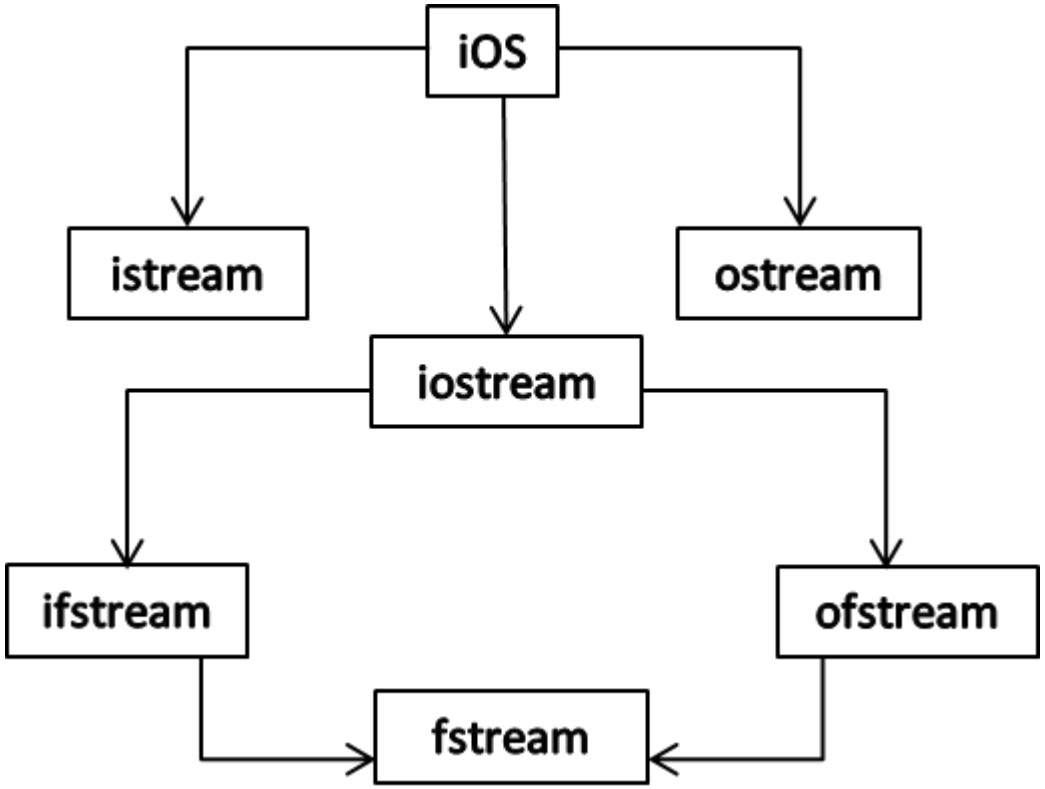
Unit 8:
File Handling and Streams

8.1 Stream Class Hierarchy

- Stream: general name given to flow of data. In C++ it is represented by an object of the particular class.
- Stream class hierarchy(partial):



- iostream provides the cin and cout methods for standard input and output. We use the ‘fstream’ to work with the files that provides following classes:
  - ofstream: this data type is used to represent the output file stream and is used to create files and to write information to the files.
  - ifstream: it represents the input file stream and is used to read information from files.
  - Fstream: it represents the general file stream that provides capabilities of both ofstream and ifstream. i.e. it can be used to create, write to files as well as to read information from files.
  - The <iostream> and <fstream> header files are must in file handling programs.

Binary File vs Character File

- A character file/text file stores data in form of alphanumeric characters and symbols in human readable format wheares a binary file stores the files in sequence of bytes.
- Files have extensions such as txt, cpp etc. are text files wheares files such as .exe, .mp3 etc. are binary files.

8.2 Working with standard I/O operations

- The I/O operations can be classified as:
  - Unformatted Input/Output: no changes in the format of the input and outputs.
  - Formatted Input/Output: the format of input/output are modified to change the appearance of data.
- Unformatted I/O Operations:
  - get(), getline(), read(), put(), write(),

8.2.1: Unformatted I/O operations:get()

- get(): method of cin object used to input a single character from keyboard.
  - Syntax: char c = cin.get()

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

getline(char *buffer, int size)

- getline(): method of cin object used to input series of characters.
  - Syntax: cin.getline(char *buffer, int size)

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

read(char *buffer, int size)

- read(): method of cin object used to input exact ‘n’ characters.
  - Syntax: cin.read(char *buffer, int size)

```
#include<iostream>
using namespace std;
int main(){
    char c[10];
    cin.read(c,10);
    cout<<c<<endl;
    return 0;
}
```

put(char c)

- cout object method used to output a single character to console
- Syntax: cout.put(char c)

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

write(char *buffer, int size)

- cout object that is used to write sequence of characters.

```
#include<iostream>
using namespace std;
int main(){
    char c[10];
    cin.read(c,10);
    cout.write(c,8);
    return 0;
}
```

8.2.2 Formatted I/O with ios member functions

- Formatting means altering the appearance of the I/O. formatting is necessary to present the data in perfect way.
- It can be done by using the
  - ios member functions or
  - ios manipulators via <iomanip> header
- The following member functions are used:

- width(int w): set the current width of the output.
    - Syntax: cout.width(15)
- Fill(char c): used to set the fill character, by default fill character is space.
    - Syntax: cout.fill(‘*’)
- precision(int p): used to set the precision (number of digits displayed) for floating point number.
    - Syntax: cout.precision(4)
- Setf(flags): set specific formatting flags
    - Syntax: setf(ios::flags) where flags can be : left, showbase, showpos, dec, oct etc.

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

using namespace std;
int main()
{
    cout << "Example for formatted IO using ios member functions" << endl;

    cout << "Default: " << endl;
    cout << 123 << endl;

    cout << "width(15): " << endl;
    cout.width(15);
    cout << 123 << endl;

    cout << "width(15) and fill('*'): " << endl;
    cout.width(15);
    cout.fill('*');
    cout << 123 << endl;

    cout.precision(3);
    cout << "precision(3) ---> " << 123.4567890 << endl;
    cout << "precision(3) ---> " << 9.876543210 << endl;

    cout << "setf(showpos): " << endl;
    cout.setf(ios::showpos);
    cout << 123 << endl;

    cout << "unsetf(showpos): " << endl;
    cout.unsetf(ios::showpos);
    cout << 123 << endl;


    return 0;
}
```

```
Example for formatted IO using ios member functions
Default:
123
width(15):
            123
width(15) and fill('*'):
***********123
precision(3) ---> 123
precision(3) ---> 9.88
setf(showpos):
+123
unsetf(showpos):
123

Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

8.2.3 Formatting with Manipulators

- The ‘iomanip.h’ header file contains multiple manipulators that can be used to format I/O data. Some manipulators are:

TABLE 12.3 `ios` Manipulators with Arguments

Manipulator	*Argument*	*Purpose*
`setw()`	field width (`int`)	Set field width for output
`setfill()`	fill character (`int`)	Set fill character for output (default is a space)
`setprecision()`	precision (`int`)	Set precision (number of digits displayed)
`setiosflags()`	formatting flags (`long`)	Set specified flags
`resetiosflags()`	formatting flags (`long`)	Clear specified flags

TABLE 12.2 No-Argument `ios` Manipulators

Manipulator	*Purpose*
`ws`	Turn on whitespace skipping on input
`dec`	Convert to decimal
`oct`	Convert to octal
`hex`	Convert to hexadecimal
`endl`	Insert newline and flush the output stream
`ends`	Insert null character to terminate an output string
`flush`	Flush the output stream

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

void line() {
    cout << "------------------------------" << endl;
}

int main()
{
    cout << "Example for formatted IO" << endl;
    line();
    cout << "setw(10): " << endl;
    cout << setw(10) << 99 << endl;
    line();
    cout << "setw(10) and setfill('*'): " << endl;
    cout << setw(10) << setfill('*') << 99 << endl;
    line();
    cout << "setprecision(5): " << endl;
    cout << setprecision(5) << 123.4567890 << endl;
    line();
    cout << "showpos: " << endl;
    cout << showpos << 999 << endl;
    line();
    cout << "hex: " << endl;
    cout << hex << 100 << endl;
    line();
    cout << "hex and showbase: " << endl;
    cout << showbase << hex << 100 << endl;
    line();

    return 0;
}
```

```
Example for formatted IO
------------------------------
setw(10):
        99
------------------------------
setw(10) and setfill('*'):
********99
------------------------------
setprecision(5):
123.46
------------------------------
showpos:
+999
------------------------------
hex:
64
------------------------------
hex and showbase:
0x64
------------------------------
```

8.3 Stream Operator Overloading

- The insertion operator ‘<<’ is used for output and the extraction operator ‘>>’ is used for input.
- If an operator is overloaded as a member, then it must be a member of the object on left side of the operator.
  - E.g. If obj1 and obj2 are objects of different classes then if ‘obj1 + obj2’ is called the ‘+’ operator must be overloaded in the class of obj1.
- Hence, to overload stream operators the functions need two parameters
  - Object of istream or ostream (cout/cin)
  - Object of user defined class
- The function must be global and friend (if need to access private members)

```cpp
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;
}

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

Example 2

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

class Person
{
private:
    string name;
    int age;
public:
    Person( )
    {  }
    friend ostream & operator << (ostream &out, const Person &p);
    friend istream & operator >> (istream &in,  Person &c);
};

ostream & operator << (ostream &out, const Person &p)
{
    out <<"Name: "<<p.name<<endl;
    out << "Age: " << p.age << endl;
    return out;
}
```

```
istream & operator >> (istream &in,  Person &p)
{
    cout << "Enter Name ";
    in >> p.name;
    cout << "Enter Age ";
    in >> p.age;
    return in;
}

int main()
{
   Person p1;
   cin >> p1;
   cout << "The person detail is:"<<endl;
   cout << p1;
   return 0;
}
```

8.4 File Input/Output with Streams

- C++ provides the following stream classes to perform I/O operation to/from files.
  - ofstream: stream class to write on files
  - ifstream: stream class to read files
  - Fstream: stream class to both read and write from/to files.

Writing to file using ofstream

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

int main () {
  ofstream myfile("example.txt");
  myfile << "Hello, This is a test line.\n";
  myfile<<"This is test line 2";
  myfile.close();
  return 0;
}
```

Reading from file using ifstream

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

int main () {
  string line;
  ifstream myfile ("example.txt");
  if (myfile.is_open())
  {
    while ( getline (myfile,line) )
    {
      cout << line << '\n';
    }
    myfile.close();
  }

  else cout << "Unable to open file";

  return 0;
}
```

8.5 Opening and Closing Files: 8.5.1 Open a file

- The first operation generally performed on an object of streams is to associate it with a real file. This process is known as opening a file.
- Syntax: open (filename, mode);
- Where filename is the name of file to be opened and mode is optional parameter that can have following values:

Modes of opening files

ios::in	Open for input operations.
ios::out	Open for output operations.
ios::ate	Set the initial position at the end of the file. If this flag is not set, 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.
ios::trunc	If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.

- The modes can be combined using bitwise OR '|'

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

8.5.2 Closing a file

- After the I/O operations are done, the file must be closed to preserve the resources. To close the file we use the stream's member function close.
- Syntax: myfile.close()

Example: Opening a file using open() and closing using close()

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

int main () {
  ofstream myfile;
  myfile.open ("example.txt");
  myfile << "Hello, This is a test line.\n";
  myfile<<"This is test line 2";
  myfile.close();
  return 0;
}
```

Example 2

```
#include <iostream>
#include <fstream>
using namespace std;
char c[50];

int main () {
  ofstream myfile("example.txt", ios::app|ios::ate);
  cout<<"Enter a line"<<endl;
  cin.getline(c,50);
  myfile<< c;
  myfile.close();
  return 0;
}
```

8.6 Read and Write from files: put(), get(), read(), write()

- The stream methods put(), get(), read() and write() can be used to perform several I/O operations on the file.
- See the examples.

Example: Create a file

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


int main () {
  fstream myfile("test.txt", ios::app|ios::ate);
  myfile<<"This is line 1\n This is next line 2\n this is line 3";
  myfile.close();
  return 0;
}
```

Example of get() and put() :create copy of existing file

```
#include <iostream>
#include <fstream>
using namespace std;
char ch;

int main () {
   ifstream file1("example.txt");
   ofstream myfile("example_copy.txt");
   while (file1){
         file1.get(ch);
         myfile.put(ch);
   }
     file1.close();
     myfile.close();
   return 0;
}
```

Example of read() and write() :create copy of existing file

```
#include <iostream>
#include <fstream>
using namespace std;
char ch[500];
int main () {
  ifstream file1("example.txt");
  ofstream myfile("example_copy.txt");
  while (file1){
        file1.read(ch,500);
        myfile.write(ch, 500);
  }
    file1.close();
    myfile.close();
  return 0;
}
```

8.7 File Access Pointers and their Manipulators

- Each file has two associated pointer known as the file pointers. One is the input pointer (get pointer) and the other is called the output pointer (put pointer).
- The input pointer is used modify the reading position of a file while the output pointer is used to specify the writing position to a file.
- The following functions are used:

- **tellg() and tellp():** these two member functions have no parameters. They are used to return the position of the get and put pointers respectively.

```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{
    ifstream fin;
    char ch;
    fin.open("manoj.text", ios::binary);

    int pos;
    pos = fin.tellg();
    cout <<"Initial pos: "<< pos<<endl;

    fin>>ch;
    pos = fin.tellg();
    cout <<"Pos after 1st fin>>ch: "<< pos<<endl;

    fin >> ch;
    pos = fin.tellg();
    cout <<"Pos after 2nd fin>>ch: "<<pos<<endl;

    return 0;
}
```

```
Initial pos: 0
Pos after 1st fin>>ch: 1
Pos after 2nd fin>>ch: 2
```

- seekg() and seekp(): These functions are used to change the get and put positions respectively. They can take two forms.
  - seekg(position) / seekp(position)
  - seekg(offset, direction)/ seekp(offset, direction)
  - Where offset is the starting value and the direction can take any of the three values:
    - ios::beg offset counted from the beginning of the stream
    - ios::cur offset counted from the current position
    - ios::end offset counted from the end of the stream

Example: finding the size of file using tellg() and seekg()

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

int main () {
  streampos begin,end;
  ifstream myfile ("test.txt");
  begin = myfile.tellg();
  myfile.seekg (0, ios::end);
  end = myfile.tellg();
  myfile.close();
  cout << "size is: " << (end-begin) << " bytes.\n";
  return 0;
}
```

8.7 Testing Errors during File Operations

- While working with streams, errors are common due to many reasons.
  - File may not exist
  - Position might be out of index
  - Operation may be invalid (writing to file opened as read)
- The following member functions can be used with streams to check for specific errors such that all of them return a bool value.

- bad(): It returns a non-zero (true) value if an invalid operation is attempted or an unrecoverable error has occurred.
- fail(): returns true if a reading or writing operation fails. For example trying to write to file that is not open.
- eof(): returns true if a file is open for reading has reached the end.
- good(): returns true if no error has occurred. i.e. all the above members are false.

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

using namespace std;

int main()
{
    fstream file;
    file.open("manoj.text", ios::out);

    string data;

    file >> data;

    if(file.bad()|| file.fail()){
        cout << "Operation not success!!!" << endl;
        cout << "Status of the badbit: " << file.bad() << endl;
        cout<<"Status of the failbit: "<<file.fail() << endl;
    }
    else {
        cout << "Data read from file - " << data << endl;
    }

    return 0;
}
```

```
Operation not success!!!
Status of the badbit: 0
Status of the failbit: 1
```

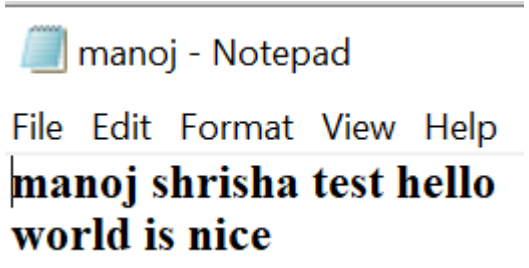
```cpp
#include <iostream>
#include <fstream>
#include <string>
#include<iomanip>
using namespace std;

int main () {
  string line;
  ifstream myfile ("manoj.txt");

    while (!myfile.eof()){
           myfile>>line;

      cout << "Data read = "<<line<<setw(20)<<"EOF bit =" <<myfile.eof()<<'\n';
    }
    myfile.close();

  return 0;
}
```




```
Data read = manoj              EOF bit =0
Data read = shrisha              EOF bit =0
Data read = test             EOF bit =0
Data read = hello             EOF bit =0
Data read = world             EOF bit =0
Data read = is            EOF bit =0
Data read = nice              EOF bit =1
```

End of Unit 8