Subject :- Object Oriented Programming

Unit 4

Files & Streams

What are files and streams?

1.Files are used to store data permanently.

2. A stream is an abstraction that represents a device on which input and output operations are performed.

3. A stream can basically be represented as a source or destination of characters of indefinite length.

Three file handling data types :

ofstream : This data type represents the output file stream and is used to create files and to write information to files.

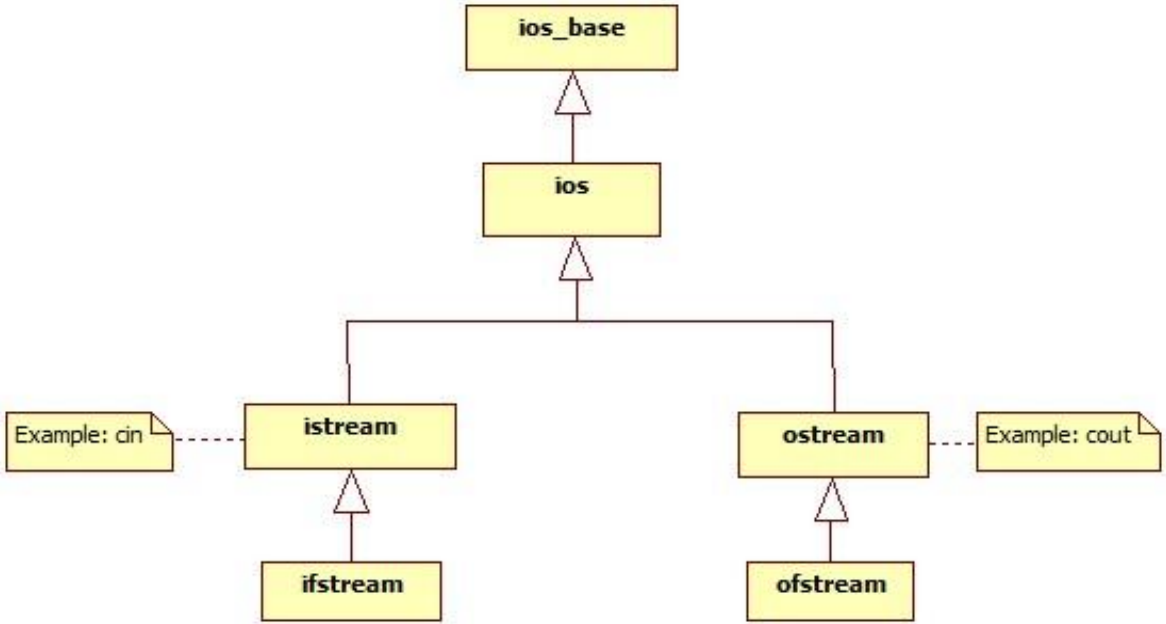
ifstream : This data type represents the input file stream and is used to read information from files.

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.

The Stream Class Hierarchy

Subject :- Object Oriented Programming

Files & Streams



The Stream Class Hierarchy

Data Type	Description
ofstream	This data type represents the output file stream and is used to create files and to write information to files.
ifstream	This data type represents the input file stream and is used to read information from 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.

The Stream Class Hierarchy

Subject :- Object Oriented Programming

Unit 4

Stream Classes

The ifstream Class

- An ***ifstream*** is an ***input file stream***, i.e. a stream of data used for reading input from a file.
- Because an ifstream is an istream, anything you can do to an istream you can also do the same way to an ifstream.
    - In particular, cin is an example of an istream, so anything that you can do with cin you can also do with any ifstream.
- The use of ifstreams (and ofstreams) requires the inclusion of the fstream header:

```
#include <fstream>
```

The Stream Class Hierarchy

Subject :- Object Oriented Programming

Unit 4

Stream Classes

The ifstream Class

Before you can use an ifstream, however, you must create a variable of type **ifstream** and connect it to a particular input file.

- This can be done in a single step, such as:

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

- Or you can create the ifstream and open the file in separate steps:

```
ifstream fin;
fin.open( "inputFile.txt" );
```

The Stream Class Hierarchy

Subject :- Object Oriented Programming

Unit 4

Stream Classes

The ifstream Class

- Before you use a newly opened file, you should always check to make sure the file opened properly. Every stream object has a fail( ) method that returns "true" if the stream is in a failed state, or "false" otherwise:

```
if( fin.fail( ) ) {
    cerr << "Error - Failed to open " << filename << endl;
    exit( -1 );  // Or use a loop to ask for a different file name.
}
```

- Once you have created an ifstream and connected it to an open file, you read data out of the file the same way that you read from cin:

```
fin >> xMin;
```

The Stream Class Hierarchy

Subject :- Object Oriented Programming

Unit 4

Stream Classes

The ifstream Class

After you are completely done using a stream,you should always close it to prevent possible corruption.

- This is especially true for output files, i.e. ofstreams.

  fin.close( );

- After you have closed a stream, you can re-open it connected to a different file if you wish. (I.e. you can reuse the stream variable.)

The Stream Class Hierarchy

Subject :- Object Oriented Programming

Unit 4

Stream Classes

Ofstream Class

- An ***ofstream*** is an ***output file stream***, and works exactly like ifstreams, except for output instead of input.
- Once an ofstream is created, opened, and checked for no failures, you use it just like cout:

```
ofstream fout( "outputFile.txt" );

fout << "The minimum oxygen percentage is " << minO2 << endl;
```

Subject :- Object Oriented Programming

Unit 4

Stream Classes

Ofstream Class

- One of the key issues when reading input data files is knowing how much data to read, and when to stop read data. There are three commonly used techniques, as shown below. (Which can also be used when reading fr the keyboard.)
  1. **Specified Number of Records**
     - One of the easiest ways is to first read in a number indicating how many data items to read in, and then read in that many data items:

```
int nData;
 double x, y, z;
 fin >> nData;
 for( int i = 0; i < nData; i++ ) {
    fin >> x >> y >> z;
    // Do something with x, y, z

 } for loop reading input data
```

     - **The difficulty with this method is that the number of data items present must be know**

2. Another commonly used method is to look for a special value (combination) as a trigger to stop reading da

```
double x, y;
    while( true ) {
fin >> x >> y;
if( x = = 0.0 && y = = 0.0 )
    break;
// Do something with x and y
} // while loop reading input data
```

1.
   - The difficulty with this method is that the sentinel value must be carefully chosen so as not to be possible valid data.

Subject :- Object Oriented Programming

Unit 4

Stream Classes

Ofstream Class

3. Detect End of File

i. If you know that the data you are reading goes all the way to the end of the file (i.e. there is no other data in the file after the data you are reading), then you can just keep on reading data until you detect that the end of the file has been reached.

ii. All istreams (and ifstreams) have a method, eof( ), that returns a boolean value of true AFTER an attempt has been made to read past the end of the file, and false otherwise.

iii. Because the true value isn't set until AFTER you have gone too far, it is important to: (1) read some data first, then (2) check to see if you've gone past the end of the file, and finally (3) use the data only after you have verified that the reading succeeded. Note carefully in the following code that the check for the end of file always occurs AFTER reading the data and BEFORE using the data that was read:

```
 double x, y, z;

fin >> x >> y >> z;

 while ( !fin.eof( ) ) {

    // Do something with x, y, z

          // Read in the new data for the next loop iteration.

          fin >> x >> y >> z;

        } // while loop reading until the end of file
```

File Mode Parameters

PARAMETER MEANING

Each one of the open() member functions of the classes ofstream, ifstream and fstream has a default mode that is used if the file is opened without a second argument:

class	default mode parameter
ofstream	ios::out
ifstream	ios::in
fstream	ios::in | ios::out

File Mode Parameters

PARAMETER	MEANING
ios::in	Open for input operations.
ios::out	Open for output operations
ios::binary	Open in binary mode.
ios::ate	Set the initial position at the end of the file. If this flag is not set to any value, the initial position is the beginning of the file.
ios::app	All output operations are performed at the end of the file, appending the content to the current content of the file. This flag can only be used in streams open for output-only operations.
ios::trunc	If the file opened for output operations already existed before, its previous content is deleted and replaced by the new one.

File Mode Parameters

PARAMETER MEANING

All these flags can be combined using the bitwise operator OR (|). For example, if we want to open the file example.bin in binary mode to add data we could do it by the following call to member function open():

```
ofstream myfile;
myfile.open ("example.bin", ios::out | ios::app | ios::binary);
```

Subject :- Object Oriented Programming

Unit 4

File Pointers

In C++ we have a get pointer and put pointer for getting (Reading) data from a file and putting data on a file (Writing) respectively.

- seekg :- seekg is used to get the pointer to desired location with respect to reference point
  Syntax:- filepointer.seekg(number of bytes,reference points);
  Ex. fin.seekg(10,ios::beg);

- tellg :- tellg is used to get to know where the get pointer is in the file.
  Syntax:- filepointer.tellg();
  Ex. int pos= fin.tellg();

Subject :- Object Oriented Programming

Unit 4

File Pointers

- seekp:- seekp is used to move the put pointer to desired location with respect to reference point

  Syntax:- filepointer.seekp(number of bytes,reference points);

  Ex. fout.seekp(10,ios::beg);

- tellp :- tellp is used to get to know where the put pointer is in the file.

  Syntax:- filepointer.tellp();

  Ex. int pos= fin.tellp();

C++ Error Handling During File Operation

Sometimes during file operations, errors may also creep in.

For example, a **file being opened for reading might not exist**. Or a **file name used for a new file may already exist**. Or a**n attempt could be made to read past the end-of-file.**

Or such as invalid operation may be performed. There might not be enough space in the disk for storing data.

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. The current state of the I/O system is held in an integer, in which the following flags are encoded :

Subject :- Object Oriented Programming

Unit 4

C++ Error Handling During File Operation

Name	Meaning
eofbit	1 when end-of-file is encountered, 0 otherwise.
failbit	1 when a non-fatal I/O error has occurred, 0 otherwise
badbit	1 when a fatal I/O error has occurred, 0 otherwise
goodbit	0 value

Subject :- Object Oriented Programming

Unit 4

C++ Error Handling Functions

There are several error handling functions supported by class ios that help you read and process the status recorded in a file stream. Following table lists these error handling functions and their meaning :

Function	Meaning
int bad()	Returns a non-zero value if an invalid operation is attempted or any unrecoverable error has occurred. However, if it is zero (false value), it may be possible to recover from any other error reported and continue operations.
int eof()	Returns non-zero (true value) if end-of-file is encountered while reading; otherwise returns zero (false value).
int fail()	Returns non-zero (true) when an input or output operation has failed.
int good()	Returns non-zero (true) if no error has occurred. This means, all the above functions are false. For example, if fin.good() is true, everything is okay with the stream named as fin and we can proceed to perform I/O operations. When it returns zero, no further operations can be carried out.
clear()	Resets the error state so that further operations can be attempted.

Subject :- Object Oriented Programming

Unit 4

C++ Error Handling Functions

The above functions can be summarized as eof() returns true if eofbit is set; bad() returns true if badbit is set. The fail() function returns true if failbit is set; the good() returns true there are no errors. Otherwise, they return false.

These functions may be used in the appropriate places in a program to locate the status of a file stream and thereby take the necessary corrective measures.

C++ Error Handling Functions

```
ifstream fin;
fin.open("master", ios::in);
while(!fin.fail())
{
        :       // process the file
}
if(fin.eof())
{
        :       // terminate the program
}
else if(fin.bad())
{
        :       // report fatal error
}
else
{
        fin.clear();    // clear error-state flags
        :
}
```

Subject :- Object Oriented Programming

Unit 4

CPP Command Line Arguments

The most important function of C/C++ is main() function. It is mostly defined with a return type of int and without parameters `int main() { /* ... */ }`

We can also give command-line arguments in C and C++. Command-line arguments are given after the name of the program in command-line shell of Operating Systems.

To pass command line arguments, we typically define **main() with two arguments** : first argument is the number of **command line arguments** and **second is list of command-line arguments.**

```
int main(int argc, char *argv[]) { /* ... */ }
```

- **argc (ARGument Count)** is int and stores number of command-line arguments passed by the user including the name of the program. So if we pass a value to a program, value of argc would be 2 (one for argument and one for program name)
- The value of argc should be non negative.
- **argv(ARGument Vector)** is array of character pointers listing all the arguments.
- If argc is greater than zero,the array elements from argv[0] to argv[argc-1] will contain pointers to strings.
- Argv[0] is the name of the program , After that till argv[argc-1] every element is command -line arguments.

- **argc (ARGument Count)** is int and stores number of command-line arguments passed by the user including the name of the program. So if we pass a value to a program, value of argc would be 2 (one for argument and one for program name)
- The value of argc should be non negative.
- **argv(ARGument Vector)** is array of character pointers listing all the arguments.
- If argc is greater than zero,the array elements from argv[0] to argv[argc-1] will contain pointers to strings.
- Argv[0] is the name of the program , After that till argv[argc-1] every element is command -line arguments.

```
// Name of program mainreturn.cpp
#include <iostream>
using namespace std;

int main(int argc, char** argv)
{
    cout << "You have entered " << argc
         << " arguments:" << "\n";

    for (int i = 0; i < argc; ++i)
        cout << argv[i] << "\n";

    return 0;
}
```