

# Introduction to C++ Programming (PLC144)

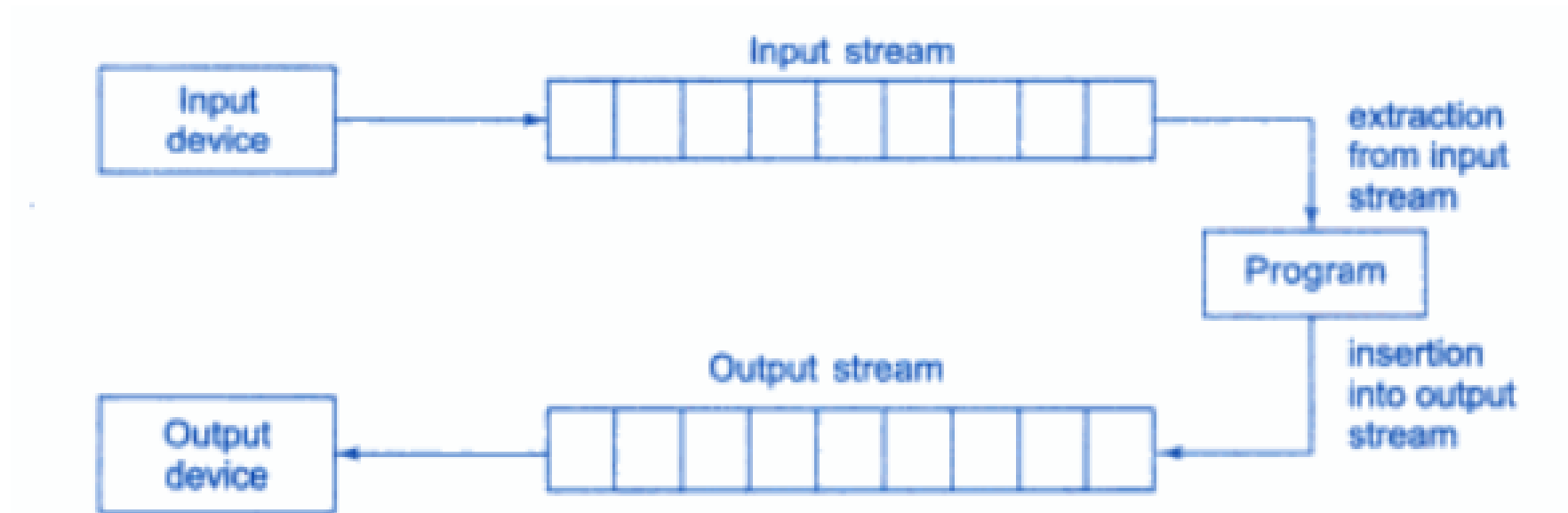
## UNIT – 4

Text Book: Object-Oriented Programming with C++ , E-Balaguruswamy.

Programming with ANSI C++, Trivedi Bhushan

# Managing console I/O Operations

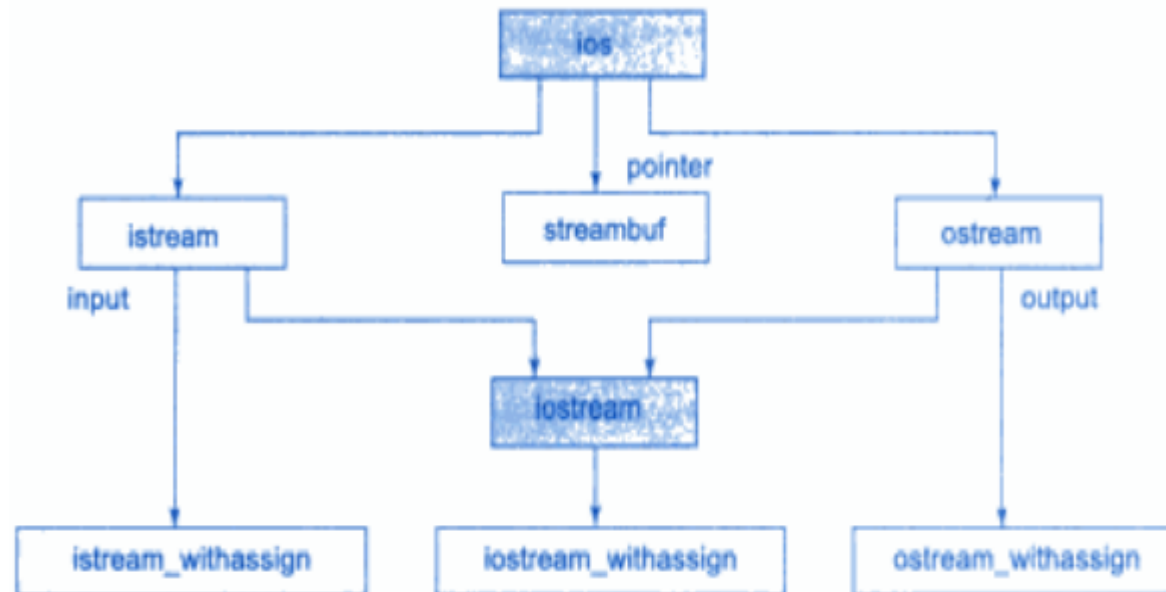
- C++ supports a rich set of I/O functions and operations.
- C++ uses stream and stream classes to implement its I/O operation with console and disk files.
- I/O system in C++ is designed to work with a wide variety of devices including terminals, disks etc..
- I/O system supplies an interface to the 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.



- The data in the input stream come from the keyboard or any other storage device.
- The data in the output stream can go the screen or any other storage device.
- C++ has predefined streams , E.g cin and cout

# C++ stream classes

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



- 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`,

# Stream Classes for console operations.

<i>Class name</i>	<i>Contents</i>
<b>ios</b> (General input/output stream class)	<ul style="list-style-type: none"><li>▪ Contains basic facilities that are used by all other input and output classes</li><li>▪ Also contains a pointer to a buffer object (<b>streambuf</b> object)</li><li>▪ Declares constants and functions that are necessary for handling formatted input and output operations</li></ul>
<b>istream</b> (input stream)	<ul style="list-style-type: none"><li>▪ Inherits the properties of <b>ios</b></li><li>▪ Declares input functions such as <b>get()</b>, <b>getline()</b> and <b>read()</b></li><li>▪ Contains overloaded extraction operator <b>&gt;&gt;</b></li></ul>
<b>ostream</b> (output stream)	<ul style="list-style-type: none"><li>▪ Inherits the properties of <b>ios</b></li><li>▪ Declares output functions <b>put()</b> and <b>write()</b></li><li>▪ Contains overloaded insertion operator <b>&lt;&lt;</b></li></ul>
<b>iostream</b> (input/output stream)	<ul style="list-style-type: none"><li>▪ Inherits the properties of <b>ios</b>, <b>istream</b> and <b>ostream</b> through multiple inheritance and thus contains all the input and output functions</li></ul>
<b>streambuf</b>	<ul style="list-style-type: none"><li>▪ Provides an interface to physical devices through buffers</li><li>▪ Acts as a base for <b>filebuf</b> class used ios files</li></ul>

# Unformatted input/output operations In C++

- **cin** and **cout** for the input and the output of data of various types with overloaded >> & << operators.
- >> is overloaded in istream class and << is overloaded in the ostream class.
- General format for reading data from keyboard

`cin>> Variable1>>Variable2.....>>VariableN`

- The input data are separated by white spaces and should match the type of variable in the cin list.
- General format for displaying data on screen is

`cout<< Item1<<Item2 .....<<ItemN`

The Item1 to ItemN may be the variables or constants of any basic type.

- The classes `istream` and `ostream` define two member functions `get()` and `put()` to handle the single character input / output operations.
- `put()` and `get()` functions
  - Eg:  
`char data;`  
`cin.get(data); // get character from keyboard and assign it to data`  
`cout.put(data); //Display the character on screen.`
- Read and display a line of text using line oriented input / output functions `getline()` and `write()`
- `getline()` reads whole line of text that ends with newline character.
- `write()` function displays an entire line
- `getline()` and `write` functions
  - Eg:  
`char line[100];`  
`cin.getline(line, 11); // The input will be terminated after reading 10 characters.`  
`cout.write(line, 11); // first arg represents name of string and 2nd arg indicates 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 format functions

<i><b>Function</b></i>	<i><b>Task</b></i>
<b>Width ()</b>	To specify the required field size for displaying an output value
<b>precision ()</b>	To specify the number of digits to be displayed after the decimal point of a float value
<b>fill()</b>	To specify a character that is used to fill the unused portion of a field
<b>setf()</b>	To specify format flags that can control the form of output display (such as left-justification and right-justification)
<b>unsetf()</b>	To clear the flags specified

# Defining field width : width()

```
cout.width(5);  
cout << 543;  
cout.width(5);  
cout << 12 << "\n";
```

This produces the following output:

		5	4	3				1	2
--	--	---	---	---	--	--	--	---	---

# Setting precision : precision()

```
cout.precision(3);  
cout << sqrt(2) << "\n";  
cout << 3.14159 << "\n";  
cout << 2.50032 << "\n";
```

will produce the following output:

```
1.414    (truncated)  
3.142    (rounded to the nearest cent)  
2.5      (no trailing zeros)
```

# Formatting Flags, Bitfields 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.
- The arg2 is the bit field specifies the group to which the formatting flag belongs.
- There are three bit fields and each has group of format flags.

# Flags and bit fields for setf()

<i>Format required</i>	<i>Flag (arg1)</i>	<i>Bit-field (arg2)</i>
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

Consider the following segment of code:

```
cout.fill('*');  
cout.setf(ios::left, ios::adjustfield);  
cout.width(15);  
cout << "TABLE 1" <<  "\n";
```

This will produce the following output:

T	A	B	L	E		1	*	*	*	*	*	*	*	*
---	---	---	---	---	--	---	---	---	---	---	---	---	---	---



# Flags that do not have bit fields: setf(arg1)

- The arg1 is one of the formatting flags specifying the action required.

<i><b>Flag</b></i>	<i><b>Meaning</b></i>
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 :: skipws	Skip white space on input
ios :: unitbuf	Flush all streams after insertion
ios :: stdio	Flush <b>stdout</b> and <b>stderr</b> after insertion



- **Some important manipulators in <ios> are:**
  - **showpos:** It forces to show a positive sign on positive numbers.
  - **noshowpos:** It forces not to write a positive sign on positive numbers.
  - **showbase:** It indicates the numeric base of numeric values.
  - **uppercase:** It forces uppercase letters for numeric values.
  - **nouppercase:** It forces lowercase letters for numeric values.
  - **fixed:** It uses decimal notation for floating-point values.
  - **scientific:** It uses scientific floating-point notation.
  - **hex:** Read and write hexadecimal values for integers and it works same as the `setbase(16)`.
  - **dec:** Read and write decimal values for integers i.e. `setbase(10)`.
  - **oct:** Read and write octal values for integers i.e. `setbase(10)`.
  - **left:** It adjusts output to the left.
  - **right:** It adjusts output to the right.

# Formatted I/O in C++:

## Formatting using Manipulators in *<iomanip>*

- 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 of the `ios` member function and flags.
- Manipulators can be used as a chain in one statement.

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

```
cout<< manip1<<item1<<manip2<<item2;
```

- 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.

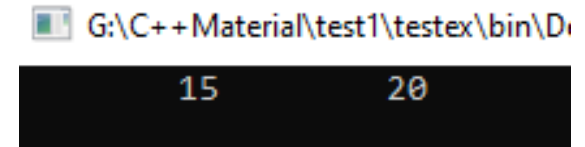
# Formatting using Manipulators in *<iomanip>*

- **setw (val):** It is used to set the field width in output operations.
- **setfill (c):** It is used to fill the character 'c' on output stream.
- **setprecision (val):** It sets val as the new value for the precision of floating-point values.
- **setbase(val):** It is used to set the numeric base value for numeric values.
- **setiosflags(flag):** It is used to set the format flags specified by parameter mask.
- **resetiosflags(m):** It is used to reset the format flags specified by parameter mask.

<b><i>Manipulators</i></b>	<b><i>Equivalent ios function</i></b>
<b>setw()</b>	<b>width()</b>
<b>setprecision()</b>	<b>precision()</b>
<b>setfill()</b>	<b>fill()</b>
<b>setiosflags()</b>	<b>setf()</b>
<b>resetiosflags()</b>	<b>unsetf()</b>

Eg 1:

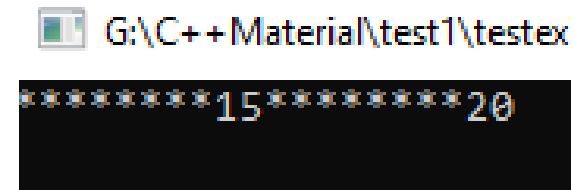
```
int a=15; int b=20;  
cout << setw(10) << a << setw(10) << b << endl;
```



G:\C++Material\test1\testex\bin\Di  
15 20

Eg 2:

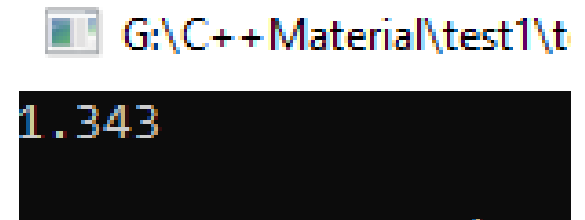
```
int a=15,b=20;  
cout<<setfill('*');  
cout<<setw(10)<<a<<setw(10)<<b<<endl;
```



G:\C++Material\test1\testex  
\*\*\*\*\*15\*\*\*\*\*20

Eg 3:

```
float A = 1.34255;  
cout <<fixed<< setprecision(3) << A << endl;
```



G:\C++Material\test1\t  
1.343

Eg 4:

```
int number = 100;
```

```
cout << "Hex Value =" << " " << hex << number << endl;
```

```
cout << "Octal Value=" << " " << oct << number << endl;
```

```
cout << "Setbase Value=" << " " << setbase(8) << number << endl;
```

```
cout << "Setbase Value=" << " " << setbase(16) << number << endl;
```

 G:\C++\Material\test1\testex\

```
Hex Value = 64
Octal Value= 144
Setbase Value= 144
Setbase Value= 64
```

# What is a File?

- A file is a collection on information, usually stored on a computer's disk.
- Information can be saved to files and then later reused.

## File Names:

All files are assigned a name that is used for identification purposes by the operating system and the user.

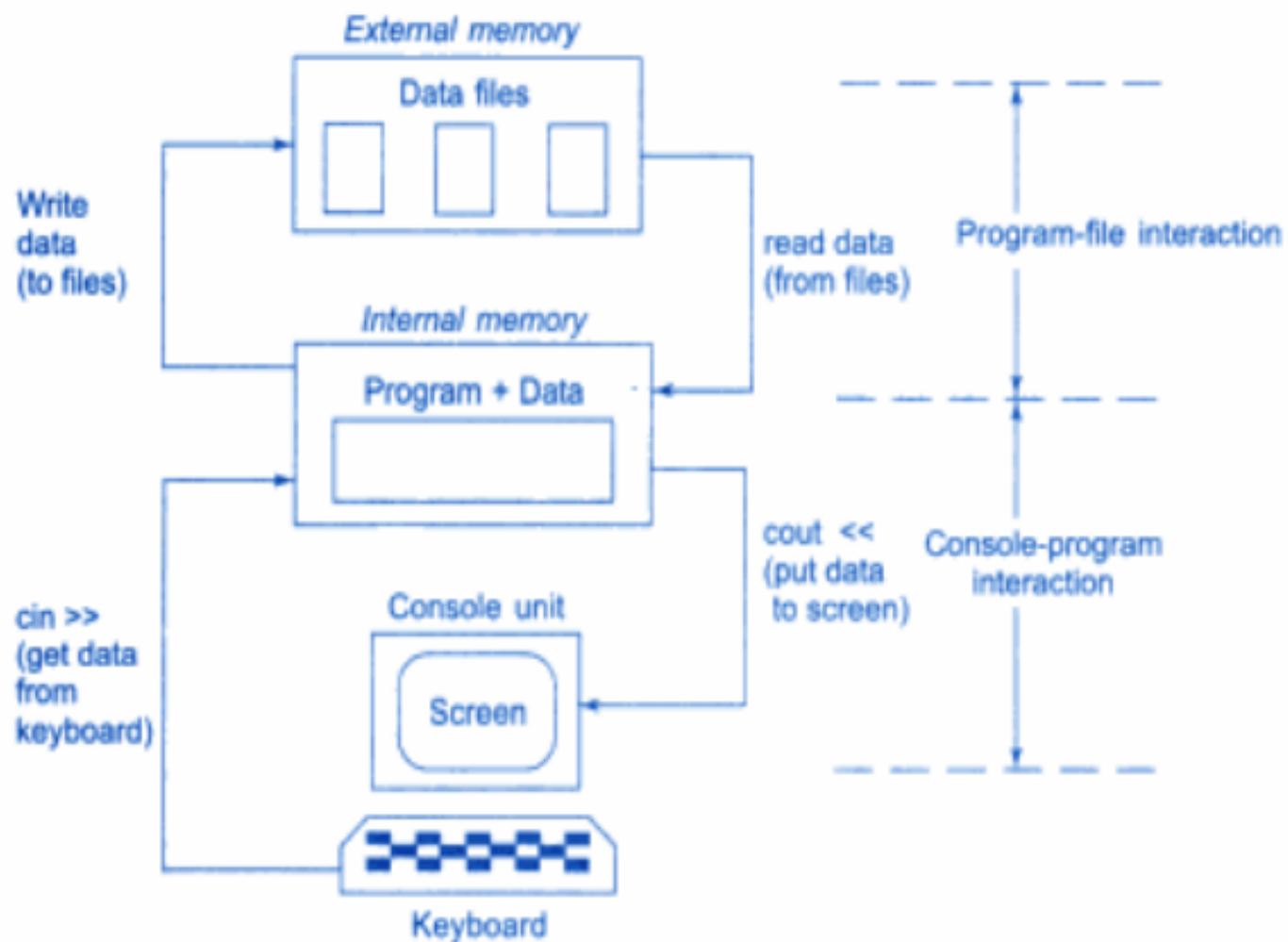


# Working With Files

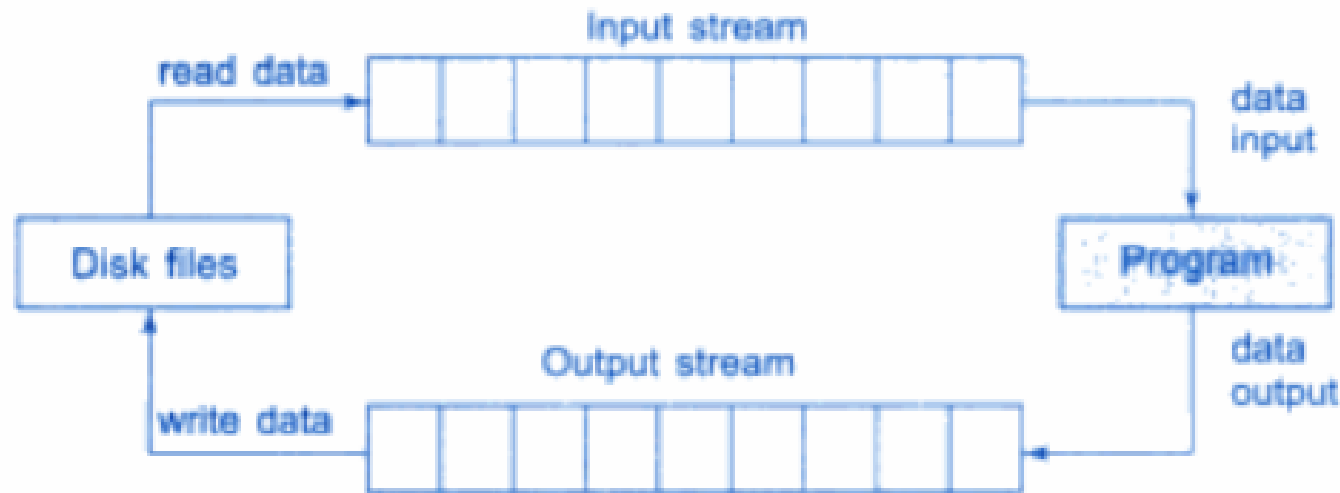
A computer file

- 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.
- input/output (I/O) operations are special because they are to be performed with the help of the operating system (OS) and the device drivers

- A program involves following kinds of data communication
  - Data Transfer between the console unit & the program.
  - Data transfer between the program and a disk file.



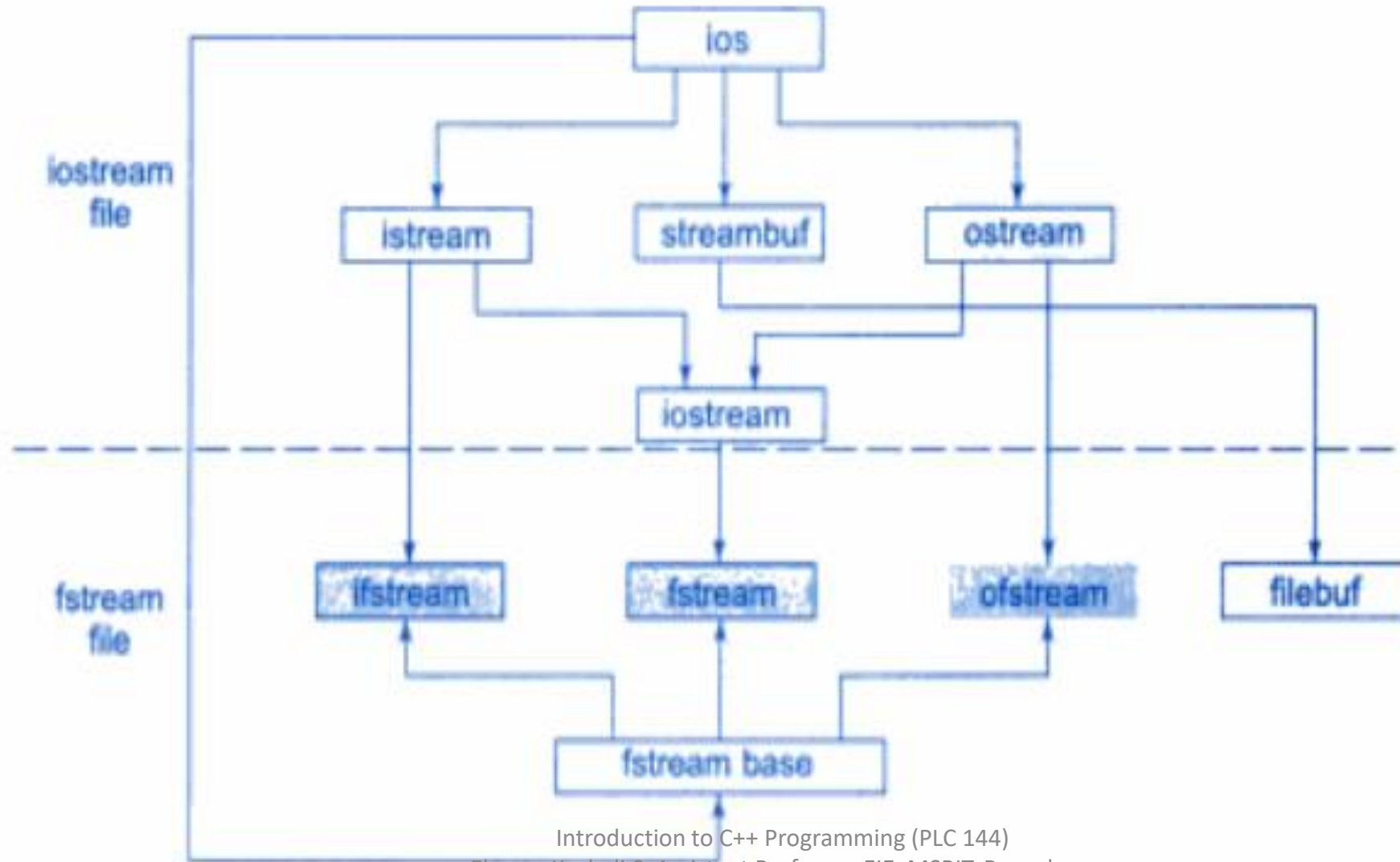
- File operations are similar to console input and output operations.
- File streams are used as interface between the programs and the files.
- The stream that supplies data to the program is called as Input Stream.(Extracts data from file).
- The stream that receives data from the program is called as Output Stream.(Inserts data to the file).



# Class Hierarchy / Classes for File Stream Operations.

- The I/O system of C++ contains set of classes that define file Handling Methods.
- These include ifstream, ofstream and fstream.
- These classes are derived from fstreambase and from iostream class.
- These classes designed to manage disk files are declared in fstream and must be included in program that uses files.

# Class Hierarchy / Classes for File Stream Operations.



# Details of file stream class

<i><b>Class</b></i>	<i><b>Contents</b></i>
<b>filebuf</b>	Its purpose is to set the file buffers to read and write. Contains <b>Openprot</b> constant used in the <b>open()</b> of file stream classes. Also contain <b>close()</b> and <b>open()</b> as members.
<b>fstreambase</b>	Provides operations common to the file streams. Serves as a base for <b>fstream</b> , <b>ifstream</b> and <b>ofstream</b> class. Contains <b>open()</b> and <b>close()</b> functions.
<b>ifstream</b>	Provides input operations. Contains <b>open()</b> with default input mode. Inherits the functions <b>get()</b> , <b>getline()</b> , <b>read()</b> , <b>seekg()</b> and <b>tellg()</b> functions from <b>istream</b> .
<b>ofstream</b>	Provides output operations. Contains <b>open()</b> with default output mode. Inherits <b>put()</b> , <b>seekp()</b> , <b>tellp()</b> , and <b>write()</b> , functions from <b>ostream</b> .
<b>fstream</b>	Provides support for simultaneous input and output operations. Contains <b>open()</b> with default input mode. Inherits all the functions from <b>istream</b> and <b>ostream</b> classes through <b>iostream</b> .

# Setting Up a Program for File Input / Output

- Before file I/O can be performed, a C++ program must be set up properly.
- File access requires the inclusion of
- `# include <fstream>`

# Input &Output with files

- C++ provides the following classes to perform output and input of characters to/from files:
- ofstream: Stream class to write on files
- ifstream: Stream class to read from files
- fstream: Stream class to both read and write from/to files.



# Path of file & How to achieve File Handling

- Before data can be written to or read from a file, the file must be opened.
- `ifstream fin("D:\\data.txt ");`
- For achieving file handling in C++ we need follow following steps
  - Naming a file
  - Opening a file
  - Reading data from file
  - Writing data into file
  - Closing a file

# Functions use in File Handling

Function	Operation
open()	To open a file
close()	To close an existing file
get()	Read a single character from a file
put()	write a single character in file.
read()	Read data from file
write()	Write data into file

# Opening of Files

- A file can be opened in 2 ways:
- Using the constructor function of class
- Using the member function `open()` of the class

Opening files using constructor:

2 Steps:

- Create a file stream object to manage stream using the appropriate class.
  - class ofstream to create output stream
  - class ifstream to create input stream

Eg:

**ofstream outfile("results")**

This statement opens the file "results" and attaches to output stream outfile.

**ifstream infile("data")**

This statement contains object infile and attaches to the file "data" for reading

Program may contain statements like

```
outfile << "Hello MSRIT"  
outfile << Var1
```

```
infile >> Var2  
infile >> name
```

- Opening files using member function open()
- Used to open multiple files with same stream object

```
ofstream outfile;  
outfile.open("results");
```

```
---
```

```
outfile.close();  
outfile.open("data");
```

```
---
```

```
outfile.close();
```

Program 1:

```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    string InputLine, OutputLine;
    ofstream EntryFile("FewLines.dat");
    cout << "Input :" << endl;
    while(true)
    {
        cin >> InputLine;

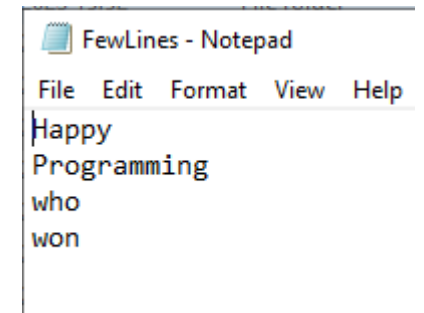
        if(InputLine == "End") break;
        EntryFile << InputLine << endl; // Writing to EntryFile
    }
    EntryFile.close();
```

```
    cout << "Output: " << endl;
        ifstream DisplayFile("FewLines.dat");
        while(!DisplayFile.eof())
        {

            DisplayFile >> OutputLine;
            cout << OutputLine << "\n";
        }
        DisplayFile.close();
    return 0;
}
```

Output:

```
Hello world!
Input :
Happy Programming
who won
End
Output:
Happy
Programming
who
won
```



- The problem of using an ifstream object in its bare form (i.e., using >>) is that it cannot work with strings containing spaces.
- If a string contains spaces, each word is counted as a separate record.
- One needs to use either get() and put() (for reading char by char) or use getline() function for reading lines with spaces
- cin.get(ch) (reads a character from cin and stores what is read in ch)
- cout.put(ch) (reads a character ch and writes to cout)

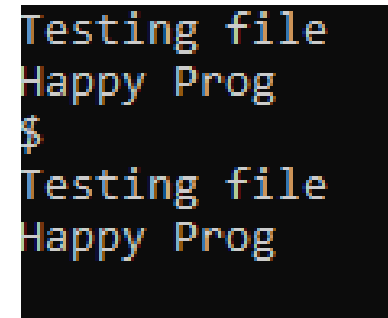
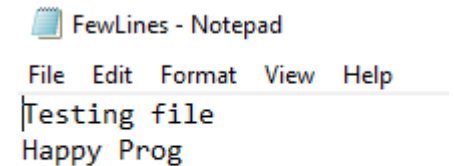
## Program 2 reads lines until \$ and displays the lines as they are back on the screen

```
#include <iostream>
#include <string>
#include <fstream>
#include <iomanip>
using namespace std;

int main()
{
    char ch;
    ofstream EntryFile("FewLines.txt");
    while(true)
    {
        cin.get(ch);
        if(ch == '$') break;
        EntryFile << ch;
    }
    EntryFile.close();
```

```
ifstream DisplayFile("FewLines.txt");
while(!DisplayFile.eof())
{
    DisplayFile >> ch;
    cout << ch;
}
DisplayFile.close();
return 0;
};
```

OUTPUT:



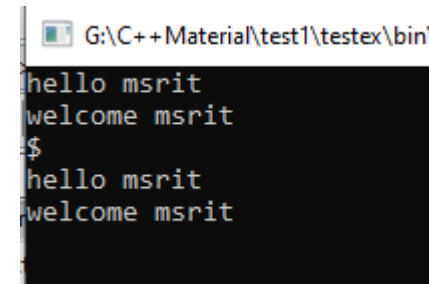


## Program 2a: reads lines until \$ and displays the lines as they are back on the screen

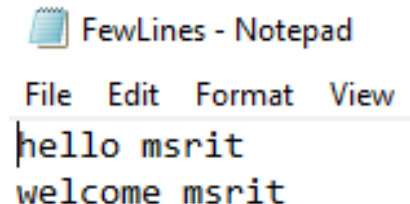
```
int main()
{
    char ch;
    ofstream EntryFile("FewLines.txt");
    while(true)
    {
        cin.get(ch);
        if(ch == '$') break;
        EntryFile.put(ch);
    }
    EntryFile.close();
    ifstream DisplayFile("FewLines.txt");
```

```
while(!DisplayFile.eof())
{
    // Do not skip white space
    DisplayFile.unsetf(ios::skipws);
    DisplayFile.get(ch);
    cout.put(ch);
}
DisplayFile.close();
return 0;
}
```

### OUTPUT



```
G:\C++Material\test1\testex\bin\
hello msrit
welcome msrit
$
hello msrit
welcome msrit
```

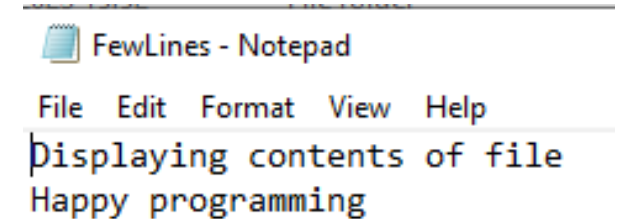


```
FewLines - Notepad
File Edit Format View
hello msrit
welcome msrit
```

Program 3 reads lines using getline and displays the lines as they are back on the screen

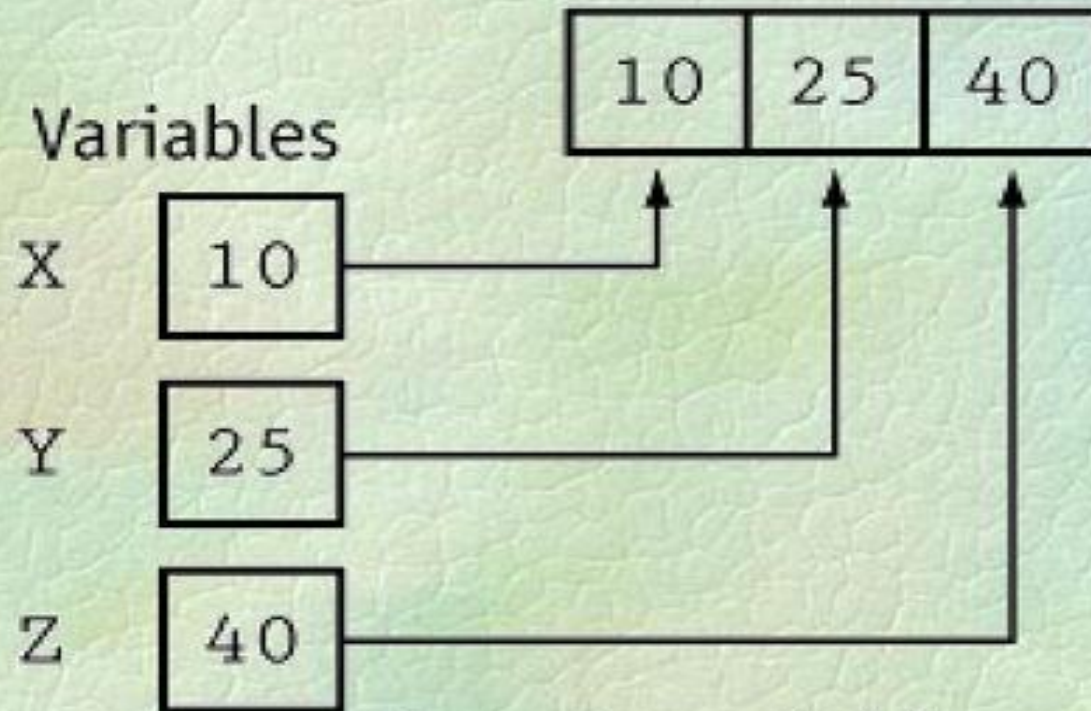
```
#include <iostream>
#include <string.h>
#include <fstream>
#include <iomanip>
using namespace std;
int main()
{
    char InputLine[80], OutputLine[80];
    ofstream EntryFile("FewLines.dat");
    while(true)
    {
        cin.getline(InputLine, 80);
        if(!strcmp(InputLine, "End")) break;
        EntryFile << InputLine << endl;
    }
    EntryFile.close();
```

```
ifstream DisplayFile("FewLines.dat");
while(!DisplayFile.eof())
{
    DisplayFile.getline(OutputLine, 80);
    cout << OutputLine << endl;
}
DisplayFile.close();
return 0;
}
```



```
Displaying contents of file
Happy programming
End
Displaying contents of file
Happy programming
```

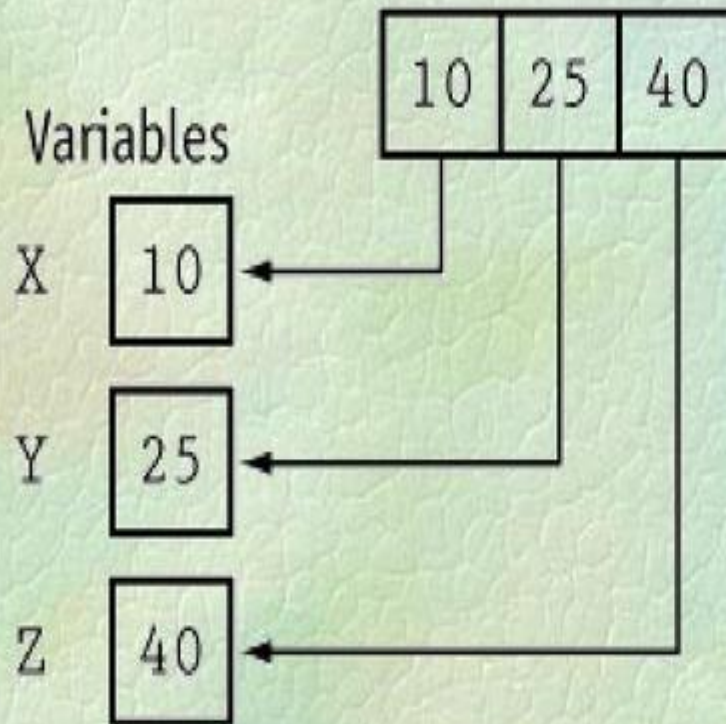
## Writing information to a file



*Data is copied from variables into the file.*



Reading information from a file



*Data is copied from the file  
into variables.*

- Program 4: Data copied from variables to file & from file to variables

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ofstream outfile("Results.txt");
    cout<<"Enter Name:";

    char name[80];
    cin>>name;

    outfile<<name<<endl;

    cout<<"Enter Age:";
    int age;
    cin>>age;

    outfile<<age<<endl;
```

```
ifstream fin("Results.txt");

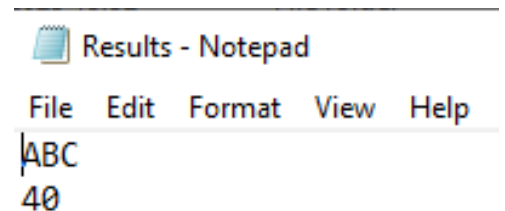
    fin>>name;
    cout<<"name:\t\t"<<name<<endl;

    fin>>age;
    cout<<"Age:\t\t"<<age<<endl;

    return 0;
}
```

OUTPUT:

```
Enter Name:ABC
Enter Age:40
name:          ABC
Age:           40
```



Results - Notepad

File Edit Format View Help

ABC

40

- Program 5: Data copied from variables to file & from file to variables

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ofstream outfile("Results.txt");
    cout<<"Enter Name:";

    char name[80];
    cin>>name;

    outfile<<"name:"<<"\t\t"<<name<<endl;

    cout<<"Enter Age:";
    int age;
    cin>>age;

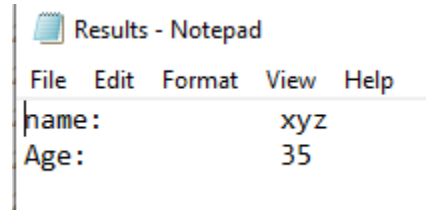
    outfile<<"Age:"<<"\t\t"<<age<<endl;
```

```
ifstream fin("Results.txt");
while(fin)
{
    fin.getline(name,80);
    cout<<name<<endl;
}

return 0;
}
```

OUTPUT:

```
Enter Name:xyz
Enter Age:35
name:           xyz
Age:            35
```



File	Edit	Format	View	Help
name: xyz				
Age: 35				

- Program 6: Write a C++ program to create a text file, check file created or not, if created write some text into the file and then read the text from the file.

```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
    char text[200];
    fstream file;
    file.open ("Results.txt", ios::out | ios::in );
    if(!file)
    {
        cout<<"Error in creating file!!!"<<endl;
        return 0;
    }

    cout<<"File created successfully."<<endl;
    cout << "Write text to be written on file." << endl;
    cin.getline(text, sizeof(text));
```

```
// Writing on file
file << text << endl;
// Reding from file
file >> text;
cout << text << endl;
//closing the file
file.close();
return 0;
}
```

OUTPUT:

```
File created successfully.
Write text to be written on file.
test file open
test file open
```

To do:

Try to open a file using file.open with name of file not physically in disk.

Try file check using file.fail()



Results - Notepad

File Edit Format View Help

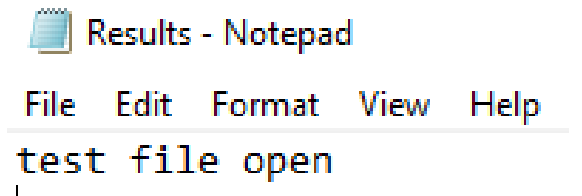
test file open

- Program 7: Write a C++ program to read the contents from a text file, count and display the number of alphabets present in it.

```
int main()
{
    ifstream fin("Results.txt");
    char ch;
    int i, c=0, sp=0;
    while(fin)
    {
        fin.get(ch);
        i=ch;
        if((i > 63 && i < 91) || (i > 96 && i < 123))
            c++;
        else
            if(ch== ' ')
                sp++;
    }
```

```
cout<<"\n No. of Characters in a File : "<<c;
cout<<"\n Space between the Words : "<<sp;
fin.close();
return 0;
}
```

OUTPUT:



```
No. of Characters in a File : 12
Space between the Words : 2
```



# Binary Files, Opening a Binary File

- These files are more useful for storing structures of information.
- A binary file can be opened using a constructor.
- For binary files, another constructor with two arguments is needed.
- The first argument is the name of the file and the second one is the file mode.

- Eg:

```
ofstream StudFile_Out;
```

```
StudFile_Out.open("MSRIT.dat", ios::out | ios::binary | ios::trunc); (Member Function)
```

```
ifstream StudFile_In("MSRIT.dat", ios::in | ios::binary); (Constructor)
```

- Note that pipes ( | ) are used in the second argument to add multiple modes

- Reading from and Writing to Binary Files:
- Two member functions for ifstream and ofstream objects are useful in reading and writing.
- writing in the file:

`OfstreamFileObject.write((char *) &<the object>, sizeof(<the same object>))`

- reading from the file

`IfStreamFileObject.read((char *) &<the object>, sizeof(<the same object>))`

- One reads the complete object from the file or writes the complete object to the file using a single read or write.
- Closing Binary Files:

`StudFile_Out.close();`

`FileObject.close()`

## Using Binary Files:

### Program 8: Writing to a binary file

```
struct student
{
    int RollNo;
    char Name[30];
    char Address[40];
};
void ReadStudent(student & TempStud)
{
    cout << "\n Enter roll no.: ";
    cin >> TempStud.RollNo;
    cout << "\n Enter name: ";
    cin >> TempStud.Name;
    cout << "\n Enter address: ";
    cin >> TempStud.Address;
    cout << "\n";
}
```

```
int main()
{
    student Student_Out;
    ofstream StudFile_Out;
    StudFile_Out.open("MSRIT.dat", ios::out | ios::binary | ios::trunc);
    if(!StudFile_Out.is_open())
        cout << "File cannot be opened \n";
    char Continue = 'y';
    do
    {
        ReadStudent(Student_Out);
        StudFile_Out.write((char*) &Student_Out, sizeof(student));
        if(StudFile_Out.fail())
            cout << "File write failed";
        cout << "Do you want to continue? (y/n): ";
        cin >> Continue;
    } while(Continue != 'n');
    StudFile_Out.close();
    return 0;
}
```

```
Enter roll no.: 123
Enter name: xyz
Enter address: wer
Do you want to continue? (y/n): y
Enter roll no.: 124
Enter name: abc
Enter address: qre
Do you want to continue? (y/n): n
```

## Using Binary Files:

### Program 9: Reading from a binary file

```
struct student
{
    int RollNo;
    char Name[30];
    char Address[40];
};
void WriteStudent(student TempStud)
{
    cout << "\n The roll no.: ";
    cout << TempStud.RollNo;
    cout << "\n The name: ";
    cout << TempStud.Name;
    cout << "\n The address: ";
    cout << TempStud.Address;
    cout << "\n";
}
```

```
int main()
{
    student Student_In;
    ifstream StudFile_In("MSRIT.dat", ios::in | ios::binary);
    while(!StudFile_In.eof())
    {
        StudFile_In.read((char*) &Student_In, sizeof(student));
        if(StudFile_In.fail())
            break;
        WriteStudent(Student_In);
    }
    StudFile_In.close();
    return 0;
}
```

```
The roll no.: 123
The name: xyz
The address: wer
```

```
The roll no.: 124
The name: abc
The address: qre
```

```
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;
```

```
class Inventory
{
    char name[30];
    int code;
    float cost;
public:
    void readdata(void);
    void writedata(void);
};
```

```
void Inventory ::readdata()
{
    cout<<"Enter Name:\t";cin>>name;
    cout<<"Enter code:\t";cin>>code;
    cout<<"Enter cost:\t";cin>>cost;
}
```

```
void Inventory ::writedata()
{
    cout<<setiosflags(ios::left)<<setw(10)<<name
        <<setiosflags(ios::right)<<setw(10)<<code
        <<setprecision(2)<<setw(10)<<cost<<endl;
}
```

```
int main()
{
    Inventory I[2],R;
    fstream file;

    file.open("STOCK.txt",ios::in|ios::out|ios::bin
        ary|ios::trunc);

    cout<<"enter the stock details"<<endl;
```

```
    for(int i=0;i<2;i++)
    {
        I[i].readdata();
        file.write((char *)&I[i],sizeof(I[i]));
    }
```

```
    file.seekg(0);

    for(int i=0;i<2;i++)
    {
        file.read((char *)&R,sizeof(R));
        R.writedata();
    }

    file.close();

    return (0);
}
```

#### OUTPUT:

```
enter the stock details
Enter Name:      as
Enter code:      001
Enter cost:      100
Enter Name:      df
Enter code:      002
Enter cost:      200
as                1      1e+002
                df      2      2e+002
```

## Using Binary Files: Program 10: Reading and Writing Class Objects

# I/O MODES

IO Modes	Effect
<code>ios :: in</code>	File opens in input mode
<code>ios :: out</code>	File opens in output mode
<code>ios :: app</code>	File opens in append mode; we can add records at the end of an existing file.
<code>ios :: ate</code>	When file is opened the file pointers move at the end of file. We can read and write anywhere in the file depending on other modes provided with this mode. The file must exist when this mode is applied. <code>ios::trunc</code> cannot be provided with this mode.
<code>ios :: trunc</code>	When the file is opened, the contents are erased
<code>ios :: noreplace</code>	Checks if the file exists; if file does not exist, the call to open fails.
<code>ios :: nocreate</code>	Checks if the file exists; if file exists, the call to open fails.
<code>ios :: binary</code>	The file is opened in binary rather than default text mode.