ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

ФАКУЛТЕТ КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ

Курсова работа

Дисциплина: „Програмни езици”

***Изготвил**: Самуил Антонов Миланов*

Фак. № 121222062

Група: 37б

III курс, КСИ

София, 2024

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



// Main class for the program - Management System. I have encapsulated the Textbook and Distributors classes inside,
// so that the access to the classes is limited only through the Menu of the ManagementSystem
class ManagementSystem {

    // The Textbook class has setters and getter as well as an overriden << operand. The setters validate the
    // parameters given to them and throw exceptions in case they are not valid. Also they are private so that
    // access even within the ManagementSystem class is limited.
    class Textbook {
    private:
        string title;
        string author;
        int edition;
        string isbn;
        string publishDate;
        int circulation;
        bool approved;
        string approvalDate;
        double price;

    public:
        Textbook(string t, string a, int e, string i, string pd, int c, bool ap, string ad, double p) {
            setTitle(t);
            setAuthor(a);
            setEdition(e);
            setISBN(i);
            setPublishDate(pd);
            setCirculation(c);
            setApproved(ap);
            setApprovalDate(ad);
            setPrice(p);
        }

        Textbook() {
            edition = 0;
            circulation = 0;
            approved = false;
            price = 0.0;
        }

        // Getters and setters
        string getTitle() const {
            return title;
        }
```

```
    string getAuthor() const {
        return author;
    }
    int getEdition() const {
        return edition;
    }
    string getISBN() const {
        return isbn;
    }
    string getPublishDate() const {
        return publishDate;
    }
    int getCirculation() const {
        return circulation;
    }
    bool isApproved() const {
        return approved;
    }
    string getApprovalDate() const {
        return approvalDate;
    }
    double getPrice() const {
        return price;
    }

private:
    // Helper function to validate date (format: DD-MM-YYYY)
    bool isValidDate(const string& date) {
        regex dateRegex(R"(^\d{2}.\d{2}.\d{4}$)");
        return regex_match(date, dateRegex);
    }

    // Updated setters
    void setTitle(string t) {
        if (t.empty()) {
            throw invalid_argument("Title cannot be empty.");
        }
        title = t;
    }

    void setAuthor(string a) {
        if (a.empty()) {
            throw invalid_argument("Author cannot be empty.");
        }
        author = a;
    }

    void setEdition(int e) {
        if (e <= 0) {
            throw invalid_argument("Edition must be greater than 0.");
        }
        edition = e;
    }

    void setISBN(string i) {
        if (i.empty()) {
            throw invalid_argument("ISBN cannot be empty.");
        }
        isbn = i;
    }
```

```
        void setPublishDate(string pd) {
            if (!isValidDate(pd)) {
                throw invalid_argument("Publish date must be in the format
DD.MM.YYYY.");
            }
            publishDate = pd;
        }

        void setCirculation(int c) {
            if (c <= 0) {
                throw invalid_argument("Circulation must be greater than
0.");
            }
            circulation = c;
        }

        void setApproved(bool ap) {
            approved = ap;
        }

        void setApprovalDate(string ad) {
            if (ad.empty() && approved) {
                throw invalid_argument("Approval date cannot be empty for
an approved textbook.");
            }
            if (!ad.empty() && !isValidDate(ad)) {
                throw invalid_argument("Approval date must be in the format
DD.MM.YYYY.");
            }
            approvalDate = ad;
        }

        void setPrice(double p) {
            if (p <= 0) {
                throw invalid_argument("Price must be greater than 0.");
            }
            price = p;
        }
    public:
        // Overload << operator
        friend ostream& operator<<(ostream& out, const Textbook& textbook)
{
            out << "Title: " << textbook.title << "\n"
                << "Author: " << textbook.author << "\n"
                << "Edition: " << textbook.edition << "\n"
                << "ISBN: " << textbook.isbn << "\n"
                << "Publish Date: " << textbook.publishDate << "\n"
                << "Circulation: " << textbook.circulation << "\n"
                << "Approved: " << (textbook.approved ? "Yes" : "No") <<
"\n"
                << "Approval Date: " << textbook.approvalDate << "\n"
                << "Price: " << textbook.price << "\n";
            return out;
        }
    };

    // The distributor class is pretty simular to the Textbook class -
private setters with data validation
    class Distributor {
    private:
        string name;
```

```
        string address;
        string phone;

    public:
        Distributor(string n, string a, string p) {
            setName(n);
            setAddress(a);
            setPhone(p);
        }

        Distributor() {}

        // Getters and setters
        string getName() const {
            return name;
        }
        string getAddress() const {
            return address;
        }
        string getPhone() const {
            return phone;
        }

    private:
        void setName(string n) {
            if (n.empty()) {
                throw invalid_argument("Name cannot be empty.");
            }
            name = n;
        }

        void setAddress(string a) {
            if (a.empty()) {
                throw invalid_argument("Address cannot be empty.");
            }
            address = a;
        }

        void setPhone(string p) {
            // Regex to match phone numbers in a standard format (e.g.,
+1234567890, 123-456-7890, etc.)
            regex phoneRegex(R"(^(\+359|0)\d{9}$)");
            if (!regex_match(p, phoneRegex)) {
                throw invalid_argument("Phone number is invalid. Please
provide a valid phone number.");
            }
            phone = p;
        }
    public:
        // Overload << operator
        friend ostream& operator<<(ostream& os, const Distributor& dist) {
            os << "Name: " << dist.name << "\n"
                << "Address: " << dist.address << "\n"
                << "Phone: " << dist.phone << "\n";
            return os;
        }
    };
    vector<Textbook> textbooks;
    vector<Distributor> distributors;
```

```
    // Here start the ManagementSystem functions. AddTextBook and
addDistributor both have a try catch segment, when trying to create an
instance,
    // so that when the input is invalid, the operation is cancelled,
therefore the user gets sent back to the menu.
    void addTextbook() {
        string title, author, isbn, publishDate, approvalDate;
        int edition, circulation;
        bool approved;
        double price;

        cout << "Enter textbook details:\n";
        cout << "Title: ";
        cin.ignore();
        getline(cin, title);
        cout << "Author: ";
        getline(cin, author);
        cout << "Edition: ";
        cin >> edition;
        cout << "ISBN: ";
        cin.ignore();
        getline(cin, isbn);
        cout << "Publish Date: ";
        getline(cin, publishDate);
        cout << "Circulation: ";
        cin >> circulation;
        cout << "Approved (1 for Yes, 0 for No): ";
        cin >> approved;
        cin.ignore();
        if (approved) {
            cout << "Approval Date: ";
            getline(cin, approvalDate);
        }
        cout << "Price: ";
        cin >> price;

        try {
            textbooks.emplace_back(title, author, edition, isbn,
publishDate, circulation, approved, approvalDate, price);
            cout << "Textbook added successfully!\n";
        }
        catch (const invalid_argument& e) {
            cerr << "Error: " << e.what() << endl;
            return;
        }
    }

    void addDistributor() {
        string name, address, phone;

        cout << "Enter distributor details:\n";
        cout << "Name: ";
        cin.ignore();
        getline(cin, name);
        cout << "Address: ";
        getline(cin, address);
        cout << "Phone: ";
        getline(cin, phone);
        try {
            distributors.emplace_back(name, address, phone);
            cout << "Distributor added successfully!\n";
```

```
        }
        catch (const invalid_argument& e) {
            cerr << "Error: " << e.what() << endl;
            return;
        }
    }

    // A simple function to place to orders
    void placeOrder() {

        if (distributors.empty() || textbooks.empty()) {
            cout << "Add distributors and textbooks before placing an
order.\n";
            return;
        }

        cout << "Select a distributor:\n";
        for (size_t i = 0; i < distributors.size(); i++) {
            cout << i + 1 << ". " << distributors[i].getName() << "\n";
        }
        int distributorIndex;
        cin >> distributorIndex;
        distributorIndex--;

        if (distributorIndex < 0 || distributorIndex >=
distributors.size()) {
            cout << "Invalid selection.\n";
            return;
        }

        double totalPrice = 0;
        while (true) {
            cout << "Select a textbook to order (0 to finish):\n";
            for (size_t i = 0; i < textbooks.size(); i++) {
                cout << i + 1 << ". " << textbooks[i].getTitle() << " - "
<< textbooks[i].getPrice() << "\n";
            }
            int textbookIndex;
            cin >> textbookIndex;
            if (textbookIndex == 0) break;

            textbookIndex--;
            if (textbookIndex < 0 || textbookIndex >= textbooks.size()) {
                cout << "Invalid selection.\n";
                continue;
            }

            totalPrice += textbooks[textbookIndex].getPrice();
        }

        cout << "Total order price: " << totalPrice << "\n";
    }

    void saveToFile() {
        // This function simply saves all the Distributors and Textbooks to
a file
        ofstream file("data.txt");
        if (!file) {
            cout << "Error opening file for writing.\n";
            return;
        }
```

```
        file << "Textbooks:\n";
        for (const auto& tb : textbooks) {
            file << tb << "\n";
        }

        file << "Distributors:\n";
        for (const auto& dist : distributors) {
            file << dist << "\n";
        }

        file.close();
        cout << "Data saved to file successfully!\n";
    }

    void readFromFile() {
        /* This Function uses two main loops to go through all the entries
of Textbooks and Distributors in the file and save them in the private
vectors of this class.*/
        ifstream file("data.txt");
        if (!file) {
            cout << "Error opening file for writing.\n";
            return;
        }

        string s;
        getline(file, s);
        while (s != "Distributors:\0"){ // This cycle finds all the
parameters for a Textbook, and then creates it

            getline(file, s);
            if (strcmp(s.c_str(), "Distributors:\0") == 0) {
                break;
            }

            for (int i = 0; i < s.length(); i++) { // Theese for loop
simply extract the parameter from the line, for ex. Tittle: Abstract
Algebra -> Abstract Algebra
                if (s.c_str()[i] == ' ') {
                    s = s.substr(i + 1);
                    break;
                }
            }
            string Title = s;

            getline(file, s);
            for (int i = 0; i < s.length(); i++) {
                if (s.c_str()[i] == ' ') {
                    s = s.substr(i + 1);
                    break;
                }
            }
            string Author = s;

            getline(file, s);
            int length = s.length();
            for (int i = 0; i < s.length(); i++) {
                if (s.c_str()[i] == ' ') {
                    s = s.substr(i + 1);
                    break;
                }
```

```
}
int edition = stoi(s);

getline(file, s);
for (int i = 0; i < s.length(); i++) {
    if (s.c_str()[i] == ' ') {
        s = s.substr(i + 1);
        break;
    }
}
string ISBN = s;

getline(file, s);
int counter = 0;
for (int i = 0; i < s.length(); i++) {
    if (s.c_str()[i] == ' ') {
        counter++;
    }
    if (s.c_str()[i] == ' ' && counter > 1) {
        s = s.substr(i + 1);
        break;
    }
}
counter = 0;
string PublishDate = s;

getline(file, s);
for (int i = 0; i < s.length(); i++) {
    if (s.c_str()[i] == ' ') {
        s = s.substr(i + 1);
        break;
    }
}
int Circulation = stoi(s);

getline(file, s);
for (int i = 0; i < s.length(); i++) {
    if (s.c_str()[i] == ' ') {
        s = s.substr(i + 1);
        break;
    }
}
bool Approved;
if (s == "Yes") {
    Approved = true;
}
else {
    Approved = false;
}

getline(file, s);
for (int i = 0; i < s.length(); i++) {
    if (s.c_str()[i] == ' ') {
        counter++;
    }
    if (s.c_str()[i] == ' ' && counter > 1) {
        s = s.substr(i + 1);
        break;
    }
}
string ApprovalDate = s;
```

```
            getline(file, s);
            for (int i = 0; i < s.length(); i++) {
                if (s.c_str()[i] == ' ') {
                    s = s.substr(i + 1);
                    break;
                }
            }
            float price = stof(s);
            textbooks.emplace_back(Title, Author, edition, ISBN,
PublishDate, Circulation, Approved, ApprovalDate, price);
            getline(file, s);
        }

        while (getline(file, s)) { // And this cycle gets all the
parameters for a Distributor object and creates it
            for (int i = 0; i < s.length(); i++) {
                if (s.c_str()[i] == ' ') {
                    s = s.substr(i + 1);
                    break;
                }
            }
            string name = s;
            getline(file, s);
            for (int i = 0; i < s.length(); i++) {
                if (s.c_str()[i] == ' ') {
                    s = s.substr(i + 1);
                    break;
                }
            }
            string Address = s;
            getline(file, s);
            for (int i = 0; i < s.length(); i++) {
                if (s.c_str()[i] == ' ') {
                    s = s.substr(i + 1);
                    break;
                }
            }
            string phone = s;
            distributors.emplace_back(name, Address, phone);
            getline(file, s);
        }

        file.close();
        cout << "Data read from file successfully!\n";
    }

public:
    ManagementSystem() {};
    // This function is the main loop of the program. It is also the only
public method of this class, again to limit unintended use.
    void displayMenu() {
        int choice;
        do {
            cout << "\nMenu:\n";
            cout << "1. Add Textbook\n";
            cout << "2. Add Distributor\n";
            cout << "3. Place Order\n";
            cout << "4. Save Data to File\n";
            cout << "5. Read Data from File\n";
            cout << "0. Exit\n";
```

```
            cout << "Enter your choice: ";
            cin >> choice;

            switch (choice) {
            case 1:
                addTextbook();
                break;
            case 2:
                addDistributor();
                break;
            case 3:
                placeOrder();
                break;
            case 4:
                saveToFile();
                break;
            case 5:
                readFromFile();
                break;
            case 0:
                cout << "Exiting program. Goodbye!\n";
            }
        } while (choice != 0);
    }
};

int main(int argc, char* argv[]) {
    ManagementSystem* manager = new ManagementSystem();
    manager->displayMenu();
    delete manager;
    return 0;
}
```