Final Project Documentation

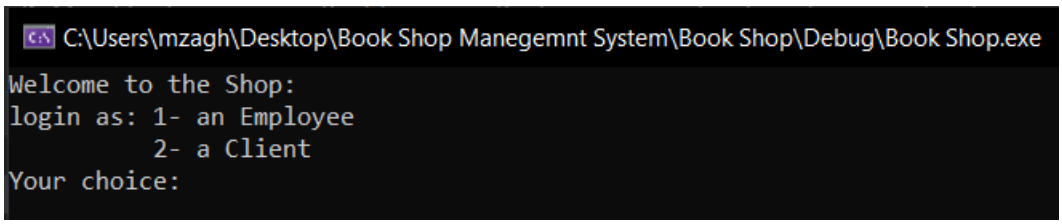
Description of the project: Book Shop Management System

The project is called book shop management system. Basically, it allows to manage the bookstore in different ways: it provides information about the stock of books, prices, authors, names, also data about the employees (ID, name, salary, address, …) and clients (ID, name, loyalty points, …). Furthermore, it allows you to modify and organize all the data so that it can be up to date.

The project contains a class that defines the employees with the necessary data also a class that defines the clients. These two classes are derived from a class called human that contains common methods and attributes between the last two classes. The project also contains a class called book that has all the necessary data of a book. The project has methods that allows to add, arrange, modify, and monitor the stock of books. It will allow to organize or add employees and clients as well.

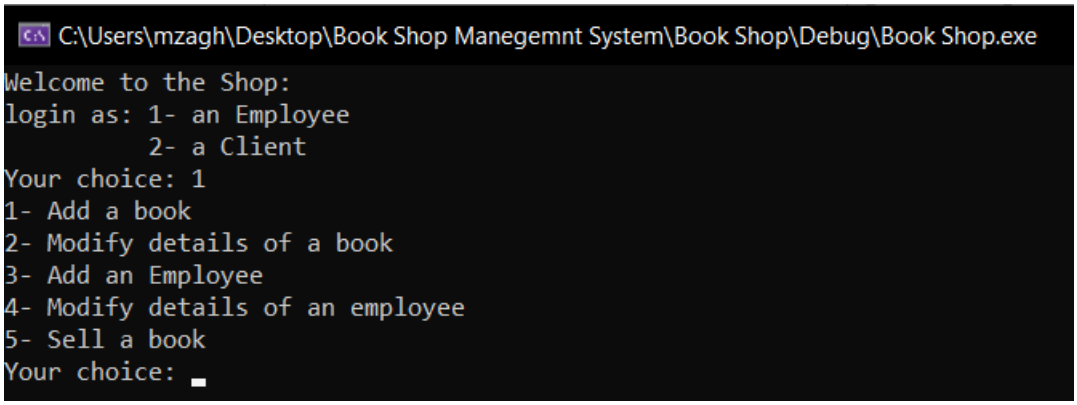
The Document contains I. User Experience and II. Code Explanations.

I. User Experience:



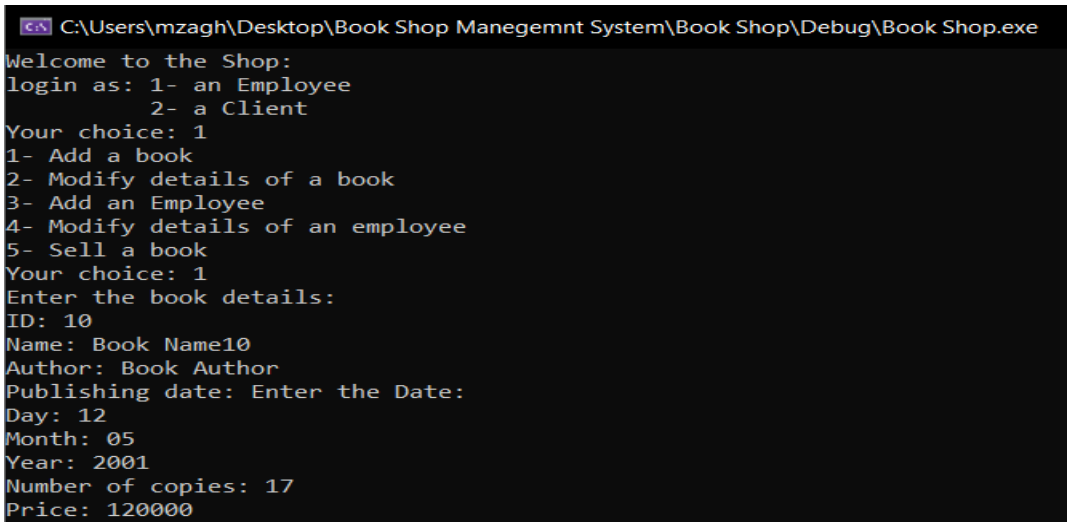
The program will start to ask the User to select whether he is an employee or a client.

CASE 1: login as an Employee:



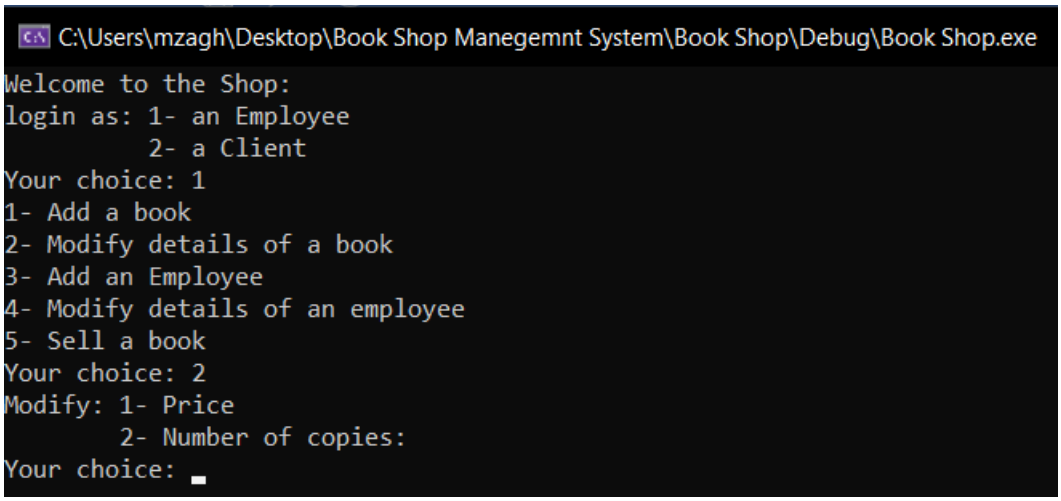
After entering 1 as a choice the program will display certain options for the employee.

*****If the employee chooses 1:**

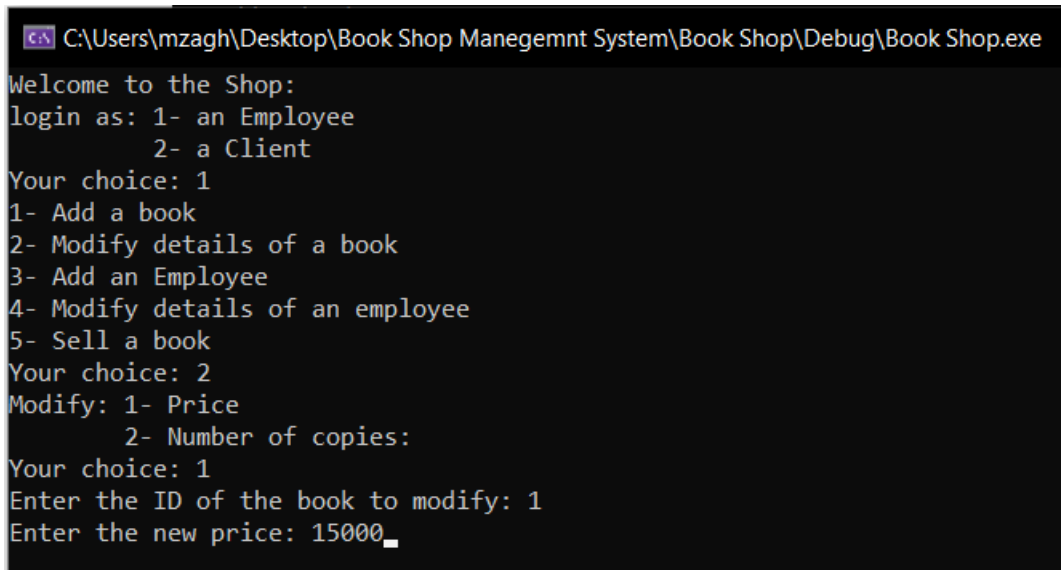


After choosing to Add a book, the program will ask the Employee to enter some details concerning the book to add: Id, name, author name, publishing date, Number of copies and Price.

*****If the employee chooses 2:**

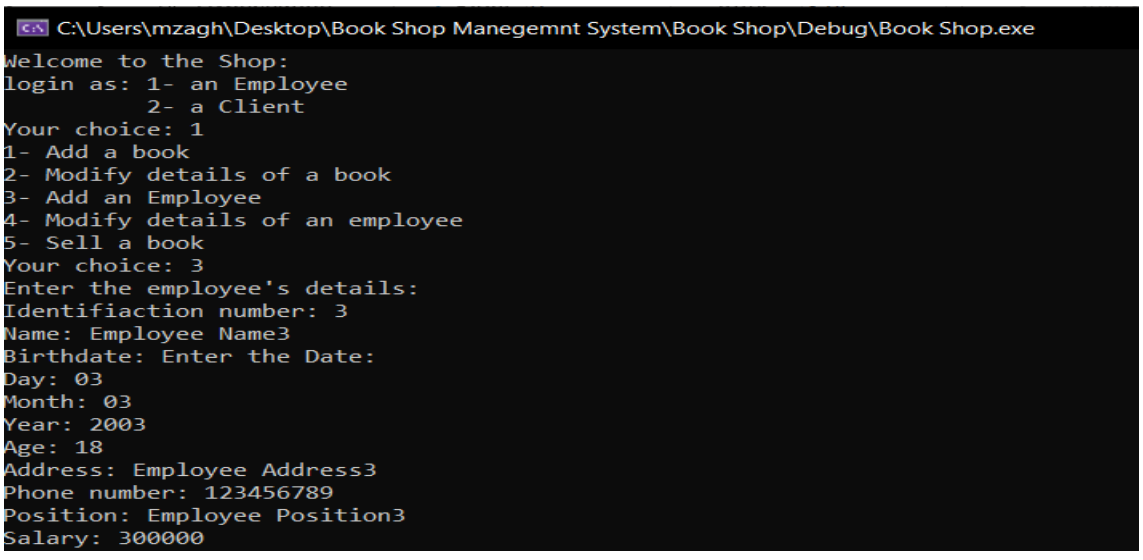


After choosing to Modify details of a book, the Employee will be asked to choose what he wants to modify.



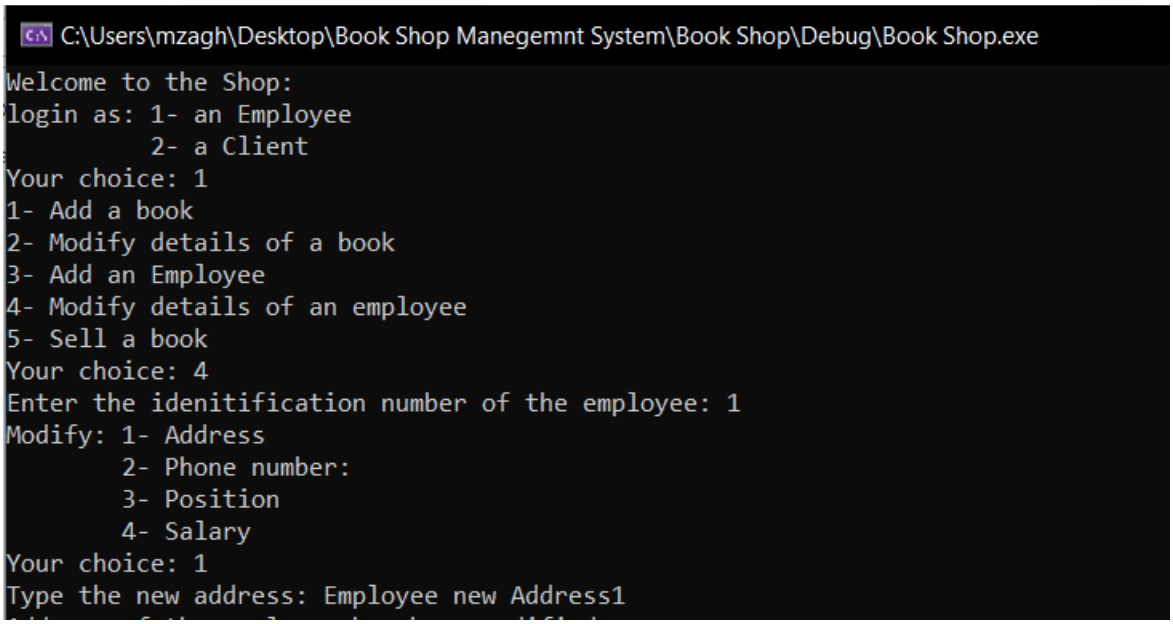
If he chose to modify the price, first he will be asked to enter the Id of the book to modify and after that the new price (It is the same if he chose to modify the Number of copies).

*****If the employee chooses 3:**



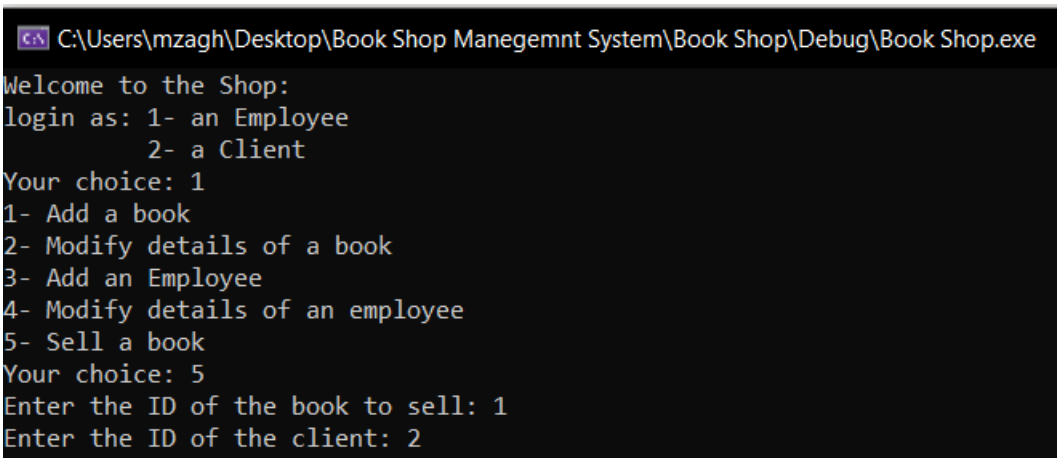
If the employee chose to add an employee, he will be asked to enter details about the new employee. He will be asked to enter: ID, name, birthdate, age, address, phone number, position, and salary.

*****If the employee chooses 4:**



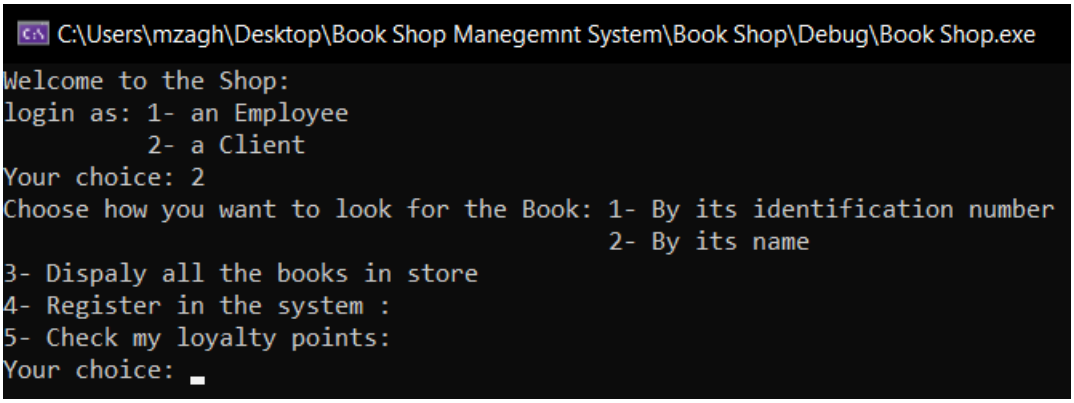
After choosing to modify the details of an Employee, the user will be asked to choose what kind of information he wants to change. There will be 4 options: address, phone number, position, and salary. After choosing one of these options, he will be asked to enter the ID of the employee to modify and to enter the new value of the variable to change.

*****If the employee chooses 5:**



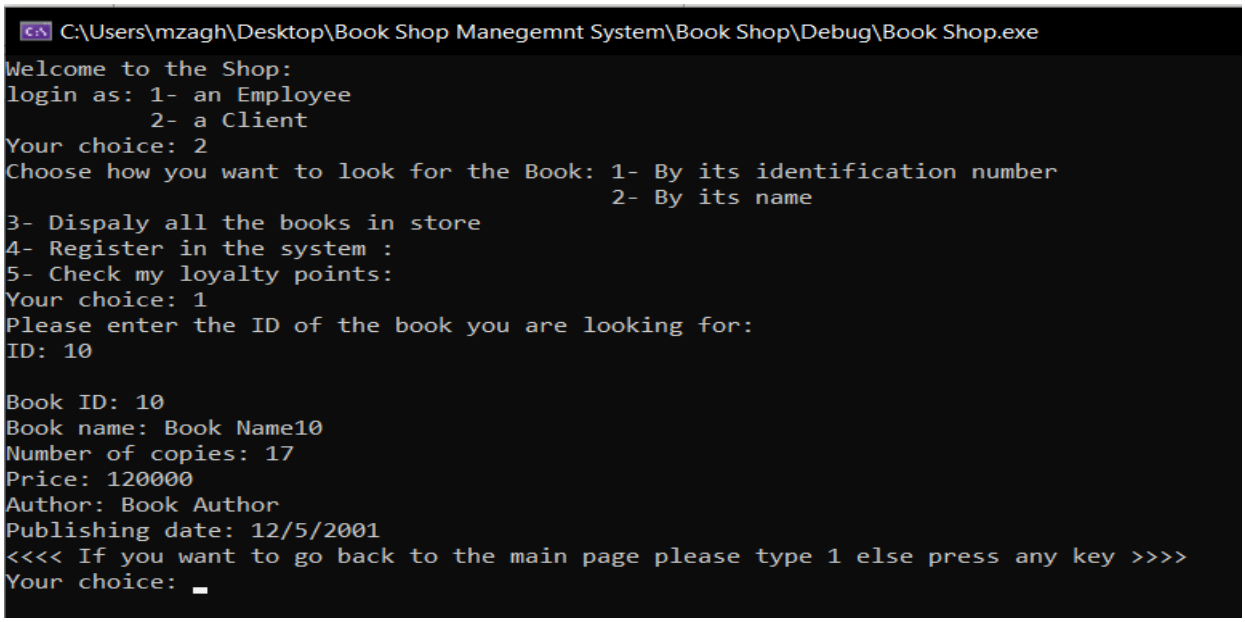
If the user chose 5, he will be asked to enter the Id of the book to check its number of copies and the Id of the client to update his loyalty points.

CASE 2: login as a Client:



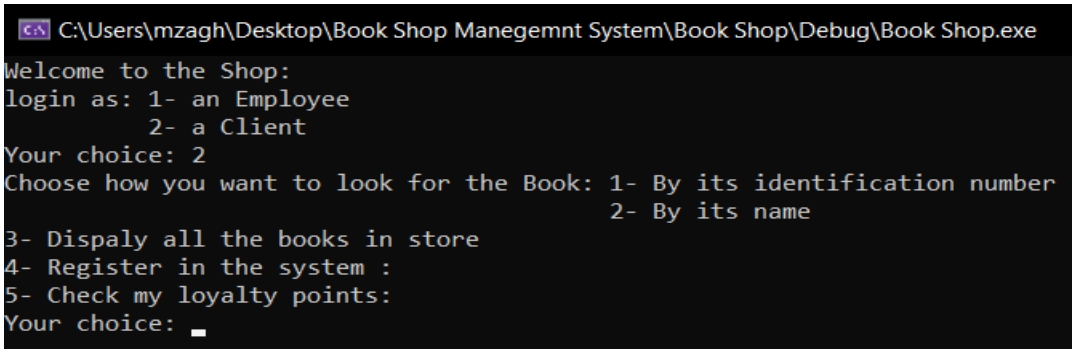
In case, the user chose to login as a client, he will get different choices as shown in the photo.

*****If the client chooses 1 or 2:**



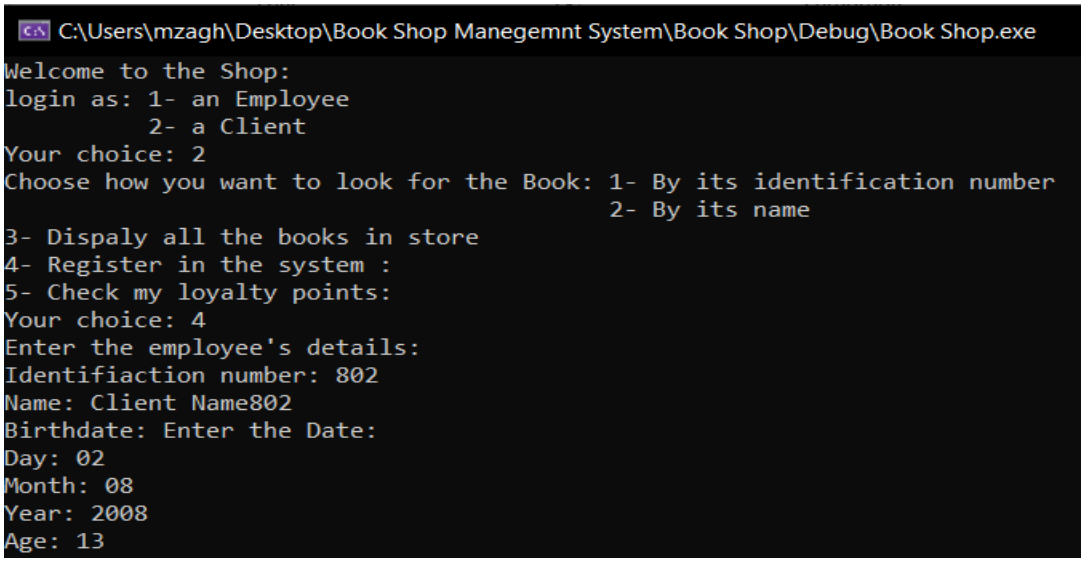
If the client chose to look for the book by its ID, he will be asked to enter the identification number and the book details will be displayed on the screen as shown in the photo. If he chose to look for the book by its name, he will be asked to enter his name.

*****If the client chooses 3:**



If the client typed, all the books in the store will be displayed on the screen so that he can order the book through its ID.

*****If the client chooses 4:**



After choosing to register in the system, the client will be asked to enter his personal information. He will have to enter his: ID, name, birthdate, and age.

*****If the client chooses 5:**

```
C:\Users\mzagh\Desktop\Book Shop Manegemnt System\Book Shop\Debug\Book Shop.exe
Welcome to the Shop:
login as: 1- an Employee
          2- a Client
Your choice: 2
Choose how you want to look for the Book: 1- By its identification number
                                          2- By its name
3- Dispaly all the books in store
4- Register in the system :
5- Check my loyalty points:
Your choice: 5
Enter your identification number: 16
You have 8 loyalty points
you have a discount on your next purchase.
```

If the Client chose to check his loyalty points, he will be asked to enter his ID. Each time he buys a book the loyalty points will be increased by one point. (in case his loyalty points are higher than 5 he will get a 10% discount as in this case).

II. Code Explanation:

Classes:

```
#pragma once
#ifndef Book_header
#define Book_header
#include"Date.h"
#include<string>
#include"Client.h"
#include<iostream>
#include<fstream>
using namespace std;

class Book
{
    int BookID;
    string BookName;
    string AuthorName;
    Date Publishing;
    int CopiesNum;
    int Price;
public:
    Book(Date d,int id, string name, int num, string author,int price);
    Book(const Book& B);
    Book();
    Book* AddBook(Book* Books, int* n);
    bool BookSold(Book* Books, int BooksNumber);
    void ModifyPrice(Book* Books, int P, int Number);
    void ModifyCopiesNum(Book* Books,int Num,int Number);
    void SetBookID(int id);
    void SetBookName(string s);
    void SearchByID(Book* Books,int Number);
    void SearchByName(Book* Books, int Number);
    friend ostream& operator<<(ostream& os, const Book& B){ ... }
    friend istream& operator>>(istream& is, Book& B){ ... }
    friend Book* GetBooks(Book* Books);
    friend void WriteBooksInFile(Book* Books, int Number);
};

#endif
```

```
#include"Book.h"
#include"Date.h"
#include<fstream>
#include<iostream>
#include<string>
#include "Exceptions.h"
using namespace std;
Book::Book(Date d, int id=0, string name="", int num=0, string author="",int price=0)
    :BookID(id), BookName(name),CopiesNum(num),AuthorName(author),Publishing(d),Price(price) {}
Book::Book(const Book& B){ ... }
Book::Book(){ ... }
Book* Book::AddBook(Book* Books, int* n){ ... }
void Book::ModifyPrice(Book* Books,int P,int Number ){ ... }
void Book::ModifyCopiesNum(Book* Books,int Num, int Number){ ... }
void Book::SetBookID(int id){ ... }
void Book::SetBookName(string s){ ... }
void Book::SearchByID(Book* Books, int Number){ ... }
void Book::SearchByName(Book* Books,int Number){ ... }
bool Book::BookSold(Book* Books, int BooksNumber){ ... }
```

This class is Called Book: inside the class there is implementation of constructors, destructors, and copy-constructors. There is also operator overloading (“<<” and”>>”).

Each method gets the array of books and its size as parameters to apply the desired modification on the books.

Methods in Book Class:

-Book* AddBook(Book* Books, int* n): the functions gets the array that contains all the books in the shop and add a book to it. N is the size of the array and it is incremented by one if the addition is successful. If the entered book's ID already exists in the shop the function will check if there is a book with the same name or not(it is case sensitive).if there is a book with same name then the number of copies entered will be added to the already stored in the shop. If the name does not exist in the shop then the program will notify the user that the ID is already used.

- bool BookSold(Book* Books, int BooksNumber): this function allows to sell the book meaning it check if the number of copies of the desired book is higher than 0 or not and if so the number will be decremented by one.

-void ModifyPrice(Book* Books, int P, int Number), void ModifyCopiesNumber(Book* Books, int Num, int Number) : these two functions allow to modify the price or the number of copies of the book.

-void SetBookID(Book* Books, int Number), void SetBookName(Book* Books, int Number) : these two function allow to set the ID or the name of the book when there is no need to enter all the details of the book.

-void SearchByID(Book* Books, int Number), void SearchByName(Book* Books, int Number) : these two functions allow to search for the book in the array that contains all available books in the array whether by ID or by Name.

Both of the functions WriteBookInFile() and GetBooks() are friend because they are declared in the main and the access to the attributes is needed.

```h
#pragma once
#ifndef Human_header
#define Human_header
#include"Date.h"
#include<string>
#include<iostream>
#include"Date.h"
using namespace std;
class Human
{protected:
    string PersonName;
    int Age;
    Date BirthDate;
public:
    Human(Date D,string name, int age);
    Human();
    Human(const Human& H);
    ~Human(){}
    friend ostream& operator<<(ostream& os, Human H){ ... }
};
#endif
```

```cpp
#include "Human.h"
#include"Date.h"
Human::Human(Date D, string name="", int age=0): PersonName(name), Age(age),BirthDate(D) {}
Human::Human(){ ... }
Human::Human(const Human& H){ ... }
```

```h
#pragma once
#ifndef Employee_header
#define Employee_header
#include"Human.h"
#include"Date.h"
#include<iostream>
#include<string>
class Employee : public Human
{
    int EmployeeID;
    string Position;
    int Salary;
    int PhoneNum;
    string Address;
public:
    Employee(Date D,int Employeeid, string name, int age,string position
    Employee(const Employee& E);
    Employee();
    ~Employee(){}
    Employee* AddEmployee(Employee* Employees,int* Number);
    void ModifyAdd(Employee* Employees,string add, int Number);
    void Modifyphone(Employee* Employees, int phone, int Number);
    void ModifyPos(Employee* Employees, string pos, int Number);
    void ModifySalary(Employee* Employees, int s, int Number);
    void SetID(int id);
    friend ostream& operator<<(ostream& os, const Employee& E){ ... }
    friend istream& operator>>(istream& is, Employee& E){ ... }
    friend Employee* GetEmployees(Employee* Employee);
    friend void WriteEmployeeInFile(Employee* Employee, int Number);
};
#endif
```

```cpp
#include "Employee.h"
#include"Date.h"
#include"Human.h"
#include<string>
#include<iostream>
#include<fstream>
Employee::Employee(Date D,int Employeeid, string name="", int age=0, string positi
Employee::Employee(const Employee& E){ ... }
Employee::Employee(){ ... }
Employee* Employee::AddEmployee(Employee* Employees, int* Number){ ... }
void Employee::ModifyAdd(Employee* Employees,string add, int Number){ ... }
void Employee::Modifyphone(Employee* Employees,int phone,int Number){ ... }
void Employee::ModifyPos(Employee* Employees, string pos , int Number){ ... }
void Employee::ModifySalary(Employee* Employees, int s, int Number){ ... }
void Employee::SetID(int id){ ... }
```

```h
#pragma once
#ifndef Client_header
#define Client_header
#include"Date.h"
#include"Human.h"
#include"Book.h"
#include<iostream>
#include<fstream>
#include<string>
class Client: public Human
{
    int ID;
    int Loyalty;
public:
    Client(Date D, int id, string name, int age);
    Client(const Client& C);
    Client();
    ~Client(){}
    void UpdateLoyalty(Client* Clients, int Number);
    void SetClientID(int id);
    Client* AddClient(Client* Clients, int* Number);
    void Ckeckloyalty(Client* Clients, int Number);
    friend istream& operator>>(istream& is, Client& C){ ... }
    friend ostream& operator<<(ostream& os, const Client& C){ ... }
    friend Client* GetClients(Client* Clients);
    friend void WriteClientInFile(Client* Clients, int Number);
};
#endif
```

```cpp
#include "Client.h"
#include"Date.h"
#include"Human.h"
Client::Client(Date D, int id=0, string name="", int age=0):Human(D
Client::Client(const Client& C){ ... }
Client::Client(){ ... }
void Client::SetClientID(int id){ ... }
Client* Client::AddClient(Client* Clients, int *Number){ ... }
void Client::Ckeckloyalty(Client* Clients,int Number){ ... }
void Client::UpdateLoyalty(Client* Clients, int Number){ ... }
```

Inside of each class there is implementation of constructors, destructors, and copy-constructors. There is also operator overloading (“<<” and”>>”).

Each method gets the array of books and its size as parameters to apply the desired modification on the books. (Employee* in case of class Employee and Client* in case of class Client).

Methods in Client Class:

-Client* AddClient(Client* Clients, int* Number): this function let the user add a client to the array that contains all the clients in the shop. If the entered Id already exists in the system, the user will be notified. In case of a successful addition the number will be incremented by one.

-void CheckLoyalty(Client* Clients, int* Number): this function allows to display the loyalty points that the Client have.

-void UpdateLoyalty(Client* Clients, int Number): this function allow to increment the loyalty points of the client when a successful purchase has occurred.

Both of the functions WriteClientInFile() and GetClients() are friend because they are declared in the main and the access to the attributes is needed.

Methods in Client Class:

-Employee* AddEmployee(Employee* Employees, int* Number): this function let the user add an employee to the array that contains all the employees in the shop. If the entered Id already exists in the system, the user will be notified. In case of a successful addition the number will be incremented by one.

- void ModifyAdd(Employee* Employees, string add, int Number), void ModifyPos(Employee* Employees, string pos, int Number), void ModifyPhone(Employee* Employees, int phone, int Number), void ModifySalary(Employee* Employees, int s, int Number) : these four functions allow to modify some of the details of the employee.

Both of the functions WriteEmployeeInFile() and GetEmployees() are friend because they are declared in the main and the access to the attributes is needed.

```cpp
#include"Date.h"
#include<string>
Date::Date(int D = 0, int M = 0, int Y = 0) :Day(D), Month(M), Year(Y) {}
Date:: Date():Day(0), Month(0), Year(0) {}
Date::Date(const Date& D){ ... }
Date::Date(int d):Day(d),Month(d),Year(d){}
int Date::getDay(){ ... }
int Date::getMonth(){ ... }
int Date::getYear(){ ... }
string function(void){ ... }
```

```cpp
#pragma once
#ifndef Date_header
#define Date_header
#include<iostream>
using namespace std;
class Date
{
    int Day;
    int Month;
    int Year;
public:
    Date(int D, int M, int Y );
    Date(int d);
    Date();
    Date(const Date& D);
    ~Date(){}
    int getDay();
    int getMonth();
    int getYear();
    friend ostream& operator<<(ostream& os, const Date& D){ ... }
    friend istream& operator>>(istream& is, Date& D){ ... }

};
string function(void);
#endif
```

In Date Class, there is implementation of constructors, destructors, and copy-constructors. There is also operator overloading (“<<” and”>>”).

Exceptions:

```cpp
#pragma once
#include<exception>
using namespace std;
class EmployeeFileNotOpenException{ ... } EmployeeFileNotEx;
class BooksFileNotOpen{ ... } BookFileNotOpenEx;
class CllientsFileNotOpen{ ... } ClientsFileNotOpenEx;
class BookOutOfStockException{ ... } BookOutOfStockEx;
class BookNotFoundException{ ... } BookNotFoundEx;
```

*****Some Functions in the main source file:**

```cpp
int NumberOfBooks(void){ ... }
int NumberOfClients(void){ ... }
int NumberOfEmployees(void){ ... }
Book* GetBooks(Book* Books){ ... }
Client* GetClients(Client* Clients){ ... }
Employee* GetEmployees(Employee* Employees){ ... }
void WriteBooksInFile(Book* Books, int Number){ ... }
void WriteClientInFile(Client* Clients,int Number){ ... }
void WriteEmployeeInFile(Employee* Employees, int Number){ ... }
```

NumberOf…..() functions allow us to know the number of books, clients, and employees.

Get…() functions allow us to put books, employees and clients in a specified array.

WriteInFile() functions allow us to write this arrays in the files when the program terminates.