ПРОЕКТ БИБЛИОТЕКА

Глава 1. - Увод

1.1 Описание и идея на проекта

Проектът представлява една информационна система за библиотека, която да може по-лесно да се структурира, контролира и обработва. Програмата е написана на езика **C++,** на **Visual Studio 2019**. Работи се с конзолата, където ще се въвеждат командите. Книгите трябва лесно да се добавят и да се премахват и да могат да се представят във вид, полезен за потребителя, като показване на наличните книги, търсене на книга, сортиране на книгите и тн. Отделно трябва да има администратор, който чрез паролата си за достъп управлява и добавянето на книгите и потребителите, които могат да използват програмата.

1.2 Цел и задачи на разработката

За цел ще си поставим написването на програма, която да е лесно разбираема от потребителя, с лесни команди .

Важна част е нашата библиотека да записва всяка една промяна, дори и при изключване, защото не искаме всеки път да добавяме всички книги наново. За целта ще записваме информацията в края на програмата и ще четем информацията в началото, посредством текстови файлове (____.txt), намиращи се в папката на проекта. Ще се работи активно с два файла, като единият няма да може да се подменя, защото ще е файл, на който се помещават потребителските имена и паролите на всеки един потребител, а те ще бъдат контролирани само от потребител с администраторски права.

В началото, когато програмата се стартира потребителят ще види изречение, което ще го помоли той да напише команда “**open**”, с която да бъде зареден файл, ако ли не в противен случай се създава автоматично празен такъв. Ако потребителят откаже да въведе име на файла, то той няма да бъде допуснат да продължи работата си с прогрмата! Ако е нов трябва да напише командата “**help**” и тя ще изведе на екрана поредица от информация, която е полезена за нови потребители. Програмата ще извежда подходящи съобщения в случай, че потребителят бъде объркан. Също така ще очаква от потребителят да направи това, което е изписано като условие за командата считано до главни и малки букви и тип на въвеждането(очаква се на година да въведе число :D). Всяка направена промяна се съхранява в програмата, но се запазва в текущият или друг посочен файл, само когато потребителят изрично подаде командата “**save**” или съответно “**save as**”. След като бъде въведена командата “**exit**” програмата освобождава заетата памет и завършва изпълнението си.

1.3 Структура на документацията

Структурата на документацията представлява обширно описание на работата и възможностите на конкретния проект. Дадена е информация за средствата, използвани за разработката на приложението, архитектурата и реализирането на основните операции.

Глава 2. – Преглед на предметната област

2.1 Основни дефиниции, концепции и алгоритми, които ще бъдат ползвани

В писането на програмата ще се използват основни ООП принципи като Класове, Обекти, Енкапсулация и Абстакция. Друга важна концепция, която ползваме са потоците *ifstream* и *ofstream*, чрез които записваме гореспоменатите файлове за съхранение на информацията.

2.2 Дефиниране на проблеми и сложност на поставената задача

Проблемите са няколко, като един от тях е удобството на крайния потребител. Изграждането на достатъчно лесно разбираемо меню е много голямо предизвикателство. Другият проблем идва в трудното преобразуване на идеите от главата на код, като се налага честото сменяне на идеята по няколко пъти.

2.3. Подходи, методи (евентуално модели и стандарти) за решаване на поставените проблемите

Решаването на един голям проблем често се основава на разбиването му на няколко малки такива. В програмата вместо да ползваме стандартния *std::vector*, ще реализираме допълнителни класове, които да преобразуваме по начин, които е пригоден за нашата задача(използване допълнителни функии).

2.4. Потребителски (функционални) изисквания (права, роли, статуси, диаграми) и качествени (нефункционални) изисквания (скалируемост, поддръжка)

Програмата започва с това, че трябва правилно да зареди файл, с който да оперира. Да може да се отварят файловете, да се променят и запазват. Програмата завършва с изход и затваряне на съответните файлове, с които е работила.

Програмата има два типа роли – на обикновен потребител, и администратор(вграден в самата програма и се сменява само с разрешението на програмист, тъй като слагането му във файл с другите потребители може да компрометира програмата). Когато

програмата се стартира няма потребител, тогава са достъпни само определени функции, които са по подразбиране. При успешно влизане на потребител, той получава достъп до самото приложение и може да борави с него. Ако на входа се влезе с правилни администраторски парола и име, на даденият потребител се отключват всякакви възможности, включително добавяне/премахване на книги и добавяне/премахване на потребители.

Глава 3. – Проектиране

3.1 Обща архитектура – ООП дизайн

Архитектурата на проекта спазва добрите практики на ООП като всички класове са разделени на два отделни файла: Заглавна логика(.h) и Изходна логика(.cpp)

Започваме с класовете **User/Usermass**. Както личи по имената двата класа са взаимносвързани.

В **User** имаме:

```
private:
    char username[256];
    char password[256];
    bool isAdmin;
public:
    User();
    User(const char* username, const char* password);
    User(const char* username, const  char* password, bool isAdmin);
    ofstream& saveUser(ofstream& os);
    ifstream& readUser(ifstream& is);

public:
    //Setters
    void setUsername(const char* userame);
    void setPassword(const char* password);
    void setisAdmin(bool isAdmin);
    //Getters
    const char* getUsername()const;
    const char* getPassword()const;
    bool getisAdmin()const;
};
```

char username[256] - променлива, която указва потребителското име, тя е с дължина максимално 256 символа(стандарт за нашата програма)

char password[256] - променлива, която указва паролата на потребителя, тя е с дължина максимално 256 символа(стандарт за нашата програма).

bool isAdmin – булева стойност, която указва правата на дадения потребител

User() – “Default constructor”(начален конструктор**)** задава началните стойности при създаването на класа. Останалите конструктури са пригодени според желани от нас нужди.

ofstream& saveUser(ofstream& os) – записва данните за потребител на файл

ifstream& readUser(ifstream& is) – чете данни на потребител от файл

“setters” и “getters”- Създадени са съответни за всяка променлива, които са част от добрите ООП практики и ни помагат да по – чист и четлив код и лесен достъп извън класа без да се нарушава капсулацията.

Usermass е ръчно направен вектор от **User**, в него имаме:

```
private:
    User* users;
    int size;
    int capacity;
private:
    void copy(const Usermass& other);
    void resize();
    void erase();
public:
    Usermass();
    Usermass(const Usermass& other);
    Usermass& operator=(const Usermass& other);
    User& operator[](int i)const;
    ~Usermass();
    Usermass& AddUser(const User& newUser);
    Usermass& RemoveUser(const char* username);
    ofstream& saveUsers(ofstream& out);
    ifstream& readUsers(ifstream& in);
    bool checkforUser(const char* username, const char* password);
    int getSize()const;
};
```

User* users- масив от потребители

Int size- текущият размер на вектора

Int capacity- капацитетът на вектора(най-добрата практика е да се зададе степен на цифрата 2)

В класа е реализирана т. Наречената „Голяма четворка“ , а за да избегнем повторение на код реализираме функциите **copy(), resize(), erase()**

Предефинирани са и оператори така че да удовлетворяват нашето условие.

checkforUser() – функция, която връща булева стойност по някаква критерия

getSize() – функция, която връща големината на вектора

AddUser(), RemoveUser() са функции, които добавят или премахват потребител от вектора.

saveUsers(), readUsers() – функции, които записват/четат даден вектор от файл.

Operator[] – предефинираме оператор скоби, за да можем да го ползваме, когато ползваме цикли, в които участва нашият вектор.

Другите два реализирани класа са **Book/Bookmass**

В **Book** имаме:

```
private:
    char author[256];
    char header[256];
    char ganre[64];
    char resume[1024];
    size_t year;
    char keywords[128];
    double rating;
    size_t ID;

public:
    //Canonic
    Book();
    Book(const char* author, const char* header, const char* ganre, const char* resume, size_t year, const char* keywords, double rating, size_t ID);
    ofstream& saveBook(ofstream& os);
    ifstream& readBook(ifstream& is);
public:
    //Setters
    void setAuthor(const char* author);
    void setHeader(const char* header);
    void setGanre(const char* ganre);
```

Char author[256] – Променлива за автора на книгата

Char header[256] – Променлива за заглавието на книгата

Char ganre[64] – Променлива за жанра на книгата

Char resume[1024] – Кратко резюме на книгата

Size_t year – Година на издаване

Size_t ID - Уникален идентификационен номер на книгата в нашата библиотека

Double rating – Рейтинг, дробно число

Реализирани са **конструктури** и функции за **запис** и **четене** на и от **файл**. Присъстват и “**setters**” и ”**getters**”.

Bookmass е вектор на класа **Book**

```
private:
    Book* books;
    int size;
    int capacity;
private:
    void copy(const Bookmass& other);
    void resize();
    void erase();
public:
    //Canonic
    Bookmass();
    Bookmass(const Bookmass& other);
    Bookmass& operator=(const Bookmass& other);
    Book& operator[](int i)const;
    ~Bookmass();
public:
    //Additional
    Bookmass& AddBook(const Book& newBook);
    Bookmass& RemoveBook(const char* header);
    void SortBooksbyAuthor();
    void SortBooksbyHeader();
    void SortBooksbyYear();
    void SortBooksbyRating();
    void BookPrintByID(size_t ID)const;
    ofstream& saveBooks(ofstream& out);
    ifstream& readBooks(ifstream& in);
    int getSize()const;
    void printByOrder(const char* order);
```

В него отново имаме **„Голяма четворка“,** която е задължителна, както и функциите **copy(), resize(), erase().**

Допълнително са направени функциите **AddBook(), RemoveBook(),** които добавят и трият книги от даден вектор.

Имаме и функции за сортиране на вектор по зададен критерии.

Налични са и функциите за четене/записване на файл.

Глава 4. – Реализация, тестване

4.1. Реализация на класове (включва важни моменти от реализацията на класовете и малки фрагменти от кода)

Както беше споменато вече класовете са разделени на (.h) и (.cpp). Сега ще разгледаме ключови реализации на функции на самите класове.

Ключови моменти са записванети и четенето на файл. За целта имплементирам функции, направени специално за нашите класове. Включена е и проверка за изправност при отваряне на файл:

```cpp
ofstream& User::saveUser(ofstream& os)
{
        if (os.is_open())
        {
            os << this->username << endl;
            os << this->password << endl;
        }
        else
        {
            cout << "Cannot open stream for saving user!" << endl;
        }
        return os;
}

ifstream& User::readUser(ifstream& is)
{
    if (is.is_open())
    {
        is >> this->username;
        is >> this->password;
    }
    return is;
```

```cpp
ofstream& Book::saveBook(ofstream& os)
{
    if (os.is_open())
    {
        os << strlen(this->author) << " ";
        os << this->author << endl;
        os << strlen(this->header) << " ";
        os << this->header << endl;
        os << strlen(this->ganre) << " ";
        os << this->ganre << endl;
        os << strlen(this->resume) << " ";
        os << this->resume << endl;
        os << strlen(this->keywords) << " ";
        os << this->keywords << endl;
        os << this->year << endl;
        os << this->rating << endl;
        os << this->ID << endl;
    }
    else
    {
        cout << "Cannot open stream for saving book!" << endl;
    }
    return os;
```

```cpp
ifstream& Book::readBook(ifstream& is)
{
    if (is.is_open())
    {
        int AuthorLength = 0;
        int HeaderLength = 0;
        int GanreLength = 0;
        int ResumeLength = 0;
        int KeywordsLength = 0;

        is >> AuthorLength;
        is.seekg(1, ios::cur);
        is.getline(this->author, AuthorLength + 1);
        is >> HeaderLength;
        is.seekg(1, ios::cur);
        is.getline(this->header, HeaderLength + 1);
        is >> GanreLength;
        is.seekg(1, ios::cur);
        is.getline(this->ganre, GanreLength + 1);
        is >> ResumeLength;
        is.seekg(1, ios::cur);
        is.getline(this->resume, ResumeLength + 1);
        is >> KeywordsLength;
        is.seekg(1, ios::cur);
```

No issues found

Записването става като всяка една променлива се записва на нов ред, заедно с нейната дължина, която е изключително важна за четенето след това. Четенето става посредством прочитане на символи до дължина, равна на дължината на символите на даден ред, а след извършване на действието се преминава на следващ ред.

4.2. Управление на паметта и алгоритми. Оптимизации.

Друг ключов момент от реализирането на класовете Usermass и Bookmass е имплементирането на „Голяма четворка“(Default constructor, Copy constructor, operator=, Destructor).

```
Usermass::Usermass()
{
    this->size = 0;
    this->capacity = 16;
    this->users = new User[this->capacity];
}

Usermass::Usermass(const Usermass& other)
{
    this->copy(other);
}

Usermass& Usermass::operator=(const Usermass& other)
{
    if (this != &other)
    {
        this->erase();
        this->copy(other);
    }
    return *this;
}
```

```
void Usermass::erase()
{
    delete[] this->users;
}

void Usermass::copy(const Usermass& other)
{
    this->size = other.size;
    this->capacity = other.capacity;
    this->users = new User[this->capacity];

    for (int i = 0; i < this->size; i++)
    {
        this->users[i] = other.users[i];
    }
}

void Usermass::resize()
{
    this->capacity *= 2;
    User* temp = new User[this->capacity];
    for (int i = 0; i < this->size; i++)
    {
        temp[i] = this->users[i];
    }
    this->erase();
    this->users = temp;
}
```

No issues found

Изключително важна реализация е заделянето на памет както и нейното изтриване в края на програмата, за да се избегне „memory leak”.

```
Usermass::~Usermass()
{
    this->erase();
}
```

Добавянето и премахването да обекти от двата вектора е също важна част от реализацият им. Също така се прави проверка в самият вектор за налично пространство и след това се добавя. Премахването също прави съответна проверка.

```
Bookmass& Bookmass::AddBook(const Book& newBook)
{
    if (this->size >= this->capacity)
    {
        this->resize();
    }
    this->books[size++] = newBook;
    return *this;
}
Bookmass& Bookmass::RemoveBook(const char* header)
{
    if (size > 0)
    {
        for (int i = 0; i < this->size; i++)
        {
            if (strcmp(books[i].getHeader(), header) == 0)
            {
                for (int j = i; j < this->size - 1; j++)
                {
                    books[j] = books[j + 1];
                }
            }
        }
        size--;
    }
        return *this;
```

Сортирането използва алгоритъм подобен да “**Bubble sort**”, като има някои разлики в оптимизирането.

```
void Bookmass::SortBooksbyAuthor()
{
    Bookmass sortedbooks;
    for (int j = 0; j <= this->size; j++) {
        for (int i = 0; i < this->size - 1; i++)
        {
            if (strcmp(books[i].getAuthor(), books[i + 1].getAuthor()) > 0)
            {
                sortedbooks[0] = books[i];
                books[i] = books[i + 1];
                books[i + 1] = sortedbooks[0];
            }
        }
    }
}
```

4.3. Планиране, описание и създаване на тестови сценарии (създаване на примери)

Преди имплементирането на методите на класовете са направени тестове за всеки един дали работи както трябва**.** Последно решеният проблем беше реализирането на така наречената “*password effect*”, която маскира паролата със символа ‘*‘(добавени са и някои гранични случаи(например “*backspace*” бутонът се взимаше като символ, а сега върши изначалната си работа – да трие символите), игнорирани са “*Space*” и “*Escape*”, а при натискане на “*Enter*” прекратява въвеждането)

```
while (g <= 255)
{
    password[g] = _getch();
    c = password[g];
    if (c == 13)
    {
        break;
    }
    if (c == 32 || c == 27)
    {
        continue;
    }
    if (c != 8)
    {
        password[g] = c;
        printf("*");
        g++;
    }
    else
    {
        g--;
        if (g < 0)
            g++;
        else
            printf("\b \b");
    }

}
printf("\b \b");
password[g] = '\0';
```

Глава 5. – Заключение

5.1. Обобщение на изпълнението на началните цели

Смея да кажа, че огромна част от първоначалните идеи са изпълнени. Винаги има възможност една програма да бъде счупена, а и смятам, че за някои неща може би има по – ефикасни методи, така че съм отворен за градивна критика.

5.2. Насоки за бъдещо развитие и усъвършенстване

В бъдещето може да се имплементират някой порпавки като: По-доброто валидиране на входните данни от потребителя, да се добяват нови функционалности към така направената задача, и да се усъвършенства готовият продукт, така че да става все по – добър за крайния потребител.