

STREAMS UNIT-5

Stream: A stream is sequence of bytes. It acts either as source from which the input data can be obtained or as a destination to which the output data can be sent. The source stream that provides data to the program is called the input stream and destination stream that receives the data is the output stream.

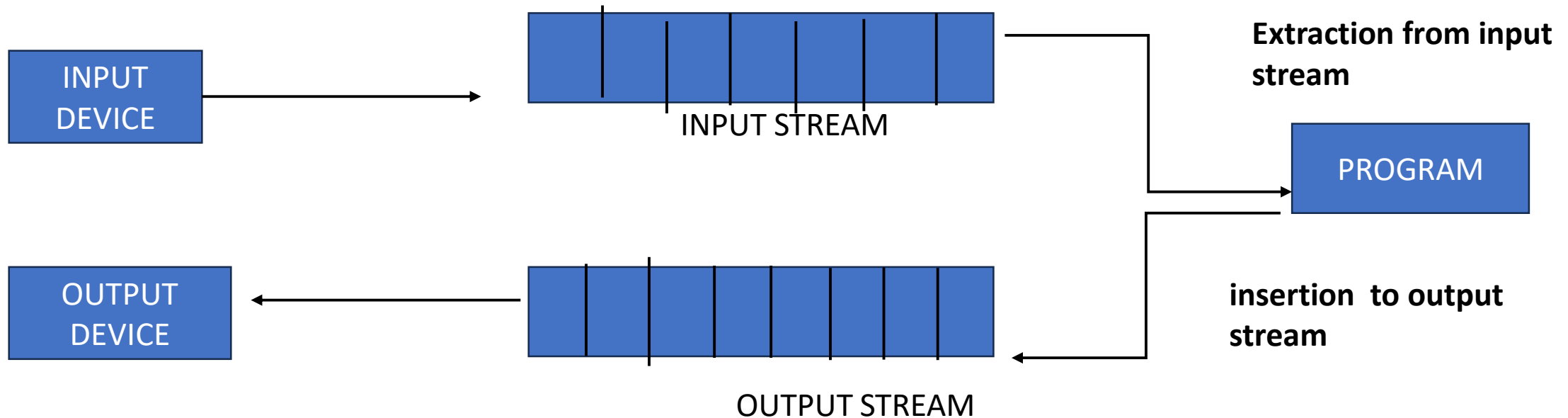
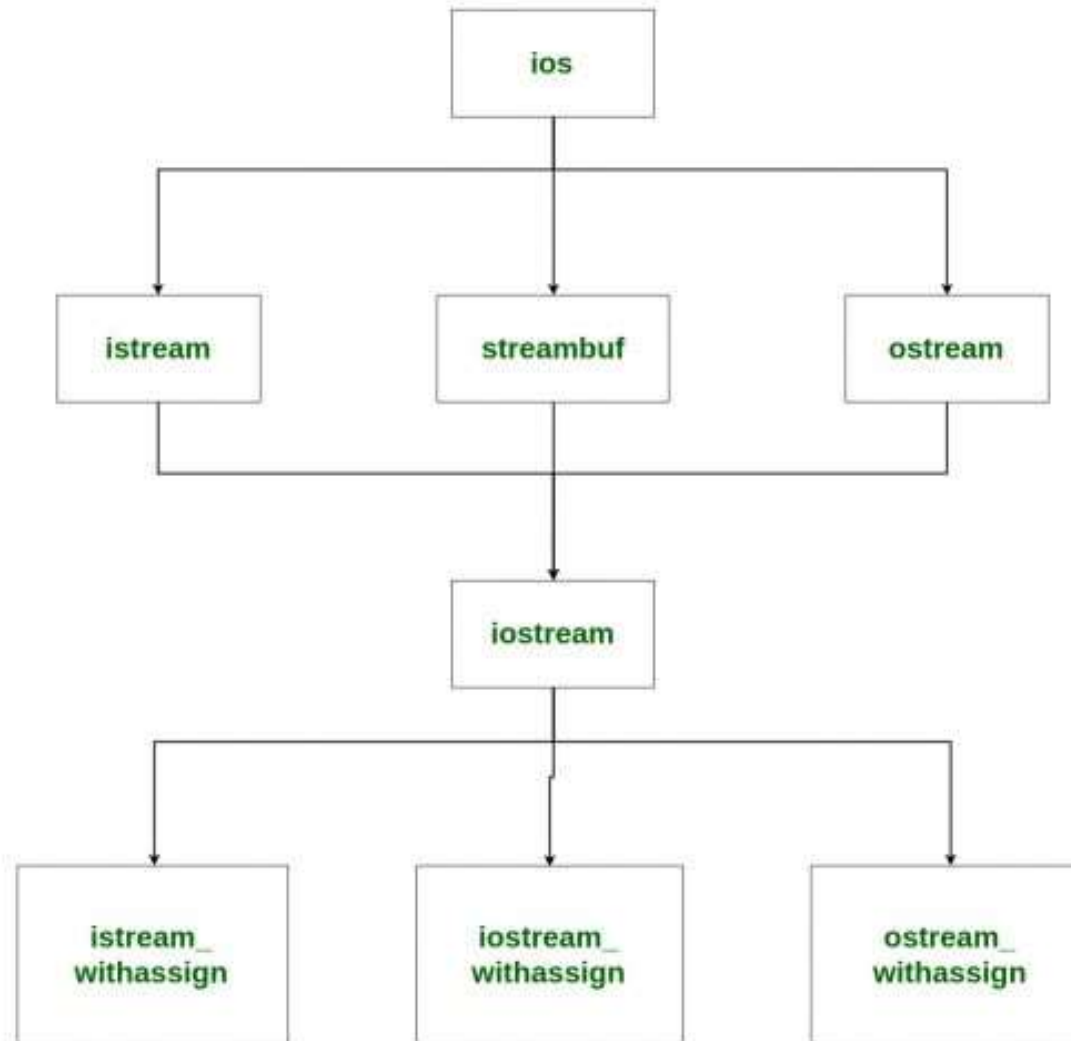


Figure of data stream

Heirarchy of Stream Classess in iostream.h



1. **ios class** is topmost class in the stream classes hierarchy. It is the base class for **istream**, **ostream**, and **streambuf** class.
2. **istream** and **ostream** serves the base classes for **iostream** class. The class **istream** is used for input and **ostream** for the output.
3. Class **ios** is indirectly inherited to **iostream** class using **istream** and **ostream**. To avoid the duplicity of data and member functions of **ios** class, it is declared as virtual base class when inheriting in **istream** and **ostream**

- **The ios class:** The ios class is responsible for providing all input and output facilities to all other stream classes.
- **The istream class:** This class is responsible for handling input stream. It provides number of function for handling chars, strings and objects such as get, getline, read, ignore, putback etc.
- **The ostream class:** This class is responsible for handling output stream. It provides number of function for handling chars, strings and objects such as write, put etc..
- **The iostream:** This class is responsible for handling both input and output stream as both istream class and ostream class is inherited into it. It provides function of both istream class and ostream class for handling chars, strings and objects such as get, getline, read, ignore, putback, put, write etc.
- **istream_withassign class:** This class is variant of istream that allows object assignment. The predefined object cin is an object of this class and thus may be reassigned at run time to a different istream object.

- **ostream_withassign class:** This class is variant of ostream that allows object assignment. The predefined objects cout, cerr, clog are objects of this class and thus may be reassigned at run time to a different ostream object.

Example :istream and ostream

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

```
int main()
{
    char x;

    // used to scan a single char
    cin.get(x);

    cout << x;
}
```

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

```
int main()
{
    char x;

    // used to scan a single char
    cin.get(x);

    // used to put a single char onto the screen.
    cout.put(x);
}
```

- standard output stream(cout): cout is the instance of the ostream class. cout is used to produce output on the standard output device which is usually the display screen. The data needed to be displayed on the screen is inserted in the standard output stream(cout) using the insertion operator(<<).
- Standard error stream (cerr): cerr is the standard error stream which is used to output the errors. It is an instance of the ostream class. As cerr stream is un-buffered so it is used when we need to display the error message immediately and does not store the error message to display later..
- The “c” in cerr refers to “character” and ‘err’ means “error”, Hence cerr means “character error”. It is always a good practice to use cerr to display errors.

// C++ program to illustrate std::cerr

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

```
int main()
{
```

```
    // This will print "Welcome to GfG"
    // in the error window
    cerr << "Welcome to GfG! :: cerr";
```

```
    // This will print "Welcome to GfG"
    // in the output window
    cout << "Welcome to GfG! :: cout";
    return 0;
```

```
}
```

```
Welcome to GfG! :: cerrWelcome to GfG! :: cout
PS C:\Users\Lenovo\c++>
```

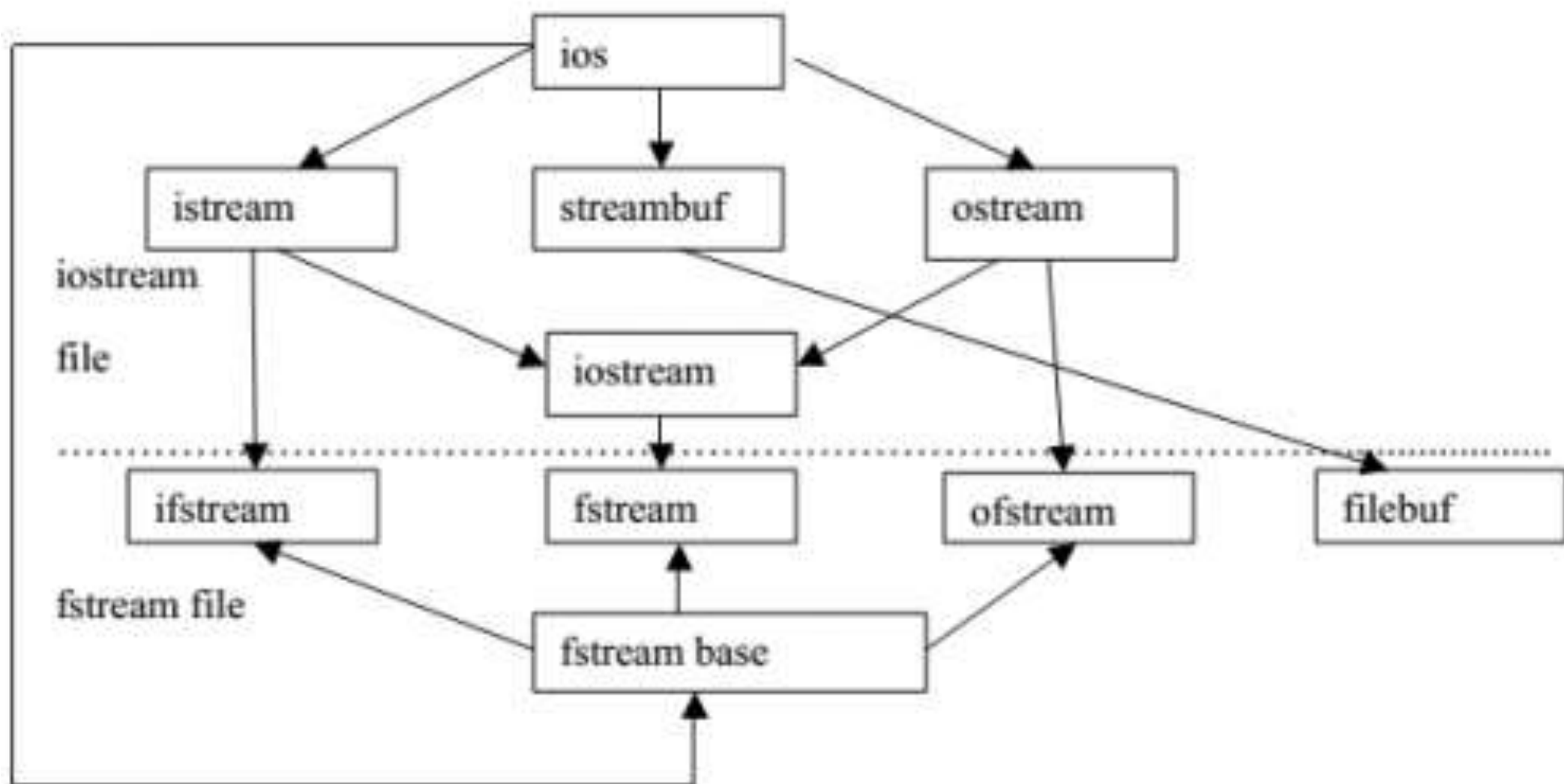



Figure 6.2 Stream classes for file operations

fstream

- `fstream` supports for simultaneous I/O operation on files
- `ifstream`- input operation on files (to read data from file)
- `ofstream`- will provide output operation on file (to write data on file)

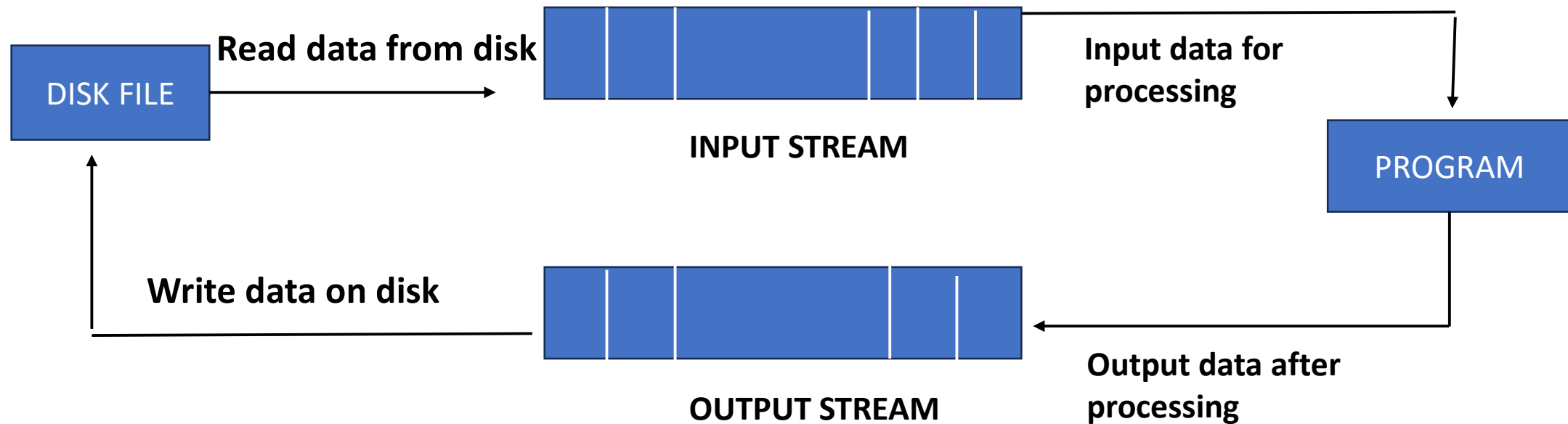


Figure of file operations

File handling

- File handling is used to store data permanently in a computer. Using file handling we can store our data in secondary memory (Hard disk).
- How to achieve the File Handling
- For achieving file handling we need to follow the following steps:-
 - STEP 1-Naming a file
 - STEP 2-Opening a file
 - STEP 3-Writing data into the file
 - STEP 4-Reading data from the file
 - STEP 5-Closing a file

- In C++, files are mainly dealt by using three classes `fstream`, `ifstream`, `ofstream` available in `fstream` headerfile.
- `ofstream`: Stream class to write on files
- `ifstream`: Stream class to read from files
- `fstream`: Stream class to both read and write from/to files.

ofstream (Output File Stream)

In C++, `ofstream` and `ifstream` are classes provided by the C++ Standard Library for file I/O operations. These classes are defined in the `<fstream>` header and are part of the `std` namespace. `ofstream` (Output File Stream) `ofstream` is used to create files and write information to files.

Usage: When you want to write data to a file.

Syntax: `std::ofstream outFile("filename.txt");`

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

```
int main() {
    // Create an ofstream object
    ofstream outFile("example.txt");
```

```
    // Check if the file is open
    if (outFile.is_open()) {
        // Write to the file
        outFile << "Hello, world!" << endl;
        outFile << "This is a test." << endl;
```

```
    // Close the file
        outFile.close();
        cout << "Data written to file successfully." << endl;
    } else {
        cout << "Unable to open file for writing." << endl;
    }

    return 0;
}
```

Using the is_open() Function

The is_open() function returns

true - if the file was opened successfully.
false - if the file failed to open or if it is in a state of error.

ifstream (Input File Stream) ifstream is used to read information from files.

- **Usage:** When you want to read data from a file.
- **Syntax:** ifstream inFile("filename.txt");

```
#include <fstream>                                     // Read the file line by line
#include <iostream>                                     while (getline(inFile, line)) {
using namespace std;                                   cout << line << endl;
                                                        }

int main() {                                           // Close the file
    // Create an ifstream object                       inFile.close();
    ifstream inFile("example.txt");                   } else {
                                                        cout << "Unable to open file for reading." << endl;
                                                        }

    // Check if the file is open
    if (inFile.is_open()) {
        string line;
        return 0;
    }
```

Open()

- Opening a File
- A file must be opened before you can read from it or 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);

Here, the first argument specifies the name and location of the file to be opened and the second argument of the open() member function defines the mode in which the file should be opened

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 the file

You can combine two or more of these values by ORing them together. For example if you want **to open a file** in write mode and want to truncate it in case that already exists, following will be the syntax –

```
ofstream outfile;  
outfile.open("file.dat", ios::out | ios::trunc );
```


Similar way, you can open a file for reading and writing purpose as follows –

```
fstream afile;
```

```
afile.open("file.dat", ios::out | ios::in );
```

Closing a File

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();
```

```
#include <fstream>
#include <iostream>
using namespace std;
int main () {
    char data[100];
    // open a file in write mode.
    ofstream outfile;
    outfile.open("afile.dat");
    cout << "Writing to the file" << endl;
    cout << "Enter your name: ";
    cin.getline(data, 100);
    // write inputted data into the file.
    outfile << data << endl;

    cout << "Enter your age: ";
    cin >> data;
    cin.ignore();
```

```
// again write inputted data into the file.
    outfile << data << endl;

    // close the opened file.
    outfile.close();

    // open a file in read mode.
    ifstream infile;
    infile.open("afile.dat");

    cout << "Reading from the file" << endl;
    infile >> data;

    // write the data at the screen.
    cout << data << endl;

    // again read the data from the file and display it.
    infile >> data;
    cout << data << endl;

    // close the opened file.
    infile.close();
    return 0;
}
```

Getline()

```
#include<iostream>
using namespace std;
int main(){
char name[20];
cout<<"enter ";
cin.getline(name,20);
cout<<"the name is:"<<name;
    return 0;
}
```

```
#include<iostream>

using namespace std;

int main(){
char name[20];fish
cout<<"enter ";
cin>>name;
cout<<"the name is:"<<name;
    return 0;
}
```

- The ignore function in C++ is a member function of input stream classes like istream, which includes cin, ifstream, and fstream when used for input. The ignore function is used to skip (ignore) characters in the input buffer. It is particularly useful when you want to skip unwanted characters or handle input that includes delimiters or extra characters that you don't need to process.

- **Syntax-**

istream& ignore(streamsize n = 1, int delim = EOF);

- **n:** The number of characters to ignore (default is 1).
- **delim:** The delimiter character up to which characters will be ignored (default is EOF)

Templates

A template is a simple yet very powerful tool in C++. **The simple idea is to pass the data type as a parameter so that we don't need to write the same code for different data types.** For example, a software company may need to sort() for different data types. Rather than writing and maintaining multiple codes, we can write one sort() and pass the datatype as a parameter.

C++ adds two new keywords to support templates: 'template' and 'typename'. The second keyword can always be replaced by the keyword 'class'.

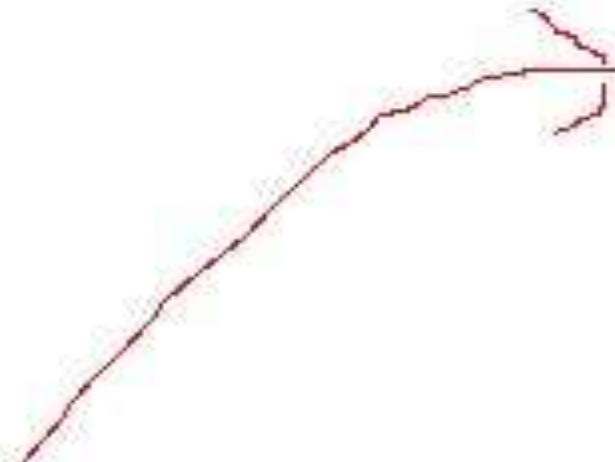
How Do Templates Work?

Templates are expanded at compiler time. This is like macros. The difference is, that the compiler does type-checking before template expansion. The idea is simple, source code contains only function/class, but compiled code may contain multiple copies of the same function/class.

Compiler internally generates and adds below code

```
template <typename T>
T myMax(T x, T y)
{
    return (x > y)? x: y;
}
```


```
int myMax(int x, int y)
{
    return (x > y)? x: y;
}
```



```
int main()
{
    cout << myMax<int>(3, 7) << endl;
    cout << myMax<char>('g', 'e') << endl;
    return 0;
}
```

Compiler internally generates and adds below code.

```
char myMax(char x, char y)
{
    return (x > y)? x: y;
}
```



Defining a Function Template

A function template starts with the **keyword template** followed by template parameter(s) inside <> which is followed by the function definition.

```
template <typename T>  
T functionName(T parameter1, T parameter2, ...) {  
    // code  
}
```

In the above code, **T** is a **template argument** that accepts different data types (int, float, etc.), and **typename** is a keyword.

When an argument of a data type is passed to functionName(), the compiler generates a new version of functionName() for the given data type.

For calling function template from any other function

Syntax: functionName<dataType>(parameter1, parameter2,...)

Example:

```
template <typename T>
T add(T num1, T num2) {
    return (num1 + num2);
}


int main() {
    int result1;
    double result2;
    result1 = add<int>(2, 3); // calling with int parameters
    cout << result1 << endl;
    result2 = add<double>(2.2, 3.3); // calling with double parameters
    cout << result2 << endl;
    return 0;
}
```



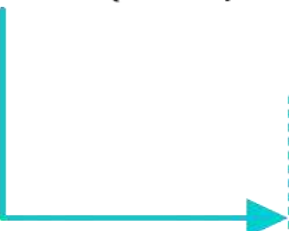
```
#include<iostream>
```

```
template<typename T>  
T add(T num1, T num2) {  
    return (num1 + num2);  
}
```

```
int main() {  
    ... ..  
  
    result1 = add<int>(2,3);  
    ... ..  
  
    result2 = add<double>(2.2,3.3);  
    ... ..  
}
```



```
int add(int num1, int num2) {  
    return (num1 + num2);  
}
```



```
double add(double num1, double num2) {  
    return (num1 + num2);  
}
```

