Потоки ввода-вывода

(в C++ нет встроенных средств)

Реализация ввода/вывода в C++

- <iostream>
  - **универсальность:** не важно, с чем связан поток, — ввод-вывод с консоли, файла, сокета, процесса происходит одинаково;
  - **типобезопасность**
  - **удобство:** программисту не нужно **явно** указывать тип вводимого или выводимого объекта;
  - **расширяемость:** программист может добавить поддержку ввода-вывода в поток для любого своего объекта, просто перегрузив для пользовательского типа соответствующие операторы.

- <cstdio>
  - эффективность
  - размер кода

Понятие потока

- **Поток** — последовательность данных, которые могут быть входным или выходным, может быть представлен как файлом, так и устройством (например, терминалом).
- Поток может быть текстовым или бинарным

Замечание: поток == **псевдо файл** (или расширение понятия «файл» на операции приема/отправки данных)

Поток С++– это совокупность средств для:

- передачи/приема данных
- промежуточного хранения данных
- преобразования данных

Передача данных

При создании любого потока мы явно или неявно «привязываем» его:

- одним концом к конкретному устройству (драйвер)
- другим – к программе (объект класса)

Входной поток: ввод с клавиатуры, из файла на диске или получение данных от другого приложения
Выходной поток: вывод на экран, принтер, запись в файл или передача данных другому приложению.

Хранение данных

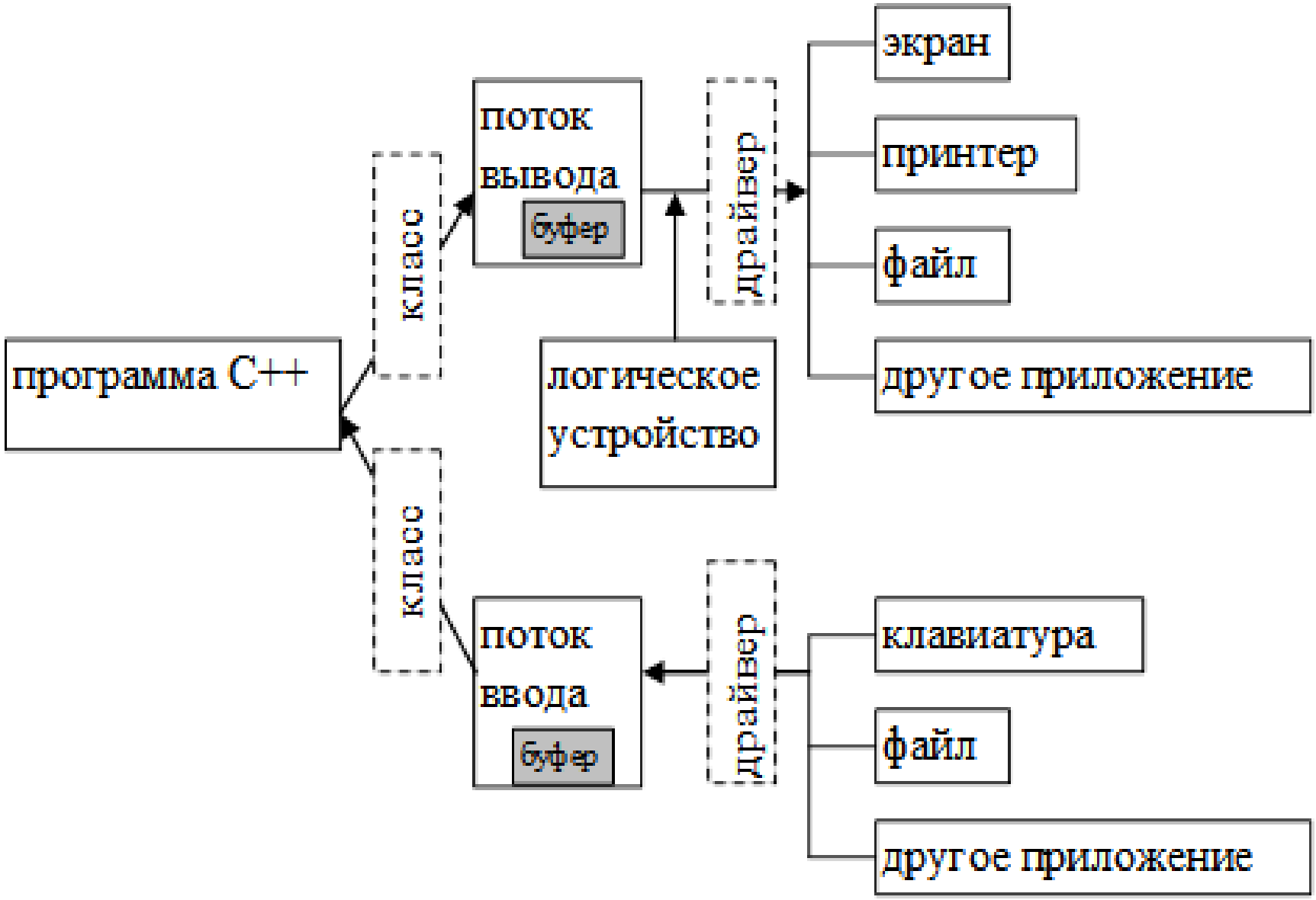
Так как передача данных - это просто чтение/запись последовательности байтов, можно представлять себе поток как **промежуточное хранилище** данных между программой и «пунктом назначения» - устройством

=> такой подход позволяет C++ программе одинаково получать данные при вводе с клавиатуры и чтении из файла (программе совершенно необязательно знать природу источника данных)

Преобразование данных

Существуют два основных способа хранения информации:

- **форматированный** - каждый байт содержит код символа (или служебный) => **преобразование значений разного типа в последовательность символов.**
- **неформатированный ввод/вывод** – это просто бинарное значение


программа C++
класс
поток вывода
буфер
драйвер
экран
принтер
файл
другое приложение
логическое устройство
класс
поток ввода
буфер
драйвер
клавиатура
файл
другое приложение

Концепция логических устройств

- Посредством драйверов устройств ОС общается с клавиатурой, экраном, принтером и коммуникационными портами как с некоторым **логическим устройством** (extended файлами в памяти).
- А классы ввода/вывода в свою очередь **умеют взаимодействовать с этими логическими устройствами**.

Система ввода-вывода C <cstdio>

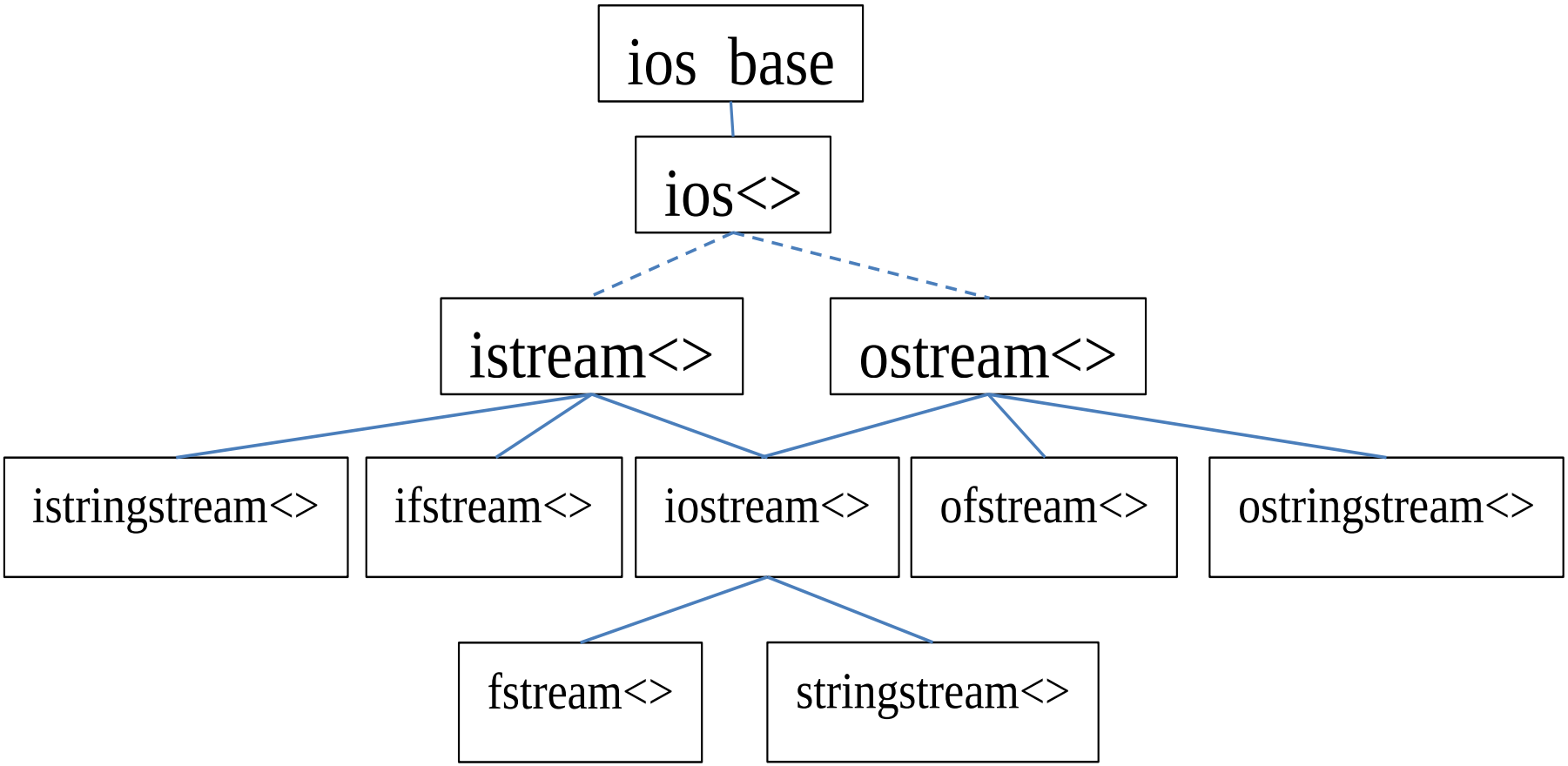
- **Функции printf() и scanf()**
  - **%**
  - **Спецификаторы вывода**
  - **Манипуляторы вывода**
- **Структура FILE:**
  - дескриптор файла
  - текущую позицию в потоке
  - индикатор конца файла
  - индикатор ошибок
  - указатель на буфер потока
- **stdin** (обычно клавиатура), **stdout** (обычно дисплей терминала), **stderr** (обычно дисплей терминала)

Система ввода-вывода C++

<iostream>

- новый **стиль ввода/вывода – объектно-ориентированный**
- главной задачей при создании новых средств ввода/вывода **было унифицировать** эти операции:
  - **операция должна выглядеть одинаково как применительно к базовым, так и пользовательским типам (???)**
  - **операция должна выглядеть одинаково независимо от устройства**.
- cin, cout, cerr, clog (то же, что и cerr, но с буферизацией)

Упрощенная иерархия классов ввода/вывода C++



ios_base
<ios>

ios_base- «**универсальный**» базовый класс:

- содержит информацию и операции, не зависящие от типа используемых символов (char или wchar_t) => это **не шаблон**:
- определяет с помощью typedef некоторые типы для удобства и лаконичности записи (например, тип streamsize – для представления числа символов, передаваемых в операциях ввода/вывода и размера буферов)
- состояние потока (задается флагами ошибок) - объект типа **iostate**
- флаги форматирования – объект типа **fmtflags**
- локализация – объект типа **locale**

basic_ios<>
<ios>

```
template<
    class CharT,
    class Traits = std::char_traits<CharT>
> class basic_ios : public ios_base
```

- указатель на basic_streambuf (буфер ввода/вывода)
- методы для заполнения, доступа, очистки буфера
- специализации для char и wchar_t:
  typedef basic_ios<char> **ios**;
  typedef basic_ios<wchar_t> **wios**;

std::char_traits<>
<string>

```
template<
    class CharT
> class  char_traits;
```

предоставляет основные операции по:

- определению конкретного типа (char, wchar_t, + C++11, + custom)
- основные действия с конкретным типом (сравнение, копирование…)

basic_istream<>
<iostream>

basic_istream:virtual public basic_ios - механизм для преобразования из последовательности символов в значения различного типа

- В классе перегружены методами класса operator>> для всех базовых типов:
  basic_istream& operator>>(int&); …
- typedef basic_istream<char> **istream;**
  typedef basic_istream<wchar_t> **wistream;**
- в прологе создаются объекты:
  istream cin;//символьный ввод – char
  wistream wcin;// wchar_t

basic_ostream<>
<iostream>

basic_ostream:virtual public basic_ios - механизм для преобразования значений различного типа в последовательность символов

- В классе перегружен методом класса operator<< для всех базовых типов: **basic_ostream& operator<<(int);**
- typedef basic_ostream<char> **ostream;**
  typedef basic_ostream<wchar_t> **wostream;**
- создаются объекты:

  ostream cout;//символьный вывод

  ostream cerr;//небуферизованный поток вывода для сообщенийоб ошибках

  ostream clog;//буфериз. поток

  wostream wcout;…

basic_iostream<>

basic_iostream **: public basic_istream,**

public basic_ostream

- ввод/вывод
- множественное наследование
- единственное предназначение – **база для более специализированных классов ввода/вывода** => кроме конструктора/деструктора ничего нет

Специализированные классы

Наследуют от basic_iostream:

- **basic_stringstream<> –** ввод/вывод в строку
- **basic_fstream <> –** ввод/вывод в файл

Вспомогательные классы поддержки ввода/вывода

- locale<>
- basic_streambuf<>

locale<>

- с каждым потоковым объектом библиотеки ввода / вывода C++ связана локаль **std::locale**
- локаль предоставляет фасеты для разбора и форматирования данных.
- C++11 - объект локали связывается с соответствующим объектом basic_regex.
- объекты локалей могут использоваться в качестве предикатов, выполняющих сравнение строк внутри стандартных контейнеров и алгоритмов;
- к ним возможен прямой доступ для получения или изменения хранимых в них фасетов.

basic_streambuf<>

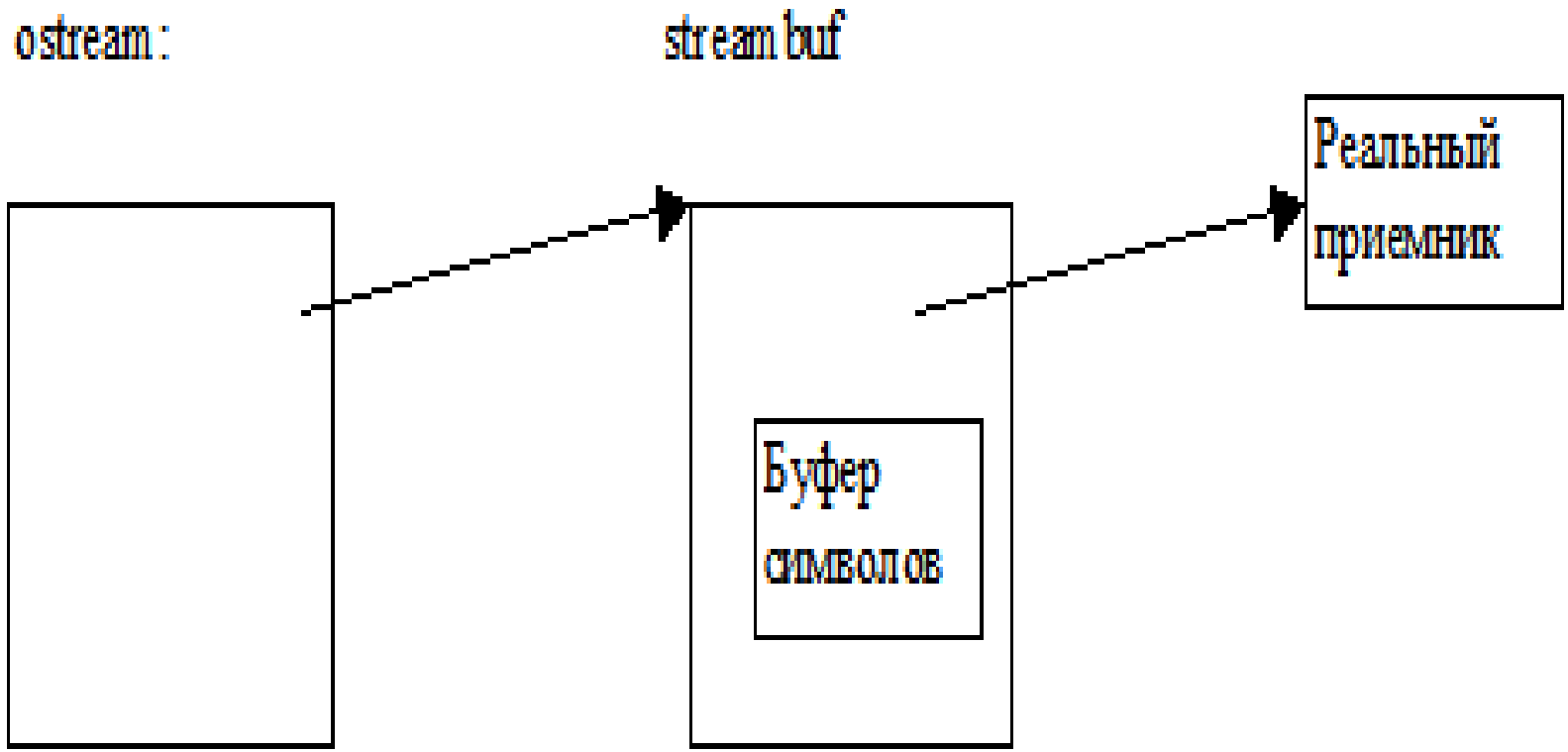
- организация и взаимосвязь буферов ввода-вывода
  - с физическими устройствами ввода-вывода
  - или строкой в памяти
- typedef basic_streambuf<char> streambuf;
- Методы и данные класса streambuf программист явно обычно не использует.

Замечания:

- в C++ поддерживается вся система ввода/вывода C. Но смешивать не стоит –порядок появления символов на экране не гарантируется
- **две версии библиотеки потоков ввода/вывода:**
  - <iostream.h> реализована посредством классов
  - <iostream> - посредством шаблонов (char и wchar_t)

Объектно-ориентированный вывод

Буферизация вывода



Буферизация означает:

данные, отправленные в поток вывода, **могут не появиться на экране**, пока буфер вывода не будет очищены (flushed)

=> существуют способы для того, чтобы заставить поток вывода немедленно вывести содержимое буфера на экран

Способы отображения буфера

- манипулятор cout<<…<<**endl**; не только вставляет ‘\n’, но и отправляет содержимое буфера в реальный приемник
- специальный метод класса: **cout.flush();**
- существует также flush-манипулятор: **cout<<flush;** //это перегрузка таким образом, что при этом вызывается flush(cout)
- **чтение** из потока ввода (или использование cerr, clog) принудительно очищает буфер вывода
- **exit()** очищает все буферы

Состояние потока

в классе ios:

- переменная **state**
- **enum**

```
public:
  enum io_state {
    goodbit, // нет ошибки 0X00
    eofbit,  // конец файла 0X01
    failbit, // последняя операция не выполнилась 0X02
    badbit,  // попытка использования недопустимой операции 0X04
    hardfail // фатальная ошибка 0X08
  };
```

- методы **bad(), eof(), fail(), good()**

Пример

```
int n;
std::cin >> n;
bool b = cin.good();
if (cin.bad()){ cout << "починке не подлежит!";}
else if (cin.eof()) { cout << "конец ввода";} //ctrl+c
else if (cin.fail()) { cout << "не int"; } //”qwerty”
```

Форматированный ввод/вывод

- универсальный,
- но не эффективный

Флаги ввода/вывода

Каждый поток ввода/вывода связан с набором флагов формата:

- флаги представляют битовые маски
- маски задаются в классе ios как константы типа **fmtflags** (enum).

Флаги ввода/вывода

- skipws
- left right internal
- boolalpha
- dec hex oct
- scientific fixed
- showbase showpoint showpos uppercase
- **adjustfield basefield floatfield**

Флаг	Назначение	По умолчанию
skipws	если установлен - начальные невидимые символы отбрасываются	установлен
dec, oct, hex	ввод/вывод целых в указанной системе счисления	dec
left, right	выравнивание вывода по левому/правому краю, если задана ширина поля вывода	right
internal	отступ между знаком и значением (-10)	
showbase	показывает префиксы для целых в формате oct (0), hex (0x)	нет
uppercase	выводит этот префикс и шестнадцатеричные буквы заглавными или строчными буквами	строчными
boolalpha	булевы переменные вводятся/ выводятся true, false	нет

Специфика флагов ввода/вывода

- Большинство именованных констант соответствуют одному биту
- среди них есть взаимоисключающие (**dec, hex, oct**)
- => помимо одиночных битов определены маски для работы с такими группами взаимоисключающих флагов - **adjustfield, basefield** и **floatfield**.

Группы флагов:

- **adjustfield -** internal | left | right
- **basefield -** dec | hex | oct
- **floatfield -** fixed | scientific

Методы для работы с флагами

- fmtflags **flags**() const; //возвращает текущие
- fmtflags **flags**(fmtflags *lFlags*); //устанавливает новые, возвращает старые, что полезно, если нужно восстановить прежнее состояние
- fmtflags **setf**(fmtflags *lFlags*); //добавляет указанные флаги к существующим
- fmtflags **unsetf**(fmtflags *lFlags*); //обнуляет (сбрасывает) указанные флаги
- fmtflags **setf**(fmtflags *set*, fmtflags *unset*); //добавляет указанные первым аргументом флаги, сбрасывает флаги, указанные вторым параметром.

Примеры:

1. int n=15;

   cout. flags(ios::hex | ios::showbase);
   cout<<n<<endl; //**0xf**

//или

   cout.setf(**ios::hex ,ios::basefield**);

   cout.setf(**ios::showbase | ios::uppercase**);

   cout<<n<<endl;//**0XF**

Примеры:

2.

```
bool b=true;
cout<<b<<endl; //1

cout.setf(ios::boolalpha);
cout<<b<<endl;    //выводит «true»

cin.setf(ios::boolalpha);
cin>>b;  //Вы печатаете false или true - вводит 0 или 1
```

Примеры

- 3.

```
double d = 1.23;
cout.setf(ios_base::scientific,
                        ios_base::floatfield);
cout<<d<<endl; //1.230000E+000
```

???

```
ios::fmtflags f =
        ios::hex | ios::showbase | ios::uppercase;
cout.flags(f);
cout<<255<<endl; //???
```

Методы width(), precision(), fill()

width() количество знакомест	для вывода задает ширину поля (количество позиций). После вывода возвращается к значению по умолчанию	По умолчанию при выводе любого значения оно занимает столько позиций, сколько символов выводится
precision() Точность плавающих	плавающие при выводе **округляются** до указанного значения. Если формат - scientific или fixed, то количество цифр после запятой	по умолчанию 6 цифр
fill() символ-заполнитель	установка нового символа для заполнения	по умолчанию - пробел

Специфика

- **width()** действует только на одну операцию ввода/вывода
- **precision()** и **fill()** – до установки новых значений

width()

```
cout.width(10);
cout<<"FF"<<endl;     //по умолчанию справа

cout.width(10); //если снова не зададим поле вывода, оно "сбросится"
cout<<-2<<endl; //-        2

cout.width(10);
cout.setf(ios::left,ios::adjustfield);
cout<<"BB"<<endl;    //слева

cin.width(4);
char ar[10];
cin>>ar; //сколько бы не вводили – в массив запишет 3 байта+0, но все остальное останется в буфере ввода и будет использовано при следующей операции!
```

precision()

```
//сколько цифр после «.»
cout.precision(3);
cout.setf(ios_base::scientific, ios_base::floatfield);
cout<<1.23456<<endl; //1.235E+000
cout.setf(ios_base::fixed, ios_base::floatfield);
cout<<1.23456<<endl; //1.235
```

fill()

```
cout.width(10);
cout << -2 << 3<<endl; //-2_____3

cout.width(10);
cout.fill('#');
cout<<-2<<3<<endl;// -2########3

cout.width(10);
cout.setf(ios::internal,ios::adjustfield);
cout.fill('#');
cout << -2 << 3<<endl;// -#########23
```

Манипуляторы ввода/вывода

- набор специальных функций

Ключевая идея – при «цепочечном выводе» **вставлять между объектами вызов требуемой функции форматирования, но в удобном виде** (если операции записываются как отдельные инструкции, логические связи между ними не очевидны, и код получается громоздкий)

Вывод «таблицы» 2*2 без манипуляторов

```
string s[4] =     { "One", "Two", "Three", "Four" };

cout.width(10);
cout << s[0];
cout.width(10);
cout << s[1] << endl;
cout.width(10);
cout << s[2];
cout.width(10);
cout << s[3] << endl;
```

Вывод «таблицы» с манипуляторами

```
string s[4] =  {"One", "Two", "Three", "Four"};

cout << setw(10) << s[0]
  << setw(10) << s[1] << endl
  << setw(10) << s[2]
  << setw(10) << s[3] << endl;
```

Замечание:

- setw() вызывает ту же функцию width()
- и возвращает ios_base& для обеспечения цепочечного вывода

Пояснение

cout<<flush; //перегруженный оператор <<, который принимает в качестве аргумента указатель на функцию и вызывает ее

это означает:

cout.operator<<(flush); //передается указатель на глобальную функцию вида ios_base& **flush**(ios_base&);

что в свою очередь:

-> flush(cout) -> cout.flush();

Пояснение

cout<< **hex**<<255<<**endl**; //передается указатель на функцию ios_base& **hex**(ios_base& io);

Которая:

- вызывает:

  io.setf(ios_base::hex, ios_base::basefield);

- и возвращает io

Замечания:

- Манипуляторы существуют для всех тех настроек, которые можно выполнить с помощью флагов и функций width(), fill(), precision() + манипуляторы без параметров (endl)
- имена многих манипуляторов совпадают с именами флагов и методов
- при использовании манипуляторов без параметров – не ставьте скобки,
- при использовании манипуляторов с параметрами – не забудьте подключить <iomanip>

Имя	Действие	Используется
dec, hex, oct	Устанавливает систему счисления (десятичная по умолчанию)	для ввода-вывода
ws	Ввод пробельных символов (по умолчанию фильтруются)	только для ввода
endl	Вставляет символ новой строки и очищает поток вывода	только для вывода
flush	Очищает поток вывода	только для вывода
ends	Добавляет нулевой байт	только для ostrstream
setbase(int)	Устанавливает систему счисления в 0, 8, 10 или 16	для ввода-вывода целых : по умолчанию - 0 (десятичная)
resetiosflags(long)	Очищает биты формата во вводном и выводном форматах числом, заданным аргументом	для ввода-вывода
setiosflags(long)	Устанавливает биты формата во вводном и выводном потоках в соответствии с аргументом	для ввода-вывода
setfill(int)	Устанавливает символ заполнитель (по умолчанию таким символом является пробел)	для вывода
setprecision(int)	Устанавливает точность для типа с плавающей точкой (по умолчанию точность 6)	для вывода
setw(int)	Устанавливает ширину поля	для ввода-вывода
ends	Вставляет завершающий нулевой символ строки	только для ostrstream

Пример:

```
double d = 1.234567;

cout<<setprecision(4)<<d; //1.2345 - 4 цифры с округлением. Количество цифр остается 4 до следующего setprecision()
cout<<setw(10)<<setfill('#')<<d<<endl;//#####1.235
```

Манипуляторы, определяемые пользователем

```
ostream& mymanip(ostream& os)
{
        os.width(10);
        os.fill('#');
        os.precision(4);
        return os;
}
#include <iostream>
int main()
{
        std::cout<< mymanip<<1.23456<<endl;
}
```

Файловый ввод/вывод

Независимость от устройства обеспечивается:

Классы потоков ввода-вывода (istream, ostream и iostream для входных, выходных и комбинированных потоков соответственно) являются базовыми для:

- создания файловых (ifstream, ofstream, fstream)
- и строковых потоков (istringstream, ostringstream, stringstream).

Замечания:

- #include <fstream>
- namespace std
- если объекты cin, cout… создаются в стартовом коде, то объекты для файлового ввода/вывода требуется создавать «вручную» => конструктор

ifstream(const char* *szName*, int *nMode* = ios::in, int *nProt* = filebuf::openprot); //по умолчанию открывается для ввода в текстовом режиме. Файл может быть sharing (разделяемым)

Примеры

ifstream f(“My.txt”); //если файл существует, он открывается для чтения, если нет – создается и открывается для чтения

ifstream f(“My.txt”, ios::nocreate | ios::in);

//если существует такой файл, открывается, если нет – отказ в доступе

ifstream f(“My.txt”, ios::binary | ios::in, filebuf::sh_none);//файл открыт для чтения в двоичном режиме, совместный доступ к файлу запрещен

Использование default-конструктора и метода open()

```
ifstream f;
f.open(“My.txt”);
```

Проверка получения доступа к файлу

```
ifstream f(“My.txt”);
if(!f){cout<<”Failed!”;…return;}
```

Метод close()

```
f.close();
```

Замечание: если программист явно не вызвал close() ???

Непоследовательный доступ

Система ввода/вывода C++ управляет двумя указателями, связанными с файлом:

- указатель считывания (**get pointer**), который задает позицию, начиная с которой произойдет следующее считывание (ввод)
- указатель записи (**put pointer**), который задает позицию в файле, с которой начнется следующая запись

Важно! при каждом вводе и выводе соответствующий указатель **перемещается** по результатам этой операции.

Функции для формирования get pointer

- istream& seekg(streampos *pos*);//задает новую текущую позицию (streampos==long)
- istream& seekg(streamoff *off*,
  ios::seek_dir *dir*);

  //*off* - смещение от того, что задано вторым параметром (streamoff==long)
  откуда считать смещение: **ios::beg** – от начала
  ios::cur – от текущего
  ios::end – от конца

Функции для формирования get pointer и put pointer

- ostream& seekp(streampos *pos*);
- ostream& seekp(streamoff *off*, ios::seek_dir *dir*);

Определение текущей позиции в файле

- pos_type tellg();
- pos_type tellp();

Неформатированный ввод/вывод

Применяется в основном для файлового ввода/вывода для компактного хранения данных

get()

Перегрузка	Описание
int get();	Извлекает один байт из потока и возвращает его значение
get(char* buf, int nCount, char delimeter);	Извлекает байты из потока пока не встретит байт (символ)- разделитель (delimeter) или пока не извлечет а) nCount байт, б) встретит конец файла. Извлеченные байты помещаются в buf + добавляется 0-ой байт. Замечание: если строка оказалась длиннее nCount, то введено будет nCount-1 + 0
get(char&);	Извлекает один байт из потока и помещает в указанный байт.
get(streambuf&, char delimeter);	Сохраняет символы в объекте streambuf

Замечание:

во всех случаях сам delimeter не извлекается из потока и не записывается в указанный буфер!!!

Пример

```
char ar[3][4];//={0};
for(int i=0; i<3; i++)
{
        cin.get(ar[i],4,'\n'); //считывает, пока не встретится указанный символ-разделитель или пока не будет считано  4 значения
        while(cin.get()!='\n'); //тогда “дочитываем” - извлекаем из потока, но никуда не записываем
}
```

getline()

istream& getline(char* *pch*, int *nCount*, char *delim* = '\n');

Извлекает байты из потока, пока не встретит delimeter или не считает nCount–1 байт в указанный массив + дополнит 0-ым байтом

Главное отличие:

функция извлекает из потока delimeter, но **не** записывает в указанный буфер

Пример

```
char ar[3][4];//={0};
for(int i=0; i<3; i++)
{
        cin.getline(ar[i],4,'\n'); //если вводим 3 и меньше, то
                                   все ОК -AAA,BB,C
}
```

ignore()

istream &ignore(int count = 1, int target = EOF);

Извлекает символ из **istream**, пока :

- функция не извлечет **count** символов;
- не будет обнаружен символ **target**;
- не будет достигнут конец файла.

read(), write(), gcount()

istream& read(char* *pch*, int *nCount*);

- извлекает из потока nCount байт или пока не достигнут конец файла
- узнать, сколько действительно было считано байт, можно с помощью gcount()

Пример

```
ifstream iff(“My.dat”, ios::in | ios::binary);
if(!iff){…}
else
{
        double d;
        iff.read( reinterpret_cast<char*>(&d),
                        sizeof(double) );
        iff.close();
}
```

Форматирование в памяти

std::streamstring

#include <sstream>

класс stringstream

позволяет связать поток ввода-вывода со строкой в памяти:

- всё, что выводится в такой поток, добавляется в конец строки;
- всё, что считывается из потока, извлекается из начала строки.

Преобразование типа с помощью stringstream

```
std::stringstream ss;
ss << "22";
int k = 0;
ss >> k; //22
std::cout << k << std::endl;
```