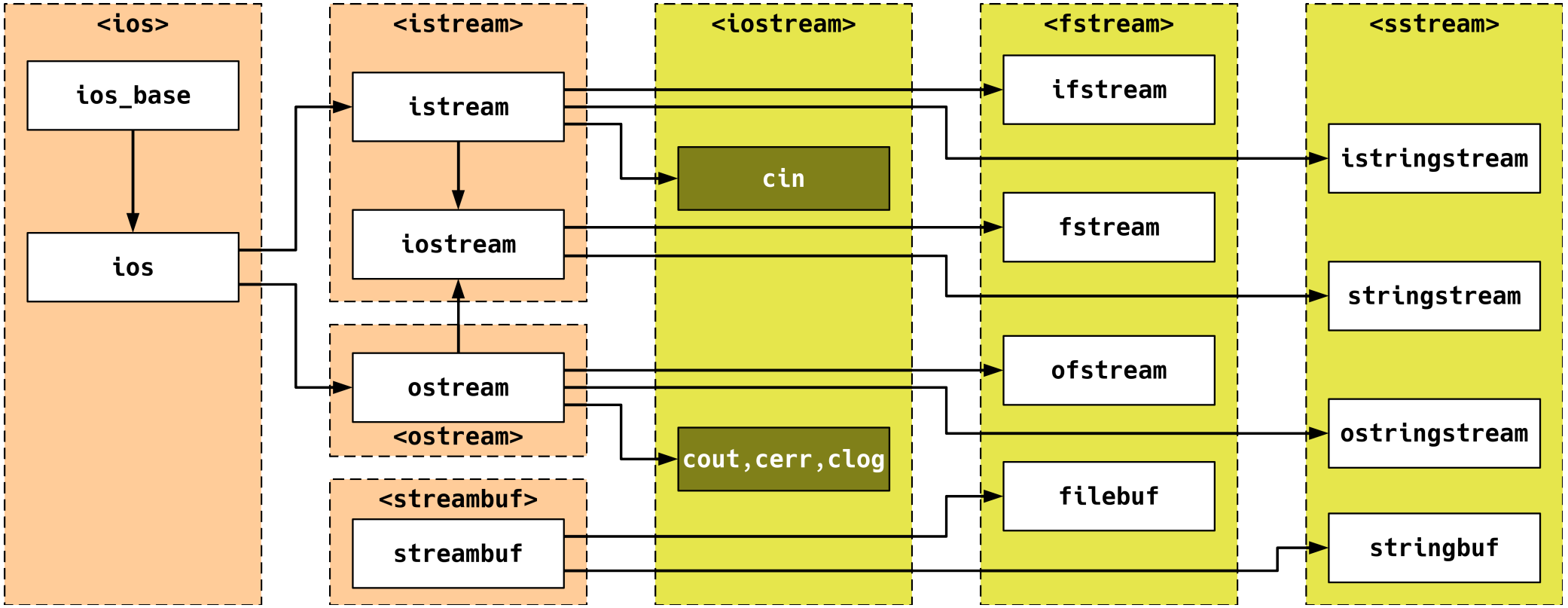
Организация библиотеки



Библиотека и ее иерархия классов разделены на разные файлы:

<ios>, **<istream>**, **<ostream>**, **<streambuf>** обычно не включаются напрямую в большинство программ на C++. Они описывают базовые классы иерархии и автоматически включаются в другие файлы заголовков библиотеки, содержащие производные классы.

<iostream> объявляет объекты, используемые для связи через стандартный ввод и вывод (включая **cin** и **cout**).



<fstream> определяет классы файлового потока (например, шаблон **basic_ifstream** или класс **ofstream**), а также объекты внутреннего буфера, используемые с ними (**basic_filebuf**).

Эти классы используются для управления файлами с помощью потоков.

<sstream> определяет классы, определенные в этом файле, используются для управления строковыми объектами, как если бы они были потоками.

<iomanip> объявляет некоторые стандартные манипуляторы с параметрами, которые будут использоваться с операторами извлечения и вставки для изменения внутренних флагов и параметров форматирования.

Классы файловых потоков

ifstream, **ofstream**, **fstream** — классы потоков ввода, вывода и ввода/вывода из/в файлы.

Объекты этого класса поддерживают объект **filebuf** в качестве своего внутреннего буфера потока, который выполняет операции ввода/вывода с файлом, с которым они связаны (если есть).

Файловые потоки связываются с файлами либо при создании, либо путем вызова функции-члена **open( )**.

Эти классы — экземпляры **basic_ifstream**, **basic_ofstream** и **basic_fstream**, соответственно, со следующими параметрами шаблона:

Параметр шаблона	Определение	Комментарий
charT	char	Псевдоним char_type
traits	char_traits<char>	Псевдоним traits_type

Помимо внутреннего буфера файлового потока, объекты этого класса хранят набор внутренних полей, унаследованных от **ios_base**, **ios** и **istream**:

Состояние

Поле	Функция-член	Описание
Состояние ошибки	**rdstate** **setstate** **clear**	Текущее состояние ошибки потока. Индивидуальные значения можно получить, вызвав **good( )**, **eof( )**, **fail( )** и **bad( )**. См. **ios_base::iostate**
Маска исключений	**exceptions**	Флаги состояния, для которых генерируется исключение при сбое.

Форматирование

Поле	Функция-член	Описание
Флаги формата	**`flags`** **`setf`** **`unsetf`**	Набор внутренних флагов, которые влияют на то, как определенные операции ввода/вывода интерпретируются или генерируются. См. **`ios_base::fmtflags`**
Ширина поля	**`width`**	Ширина следующего отформатированного элемента для вставки.
Точность отображения	**`precision`**	Десятичная точность для следующего вставленного значения с плавающей запятой.
Языковый стандарт	**`getloc`** **`imbue`**	Объект языкового стандарта, используемый функцией для операций форматированного ввода/вывода, на которые влияют свойства локализации.
Символ заполнения	**`fill`**	Символ для заполнения отформатированного поля до ширины (**`width`**) поля.

Разное

Поле	Функция-член	Описание
Стек обратного вызова	**`register_callback`**	Стек указателей на функции, которые вызываются при наступлении определенных событий.
Расширяемые массивы	**`iword`** **`pword`** **`xalloc`**	Внутренние массивы для хранения объектов типа **`long`** и **`void *`**.
Связанный поток	**`tie`**	Указатель на выходной поток, который сбрасывается перед каждой операцией ввода-вывода в этом потоке.
Буфер потока	**`rdbuf`**	Указатель на связанный объект **`streambuf`**, который отвечает за все операции ввода/вывода.
Количество символов	**`gcount`**	Количество символов, прочитанных последней операцией неформатированного ввода.

ios_base::iostate — тип битовой маски для представления флагов состояния ошибки потока.

Все потоковые объекты хранят внутри информацию о состоянии объекта.

Эту информацию можно получить как элемент этого типа, вызвав функцию-член **basic_ios::rdstate()**, или установить, вызвав **basic_ios::setstate()**.

Значения, передаваемые и получаемые этими функциями, могут быть любой допустимой комбинацией (с использованием логического оператора ИЛИ, "|") следующих констант-членов:

Значение флага	Означает
eofbit	Достигнут конец файла при выполнении операции извлечения во входном потоке.
failbit	Последняя операция ввода завершилась неудачно из-за ошибки, связанной с внутренней логикой самой операции.
badbit	Ошибка из-за сбоя операции ввода/вывода в буфере потока.
goodbit	Нет ошибки. Представляет отсутствие всего вышеперечисленного (значение 0).

Эти константы определены в классе **ios_base** как открытые члены. Следовательно, на них можно ссылаться либо напрямую по их имени как члены ios_base (например, **ios_base::badbit**), либо с использованием любого из их унаследованных классов или созданных объектов, например, **ios::eofbit** или **cin.goodbit**.

ios_base::fmtflags — тип битовой маски для представления флагов формата потока.

Этот тип используется в качестве параметра и/или возвращаемого значения функциями-членами **flags()**, **setf()** и **unsetf()**.

Значения, передаваемые и получаемые этими функциями, могут быть любой допустимой комбинацией следующих констант-членов:

boolalpha читает/пишет элементы типа **bool** в виде буквенных строк (**true** и **false**).

showbase пишет целые значения, которым предшествует соответствующий им числовой базовый префикс.

showpoint пишет значения с плавающей запятой, всегда включая десятичную точку.

showpos пишет неотрицательные числовые значения, которым предшествует знак плюс (+).

skipws пропускает начальные пробелы при определенных операциях ввода.

unitbuf сбрасывает вывод после каждой операции вставки.

uppercase пишет прописными буквами, вместо строчных в некоторых операциях вставки.

dec чтение/запись целых значений с использованием десятичного формата.

hex чтение/запись целых значений с использованием шестнадцатиричного формата.

oct чтение/запись целых значений с использованием восьмеричного формата.

fixed запись значений с плавающей запятой в нотации с фиксированной запятой.

scientific запись значений с плавающей запятой в экспоненциальном представлении.

internal вывод дополняется до ширины поля путем вставки символов-заполнителей в указанной внутренней точке.

left вывод дополняется до ширины поля, добавляя символы заполнения в конце.

right вывод дополняется до ширины поля путем вставки символов заполнения в начале.

Также могут использоваться три дополнительные константы битовой маски, состоящие из комбинации значений каждой из трех групп селективных флагов:

Значение флага	Эквивалентно
adjustfield	**left | right | internal**
basefield	**dec | oct | hex**
floatfield	**scientific | fixed**

Значения всех этих констант можно объединить в одно значение **fmtflags** с помощью побитового оператора ИЛИ (|).

Данные константы определены как открытые члены в классе **ios_base**. Следовательно, на них можно ссылаться либо напрямую по их имени как члены **ios_base** (например, **ios_base::hex**), либо с помощью любого из их унаследованных классов или созданных объектов, например, **ios::left** или **cout.oct**.

Значения типа **ios_base::fmtflags** не следует путать с манипуляторами с тем же именем, но в глобальной области видимости, потому что они используются в разных обстоятельствах.

Манипуляторы нельзя использовать в качестве значений для **ios_base::fmtflags**, а также эти константы не следует использовать вместо манипуляторов:

```
ios_base::skipws     // constant value of type ios_base::fmtflags
skipws               // manipulator (global function)
```

Несколько манипуляторов имеют то же имя, что и эти константы-члены, но являются глобальными функциями. Поведение этих манипуляторов обычно соответствует их установке или отключению с помощью **ios_base::setf** или **ios_base::unsetf**, но их не следует путать! Манипуляторы - это глобальные функции, а вышеприведенные константы являются константами-членами.

Например, **showbase** — это манипулятор, а **ios_base::showbase** — постоянное значение, которое можно использовать в качестве параметра с **ios_base::setf**.

Пример использования ios_base::fmtflags

```
#include <iostream>     // std::cout, std::ios_base, std::ios,
                        // std::hex, std::showbase
int main () {

  // использование fmtflags в качестве констант-членов класса
  std::cout.setf(std::ios_base::hex, std::ios_base::basefield);
  std::cout.setf(std::ios_base::showbase);
  std::cout << 100 << '\n';

  // использование fmtflags в качестве констант унаследованных членов класса
  std::cout.setf(std::ios::hex, std::ios::basefield);
  std::cout.setf(std::ios::showbase);
  std::cout << 100 << '\n';

  // использование fmtflags в качестве констант-членов объекта:
  std::cout.setf(std::cout.hex, std::cout.basefield);
  std::cout.setf(std::cout.showbase);
  std::cout << 100 << '\n';
```

```
  //  использование fmtflags в качестве типа
  std::ios_base::fmtflags ff;
  ff = std::cout.flags();
  ff &= ~std::cout.basefield;   // сбросить биты basefield
  ff |= std::cout.hex;          // установить hex
  ff |= std::cout.showbase;     // установить showbase
  std::cout.flags(ff);
  std::cout << 100 << '\n';

  // не используя fmtflags, а используя манипуляторы:
  std::cout << std::hex << std::showbase << 100 << '\n';

  return 0;
}
```

std_ios::rdstate/setstate/clear — получить/установить/очистить состояние потока

```
iostate rdstate() const;              // Получить флаги состяния
void setstate(iostate state);        // Установить флаг состояния ошибки
void clear(iostate state = goodbit); // Установить флаги состояния ошибки
```

rdstate — получить флаги состояния

Возвращает текущие флаги состояния внутренней ошибки потока.

В случае возникновения определенных ошибок при вызовах функций ввода/вывода в потоке автоматически устанавливаются флаги состояния внутренней ошибки.

Установить флаг состояния ошибки

Изменяет текущие флаги состояния внутренней ошибки, комбинируя текущие флаги с флагами в аргументе **state** (как при выполнении побитовой операции ИЛИ).

Ни один битовый флаг ошибки из уже установленных не сбрасывается.

Есть похожая функция **clear()** которая это делает.

Эта функция ведет себя так, как если бы она была определена как:

```
void ios::setstate (iostate state) {
    clear(rdstate() | state);
}
```

Установить флаги состояния ошибки

Устанавливает новое значение для флагов состояния внутренней ошибки потока.

Текущее значение флагов перезаписывается — все биты заменяются на биты, указанные в аргументе **state**. Если в аргументе **state** указано **goodbit** (которое равно нулю), все флаги ошибок сбрасываются.

В случае, если при вызове функций **setstate()** и **clear()** нет ни одного буфера, связанного с потоком, автоматически устанавливается флаг **badbit**, независимо от значения этого бита, переданного в **state**.

Изменение состояния может вызвать исключение в зависимости от последних настроек, переданных в члене **exception**.

state — объект типа **ios_base::iostate**.

Может принимать любую комбинацию следующих констант- членов:

Значение iostate (константа-член)	**Означает**	**Функции проверки флага состояния**				
		good()	**eof()**	**fail()**	**bad()**	**rdstate()**
goodbit	Нет ошибок (значение **iostate == 0**)	true	false	false	false	goodbit
eofbit	При вводе достигнут EOF	false	true	false	false	eofbit
failbit	Логическая ошибка в i/o операции	false	false	true	false	failbit
badbit	Ошибка чтения/записи в i/o операции	false	false	true	true	badbit

eofbit, **failbit** и **badbit** — это константы-члены с определяемыми реализацией значениями, которые можно комбинировать по ИЛИ ("|").

goodbit равен нулю, что указывает на то, что ни один из других битов не установлен.

`std::ios_base::flags` — получить/установить флаги форматирования

```
get (1)  fmtflags flags() const;
set (2)  fmtflags flags(fmtflags fmtfl);
```

Первая форма (1) возвращает флаги формата, действующие в данный момент в потоке.

Вторая форма (2) устанавливает новые флаги формата и возвращает их прежние значения.

Флаги формата потока влияют на то, как данные интерпретируются при их получении определенными функциями ввода и как они записываются определенными функциями вывода.

Возможные значения аргумента этой функции и интерпретация возвращаемого ею значения описаны в **`ios_base::fmtflags`**.

Вторая форма этой функции устанавливает значение для всех флагов формата потока, перезаписывая существующие значения и очищая любой флаг, явно не установленный в аргументе.

Чтобы получить доступ к отдельным флагам, используются функции-члены **`setf()`** и **`unsetf()`**.

Пример

```
#include <iostream>     // std::cout, std::ios

int main () {
  std::cout.flags( std::ios::right | std::ios::hex | std::ios::showbase );
  std::cout.width(10);
  std::cout << 100 << '\n'; // 0x64
}
```

std::ios_base::setf()/unsetf() — установить определенные флаги формата

```
set  (1) fmtflags setf(fmtflags fmt_flags);
mask (2) fmtflags setf(fmtflags fmt_flags, fmtflags mask);
         void     unsetf(fmtflags mask);
```

Первая форма (1) устанавливает флаги формата, биты которых установлены в **fmt_flags**, оставляя без изменений остальные, как если бы это был вызов **flags(fmtfl | flags())**.

Вторая форма (2) устанавливает флаги формата потока, биты которых установлены как в **fmt_flags**, так и в маске, и очищает флаги формата, биты которых установлены в маске, но не в **fmt_flags**, как если бы это был вызов **flags((fmt_flags & mask) | (flags() & ~mask))**.

Оба вызова возвращают значения флагов формата потока перед вызовом.

Флаги формата потока влияют на то, как данные интерпретируются в определенных функци

Первая форма **setf()** (1) обычно используется для установки независимых флагов формата: **boolalpha**, **showbase**, **showpoint**, **showpos**, **skipws**, **unitbuf** и **uppercase**, которые также можно сбросить напрямую с помощью члена **unsetf()**.

Вторая форма (2) обычно используется для установки значения для одного из селективных флагов с использованием одной из битовых масок поля в качестве аргумента маски:

Значение флагов fmt_flags	битовая маска mask
left, right, internal	adjustfield
dec, oct, hex	basefield
scientific, fixed	floatfield

Пример изменения флагов с помощью setf/unsetf

```
#include <iostream>     // std::cout, std::ios

int main () {

  std::cout.setf(std::ios::hex,           // установить hex
                 std::ios::basefield );  // для basefield
  std::cout.setf(std::ios::showbase );   // и показывать этот hex
  std::cout << 100 << '\n';
  std::cout.unsetf( std::ios::showbase); // деактивировать showbase
  std::cout << 100 << '\n';
  return 0;
}
```

Вывод

```
0x64
64
```

std::ios_base::width — получить/установить ширину поля вывода

```
get (1)  streamsize width( ) const;
set (2)  streamsize width (streamsize wide);
```

Первая форма (1) возвращает текущее значение ширины поля.

Вторая форма (2) также устанавливает новую ширину поля для потока.

Ширина поля определяет минимальное количество символов, которые должны быть записаны в некоторых выходных представлениях.

Если стандартная ширина представления короче ширины поля, представление дополняется символами заполнения в точке, определяемой флагом формата **adjustfield**[1] (**left**, **right** или **internal**).

Символ заполнения можно получить или изменить, вызвав функцию-член **fill( )**.

Флаг формата **adjustfield** можно изменить, вызвав функцию-член **flags( )** или **setf( )**, можно вставить в поток один из следующих манипуляторов — **left**, **right** или **internal**, а также вставить в поток параметризованный манипулятор **setiosflags**.

Ширина поля также может быть изменена с помощью параметризованного манипулятора **setw**. (см. ниже — красным два варианта)

1 комбинация left, right, internal

Пример

```
#include <iostream>     // std::cout, std::left

int main () {

  std::cout << "....5....0" << '\n';

  std::cout << 100 << '\n';

  std::cout.width(10);
  std::cout << 100 << '\n';

  std::cout.fill('x');
  std::cout.width(15);
  std::cout << std::setw(15) << std::left << 100 << '\n';

  return 0;
}
```

Вывод

```
....5....0....5
100
       100
100xxxxxxxxxxxx
```

std::ios_base::precision — получить/установить десятичную точность с плавающей запятой

```
get (1)  streamsize precision( ) const;
set (2)  streamsize precision(streamsize prec);
```

Первая форма (1) возвращает для потока текущее значение поля точности.

Вторая форма (2) также устанавливает новое значение и возвращает предыдущее.

Точность с плавающей запятой определяет максимальное количество цифр, записываемых при операциях записи в поток значений с плавающей запятой. Как это интерпретируется, зависит от того, установлен ли флаг формата поля с плавающей запятой на определенную нотацию (фиксированную или научную) или он не установлен (с использованием нотации по умолчанию, которая не обязательно будет эквивалентна фиксированной или научной).

Для локали по умолчанию:

При использовании нотации с плавающей запятой по умолчанию, в поле точности указывается максимальное количество значащих цифр, отображаемых *в общей сумме*, включая *цифры до и после десятичной точки*. Это не является минимальным значением, и поэтому, если число может отображаться с меньшим количеством цифр, чем **prec**, отображаемое число конечными нулями дополнено не будет,

В случае фиксированной и в экспоненциальной нотации **prec** указывает, сколько цифр следует отображать после десятичной точки, даже если там будут конечные десятичные нули. Цифры перед десятичной точкой в этом случае не влияют на точность. Эту десятичную точность также можно изменить с помощью параметризованного манипулятора **setprecision**.

Пример

```
// modify precision
#include <iostream>     // std::cout, std::ios

int main () {

  double f = 3.14159;
  std::cout.unsetf(std::ios::floatfield[2]);         // floatfield not set
  std::cout.precision(5);
  std::cout << f << '\n';
  std::cout.precision(10);
  std::cout << f << '\n';

  // установим floatfield в значение fixed
  std::cout.setf(std::ios::fixed, std:: ios::floatfield);
  std::cout << f << '\n';
  return 0;
}
```

Вывод

```
3.1416
3.14159
3.1415900000
```

2 комбинация флагов fixed и scientific

std::ios::fill — получить/установить символ заполнения

```
get (1)  char fill() const;
set (2)  char fill(char fill_char);
```

Первая форма (1) возвращает символ заполнения.

Вторая форма (2) в качестве нового символа заполнения устанавливает **fill_char** и возвращает символ заполнения, который использовался до вызова.

Символ заполнения — это символ, используемый функциями вывода при заполнении результатов до ширины поля.

Для установки символа заполнения также может использоваться параметрический манипулятор **setfill**.

std::ios::exceptions — получить/установить маску исключений

```
get (1)  iostate exceptions() const;
set (2)  void exceptions (iostate except);
```

Первая форма (1) возвращает текущую маску исключения для потока.

Вторая форма (2) устанавливает новую маску исключения для потока и очищает флаги состояния ошибки потока (как если бы был вызван член **`clear()`**).

Маска исключения — это внутреннее значение, сохраняемое всеми потоковыми объектами, указывающее, для каких флагов состояния генерируется исключение типа **`failure`** (или некоторого производного типа). Эта маска является объектом типа члена **`iostate`**, который представляет собой значение, образованное любой комбинацией следующих констант-членов:

Значение `iostate` (константа-член)	**Означает**	**Функции проверки флага состояния**				
		`good()`	**`eof()`**	**`fail()`**	**`bad()`**	**`rdstate()`**
`goodbit`	Нет ошибок (значение **`iostate == 0`**)	`true`	`false`	`false`	`false`	`goodbit`
`eofbit`	При вводе достигнут EOF	`false`	`true`	`false`	`false`	`eofbit`
`failbit`	Логическая ошибка в i/o операции	`false`	`false`	`true`	`false`	`failbit`
`badbit`	Ошибка чтения/записи в i/o операции	`false`	`false`	`true`	`true`	`badbit`

`eofbit`, **`failbit`** и **`badbit`** — это константы-члены с определяемыми реализацией значениями, которые можно комбинировать по ИЛИ ("`|`").

`goodbit` равен нулю, что указывает на то, что ни один из других битов не установлен.

ios_base::getloc, ios_base::imbue — языковые стандарты (локаль)

```
locale getloc() const;           // получить текущую локаль для потока
locale imbue(const locale& loc); // установить локаль для потока
```

getloc() возвращает объект локали, связанный в данный момент с потоком.

imbue() связывает **loc** с потоком как новый объект локали, который будет использоваться с операциями, зависящими от нее (то локали).

Перед этим функция вызывает все функции, зарегистрированные через член **register_callback()** с **imbue_event** в качестве первого аргумента.

Стандартные классы потоков не наследуют этот член, а вместо этого наследуют **basic_ios::imbue**, который вызывает эту функцию, но также добавляет локаль в ассоциированный потоковый буфер, если таковой имеется.

std::ios_base::register_callback — зарегистрировать функцию обратного вызова

```
void register_callback(event_callback fn, int index);
```

Регистрирует **fn** как функцию обратного вызова, которая будет автоматически вызываться с **index** в качестве аргумента при возникновении события в потоке.

Если зарегистрировано более одной функции обратного вызова, все они вызываются в обратном порядке регистрации.

Функция обратного вызова должна иметь тип, конвертируемый в **event_callback**.

fn — указатель на функцию, которую следует вызвать

```
typedef void (*event_callback)(event ev, ios_base& ios, int index);
```

index — целочисленное значение, которое будет передано в функцию при ее вызове.

stream — это указатель на потоковый объект, где возникают события;

ev — это объект события типа члена перечисления, указывающий, какое событие произошло.

Событие может быть одним из:

copyfmt_event — при вызове члена **basic_ios::copyfmt** в тот момент, когда все флаги формата были скопированы, но до того, как была скопирована маска исключения;

erase_event — при вызове деструктора потока (также вызывается в начале **copyfmt**).

imbue_event — при вызове функции **imbue()**.

Пример использования функции обратного вызова

```
#include <iostream>     // std::cout, std::ios_base
#include <fstream>      // ofstream

void testfn(std::ios::event ev, std::ios_base& stream, int index) {

  switch (ev) {
    case stream.copyfmt_event:
        std::cout << "copyfmt_event\n"; break;
    case stream.imbue_event:
        std::cout << "imbue_event\n"; break;
    case stream.erase_event:
        std::cout << "erase_event\n"; break;
} }

int main () {

  std::ofstream fs;
  fs.register_callback(testfn, 0); // imbue_event
  fs.imbue(std::cout.getloc());
  return 0;                        // erase_event
}
```

Вывод

```
imbue_event
erase_event
```