

## Dynamic memory Allocation

```
#include <iostream>
using namespace std;
int main() {
    int* pointInt; // declare an int pointer
    float* pointFloat; // declare a float pointer
    pointInt = new int; // dynamically allocate memory
    pointFloat = new float;
    *pointInt = 45; // assigning value to the memory
    *pointFloat = 45.45f;
    cout << *pointInt << endl;
    cout << *pointFloat << endl;
    delete pointInt; // deallocate the memory
    delete pointFloat;
    return 0;
}
```

```
// C++ Program to store GPA of n number of students and display it
// where n is the number of students entered by the user
#include <iostream>
using namespace std;
int main() {
    int num;
    cout << "Enter total number of students: ";
    cin >> num;
    float* ptr;
    // memory allocation of num number of floats
    ptr = new float[num];
    cout << "Enter GPA of students." << endl;
    for (int i = 0; i < num; ++i) {
        cout << "Student" << i + 1 << ": ";
        cin >> *(ptr + i);
    }
}
```

```

    cout << "\nDisplaying GPA of students." << endl;
}
for (int i = 0; i < num; ++i)
{
    cout << "Student" << i + 1 << ": " << *(ptr + i) << endl;
}
// ptr memory is released
delete[] ptr;
return 0;
}

```

```

#include <iostream>
using namespace std;

```

```

class Student {
private:
    int age;
    string name;

```

```

public:

```

```

    // constructor initializes age to 12
    Student()
    {
        age= 23;
        name=abcd;
    }

    void getAgenName() {
        cout << "Age = " << age << " name=" << name << endl;
    }
};
int main() {

```

```

    // dynamically declare Student object
    Student* ptr = new Student();

```

```

    // call getAge() function
    ptr->getAge();

```

```

    // ptr memory is released

```

```
delete ptr;

return 0;
}
```

## File Handling

```
/* File Handling with C++ using ifstream & ofstream class object*/
/* To write the Content in File*/
/* Then to read the content of file*/
#include <iostream>
```

```
/* fstream header file for ifstream, ofstream,
fstream classes */
#include <fstream>
```

```
using namespace std;
```

```
// Driver Code
```

```
int main()
```

```
{
```

```
    // Creation of ofstream class object
    ofstream fout;
```

```
    string line;
```

```
    // by default ios::out mode, automatically deletes
    // the content of file. To append the content, open in ios::app
    // fout.open("sample.txt", ios::app)
    fout.open("sample.txt");
```

```
    // Execute a loop If file successfully opened
    while (fout) {
```

```
        // Read a Line from standard input
        getline(cin, line);
```

```
        // Press -1 to exit
        if (line == "-1")
            break;
```

```

        // Write line in file
        fout << line << endl;
    }

    // Close the File
    fout.close();

    // Creation of ifstream class object to read the file
    ifstream fin;

    // by default open mode = ios::in mode
    fin.open("sample.txt");

    // Execute a loop until EOF (End of File)
    while (getline(fin, line)) {

        // Print line (read from file) in Console
        cout << line << endl;
    }

    // Close the file
    fin.close();

    return 0;
}

```

Q: write a single file handling program in c++ to reading and writing data on a file.

```

#include<iostream>
#include<fstream>

using namespace std;
main()
{
    int rno,fee;
    char name[50];

    cout<<"Enter the Roll Number:";
    cin>>rno;

    cout<<"\nEnter the Name:";
    cin>>name;
}

```

```

    cout<<"\nEnter the Fee:";
    cin>>fee;

    ofstream fout("d:/student.doc");

    fout<<rno<<"\t"<<name<<"\t"<<fee; //write data to the file student

    fout.close();

    ifstream fin("d:/student.doc");

    fin>>rno>>name>>fee; //read data from the file student

    fin.close();

    cout<<endl<<rno<<"\t"<<name<<"\t"<<fee;

    return 0;
}

```

### **Read/Write Class Objects from/to File in C++**

Given a file “Input.txt” in which every line has values same as instance variables of a class. Read the values into the class’s object and do necessary operations.

#### **Theory :**

The data transfer is usually done using '>>'

and '<<\' operators. But if you have

a class with 4 data members and want

to write all 4 data members from its

object directly to a file or vice-versa,

we can do that using following syntax :

#### **To write object's data members in a file :**

```

// Here file_obj is an object of ofstream
file_obj.write((char *) & class_obj, sizeof(class_obj));

```

#### **To read file's data members into an object :**

```

// Here file_obj is an object of ifstream
file_obj.read((char *) & class_obj, sizeof(class_obj));

```

#### **Examples:**

##### **Input :**

Input.txt :  
Michael 19 1806  
Kemp 24 2114  
Terry 21 2400  
Operation : Print the name of the highest  
rated programmer.

**Output :**

Terry

```
// C++ program to demonstrate read/write of class
// objects in C++.
#include <iostream>
#include <fstream>
using namespace std;

// Class to define the properties
class Contestant {
public:
    // Instance variables
    string Name;
    int Age, Ratings;

    // Function declaration of input() to input info
    int input();

    // Function declaration of output_highest_rated() to
    // extract info from file Data Base
    int output_highest_rated();
};

// Function definition of input() to input info
int Contestant::input()
{
    // Object to write in file
    ofstream file_obj;

    // Opening file in append mode
    file_obj.open("Input.txt", ios::app);

    // Object of class contestant to input data in file
    Contestant obj;
```

```

// Feeding appropriate data in variables
string str = "Michael";
int age = 18, ratings = 2500;

// Assigning data into object
obj.Name = str;
obj.Age = age;
obj.Ratings = ratings;

// Writing the object's data in file
file_obj.write((char*)&obj, sizeof(obj));

// Feeding appropriate data in variables
str = "Terry";
age = 21;
ratings = 3200;

// Assigning data into object
obj.Name = str;
obj.Age = age;
obj.Ratings = ratings;

// Writing the object's data in file
file_obj.write((char*)&obj, sizeof(obj));

//close the file
//It's always a good practice to close the file after opening them
file_obj.close();

return 0;
}

// Function definition of output_highest_rated() to
// extract info from file Data Base
int Contestant::output_highest_rated()
{
    // Object to read from file
    ifstream file_obj;

    // Opening file in input mode
    file_obj.open("Input.txt", ios::in);

    // Object of class contestant to input data in file

```

```

Contestant obj;

// Reading from file into object "obj"
file_obj.read((char*)&obj, sizeof(obj));

// max to store maximum ratings
int max = 0;

// Highest_rated stores the name of highest rated contestant
string Highest_rated;

// Checking till we have the feed
while (!file_obj.eof()) {
    // Assigning max ratings
    if (obj.Ratings > max) {
        max = obj.Ratings;
        Highest_rated = obj.Name;
    }

    // Checking further
    file_obj.read((char*)&obj, sizeof(obj));
}

// close the file.
//It's always a good practice to close the file after opening them
file_obj.close();

// Output is the highest rated contestant
cout << Highest_rated;
return 0;
}

// Driver code
int main()
{
    // Creating object of the class
    Contestant object;

    // Inputting the data
    object.input();

    // Extracting the max rated contestant
    object.output_highest_rated();
}

```

```

        return 0;
    }

// C++ program to Read a record from a File
// using seekg() and tellg()

#include <bits/stdc++.h>
using namespace std;

class student {
    int id;
    char Name[20];

public:
    void display(int K);
};

void student::display(int K)
{
    fstream fs;
    fs.open("student.dat", ios::in | ios::binary);

    // using seekg(pos) method
    // to place pointer at 7th record
    fs.seekg(K * sizeof(student));

    // reading Kth record
    fs.read((char*)this, sizeof(student));

    // using tellg() to display current position
    cout << "Current Position: "
         << "student no: "
         << fs.tellg() / sizeof(student) + 1;

    // using seekg() place pointer at end of file
    fs.seekg(0, ios::end);

    cout << " of "
         << fs.tellg() / sizeof(student)
         << endl;
    fs.close();
}

```

```
// Driver code
int main()
{

    // Record number of the student to be read
    int K = 7;

    student s;
    s.display(K);

    return 0;
}
```

### **The C++ Standard Template Library (STL)**

The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators. It is a generalized library and so, its components are parameterized. Working knowledge of template classes is a prerequisite for working with STL.

Some of the key components of the STL include:

1. Containers: The STL provides a range of containers, such as vector, list, map, set, and stack, which can be used to store and manipulate data.
2. Algorithms: The STL provides a range of algorithms, such as sort, find, and binary\_search, which can be used to manipulate data stored in containers.
3. Iterators: Iterators are objects that provide a way to traverse the elements of a container. The STL provides a range of iterators, such as forward\_iterator, bidirectional\_iterator, and random\_access\_iterator, that can be used with different types of containers.
4. Function Objects: Function objects, also known as functors, are objects that can be used as function arguments to algorithms. They provide a way to pass a function to an algorithm, allowing you to customize its behavior.
5. Adapters: Adapters are components that modify the behavior of other components in the STL. For example, the reverse\_iterator adapter can be used to reverse the order of elements in a container.

#### **1. Algorithms**

The header algorithm defines a collection of functions specially designed to be used on a range of elements. They act on containers and provide means for various operations for the contents of the containers.

- Algorithm

- Sorting
- Searching
- Important STL Algorithms
- Useful Array algorithms
- Partition Operations
- Numeric
  - valarray class

## 2. Containers

Containers or container classes store objects and data. There are in total seven standards “first-class” container classes and three container adaptor classes and only seven header files that provide access to these containers or container adaptors.

- Sequence Containers: implement data structures that can be accessed in a sequential manner.
  - vector
  - list
  - deque
  - arrays
  - forward\_list( Introduced in C++11)
- Container Adaptors: provide a different interface for sequential containers.
  - queue
  - priority\_queue
  - stack
- Associative Containers: implement sorted data structures that can be quickly searched ( $O(\log n)$  complexity).
  - set
  - multiset
  - map
  - multimap
- Unordered Associative Containers: implement unordered data structures that can be quickly searched

## 3. Functors

The STL includes classes that overload the function call operator. Instances of such classes are called function objects or functors. Functors allow the working of the associated function to be customized with the help of parameters to be passed. **Must Read – Functors**

## 4. Iterators

As the name suggests, iterators are used for working on a sequence of values. They are the major feature that allows generality in STL. **Must Read – Iterators**