Managing Input Output Console in C++

Contents

- Managing I/O console
- C++ Stream Classes
- Formatted and Unformatted Console I/O
- Usage of Manipulators

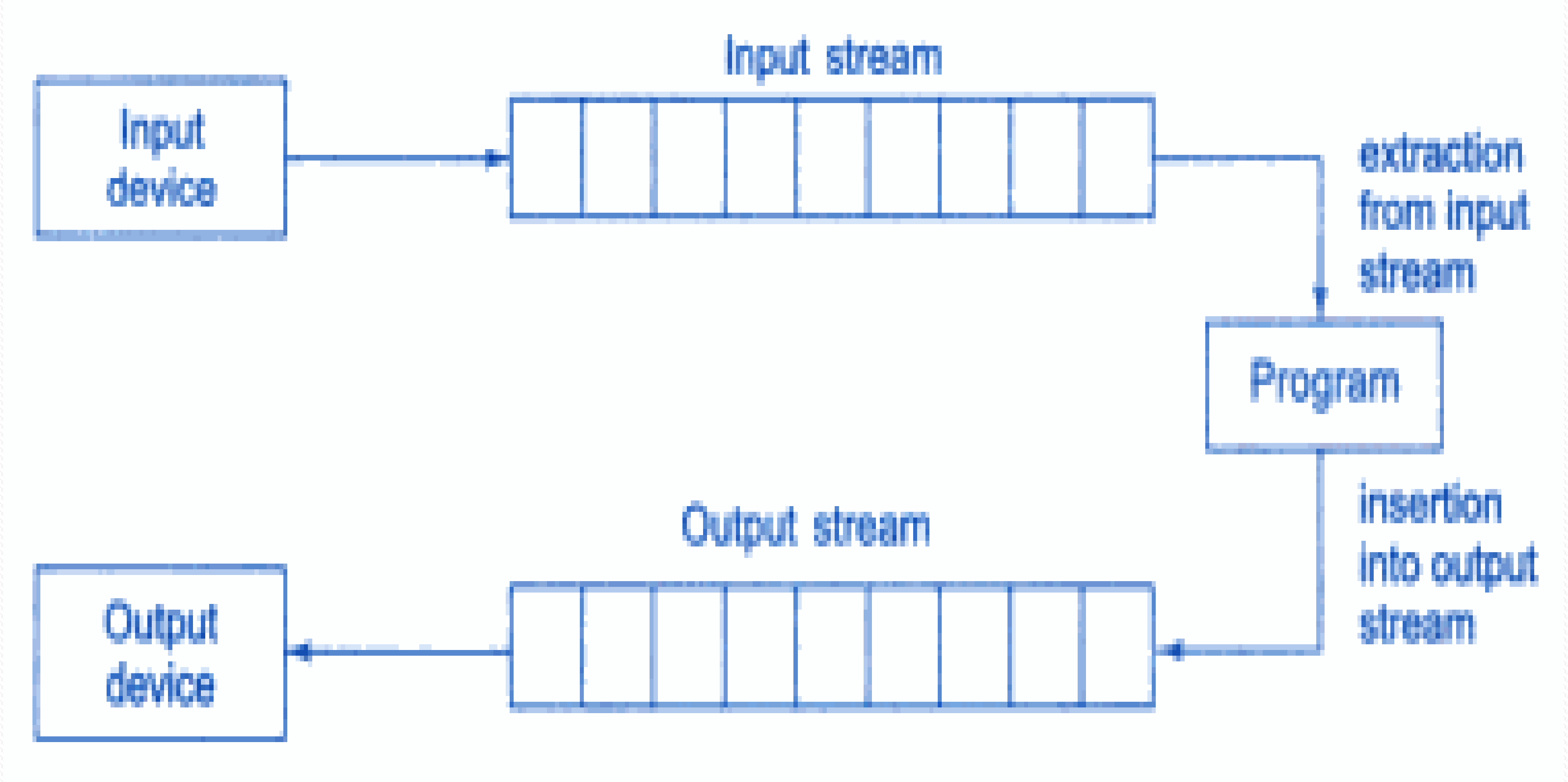
Managing I/O console

- C++ supports a rich set of I/O functions and operations.
- It uses the concept of stream and stream classes to implement its I/O operations with the console and disk files.

C++ Streams

- The I/O system supplies an interface to the programmer that is independent of the actual device being accessed.
- This interface is known as **stream**.
- A stream is a sequence of bytes.
- The source stream that provides data to the program is called the **input stream**.
- The destination stream that receives output from the program is called the **output stream**.

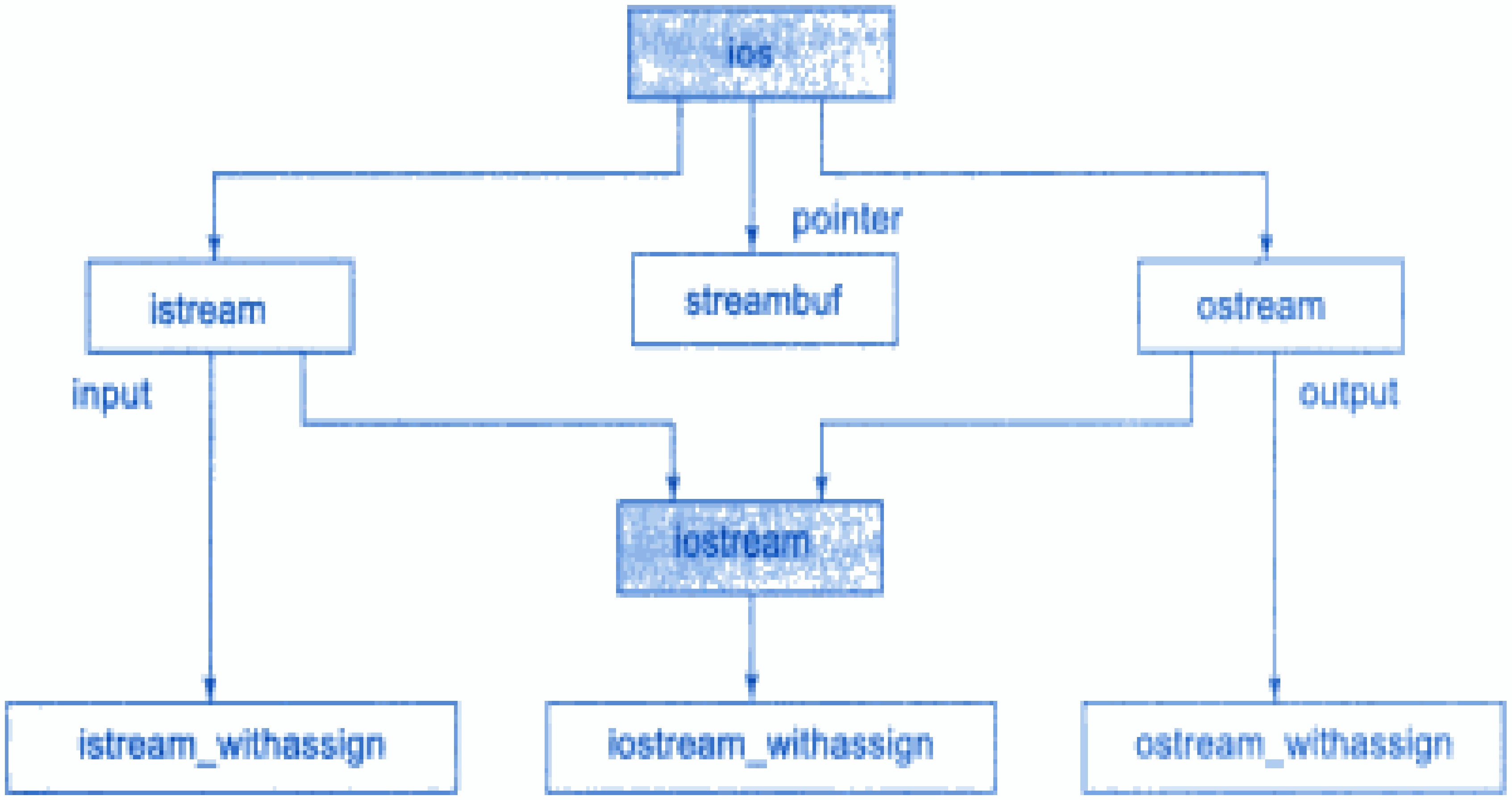
C++ Streams


Input stream
Input device
extraction from input stream
Program
insertion into output stream
Output stream
Output device

C++ Stream Classes

- The C++ I/O system contains a hierarchy of classes that are used to define various streams to deal with both the console and disk files.
- These classes are called stream classes.
- These classes are declared in the header file iostream.

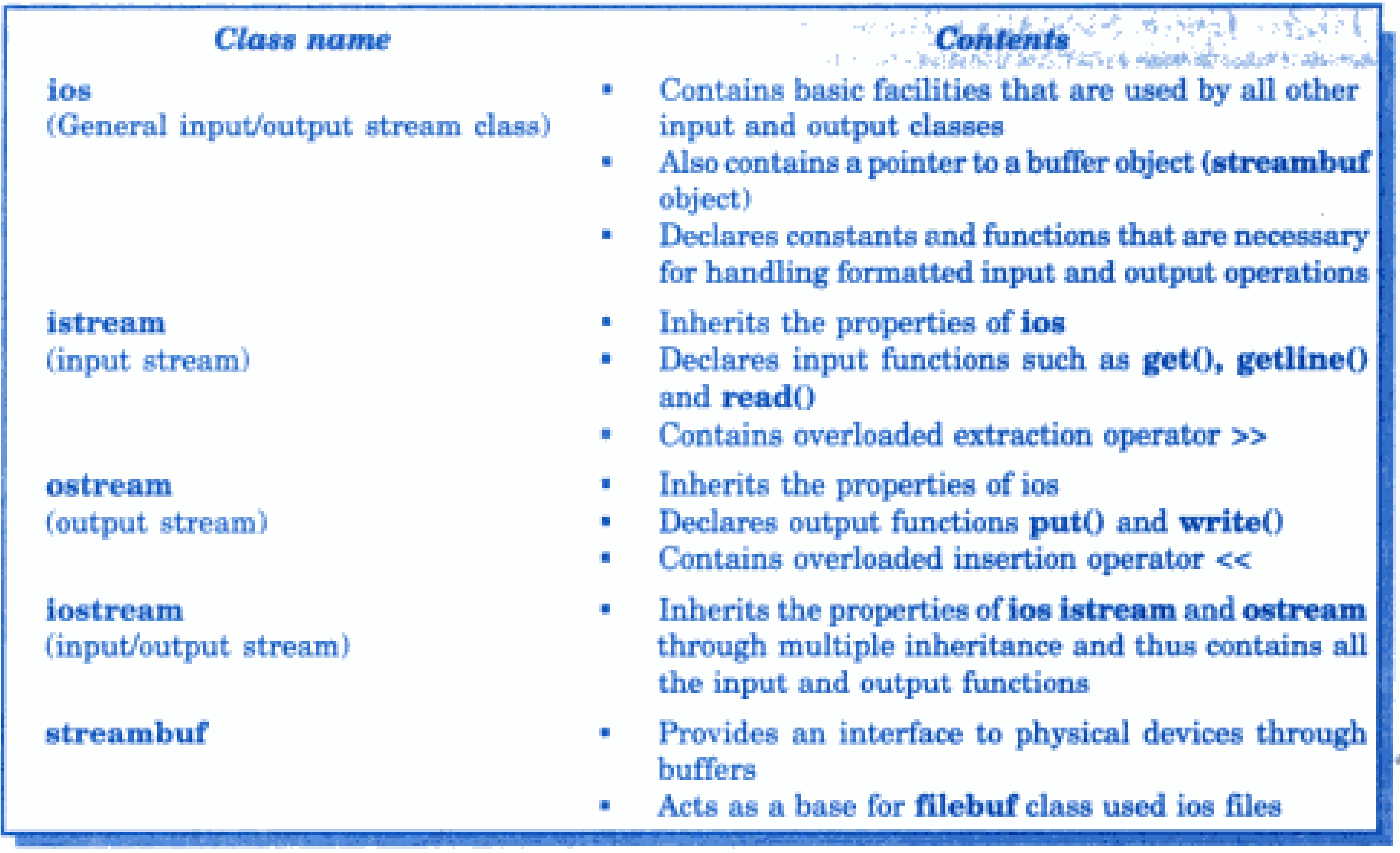
C++ Stream Classes



C++ Stream Classes

- The class ios provides the basic support for formatted and unformatted I/O operations.
- The class istream provides the facilities for formatted and unformatted input while the class ostream provides the facilities for formatted output.
- The class iostream provides the facilities for handling both input and output streams.
- Three classes istream_withassign, ostream_withassign and iostream_withassign add assignment opreators to these classes.

Stream classes for console operations



Class name	*Contents*
ios (General input/output stream class)	▪ Contains basic facilities that are used by all other input and output classes ▪ Also contains a pointer to a buffer object (**streambuf** object) ▪ Declares constants and functions that are necessary for handling formatted input and output operations
istream (input stream)	▪ Inherits the properties of **ios** ▪ Declares input functions such as **get()**, **getline()** and **read()** ▪ Contains overloaded extraction operator >>
ostream (output stream)	▪ Inherits the properties of ios ▪ Declares output functions **put()** and **write()** ▪ Contains overloaded insertion operator <<
iostream (input/output stream)	▪ Inherits the properties of **ios istream** and **ostream** through multiple inheritance and thus contains all the input and output functions
streambuf	▪ Provides an interface to physical devices through buffers ▪ Acts as a base for **filebuf** class used ios files

Unformatted I/O Operations

- Overloaded Operators >> and <<
- put() and get() functions
- getline() and write() functions

Overloaded Operators >> and <<

- The objects cin and cout are used for the input and output of data of various types by overloading >> and << operators.
- The >> operator is overloaded in the istream class and << is overloaded in the ostream class.
- The general format for reading data from keyboard is:

  cin >> variable1 >> variable2 >> ……. >> variableN

- The input data are separated by white spaces and should match the type of variable in the cin list.

Overloaded Operators >> and <<

- The operator reads the data character by character and assigns it to the indicated location.
- The reading for a variable will be terminated at the encounter of a whitespace or a character that does not match the destination type.
- Eg:

```
int code;
cin >> 4258D
```

- The operator will read the characters upto 8 and the value 4258 is assigned to code.

Overloaded Operators >> and <<

- The general form for displaying data on screen is:

  cout << item1 << item2 << ….. << itemN

- The items item1 through itemN may be the variables or constants of any basic type.

put () and get () Functions

- The classes istream and ostream define two member functions get() and put() to handle the single character input and output operations.
- There are two types of get() functions : get (char *) and get (void).
- **get(char *)** version assigns the input character to its argument.
- **get(void)** version returns the input character.

put () and get () Functions

- The function put() is used to output a line of text character by character.

  cout.put('x');   -   D

  cout.put(ch);   -   Di alue of variable ch.

What happen if this statement is executed ?

- The variable ch must co n a character value.

  cout.put(68);

The statement will convert 68 to char value and display character D.

getline() and write () Functions

- The getline() function reads a whole line of text that ends with a newline character.
- This function can be invoked by using the object cin.

```
cin.getline (line, size);
```

- The function getline() which reads character input into the variable line.
- The reading is terminated as soon as either the newline character is read or size-1 characters are read.

getline() and write () Functions

- The write() function displays an entire line.

  cout.write(line, size);

- The first argument line represents the name of the string to be displayed and the second argument size indicates the number of characters to display.

Formatted I/O Operations

- C++ supports a number of features that could be used for formatting the output.
  - ios class functions and flags
  - Manipulators
  - User-defined output functions.

ios class functions and flags

Functions	Task
Width()	To specify the required field size for displaying an output value.
Precision()	To specify the number of digits to be displayed after the decimal point of a float value.
Fill()	To specify a character that is used to fill the unused portion of a field.
Setf()	To specify format flags that can control the form of output display.
Unsetf()	To clear the flags specified

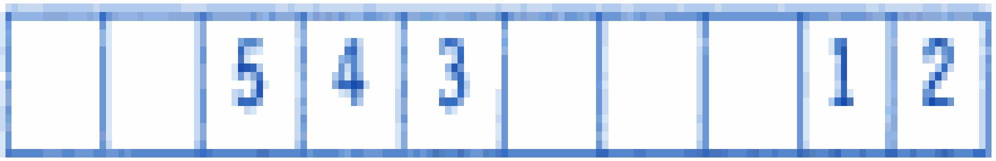
Defining Field Width: width()

- The width() function to define the width of a field necessary for the output of an item.

  cout.width(w);

- Where w is the field width. The output will be printed in a field of w characters wide at the right end of the field.
- Eg:

```
cout.width(5);
  cout<< 543;
cout.width(5);
cout << 12;
```



Setting Precision: precision()

- We can specify the number of digits to be displayed after the decimal point while printing the floating point numbers.

  cout.precision(d);

- Where d is the number of digits to the right of the decimal point.
- Eg: cout.precision(3);

```
 cout<< sqrt(2) << “\n”;          -  1.141
cout << 3.14159;                  -  3.142
```

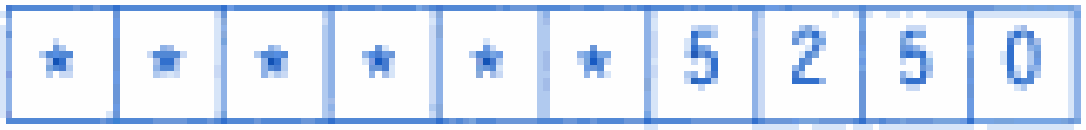
Filling and Padding: fill()

- The unused positions of the field are filled with white spaces.
- However, the fill() function can be used to fill the unused positions by any desired character.

cout.fill(ch);

- Where ch represents the character which is used for filling the unused positions.
- Eg:

```
cout.fill('*');
cout.width(10);
cout << 5250 ;
```



*	*	*	*	*	*	5	2	5	0

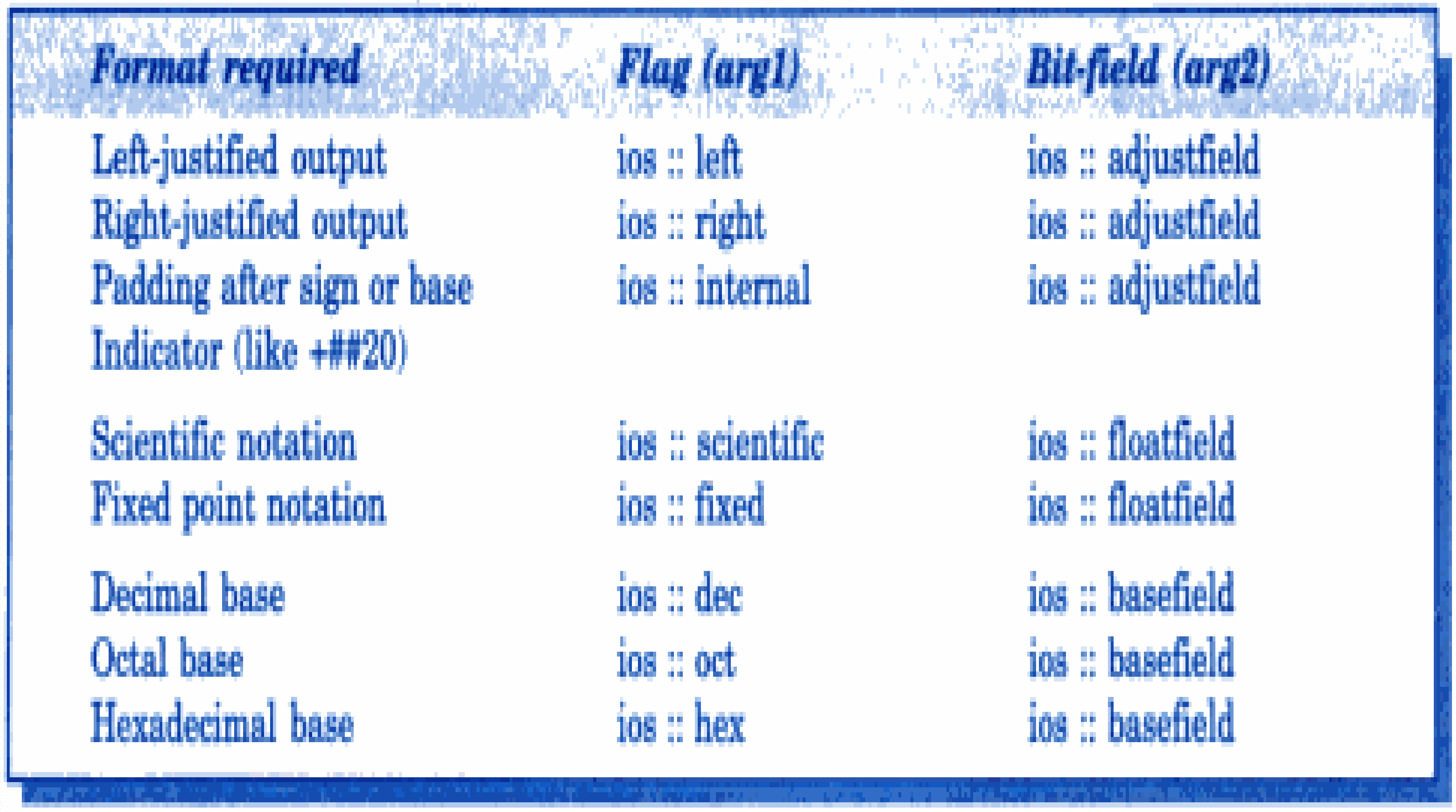
Formatting Flags, Bit-fields and setf()

- The setf() member function of the ios class is used for various types of formatting.
- Syntax:

  cout.setf(arg1, arg2)

- The arg1 is one of the formatting flags, specifying the action required for the output.
- The arg2 known as bit field specifies the group to which the formatting flag belongs.
- There are three bit fields and each has a group of format flags.

Formatting Flags, Bit-fields and setf()



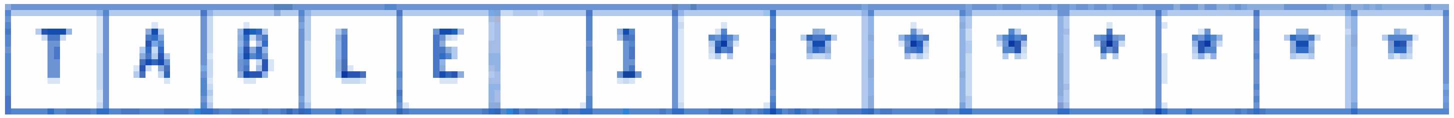
Format required	*Flag (arg1)*	*Bit-field (arg2)*
Left-justified output	ios :: left	ios :: adjustfield
Right-justified output	ios :: right	ios :: adjustfield
Padding after sign or base	ios :: internal	ios :: adjustfield
Indicator (like +##20)		
Scientific notation	ios :: scientific	ios :: floatfield
Fixed point notation	ios :: fixed	ios :: floatfield
Decimal base	ios :: dec	ios :: basefield
Octal base	ios :: oct	ios :: basefield
Hexadecimal base	ios :: hex	ios :: basefield

Formatting Flags, Bit-fields and setf()

- Consider the following segment of code:

```
cout.fill(‘*’);
cout.setf(ios::left, ios::adjustfield);
cout.width(15);
cout<< “TABLE  1” << “\n”;
```

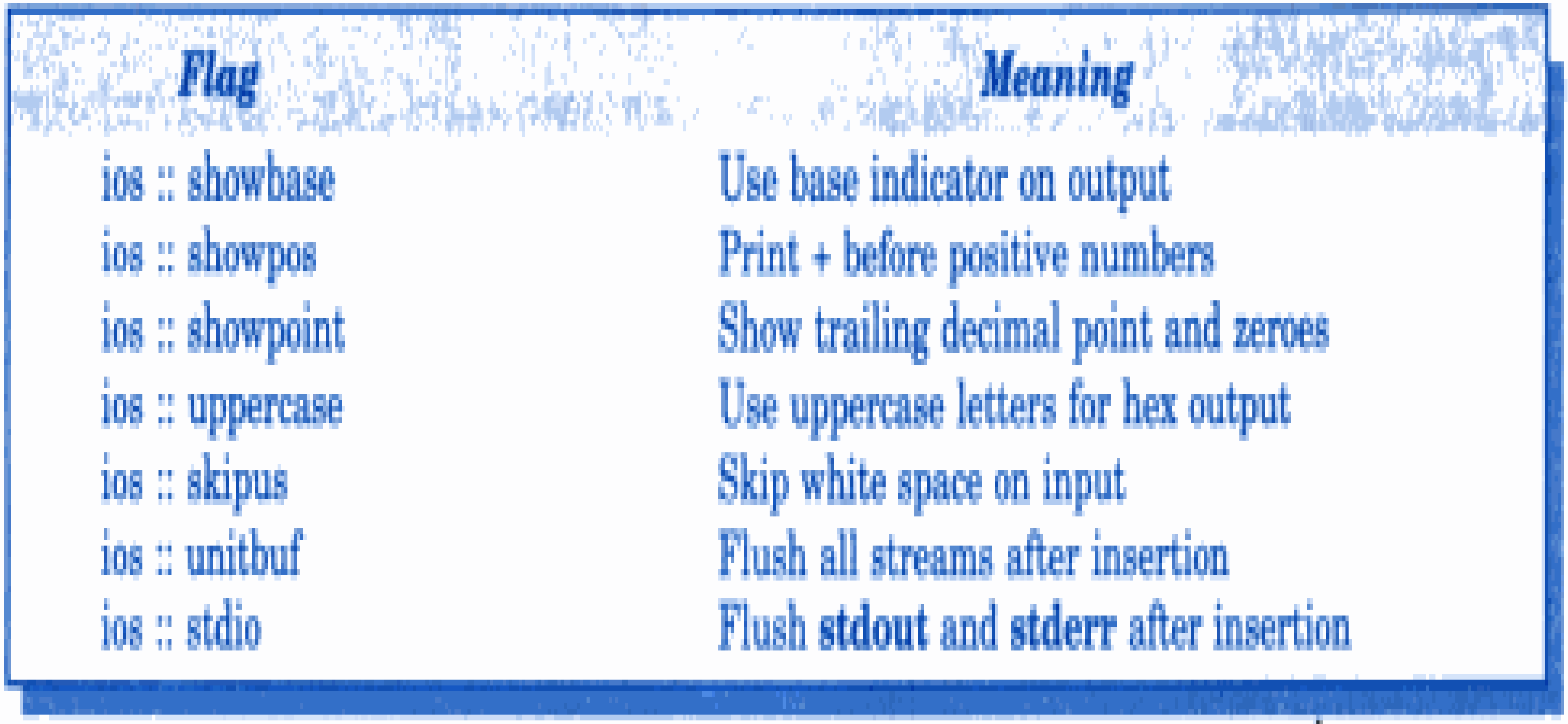
- Will produce the following output:



```
TABLE 1********
```

Displaying trailing zeros and Plus sign

- The setf() can be used with a single argument for achieving various format of output.
- Their are some flags that do not have bit fields.



Flag	*Meaning*
ios :: showbase	Use base indicator on output
ios :: showpos	Print + before positive numbers
ios :: showpoint	Show trailing decimal point and zeroes
ios :: uppercase	Use uppercase letters for hex output
ios :: skipus	Skip white space on input
ios :: unitbuf	Flush all streams after insertion
ios :: stdio	Flush **stdout** and **stderr** after insertion

Managing Output with Manipulators

- The header file iomanip provides a set of functions called manipulators which can be used to manipulate the output formats.
- They provide the same features as that if the ios member functions and flags.
- Manipulators can be used as a chain in one statement as:

```
cout << manip1 << manip2 << mainp3 << item;
cout << mainp1 << item1 << manip2 << item2;
```

Manipulators and their meaning

Manipulator	Meaning	Equivalent
setw (int w)	Set the field width to w.	width()
setprecision (int d)	Set the floating point precision to d.	precision()
setfill (int c)	Set the fill character to c.	fill()
setiosflags (long f)	Set the format flag f.	setf()
resetiosflags (long f)	Clear the flag specified by f.	unsetf()
endl	Insert new line and flush stream.	“\n”

Managing Output with Manipulators

- Examples:
- cout << setw(10) << 12345;
  - Prints the value 12345 right justified in a field width 10.
- cout << setw(10) <<setprecision(4) << sqrt(2);
  - Prints the value of sqrt(2) with 4 decimal places in the field width 10.
- cout << endl;
  - Inserts a new line.

User-defined output functions

- The programmer can also define his own manipulator according to the requirement of the program.
- Syntax:

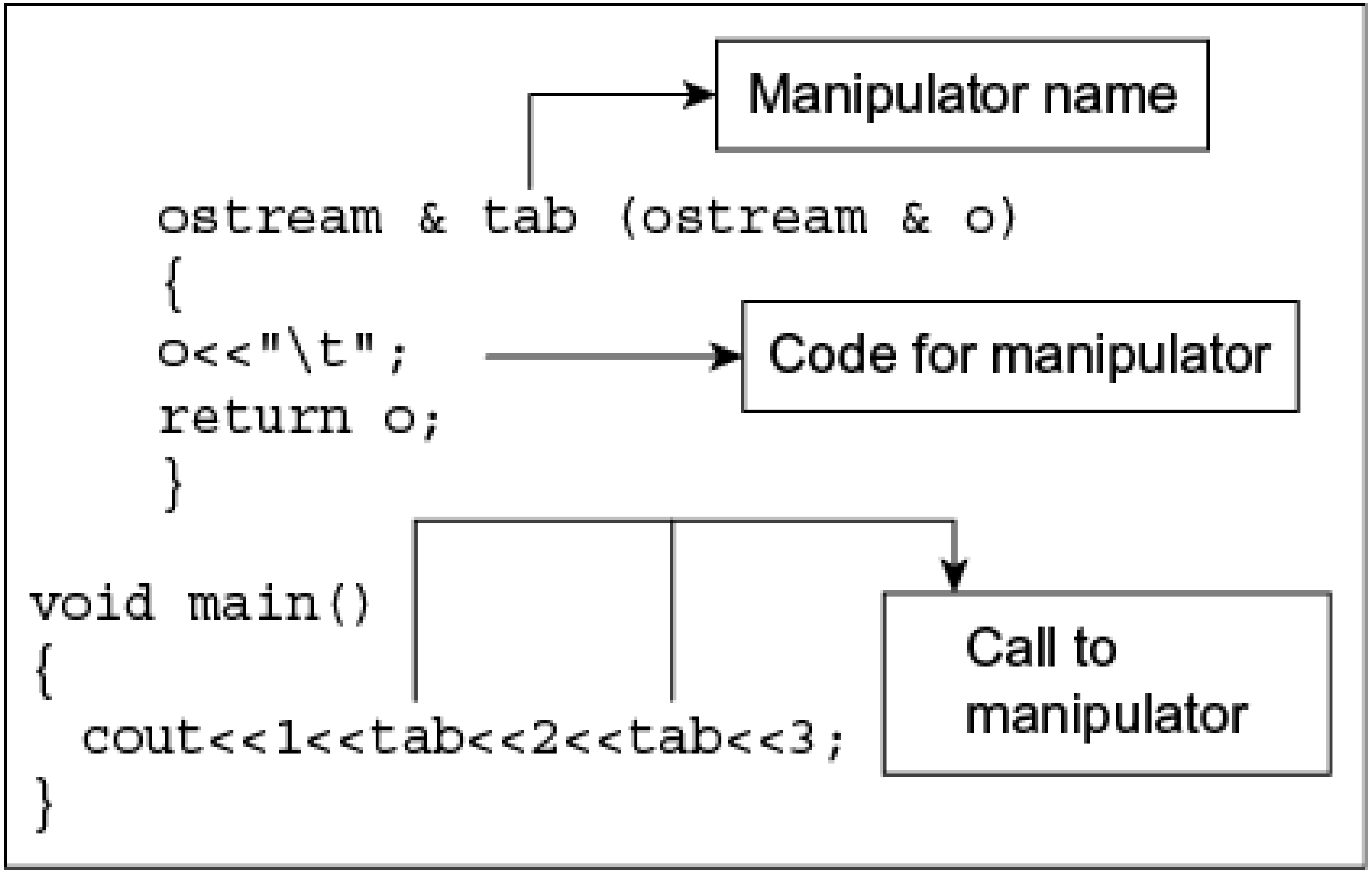
```
ostream & m_name (ostream & o)
{
        statement 1;
        statement 2;
        return o;
}
```

- The m_name is the name of the manipulator.

User-defined output functions

```
ostream & tab (ostream & o)
{
        o << “\t”;
        return o;
}
void main()
{
        clrscr();
        cout <<  1  <<  tab  <<  2  << tab  <<  3 ;
}
```

Working of tab manipulator



```
    ostream & tab (ostream & o)
    {
    o<<"\t";
    return o;
    }

void main()
{
  cout<<1<<tab<<2<<tab<<3;
}
```

Summary

- ______ is a sequence of bytes and serves as a source or destination for an I/O data.
- The source stream that provides data to the program is called _____ stream and the destination stream that receives output from the program is called ________ stream.
- The istream and ostream classes define two member functions _____ and _____ to handle the single character I/O operations.
- The >> operator is overloaded in the _____ class and an extraction operator << is overloaded in the ____ class.
- The functions width(), precision(), fill(), setf() for formatting the output are present in ____ class.
- _____ provides a set of manipulators functions to manipulate output formats.

Short Answer Questions

- Discuss the various forms of get() functions supported by the input stream. How are they used?
  - There are two types of get() functions : get (char *) and get (void).
  - **get(char *)** version assigns the input character to its argument.
  - **get(void)** version returns the input character.

Short Answer Questions

- How do the following two statements differ in operation?

```
cin >> c;
cin.get(c);
```

  - The first statement using the overloaded >> operator will skip the white spaces and newline character.
  - The second statement will fetch a character including the blank space, tab and newline character.

Short Answer Questions

- What does the following statement do?

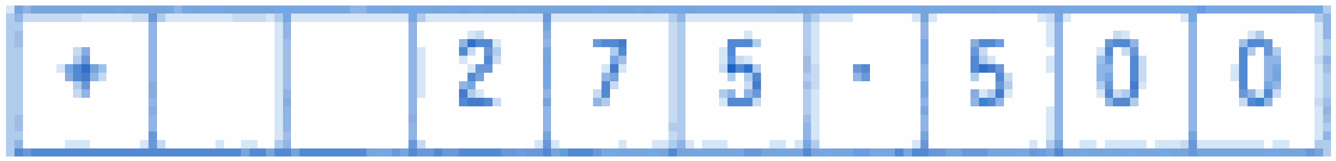
  cout.write(s1,m).write(s2,n);

  - The above statement is used to concatenate two strings using the write() function.
- What is the difference between put() and write() ?
  - The put() is used to output a line of test character by character.
  - The write() is used to display an entire line.

Short Answer Questions

- What will be the output of following statements:

```
cout.setf(ios :: showpoint);
cout.setf(ios :: showpos);
cout.precision(3);
cout.setf(ios :: fixed, ios :: floatfield);
cout.setf(ios :: internal, ios :: adjustfield);
cout.width(10);
cout << 275.5 << “\n”;
```



+			2	7	5	.	5	0	0

Short Answer Questions

- What is the basic difference between manipulators and ios member functions in implementation? Give examples.
    - Manipulators are more convenient to use than compare to ios member functions.
    - The manipulators cab ne used as a chain in one statement as:

```
cout << manip1 << manip2 << manip3 << item;
```

References

- Object Oriented Programming with C++ by E. Balagurusamy.

End of unit

Contents

- File operations : Text files , Binary Files
- File stream class and methods
- File updation with random access
- Overloading insertion and extraction operator

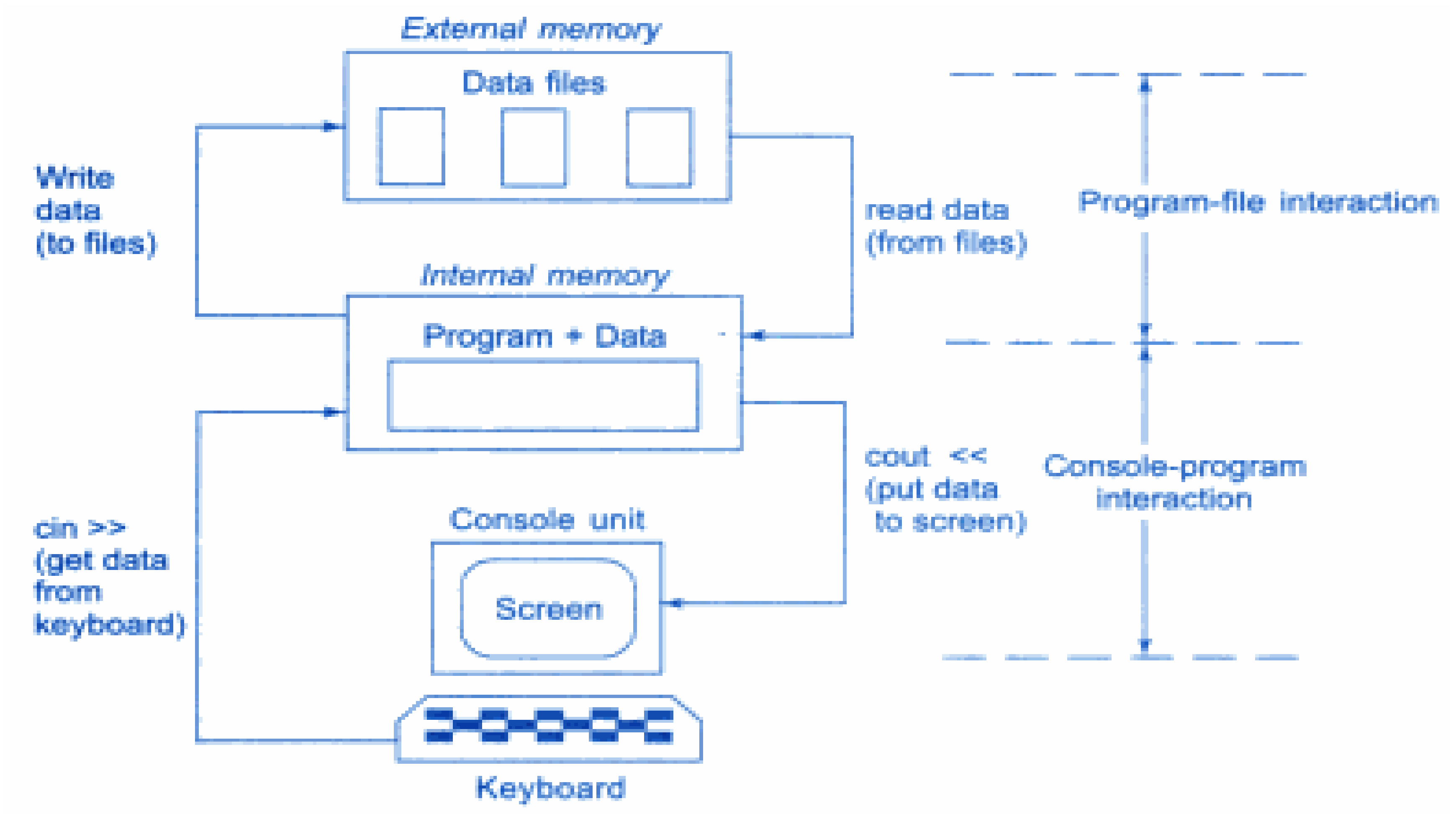
Quiz 1

- **What is a file?**
  - A **computer file** is a computer resource for recording data discretely in a computer storage device.

Introduction

- Many real-life problems handle large volumes of data.
- The data is stored in the devices using the concept of files.
- A file is a collection of related data stored in a particular area on the disk.
- Programs are designed to perform read and write operations on these files.

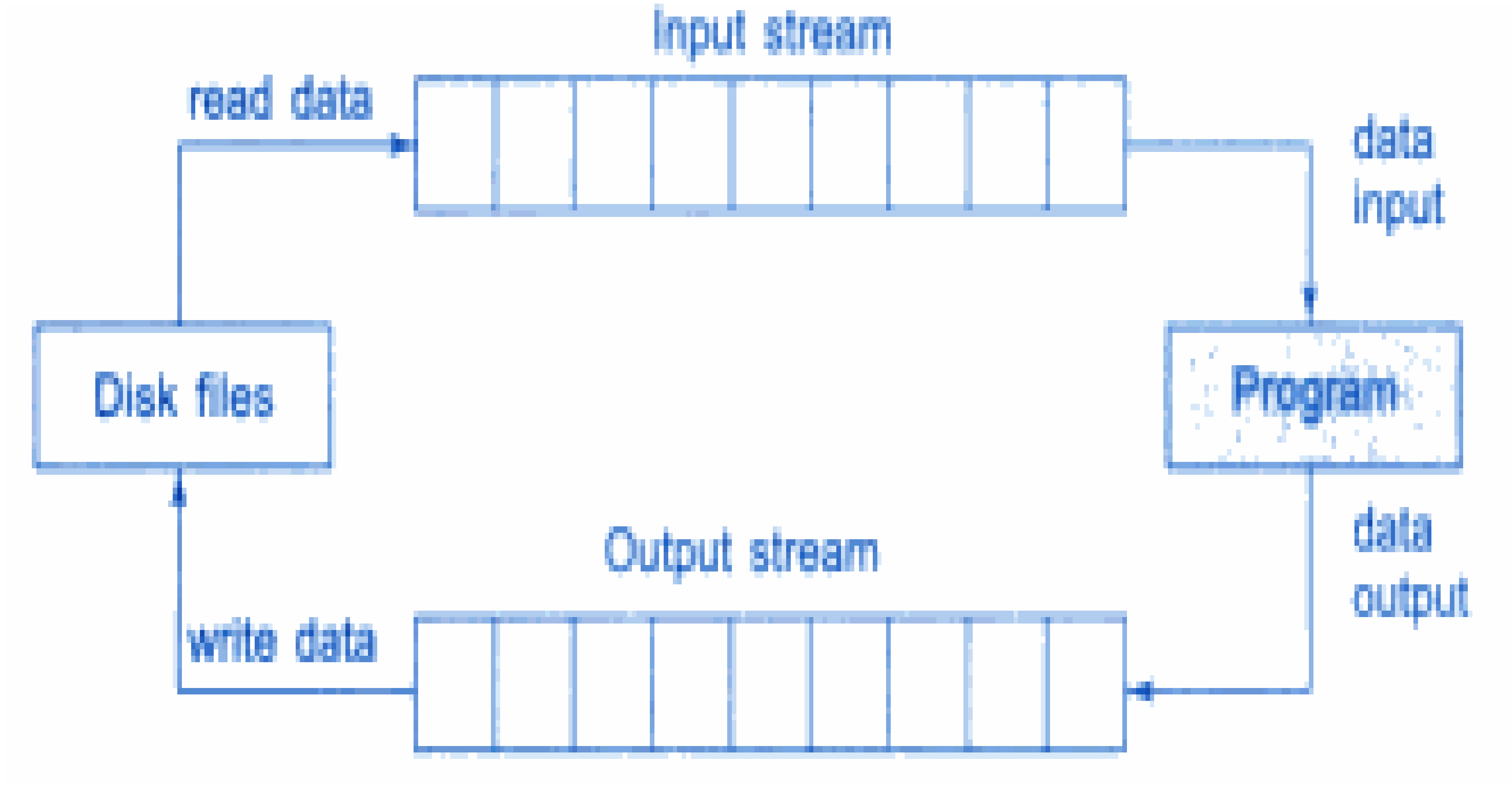
Console-Program-File interaction



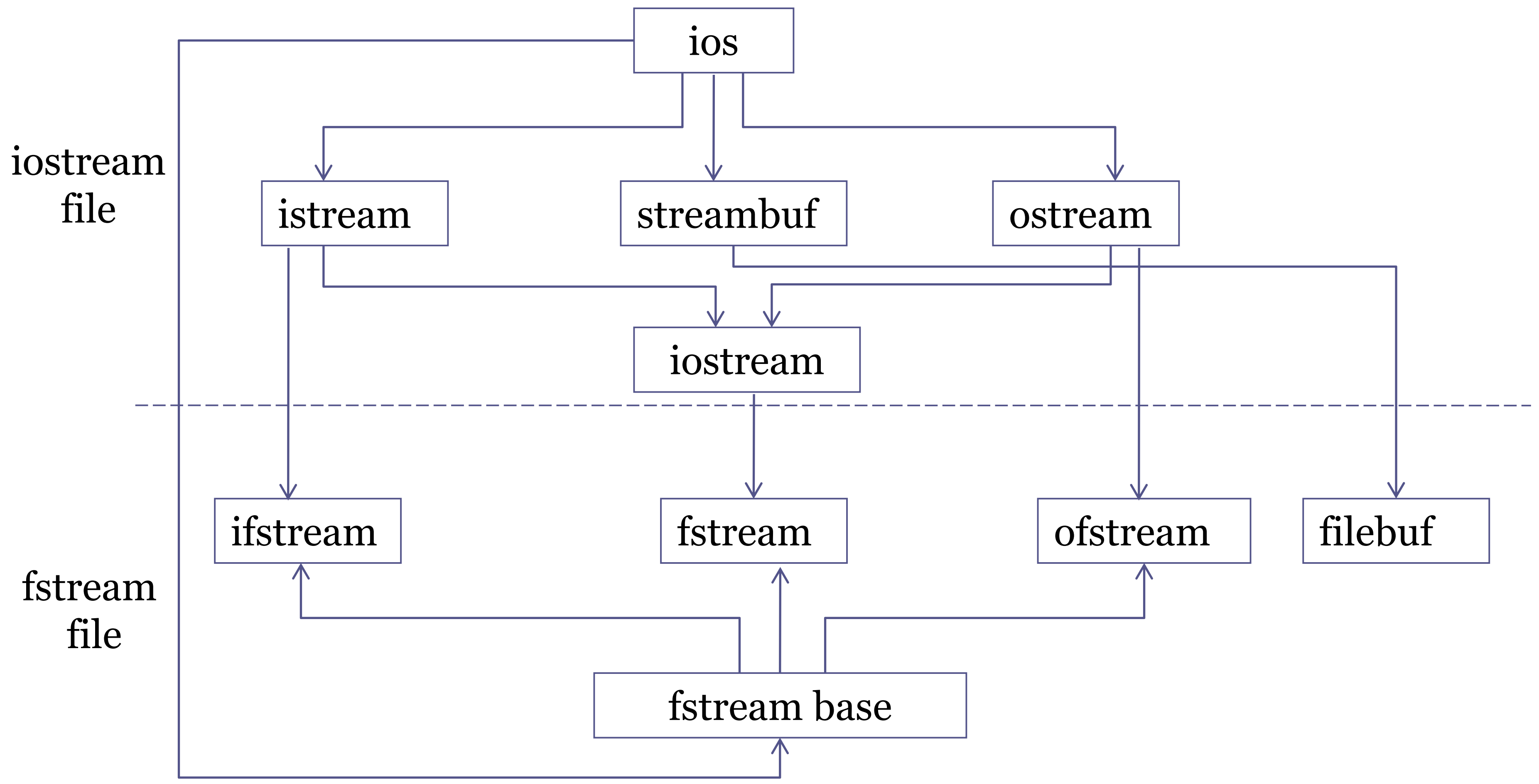
Program File Communication

- In C++ **file streams** are used as an interface between the program and the files.
- The stream that supplies data to the program is known as **input stream** and the one that receives data from the program is known as **output stream**.
- Input stream => reads data
- Output stream => writes data

File input and Output Streams



Classes for File Stream Operations



Opening and Closing a File

- To open a file, a file stream is created and then it is linked to the filename.
- A file can be opened in two ways:
  - Using the constructor function of the class.
  - Using the member function open() of the class.
- A file is closed by using the function close().
- eg: outfile.close();

Opening File using Constructor

- Filename is used to initialize the file stream object.
- Create a file stream object to manage the stream.
  - ofstream is used to create output stream.
  - ifstream is used to create input stream.
- Initialize the file object with the desired filename.
- Eg:

```
ofstream  outfile (“results”);  //      output only
ifstream   infile (“data”);             // input only
```

Opening File using Constructor

```
#include<iostream.h>
#include<fstream.h>
int main()
{
   ofstream   outf(“Item”);
   cout<<“Enter  item  name:”;
   char name[30];
   cin >> name;

   outf<<name;
   cout<<“Enter  item  cost:”;
   float cost;
   cin >> cost;

   outf << cost;
   outf.close();
```

Opening File using Constructor

```
    ifstream   inf(“Item”);

    inf >> name;
    inf >> cost;

    cout << “Item name :” << name;
    cout << “Item cost :”    << cost ;

    inf.close();
    return 0;
}
```

Output:

```
Enter item name: CD-ROM
Enter item cost: 250

Item name: CD-ROM
Item cost: 250
```

Opening Files Using open()

- The function open() can be used to open multiple files that use the same stream object.
- Syntax:

```
file-stream-class   stream-object;
stream-object.open("filename");
```

- A stream object can be connected to only one file at a time.

Opening Files Using open()

```
#include<iostream.h>
#include<fstream.h>

int main()
{
	ofstream  fout;
	fout.open(“Country”);

	fout<<“United state of America”;
	fout<<“United Kingdom”;

	fout.close();

	fout.open(“Capital”);

	fout<<“Washington”;
	fout<<“London”;

	fout.close();
```

Opening Files Using open()

```
const int N=80;
char line[N];

ifstream  fin;
fin.open(“Country”);

cout<<“Contents of country file” ;
while(fin)
{
        fin.getline(line, N);
        cout<<line;
}
fin.close();
```

Opening Files Using open()

```
        fin.open("Capital");

        cout<<"Contents of capital file";

        while(fin)
        {
                fin.getline(line, N);
                cout<<line;
        }
        fin.close();
        return 0;
}
```

Detecting End-of File

- Detection of the end-of-file condition is necessary for preventing any further attempt to read data from the file.

```
while(fin)
```

- An ifstream object return a value zero if any error occurs in the file operation including the end-of-file condition.

```
if(fin1.eof() != 0 ) { exit(1); }
```

- The eof() of ios class returns a non zero value if the end-of-file condition is encountered and zero otherwise.

Opening two files simultaneously

- When two or more files are used simultaneously ie: when we want to merge two files into a single file.
- In such case we create two separate input streams for handling the two input files and one output stream for handling the output file.

Opening two files simultaneously

```
#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>

int main()
{
    const int size = 80;
    char line[size];
    ifstream fin1, fin2;
    fin1.open("country");
    fin2.open("capital");
```

Opening two files simultaneously

```
 for(int i = 1; i <= 10; i++ )
{
     if ( fin1.eof() ! = 0)
     {
            cout << “\n Exit from country \n ” ;
            exit(1);
     }
     fin1.getline(line,size);
     cout << “Capital of “ << line;
```

Opening two files simultaneously

```
    if ( fin2.eof() ! = 0 )
    {
          cout << “ \n Exit from capital \n“ ;
          exit(1);
    }
    fin2.getline(line, size);
    cout << line << “\n”;
 }
   return 0;
 }
```

Quiz 2

- **What are default arguments?**
  - A **default argument** is a value provided in function declaration that is automatically assigned by the compiler if caller of the function doesn't provide a value for the **argument.**

File Modes

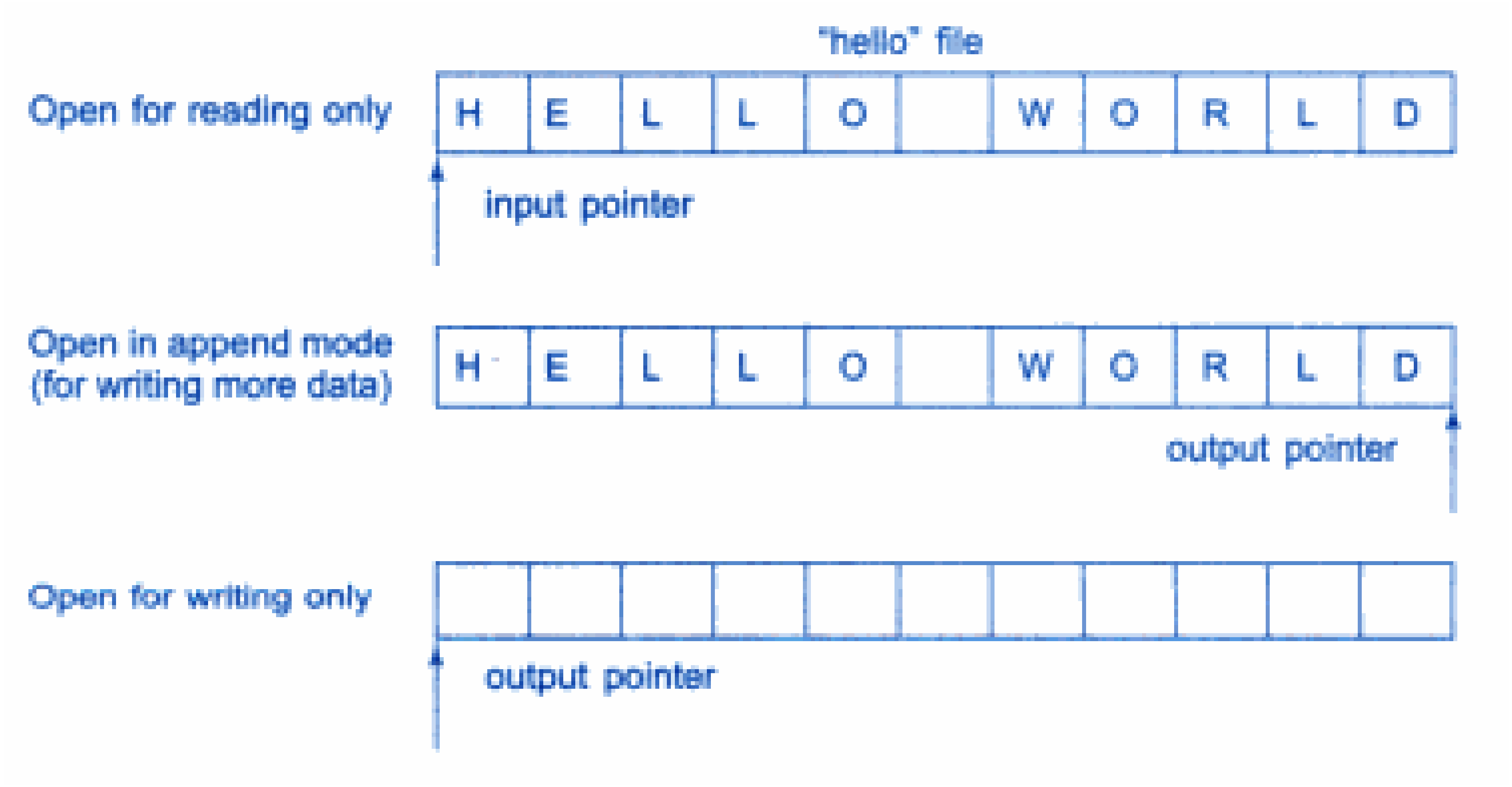
- File mode specifies the purpose for which the file is opened.
- File mode parameters:

ios::app	//Append to end-of-file
ios::ate	//go to end-of-file on opening
ios::binary	//Binary file
ios::in	//open file for reading only
ios::nocreate	//open fails if the file does not exists.
ios::noreplace	//open fails if the file already exists.
ios::out	//open file for writing only
ios::trunc	//delete the contents of file if it exists

File Pointers

- Each file has two associated pointers:
  - get pointer or input pointer: used for reading the contents of the file.
  - put pointer or output pointer: used for writing to a given file location.
- Default actions are associated with both the pointers.
  - When a file is opened in read mode the input pointer is set at the beginning.
  - When a file is opened in write mode the existing contents are deleted and output pointer is set at beginning.

Default Actions



Manipulation of File Pointers

- The user can control the movement of the pointers as per his need by using the following functions:
  - seekg() : moves get pointer to a specified location.
  - seekp(): moves put pointer to a specified location.
  - tellg() : gives the current position of the get pointer.
  - tellp() : gives the current position of the put pointer.

Quiz 3

- Where will the file pointer point in the following statement?

```
infile.seekg(10);
```

  - The file pointer will point to the 11th byte in the file.

- Where will the file pointer point after execution of the following statement?

```
ofstream fileout;
fileout.open("hello", ios::app);
int p = fileout.tellp();
```

  - The output pointer is moved to the end of the file and the value of p will represent the number of bytes in the file.

File Pointers

- File pointers seekg() and seekp() can also be used with two arguments:
  - seekg(offset, refposition);
  - seekp(offset, refposition);
- The parameter offset represents the number of bytes the file pointer is to be moved from the location specified by the parameter refposition.
- The refposition can take one of the following three constants defined in the ios class:
  - ios::beg   // start of the file
  - ios::cur   // current position of the pointer
  - ios::end   //end of the file

File Pointers (Pointer Offset calls)

- fout.seekg(0,ios::beg);   // go to start
- fout.seekg(0,ios::cur);   // stay at the current position
- fout.seekg(0,ios::end);   // go to end of the file
- fout.seekg(m,ios::beg);   // move to (m+1)th byte in the file
- fout.seekg(m,ios::cur);   //go forward by m byte from the current position
- fout.seekg(-m,ios::cur);   // go backward by m bytes from the current position
- fout.seekg(-m,ios::end);   //go backward by m bytes from the end

Sequential Input & Output Operations

- The file stream class support a number of member functions for performing the input and output operations on files.
  - put() and get() : used for handling a single character.
  - read() and write() : used for handling large blocks of binary data.

Put() & Get() functions

- Function put() writes a single character to the associated stream.
- Function get() reads a single character from the associated stream.

Put() & Get() functions

```
#include<iostream.h>
#include<fstream.h>
#include<string.h>

int main()
{
        char string[80];
        cout << “Enter a string :\n ”;
        cin >> string;
```

Put() & Get() functions

```
int len = strlen(string);
fstream file;
file.open(“Text”, ios::in | ios::out);

for(int i = 0; i < len ;  i++)
        file.put(string[i]);

file.seekg(0);
```

Put() & Get() functions

```
        char ch;
        while(file)
        {
                file.get(ch);
                cout << ch;
        }

        return 0;
}
```

Reading & Writing a class object

- C cannot handle user defined data types such as class objects.
- C++ provides read() and write() functions to read and write the objects directly.
- The length of the object is obtained using the sizeof operator.
- This length represents the sum total of lengths of all data members of the object.

Reading & Writing a class object

- Syntax:
  - infile.read ((char *) & V, sizeof (V));
  - outfile.write ((char *) & V, sizeof (V));
- The first argument is the address of the variable V.
- The second is the length of that variable in bytes.
- The address of the variable must type cast to char * (ie: pointer to character type).

Example

```
  #include<iostream.h>
  #include<fstream.h>
  class inventory
  {
        char name[10];
        int code;
        float cost;
        public:
              void readdata(void);
              void writedata(void);
};
```

Example cont…

```
void inventory :: readdata(void)
{
        cout<<"Enter name:";
        cin>> name;
        cout<<"Enter code:";
        cin>>code;
        cout<<"Enter cost:";
        cin>> cost;
}
void inventory :: writedata(void)
{
        cout<<name;
        cout<<code;
        cout<<cost;
}
```

Example cont…

```
int main()
{
    inventory item[3];
    fstream file;
    file.open("Stock.dat", ios::in | ios::out);
    cout<<"Enter the details for three items:";
    for(int i=0; i<3;i++)
    {
        item[i].readdata();
        file.write((char *) & item[i], sizeof(item[i]));
    }
```

Example cont…

```
file.seekg(0);

for(i=0;i<3;i++)
{
      file.read((char *) & item[i], sizeof(item[i]));
      item[i].writedata();
}
file.close();
return 0;
}
```

Updating a File: Random Access

- Updating is a routine task in the maintenance of any data file.
- The updating would include one or more of the following tasks:
    - Displaying the contents of a file.
    - Modifying an existing item.
    - Adding a new item.
    - Deleting an existing item.

Updating a File: Random Access

- The size of each object can be obtained using the statement:

```
int    obj_len  =  sizeof(object);
```

- The location of a desired object (say m) is obtained as:

```
int   location   =   m * obj_len;
```

- The total number of objects in a file can be obtained by using object length as:

```
int  n  =  file_size / obj_len;
```

Updating a File: Random Access

```
#include<iostream.h>
#include<fstream.h>

class inventory
{
   char name[10];
   int code;
   float cost;
   public:
         void getdata(void)
         {
               cout<<"Enter name : " ;      cin>> name;
               cout<<"Enter code : " ;      cin>>code;
               cout<<"Enter cost : " ;      cin>> cost;
         }
```

Updating a File: Random Access

```
        void putdata(void)
        {
                cout<<name;
                cout<<code;
                cout<<cost;
        }
};

int main()
{
   inventory item;
   fstream inoutfile;
   inoutfile.open(“stock.dat”, ios::ate | ios::in | ios::out|
   ios::binary);
   inoutfile.seekg(0, ios::beg);
```

Updating a File: Random Access

```
while(inoutfile.read((char * ) & item, sizeof item))
{
        item.putdata();
}
inoutfile.clear();                    //turn off EOF flag

cout<<"Add an item:";
item.getdata();
inoutfile.write((char * ) & item, sizeof item);
inoutfile.seekg(0);
while(inoutfile.read((char * ) & item, sizeof item))
{
        item.putdata();
}
```

Updating a File: Random Access

```
int last = inoutfile.tellg();   // finds the no. of objects
int n = last/sizeof(item);
cout<<“Number of objects:”<<n;
cout<<“Enter the object number to be updated:”;
int object;
cin>> object;
int location = (object-1)* sizeof(item);
inoutfile.seekp(location);

cout<<“Enter the new values of the object:”;
item.getdata();
inoutfile.write((char *) & item, sizeof item);
```

Updating a File: Random Access

```
  inoutfile.seekg(0);

  cout<<“Contents of updated file are:”;
  while(inoutfile.read((char * ) & item, sizeof item))
  {
      item.putdata();
  }
  inoutfile.close();

  return 0;
}
```

Updating a File: Random Access

Output:

```
current contents of stock:
AA        11        100
BB        22        200
CC        33        300

Add an item:
Enter name: DD
Enter code: 44
Enter cost: 400
```

Updating a File: Random Access

```
Contents of Appended file:
AA          11          100
BB          22          200
CC          33          300
DD          44          400

Number of objects: 4
Enter the object to be updated: 4
Enter new values for object:
Enter name: EE
Enter code: 55
Enter cost: 500
Contents of updated file:
AA          11          100
BB          22          200
CC          33          300
EE          55          500
```

Error Handling During File Operations

- Following conditions may arise while dealing with files:
  - A file which we are attempting to open for reading does not exists.
  - The file name used for a new file may already exists.
  - We may attempt an invalid operation such as reading past the end-of-file.
  - There may not be any space in the disk for storing more data.
  - We may use invalid file name.
  - We may attempt to perform an operation when the file is not opened for that purpose.

Error Handling During File Operations

- The ios class supports several member functions that can be used to read the status recorded in a file stream.

Function	Return value and meaning
eof( )	Returns true (non zero value) if end-of-file is encountered while reading otherwise returns false (zero).
fail( )	Returns true when an input or output operation has failed.
bad ( )	Returns true if an invalid operation is attempted or any unrecoverable error has occurred. However, if it is false, it may be possible to recover from any other error reported, and continue operation.
good ( )	Returns true if no error has occurred. When is returns false, no further operations can be carried out.

Error Handling During File Operations

```
……
……
ifstream infile;
infile.open("ABC");
while ( !infile.fail( ))
{
        ………  ( process the file )
        ………
}
if (infile.eof( ) )
{
        ………          (terminate program normally)
}
else
        if(infile.bad ( ))
{
        ……….   (report fatal error )
}
else
{
        infile.clear ( );     // clear error state
        ………
}
……….
……….
```

Summary

- ______ function is used to open multiple files that use the same stream object.
- The second argument to the open() is ________.
- The default values for opening a file with input and output stream are _______ and ________.
- Each file is associated with two pointers _____ and _______.
- ______ and ______ functions write and read blocks of binary data.
- The ios class supports many ________ for managing errors that may occur during file operations.
- A steam may be connected to more than one file at a time. (True / False)
- The fin.fail() call returns non-zero when an operation on the file has failed. (True / False)

Short Answer Questions

- What are the steps involved in using a file in a C++ program?
  - The steps involved in using a file are:
    - Opening a File
    - Perform Read and Write to a file.
    - Closing a file.
- Describe the various classes available for file operations?
  - fstream, ifstream and ofstream are the classes available for file operations.

Short Answer Questions

- What is the difference between opening a file with a constructor function and opening a file with open() function? When is one method preferred over the other?
  - When a file is opened using constructor the filename is passed to initialize the respective file stream class object whereas with open() function the file stream object is created and the filename is passed as an argument to the open() function.
  - Constructor method is used when there is only one file in the stream while the open() is used to manage multiple files using one stream.

Short Answer Questions

- What is a file mode? Describe the various file mode options available.
  - File mode specifies the purpose for which the file is opened.
  - File mode parameters:

ios::app	//Append to end-of-file
ios::ate	//go to end-of-file on opening
ios::binary	//Binary file
ios::in	//open file for reading only
ios::nocreate	//open fails if the file does not exists.
ios::noreplace	//open fails if the file already exists.
ios::out	//open file for writing only
ios::trunc	//delete the contents of file if it exists

Short Answer Questions

- Both ios::ate and ios::app place the file pointer at the end of the file. What then is the difference between them?
  - ios::app allows user to add data to end-of-file only while ios::ate mode permits to add data anywhere in the file.
- What does current position mean when applied to files?
  - Current position represent the number of bytes in the file.

Short Answer Questions

- What are the advantages of saving data in binary form?
  - Saving data in binary form has following advantages:
    - The values are stored in the same format in which they are stored in the internal memory.
    - As there is no conversions while saving the data, it is much faster.
- Describe how would we determine number of objects in a file.
  - The total number of objects in a file can be obtained by using object length as:

int n = file_size / obj_len;

Short Answer Questions

- Describe the various approaches by which we can detect the end-of-file condition successfully.
  - Various approaches which are used to detect end-of-file are:
    - **while(fin)**
      - An ifstream object return a value zero if any error occurs in the file operation including the end-of-file condition.
    - **if(fin1.eof() != 0) { exit(1); }**
      - The eof() of ios class returns a non zero value if the end-of-file condition is encountered and zero otherwise.

References

- Object Oriented Programming with C++ by E. Balagurusamy.

END OF UNIT