- Create a class named TestClass that holds a single private integer field and a public constructor. The only statement in the constructor is one that displays the message “Constructing”. Write a main()function that instantiates one object of the TestClass. Save the file as **TestClass.cpp**. Run the program and observe the results.

- Write another main()function that instantiates an array of 10 TestClass objects. Save the file as **TestClassArray.cpp**. Run this program and observe the results.

TestClass.cpp

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

class TestClass{
    private:
        int integer;
    public:
        TestClass(){
            cout << "Constructing" << endl;
        }
};

int main(){
    // test TestClass
    TestClass object;
    return 0;
}
/* OUTPUT
    PS .\GU\OOP\LEC\Assignments\#3\code> c++ .\TestClass.cpp -o TestClass.exe; .\TestClass.exe
    Constructing
*/
```

TestClassArray.cpp

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

class TestClass{
    private:
        int integer;
    public:
        int test;
        TestClass(){
            cout << "Constructing" << endl;
        }
};

int main(){
    TestClass data[10];
    return 0;
}

/* OUTPUT
    PS GU\OOP\LEC\Assignments\#3\code> c++ .\TestClass.cpp -o TestClass.exe; .\TestClass.exe
    Constructing
    Constructing
    Constructing
    Constructing
    Constructing
    Constructing
    Constructing
    Constructing
    Constructing
    Constructing
*/
```

2. Write the class definition for a Date class that contains three integer data members: month, day, and year. Include a default constructor that assigns the date 1/1/2000 to any new object that does not receive arguments. Also include a function that displays the Date object. Write a main()function in which you instantiate two Date objects—one that you create using the default constructor values, and one that you create using three arguments—and display its values. Save the file as **Date.cpp**.

Date.cpp

```cpp
#include <iostream>
using namespace std;
#define usi unsigned short int

class Date{
    private:
        usi _day, _month, _year;
    public:
        // Constrcutors
        Date(): _day(1), _month(1), _year(2000){}
        Date(usi day, usi month, usi year): _day(day), _month(month), _year(year){}

        // Functionality
        void show(){
            printf("Date(day = %i, month = %i, year = %i)\n", _day, _month, _year);
        }
};

int main(){
    // Result
    Date object1, object2(13, 5, 2005);
    object1.show();
    object2.show();
    return 0;
}

/* OUTPUT
    PS GU\OOP\LEC\Assignments\#3\code> c++ .\Date.cpp -o Date.exe; .\Date.exe
    Date(day = 1, month = 1, year = 2000)
    Date(day = 13, month = 5, year = 2005)
*/
```

3- Create a Person class that includes fields for last name, first name, and zip code. Include a default constructor that initializes last name, first name, and zip code to “X” if no arguments are supplied. Also include a display function. Write a main()function that instantiates and displays two Person objects: one that uses the default values, and one for which you supply your own values. Save the file as **Person.cpp**.

Person.cpp

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

class Person{
    private:
        string _name, _lastName, _zipCode;
    public:
        // Constructors
        Person() : _name("X"), _lastName("X"), _zipCode("X") {}
        Person(string name, string lastName, string zipCode) : _name(name), _lastName(lastName), _zipCode(zipCode){}

        // Functionality
        void show(){
            printf("Person(name = %s, lastName = %s, code = %s)\n", _name.data(), _lastName.data(), _zipCode.data());
        }
};


int main(){
    Person defaultPerson, mahros("mahros", "alqabasy", "K205"s);
    defaultPerson.show();
    mahros.show();
    return 0;
}

/* OUTPUT
    PS GU\OOP\LEC\Assignments\#3\code> c++ .\Person.cpp -o Person.exe; .\Person.exe
    Person(name = X, lastName = X, code = X)
    Person(name = mahros, lastName = alqabasy, code = K205)
*/
```

4- Create a class named SavingsAccount. Provide fields for the customer (use the Person class from Exercise 3), account number, balance, and interest rate. Provide two constructors. One requires a customer and sets each numeric field to 0; the other requires a customer and an account number and sets the account's opening balance to $100 at 3% interest. Include
a function that displays an account's data fields. Write a main()function that instantiates one SavingsAccount object of each type and displays their values. Save the file as **SavingsAccount.cpp**.

SavingAccount.cpp

```cpp
#include <iostream>
using namespace std;
#define ld long double
#define lli long long int
#define usi unsigned short int

class SavingsAccount{
    private:
        string _name, _lName, _code;
        usi _intrest; lli _account; ld _balance;
    public:
        // Constructors
        SavingsAccount() : _account(0), _balance(0), _intrest(0){}
        SavingsAccount(string name, string lastName, string zipCode, lli account) : _name(name), _lName(lastName), _code(zipCode), _account(account),
         _balance(100), _intrest(3){}

        // Functionality
        void show(){
            printf("Person(name = %s, account = %lli, balance = $%.2lf, rate = %i%)\n", _name.data(), _account, _balance, _intrest);
        }
};


int main(){
    SavingsAccount defaultAccount, account1("mahros", "alqabasy", "K205", 223106831);
    defaultAccount.show();
    account1.show();
    return 0;
}

/* OUTPUT
    PS GU\OOP\LEC\Assignments\#3\code> c++ .\SavingAccount.cpp -o SavingAccount.exe; .\SavingAccount.exe
    Person(name = , account = 0, balance = $0.00, rate = 0%)
    Person(name = mahros, account = 223106831, balance = $0.00, rate = 3%)
*/
```

5- Create a class named Car. The Car class contains a static integer field named count. Create a constructor that adds 1 to the count and displays the count each time a Car is created. Create a destructor that subtracts 1 from the count and displays the count each time a Car goes out of scope. Write a main()function that declares an array of five Car objects. Output consists of five constructor messages and five destructor messages, each displaying the current count, similar to the output in Figure 8-34. Save the file as Car.cpp.

Car.cpp

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

class Car{
    private:
        static long long int _count;
    public:
        // Constructors
        Car(){
            _count++;
            printf("Cars(count = %li)\n", _count);
        }

        // Destructors
        ~Car(){
            _count--;
            printf("Cars(count = %li)\n", _count);
        }
};

long long int Car::_count = 0;

int main(){
    Car cars[5];
    return 0;
}

/* OUTPUT
    PS GU\OOP\LEC\Assignments\#3\code> c++ .\Car.cpp -o Car.exe; .\Car.exe
    Cars(count = 1)
    Cars(count = 2)
    Cars(count = 3)
    Cars(count = 4)
    Cars(count = 5)
    Cars(count = 4)
    Cars(count = 3)
    Cars(count = 2)
    Cars(count = 1)
    Cars(count = 0)
*/
```

6- Create two classes. The first holds customer data—specifically, a customer number and zip code. The second, a class for cities, holds the city name, state, and zip code. Additionally, each class contains a constructor that takes parameters to set the field values. Create a friend function that displays a customer number and the customer's city, state, and zip code. Write a brief main()function to test the classes and friend function. Save the file as Customer.cpp.

Customer.cpp

```cpp
#include <iostream>
using namespace std;
#define ll long long

// Forward Declaration
class City;
class Customer{
    private:
        string _zip;
        ll int _number;
    public:
        // Constructor
        Customer(ll int number, string zip) : _number(number), _zip(zip){
        }

        // Functionality
        friend void show(Customer customer, City state);
};

class City{
    private:
        string _name, _zip, _state;
    public:
        // Constructor
        City(string name, string zip, string state) : _name(name), _zip(zip), _state(state){
        }

        // Functionality
        friend void show(Customer customer, City state);
};

// Friend Function Implementation
void show(Customer customer, City city){
    printf("Customer(number = %i, city = %s, state = %s, zip = %s)\n", customer._number, city._name.c_str(), city._state.c_str(), city._zip.c_str());
}

int main(){
    Customer customer = {223106831, "22310"}; City city = {"Desouk", "22310", "Kafrelsheikh"};
    show(customer, city);
    return 0;
}

/* OUTPUT
    PS GU\OOP\LEC\Assignments\#3\code> c++ .\Customer.cpp -o Customer.exe; .\Customer.exe
    Customer(number = 223106831, city = Desouk, state = Kafrelsheikh, zip = 22310)
*/
```

7- Create two classes. The first, named Sale, holds data for a sales transaction. Its private data members include the day of the month, amount of the sale, and the salesperson' ID number. The second class, named Salesperson, holds data for a salesperson, and its private data members include each salesperson's ID number and last name. Each class includes a constructor to which you can pass the field values. Create a friend function named display()that is a friend of both classes and displays the date of sale, the amount, and the salesperson ID and name. Write a short main()demonstration program to test your classes and friend function. Save the file as **Sales.cpp**.

- Add a function to both the Sale and Salesperson classes that returns the private salesperson ID number. Write a main()function that contains an array of five Salesperson objects and store appropriate data in it. Then, continue to prompt the user for Sale data until the user enters an appropriate sentinel value. For each Sale transaction entered, determine whether the salesperson's ID number is valid. Either display an error message, or use the friend display()function to display all the data. Save the file as **cpp**.

Sales.cpp

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


#define ll long long
#define usi unsigned short int

class Salesperson;
class Sale {
    private:
        usi _day;
        ll int _id;
        long double _amount;
    public:
        // Constructor
        Sale(ll id, usi day, long double amount) : _id(id), _day(day), _amount(amount) {}

        // Setters

        // Getters
        ll int id() const {
            return _id;
        }

        // Functionality
        friend void display(Sale sale, Salesperson salesperson);
};

class Salesperson {
    private:
        ll int _id;
        string _lastName;
    public:
        Salesperson(ll int id, string lastName) : _id(id), _lastName(lastName) {}

        ll int id() const {
            return _id;
        }

        friend void display(Sale sale, Salesperson salesperson);
};

void display(Sale sale, Salesperson salesperson) {
    printf("Transaction(date = %i, amount = %.2lf, Person(id = %li, name = %s))\n", sale._day, sale._amount, salesperson._id, salesperson._lastName.c_str());
}

int main() {
    vector<Salesperson> SalesPerson = {
        {1, "Mahros"},
        {2, "Mohamed"},
        {3, "Yser"},
        {4, "ALQabasy"},
        {5, "Khaled"}
    };

    while (1) {
        ll int day, id;
        long double amount;
        cout << "Day: ";
        cin >> day;
        if (day == -1){
            break;
        }

        cout << "Amount: ";
        cin >> amount;

        cout << "Person ID: ";
        cin >> id;

        Sale sale(day, amount, id);

        bool CHECK_ID = false;
        for (Salesperson salesperson : SalesPerson) {
            if (salesperson.id() == id) {
                display(sale, salesperson);
                CHECK_ID = true;
                break;
            }
        }

        if (!CHECK_ID) {
            cout << "Error: Invalid salesperson ID." << endl;
        }
    }

    return 0;
}

/* OUTPUT
    PS GU\OOP\LEC\Assignments\#3\code> c++ .\Sales.cpp -o Sales.exe; .\Sales.exe
    Day: 1
    Amount: 10
    Person ID: 102
    Transaction(date = 10, amount = 0.00, Person(id = 102, name = Johnson))

    PS GU\OOP\LEC\Assignments\#3\code> c++ .\Sales.cpp -o Sales.exe; .\Sales.exe
    Day: 20
    Amount: 3500.20
    Person ID: 15
    Error: Invalid salesperson ID.
*/
```