AMBITION GURU

Object-Oriented programming in C++ (CSIT 202)

II semester, BScCSIT

Compiled by Ankit Bhattarai
Ambition Guru College

Syllabus

AMBITION GURU

Unit	Contents	Hours	Remarks
1.	Introduction to C++ and OOP	3	
2.	Classes and objects	7	
3.	Operator Overloading & Type Conversion	7	
4.	Inheritance	7	
5.	Polymorphism and Virtual Functions	7	
6.	Templates and Exception Handling	7	
7.	**I/O Stream**	**8**	

Practical Works

Credit hours : 3

Unit 7
File Handling in C++
(7 hrs.)

File streams (ifstream, ofstream, fstream), Reading from and writing to files/console, File modes, Binary and text file operations, command line arguments, file pointer and their manipulator: specifying the position, specifying the objects, tellg(), seekg(), tellp(), seekp().

Unit 7: Part 1

Input/output: Stream based input/output, input/output class hierarchy

Testing Stream Errors

Input/output: Stream based input/output, input/output class hierarchy

AMBITION GURU

C++ stream:

- In C++, input and output are performed using predefined stream objects:
  - cin with >> operator (input)
  - cout with << operator (output)
- A stream is a flow of data (sequence of bytes).
  - Input stream → source that provides data to the program.
  - Output stream → destination that receives data from the program.
- Streams are used for both console I/O and file I/O, offering multiple ways to accept input and present output.

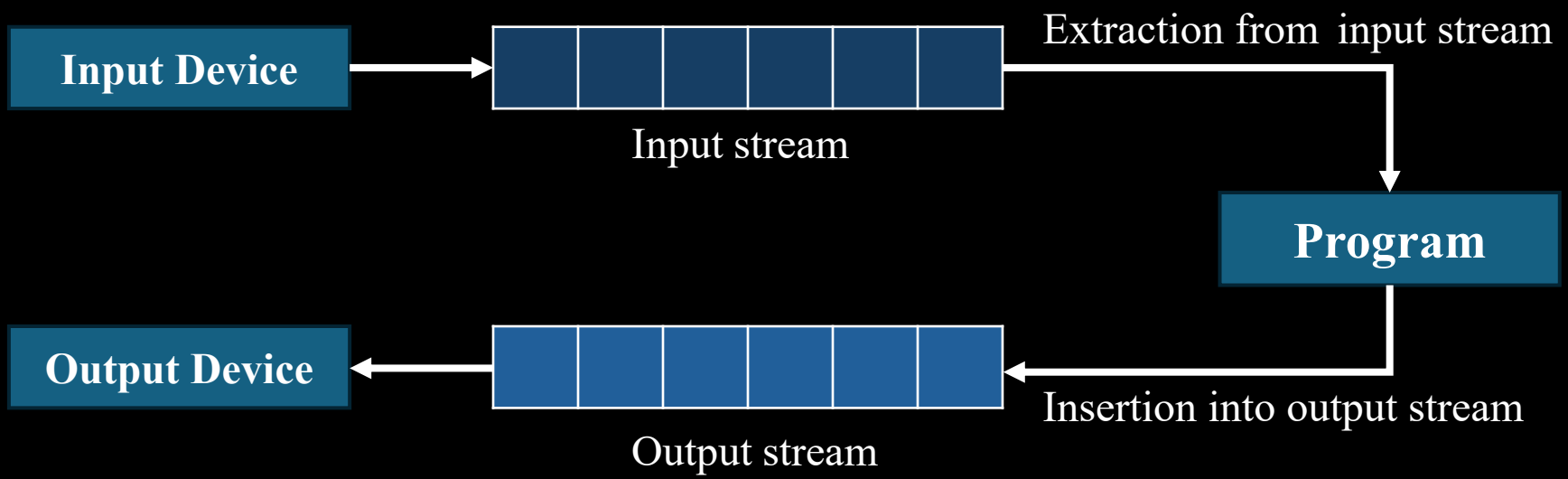


Figure: Input and Output Stream

Stream class hierarchy for console input/output

- C++ I/O system contains a *hierarchy of classes that are used to define various streams to deal with both the console and disk files*. These classes are called *stream classes*.
- These classes are declared in the header file iostream and hence should be included in all the programs that communicate with the console unit.
- Following figure shows the hierarchy of the stream classes used for input/output operations with the console unit.

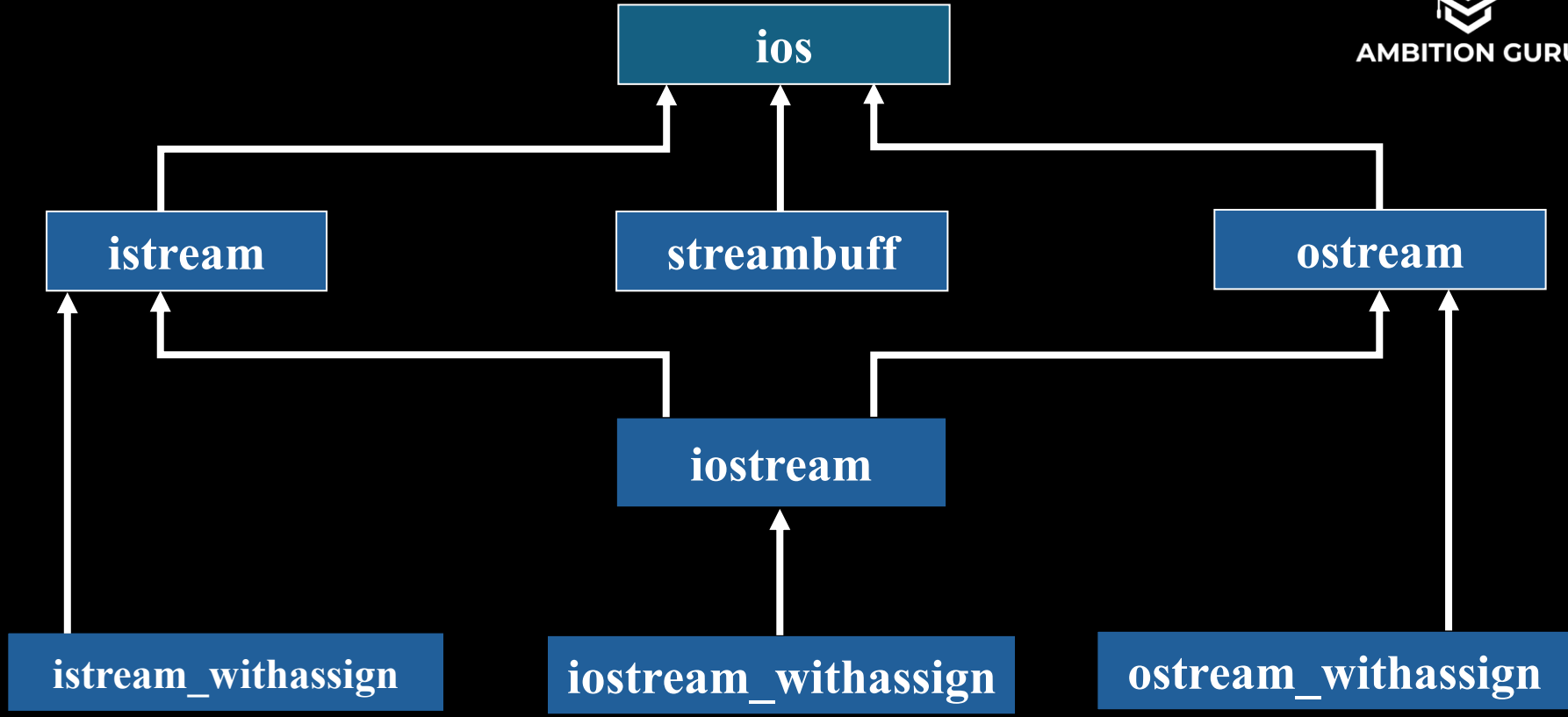


Figure: Stream classes for console Input/Output operation

Different stream classes associated with console input/output are as follows:

AMBITION GURU

i. ios(General input/output stream class)

Contains basic facilities that are used by all other input/output classes.

ii. streambuf

The streambuf class holds the characters written or read from input devices before they are sent to actual destination.

iii. istream(input stream)

- Inherits the properties of ios
- Declares input functions such as get( ),getline( ) and read( ) as member functions.
- Contains overloaded extraction operator (>>).

iv. ostream(output stream):

- inherits the properties of *ios*.
- contains output member functions put( ) and write( ).
- contains overloaded insertion operator (<<).

v. iostream(input/output stream):

- inherits the properties of *istream* and *ostream* classes.
- all of the functions and operators available with istream and ostream classes are available in iostream class.

Testing Stream Errors

Testing Stream Errors

Each stream has an internal **state flag** (to test for errors):

- **good()** → no errors
- **eof()** → end of file reached
- **fail()** → logical error (e.g., wrong data type)
- **bad()** → read/write failure

Testing Stream Errors: Example

```
#include <iostream>
using namespace std;
int main() {
    int x;
    cin >> x;
    if (cin.fail()) {
        cout << "Input failed!" << endl;
    }
    return 0;
}
```

Unit 7: Part 2

Unformatted console input/output functions
& Formatted console input/output functions

Unformatted console input/output functions

& Formatted console input/output functions

Unformatted console input/output functions

put() and get()

getline() & write()

read() & ignore()

Formatted console input/output functions

width()

precision()

fill()

setf()

unsetf()

Unformatted console input/output functions

Unformatted console input/output functions

AMBITION GURU

It works at the **character level**. No formatting, raw I/O.

Common Functions:

- ***get() / put()*** → read/write single character
- ***getline()*** → read line of text
- ***write()*** → write string with specified length
- ***read()*** → read raw bytes (used in binary files)
- ***ignore()*** → skip characters

AMBITION GURU

Unformatted console input/output functions

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

int main() {
    char c;
    cout << "Enter a character: ";
    cin.get(c);              // unformatted input
    cout << "You entered: ";
    cout.put(c);             // unformatted output

    cin.ignore();            // clear leftover newline from buffer

    char str[20];
    cout << "\nEnter a line: ";
    cin.getline(str, 20);    // reads full line
    cout << "First 10 characters: ";
    cout.write(str, 10);     // prints first 10 chars

    return 0;
}
```

Formatted console input/output functions

Formatted console input/output functions

AMBITION GURU

It provides **structured** output (width, precision, alignment).

Using ios class functions:

- *width(n)* → field width (next output only)
- *precision(n)* → number of digits displayed
- *fill(ch)* → fill empty spaces with a character
- *setf() / unsetf()* → set/unset formatting flags

Using manipulators (<iomanip>):

- *setw(n), setprecision(n), setfill(ch)*
- *hex, oct, dec, fixed, scientific*

Formatted console input/output functions

AMBITION GURU

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

int main() {
    double pi = 3.14159265;

    cout << setw(10) << setfill('*') << 123 << endl;    // ****123
    cout << fixed << setprecision(3) << pi << endl;     // 3.142
    cout << hex << 255 << endl;                         // ff

    return 0;
}
```

Unit 7: Part 3

File input/output with streams

Operations on file

File modes

File input/output with streams

File input/output with streams:

AMBITION GURU

- File Handling concept in C++ language is used to store the data permanently in computer. Using file handling we can store our data in secondary memory (Hard disk).
- The I/O system of C++ handles file operations which are very much similar to the console input and output operations. It uses file streams as an interface between the programs and the file.
- The file stream that supplies data to the program is known as input stream. The file stream that receives data from the program is known as output stream.
- Streams used for reading/writing files.
  - *ifstream* → input (read from file)
  - *ofstream* → output (write to file)
  - *fstream* → both input and output

filebuf:

- The filebuf class is used to set the file buffer to read and write.
- The filebuf class is internally used by the file stream classes for the buffer management. It contains close( ) and open( ) as members.

fstrembase:

- This class provides operations common to the file streams(ifstream,ofstream and fstream). It serves as a base for fstream, ifstream and ofstream class and it contains open( ) and close( ).

ifstream:

- This class provides input operations.
- It contains open( ) with default input mode. It inherits the function get(), getline(), read(), seekg() and tellg() from istream class.

ofstream:

- This class provides output operations. It contains open() with default output mode.It inherits put(), seekp(), tellp(), write() functions from ostream.

fstream: It provides support for simultaneous input and output operations. It inherits all the functions of istream and ostream classes through iostream.

Operations on file

Operations on file:

- There are several operations related to file such as opening, closing, reading, writing, appending, searching, deleting, modifying, checking errors etc.
- For every type of file operations, the first task is to open the file. Similarly, after the completion of each process of a file, itmust be closed too.

Opening and closing files

A file can be opened in either of the following ways:

i Using the constructor function of the stream class.

ii. Using the member function open( )

i. Using constructor:

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

ii. Using open():

```
ofstream fout;

fout.open("output.txt");
```

File Modes

File Modes

- While opening a file, before we specified only a single argument in both constructor or member function open( ).In this case the default modes are used. The fstream class does not provide a mode by default and therefore, we must provide the mode explicitly when using an object of *fstream class*.
- We can explicitly specify the mode while opening the file. The file mode is passed in second argument explicitly. The format for specifying file mode is as follows.

```
fstream streamobj;
streamobj.open("filename",mode);
```

Different files mode are listed as below:

AMBITION GURU

Mode	Meaning
ios::app	Append to the end of file
ios::ate	Go to the end of file on opening and input as well as output operations can be performed anywhere within the file (For modifying or for update)
ios::binary	Binary file(By default files are opened in text mode)
ios::in	Open file for reading only
ios::out	Open file for writing only.This also opens the file in ios::trunc mode by default i.e.its previous content are discarded.
ios::nocreate	Open fails if the file does not existalready
ios::noreplace	Open fails if tried to open already existing file in writing mode.
ios::trunc	Deletes the contents of the file if itexists

Note: The mode can combine two or more parameters with the bitwise operator.

Example:

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

- This opens the file in append mode but fails to open the file ifit("data") doesn't exist.

We can read data from file and write data to file in three ways:

- Reading or writing characters using get( ) and put( ) member functions.
- Reading or writing formatted I/O using insertion operator(<<) and extraction operator(>>)
- Reading or writing object using read( ) and write( ) member functions.

Example

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

int main() {
    ifstream fin("input.txt");   // reading
    ofstream fout("output.txt"); // writing
    fstream fio("data.txt", ios::in | ios::out); // both
    return 0;
}
```

Simple Example to Reading from and Writing to Files/Console

AMBITION GURU

Writing to a file:

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

int main() {
    ofstream fout("example.txt");
    fout << "Hello, File Handling in C++!" << endl;
    fout.close();
    return 0;
}
```

Reading from a file:

AMBITION GURU

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

int main() {
    ifstream fin("example.txt");
    string line;
    while (getline(fin, line)) {
        cout << line << endl; // prints to console
    }
    fin.close();
    return 0;
}
```

Unit 7: Part 4

Text file and Binary file

Types of Files:

Types of file supported by C++:

Text Files

Binary Files

- Text file → stores data as readable characters, separated by newline (\n).
  Editable with a text editor.
- Binary file → stores data in the same format as in memory (raw bytes).
  No line separators, not human-readable, and usually smaller in size.

Text I/O:

```
fout << data;    // write
fin >> data;     // read
```

Binary I/O:

```
fout.write((char*)&var, sizeof(var));    // write
fin.read((char*)&var, sizeof(var));      // read
```

Program to write and read a text file in C++

```
//Program to read a text file in C++
#include<iostream>
#include<fstream>
using namespace std;
int main( )
{
     char text[200];
     cout<<"Enter the text to write in to the file:"<<endl;
     cin.get(text,200);
     fstream fout;
     fout.open("student.txt",ios::out|ios::app);
     fout<<text;
     fout.close();
```

```
    fout.open("student.txt",ios::in);
    while(!fout.eof())
    {
        fout>>text;
        //skip whitespace,if fout.get(text,200); skip no white space
        cout<<text;
    }
    fout.close( );
    return 0;
}
```

Unit 7: Part 5

write() and read() function

The write () and read () function

WAP to write records of students in a file and display the records of the students according to the descending order of marks obtained by them.

```
#include<iostream>
#include<fstream>
using namespace std;
class student
{
  char name[30];
  int roll;
  float marks;
  public:
  void input( )
  {
    cout<<"Enter the name of an student:"<<endl;
    cin>>name;
```

```
        cout<<"Enter the roll of the student:"<<endl;
        cin>>roll;
        cout<<"Enter the marks obtained:"<<endl;
        cin>>marks;
    }
    void display( )
    {
        cout<<"Name of the student="<<name<<endl;
        cout<<"Roll of the student="<<roll<<endl;
        cout<<"Marks obtained="<<marks<<endl;
    }
    float ret_marks( )
    {
        return marks;
    }
};
```

```
void add_records( )
{
    student s[10];
    fstream fout;
    fout.open("Student.txt",ios::out|ios::app|ios::binary);
    for(int i=0;i<5;i++)
    {
          s[i].input( );
          fout.write((char*)&s[i], sizeof(s[i]));
    }
    fout.close( );
}
void display_records( )
{
    student s[10];
    fstream fin;
```

```
fin.open("Student.txt", ios::in|ios::binary);
for(int i=0;i<5;i++)
{
   fin.read((char*)&s[i], sizeof(s[i]));
}
for(int i=0;i<4;i++){
   for(int j=i+1;j<5;j++)
   {
        if(s[j].ret_marks( )>s[i].ret_marks( ))
        {
             student temp=s[j];
             s[j]=s[i];
             s[i]=temp;
        }
}
```

```
  for(int i=0;i<5;i++)
  {
     s[i].display();
  }
fin.close( );
}
int main()
  {
     int choice;
     while(1)
     {
     cout<<"1.Add records to the file:"<<endl;
     cout<<"2.Display records from the file:"<<endl;
     cout<<"3.Exit"<<endl;
```

```
        cout<<"Enter the choice:"<<endl;
        cin>>choice;
        switch(choice)
       {
      case 1:
          add_records( );
          break;
      case 2:
          display_records();
          break;
      case 3:
          exit(0);

      default:
          cout<<"Wrong Choice"<<endl;
      }
   }
   return 0;
}
```

Write a class student with roll, name, address, marks as member variables and member functions to read and display the information of the students. Write records of 10 students in a binary file and also read the records of the student from the binary file. Also search a specific record of the student using rollnumber as a key from the user input.

```
#include<iostream>
#include<fstream>
using namespace std;
class student
{
    int roll;
    char name[30];
    char address[30];
    float marks;
    public:
```

```
void input()
{
     cout<<"Enter the name of the student:"<<endl;
     cin>>name;
     cout<<"Enter the roll of the student:"<<endl;
     cin>>roll;
     cout<<"Enter the address of the student:"<<endl;
     cin>>address;
     cout<<"Enter the marks obtained by the student:"<<endl;
     cin>>marks;
}
```

```
    void display()
    {
        cout<<"Name of the student="<<name;
        cout<<"Roll of the student="<<roll<<endl;
        cout<<"Address of the student="<<address<<endl;
        cout<<"Marks of the student="<<marks<<endl;
    }

    int check(int r)
    {
    if(r==roll)
        return 1;
    else
        return 0;
    }
};
```

```
void write_records()
{
     int i;
     student s[10];
     fstream fout;
     fout.open("Student",ios::out|ios::app|ios::binary);
     cout<<"Enter the information of 10 student:"<<endl;
     for(i=0;i<10;i++)
     {
          s[i].input();
          fout.write((char*)&s[i], sizeof(s[i]));
     }
     fout.close( );
}
```

```
void read_records()
{
     student s;
     fstream fin;
     fin.open("Student",ios::in|ios::binary);
     while(fin.read((char*)&s, sizeof(s)))
     {
          s.display( );
     }
     fin.close( );
}
```

```
void search_specific_record()
{
  student s;
  fstream fin;
  int Roll,flag=0;
  cout<<"Enter the roll number of the student to search the record for:";
  cin>>Roll;
  fin.open("Student",ios::in|ios::binary);
  while(fin.read((char*)&s,sizeof(s)))
  {
     if(s.check(Roll))
     {
          s.display( );
          flag=1;
          break;
     }
```

```
}
      if(flag==0)
      {
            cout<<"Record not found:"<<endl;
      }
fin.close( );
   }

 void deleteRecord()
{
      fstream fin,fout;
      student s;
      int roll,flag=0;
      cout<<"Enter the roll number to be deleted:"<<endl;
      cin>>roll;
      fin.open("Student",ios::in|ios::binary);
      fout.open("Temp",ios::out|ios::binary);
```

```
while(fin.read((char*)&s, sizeof(s)))
{
      if(s.check(roll))
      {
            flag=1;
      }
      else
      {
            fout.write((char*)&s,sizeof(s));
      }
}
if(flag==0)
      cout<<"Not found:"<<endl;
else
      cout<<"Record Deleted:"<<endl;
fout.close( );
```

```
        fin.close( );
        remove("Student");
        rename("Temp","Student");
}

int main()
{
        int choice;
        while(1)
        {
                cout<<"1.Add Records:"<<endl;
                cout<<"2.Display Records:"<<endl;
                cout<<"3.Search the Record:"<<endl;
                cout<<"4.Delete the record:"<<endl;
                cout<<"5.Exit"<<endl;
                cout<<"Enter the choice:"<<endl;
                cin>>choice;
```

```
        switch(choice)
        {
        case 1:
             write_records( );
             break;
        case 2:
             read_records( );
             break;
        case 3:
             search_specific_record( );
             break;
        case 4:
             deleteRecord( );
             break;

        case 5:
             exit(0);
        default:
             cout<<"wrong choice"<<endl;
        }
        }
        return 0;
}
```

AMBITION GURU

Unit 7: Part 6

File access pointer and their manipulators

File access pointer and their manipulators

File access pointer and their manipulators

AMBITION GURU

- Each file has two associated pointers known as *file pointers*. One of them is called the *input pointer(or get pointer)* and the other is called the *output pointer(or put pointer).*
- We can use these pointers to move through the file while reading or writing(these pointers help to implement **random access** i.e. we can access any point in the file).
- The *input pointer(get pointer)* is used for reading the contents of a given file location.
- The *output pointer(put pointer)* is used for writing the given file location.
- Each time an input or output operation takes place the appropriate pointer is automatically advanced.

Default Action:

Input Pointer

Open for read only

Open in append mode

Output Pointer

Output Pointer

Open for writing only

- There are four function related to manipulation of file access pointers and they are as follows

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.

Example:

```
ifstream fin;
fin.seekg(10);
```

- Moves the get(input) pointer to the byte number 10 (i.e.the pointer will be pointing to the 11th byte in the file)

```
ofstream fout;    fout.open("Hello", ios::app);
int p=fout.tellp( );
```

- The output pointer is moved to the end of the file "Hello" and the value of p will represent the number of bytes in the file.

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

  seekg(offset, refposition);

  seekp(offset,refposition);

- Offset is the number of bytes the file pointer to be moved from the location specified by the parameter refposition which takes one of the following.

  ***ios::beg**-start from the file*

  ***ios::cur**-current position of the pointer.*

  ***ios::end**-end of file*

Examples:

- *fin.seekg(20,ios::beg)* will move get pointer of ifstream to 20 byte forward from the beginning of the file

- *fout.seekp(-40,ios::end)* will move put pointer to 40 bytes from the end of the file
- *fin.seekg(-10,ios::cur)* will move 10 byte back from the current location in the file.

Example:

```
char ch;
ifstream fin("myfile.txt",ios::binary);
fin.seekg(5);
fin.read((char*)&ch,sizeof(ch));
fin.seekg(-3,ios::cur);
fin.read((char*)&ch,sizeof(ch));
```

- Here first the character will be read from 5th position of the file "myfile.txt" and another character from 3rd position of the file(because first read moves the file access pointer to the 6th position).

AMBITION GURU

First: seekg(5)

0	1	2	3	4	5	6
1st	2nd	3rd	4th	5th	6th	7th

Second: seekg(-3,ios::cur)

0	1	2	3	4	5	6
1st	2nd	3rd	4th	5th	6th	7th

Example:

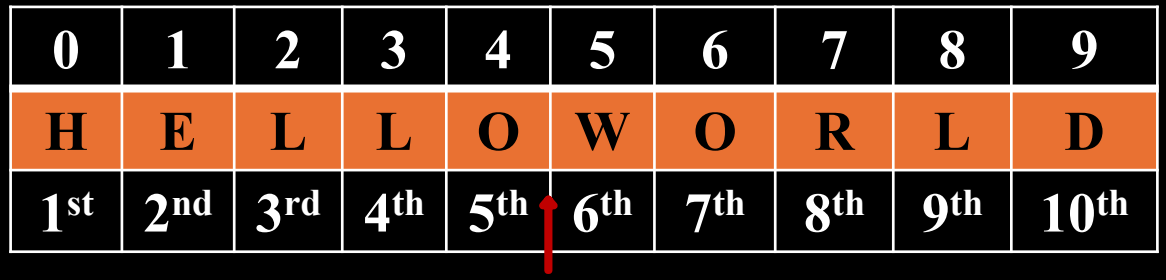
AMBITION GURU

```
ofstream fout("myfile.txt");
fout.seekp(5);
fout<<"**";
fout.seekp(-2,ios::cur);
fout<<"#";
```

- Here, first will be written '**' to the 6th and 7th position and points to 8th position. Again returns back to the 6th position and writes '#' to it.

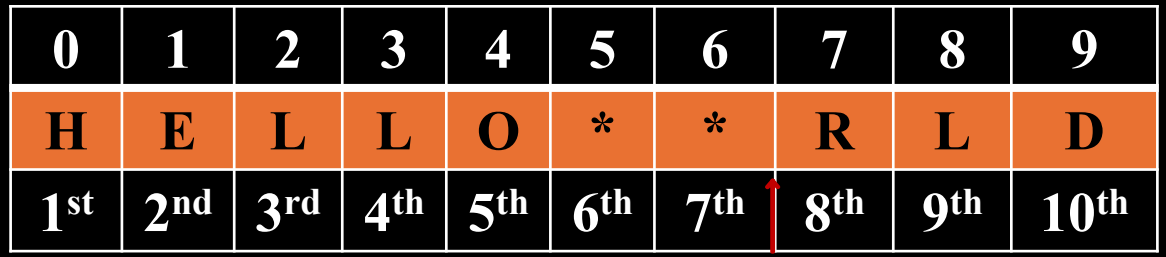
0	1	2	3	4	5	6	7	8	9
H	E	L	L	O	W	O	R	L	D
1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th

seekp(5)



0	1	2	3	4	5	6	7	8	9
H	E	L	L	O	W	O	R	L	D
1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th

fout<<“***”;



0	1	2	3	4	5	6	7	8	9
H	E	L	L	O	*	*	R	L	D
1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th

AMBITION GURU

seekp(-2,ios::cur);

0	1	2	3	4	5	6	7	8	9
H	E	L	L	O	*	*	R	L	D

fout<<“#”;

0	1	2	3	4	5	6	7	8	9
H	E	L	L	O	#	*	R	L	D

AMBITION GURU

seekp(-2,ios::cur);

0	1	2	3	4	5	6	7	8	9
H	E	L	L	O	*	*	R	L	D
1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th

fout<<“#”;

0	1	2	3	4	5	6	7	8	9
H	E	L	L	O	#	*	R	L	D
1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th

```
//Example program of tellg() and tellp()
#include<iostream>
#include<fstream>
using namespace std;
int main( ){
      ifstream fin;
      ofstream fout;
      fout.open("myfile.txt");
      fout<<"angry";
      int n=fout.tellp( );
      fout.close( );
      cout<<"Position of put pointer="<<n<<endl;
      fin.open("myfile.txt");
      int m=fin.tellg( );
      fin.close( );
      cout<<"Position of get pointer="<<m<<endl;
      return 0;
}
```

put pointer

get pointer

Example Program of Random Access

Q) WAP to store and retrieve all the information of the students in a file. Also make sure to read the nth information of a student from the file and display it.

```
#include<iostream>
#include<fstream>
using namespace std;
class student
{
  char name[30];
  int roll;
  float marks;
  public:
  void input( ){
     cout<<"Enter the name of the student:"<<endl;
     cin>>name;
     cout<<"Enter the roll number of the student:"<<endl;
     cin>>roll;
```

```
        cout<<"Enter the marks obtained by the student:"<<endl;
        cin>>marks;
    }
    void display( )
    {
        cout<<"Name of the student="<<name<<endl;
        cout<<"Roll of the student="<<roll<<endl;
        cout<<"Marks obtained by the student="<<marks<<endl;
    }
    int check(int r)
    {
        if(r==roll)
                    return 1;
        else
                    return 0;
    }
};
```

```
void adddetails( )
{
      student s;
      fstream fout;
      fout.open("Student",ios::out|ios::app|ios::binary);
      s.input( );
      fout.write((char*)&s,sizeof(s));
      fout.close( );
}
void displayrecords( )
{
      student s;
      fstream fin;
      fin.open("Student",ios::in|ios::binary);
      while(fin.read((char*)&s,sizeof(s)))
      {
            s.display( );
```

```
        }
        fin.close( );
}
void display_nth_record( )
{
        student s;
        fstream fin;
        int count=0,n;
        fin.open("Student", ios::in|ios::binary);
        while(fin.read((char*)&s, sizeof(s))){
        count ++;
        }
        fin.close( );
        if(count==0)
        {
                cout<<"Record does not exist:"<<endl;
        }
```

```
        else
        {
                cout<<"The total number of records="<<count<<endl;
                fin.open("Student_b",ios::in|ios::binary);
                cout<<"Enter the number of record to read:"<<endl;
                cin>>n;
                if(n>0 && n<=count)
                {
                        fin.seekg((n-1)*sizeof(s));
                        fin.read((char*)&s,sizeof(s));
                        s.display( );
                        fin.close( );
}}}
void modify( )
{
        fstream fout;
        student s;
        int roll,flag=0,size=sizeof(s),count=0,location;
```

```
cout<<"Enter the roll number to be modified:"<<endl;
cin>>roll;
fout.open("Student", ios::ate|ios::out|ios::in|ios::binary);
fout.seekg(0);
while(fout.read((char*)&s, sizeof(s)))
{
      count ++;
      if(s.check(roll))
      {
            s.input( );
            location=size*(count-1);
            fout.seekp(location,ios::beg);
            fout.write((char*)&s,sizeof(s));
            cout<<"Modified"<<endl;
            flag=1;
            break;
      }
}
```

```
      if(flag==0)
            cout<<"Not found"<<endl;
      fout.close( );
}
int main( )
{
     int choice;
     while(1)
     {
      cout<<"1.Add Details:"<<endl;
      cout<<"2.Display All Records:"<<endl;
      cout<<"3.Read nth Record:"<<endl;
      cout<<"4.Exit:"<<endl;
      cout<<"Enter the choice:"<<endl;
      cin>>choice;
      switch(choice)
      {
        case 1:
           adddetails( );
           break;
        case 2:
           displayrecords();
           break;
        case 3:
           display_nth_record();
           break;
        case 4:
           exit(0);
        default:
           cout<<"Wrong choice"<<endl;
        }
    }
    return 0;
    }
```

Testing errors during file operation

Testing errors during file operation

- So far, we have been opening and using the files for reading and writing on the assumptions that everything is fine with files. This may not be true always. For instance, one of the followings may happen while dealing with files.

i. *A file which we are attempting to open for reading does not exist.*

ii. *The file name used for a new file may already exist.*

iii. *We may attempt an invalid operation such as reading by placing file pointer at the end of the file.*

iv. *There may not be any space in the disk for storing the data.*

v. *We may use an invalid filename.*

vi. *We may attempt to perform an operation when the file is not opened for that purpose.*

- The class ios supports various member functions that can be used to read the status recorded in a file stream. Someof these functions are listed as below

1. eof(): Returns true(non-zero) if end-of-file is encountered otherwise returns false(zero).
2. fail(): Returns true when input or output operation has failed.
3. bad(): Returns true if an invalid operation is attempted or any unrecoverable error has occurred.
4. good(): Returns true if no error has occurred and further I/O operations can be carried out. When it returns false no further operations can be carried out or performed.

Examples:

```
ifstream fin;
while(1)
{
…..//Reading contents of the file;
if(fin.eof( ))
{
//…end of file;
break;
}
}
ofstream fout; fout.open(“student.txt” );
```

```
if(fout.bad())
{
//file cannot be written;
}
ofstream fout;
fout.write((char*)&s, sizeof(s));
if(fout.fail())
{
//file not opened;
}
ifstream fin;
fin.open("Student.txt");
if(fin.good())
{
//no error has occurred,perform read operation.
}
```

```
ifstream fin;
fin.open(“Teacher.txt”)
if(!fin)
{
        //file does not exist for reading.
}
```

THANK YOU
Any Queries ?