Lesson 4: I/O Streams

Objectives

After completing the lesson you will understand about:

- The streams, advantages of using files and about file input and output.
- The different functions in file I/O.
- The tools for formatting the output functions and manipulators
- The different character functions used in I/O and predefined character functions.
- The different file modes, file pointers and file manipulation.
- Sequential and random access files.

Structure Of The Lesson

4.1 Streams

Stream is a flow of characters or data. If the flow is into the program then it is called input stream. If the flow is out of the program it is called an output stream. cin is an input stream object connected to the keyboard. cout is an output stream object connected to the screen. These two streams are automatically available to the program through the header file <iostream.h>. However, we can define stream that come from or go to the files. These streams are called file streams.

4.1.1 Advantages Of Using Files For I-O

The keyboard input and the screen output deals with temporary data. When the program ends, the data typed in at the keyboard and the data left on the screen go away. If we want to store data permanently then we should store it in a file. The contents of the file remain until it is modified by a person or a program. If the output from the program is sent to a file it will remain there even after the program has finished running. An (input) file can be used any number of times, by any number of programmers.

4.1.2 File I-O

When the program takes input from a file, it is said to be reading from the file. When the program sends output to a file it is said to be writing to a file. In C++, a stream is a special kind of variable known as an object.

The stream objects are used to read from and write into the files. An input stream that reads from a file is called input file stream objects and output stream that writes to a file is called output file stream object. An input file stream object is created using the class "**ifstream**" and output file stream object is created using the class "**ofstream**". The two types are defined in the library with the header file "**fstream.h**". Hence a program using these two types must include "fstream.h".

ex: for stream variables
```
ifstream   instream, fin;
ofstream   outstream, fout;
```

instream, fin are ifstream objects. outstream, fout are ofstream objects.

A stream object should be connected to a file after it is declared. This is called opening of a file and it is done with a function name “open”. The syntax of opening the file is

i/o streamobject.open(“file name”);
e.g.: fin.open(“in file.dat”);

After connecting an input file stream object, we can access data from the input file using the stream object and extraction operator.
fin >> y;

An output stream is also opened in the same way.
e.g: fout.open(“outfile.dat”);

The open function in the above example will create the output file, outfile dat, if it does not exist. If the outputfile already exists, the open function discards(removes) the contents of the file. Every input and output file used in the program has two names, the external name and the internal name. The external filename is the real name of the file that is used as an argument in the “open” function. The stream object connected to this file is the internal name of the file. This internal name is used in rest of the program.Every file should be closed when the program has finished getting input or sending output to the file. Closing a file disconnects the stream from the file. A file is closed with a function called close. The close function does not require any arguments.

Syntax:

```
        i/p streamobject.close( );
        o/p streamobject.close( );
eg:-    fout.close();
        fin.close();
```

Sample program using file concept to store the data and keyboard to input the data

```
#include<iostream.h>
#include<fstream.h>
void main()
{
int x,y,z;
ofstream fout;
fout.open("ex1.dat");
cout<<"enter 3; integers:";
cin>>x>>y>>z;
fout<<"The 3 integers are: ";
fout<<x<<"  "y<<" "<<z;
fout.close();
return;
}
```

In the above program, the values for x, y and z are taken from keyboard. Let it be 3 4 5. After execution of the program,

The 3 integers are 3 4 5

are stored in the file ex1.dat. If the file ex1.dat does not exist, it is created and the information is stored. If the file ex1.dat already exists, the information in it is cleared and the new information is stored.

Sample program using file concept to store the result as well as to read the data

```
#include<iostream.h>
#include<fstream.h>
void main()
{
int x,y,z;
ifstream fin;
ofstream fout;
fin.open(ex1.dat);
fout.open("ex2.dat");
fin>>x>>y>>z;
fout<<"The 3 integers are: ";
fout<<x<<"  "y<<" "<<z;
fout.close();
return;
}
```

infile.dat
(Not changed by the program)

```
3
4
5
```

outfile.dat
(After the program is run)

```
The 3 integers are 3 4 5
```

Note: There is no output to the screen and no input from the keyboard.

The streams fin, fout discussed and predefined streams cin and cout are objects. An object is a variable that has functions as well as data associated with it. fin, fout both have function named "open" associated with them. Two sample calls of these functions, along with the declaration of objects is as follows:

```
ifstream fin;
ofstream fout;
fin.open(f1);
fout.open(f2);
```

The function named open associated with "fin" is a different from the function named open associated with object "fout". One function opens a file for input and other opens a file for output. The compiler determines to which object the open function belongs to, by looking at the name of the object that precedes the dot operator. If two objects are of the same type, they may have different values, but uses the same member function.

Eg:

```
ifstream fin, fin1;
ofstream fout, fout1;
```

A datatype, whose variables are objects are called class. Since the member functions of an object are determined by its class, these functions are called as member functions of the class.

Here, as the "open" function of ifstream is different from the "open" function of ofstream. Similarly, both ifstream and ofstream has "close" as their member function, but they close the files in different ways.

4.1.3 To Check For Successful Opening Of The File

A call to the open function can be unsuccessful for a number of reasons. If we open an input file with an external file name and if there is no file

with that name then the open function fails. When this function fails we may not get an error message and the program will do some unexpected work. Hence we should test to see whether the call to open was successful and if it is not successful the program has to end. There is a member function called fail() for each of the classes “ifstream” and “ofstream” . The “fail” function takes no arguments and returns a Boolean value. This Boolean value can be used in a repetitive or conditional statement.

A call to fail() should be placed immediately after each call to open. If the call to open fails when the function returns true otherwise it returns false. Whenever an opening fails an exit function can be called using the header file < stdlib.h>. When the exit statement is executed, the program ends immediately. Any integer value may be used with the exit statement.

Syntax:

exit(integer value);

When the exit statement will be executed the program ends immediately. Any integer value will be used as argument, but generally “1” is used for call to exit i.e., caused by an error and “0” is used in other cases.

e.g:
```
ifstream fin;
fin.open(“f1.dat”);
if fin.fail()
{
cout<<”error while file opening”;
exit(1);
}
```

We can declare the input and output file names as variables in character array. These variables can be used to read the file names as string data.

Ex: Program to store some integers in a file "infile.dat" and the result is stored in another file "outfile.dat".

```
#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>
int main()
{
int fact(int x);
char f1[20],f2[20];
ifstream fin;
ofstream fout;
cout<<"enter i/p file name";
cin >>f1;
```

```
cout<<"enter o/p file name";
cin >>f2;
fin.open(f1);
if(fin.fail())
{
cout<<"error while i/p file opening";
exit(1);
}
fout.open(f2);
if(fout.fail())
{
cout<<"error while o/p file opening";
exit(1);
}
fout<<"value \t factorial \n";
int x;
while(fin>>x)                    //satisfied if there is a next number to read
{
fout<<x<<"\t"<<fact(x)<<"\n";
}
fin.close();
fout.close();

return 0;
}
int fact(int x)
{
int f=1, I=1;
while(I<=x)
{
f=f*I;
I++;
}
return f;
}
```

output:

enter i/p file name infile.dat
enter o/p file name outfile.dat

infile.dat

5
6

outfile.dat

value	factorial
5	120
6	720

Before executing the above program, a new file (any name can be given) "infile.dat" is created and the values whose factorials to be calculated are entered and saved. When the above program is executed, the system prompts to enter the input file name. The filename used to store the input data is entered. Here it is "infile.dat" because the input data is stored here. Next, the system prompts for an output file name. An output filename has to be entered. Here, "outfile.dat" is the output file. The data from the file infile.dat is read, using the statement,

"fin>>x"

in the while loop. "fin>>x" reads the next number stored in the inputfile and the while loop is executed till there is next number to read in the file.

Note: A stream can be an argument to a function. The formal parameter passed must be a call by reference stream parameter. Depending upon the number of files used, stream parameters can be passed.

4.2 Character I/O

C++ allows to read and display character values. The character values are stored in character type variables. The extraction operator can be used to read a character of input but when extraction operator is used to skipping of blank, new line etc., are done automatically.

4.2.1 Member functions get(),put() and putback()

The member functions get() and put() can be used to perform character input and output. Every input stream(input-file stream or cin) has a member function *get* , which can be used to read one character of input. get() reads the next input character no matter if it is a blank or a newline character. The get() takes one argument which should be of type character. When the get() is called the next input character is read and passed into the argument variable.

Syntax:

Inputstream.get(char vari);

Suppose the program code is as follows:

```
char c1,c2,c3;
```

```
cin.get(c1);
cin.get(c2);
cin.get(c3);
```
and the following lines of input are read:
```
XY
ZA
```

then c1 is set to 'X', c2 is set to 'Y', c3 is set to '\n' as we press "enter" after writing the first line of input.

//Program to demonstrate get(),put() and putback()

```
#include<iostream.h>
#include<ctype.h>
void main()
{
char next;
do
{
cin.get(next);
cout.put(next);
}
while(next!='\n');
cin.putback(next);
cin.get(next);
cout.put(next);
}
```

output:

```
hello
hello
```

To read data from a file, input file stream object is placed instead of cin. Every output stream has a member function named put(). The member function put() takes one argument which should be an expression of type character. When a member function put() is called the value of its argument is output to the output stream.

Syntax:

```
outstream.put(char expression);
```

e.g.:

```
cout.put(next);
cout.put('a');
```

Note: The function putback() is a member of input stream. It takes one argument of type character and places it back into the input stream. The argument can be any expression that evaluates to a value of type character.

Syntax: Inputstream.putback(char exp);
e.g: cin.putback(next);

4.2.2 The eof() member function

Every input-file stream has a member function called eof(),that can be used to determine when all of the file is read and there is no more input left for the program. The eof() works best for the input that is text. If *fin* is a input file stream , then call to the function is written by

fin.eof().

This is a Boolean expression that can be used to control a while loop, **do-while** or **if-else** statement. This expression is *true* if the program has read past the end of input file, otherwise the above expression is not satisfied. Generally the call to this function is done with (!) symbol, as we test that we are not at the end of file.
Eg:
Suppose fin is the input file stream, then the entire contents of the file can be written to the screen with the following while-loop:

```
fin.get(next);
while(!fin.eof())
{
cout<<next;
fin.get(next);
}
```

4.2.3 Predefined Character Functions

Some predefined character functions are put in <ctype.h> header file. So, the functions in ctype.h can be used by including the include directive.

#include<ctype.h>

Some of the predefined functions in ctype.h are:

int toupper (char expr);
It returns the uppercase version of the character expression.

e.g.:

```
char c=toupper('a')
cout<<c;
Or
cout<< char(toupper('a'));
```

int tolower(char expr);
It returns the lowercase version of the character expression.

int isupper(char expr);
It returns true if the char expr is in upper case otherwise it returns false.

e.g.:

```
char c;
c='a';
if(isupper(c)
cout<< "is upper case";
else
cout<<"is lower case.";
```

int islower(char exp);
Returns true if character exp is a lower case letter otherwise returns false.

e.g:

```
char c;
c = 'a';
If(islower(c)
cout << c << " is a lower case.";
```

int isalpha (char exp);
Returns true if the character exp is a letter of alphabet otherwise returns false.

e.g.:

```
char c;
c='$';
if(isalpha(c)
cout << "is a letter";
else
cout << c<< "is not a letter";
```

int isdigit (char exp);

Returns true if the character exp is one of the digits from '0' to '9' otherwise returns false.

e.g.:
```
char c;
c='3';
If (isdigit(c)
cout<<c<< "is a digit";
```

int isspace(char exp);
It returns true if the character expression is a wide space such as blank space, \n, \t etc otherwise it returns false.

e.g.:
```
do
 {
cin.get(c);
cout.put(c);
}
while(!isspace(c);
```

4.2.4 Inheritance

When one class was derived from another class, the derived class was obtained from the previous class by adding new features.

Ex: The class of input file stream is derived from the class of all input streams by adding additional member functions such as open(), close(), fail(), eof(). The stream cin belongs to the class of all inputstream, but does not belong to the class of input file streams because cin has no member functions open(), close() etc.

If " **istream**" is used as the type for an input stream parameter then the argument corresponding to that formal parameter can be either the stream cin or an input file stream of type ifstream.
If "**ostream** " is used as the type of an output stream parameter then the argument corresponding to that formal parameter can be either the stream cout or an output file stream of type of stream.

```
#include<iostream.h>
#include<fstream.h>
void add(istream &in);
void main()
{
ifstream fin;
fin.open("file1.dat");
```

```
cout<<"data from file\n";
add(fin);
cout<<"enter 2 integers :";
add(cin);
fin.close();
}
void add(istream&in)
{int n1,n2;
in>>n1>>n2;
cout<<"sum of" <<n1<<" and " << n2 << " is  " << n1+n2 <<"\n";
}
```

output:
data from file
sum of1and2is3
enter 2 integers :4
5
sum of 4 and 5 is 9

Note: The function parameters can have default arguments that provide values for the parameters, when the corresponding argument is omitted in the call.

4.3 Formatting Output

The layout of a programs output is called the format of the output. C++ supports a number of features that could be used for formatting the output. These features include:

- Manipulators
- Ios class functions and flags.

Manipulators are operators used to format the data display. Some of the manipulators are endl, setw, set precision etc. In order to use a manipulator the header file<iomanip.h> must be included in the program. The manipulators are used with insertion operator. The “endl” manipulator, when used in output statement causes a line feed to be inserted. It works like new line character “\n”. For every manipulator there is an equivalent “ios” class function. The ios class contains a large number of member functions that would help to format the output. These

can be used to format the screen as well as format the files. To format the file the respective ofstream object should be attached with the functions.

Formatting with manipulators and class ios functions:
(Tools for i-o stream)

Manipulators	**Equivalent ios class** function
setw()	width();
eg:	eg:
float m=15	cout.width(7)
setw(m);	cout<<m.
cout<<m;	

It specifies a common field width for all the numbers and make them print right justified.

Setprecision()	precision()

It specifies the number of digits to be displayed after the decimal point. The setting is retained until is changed or reset. The output will be rounded.

e.g.:	Eg:	
cout << setpricision(2)<<5.678	cout.precision(2);	cout<<5.678;

setiosflags(ios::fixed) setf(ios::fixed, ios::floatfield)
It shows output in fixed floating point.

setiosflags(ios::showpoint) setf(ios::showpoint)

It shows decimal point in following point is floating point Output.

```
//program using manipulators
#include<iostream.h>
#include<iomanip.h>
int main()
{
double num = 67857.765;
cout<<setiosflags(ios::fixed)
<<setiosflags(ios::showpoint)
<<setprecision(1)
<<setw(12)
<<num<<"\n";
return 0;
```

```
}
output:
  67857.8
```

//program using iosclass functions

```
#include<iostream.h>
#include<iomanip.h>
int main()
{
double num = 67857.765;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);       //each function should be called
cout.precision(1);               //seperately
cout.width(12);
cout.precision(1);
cout<<num<<"\n";
return 0;
}
```

output:
67857.8

setiosflags(ios::left) setf(ios::left,ios::adjust field)
It left justifies the output.

setiosflags(ios::right) setf(ios::right,ios::adjust field)
It right justifies the output.

setiosflags ios:showpos) setf(ios::showpos)
It prefix + before positive integers.

setiosflags(ios:internal) setf(ios::internal,ios::adjust field)
It is used for padding after sign.
e.g.: + 65.74

setiosflags(ios::scientific) setf(ios::scientific,ios::float field)
It shows the o/p in exponent form.

Resetiosflags(ios::internal) unsetf(ios::internal)
The flags can be reset using the above function. The respective argument is passed to reset the option. Here the internal flag is reset. Similarly the other flags like left, fixed etc., can also be reset.

Setfill(int C); fill(int c)

It fills the field with a particular character.

4.4 Working With Files

A **file** is a collection of related data stored in a particular area of the disk. We have already studied that the I/O system of C++ handles file operation by using file streams as an interface between the programs and the files. The stream that supplies data to the program (reads from the file) is known as input stream and the stream that receives data from the program (writes to the file) is known as the output stream.

The i/p operation creates an input stream and links it with the program and the input file. The o/p operation establishes an o/p stream and links it with the program and the o/p file.

The ifstream, ofstream and fstream are the classes that define file handling methods. These are derived from fstreambase and also i/ostream. These are defined in fstream. So, fstream should be included in this file.

Class	**Content**
fstreambase	Provides operations common to the file streams. It serves as base for ifstream, ofstream, stream classes. Contains open() and close() functions.
ifstream	Provides i/p operations. Contains open() with default input mode. Inherits get(), getline(),read(),tellg(),seekg() functions from istream.
ofstream	Provides output operations. Contains open() with default output mode. Inherits put(),seekp(),tellp(),write() from ostream.
fstream	Provides support for simultaneous i/o operations. Contains open() with default i/p mode. Inherits all the functions from istream and ostream classes through iostream.

4.4.1 File Modes

open() function is used to create new files as well as to open existing files. These functions can take two arguments. The general form of function open() with two arguments is:

stream-object.open("filename",mode);

The second argument mode specifies the purpose for which the file is opened. The file modes can take one or more of the following parameters.

Parameter	Meaning
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	Opens fails if the file does not exist.
ios::noreplace	Opens files if files already exists.
ios::out	Open file for writing only.
ios::trunc	Delete the contents of the file if it already exists.

- Opening a file in ios::out mode opens it in ios::trunc mode by default.
- Using both ios::app and ios::ate , opens the file and pointer points to the end of the file. ios::app allows to add data to the end of the file only. Ios::ate permits to add or modify the existing data anywhere in the file. In both cases, a file is created if it does not exist.
- Ios::app can be used only with files capable of output.
- Creating a stream using ifstream, means input and creating using ofstream, means output.
- Fstream class does not provide a mode by default. So, the mode parameters are provided.
- The mode can combine two or more parameters using bit wise OR operator.

Ex: fout.open("data",ios::app| ios::nocreate)

4.4.2 File Pointers And Manipulators

Each file has two associated pointers known as file pointers. One of them is called input or get pointer and the other is called output or put pointer. These pointers are used to move through the files while reading or writing. 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.

- When a file is opened in i/p(read) mode, the i/p pointer is automatically set at the beginning and the file is read from the start.
- When a file is opened in o/p(write) mode, the existing contents are deleted and the o/p pointer is automatically set at the beginning to write.
- To open a file in the existing mode and add more data, the file is opened in append mode and the o/p pointer is placed at the end of the existing file for writing purpose.

4.4.3 Functions For Manipulation Of File Pointers

The file stream classes support the following functions to manage file pointers:

- Seekg() moves i/p(read or get) pointer to a specified location.
- Seekp() moves o/p(write or put) pointer to a specified location.
- Tellg() gives the current position of the get pointer.
- Tellp() gives the current position of the put pointer.

Ex: infile.seekg(10) moves the file pointer to the byte number 10. The byte number starts from zero. That is, it will point to byte number 11.

Ex: To find the size of a given file:

```
ofstream fout;
fout.open("hello",ios::app);
cout<<fout.tellp();
```

The o/p pointer is moved to the end of the file and tellp() gives the number of bytes in the file.

Format:
seekg(offset,refpos); //Moves the files get pointer
seekp(offset,refpos);//Moves the files put pointer

The parameter offset represents the number of bytes to be moved from the location specified by the parameter refpos. The refpos can take one of the following:

- ios::beg start of the file
- ios::cur current position of the pointer
- ios::end end of the file

e.g.:

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 file
fout.seekg(m,ios::beg)	Move to (m+1)th byte in the file
fout.seekg(m,ios::cur)	Move forward m bytes from the current position
Fout.seekg(-m,ios::cur)	Move backward m bytes from the current position
Fout.seekg(-m,ios::end)	Move backward m bytes from the end position

4.4.4 Sequential Input And Output Operations

Sequential access of files is to access the contents of the file from the beginning one after the other.

Ex: Accessing in tapes and disks. The operations that are performed on the file sequentially are reading and writing.

File streams support a number of member functions for performing the i/p and o/p operations on files.
put() and get()functions are designed for handling a single character at a time.

Ex:
```
fout.put(ch);
fin.get(ch);
```

write() and read() functions are designed to write and read blocks of binary data.

Formats:

```
Infile.read((char *) &V, sizeof(V));
Outfile.write((char *) &V,sizeof(V));
```

These functions take two arguments. First is the address of the variable V and the second is the length of that variable in bytes. The address of the variable has to be type cast to char*.

```
#include<iostream.h>
#include<fstream.h>
#include<iomanip.h>
class invent
{
char name[20];
int code;
float cost;
public:
void readd();
void writed();
};
void invent:: readd()
{
cout<<"Enter name:";
cin>>name;
cout<<"Enter code:";
cin>>code;
cout<<"Enter cost:";
cin>>cost;
}
void invent::writed()
{
cout<<”Name:”<<name<<"\n"<<”Code:”<<code;
cout<<”\n"<<”Cost:”<<cost<<endl;
}
void main()
{
int i;
invent item[5],item1;
fstream fil;
fil.open("STOCK.DAT",ios::in|ios::out);
cout<<”Enter the details of 3 items\n”;
for(i = 0;i < 3; i++)
{
item[i].readd();
fil.write((char*)& item[i],sizeof(item[i]));
```

```
}
fil.seekp(sizeof(item1));
item1.readd();
fil.write((char*)& item1,sizeof(item1));
fil.seekg(0);
for(i=0;i<3;i++)
{
fil.read((char*)& item[i],sizeof(item[i]));
item[i].writed();}
fil.close();
}
```

Output:

```
Enter the details of 3 items
Enter name:c++
Enter code:1
Enter cost:123
Enter name:Datastructures
Enter code:2
Enter cost:345
Enter name:DataMining
Enter code:3
Enter cost:432
Name:C++
Code:1
Cost:123
Name:Datastructures
Code:2
Cost:345
Name:DataMining
Code:3
Cost:432
```

4.4.5 Updating Random Access Files

Random access of files is the process of accessing a file randomly from any location. Ex: Access from disk files. The operations that are performed randomly are

- Read
- Write
- Update

- ✓ Display
- ✓ Modify an existing item
- ✓ Adding a new item
- ✓ Deleting an existing item

The object length can be found using sizeof() operator. The location of the desired (say m th) object can be found by multiplying object size with m. The file pointer is set to that object by using seekg() or seekp(). To find the total number of objects in a file is found by dividing filesize with the objectsize. The filesize can be found using the function tellp() or tellg().

Program to demonstrate the random access file updating

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

class invent
{
char name[10];
int code;
float cost;
public:
void readd();
void writed();
};
void invent:: readd()
{
cout<<"Enter name:";
cin>>name;
cout<<"Enter code:";
cin>>code;
cout<<"Enter cost:";
cin>>cost;
}
void invent::writed()
{
cout<<name<<"\t"<<code<<"\t"<<cost<<endl;
}
void main()
{
invent item;
fstream fil;
```

```
fil.open("STOCK1.DAT",ios::in|ios::out|ios::ate);
fil.seekg(0,ios::beg);
cout<<"Current contents of stock\n";
while(fil.read((char*)&item,sizeof(item)))
{item.writed();
}
fil.clear();

cout<<"ADD AN ITEM\n";
item.readd();
char ch;
cin.get(ch);
fil.write((char *)&item,sizeof(item));
fil.seekg(0);
cout<<"Contents of appended file\n";
while(fil.read((char *) &item,sizeof(item)))
item.writed();

int last = fil.tellg();
int n = last/sizeof(item);

cout<<"Number of objects="<<n<<endl;
cout<<"Total bytes="<<last<<endl;
cout<<"Enter object no to be updated:"<<endl;
int object;
cin>>object;
cin.get(ch);
int loc = (object-1) * sizeof(item);
if(fil.eof())
fil.clear();
fil.seekp(loc);

cout<<"Enter new values\n";
item.readd();
cin.get(ch);
fil.write((char*)&item,sizeof(item))<<flush;

fil.seekg(0);
cout<<"Contents of updated file\n";
while(fil.read((char *)&item,sizeof(item)))
{item.writed();}
```

```
fil.close();
return ;
}
```
output:
```
Enter code:1
Enter cost:123
Contents of updated file
c       1       123
Current contents of stock
c       1       123
ADD AN ITEM
Enter name:java
Enter code:2
Enter cost:432
Contents of appended file
c       1       123
java    2       432
Number of objects=2
Total bytes=32
Enter object no to be updated:
1
Enter new values
Enter name:c++
Enter code:1
Enter cost:123
Contents of updated file
c++     1       123
java    2       432
```

4.5 Summary

- We have covered about the streams, the advantages of using files and about file input and output.
- We have studied the different character member functions like get(),put(), and putback(). Also we covered the usage of eof() member function.

- We have studied in detail about different stream functions and manipulators for formatting the output.

- We have studied in detail the working of file, file modes, filepointers, file manipulation functions. The details of writing sequential files and random access files are covered.

4.6 Model Questions

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

File pointer: The input and output file pointers are to move around the file.

Stream: A flow of characters.

4.7 Model Questions

1. Explain in detail about file I/O.
2. Explain the usage of put(),get() and putback() functions.
3. Give some of the predefined character functions and explain their usage.
4. Write in detail the working of files.
5. Explain the tools for formatting the output.

4.8 References

Object-oriented programming with C++,
by E. Bala Gurusamy.

Problem solving with C++ **by Walter Savitch**

AUTHOR:

M. NIRUPAMA BHAT, MCA., M.Phil.,
Lecturer,
Dept. Of Computer Science,
JKC College, Guntur.