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

int main() {
    // Create an output file stream object to write data
    ofstream outFile;
    outFile.open("output.txt");

    // Check if the file was successfully opened
    if (!outFile) {
        cerr << "Error opening the output file!" <<endl;
        return 1;
    }

    // Write some information to the file
    outFile << "Hello, this is a test message!" <<endl;
    outFile << "This is the second line of the file." <<endl;

    // Close the output file
    outFile.close();

    // Create an input file stream object to read data
    ifstream inFile;
    inFile.open("output.txt");

    // Check if the file was successfully opened for reading
    if (!inFile) {
        cerr << "Error opening the input file!" << endl;
        return 1;
    }

    // Read and display the content from the file
    string line;
    while (getline(inFile, line)) {
        cout << line <<endl; // Output the content to the console
    }

    // Close the input file
    inFile.close();

    return 0;
}
```

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

// Class definition for Student
class Student {
    string name, dob, bloodGroup, address, licenseNo;
    int rollNumber, stdClass;
    char division;
    long long telephoneNumber;

    // Static member to count number of students
    static int studentCount;

public:
    // Default constructor
    Student() {
        name = "";
        rollNumber = 0;
        stdClass = 0;
        division = 'A';
        dob = "";
        bloodGroup = "";
        address = "";
        telephoneNumber = 0;
        licenseNo = "";
        studentCount++;
    }

    // Parameterized constructor
    Student(string name, int rollNumber, int stdClass, char division, string dob,
            string bloodGroup, string address, long long telephoneNumber, string licenseNo) {
        this->name = name;
        this->rollNumber = rollNumber;
        this->stdClass = stdClass;
        this->division = division;
        this->dob = dob;
        this->bloodGroup = bloodGroup;
        this->address = address;
        this->telephoneNumber = telephoneNumber;
        this->licenseNo = licenseNo;
        studentCount++;
    }

    // Copy constructor
    Student(const Student &s) {
        name = s.name;
        rollNumber = s.rollNumber;
        stdClass = s.stdClass;
        division = s.division;
        dob = s.dob;
        bloodGroup = s.bloodGroup;
        address = s.address;
        telephoneNumber = s.telephoneNumber;
        licenseNo = s.licenseNo;
        studentCount++;
    }

    // Destructor
    ~Student() {
        studentCount--;
    }
    // Inline function to display student data
    inline void display() const {
        cout << setw(15) << name << setw(10) << rollNumber << setw(10) << stdClass << setw(10)
             << division << setw(15) << dob << setw(15) << bloodGroup << setw(20) << address
             << setw(15) << telephoneNumber << setw(15) << licenseNo << endl;
    }

    // Friend class to access private data
    friend class Admin;

    // Static function to get student count
    static int getStudentCount() {
        return studentCount;
    }
};

// Initialize static member
int Student::studentCount = 0;

// Admin class to manage student database
class Admin {
public:
    // Function to display student details (using friend class access)
    static void showStudent(const Student &s) {
        cout << "Admin View: " << endl;
        cout << "Name: " << s.name << ", Roll Number: " << s.rollNumber << endl;
    }
};

// Main function
int main() {
    try {
        // Dynamic memory allocation for students
        Student *students = new Student[3];

        // Adding student details
        students[0] = Student("Alice", 1, 10, ’A’, "01-01-2005", "O+", "123 Lane", 1234567890, "DL12345");
        students[1] = Student("Bob", 2, 10, ’B’, "02-02-2005", "A+", "456 Street", 9876543210, "DL67890");

        // Copy constructor example
        students[2] = students[0];

        // Display student details
        cout << setw(15) << "Name" << setw(10) << "Roll No" << setw(10) << "Class" << setw(10)
             << "Div" << setw(15) << "DOB" << setw(15) << "Blood Group" << setw(20)
             << "Address" << setw(15) << "Phone No" << setw(15) << "License No" << endl;
        cout << string(130, ’-’) << endl;

        for (int i = 0; i < 3; i++) {
            students[i].display();
        }

        // Static member function usage
        cout << "\nTotal Students: " << Student::getStudentCount() << endl;
        // Friend class usage
        Admin::showStudent(students[1]);

        // Free allocated memory
        delete[] students;
    } catch (const bad_alloc &e) {
        cerr << "Memory allocation failed: " << e.what() << endl;
    }

    return 0;
}

#include<iostream>
using namespace std;
class Complex
{
    double r;
    double i;
    public:
    Complex()
    {
        r=0;
        i=0;
    }
    Complex operator + (Complex);
    Complex operator * (Complex);
    friend istream &operator >> (istream &in , Complex &t)
    {
        cout<<"Enter the real part :\n";
        in>>t.r;
        cout<<"Enter the imaginary part :\n";
        in>>t.i;
    }
    friend ostream &operator <<(ostream &out , Complex &t)
    {
        out<<t.r<<"+"<<t.i<<"i"<<endl;
    }
};
Complex Complex :: operator + (Complex c)
{
    Complex temp;
    temp.r=r+c.r;
    temp.i=i+c.i;
    return temp;
}
Complex Complex :: operator * (Complex c)
{
    Complex temp2;
    temp2.r=(r*c.r)-(i*c.i);
    temp2.i=(i*c.r)+(r*c.i);
    return temp2;
}
int main()
{
    Complex c1,c2,c3,c4;
    cout<<"Default Values :";
    cout<<c1;
    cout<<"Enter first number : \n";
    cin>>c1;
    cout<<"Enter second number : \n";
    cin>>c2;
    c3=c1+c2;
    c4=c1*c2;
    cout<<"First Number :"<<endl;
    cout<<c1;
    cout<<"Second Number : "<<endl;
    cout<<c2;
    cout<<"Addition : "<<endl;
    cout<<c3;
    cout<<"Multiplication : "<<endl;
    cout<<c4;
    return 0;

}


#include<iostream>
#include<map>
using namespace std;
int main()
{
    typedef map<string,int> mapType;
    mapType populationMap;
    populationMap.insert(pair<string,int>("Maharashtra",458888));
    populationMap.insert(pair<string,int>("kerala",84935));
    populationMap.insert(pair<string,int>("West Bengal",65469962));
    populationMap.insert(pair<string,int>("Rajasthan",245276));
    mapType :: iterator it;
    cout<<"********Population of States in India**********"<<endl;
    cout<<"Size of populationMap : "<<populationMap.size()<<endl;
    string state_name;
    cout<<"Enter the state name : "<<endl;
    cin>>state_name;
    it=populationMap.find(state_name);
    if(it!=populationMap.end())
    {
        cout<<state_name<<" ’s population is "<<it->second<<endl;
    }
    else
    {
        cout<<"Key is not present in PopulationMap "<<endl;
    }
    return 0;
}

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

```
struct Item {
    int code;
    string name;
    double cost;
    int quantity;
};

// Function to display all items
void displayItems(const vector<Item>& items) {
    for (const auto& item : items) {
        cout << "Code: " << item.code
             << ", Name: " << item.name
             << ", Cost: " << item.cost
             << ", Quantity: " << item.quantity << endl;
    }
}

int main() {
    vector<Item> items = {
        {101, "Pen", 10.5, 100},
        {102, "Notebook", 45.0, 50},
        {103, "Pencil", 5.0, 200},
        {104, "Eraser", 3.0, 300}
    };

    // Sort items by cost
    sort(items.begin(), items.end(), [](const Item& a, const Item& b) {
        return a.cost < b.cost;
    });

    cout << "Items after sorting by cost:" << endl;
    displayItems(items);

    // Search for an item by code
    int searchCode;
    cout << "\nEnter item code to search: ";
    cin >> searchCode;

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

    if (it != items.end()) {
        cout << "Item found: Code: " << it->code
             << ", Name: " << it->name
             << ", Cost: " << it->cost
             << ", Quantity: " << it->quantity << endl;
    } else {
        cout << "Item with code " << searchCode << " not found." << endl;
    }

    return 0;
}
```

```
#include<iostream>
using namespace std;
#define size 10
int n;
template<class T>
void selection(T A[size])
{
    int i,j,min;
    T temp;
    for( i=0;i<=n-2;i++)
    {
        min=i;
        for(j=i+1;j<=n-1;j++)
        {
            if(A[j]<A[min])
            {
                min=j;
            }
        }
        temp=A[i];
        A[i]=A[min];
        A[min]=temp;
    }
    cout<<"Sorted Lists"<<endl;
    for(int i=0;i<n;i++)
    {
        cout<<A[i]<<"  ";
    }
    cout<<endl;
}
int main()
{
    int A[size];
    float B[size];
    cout<<"Integer Elements :\n";
    cout<<"How many elements do you want to enter ?\n";
    cin>>n;
    cout<<"Enter the integer elements :\n";
    for(int i=0;i<n;i++)
    {
        cin>>A[i];
    }
    selection(A);
    cout<<"Float Elements :\n";
    cout<<"How many elements do you want to enter ?\n";
    cin>>n;
    cout<<"Enter the Float elements :"<<endl;
    for(int i=0;i<n;i++)
    {
        cin>>B[i];
    }
    selection(B);
    return 0;
```

```
}

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

// Base class: Publication
class Publication {
protected:
    string title;
    float price;

public:
    Publication() : title(""), price(0.0) {}

    void getData() {
        try {
            cout << "Enter title: ";
            cin.ignore();
            getline(cin, title);

            cout << "Enter price: ";
            cin >> price;
            if (price < 0) {
                throw invalid_argument("Price cannot be negative.");
            }
        } catch (const exception &e) {
            cout << "Error: " << e.what() << endl;
            title = "";
            price = 0.0;
        }
    }

    void putData() const {
        cout << "Title: " << title << endl;
        cout << "Price: " << price << endl;
    }
};

// Derived class: Book
class Book : public Publication {
private:
    int pageCount;

public:
    Book() : pageCount(0) {}

    void getData() {
        try {
            Publication::getData();
            cout << "Enter page count: ";
            cin >> pageCount;
            if (pageCount < 0) {
                throw invalid_argument("Page count cannot be negative.");
            }
        } catch (const exception &e) {
            cout << "Error: " << e.what() << endl;
            title = "";
            price = 0.0;
            pageCount = 0;
        }
    }

    void putData() const {
        Publication::putData();
        cout << "Page Count: " << pageCount << endl;
    }
};

// Derived class: Tape
class Tape : public Publication {
private:
    float playingTime;

public:
    Tape() : playingTime(0.0) {}

    void getData() {
        try {
            Publication::getData();
            cout << "Enter playing time (in minutes): ";
            cin >> playingTime;
            if (playingTime < 0) {
                throw invalid_argument("Playing time cannot be negative.");
            }
        } catch (const exception &e) {
            cout << "Error: " << e.what() << endl;
            title = "";
            price = 0.0;
            playingTime = 0.0;
        }
    }

    void putData() const {
        Publication::putData();
        cout << "Playing Time (minutes): " << playingTime << endl;
    }
};

int main() {
    Book myBook;
    Tape myTape;

    cout << "Enter details for Book:\n";
    myBook.getData();

    cout << "\nEnter details for Tape:\n";
    myTape.getData();
    cout << "\nBook Details:\n";
    myBook.putData();

    cout << "\nTape Details:\n";
    myTape.putData();

    return 0;
}
```