Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет инженерно-экономический

Кафедра экономической информатики

**Отчет к лабораторной работе №5**

**по теме**

**«ПОТОКОВЫЙ ФАЙЛОВЫЙ ВВОД/ВЫВОД»»**

|  |  |
| --- | --- |
| Выполнил: | Вдовенко Н. Д.  студент группы 272303 |
| Проверила: | Салапура М. Н.  ст. преподаватель |

Минск 2023

Общая постановка задачи

**Цель работы:** изучить особенности файловых потоков ввода/вывода на языке С++.

**Индивидуальное задание:**

2.На основе разработанной иерархии классов, реализованной в лабораторной работе «ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ В ЯЗЫКЕ С++», реализовать программу для работы с данными, используя потоки файлового ввода-вывода. Реализовать функций 1. добавления данных в файл, 2. удаления данных из файла, 3. редактирования данных в файле, 4. просмотра данных из файла, 5. поиска данных по необходимым параметрам в файле. Использовать функции open(), is\_open(), bad(), good(), close(), eof(), fail() для анализа состояния файла, перегрузить операторы вывода в поток << и считывания из потока >>, использовать функции seekg(), seekp(), tellg().

**Краткие теоретические сведения:**

Операции над файлами выполняются с использованием файлового ввода-вывода.

Прочитать файл, созданный предыдущей программой, можно с использованием объекта типа ifstream. Файл автоматически открывается при создании объекта. Затем данные из него можно считать с помощью оператора извлечения из потока (>>).

Для работы со строками, содержащими пробелы после каждой строки необходимо писать специальный символ-ограничитель и использовать метод getline() для считывания строки.

Объекты классов заголовка ios содержат флаги статуса ошибок, с помощью которых можно проверить результат выполнения операций. При чтении файла порциями будет достигнуто условие окончания файла. Сигнал EOF посылается в программу операционной системой, когда больше нет данных для чтения.

Методы put() и get() классов basic\_ostream и basic\_istream соответственно, могут быть использованы для ввода и вывода единичных символов.

Оператор reinterpret\_cast изменяет тип данных в определенной области памяти, не анализируя смысл проводимых преобразований. Вопрос целесообразности использования этого оператора находится в рамках ответственности программиста.

Не имеет значения, какие используются методы в классах т.к. они не записываются в файл вместе с данными. Для данных важен единый формат записи, а разногласие между методами не имеет значения. Это утверждение верно только для классов, в которых не используются виртуальные функции.

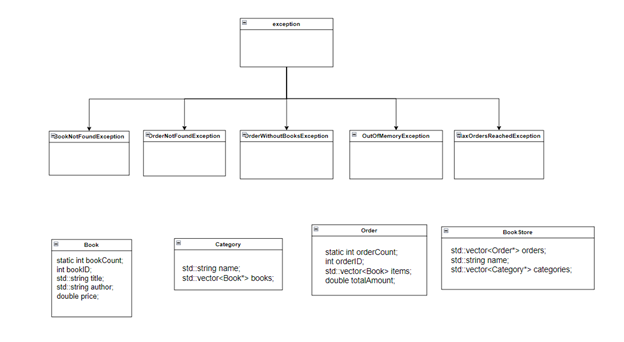
В этом случае, класс, использующийся для чтения объекта, должен быть идентичен классу, использовавшемуся при его записи.

Вертикальные слеши между флагами 16 нужны для того, чтобы из битов сформировалось единое целое число. При этом несколько флагов будут применяться одновременно.

У каждого файлового объекта есть два ассоциированных с ним значения, называемые указатель чтения и указатель записи (текущая позиция чтения и текущая позиция записи).

Оптимально в полноценных проектах использовать все эти функции после каждой операции ввода/вывода.

**Диаграмма классов:**



**Контрольные вопросы:**

1. **ifstream** и **ofstream** - это классы потоков для ввода и вывода данных из/в файлы в C++.
2. **fopen()** - это функция в стандартной библиотеке языка C, используемая для открытия файла. Прототип: **FILE\* fopen(const char\* filename, const char\* mode)**, где **filename** - имя файла, **mode** - режим открытия файла (например, "r" для чтения, "w" для записи).
3. **fseek()** - функция для установки позиции в файле. Прототип: **int fseek(FILE\* stream, long int offset, int whence)**, где **stream** - указатель на файл, **offset** - смещение относительно позиции, определенной параметром **whence** (например, SEEK\_SET - начало файла, SEEK\_CUR - текущая позиция, SEEK\_END - конец файла).
4. **ofstream saleFile("Sales.txt")** - это выражение создает объект **saleFile** класса **ofstream** и связывает его с файлом "Sales.txt" для записи данных.
5. **if (!saleFile.eof()) { /\* код \*/ }** - этот оператор проверяет, достигнут ли конец файла (EOF) или возникла ошибка при чтении данных из объекта **ifstream** с именем **saleFile**.
6. **fileOut.put('A')** - этот оператор записывает символ 'A' в объект **fileOut** класса **ofstream**.
7. **ifstream iFile; char buff[100]; iFile.read(buff, sizeof(buff))** - эти операторы читают все содержимое объекта **iFile** класса **ifstream** и сохраняют его в массив **buff**.
8. a) Верно, биты режимов **app** и **ate** определяются в классе **ios**. б) Верно, они могут устанавливаться как для чтения, так и для записи файлов. г) Неверно, они не устанавливают режимы открытия файлов.
9. "Текущая позиция" в файле - это место, куда указатель позиции в потоке указывает в данный момент времени, обозначает место, где будет происходить следующая операция ввода/вывода.
10. **f1.seekg(-13, ios::cur)** - это выражение сдвигает текущую позицию в файле на 13 байтов назад в потоковом объекте **f1**.

**Листинг кода:**

#pragma once

#include "libs.h"

class BookNotFoundException : public std::exception {

public:

BookNotFoundException();

const char\* what() const noexcept override;

private:

std::string message;

};

class OrderNotFoundException : public std::exception {

public:

OrderNotFoundException();

const char\* what() const noexcept override;

private:

std::string message;

};

class OrderWithoutBooksException : public std::exception {

public:

OrderWithoutBooksException();

const char\* what() const noexcept override;

private:

std::string message;

};

class OutOfMemoryException : public std::exception {

public:

OutOfMemoryException();

const char\* what() const noexcept override;

private:

std::string message;

};

class MaxOrdersReachedException : public std::exception {

public:

MaxOrdersReachedException();

const char\* what() const noexcept override;

private:

std::string message;

};

class Book {

public:

Book();

Book(const std::string& title, const std::string& author, double price);

~Book();

void DisplayInfo() const;

void WriteBookToFile(std::ofstream& outfile);

std::string GetTitle();

std::string GetAuthor();

double GetPrice();

int GetBookCount();

int GetBookID();

void SetTitle(std::string title);

void SetAuthor(std::string author);

void SetPrice(double price);

private:

static int bookCount;

int bookID;

std::string title;

std::string author;

double price;

};

class Category {

public:

Category(const std::string& name);

void AddBook(Book\* book);

void DisplayBooks() const;

std::vector<Book\*>& GetBooks();

private:

std::string name;

std::vector<Book\*> books;

};

class Order {

public:

Order();

void AddBook(Book& book);

void DisplayOrder() const;

void WriteOrderToFile(std::ofstream& outfile);

int GetOrderCount();

int GetOrderID();

float GetTotalAmount();

int GetCountOfBooks();

vector<Book> GetOrderBooks();

private:

static int orderCount;

int orderID;

std::vector<Book> items;

float totalAmount;

};

class BookStore {

public:

BookStore(const std::string& name);

void AddCategory(Category\* category);

Book GetBookByID(int id);

Order& GetOrderByID(int id);

int GetOrderPosition(Order& order);

void DisplayCategories();

void PlaceOrder();

void ShowOrders();

void FindOrder();

void DeleteOrder();

void ChangeOrder();

//void SortOrdersByPrice();

//void SortBooksByPrice();

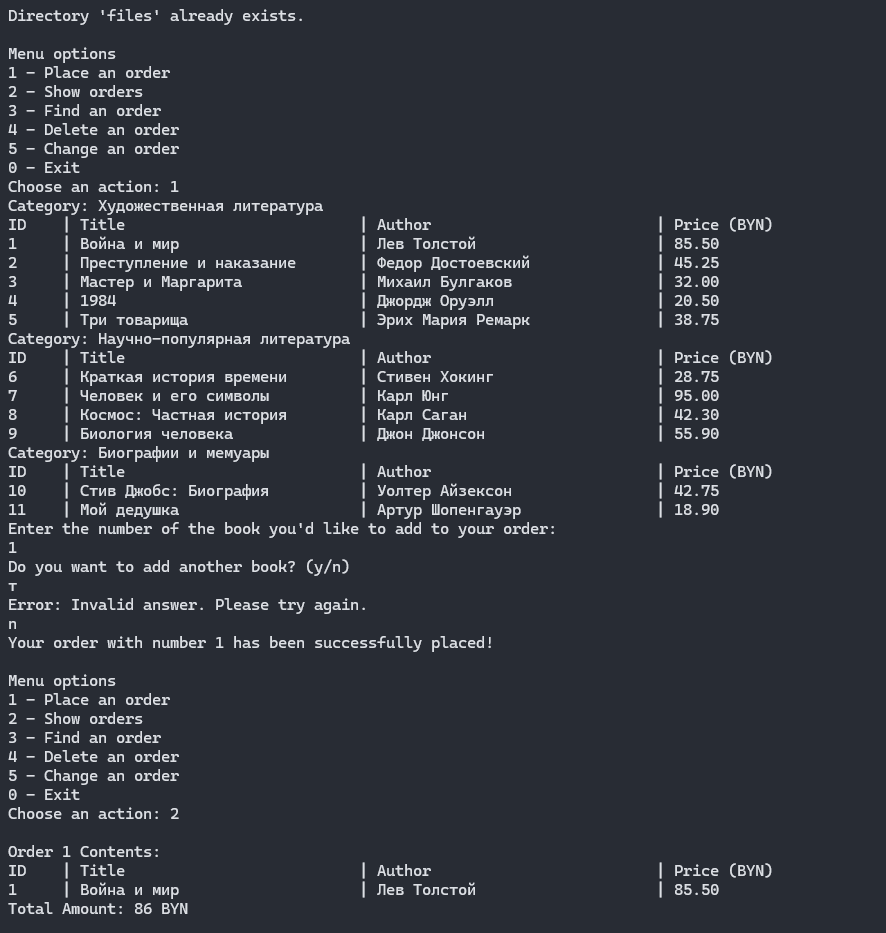
private:

std::vector<Order\*> orders;

std::string name;

std::vector<Category\*> categories;

};





**Вывод**: в ходе выполнения данной лабораторной работы по теме "Файловые потоки ввода/вывода" были изучены и применены основные принципы работы с файловыми потоками в программировании.