Unit → 8

File Handling and streams

Date.
Page No.

Part 1 → File Streams:

Stream Classes

We have been using the iostream standard library, which provides cin and cout methods for reading from standard input and writing to standard output respectively. The stream classes help to read and write from a file. For this it requires another standard C++ library called fstream, which defines three new data types as follows:-

- ifstream → This data type represents the input file stream and is used to read information from files.
- ofstream → This data type represents the output file stream and is used to create files and to write information to files.
- fstream → This data type represents the file stream generally, and has the capabilities of both ofstream and ifstream which means it can create files, write information to files, and read information from files.

To perform file processing in C++, header files <iostream> and <fstream> must be included in our C++ source file.

Binary file vs. Character file.

- A text file stores data in the form of alphabets, digits and other special symbols by storing their

ASCII values and are in a human readable format. For e.g. any file with a .txt, .C file extension. Whereas a binary file contains a sequence or collection of bytes which are not in a human readable format. For e.g. files with .exe, .mp3 file extension.

ii) A small error in a textual file can be recognized and eliminated when seen. Whereas, a small error in a binary file corrupts the file and is not easy to detect.

Console I/O Operations:-

The standard C++ library is iostream and standard input/output functions in C++ are:

i) cin
ii) cout

There are mainly two types of consol I/O operations form:

i) Unformatted consol I/O.
ii) Formatted consol I/O.

1) Unformatted I/O Operations:

In unformatted input/output operations are in unformatted. The following are operations of unformatted consol input/output operations.

a) get() → It is the method of cin object used to input a single character from keyboard. But its main property is that it allows wide spaces and newline character.

Syntax: char c=cin.get();

getline(char *buffer, int size)

Date.
Page No.

Example:

```
#include<iostream>
using namespace std;
int main(){
    char c=cin.get();
    cout<<c<<endl;
    return 0;
}
```

Output

W ← our entered char.
W ← shown by program

for understanding run program

b) put() → It is a method of cout object and it is used to print the specified character on the screen or monitor.

Syntax: cout.put(variable/character);

Example:

```
#include<iostream>
using namespace std;
int main(){
    char c=cin.get();
    cout.put(c);
    return 0;
}
```

Output

d ← entered char
d ← shown by program the cout

c) getline() → This is a method of cin object and it is used to input a string with multiple spaces.

Syntax:
```
char x[30];
cin.getline(x,30);
```

Example:
```
#include<iostream>
using namespace std;
```

c) write(char *buffer, int n)

```
int main(){
        cout<<"Enter name:";
        char c[10];
        cin.getline(c,10); //It takes 10 charcters as input;
        cout<<c<<endl;
        return 0;
        }
```

Output

Enter name:Roshan
Roshan

d) write() → It is a method of cout object. This method is used to read n character from buffer variable.

Syntax: cout.write(x,2);

Example:

```
#include<iostream>
using namespace std;
int main(){
        cout<<"Enter name:";
        char c[10];
        cin.getline(c,10);
        return 0;
        }
```

If we write 9 instead it will read only 9 character

Output:

Enter name:Roshan
Roshan.

e) cin → It is the method to take input any variable/ character / string.

Syntax: cin >> variable /character /string /;

f) cout → This method is used to print variable / string/ Character.

Syntax: cout << variable /string/character/;

Date.
Page No.

2) Formatted I/O Operations: (with ios Member functions):

In formatted I/O operations we use following functions to make output in perfect alignment. In industrial programming all the output should be perfectly formatted due to this reason C++ provides many function to convert any file into perfect aligned format. These functions are available in header file <iomanip>. iomanip refers input output manipulators.

a) width(n) → This function is used to set width of the output.
Syntax: cout<<setw(int n);
where, n is size.

b) fill(char) → This function is used to fill specified character at unused space.
Syntax: cout<<setfill('character')<<variable;

c) precision(n) → This method is used for setting floating point of the output.
Syntax: cout<<setprecision('int n')<<variable;

d) setflag(arg 1, arg 2) → This function is used for setting format flags for output.
Syntax: setiosflags (argument 1, argument 2);

e) unsetflag(arg 2) → This function is used to reset set flags for output. Syntax: resetiosflags(argument 2);

f) setbase(arg) → This function is used to set base field of the flag.
Syntax: setbase(argument);

Date.
Page No.

⊛ Formatting with Manipulators:

Manipulators	Equivalent ios functions
setw()	width()
setprecision()	precision()
setfill()	fill()
setiosflags()	setflags() or setf()
resetiosflags()	unsetflags() or unsetf()

Part → 2 (File Handling)

⊛ Opening and Closing Files:

1) Opening Files: @A file must be opened for performing any sort of operations like read, write to it. Either ofstream or fstream object may be used to open a file for writing. And ifstream object is used to open a file for reading purpose only. Following is the standard syntax for open() function, which is a member of fstream, ifstream, and ofstream objects.

```
void open(const char *filename, ios::openmode mode);
```

List of file modes in C++:

Sr.No	Mode Flag	Description
1.	ios::app	Append mode. All output to that file to be appended to the end.
2.	ios::ate	Open a file for output and move the read/write control to the end of the file.
3.	ios::in	Open a file for reading
4.	ios::out	Open a file for writing.
5.	ios::trunc	If the file already exists, its contents will be truncated before opening a file.

Additional [Not more imp (exam point of view) this additional part]

Opening of files can be achieved in following two ways:

i) Using the constructor function of the stream class

ii) Using the function open().

Using Opening of file using the function open() have been discussed already just before in @. Now we discuss opening of files using constructors in short as follows.

We know that a constructor of class initializes an object of its class when it (the object) is being created. Same way, the constructors of stream classes (ifstream, ofstream, or fstream) are used to initialize file stream objects with the filenames passed to them.

Example

```
ifstream fin("myfile", ios::in);
```

The above given statement in example will create an object fin, of input file stream which is user-defined name. After creating the ifstream object fin, the file myfile is opened and attached to the input stream, fin.

Similarly when we want a program to open (i.e, to write a file), we can perform as:

Example

```
ofstream fout("secret", ios::out);
```

This would create an output stream object named as fout and attach the file secret with it.

Date.
Page No.

2) Closing Files:

When a C++ program terminates it automatically flushes all the streams, release all the allocated memory and close all the opened files. But it is always a good practice that a programmer should close all the opened files before program termination.

Following is the standard syntax for close() function, which is a member of fstream, ifstream and ofstream objects.

```
void close();
```

⊛ Read / Write from File:

Writing to a file → While doing C++ programming, we write information to a file from our program using the stream insertion operator (<<) just as we ~~See~~ use that operator to output information to the screen. The only difference is that we use an ofstream or fstream object instead of the cout object.

Reading from a file → We read information from a file into our program using the stream extraction operator (>>) just as we use an ifstream or fstream object instead of the cin object.

Date.
Page No.

General functions used for file handling

i) open() → To create a file.
ii) close() → To close an existing file.
iii) get() → To read a single character from file.
iv) put() → To write a single character in the file.
v) read() → To read data from a file.
vi) write() → To write data into a file.

* Example program for writing and reading from file.

```
#include<iostream>
#include<fstream>
using namespace std;
int main(){
    char text[200];
    fstream file;
    file.open("example.txt", ios::out|ios::in);
    cout<<"Write text to be written on file."
    cin.getline(text, sizeof(text));
    //Writing on file.
    file<<text<<endl;
    //closing the file
    file.close();
    return 0;
}
```

⊛ File access Pointers and Their Manipulators:

Each file has 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. Each time an input or output takes place, the appropriate pointer is automatically advanced.

Functions for manipulation of File Pointers.

To move a file pointer to any other desired position inside the file, the stream classes support the following functions to manage these situations.

i) seekg() → Moves get pointer (input) to a specified location.

ii) seekp() → Moves put pointer (output) to a specified location.

iii) tellg() → Gives the current position of the get pointer.

iv) tellp() → Gives the current position of the put pointer.

Specifying the offset

'Seek' functions seekg() and seekp() can also be used with two arguments as follows:

```
seekg( offset, refposition);
seekp( offset, refposition);
```

Date.
Page No.

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 takes one of the following three constants defined in the ios class:

i) ios::beg → start of the file.
ii) ios::cur → current position of the pointer.
iii) ios::end → End of the file.

Testing Errors during File Operations:

Sometimes during file operations, errors may occur. For example, a file being opened for reading might not exist, or such as invalid operation may be performed. To check for such errors and to ensure smooth processing, C++ file streams inherit 'stream-state' members from the ios class that store the information on the status of a file that is being currently used, like eof, fail, bad, good etc.

eof() → It returns non-zero (true value) if end-of-file is encountered while reading; otherwise returns zero (false value).

fail() → It returns non-zero when an input or output operation has failed.

bad() → Returns non-zero value if an invalid operation is attempted or any uncoverable error has occured.

good() → Returns non-zero if no errors has occured. This means, all the above discussed functions used in program are false.

Imp ⑧ **Stream Operator Overloading (Overloading extraction and insertion operators):**

In C++, stream insertion operator "<<" is used for output and extraction operator ">>" is used for input. We must know following things before we ~~want~~ start overloading these operators.

i) cout is an object of ostream class and cin is an object of istream class.

ii) These operators must be overloaded as a global function. And if we want to allow them to access private data members of class, we must make them friend.

⑧ Why these operators must be overloaded as global?

Ans In operator overloading, if an operator is overloaded as member, then it must be a member of the object on left side of the operator. For example consider the statement "ob1 + ob2" (let ob1 and ob2 be objects of two different classes). To make this statement compile, we must overload '+' in class of 'ob1' or make '+' a global function.

The operators '<<' and '>>' are called like 'cout << ob1' and 'cin >> ob1'. So, if we want to make them a member method, then they must be made members of ostream and istream classes, which is not a good option most of the time. Therefore, these operators are overloaded as global functions with two parameters, cout and object of user defined class.

Date.
Page No.

Following is complete C++ program to demonstrate overloading of extraction and insertion operators.

```
#include<iostream>
using namespace std;
class Complex{
    private:
        int real, imag;
    public:
        Complex(int r=0, int i=0){
            real=r;
            imag=i;
        }
    friend ostream &operator<<(ostream &out, const
                Complex &c);
    friend istream &operator>>(istream &in, Complex &c);

};
ostream &operator<<(ostream &out, const Complex &c)
{
    out<<c.real;
    out<<"+i"<<c.imag<<endl;
    return out;
}

istream &operator>>(istream &in, Complex &c)
{
    cout<<"Enter Real part";
    in>>c.real;
    cout<<"Enter imaginary part";
    in>>c.imag;
    return in;
}
```

Date.
Page No.

```
int main(){
    Complex c1;
    cin >> c1;
    cout << "The complex object is";
    cout << c1;
    return 0;
}
```

Output

Enter Real Part 10
Enter Imagenory Part 20
The complex object is 10+i20