

**МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»**

**(РУТ (МИИТ))**

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

отчёт по курсовой работе

По дисциплине «Объективно-ориентированное программирование»

**Выполнил:** студент группы ТКИ-342

Фещенко В. А.

**Проверил:** доцент кафедры УиЗИ, к.т.н. Балакина Е. П.

**Москва 2022 г.**

**Постановка задачи**

1. Создать класс элементов заданной структуры.

2. Работа программы осуществляется с помощью главного меню.

3. Предусмотреть ввод элементов класса, производить проверку корректности ввода

4. Предусмотреть сохранение введенных данных заданной структуры в файл, а также считывание данных из файла.

5. Предусмотреть сортировку элементов класса по выбранному параметру

6. Предусмотреть поиск определенного элемента класса по заданному параметру и вывод всех параметров данного элемента на экран.

7. Предусмотреть редактирование выбранного элемента класса.

Вариант 14

Название книги, автор, издательство, год выпуска, номер полки, в наличии в библиотеке/на руках. Выдавать список книг определенного автора, имеющихся в библиотеке.

**Описание переменных и констант**

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Наименование переменной или константы | Описание | Тип переменной или константы |
| 1 | lib | Вектор, содержащий объекты класса Book(основной) | Library |
| 2 | author | Автор | string |
| 3 | tmp | Вектор, содержащий объекты класса Book(буфер) | Library |
| 4 | menu | Управление меню | int |
| 5 | book | Буфер для добавления в библиотеку | Book |
| 6 | av | Аренда/возврат книги | bool |
| 7 | name | Имя файла для вывода | string |
| 8 | in/out | используется для ввода/вывода из файла/в файл вектора. | ifstream / ofstream |
| 9 | fileName | Имя файла для ввода | string |
| 10 | ret | Вектор, содержащий объекты класса Book(буфер для ввода из файл) | Library |

**Описание сущности класса**

|  |  |  |
| --- | --- | --- |
| № п/п | Наименование параметра или функции | Описание |
| Book | | |
| 1 | title | поле класса, название |
| 2 | author | поле класса, автор |
| 3 | publisher | поле класса, издатель |
| 4 | year | поле класса, год |
| 5 | shelfNumber | поле класса, номер полки |
| 6 | available | поле класса, доступность в библиотеке |
| 7 | Book() | конструкторы класса |
| 9 | Book(std::string title, std::string author, std::string publisher, int year, int shelfNumber, bool available) |
| 8 | Book(const Book &book); |
| 9 | std::string getTitle(); | Методы ввода/вывода приватных полей класса |
| 10 | std::string getAuthor(); |
| 11 | std::string getPublisher(); |
| 12 | int getYear(); |
| 13 | int getShelfNumber(); |
| 14 | bool getAvailable(); |
| 15 | void setTitle(std::string title); |
| 16 | void setAuthor(std::string author); |
| 17 | void setPublisher(std::string publisher); |
| 18 | void setYear(int year); |
| 19 | void setShelfNumber(int shelfNumber); |
| 20 | void setAvailable(bool available); |
| 21 | std::string presentAvailable(); |
| 22 | friend std::istream &operator>>(std::istream &out, Book &book) | Перегрузка оператора ввода |
| 23 | Friend std::ostream &operator<<(std::ostream &out, Book &book) | Перегрузка оператора вывода |
| 24 | ~Book() | Деструктор класса |
| Library | | |
| 1 | std::vector<Book> libShelf | Вектор книг в библиотеке |
| 2 | Library() | Конструктор класса |
| 3 | ~Library() | Деструктор класса |
| 4 | void sortByTitle(); | Сортировка по названию |
| 5 | std::vector<Book> findByAuthor(std::string author); | Поиск по автору |
| 6 | void addBook(const Book &book); | Добавление книги в вектор |
| 7 | std::vector<Book>::iterator getBegin(); | Обращение к началу/концу вектора книг |
| 8 | std::vector<Book>::iterator getEnd(); |
| 9 | void showAuthor(std::string author); | Вывод вектора книг определенного автора |
| 10 | void changeAvailable(bool av); | Изменение доступности книги в библиотеке |
| 11 | std::vector<Book> getVector(); | Обращение к вектору |
| 12 | friend std::ostream &operator<<(std::ostream &out, Library &lib); | Перегрузка оператора вывода |

**Блок схема алгоритма**



Рисунок - Блок-схема алгоритма



Рисунок - Блок-схема алгоритма добавления новой книги



Рисунок - Блок-схема сортировки по названию



Рисунок - Блок-схема алгоритма поиска по автору



Рисунок - Блок-схема алгоритма изменения доступности книги в библиотеке

**Код программы**

Main.cpp

#include <iostream>

#include <fstream>

#include <string>

#include "Library.hpp"

int input();

void fileOutput(std::vector<Book> lib);

void fileInput(Library &lib);

int main()

{

    int menu = 0;

    Library lib;

    Library tmp;

    std::string author;

    while(true)

    {

        std::cout << "Menu: \n";

        std::cout << "1)Add book in library\n2)Sort by title\n3)Find by author\n4)Library output\n5)Change book available status in lib\n6)File output\n7)File input\n0)Exit\n";

        menu = input();

        switch (menu)

        {

            case 1:

            {

                std::cout << "input book:\n";

                Book book;

                try

                {

                    std::cin >> book;

                }

                catch(const std::exception& e)

                {

                    std::cerr << e.what() << "\n";

                }

                lib.addBook(book);

                break;

            }

            case 2:

            {

                lib.sortByTitle();

                std::cout << lib;

                break;

            }

            case 3:

            {

                std::cout << "input author to find:\n";

                std::cin >> author;

                lib.showAuthor(author);

                break;

            }

            case 4:

            {

                try

                {

                    std::cout << lib;

                }

                catch(const std::exception& e)

                {

                    std::cerr << e.what() << "\n";

                }

                break;

            }

            case 5:

            {

                bool av;

                std::cout << "1)Return book to library\n0)Take book from library\n";

                std::cin >> av;

                lib.changeAvailable(av);

                break;

            }

            case 6:

            {

                fileOutput(lib.getVector());

                break;

            }

            case 7:

            {

                fileInput(lib);

                break;

            }

            case 0:

            {

                exit(0);

            }

            default:

            {

                std::cout << "default!\n";

                break;

            }

        }

    }

    return 0;

}

void fileOutput(std::vector<Book> lib)

{

    std::ofstream out;

    std::string name;

    std::cout << "Enter name of file:\n";

    std::cin >> name;

    out.open(name.c\_str());

    if (out.is\_open())

    {

        for (std::vector<Book>::iterator i = lib.begin(); i != lib.end(); i++)

        {

            out << (\*i).getTitle() << " " << (\*i).getAuthor() << " " << (\*i).getPublisher() << " " << (\*i).getYear() << " " << (\*i).getShelfNumber() << " " << (\*i).presentAvailable() << "\n";

        }

    }

    else

    {

        std::cout << "Error! File can't open.\n";

    }

    out.close();

    name.append(".txt");

    out.open(name.c\_str());

    if (out.is\_open())

    {

        for (std::vector<Book>::iterator i = lib.begin(); i != lib.end(); i++)

        {

            out << (\*i) << "\n";

        }

    }

    else

    {

        std::cout << "Error! File can't open.\n";

    }

    out.close();

}

void fileInput(Library& lib)

{

    std::string fileName;

    std::cout << "Enter file name for file input.\n";

    std::cin >> fileName;

    std::vector<Book> ret;

    std::ifstream in;

    in.open(fileName.c\_str());

    if (in.is\_open() == 1)

    {

        std::string title, author, publisher, year, shelfNumber, available;

        while (in >> title >> author >> publisher >> year >> shelfNumber >> available)

        {

            Book tmp;

            tmp.setTitle(title);

            tmp.setAuthor(author);

            tmp.setPublisher(publisher);

            tmp.setYear(stoi(year));

            tmp.setShelfNumber(stoi(shelfNumber));

            if(available.compare("on hands"))

            {

                tmp.setAvailable(false);

            }

            else

            {

                tmp.setAvailable(true);

            }

            ret.push\_back(tmp);

        }

    }

    else

    {

        std::cout << "invalid filename\n";

    }

    in.close();

    for (std::vector<Book>::iterator i = ret.begin(); i != ret.end(); i++)

    {

        lib.addBook((\*i));

    }

}

}

int input()

{

    int value;

    while (!(std::cin >> value))

    {

        std::cin.clear();

        std::cin.ignore(1000, '\n');

        std::cout << "Only num\n";

    }

    return value;

}

Library.hpp

#pragma once

#include "Book.hpp"

#include <vector>

#include <iostream>

class Library

{

    private:

    std::vector<Book> libShelf;

    public:

    Library();

    ~Library();

    class VectorIsEmpty : public std::exception {

        const char \*what() const throw();

    };

    void sortByTitle();

    std::vector<Book> findByAuthor(std::string author);

    void addBook(const Book &book);

    std::vector<Book>::iterator getBegin();

    std::vector<Book>::iterator getEnd();

    void showAuthor(std::string author);

    void changeAvailable(bool av);

    std::vector<Book> getVector();

    friend std::ostream &operator<<(std::ostream &out, Library &lib);

};

Library.cpp

#include "Library.hpp"

Library::Library()

{

}

Library::~Library()

{

}

const char \*Library::VectorIsEmpty::what() const throw()

{

    return "Library is empty\n";

}

void Library::sortByTitle()

{

    for (std::vector<Book>::iterator i = libShelf.begin(); i != libShelf.end(); i++)

    {

        for (std::vector<Book>::iterator j = i + 1; j != libShelf.end(); j++)

        {

            if (((\*i).getTitle()).compare(((\*j).getTitle())) > 0)

            {

                std::swap((\*j), (\*i));

            }

        }

    }

}

std::vector<Book> Library::findByAuthor(std::string author)

{

    std::vector<Book> authorLib;

    for (std::vector<Book>::iterator i = libShelf.begin(); i != libShelf.end(); i++)

    {

        if (((\*i).getAuthor()).compare(author) == 0)

        {

            authorLib.push\_back((\*i));

        }

    }

    return authorLib;

}

void Library::addBook(const Book &book)

{

    libShelf.push\_back(book);

}

std::vector<Book>::iterator Library::getBegin()

{

   return libShelf.begin();

}

std::vector<Book>::iterator Library::getEnd()

{

    return libShelf.end();

}

void Library::showAuthor(std::string author)

{

    std::vector<Book> tmp;

    tmp = this->findByAuthor(author);

    for (std::vector<Book>::iterator i = tmp.begin(); i != tmp.end(); i++)

    {

        std::cout << "" << (\*i) << "\n";

    }

}

void Library::changeAvailable(bool av)

{

    int temp = 0;

    std::vector<Book> tmp;

    for (std::vector<Book>::iterator i = this->getBegin(); i != this->getEnd(); i++)

    {

        if ((\*i).getAvailable() != av)

        {

            std::cout << (\*i);

            tmp.push\_back(\*i);

        }

    }

    if(av == 1)

    {

        std::cout << "\nWhat book you need to return?\n";

    }

    else

    {

        std::cout << "\nWhat book you want to take?\n";

    }

    std::cin >> temp;

    std::vector<Book>::iterator j = this->getBegin() + temp;

    for(std::vector<Book>::iterator i = this->getBegin(); i != this->getEnd(); i++)

    {

        if (i == j)

        {

            (\*i).setAvailable(av);

        }

    }

}

std::vector<Book> Library::getVector()

{

    return libShelf;

}

std::ostream &operator<<(std::ostream &out, Library &lib)

{

    if (lib.libShelf.size() == 0)

    {

        throw Library::VectorIsEmpty();

    }

    for (std::vector<Book>::iterator i = lib.getBegin(); i != lib.getEnd(); i++)

    {

        std::cout << " " << (\*i) << "\n";

    }

    return out;

}

Book.hpp

#pragma once

#include <iostream>

class Book

{

    private:

    std::string title, author, publisher;

    int year, shelfNumber;

    bool available;

    public:

    class AvailableNotAccept : public std::exception {

        const char \*what() const throw();

    };

    class YearNotAccept : public std::exception {

        const char \*what() const throw();

    };

    class ShelfNumberNotAccept : public std::exception {

        const char \*what() const throw();

    };

    Book();

    Book(std::string title, std::string author, std::string publisher, int year, int shelfNumber, bool available);

    Book(const Book &book);

    ~Book();

    std::string getTitle();

    std::string getAuthor();

    std::string getPublisher();

    int getYear();

    int getShelfNumber();

    bool getAvailable();

    std::string presentAvailable();

    void setTitle(std::string title);

    void setAuthor(std::string author);

    void setPublisher(std::string publisher);

    void setYear(int year);

    void setShelfNumber(int shelfNumber);

    void setAvailable(bool available);

    friend std::istream &operator>>(std::istream &out, Book &book);

    friend std::ostream &operator<<(std::ostream &out, Book &book);

};

Book.cpp

#include "Book.hpp"

Book::Book()

{

    title = "none";

    author = "none";

    publisher = "none";

    year = 0;

    shelfNumber = 0;

    available = false;

}

Book::Book(std::string title, std::string author, std::string publisher, int year, int shelfNumber, bool available)

{

    this->title = title;

    this->author = author;

    this->publisher = publisher;

    this->year = year;

    this->shelfNumber = shelfNumber;

    this->available = available;

}

Book::Book(const Book &book)

{

    this->title = book.title;

    this->author = book.author;

    this->publisher = book.publisher;

    this->year = book.year;

    this->shelfNumber = book.shelfNumber;

    this->available = book.available;

}

Book::~Book()

{

}

std::string Book::getTitle()

{

    return title;

}

std::string Book::getAuthor()

{

    return author;

}

std::string Book::getPublisher()

{

    return publisher;

}

int Book::getYear()

{

    return year;

}

int Book::getShelfNumber()

{

    return shelfNumber;

}

bool Book::getAvailable()

{

    return available;

}

std::string Book::presentAvailable()

{

    if(available)

    {

        return "available in library";

    }

    else

    {

        return "on hands";

    }

}

void Book::setTitle(std::string title)

{

    this->title = title;

}

void Book::setAuthor(std::string author)

{

    this->author = author;

}

void Book::setPublisher(std::string publisher)

{

    this->publisher = publisher;

}

void Book::setYear(int year)

{

    this->year = year;

}

void Book::setShelfNumber(int shelfNumber)

{

    this->shelfNumber = shelfNumber;

}

void Book::setAvailable(bool available)

{

    this->available = available;

}

std::istream &operator>>(std::istream &in, Book &book)

{

    std::string tmpTitle, tmpAuthor, tmpPublisher;

    int tmpYear, tmpShelfNumber;

    int tmpAvailable;

    std::cout << "Title: ";

    in >> tmpTitle;

    std::cout << "Author: ";

    in >> tmpAuthor;

    std::cout << "Publisher: ";

    in >> tmpPublisher;

    std::cout << "Year: ";

    in >> tmpYear;

    std::cout << "Shelf number: ";

    in >> tmpShelfNumber;

    std::cout << "Available: ";

    in >> tmpAvailable;

    if (tmpAvailable != 0 && tmpAvailable != 1)

    {

        throw Book::AvailableNotAccept();

    }

    book.setTitle(tmpTitle);

    book.setAuthor(tmpAuthor);

    book.setPublisher(tmpPublisher);

    book.setYear(tmpYear);

    book.setShelfNumber(tmpShelfNumber);

    book.setAvailable(tmpAvailable);

    return in;

}

std::ostream &operator<<(std::ostream &out, Book &book)

{

    out << "Title: " << book.getTitle() << " Author: " << book.getAuthor() << " Publisher: " << book.getPublisher() << " Year: " << book.getYear() << " Shelf number: " << book.getShelfNumber() << " Available in lib: " << book.presentAvailable();

    return out;

}

const char \*Book::AvailableNotAccept::what() const throw()

{

    return "Available:\n'0' - on hands\n'1' - available in library\n";

}

const char \*Book::YearNotAccept::what() const throw()

{

    int value;

    while (!(std::cin >> value))

    {

        std::cin.clear();

        std::cin.ignore(1000, '\n');

    }

    return "Year definition is number\n";

}

const char \*Book::ShelfNumberNotAccept::what() const throw()

{

    int value;

    while (!(std::cin >> value))

    {

        std::cin.clear();

        std::cin.ignore(1000, '\n');

    }

    return "Shelf number definition is number\n";

}

**Результат выполнения программы**

Изображение выглядит как текст

Автоматически созданное описание  
Изображение выглядит как текст

Автоматически созданное описание

Рисунок - Добавление книги в библиотеку

Рисунок - Вывод списка книг в библиотеке

Изображение выглядит как текст

Автоматически созданное описание

Рисунок - Сортировка по названию

Изображение выглядит как текст

Автоматически созданное описание

Рисунок - Поиск по автору

Изображение выглядит как текст

Автоматически созданное описание

Рисунок - Изменение доступности книги в библиотеке

Изображение выглядит как текст

Автоматически созданное описание

Рисунок - Вывод в файл

Изображение выглядит как текст

Автоматически созданное описание

Рисунок - Ввод с файла

**Вывод**

В ходе выполнения курсовой работы были освоены навыки объектно-ориентированного программирования, изучены контейнеры из стандартной библиотеки шаблонов языка C++, а также работа с файлами.