OBJECT ORIENTED PROGRAMMING (PCC-CS503)

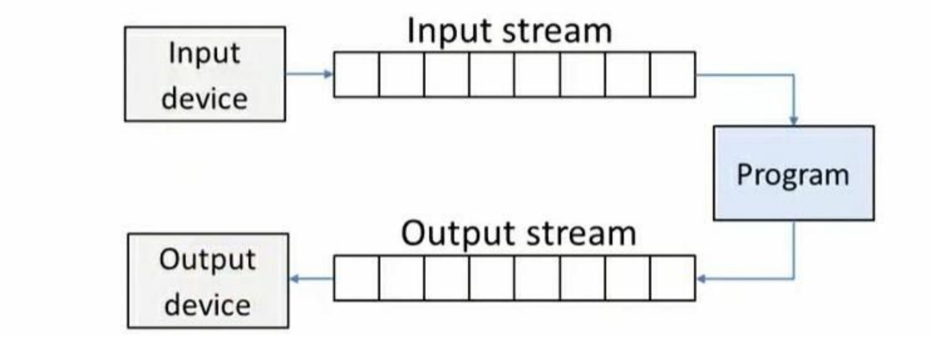
Unit – 7

Input and Output

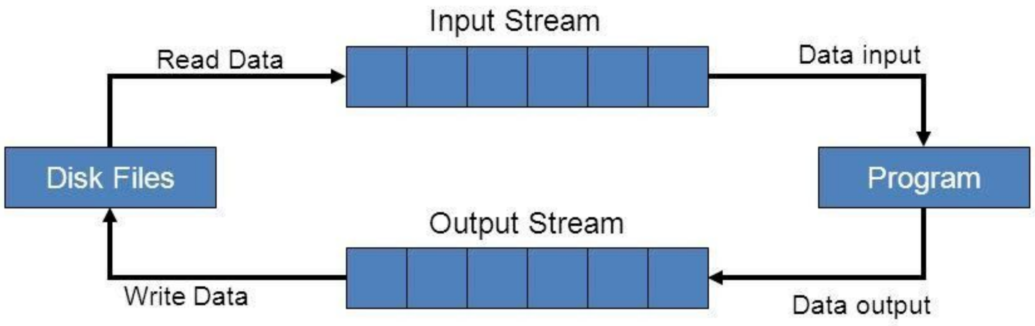
Streams: C++ uses file streams (sequence of bytes) as an interface between the programs and the I/O devices. There can be two types of I/O operations:

- Console oriented (cin, cout)
- File oriented (file handling functions)

In case the data is being read from/displayed into I/O device: use standard I/O operations (cin and cout) which are included in header file <iostream.h>



In case the data is being read from/displayed into Disk Files: use file handling operations which are included in header file <fstream.h>



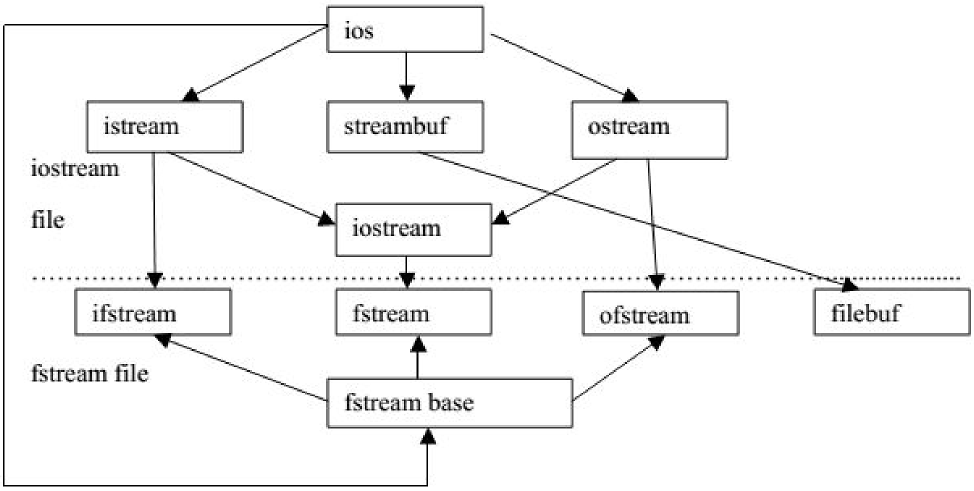
Files:

- Files are used to store data in a storage device permanently.
- **File handling** provides a mechanism to store the output of a program in a file and to perform various operations on it.
- In C++ we have a set of in-built file handling methods and classes.
- The classes for file handling include: ifstream, ofstream, and fstream.
- These classes are derived from fstrembase and from the corresponding iostream class.
- These classes are designed to manage the disk files and are declared in fstream and therefore we must include fstream header file in any program that uses files.

In C++, files are mainly dealt by using three classes fstream, ifstream, ofstream:

- **ofstream**: This Stream class signifies the output file stream and is applied to create files for writing information to files
- **ifstream**: This Stream class signifies the input file stream and is applied for reading information from files
- **fstream**: This Stream class can be used for both read and write from/to files.

Stream classes for I/O operations



Stream classes for file operations

Library functions: The C++ Standard Library provides a rich collection of functions for performing various tasks like mathematical calculations, string manipulations, character manipulations, input/output, error checking and many other useful operations.

Following is the list of library functions available for file handling.

1. open(): To create a file
2. close(): To close an existing file
3. get(): to read a single character from the file
4. put(): to write a single character in the file
5. read(): to read data from a file
6. write(): to write data into a file

Error handling functions:

1. eof(): returns true (non zero) if end of file is encountered while reading; otherwise return false(zero)
2. fail(): return true when an 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.

Ques 1: What are the different types of errors that can happen during file handling?

Ques 2: Try out the syntax for C++ Standard Library functions used in File handling.

File pointer

- 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.

Operations on file: opening, writing, reading, closing

Opening a File in File Handling

A file must be opened before performing any function on it. An open file is represented within a program by a stream and any input or output task performed on this stream will be applied to the physical file associated with it. The syntax of opening a file in C++ is:

open (filename, mode);

There are some mode flags used for file opening. These are:

Flag	**Meaning**
ios::in	Searches for the file and opens it in the read mode only(*if the file is found*).
ios::out	Searches for the file and opens it in the write mode. If the file is found, its content is overwritten. If the file is not found, a new file is created. *Allows you to write to the file.*
ios::app	Searches for the file and opens it in the append mode i.e. this mode allows you to append new data to the end of a file. If the file is not found, a new file is created.
ios::binary	Searches for the file and opens the file (if the file is found) in a binary mode to perform binary input/output file operations.
ios::ate	Searches for the file, opens it and positions the pointer at the end of the file. This mode when used with ios::binary, ios::in and ios::out modes, *allows you to modify the content of a file.*

Example:

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

 void main()
 {
     fstream st; // Step 1: Creating object of fstream class
     st.open("E:\sitesbay.txt",ios::out);  // Step 2: Creating new file
     if(!st) // Step 3: Checking whether file exist
     {
         cout<<"File creation failed";
     }
     else
     {
         cout<<"New file created";
         st.close(); // Step 4: Closing file
     }
     getch();
 }
```

Writing to File in File Handling

```
void main()
{
    fstream st; // Step 1: Creating object of fstream class
    st.open("E:\studytonight.txt",ios::out);  // Step 2: Creating new file
    if(!st) // Step 3: Checking whether file exist
    {
        cout<<"File creation failed";
    }
    else
    {
        cout<<"New file created";
        st<<"Hello";    // Step 4: Writing to file
        st.close(); // Step 5: Closing file
    }
    getch();
}
```

Reading from File in File Handling

```
void main()
{
    fstream st; // step 1: Creating object of fstream class
    st.open("E:\sitesbay.txt",ios::in);   // Step 2: Creating new file
    if(!st) // Step 3: Checking whether file exist
    {
        cout<<"No such file";
    }
    else
    {
        char ch;
        while (!st.eof())
        {
            st >>ch;  // Step 4: Reading from file
            cout << ch;   // Message Read from file
        }
        st.close(); // Step 5: Closing file
    }
    getch();
}
```

Closing a File in File Handling

```
void main()
{
    fstream st; // Step 1: Creating object of fstream class
    st.open("E:\sitesbay.txt",ios::out);  // Step 2: Creating new file
    st.close(); // Step 4: Closing file
    getch();
}
```

Formatted output

- C++ provides both the *formatted* and *unformatted* IO functions.
- In unformatted or low-level IO, bytes are treated as raw bytes and unconverted to any particular format (cin, cout, put, get, getline, write)
- In formatted or high-level IO, bytes are grouped and converted to a particular format.
- C++ provides a variety of features that can be used for this purpose:
    - Using the ios class and various ios member functions, along with flags.
    - Using manipulators(special functions)

Using the ios Stream class functions and flags:

The ios Stream class consists of number of functions which help in formatting the output in a variety of ways, along with the various flags.

Few standard ios class functions are: width, precision, fill, setf, unsetf.

1. **width():** The width method is used to set the required field width. The output will be displayed in the given width

```
cout.width(10);
cout << "Hello";
```

O/P:

```
     Hello
//5 Blank spaces before "Hello"
```

2. **precision():** The precision method is used to set the number of the decimal point to a float value

```
cout.precision(3);
cout << 3.144678;
```

O/P:

```
3.14
```

3. **fill():** The fill method is used to set a character to fill in the blank space in a field, when using width. Takes the character as parameter.

```
cout.fill('*');
cout.width(10);
cout << 1234;
```

O/P:

```
******1234
```

4. **setf():** The setf method is used to set various flags for formatting output

Common Stream Flags

Flag Name	Corresponding Stream Manipulator	Description
`ios::fixed`	`fixed`	if this is set, floating point numbers are printed in fixed-point notation. When this flag is set, `ios::scientific` is automatically unset
`ios::scientific`	`scientific`	if this is set, floating point numbers are printed in scientific (exponential) notation. When this flag is set, `ios::fixed` is automatically unset
`ios::showpoint`	`showpoint`	if this is set, the decimal point is always shown, even if there is no precision after the decimal. Can be unset with the manipulator **`noshowpoint`**
`ios::showpos`	`showpos`	if set, positive values will be preceded by a plus sign + . Can be unset with the manipulator **`noshowpos`**.
`ios::right`	`right`	if this is set, output items will be right-justified within the field (when using `width()` or `setw()`), and the unused spaces filled with the fill character (the space, by default).
`ios::left`	`left`	if this is set, output items will be left-justified within the field (when using `width()` or `setw()`), and the unused spaces filled with the fill character (the space, by default).
`ios::showbase`	`showbase`	Specifies that the base of an integer be indicated on the output. Decimal numbers have no prefix. Octal numbers (base 8) are prefixed with a leading **`0`**. Hexadecimal numbers (base 16) are prefixed with a leading **`0x`**. This setting can be reset with the manipulator **`noshowbase`**.
`ios::uppercase`	`uppercase`	specifies that the letters in hex outputs (a-f) and the letter 'e' in scientific notation will be output in uppercase. This can be reset with the manipulator **`nouppercase`**.

Example:

```
int x = 1234;
cout.setf(ios::right);
cout.width(10);
cout << "Hello";
cout.width(15);
cout << x;
```

O/P:

```
     Hello           1234
```

5. **unsetf():** The unsetf method is used to remove the flag setting

Example 1

```
float x = 18.0;
cout<< x << endl;              //displays 18
cout.setf(ios::showpoint);
cout<< x << endl;              //displays 18.0000
cout.setf(ios::scientific);
cout<< x << endl;              //displays 1.800000e+001
cout.unsetf(ios::showpoint);
cout.unsetf(ios::scientific);
cout<<x<<endl;                 //displays 18
```

O/P:

```
18
18.0000
1.800000e+01
18
```

Example 2

```
double a = 3.1415926534;
double b = 2006.0;
double c = 1.0e-10;

std::cout.precision(5);
std::cout << a << '\n'

std::cout << "fixed:\n" << std::fixed;
std::cout << a << '\n' << b << '\n' << c << '\n';

std::cout << '\n';

std::cout << "scientific:\n" << std::scientific;
std::cout << a << '\n' << b << '\n' << c << '\n';
```

O/P:

```
 3.1416

fixed:
3.14159
2006.00000
0.00000

scientific:
3.14159e+000
2.00600e+003
1.00000e-010
```

Stream Manipulators

- A **stream manipulator** is a symbol or function that is used by placing it on the right side of the *insertion operator* << .
  - A plain manipulator is just a symbol, like a variable:

    ```
      cout << endl;
    // endl is a stream manipulator
    ```

  - A *parameterized stream manipulator* looks like a function call -- it has one or more parameters:

    ```
      cout << setw(10);
    // setw() is a parameterized manipulator
    ```

  - To use parameterized stream manipulators, you need to include the `<iomanip>` library

- Many of the stream manipulators are just alternate ways of doing tasks performed by member functions. A nice benefit is that cascading can be used, intermixing manipulators and other output statements that use the insertion operator

  ```
  cout << setw(10) << "Hello" << endl;
  ```

- **`setprecision()`** is a parameterized stream manipulator that performs the same task as the member function `precision()`

  ```
  cout.precision(2);
  // sets decimal precision to 2 significant digits
  cout << setprecision(2);
  // does the same thing!
  ```

- **`setw()`** is a parameterized stream manipulator that performs the same task as the member function `width()`

  ```
    cout.width(10);
    // sets field width to 10 for next output
    cout << setw(10);
    // does the same thing!
  ```

- **`setfill()`** is a parameterized stream manipulator that performs the same task as the member function `fill()`

  ```
    cout.fill('*');          // sets fill character to '*'
    cout << setfill('*');      // does the same thing!
  ```

- **`setiosflags()`** is a parameterized stream manipulator that performs the same task as the member function `setf()`

  ```
    cout.setf(ios::left);
  ```

```
// sets left justification flag
cout << setiosflags(ios::left);
// does the same thing!
```

- There are also some newer stream manipulators that correspond to some of the formatting flags. For example:

```
cout.setf(ios::left);
// sets left justification for cout
cout << left;
// also sets left justification for cout
```

Some other Stream Manipulators:

Manipulator	**Description**
`flush`	causes the output buffer to be flushed to the output device before processing proceeds
`endl`	prints a newline and flushes the output buffer
`dec`	causes integers to be printed in decimal (base 10)
`oct`	causes integers from this point to be printed in octal (base 8)
`hex`	causes integers from this point to be printed in hexadecimal (base 16)
`setbase()`	a parameterized manipulator that takes either 10, 8, or 16 as a parameter, and causes integers to be printed in that base. `setbase(16)` would do the same thing as `hex`, for example
`internal`	if this is set, a number's sign will be left-justified and the number's magnitude will be right-justified in a field (and the fill character pads the space in between). Only one of `right`, `left`, and `internal` can be set at a time.
`boolalpha`	causes values of type `bool` to be displayed as words (`true` or `false`)
`noboolalpha`	causes values of type `bool` to be displayed as the integer values 0 (for false) or 1 (for true)