Week 13

File IO

Streams

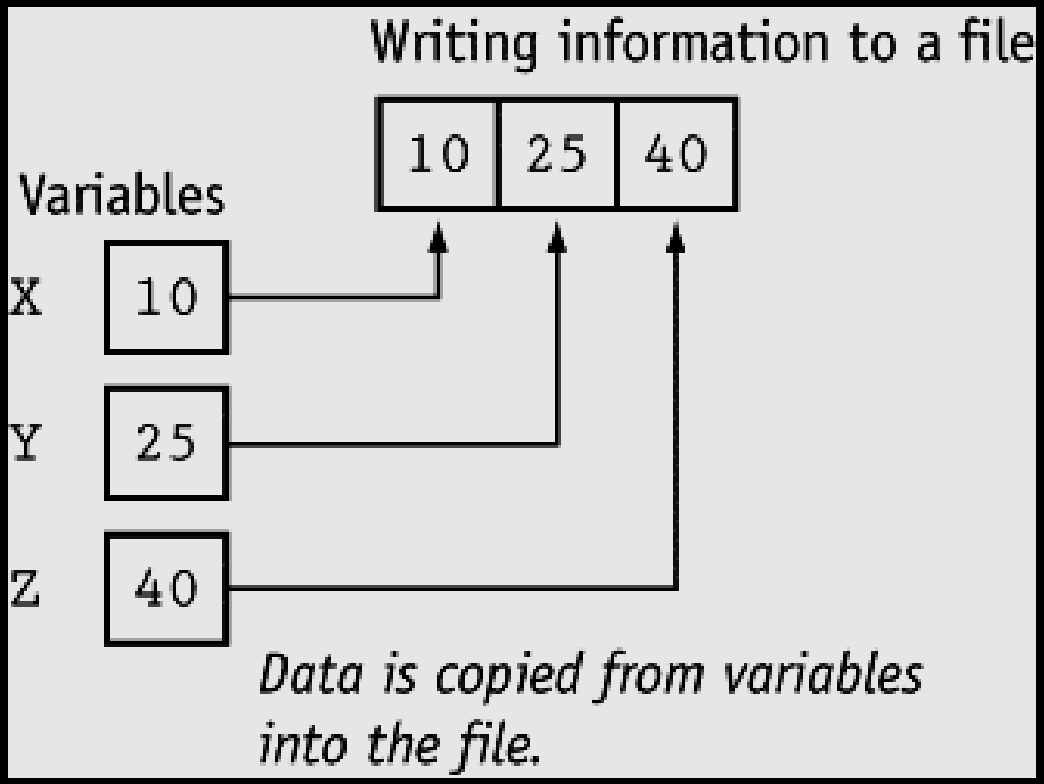
- Sequence of bytes or more commonly known as **streams**.

Input / Output Stream

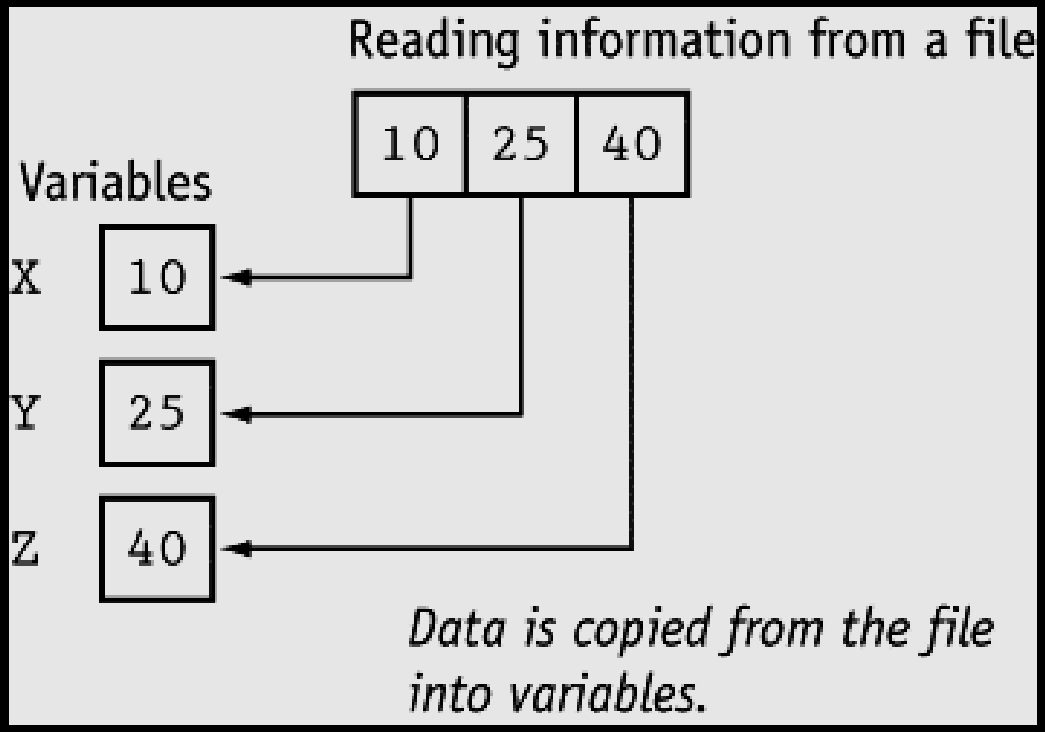
- **Input Stream:** If the direction of flow of bytes is from device(for example: Keyboard) to the main memory then this process is called input.

- **Output Stream:** If the direction of flow of bytes is opposite, i.e. from main memory to device(display screen) then this process is called output.

ofstream


Writing information to a file
10
25
40
Variables
X
10
Y
25
Z
40
Data is copied from variables into the file.

ifstream


Reading information from a file
10
25
40
Variables
X
10
Y
25
Z
40
Data is copied from the file into variables.

Header files available in C++ for Input – Output operation

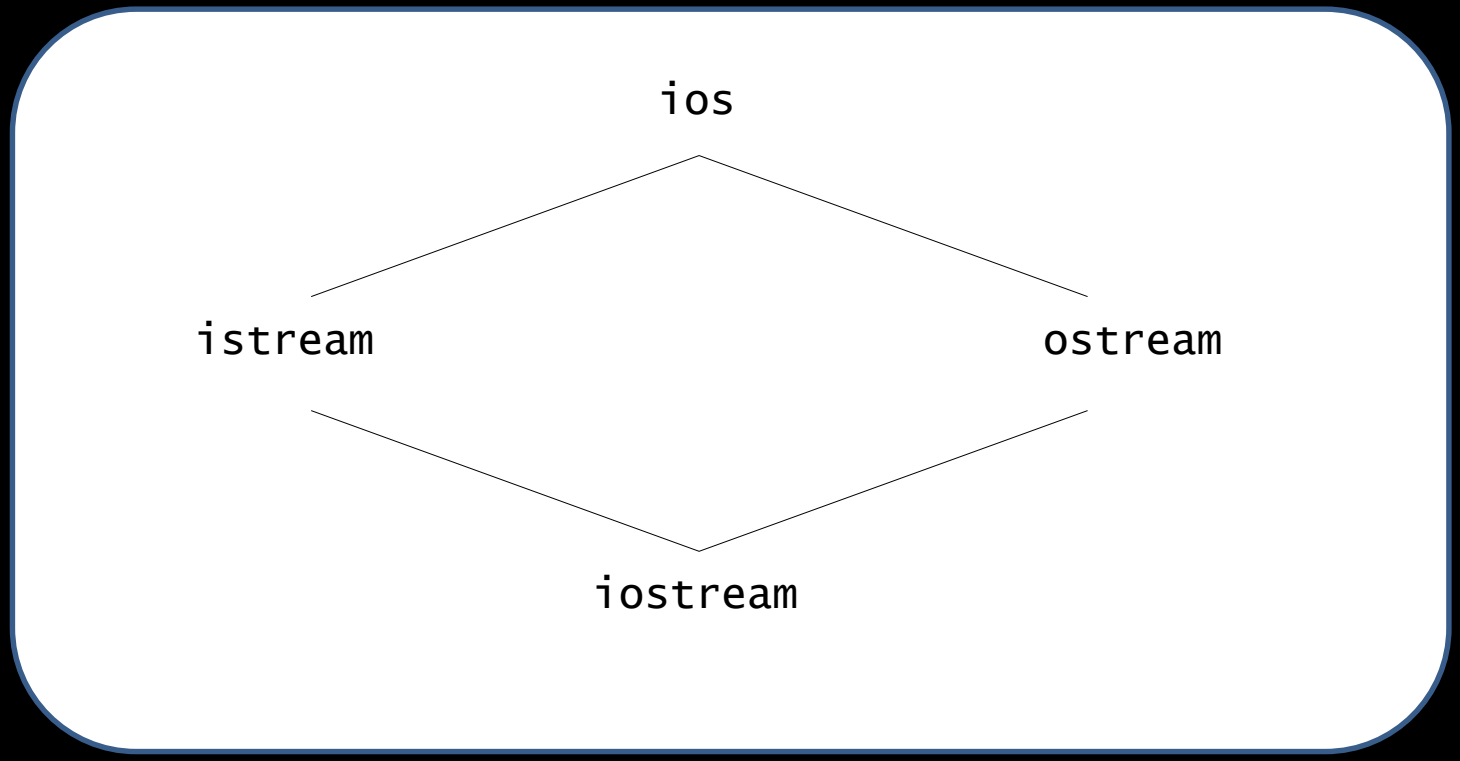
- **iostream**: iostream stands for standard input output stream. This header file contains definitions to objects like cin, cout, cerr etc.

- **iomanip**: iomanip stands for input output manipulators. The methods declared in this files are used for manipulating streams. This file contains definitions of setw, setprecision etc.

- **fstream**: This header file mainly describes the file srteam. This header file is used to handle the data being read from a file as input or data being written into the file as output.

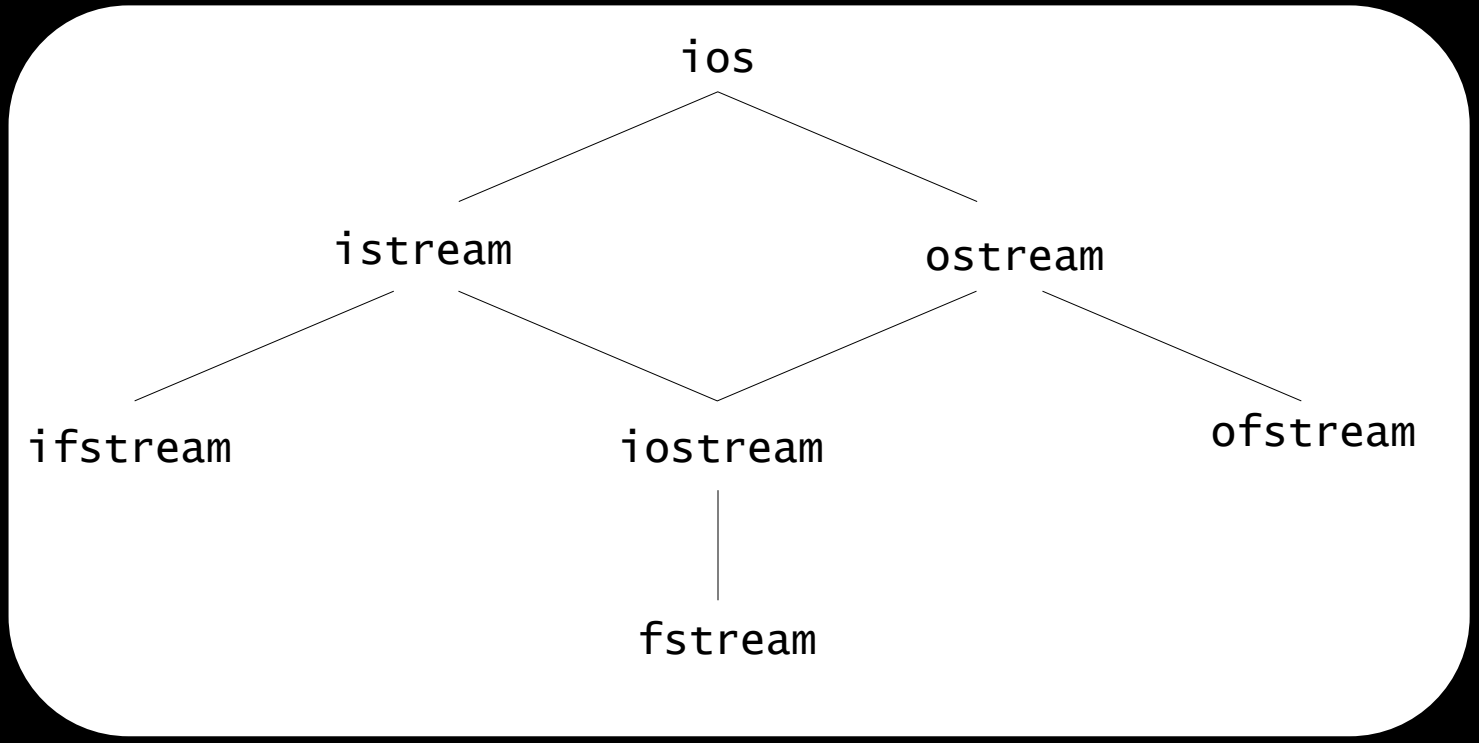
Stream Input/Output Classes and Objects

Portion of the stream I/O class hierarchy.



Stream Input/Output Classes and Objects

Portion of stream-I/O class hierarchy with key file-processing classes.



Standard output stream (cout)

- Usually the standard output device is the display screen.
- **cout** is the instance of the ostream class.
- cout is used to produce output on the standard output device which is usually the display screen.
- The data needed to be displayed on the screen is inserted in the standard output stream (cout) using the insertion operator (**<<**).

Standard input stream (cin)

- Usually the input device is the keyboard.
- cin is the instance of the class **istream** and is used to read input from the standard input device which is usually keyboard.
- The extraction operator(**>>**) is used along with the object **cin** for reading inputs.
- The extraction operator extracts the data from the object **cin** which is entered using the keyboard.

Using Input/Output Files

- ***stream*** - a sequence of characters
  - interactive (iostream)
    - **cin** - input stream associated with **keyboard.**
    - **cout** - output stream associated with **display**
  - file (fstream)
    - **ifstream** - defines new input stream (normally associated with a file).
    - **ofstream** - defines new output stream (normally associated with a file).

Stream I/O Library Header Files

- Note: There is no “.h” on standard header files : <fstream>
- iostream -- contains basic information required for all stream I/O operations
- fstream -- contains information for performing file I/O operations

File

A computer file

- is stored on a secondary storage device (e.g., disk)
- is permanent
- can be used to
  - provide input data to a program
  - or receive output data from a program
  - or both
- should reside in Project directory for easy access
- must be opened before it is used.

Types of File(s)

- Types of file supported by C++:
  - Text Files
  - Binary Files

Text files

- Text files are structured as a sequence of lines, where each line includes a sequence of characters.
- Each line is terminated with a special character, called the EOL.
- There are several types, but the most common is the comma {,} or newline character.
- There are several types, but the most common is the comma {,} or newline character.
- It ends the current line and tells the interpreter a new one has begun.

Difference between text file and binary file

- Text file is human readable because everything is stored in terms of text.
- In binary file everything is written in terms of 0 and 1, therefore binary file is not human readable.

- A newline(\n) character is converted into the carriage return-linefeed combination before being written to the disk.
- In binary file, these conversions will not take place.

Difference between text file and binary file

- In text file, a special character, whose ASCII value is 26, is inserted after the last character in the file to mark the end of file.
- There is no such special character present in the binary mode files to mark the end of file

Table 1: ASCII Reference: Nonprintable Characters

Dec	Hex	Oct	Char	Comment
0	0	0	NUL	Null
1	1	1	SOH	Start of Heading
2	2	2	STX	Start of Text
3	3	3	ETX	End of Text
4	4	4	EOT	End of Transmission
5	5	5	ENQ	Enquiry
6	6	6	ACK	Acknowledge
7	7	7	BEL	Bell
8	8	10	BS	Backspace
9	9	11	TAB	Horizontal Tab
10	A	12	LF	Line Feed
11	B	13	VT	Vertical Tab
12	C	14	FF	Form Feed
13	D	15	CR	Carriage Return
14	E	16	SO	Shift Out
15	F	17	SI	Shift In
16	10	20	DLE	Data Link Escape
17	11	21	DC1	Device Control 1
18	12	22	DC2	Device Control 2
19	13	23	DC3	Device Control 3
20	14	24	DC4	Device Control 4
21	15	25	NAK	Negative Acknowledgement
22	16	26	SYN	Synchronous Idle
23	17	27	ETB	End of Transmission Block
24	18	30	CAN	Cancel
25	19	31	EM	End of Medium
26	1A	32	SUB	Substitute
27	1B	33	ESC	Escape
28	1C	34	FS	File Separator
29	1D	35	GS	Group Separator
30	1E	36	RS	Record Separator
31	1F	37	US	Unit Separator

The fstream.h Header File

- C++'s standard library called **fstream**, defines the following classes to support file handling.

- **ofstream class :** Provides methods for writing data into file. Such as, open(), put(), write(), seekp(), tellp(), close(), etc.

- **ifstream class :** Provides methods for reading data from file. Such as, open(),get(), read(), seekg(), tellg(), close(), etc.

- **fstream class :** Provides methods for both writing and reading data from file. The fstream class includes all the methods of ifstream and ofstream class.

Reading and Writing into File

- We can read data from file and write data to file in four ways.

1. Reading or writing characters using get() and put() member functions.

2. Reading or writing formatted I/O using insertion operator (<<) and extraction operator (>>).

3. Reading or writing object using read() and write() member functions.

General File I/O Steps

- When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.

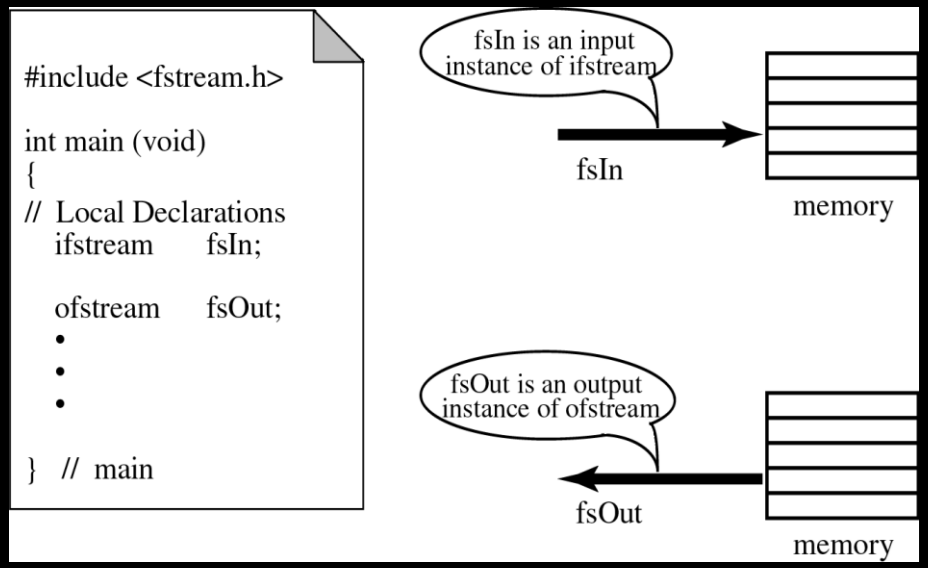
- Hence, a file operation takes place in the following order.
  - Open a file
  - Read or write (perform operation)
  - Close the file

General File I/O Steps in C++

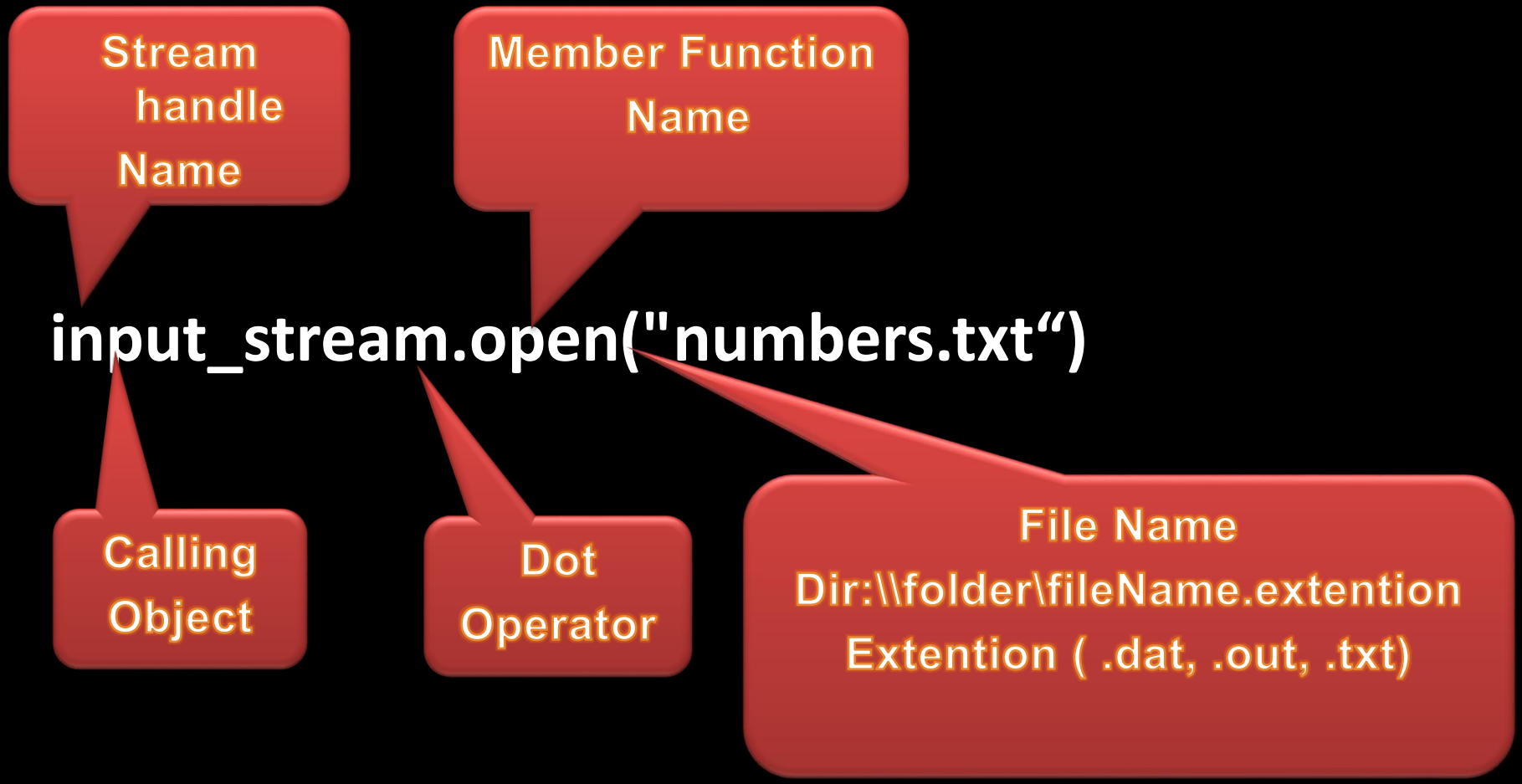
1. Include the header file fstream in the program.
2. Declare file stream variables.
3. Associate the file stream variables with the input/output sources.
4. Open the file
5. Use the file stream variables with >>, <<, or other input/output functions.
6. Close the file.

C++ streams

```
#include <fstream>
int main ()
{
ifstream  fsIn;//input
ofstream fsOut; // output
fstream both //input & output
fsIn.open("prog1.txt");
fsOut.open("prog2.txt");
//Code for data manipulation
.
.
fsIn.close();
fsOut.close();
return 0; }
```



Object and Member Functions



Open()

- Opening a file associates a file stream variable declared in the program with a physical file at the source, such as a disk.
- In the case of an input file:
  - the file must exist before the open statement executes.
  - If the file does not exist, the open statement fails and the input stream enters the fail state
- An output file does not have to exist before it is opened;
  - if the output file does not exist, the computer prepares an empty file for output.
  - If the designated output file already exists, by default, the old contents are erased when the file is opened.

Opening a File

- Before data can be written to or read from a file, the file must be opened.

  File can be opened either by:

  1. ifstream inputFile;

  inputFile.open(“customer.txt”);

  OR

  2. Ifstream inputFile(“customer.txt”);

File I/O Example: Writing

First Method (use the constructor)

```
#include <fstream>
using namespace std;
int main()
{/* declare and
automatically open the
file*/
ofstream
outFile("fout.txt");
outFile << "Hello
World!";
outFile.close();
return 0;
}
```

Second Method (use Open function)

```
#include <fstream>
using namespace std;
int main()
{
ofstream outFile;
outFile.open("fout.txt
”);
outFile << "Hello
World!";
outFile.close();
return 0;
}
```

Validate the file before trying to access

First method

By checking the stream variable;

```
If ( ! Mystream)
{
Cout << “Cannot open file.\n ”;
}
```

Second method

By using bool is_open() function.

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

File I/O Example: Reading

Read char by char

```
#include <iostream>
#include <fstream>

int main()
{//Declare and open a text file
ifstream openFile(“data.txt");
 char ch;
 //do until the end of file
while( ! OpenFile.eof() )
{
OpenFile.get(ch); // get one character
cout << ch;   // display the character
}
OpenFile.close(); // close the file
```

Read a line

```
#include <iostream>
#include <fstream>
#include <string>
int main()
{//Declare and open a text file
ifstream openFile("data.txt");
string line;
while(!openFile.eof())
{//fetch line from data.txt and put it in a string
getline(openFile, line);
cout << line;
}
openFile.close(); // close the file
    return 0; }
```

File Open Mode

Name	Description
ios::in	Open file to read
ios::out	Open file to write
ios::app	All the data you write, is put at the end of the file. It calls ios::out
ios::ate	All the data you write, is put at the end of the file. It does not call ios::out
ios::trunc	Deletes all previous content in the file. (empties the file)
ios::nocreate	If the file does not exists, opening it with the open() function gets impossible.
ios::noreplace	If the file exists, trying to open it with the open() function, returns an error.
ios::binary	Opens the file in binary mode.

File Open Mode

```
#include <fstream>
int main(void)
{
ofstream outFile("file1.txt", ios::out);
outFile << "That's new!\n";
outFile.close();
        Return 0;
}
```

If you want to set more than one open mode, just use the **OR** operator- **|**. This way:

ios::ate | ios::binary

Mode of file opening

ios :: out = open file for write only

ios :: in = open file for read only

ios :: app = append to end-of-file

ios :: ate = take us to the end of the file when it is opened

Both ios :: app and ios :: ate take us to the end of the file when it is opened. The difference between the two parameters is that the ios :: app allows us to add data to the end of file only, while ios :: ate mode permits us to add data or to modify the existing data any where in the file.

The mode can combine two or more parameters using the bitwise **OR** operator (symbol |)

eg :-

```
fstream file;
file . Open(“ data . txt”, ios :: out | ios :: in);
```

Detecting the End of a File

- The `eof()` member function reports when the end of a file has been encountered.

```
if (inFile.eof())
    inFile.close();
```

File pointer

- Each file have two associated pointers known as the file pointers.
- One of them is called the input pointer (or get pointer) and the other is called the output pointer (or put pointer).
- The input pointer is used for reading the contents of a given file location and the output pointer is used for writing to a given file location.

Function for manipulation of file pointer

When we want to move file pointer to desired position then use these function for manage the file pointers.

Seekg () = moves get pointer (input) to a specified location

Seekp () = moves put pointer (output) to a specified location

tellg () = gives the current position of the get pointer

tellp () = gives the current position of the put pointer

fout . seekg(0, ios :: beg) -- go to start

fout . seekg(0, ios :: cur) -- stay at current position

fout . seekg(0, ios :: end) -- go to the end of file

fout . seekg(m, ios :: beg) -- move to m+1 byte in the file

fout . seekg(m, ios :: cur) -- go forward by m bytes 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

put() and get() function

The function put() write a single character to the associated stream. Similarly, the function get() reads a single character from the associated stream.

C++ put() function

- The put() function is member of ofstream class. It is used to write single character into file

Syntax of put() function

```
ofstream obj;
obj.put(char ch);
```

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

void main()
{
    ofstream fout;
    char ch;

    fout.open("demo.txt");              //Statement 1

    do                                  //Statement 2
    {

        cin.get(ch);
        fout.put(ch);

    }while(ch!=EOF);

    fout.close();

    cout<<"\nData written successfully...";

}
```

Output :

```
Hello friends, my name is kumar.^Z
Data written successfully...
```

In this example, statement 1 will create a file named demo.txt in write mode. Statement 2 is do-while loop, which take characters from user and write the character to the file, until user press the ctrl+z to exit from the loop.

C++ get() function

- The get() function is member of ifstream class. It is used to read character form the file.
- The get() function reads a single character from the file and puts that value in ch.

Syntax of get() function

```
ifstream obj;
obj.get(char &ch);
```

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

void main()
{
    ifstream fin;
    char ch;

    fin.open("demo.txt");              //Statement 1

    cout<<"\nData in file...";

    while(!fin.eof())                  //Statement 2
    {
        fin.get(ch);
        cout<<ch;
    }

    fin.close();

}
```

Output :

```
Data in file...
Hello friends, my name is kumar.
```

In this example, Statement 1 will open an existing file demo.txt in read mode and statement 2 will read all the characters one by one upto EOF(end-of-file) reached. The eof() function is member of ifsream class. It returns zero, if end-of-file reached and returns non-zero value, if any character found.

read() and write() function

```
file . read ((char *)&V , sizeof (V));
file . Write ((char *)&V , sizeof (V));
```

These function take two arguments. The first is the address of the variable V , and the second is the length of that variable in bytes . The address of variable must be cast to type char * (i.e pointer to character type) .

Program for file handling

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

Class student
{
     Public:
     Struct  stu
     {
       char name[20];
       int roll;
     }s;
   Void put_data();
   Void get_data();
 };
```

```
void student :: put_data()
   {
     cout<<"enter name  ";
     cin>>s. name;
     cout<<"enter roll  ";
     cin>>s. roll;
     file. Open ("hit. txt“ , ios :: out | ios :: app);
     file. write ((char *)this, sizeof (student));
     file. close();
     getch();
     get_data();
    }
```

```
void student :: get_data()
  {
    int  temp;
    cout<<"enter roll no. ";
    cin >>temp;
    fstream file;
    file. open ("hit . txt", ios :: in);
    file.seekg(0,ios::beg);
    While (file . read ((char *) this, sizeof (student)))
     {
       If (temp==s. roll)
        {
           cout<<"student name "<< s . name<<"\n";
           cout<<"student roll "<< s . roll;
         }
      }
     getch ();
    }
```

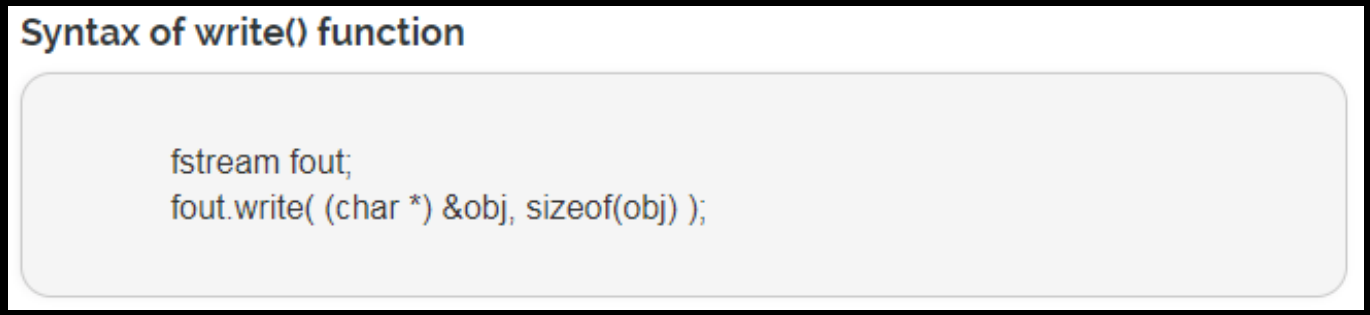
```
void main()
{
  clrscr();
  student st;
  st .put_data();
}
```

- file_obj.write((char*)&obj, sizeof(obj));

- // Reading from file into object "obj"
- file_obj.read((char*)&obj, sizeof(obj));

C++ write() function

- The write() function is used to write object or record (sequence of bytes) to the file. A record may be an array, structure or class.



- The write() function takes two arguments.
  &obj : Initial byte of an object stored in memory.
  sizeof(obj) : size of object represents the total number of bytes to be written from initial byte.

C++ write() function

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

class Student
{

    int roll;
    char name[25];
    float marks;

    void getdata()
    {
        cout<<"\n\nEnter Roll : ";
        cin>>roll;

        cout<<"\nEnter Name : ";
        cin>>name;

        cout<<"\nEnter Marks : ";
        cin>>marks;
    }

    public:
    void AddRecord()
    {

        fstream f;
        Student Stu;

        f.open("Student.dat",ios::app|ios::binary);

        Stu.getdata();

        f.write( (char *) &Stu, sizeof(Stu) );

        f.close();

    }
};
```

```
void main()
{

    Student S;
    char ch='n';

    do
    {
        S.AddRecord();

        cout<<"\n\nDo you want to add another data (y/n) : ";
        ch = getche();

    } while(ch=='y' || ch=='Y');

    cout<<"\nData written successfully...";
}
```

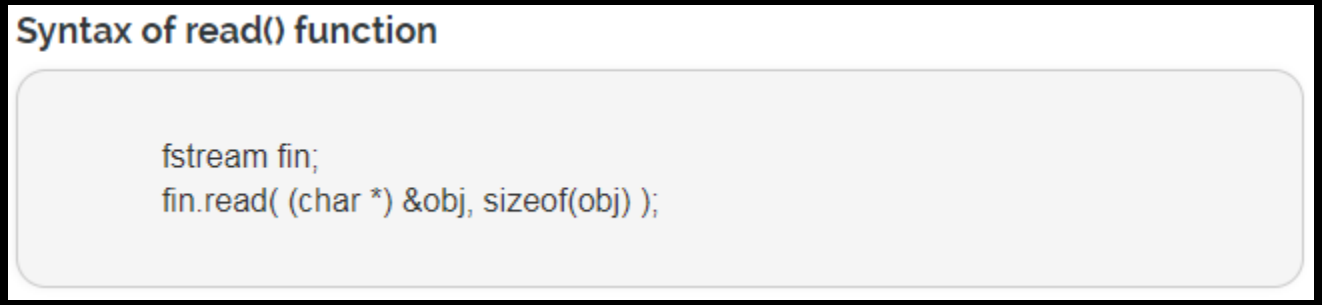
C++ write() function

Output :

```
Enter Roll : 1
Enter Name : Ashish
Enter Marks : 78.53

Do you want to add another data (y/n) : y

Enter Roll : 2
Enter Name : Kaushal
Enter Marks : 72.65

Do you want to add another data (y/n) : y

Enter Roll : 3
Enter Name : Vishwas
Enter Marks : 82.65

Do you want to add another data (y/n) : n

Data written successfully...
```

C++ read() function

- The read() function is used to read object (sequence of bytes) to the file.



- The read() function takes two arguments.
  &obj : Initial byte of an object stored in file.
  sizeof(obj) : size of object represents the total number of bytes to be read from initial byte.
- The read() function returns NULL if no data read.

C++ read() function

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

class Student
{

    int roll;
    char name[25];
    float marks;

    void putdata()
    {

       cout<<"\n\t"<<roll<<"\t"<<name<<"\t"<<marks;

    }

    public:

    void Display()
    {

        fstream f;
        Student Stu;

        f.open("Student.dat",ios::in|ios::binary);

        cout<<"\n\tRoll\tName\tMarks\n";

        while( (f.read((char*)&Stu,sizeof(Stu))) != NULL )
              Stu.putdata();

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

```
void main()
{

    Student S;

    S.Display();
}
```

Output :

Roll	Name	Marks
1	Ashish	78.53
2	Kaushal	72.65
3	Vishwas	82.65

C++ Insertion operator (<<)

- Insertion operator (<<) is similar to fprintf() function available in C language.
- It is used in situation where we need to write characters, strings and integers in a single file or we can say, it is used to write mixed type in the

Syntax of Insertion operator (<<)

```
ofstream obj;
obj << variable-list;
```

operator (<<)

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

void main()
{
      ofstream fout;

      int roll;
      char name[25];
      float marks;

      char ch;

      fout.open("demo.txt");                    //Statement   1

      do
      {
            cout<<"\n\nEnter Roll : ";
            cin>>roll;

            cout<<"\nEnter Name : ";
            cin>>name;

            cout<<"\nEnter Marks : ";
            cin>>marks;

            fout << roll << "  " << name << "  " << marks << "  "; //Statement   2

            cout<<"\n\nDo you want to add another data (y/n) : ";
            ch = getche();

      }while(ch=='y' || ch=='Y');

      cout<<"\nData written successfully...";

      fout.close();
}
```

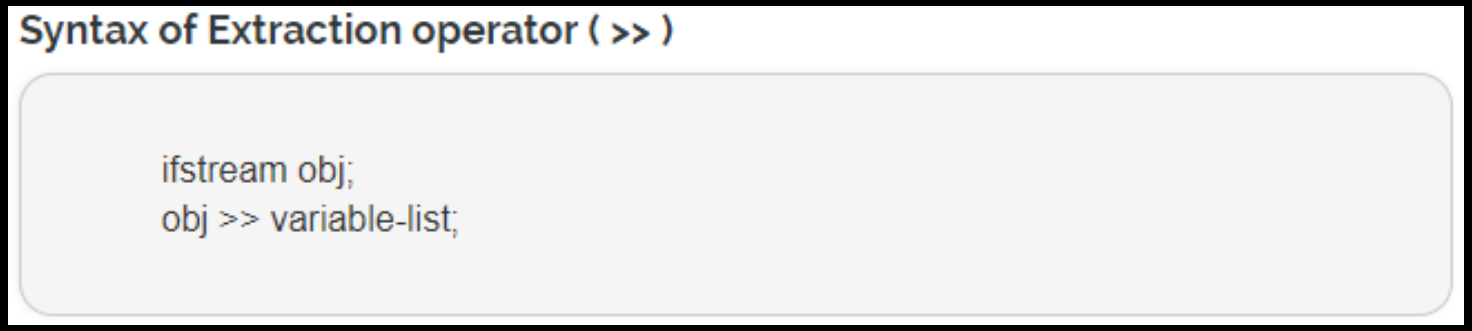
Output :

```
Enter Roll : 1
Enter Name : Kumar
Enter Marks : 78.53

Do you want to add another data (y/n) : y

Enter Roll : 2
Enter Name : Sumit
Enter Marks : 89.62

Do you want to add another data (y/n) : n

Data written successfully...
```

In the above example, Statement 1 will open an existing file **demo.txt** in write mode and statement 2 will write all the data in file.
Note : We must insert blank space in file to separate each values, as shown in **statement 2**, because extraction operator terminates at blank space and consider next value as new value.

C++ Extraction operator (>>)

- The extraction operator (>>) is similar to fscanf() function available in C language. It is used to read mixed type form the file.



action operator (>>)

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

void main()
{
      ifstream fin;

      int roll;
      char name[25];
      float marks;

      char ch;

      fin.open("demo.txt");          //Statement  1

      cout<<"\n\tData in file...\n";

      while(fin>>roll>>name>>marks)              //Statement  2
      {

         cout << "\n\t" << roll << "\t" << name << "\t" << marks;
      }

      fin.close();
}
```

Output :

```
      Data in file...

      1    Kumar       78.53
      2    Sumit       89.62
```

In the above example, Statement 1 will open an existing file demo.txt in read mode and statement 2 will read all the data upto EOF(end-of-file) reached.

Review

See Code Examples