# C++ I/O

Prepared

by

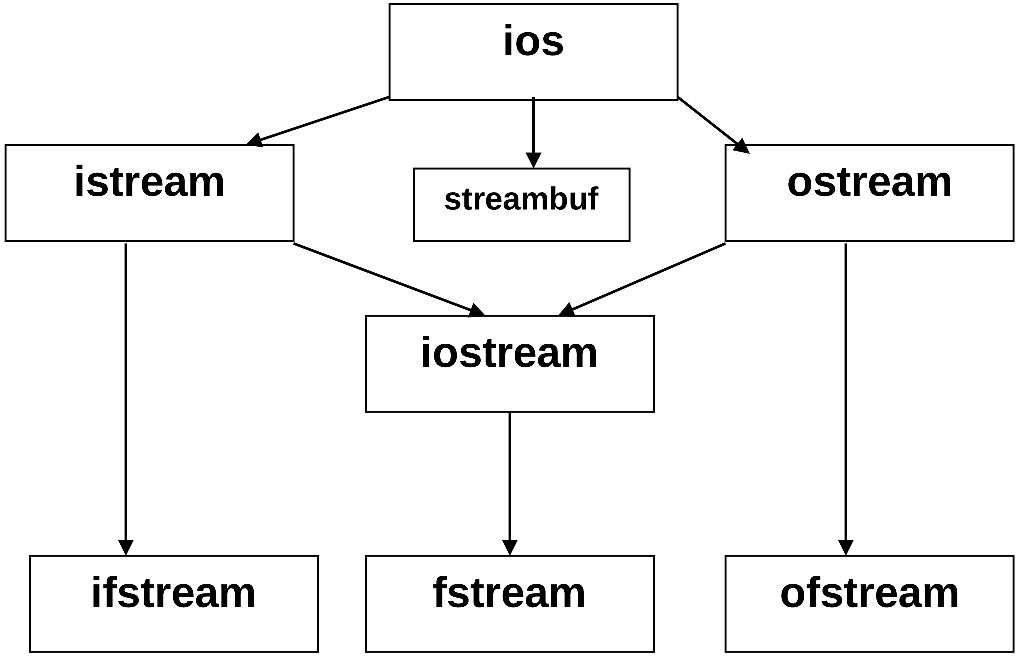
Naveen choudhary

# C++ I/O

include <iostream.h>
using namespace std;

A stream is a logical device that either produces or consumes information. A stream is  linked to physical device by the I/O system. Device can be of different types like keyboard, disk but stream provides a uniform interface to the user.

Stream → logical device, connected to physical device
Objective → uniform interface



**Note: -** Some more classes are there in the hierarchy but is not shown here

***Predefined object of I/O streams***

| | | | |
|---|---|---|---|
| ***cin*** | Standard Input | Keyboard | (Object of istream with assignment ) |
| ***cout*** | Standard Output | Screen | (Object of ostream with assignment) |
| ***cerr*** | Stanadrd Error | Screen | ---------- do ---------------- |
| ***clog*** | Buffered version of cerr | Screen | ---------- do ---------------- |

**Formatted I/O Operation** : Formatting data to be Input / Output

*Two Ways :*

**Using ios member function** :- Using *ios* member function & setting status flags of *ios* class using the member function

**Using manipulators** :- These are special functions that can be included as part of the I/O expression

# Formatting I/O using member functions

*ios Formatting* flags *(there are two type of flags on/off and flags that work in a group )*

Skipws When set, leading, white-space characters (space, tab & new lines) are ignored/skipped on/off flag
when performing input on a stream

left Left justified [12.34 _________ ] (Output)
right Right justified [_____12.34 ] (Output)
internal Use padding between sign or base indicator & number [ + 12.54 ]

dec Convert to decimal (output)
oct Convert to Octal (output)
hex Convert to Hexadecimal (output)

boolalpha When set, Booleans can be input or output using the keyboard True or False

showbase Use base indicator on output ( o for Octal, 0x for Hex ) -- on/off flag

showpoint Show decimal point on output (ie causes a decimal point & trailing zeros to be displayed
for all floating point output whether needed or not) -- -- on/off flag

uppercase By default, when scientific notation is displayed, the e is in lowercase. Also when a hexadecimal value is displayed, the x is in lower case. When uppercase is set, these characters are displayed in uppercase -- on/off flag

showpos Display + before positive intergers -- -- on/off flag

scientific Use exponential format on floating point output [9.1234E2]

fixed Use fixed format on floating point output [912.34]

unitbuf When unitbuf is set, the buffer is flushed after each insertion operation -- on/off flag

# Formatting I/O using member functions …..contd

stdio Flush stdout, stderror after insertion ( used when you have a program that uses both iostream and the C standaed I/O library. If you discover your iostream output and printf() output are occuring in the wrong order, try setting this flag -- on/off flag

basefield Oct, Dec & Hex fields are collectively called basefield

adjustfield Left, right, internal are collectively called adjustfield

floatfield Scientific, Fixed are collectively called floatfield

The ios class declares a bitmask enumeration called fmtflags in which the above mentioned flags are defined

## Setting Format Flag

fmtflag setf (fmtflag flags1)

**The function returns the previous setting of the format flag & turns on those flags specified flags1**

stream.setf (ios :: showpos)

**you can set more than one flag with single setf command using**

stream.setf (ios :: showpoint | ios :: showpos);

## Clearning Formatting Flags

void unsetf (fmtflags flags);

**The flags specified by flags are cleared (all other flags are unaffected )**

Example-1

```
#include <iostream>
using namespace std;
int main()
{
  cout.setf(ios::showpoint);
  cout.setf(ios::showpos);
  cout << 100.0; // displays +100.0
  return 0;
}
```

Example-2

```
#include <iostream>
using namespace std;
int main()
{
  cout.setf(ios::uppercase |
ios::scientific);
  cout << 100.12;  // displays
                          //1.0012E+02
  cout.unsetf(ios::uppercase); // clear
                                 //uppercase
  cout << " \n" << 100.12; // displays
                               //1.0012e+02
  return 0;
}
```

## An overloaded form of setf ( )

Used for group flags

fmtflags setf (fmtflags flag1. fmtflags flag2);

**In this version, only the flags specified by flag2 are affected. They are first cleared and then set according to the flag specified by flag1. note that even if flag1 contains other flags, only those specified by flag2 will be affected. The previous flags setting is returned.**

Example – 1

```
#include <iostream>
using namespace std;
int main( )
{
  cout.setf(ios::showpoint | ios::showpos,
               ios::showpoint);
  cout << 100.0; // displays 100.0, and not +100.0
  return 0;
}
```

*Example – 2*

```
#include <iostream>
using namespace std;
int main()
{
  cout.setf(ios::hex, ios::basefield);
  cout << 100; // this displays 64 ie
                         100 in hex form
  return 0;
}
```

*Example – 3***(skip)**

```
#include <iostream>
using namespace std;
int main()
{
  cout.setf(ios::showpos, ios::hex); //
                      //error, showpos not set
  cout << 100 << '\n'; // displays
                                 100, not +100
  cout.setf(ios::showpos, ios::showpos);
                          // this is correct
  cout << 100; // now displays +100
  return 0;
}
```

**Examining the format flags**

```
fmtflag flags ( );
```

Simply returns the current setting of each format flag

**Setting all flags**

```
fmtflag flags(fmtflag f);
```

The bit pattern found in *f* is used to set the formatflag associated with the stream. Thus, all format flags are affected. The function returns the previous setting.

*Example – 1*

```
#include <iostream>
using namespace std;
void showflags() ;
int main()
{
 // show default condition of format flags
  showflags();
  cout.setf(ios::right | ios::showpoint |
             ios::fixed);
  showflags();
  return 0;}
```

```
// This function displays the status of the format flags.
void showflags()
{
  ios::fmtflags f;
  long i;
  f = (long) cout.flags(); // get flag settings
  // check each flag
  for(i=0x4000; i; i = i >> 1)
    if(i & f) cout << "1 ";
    else cout << "0 ";
  cout << " \n"; }
```

**Output:**

```
0 0 0 0 0 1 0 0 0 0 0 0 0 0 1
0 1 0 0 0 1 0 1 0 0 1 0 0 0 1
```

*Example – 3*

```
#include <iostream>
using namespace std;
void showflags();
int main()
{
  // show default condition of format flags
  showflags();
  // showpos, showbase, oct, right are on,
 //  others off
  long f = ios::showpos | ios::showbase |
               ios::oct | ios::right;
  cout.flags(f);  // set all flags
  showflags();
  return 0;
}
```

**Output –**

```
0 0 0 0 0 1 0 0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 1 0 1 0 1 0 0 0
```

## Other ios Functions

ch = fill ()      reads the current fill character (default is space ) {int ios :: fill() }

fill (ch)      Set the fill character & returns the previous fill character { int ios :: fill(int n )

p = precision ()  Get the precision (no. of digits displayed after floating point)

precision (p)      Set the precision

w = width ()      Get the current field width (in characters)

width ()      Set the current field width

Note :: the width is reset to zero by each insertion and extraction. If we want to have a constant width, we need to call width() after each insertion or extraction

***Example-1 ()***

```
#include <iostream>
using namespace std;
int main()
{
cout.precision(4) ;//total 4 digits will be displayed
                    //if used with cout.setf(ios ::
                    //fixed) then precision will mean
                   //digits after decimal point only
  cout.width(10);
  cout << 10.12345 << "\n";  // displays 10.12
  cout.fill('*');
  cout.width(10);
  cout << 10.12345 << "\n"; // displays ***10.12
  // field width applies to strings, too
  cout.width(10);
  cout << "Hi!" << "\n"; // displays *******Hi!
  cout.width(10);
  cout.setf(ios::left); // left justify
  cout << 10.12345; // displays 10.12*****
  return 0;
}
```

**24. Write a program implementing basic operation of *class ios* i.e. setf,unsetf,precision etc.   (skip)**

```
#include     <iostream.h>
#include     <conio.h>
void main( )
{
    int   i  = 52;
    float a  = 425.0;
    float b  = 123.500328;
    char  str[ ]  = "Dream. Then make it happend!";
    clrscr( );
    cout.setf( ios::unitbuf );
    cout.setf( ios::stdio );
    cout.setf( ios::showpos );
    cout << i << endl;
    cout.setf( ios::showbase );
    cout.setf( ios::uppercase );
    cout.setf( ios::hex, ios::basefield );
    cout << i << endl;
    cout.setf( ios::oct, ios::basefield );
    cout << i << endl;
    cout.fill( '0' );
    cout << "Fill character " << cout.fill( ) << endl;
    cout.setf( ios::dec, ios::basefield );
    cout.width( 10 );
    cout << i << endl;
    cout << setf( ios::left, ios::adjustfield );
    cout.width( 10 );
    cout << i << endl;
    cout.setf( ios::internal, ios::adjustfield );
    cout.width( 10 );
```

```
cout << endl;
    cout << endl;
    cout.width( 10 );
    cout << str << endl;
    cout.width( 40 );
    cout.setf( ios::left, ios::adjustfield );
    cout.width( 40 );
    cout << str << endl;
    cout.precision( 6 );
    cout << "Precision" << cout.precision( );
    cout.setf( ios::showpoint );
    cout.unsetf( ios::showpos );
    cout << endl << a;
    cout.unsetf( ios::showpoint );
    cout << endl << a;
    cout.setf( ios::fixed, ios::floatfield );
    cout << endl << b;
    cout.setf( ios::scientific, ios::floatfield );
    cout << endl << b;
    b = 5.375;
    cout.precision( 14 );
    cout.setf( ios::fixed, ios::floatfield );
    cout << endl << b;
    cout.setf( ios::scientific, ios::floatfield );
    cout << endl << b;
    cout.unsetf( ios::showpoint );
    cout.unsetf( ios::unitbuf );
    cout.unsetf( ios::stdio );
}
```

# Manipulators

```
cout<< setiosflags (ios::fixed) << setiosflags (ios::showpoint);
```

## Non-Argument ios Manipulators (include < iostream.h> )

| | |
|---|---|
| ws/skipws | Turn on whitespace skipping on input |
| Dec | Convert to decimal |
| Oct | Convert to Octal |
| Hex | Convert to Hexadecimal |
| Endl | Insert newline and flush the output stream |
| Ends | Insert Null character to terminate a output string |
| Flush | Flush the output stream |
| Lock | Lock file handle |
| Unlock | Unlock file handle |
| Boolalpha | Turns on boolalpha flag I/O |
| Noboolalpha | Turns off boolalpha Output |
| Fixed | Use fixed notation for printing float values |
| Internal | Use padding between sign or base indicator and value |
| Left | Left align, pad on right |
| Noshowbase | Turns off showbase flag Output |
| Noshowpoint | Do not Show decimal point & trailing zeros for float values |

| | |
|---|---|
| Noshowpos | Turns off showpos flag Output |
| Showpoint | Turns on showpoint flag Output |
| Showpos | Turns on showpos flag Output |
| Skipws | Turns on skipws flag Input |
| Noskipws | Turns off skipws flag Input |
| Unitbuf | Turns on unitbuf flag Input |
| Nounitbuf | Turns off unitbuf flag Output |
| Nouppercase | Turns off uppercase flag Output |
| Left | Turns on Left flag Output |
| Right | Turns on Right flag Output |
| Scientific | Turns on scientific flag Output |

Note that manipulators affect only the data that follows them in the stream, not the data that precedes them. Table summarizes the important manipulators that take arguments. You need the IOMANIP header file for these function.

**# include <iomanip.h>**

## Table
## ios Manipulators with Arguments

| Manipulators | **Arguments** | **Purpose** |
|---|---|---|
| setw (int ) | Field width (int) | Set field width for output |
| setfill (int ) | Fill character (int) | Set fill character for output (default is space) |
| setprecision (int ) | Precision (int) | Set precision (number of digits displayed after decimal point) |
| setiosflags (long) | Formatting flags (long) | Set format flags specified by n. setting remains in effect until next change |
| resetiosflags (long) | Formatting flags (long) | Clear specified flags. setting remains in effect until next change |
| setbase (int ) | Base (int) | Set the no. base to base ( 0 – base 10, 8 – octol, 16 – hex ) |

Example –1

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  cout << hex << 100 << endl;//64
  cout << setfill('?') << setw(10) << 2343.0;//??????2343
 return 0;}
```

Example –2  **(skip)**

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  cout.setf(ios::hex, ios::basefield);
  cout << 100 << "\n";  // 100 in hex ie 64
  cout.fill('?');
  cout.width(10);
  cout << 2343.0;
  // cout << 1232.0 → now fill() & width()
  //will not be applicable here
  return 0;
}
```

*Example –3* ***(SKIP )***

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  cout << setiosflags(ios::showpos);
  cout << setiosflags(ios::showbase);
  cout << 123 << " " << hex << 123;//
                //+123 0x7b
  return 0;}
```

```
#include<iostream.h>
#include <conio.h>
#include <iomanip.h>
void main( )
{
int i=52;
float a = 425.0 ;
float b = 123.500328 ;
char str[ ] = "Dream. Then make it happen!" ;
clrscr( ) ;
cout « setiosflags ( ios::unitbuf | ios::stdio | ios::showpos )
;
cout« I «endl;  // +52
cout « setiosflags ( ios::showbase | ios::uppercase ) ;
cout « hex « i « endl ;  // 0X34
cout « oct « i « endl ;  // O64
cout « setfill ( '0' ) ;
cout « "Fill character:" « cout.fill( ) « endl ;//0
cout«dec«setw(10)«i«endl; // 0000000+52
cout « setiosflags ( ios::left)
«dec «setw(10)«i«endl; // +520000000
cout « setiosflags ( ios::internal )
«dec«setw(10)«i«endl;  // +000000052
cout « i « endl ; // +52
cout « setw (10 ) « str « endl ; // dream.then make it happen
cout « setw ( 30 ) « str « endl ;//00000dream.then make it
                                                    //happen
cout « setiosflags ( ios::left ) « setw ( 40 ) « str « endl ;
            //dream.then make it happen00000

cout«precision ( 6 ) ;
cout « "Precision: " « cout.precision( ) ; // +6
cout « setiosflags ( ios::showpoint ) «
```

```
resetiosflags ( ios::showpos )
«endl«a; //425.000000
cout « resetiosflags ( ios::showpoint )
«endl«a; //425
cout « setiosflags ( ios::fixed ) « endl « b ; //123.500328
cout « setiosflags ( ios::scientific ) « endl « b ;
                                     //1.235003E+02
b = 5.375 ;
cout.precision ( 14 ) ;
cout « setiosflags ( ios::fixed ) « endl « b ; // 5.375
cout « setiosflags ( ios::scientific ) « endl « b ;
                                         //5.375E+00
cout « resetiosflags ( ios::showpoint | ios::unitbuf |
ios:stdio ) ;
}
```

*Example –4*

```
#include <iostream>
using namespace std;
int main()
{
  bool b;
  b = true;
  cout << b << " " <<boolalpha<< b << endl; // 1 true
  cout << "Enter a Boolean value: ";//false
  cin >> boolalpha >> b;
  cout << "Here is what you entered:"<<b; //false
  return 0;
}
```

## Overloading << & >> (insertion & extraction operator )

***Inserter overloading***

```
ostream & operator << ( ostream &stream, class_type obj)
{
                //body of inserter
                return stream;
}
```

*Example –4*

```
#include <iostream>
#include <cstring>
using namespace std;
class phonebook {
public:
  char name[80];
  int areacode;
  int prefix;
  int num;
  phonebook(char *n, int a, int p, int nm)
  {
    strcpy(name, n);
    areacode = a;
    prefix = p;
    num = nm;
  }
};
// Display name and phone number.
ostream &operator<<(ostream &stream, phonebook o) // can not be member function of
                                                   //phonebook
{
  stream << o.name << " ";
  stream << "(" << o.areacode << ") ";
  stream << o.prefix << "-" << o.num << "\n";
  return stream; }     // must return stream
```

```
int main()
{
  phonebook a("Ted", 111, 555, 1234);
  phonebook b("Alice", 312, 555, 5768);
  phonebook c("Tom", 212, 555, 9991);
  cout << a << b << c;
  return 0;
}
```

**In the preceding program, notice that the phone book, inserter is not a member of phone book and it can not be as left side of the << operator is stream and not the phonebook class**
Beside 1st argument must be of type ostream & not of phone book.

But if inserter operator is not a member function then we can’t access the private members of the class phonebook in the overloaded inserter operator.
***Solution : make the inserter operator function , friend of the phonebook class.***

```
#include <iostream>
#include <cstring>
using namespace std;
class phonebook {
  // now private
  char name[80];
  int areacode;
  int prefix;
  int num;
public:
  phonebook(char *n, int a, int p, int nm)
  {
    strcpy(name, n);
    areacode = a;
    prefix = p;
    num = nm;
  }
  friend ostream &operator<<(ostream
&stream, phonebook o);
};
// Display name and phone number.
ostream &operator<<(ostream &stream,
phonebook o)
{
  stream << o.name << " ";
  stream << "(" << o.areacode << ") ";
stream << o.prefix << "-" << o.num << "\n";
return stream; // must return stream
}
int main()
{
  phonebook a("Ted", 111, 555, 1234);
  phonebook b("Alice", 312, 555, 5768);
  phonebook c("Tom", 212, 555, 9991);
  cout << a << b << c;
  return 0;
}
```

Inside inserter operator we have used:

1. stream << o.name << “    ”;

   we could have also used

2. cout << o.name << “   ”;

but 1 is more generic & can be used with any ostream. Whereas 2 can only be used with cout.

***Overloading extractor operator***

```
istream & operator >> (istream & stream, class_type &obj)
{
            //body of extractor
            return streanm;
}
```

Example

```
#include <iostream>
#include <cstring>
using namespace std;
class phonebook {
  char name[80];
  int areacode;
  int prefix;
  int num;
public:
  phonebook() { };
  phonebook(char *n, int a, int p, int nm)
  {
    strcpy(name, n);
    areacode = a;
    prefix = p;
    num = nm;
  }
  friend ostream &operator<<(ostream &stream, phonebook o);
  friend istream &operator>>(istream &stream, phonebook &o);
};
```

```
// Display name and phone number.
ostream &operator<<(ostream &stream,
phonebook o)
{
  stream << o.name << " ";
  stream << "(" << o.areacode << ") ";
  stream << o.prefix << "-" << o.num << "\n";
  return stream; // must return stream
}
// Input name and telephone number.
istream &operator>>(istream &stream,
phonebook &o)
{
  cout << "Enter name: ";   stream >> o.name;
  cout << "Enter area code: ";
  stream >> o.areacode;
  cout << "Enter prefix: ";  stream >> o.prefix;
  cout << "Enter number: ";  stream >> o.num;
  cout << "\n";
  return stream;
}
int main()
{
  phonebook a;
  cin >> a;   cout << a;
  return 0;}
```

**(skip)** Actually extractor operator defined in the previous example is not correct. If istream is cin then ok but if istream is some disk file then we should not use cout (as we have used in the example).

## Creating your own manipulator function

Manipulator can be with parameter or parameterless.

1. Parameterized manipulators can take some argument in addition to istream/ ostream.
2. //(skip) Parameterless manipulators can take only a single argument which is of type istream/ ostream.
3. //(skip) creating customized parameterized manipulators is compiler dependent & you need to go through, the compiler documentation. So these type of manipulators are not further discussed.

4. O/P manipulators

```
    ostream & manip_name(ostream
        &stream)
{
    // your code here
}
    istream & manip_name (istream &
        stream)
{
    // your code here
}
```

The call to these manipulators is from the insertion/ extraction operation as shown below & we need not pass arguments during the call.

```
cout << 256 << “  ” << sethex <<256;
```

**Or**

```
cin>> getpass >> pw
```

Example-1(skip)

```
#include <iostream>
#include <iomanip>
using namespace std;
// A simple output manipulator.
ostream &sethex(ostream &stream)
{
  stream.setf(ios::showbase);
  stream.setf(ios::hex, ios::basefield);
  return stream;
}
int main()
{
  cout << 256 << " " << sethex << 256;
  return 0;
}
```

*Example*-2

```
#include <iostream>
#include <iomanip>
using namespace std;
// Right Arrow
ostream &ra(ostream &stream)
{
  stream << "---------> ";
  return stream;
}
// Left Arrow
ostream &la(ostream &stream)
{
  stream << " <-------";
  return stream;
}
int main()
{
 cout << "High balance " << ra << 1233.23 << "\n";// -- > 1233.23
 cout << "Over draft " << ra << 567.66 << la;// --- >567.66 < ---
  return 0;
}
```

*Example*-3

```
#include <iostream>
#include <cstring>
using namespace std;
// A simple input manipulator.
istream &getpass(istream &stream)
{
  cout << '\a';  // sound bell
  cout << "Enter password: ";
  return stream;
}
int main()
{
  char pw[80];
  do {
    cin >> getpass >> pw;
  } while (strcmp(pw, "password"));
  cout << "Logon complete\n";
  return 0;
}
```

# FILE I/O

## istream function

| Function | Purpose |
|---|---|
| >> | extraction for all basic(and overloaded) types |
| get (ch) | Extract one character into ch |
| get (str) | Extract characters into array *str*, until NULL |
| get (str, max) | Extract upto max characters into array |
| get (str, DELIM) | Extract characters into array str until specified delimeter (typically ‘\n’). Leave delimiting character in stream |
| get (str, max, DELIM) | Extract characters into array str until MAX characters or the delimeter character. Leave delimiting character in stream. |
| getline (str, max, DELIM) | Extract characters into array str until MAX characters or the delimeter character. Extract delimiting character |
| putback (ch) | Insert last character read back into input stream |
| ignore (max, DELIM) | Extract and discard upto MAX characters until (and including) the specified delimiter ( typically ‘\n’ ). |
| peek (ch) | Read one character, leave it in stream |
| count = gcount () | Return number of character read by a immediately preceding call to get(), getline (), or read () |
| read (str, max) | For files – extract upto to MAX character into str, until EOF |
| | |
| seekg (pos, seek_dir) | Set distance (in bytes) of file pointer from specified place in the file. seek_dir can be ios::beg, ios::cur, ios::end |
| pos = tellg(pos) | Return position (in bytes) of file pointer from start of file. |

## ostream function

| Function | Purpose |
|---|---|
| << | Formatted insertion for all basic (and overloaded) types |
| put (ch) | Insert character ch into stream |
| flush () | Flush buffer contents and insert newline |
| write (str, SIZE) | Insert SIZE character from array str into file |
| seekp (position) | Set distance in bytes of file pointer from start of file |
| seekp (position, seek_dir) | Set distance in bytes of file pointer, from specified place in file. seek_dir can be ios::beg, ios::cur or ios::end |
| pos = tellp() | Return position of file pointer |

| Error status flags Name | Meaning |
|---|---|
| goodbit | No errors (no flags set, value =0) |
| eofbit | Reached end of file |
| failbit | Operation failed (user error, premature EOF) |
| badbit | Invalid operation (no associated streambuf) |
| hardfail | Unrecoverable error |

## Function for Error flags

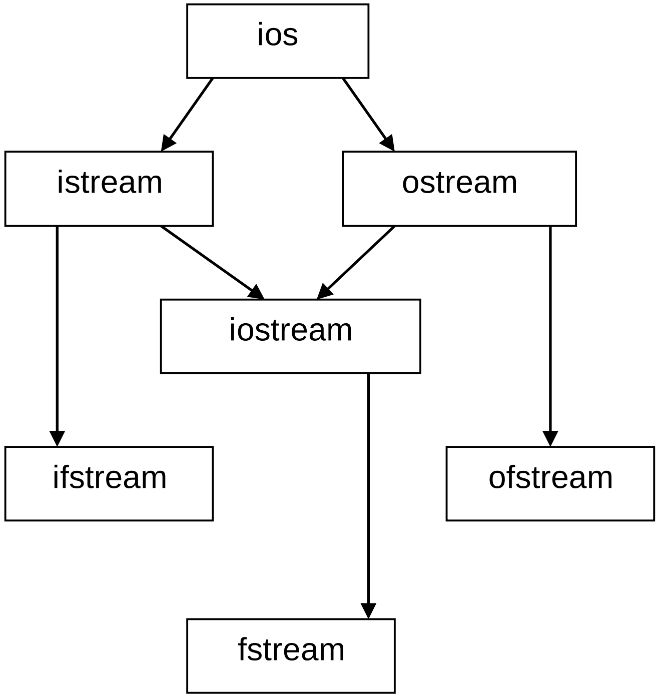
| **Function** | **Purpose** |
|---|---|
| int = eof () | Returns true if EOF flag set |
| int = fail () | Returns true if failbit or badbit flag set |
| int = bad () | Returns true if badbit or hardfail flag set |
| int = good () | Returns true if everything OK; no flag set |
| clear (int = 0) | With no arguments, clears all error bits; otherwise sets specified flags, as in clear (ios::failbit) |

Note: (1)
**Probem of using cout with files**
**ofstream outfile;**
**outfile.open("SAMPLE.TXT");**
**outfile <<ch <<endl << j << endl << a <<endl <<str**;

in the above statement we need to give **endl** ie "\n" after each variable because since all these variables will be written in the file in char format and without the delimiter the >>(extrraction operator) will not be able to make out when the value for a particular variable ends.

Note (2):**(skip)**
The ifstream, ofstream and fstream classes are declared in the header file 'fstream.h'. this file is also includes the 'iostream.h' header file, so there is no need to do it explicitly.

#include <fstream.h> {for disk I/O to/from files}



***First create file streams***

```
ifstream   in;            //input
ofstream  out;          //output
fstream    io;            //input & output
```

**After creating file streams, you need to associate these stream to files with the help of open ().**

void ifstream::open (const char *filename, ios::openmode) → ios::in     (default)
void ofstream::open (const char *filename, ios::openmode) → ios::out | ios::trunc   (default)
void fstream::open(const char *filename, ios::openmode) → ios::in   |  ios::out     (default)

**<u>modes</u>**

ios::app   -     Append to the end of file
ios::ate    -     File offset is set to the end of the file, although I/O operation are allowed to occur anywhere in file.
ios::in      -     File capable of input
ios::out    -     File capable of output
ios::trunc -     If the named file already exist, the file is destroyed & is truncated to zero length.
ios::binary-     ios::binary value causes a file to be opened in binary mode. By default, all files are opened in text mode. In text mode the file store information  as character (like 439) will be stored as these characters '4','3','9' moreover various character translation may take place such as carriage return/ linefeed sequence being converted into newlines. However when a file is opened in binary mode, no such character translation will occur (also bytes in memory will be stored as it is in the file like if 257 = 0000000100000001 in memory will be stored as it is in the file.

**Note :: Any file, whether it contains formatted text or raw data, can be opened in either binary or text mode.**

```
ofstream out;
out.open ("test", ios::out); //out.open ("test"); as ios::out is default for ofstream.
```

To close → out.close()

**If open ( ) fails, the stream will evaluated to false when used in a Boolean expression.**

```
if (!mystream)
{
        cout<<"cannot open file\n"; //handle error
}
```

The ifstream, ofstream & fstream classes have constructor function that automatically open the file. The constructor function have the same parameters & defaults as the open ( ) function.

**Ifstream mystream ("myfile"); // open file for input**

> **As stated, if for some reason the file cannot be opened, the value of the associated stream variable will evaluated to false.**
>
> ***You can also check to see if you have successfully opened a file by using the is_open ( ) function, which is a member of fstream, ifstream ofstream.***

```
        bool is_open ( );
```

**it returns true if the stream is linked to an open file & false otherwise.**

```
If (! mystream.is_open () )
{
              cout<< “File is not open
\n”;
}
```

**To close a file – mystream.close();**
**The close() function takes no arguments and return no value**

## Reading and writing text files (Formatted I/O)

**Use << && >> as with other stream**

**Example-1**

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  ofstream out("INVNTRY"); // output,normal file
  if(!out) {
    cout << "Cannot open INVENTORY file.\n";
    return 1;
  }
  out << "Radios " << 39.95 << endl;
  out << "Toasters " << 19.95 << endl;
  out << "Mixers " << 24.80 << endl;
  out.close();
  return 0;
}
```

**Example-2**

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  ifstream in("INVNTRY"); // input
  if(!in) {
    cout << "Cannot open INVENTORY
file.\n";
    return 1;
  }
  char item[20];
  float cost;
  in >> item >>  cost;
  cout << item << " " << cost << "\n";
  in >> item >> cost;
  cout << item << " " << cost << "\n";
  in >> item >> cost;
  cout << item << " " << cost << "\n";
  in.close();
  return 0;
}
```

**•When reading text files using >> operator keeps in mind that certain character translation will occur. For example white-space characters are omitted. If you want to prevent any character translation, you must open a file for binary access & use the function discussed in the next section.**

**•When inputting, if end of file is encountered, the stream linked to that file will evaluate to false.**

*Unformatted (raw/ binary) reading/ writing of files using various function { the function can be applied to text files also but then some character translation may occur}*

**put () & get()**

```
                istream &get(char &ch);
                ostream &put(char &ch);
```

the get() function reads a single character from the invoking stream & put that value in ch. It returns a reference to the stream. The put() function writes ch to the stream & returns a reference to the stream.

*Example-1*

```
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc, char *argv[])
{
  char ch;
  if(argc!=2) {
    cout << "Usage: PR <filename>\n";
    return 1;
  }
  ifstream in(argv[1], ios::in | ios::binary);
  if(!in) {
    cout << "Cannot open file.";
    return 1;
  }
while(in) { /* in will be false when eof is
 reached, as get() returns  a reference to
the stream in & in will be false when the
end of file is encountered
    in.get(ch);
    if(in) cout << ch;
  }
  return 0;
}
```

*Example-2*

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  int i;
  ofstream out("CHARS", ios::out | ios::binary);
  if(!out) {
    cout << "Cannot open output file.\n";
    return 1;
  }
  // write all characters to disk
  for(i=0; i<256; i++)
              out.put((char) i);
  out.close();
  return 0;
}
```

## read() & write()

**to read block of binary data**

***istream & read(char *buf, streamsize num);***

***ostream &write(const char *buf, streamsize num);***

**the read() function reads num characters from the invoking stream & puts them in the buffer pointed to by buf. The write() function writes num characters to the invoking stream from the buffer pointed by buf. As mentioned in the preceeding chapter, streamsize is a type defined by the C++ libarary as some form of integer. It is capable of holding the largest no. of characters that can be transferred in any one I/O operation.**

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
struct status {
  char name[80];
  double balance;
  unsigned long account_num;
};
```

```
int main()
{
  struct status acc;
  strcpy(acc.name, "Ralph Trantor");
  acc.balance = 1123.23;
  acc.account_num = 34235678;
  // write data
 ofstream outbal("balance", ios::out | ios::binary);
  if(!outbal) {
    cout << "Cannot open file.\n";
    return 1;
  }
  outbal.write((char *) &acc, sizeof(struct status));
  outbal.close();
  // now, read back;
  ifstream inbal("balance", ios::in | ios::binary);
  if(!inbal) {
    cout << "Cannot open file.\n";
    return 1;  }
  inbal.read((char *) &acc, sizeof(struct status));
  cout << acc.name << endl;
  cout << "Account # " << acc.account_num;
  cout.precision(2);
  cout.setf(ios::fixed);
  cout << endl << "Balance: $" << acc.balance;
  inbal.close();
  return 0;}
```

- **As you can see, only a single call to read() or write() is necessary to read or write the entire structure. Each individual field need not be read or written separately & this example illustrates, the buffer can be any type of object.**

- **If the end of the file is reached before num characters have been read, then read() simply stops and the buffer contains as many characters as were available, you can find out how many characters have been read by using another member function called gcount().**

**streamsize gcount();**

***It returns the no. of characters read by the last binary input operation.***

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  double fnum[4] = {99.75, -34.4,
1776.0, 200.1};
```

```
int i;
  ofstream out("numbers", ios::out | ios::binary);
  if(!out) {
    cout << "Cannot open file.";
    return 1;
   }
  out.write((char *) &fnum, sizeof fnum);
  out.close();
  for(i=0; i<4; i++) // clear array
    fnum[i] = 0.0;
  ifstream in("numbers", ios::in |
ios::binary);
  in.read((char *) &fnum, sizeof fnum);
  // see how many bytes have been read
  cout << in.gcount() << " bytes read\n";
  for(i=0; i<4; i++) // show values read from file
  cout << fnum[i] << " ";
  in.close();
  return 0;
}
```

## *More get() function*

istream *get(char *buf, streamsize num); →1

istream *get(char *buf, streamsize num, char delim); →2

istream *get(); →3

1. read characters into array pointed by buf until num-1 characters have been read or newline is found or the end of the file has been encountered. The array pointed to by buf will be NULL terminated by get(). If the newline character is encountered in the input stream, it is not extracted, instead, it remains in the stream until the next input operation.

2. reads char until num -1 characters have been read or char specified by delim has been found or the end of the file has been encountered. buf will be NULL terminated. If the delim char is encountered in the input stream, it is not extracted.

3. get() returns the next char, from the stream. It returns EOF if the end of the file is encountered..

## <u>getline() // </u>can be very useful with textfiles

istream &getline(char *buf, streamsize num);

similar to 1 but the new line character is extracted from the stream but is not put into buf (i.e. it is basically removed)

istream &getline (char *buf, streamsize num, char delim)

similar to 2 but the delim char is extracted from the stream but is not put into buf (i.e. it is basically removed)

```
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc, char *argv[])
{
  if(argc!=2) {
    cout << "Usage: Display <filename>\n";
    return 1;
  }
  ifstream in(argv[1]); // input
  if(!in) {
    cout << "Cannot open input file.\n";
    return 1;
  }
  char str[255];
  while(in) {
    in.getline(str, 255);  // delim defaults to '\n'
    if(in) cout << str << endl;
  }
  in.close();
  return 0;
}
```

Detecting EOF

You can detect when the end of the file is reached by using the member function eof(). Which has this prototype.

**bool eof();**

**it returns true when the end of the file has been reaches; otherwise it returns false**

**<u>The following programs uses eof() to display the content of a file in both hexadecimal & ASCII</u>**

**(skip)**

```
#include <iostream>
#include <fstream>
#include <cctype>
#include <iomanip>
using namespace std;
int main(int argc, char *argv[])
{
  if(argc!=2) {
    cout << "Usage: Display <filename>\n";
    return 1;
  }
  ifstream in(argv[1], ios::in | ios::binary);
  if(!in) {
    cout << "Cannot open input file.\n";
    return 1;
  }
  register int i, j;
  int count = 0;
  char c[16];
  cout.setf(ios::uppercase);
  while(!in.eof()) {
    for(i=0; i<16 && !in.eof(); i++) {
      in.get(c[i]);
    }
if(i<16) i--; // get rid of eof
    for(j=0; j<i; j++)
      cout << setw(3) << hex << (int) c[j];
    for(; j<16; j++) cout << "  ";
    cout << "\t";
    for(j=0; j<i; j++)
      if(isprint(c[j])) cout << c[j];
      else cout << ".";
    cout << endl;
    count++;
    if(count==16) {
      count = 0;
      cout << "Press ENTER to continue: ";
      cin.get();
      cout << endl;
    }
  }
  in.close();
  return 0;
}
```

## ignore () function

**istream & ignore(streamsize num=1, int_type delim = EOF);**
**it reads & discards characters until either num characters have been ignored (1 by default) or the character specified by the delim is encountered (EOF by default). If the delimiting characters is encountered , it is not removed from the input stream. Here int_type is some form of integer**

The next program reads a file called TEST. It ignores characters until either a space is encountered or 10 characters have bee read. It then displays the rest of the file.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  ifstream in("test");
  if(!in) {
    cout << "Cannot open file.\n";
    return 1;
  }
  /* Ignore up to 10 characters or until first
     space is found. */
  in.ignore(10, ' ');
  char c;
  while(in) {
    in.get(c);
    if(in) cout << c;
  }
  in.close();
  return 0;}
```

**int_type peek();**

// it returns the next characters in the stream without removing the characters from the stream or EOF if the end of the file is encountered

**istream &putback(char c);**

// returns the last char read from a stream here c is the last char read

**flush()** → when output is performed, data is not necessary immediately written to the physical device linked to the stream. Instead, information is stored in an internal buffer until the buffer is full. Only then are the content of that buffer is written to disk. However you can force the information to be physically written to disk before the buffer is full by calling flush().

**ostream &flush();**

closing a file or terminating a program also flushes all buffers.

## Random Access

The C++ I/O system manages two pointers associated with a file. one is the get pointer, which specifies where is the file's the next input/read operation will occur. The other is the put pointer, which specifies where in the file, next output operation will occur. Each time an input or output operation takes place, the appropriate pointer is automatically sequentially advanced. However, using the seekg() & seekp() function allow you to access the file in a non sequential fashion.

*iostream &seekg (off_type offset, seekdir origin);*

| | | |
|---|---|---|
| *ios::beg* | - | *beginning of file* |
| *ios::cur* | - | *current location* |
| *ios::end* | - | *end of file* |

seekp()/ seekg() – moves the associated file's current put/ get pointer offset no. of character from the specified origin.

*istream &seekp(off_type offset, seekdir origin)*

note:- generally random access should only be performed on binary files as the character translation that may occur on text files could cause a position request to be out of sync with the actual contents of the file ex.

change test 12 z

here test is file to be modify

and 12 is the 12th char in the file test should be changed to z

z is new char which will replace the already existing char in file.

```
listing 19// shows the use of seekp()
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
int main(int argc, char *argv[])
{
  if(argc!=4) {
    cout << "Usage: CHANGE <filename>
<character> <char>\n";
    return 1;
  }
  fstream out(argv[1], ios::in | ios::out | ios::binary);
  if(!out) {
    cout << "Cannot open file.";
    return 1;  }
  out.seekp(atoi(argv[2]), ios::beg);
  out.put(*argv[3]);
  out.close();
  return 0;}
```

**next program display the contents of a file beginning with the location you specify on the command line.(use of seekg()) (skip)**

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
int main(int argc, char *argv[])
{
  char ch;
  if(argc!=3) {
    cout << "Usage: SHOW <filename> <starting
location>\n";
    return 1;
  }
  ifstream in(argv[1], ios::in | ios::binary);
  if(!in) {
    cout << "Cannot open file.";
    return 1;
  }
  in.seekg(atoi(argv[2]), ios::beg);
  while(in.get(ch))
    cout << ch;
  return 0;
}
```

**The following program uses both seekp() & seekg() to reverse the first <num> character in a file. (skip)**

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
int main(int argc, char *argv[])
{
  if(argc!=3) {
    cout << "Usage: Reverse <filename> <num>\n";
    return 1;  }
  fstream inout(argv[1], ios::in | ios::out | ios::binary);
  if(!inout) {
    cout << "Cannot open input file.\n";    return 1;  }
  long e, i, j;
  char c1, c2;
  e = atol(argv[2]);
  for(i=0, j=e; i<j; i++, j--) {
    inout.seekg(i, ios::beg);
    inout.get(c1);
    inout.seekg(j, ios::beg);
    inout.get(c2);
    inout.seekp(i, ios::beg);
    inout.put(c2);
    inout.seekp(j, ios::beg);
    inout.put(c1);  }
  inout.close();
  return 0;}
```

```
> reverse test 10

   concerned file
O/P         this is a test
          a si sithtest
```

**Reverese the first 10 char (of 14 test file)**

## Obtaining the current file position

**You can determine the current position of each file pointer by using these function.**

**pos-type     tellg();**
**pos-type     tellp();**

**-some sort of int. defined in ios  and is capable of holding the largest value that either function can return.**

**-You can use the values returnd by tellg() & tellp() as arguments to the seekg() & seekp().**

**-  These function allow you to save the current file location, perform other file operation, and then reset the file location to its previously saved location.**

# I/O Status

**The C++ I/O system maintains status information about the out come of each I/O operation. The current state of the I/O system is held in an object of type iostate, which is an enumeration defined by ios that include the following members:**

**ios:: goodbit**
**ios:: eofbit**
**ios:: failbit**
**ios:: badbit**
**ios:: hardfailbit**

**there are two ways in which you can obtain I/O status information**

**(1)- rdstate() -> iostate rdstate();**
**returns goodbit when no error has occurred otherwise, an error flag is turned on**

the program illustrates rdstate(). It displays the contents of a text file. If an error occurs, the program reports it, using checkstatus()

```
#include <iostream>
#include <fstream>
using namespace std;
void checkstatus(ifstream &in);
int main(int argc, char *argv[])
{
  if(argc!=2) {
    cout << "Usage: Display <filename>\n";
    return 1;
  }
  ifstream in(argv[1]);
  if(!in) {
    cout << "Cannot open input file.\n";
    return 1;
  }
  char c;
  while(in.get(c)) {
    if(in) cout << c;
    checkstatus(in);
  }
  checkstatus(in);  // check final status
  in.close();
  return 0;
}
```

```
void checkstatus(ifstream &in)
{
  ios::iostate i;
  i = in.rdstate();
  if(i & ios::eofbit)
    cout << "EOF encountered\n";
  else if(i & ios::failbit)
    cout << "Non-Fatal I/O error\n";
  else if(i & ios::badbit)
    cout << "Fatal I/O error\n";
}
```

(2) **The other way that you can determine if an error has occurred is by using one or more of these functions .**

**bool bad(); // returns true if badbit or hardfail flag is set**

**bool eof(); // returns true if eofbit flag is set**

**bool fail(); // returns true if failbit or badbit or hardfail flag is set**

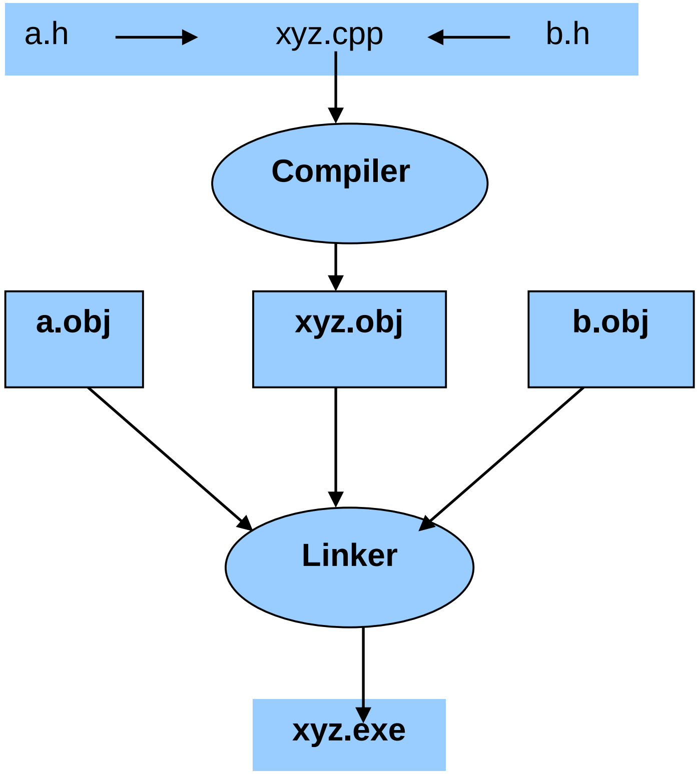
**bool good(); // returns true if no error bit is set and no flag is set**

**clear(int = 0) → with no argument, clears all error bits;**
**otherwise sets specified flags, as in**
**clear (ios::failbit) → failbit will be set**

**once an error has occurred, it may need to be cleared before your program continue. To do this, use the clear() function, which has this prototype.**
void clear(iostate flags = ios::goodbit);

if flags is goodbit (as it is by default), all error flags are cleared otherwise, set flags as you desire.

## Multifile program



(1)
To open a file for read as well as write use File.open (“EMP.DAT”, ios::binary |ios::in |ios::out);

(2)
once a file reaches EOF while reading then even if you take the read pointer to the first char of the file using seekg() → even than you will not be able to read the file as the eof flag whould have been set → to overcome this problem you will need to clear the flag using ios::clear() member function

(3)
**streambuf class**

Each stream object contains a pointer to a streambuf object.This pointer can be accessed through a member function called rdbuf(). Every stream object has this function. It returns a pointer to the streambuf object.

One of the most interesting things that we can do with this pointer is to connect it to another stream object using the << operator. This would move all the bytes from the buffer of one object into that of another. We can use this facility for file copying

```
ifstream infile(sourcefile);
ofstream outfile(targetfile);
outfile << infile.rdbuf();
```

this causes the entire contents of the source file to be sent to the outfile/targetfile

(4)

strstreams class

**strstreams class is used to treat memory array to be used as stream and allow all the formatting function as is available with other streams.**

**If you want to extract character use istrstream**

**If you want to insert characters into stream use ostrstream.**

```
#include <strstream.h>
#include <iomanip.h>
int main()
{
const int MAX = 100;
int i = 350;
char ch = ‘Z’
float a = 3.14152869;
char str[] = “strstreams at work”;
char buffer[MAX];
ostrstream s(buffer, MAX);
/* buffer is assumed to be a zero terminated string, and any new characters are added starting at the zero terminator */
s <<endl
    << setw(8) <<”ch = “ << ch << endl
    << setw(8) << “i = “ << hex << i << endl
 << setw(8) <<”a = “
<<setiosflags(ios::fixed) << a <<endl
  << setw(8) <<”str = “ <<str
   << ends;
// An important thing to remember about ostrstreams is that zero terminator we normally need at the end of a character array is not inserted for us. We need to specifically insert it using the manipulator ends.

cout << s.rdbuf();

// when we rdbuf the contents of the buffer the get pointer inside the streambuf is moved forward as the characters are output. For this reason, if we say cout << s.rdbuf()  a second time , nothing will happen because the get pointer is already at the end.
}
```

## USING istrstreams

```
int main()
{
int age;
float salary;
char name[50];
char str[] = “35 12004.50 sammer shekhar deshpande”;
istrstream s (str);
```

**// the constructor takes zero terminated char array or pointer to zero terminated string allocated on the heap**

**//once the istrstream object is built we can now extract bytes from it until the ‘\0’ is encountered**

**// (skip) there is one more form of istrstream constructor available, this takes two argument → pointer and size of array**
**In this case array does’t have to be zero terminated and from such istrstream object you can extract bytes all the way to buff[size], whether or not you encounter a zero along the way.**

```
s >> age >> salary >> name;
// extract age salary and name from istrstream s
```

**// but while extracting the name only first component of the name is extracted because of whitespace. The balance (left over) need to be extracted using rdbuf()**

```
cout << age << endl
        << salary << endl
         << name;
count << endl << s.rdbuf();
}
```

→<u>automatic storage allocation</u> **(skip)**
while using output strstreams (but not istrstreams) we have an option of allowing the ostrstream to do its own memory allocation. For this we need to create ostrstream object with no constructor argument ie → ostrstream s;
details if needed do from pg. 460 let us c++

(5)

**predefined filenames for hardware devices**

**con → console (keyboard and screen)**
**aux or com1 → first serial port**
**com2 → second serial port**
**prn or lpt1 → first parallel port/ printer**
**lpt2 → second parallel port/printer**
**lpt3 → third parallel printer**
**nul → dummy (non existent ) device**

**The following program prints the contents of a disk file on the printer. After printing the entire file contents a ‘\x0c’ is sent to the printer to eject the paper**

```
Ofstream outfile(“PRN”);
While (infile.get(ch) !=0)  // ifstream infile(filename)
Outfile.put(ch)
Outfile.put(‘\x0C’);
```

Q:  Design a vector class

- vector (int,vsize)
- with overloaded multiplication & addition operator
- two vector can be added or multiplied
- different type of costructors

Q: Design a string class

- sining [char *p, int item]
- allocate memory dynamically
- create string from array of char
- copy costructor
- + -> concatinate
- overloaded = operator