1. Implement a class Complex which represents the Complex Number data type. Implement the following 1. Constructor (including a default constructor which creates the complex number 0+0i).
2. Overload operator+ to add two complex numbers.
3. Overload operator* to multiply two complex numbers.
4. Overload operators << and >> to print and read Complex Numbers give the code small and easy and the simplest

```
#include <iostream>
using namespace std;
class complex{
  private:
  double real,img;
  public:
    complex(double r=0, double i=0){
      real = r;
      img = i;
    }
    complex operator+(const complex& other)const{
      return{real + other.real, img + other.img};
    }
    complex operator*(const complex& other)const{
      return{real * other.real - img * other.img , real * other.img + img * other.real};
    }
    friend ostream& operator<<(ostream& out, const complex& c){
      if(c.img >= 0)
        out << c.real << " + " << c.img << "i";
      else
        out << c.real << " - " << -c.img << "i";
      return out;
    }
    friend istream& operator>>(istream& in, complex& c){
      cout << "Enter real part : ";
      in >> c.real;
      cout << "Enter imaginary part : ";
      in >> c.img;
      return in;
    }
};
int main(){
  complex c1,c2;
  cin >> c1 >> c2;
  cout << "Sum : " << (c1+c2) << " product : " << (c1*c2) << endl;
  return 0;
}
```

2. Experiment Number 2 : Develop a program in C++ to create a database of student's information system containing the following information: Name, Roll number, Class, Division,Date of Birth, Blood group, Contactaddress, Telephone number, Driving license no. and other.
Construct the database with suitable member functions. Make use of constructor, default constructor, copy constructor, destructor, static member functions, friend class, this pointer, inline function and dynamic memory allocation operators-new and delete as well as exception handling.

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

class StudData;

class Student {
   string name, cls, dob, bloodgroup;
   int roll_no;
char* division;
   static int count;

public:
   Student() : name(""), roll_no(0), cls(""), dob("dd/mm/yyyy"), bloodgroup("") {
division = new char[2]; // Minimum size for string division
   }

   ~Student() {
delete[] division;
   }

   static int getCount() { return count; }

   void getData(StudData*);
void dispData(StudData*) const;
};

class StudData {
string caddress;
long int* telno;
   long int* dlno;

   friend class Student;

public:
   StudData() : caddress(""), telno(new long), dlno(new long) {}
~StudData() { delete telno; delete dlno; }

   void getStudData() {
      cout << "Enter Contact Address: "; cin.ignore(); getline(cin, caddress);
cout << "Enter Telephone Number: "; cin >> *telno;        cout << "Enter
Driving License Number: "; cin >> *dlno;
```

```
    }

    void dispStudData() const {
        cout << "Contact Address: " << caddress << endl;
cout << "Telephone: " << *telno << endl;         cout <<
"Driving License No: " << *dlno << endl;
    }
};

inline void Student::getData(StudData* st) {     cout <<
"Enter Name: "; getline(cin, name);     cout << "Enter
Roll No: "; cin >> roll_no;     cout << "Enter Class: ";
cin.ignore(); getline(cin, cls);     cout << "Enter
Division: "; cin >> division;     cout << "Enter DOB: ";
cin.ignore(); getline(cin, dob);     cout << "Enter Blood
Group: "; cin >> bloodgroup;     st->getStudData();
count++;
}

inline void Student::dispData(StudData* st1) const {
    cout << "Name: " << name << "\nRoll No: " << roll_no << "\nClass: " << cls
        << "\nDivision: " << division << "\nDOB: " << dob << "\nBlood Group: " << bloodgroup <<
endl;
    st1->dispStudData();
}

int Student::count = 0;

int main() {
    Student* stud1[100];
StudData* stud2[100];     int
n = 0;
    char ch;

    do {
        stud1[n] = new Student;
stud2[n] = new StudData;
stud1[n]->getData(stud2[n]);
n++;
        cout << "Add another student (y/n): "; cin >> ch;
    } while (ch == 'y' || ch == 'Y');

    for (int i = 0; i < n; i++) {
        cout << "------------------------------------------------------------\n";         stud1[i]-
>dispData(stud2[i]);
    }

    cout << "------------------------------------------------------------\n";
```

```
    cout << "Total Students: " << Student::getCount() << "\n----------------------------------------------
-------------\n";

    for (int i = 0; i < n; i++) {
        delete stud1[i];
delete stud2[i];
    }

    return 0;
}
```

3. Imagine a publishing company which does marketing for book and audiocassette versions. Create a class publication that stores the title (a string) and price (type float) of a publication. From this class derive two classes: book, which adds a page count(type int), and tape, which adds a playing time in minutes(type float). Write a program that instantiates the book and tape classes, allows user to enter data and displays the data members. If an exception is caught, replace all the data member values with zero values.

```
#include <iostream>
#include <string>  // Correct spacing and ordering of header files
using namespace std;
class Publication {
  protected:
    string title;  // Title of the publication
    float price;   // Price of the publication
  public:
    // Method to get publication details
    void get() {
      cout << "Enter Title: ";
      cin.ignore(); // Ignore leftover input from previous cin
      getline(cin, title); // Allows multi-word titles
      cout << "Enter Price: ";
      cin >> price;
    }
    // Method to display publication details
    void put() const {
      cout << "Title: " << title << endl;
      cout << "Price: " << price << endl;
    }
};
class Book : public Publication {
  private:
    int pageCount; // Number of pages in the book
  public:
    // Method to get book details
    void get() {
      Publication::get(); // Call base class get()
      cout << "Enter Page Count: ";
      cin >> pageCount;
```

```
    }
    // Method to display book details
    void put() const {
      Publication::put(); // Call base class put()
      cout << "Page Count: " << pageCount << endl;
    }
};
class Tape : public Publication {
  private:
    float playTime; // Play time of the tape in minutes
  public:
    // Method to get tape details
    void get() {
      Publication::get(); // Call base class get()
      cout << "Enter Play Time (in minutes): ";
      cin >> playTime;
    }
    // Method to display tape details
    void put() const {
      Publication::put(); // Call base class put()
      cout << "Play Time: " << playTime << " minutes" << endl; // Corrected spelling from 'Plat
Time'
    }
};
int main() {
  Book b;
  Tape t;
  // Get and display book details
  cout << "Enter Book Details: " << endl;
  b.get();
  cout << "--- Book Information ---" << endl;
  b.put();
  // Get and display tape details
  cout << "\nEnter Tape Details: " << endl;
  t.get();
  cout << "--- Tape Information ---" << endl;
  t.put();
  return 0;
}
```

4. Write a C++ program that creates an output file, writes information to it, closes the file, open it again as an input file and read the information from the file.

```
#include <iostream>
#include <fstream>  // For file handling

using namespace std;

int main() {
   // Create and write to a file
   ofstream outFile("example.txt");  // Open file in write mode
if (outFile) {
      outFile << "Hello, this is a simple file handling example." << endl;
outFile << "This file contains two lines of text." << endl;        outFile.close();
// Close the file
   } else {
      cerr << "Error opening file for writing." << endl;
return 1;  // Exit with an error code
   }

   // Open and read from the file
   ifstream inFile("example.txt");  // Open file in read mode
if (inFile) {        string line;
      while (getline(inFile, line)) {  // Read line-by-line
cout << line << endl;  // Display the content of the file
      }
      inFile.close();  // Close the file
   } else {
      cerr << "Error opening file for reading." << endl;
      return 1;  // Exit with an error code
   }

   return 0;
}
```

5. Write a function template for selection sort that inputs, sorts and outputs an integer array and a float array.

```
#include <iostream>
#include <fstream>  // For file handling using
namespace std;

// Function template for selection sort
template <typename T> void
selectionSort(T arr[], int size) {    for
(int i = 0; i < size - 1; ++i) {        int
minIndex = i;        for (int j = i + 1; j
< size; ++j) {           if (arr[j] <
```

```
arr[minIndex]) {                minIndex
= j;
            }
        }
        swap(arr[i], arr[minIndex]);
    }
}

// Function to print the array
template <typename T> void
printArray(T arr[], int size) {
for (int i = 0; i < size; ++i) {
cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {     //
Integer array
    int intArray[] = {64, 25, 12, 22, 11};
    int intSize = sizeof(intArray) / sizeof(intArray[0]);

    cout << "Original integer array: \n";
    printArray(intArray, intSize);

    selectionSort(intArray, intSize);

    cout << "Sorted integer array: \n";
    printArray(intArray, intSize);

    // Float array     float floatArray[] = {64.5, 25.1, 12.3,
22.8, 11.7};     int floatSize = sizeof(floatArray) /
sizeof(floatArray[0]); cout << "\nOriginal float array: \n";
    printArray(floatArray, floatSize);

    selectionSort(floatArray, floatSize);

    cout << "Sorted float array: \n";
    printArray(floatArray, floatSize);

    return 0;
}
```

6. Write C++ Program using STL for sorting and searching user defined records such as item records using vector container.

```
#include <iostream>
#include <vector>
#include <algorithm>  // For sort and find_if
```

```
using namespace std;

// Structure to hold item record
struct Item {     int id;     string
name;
    float price;

    // Overload the < operator for sorting
bool operator<(const Item& other) const {
return price < other.price;  // Sort by price     }
};

// Function to display the list of items void
displayItems(const vector<Item>& items) {
    for (const auto& item : items) {         cout << "ID: " << item.id << ", Name: " << item.name
<< ", Price: " << item.price << endl;
    }
}

int main() {
    vector<Item> items = {
{101, "ItemA", 29.99},
        {102, "ItemB", 19.99},
        {103, "ItemC", 49.99},
        {104, "ItemD", 39.99}
    };

    // Display original list of items
cout << "Original List of Items:\n";
displayItems(items);

    // Sort the items by price
    sort(items.begin(), items.end());

    // Display sorted list of items
    cout << "\nSorted List of Items (by price):\n";
displayItems(items);

    // Search for an item by name
string searchName;
    cout << "\nEnter the name of the item to search for: ";
cin >> searchName;

    auto it = find_if(items.begin(), items.end(), [&searchName](const Item& item) {
return item.name == searchName;
    });

    if (it != items.end()) {
```

```
        cout << "\nItem found: ID: " << it->id << ", Name: " << it->name << ", Price: " << it->price
<< endl;     } else {
        cout << "\nItem not found." << endl;
    }

    return 0;
}
```

7. Write a program in C++ to use map associative container. The keys will be the names of states and the values will be the populations of the states. When the program runs, the user is prompted to type the name of a state. The program then looks in the map, using the state name as an index and returns the population of the state.

```
#include <iostream>
#include <map>  // For map container
#include <string>

using namespace std;

int main() {
    // Create a map to store state names as keys and populations as values
map<string, int> statePopulation = {
        {"California", 39538223},
        {"Texas", 29145505},
        {"Florida", 21538187},
        {"New York", 20201249},
        {"Pennsylvania", 13002700}
    };

    string stateName;
    cout << "Enter the name of the state to search for its population: ";
cin >> stateName;

    // Search for the state in the map
    auto it = statePopulation.find(stateName);
    if (it != statePopulation.end()) {
        cout << "The population of " << stateName << " is " << it->second << "." << endl;
    } else {
        cout << "State not found in the map." << endl;
    }

    return 0;
}
```