Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

ОТЧЕТ

к лабораторной работе №2

**«Использование потоков файлового ввода/вывода данных»**

|  |  |
| --- | --- |
|  | |
|  |
|  |
|  |
|  |  | | |
|  | | |  | |  |
|  | | |  | |  |
| Студент гр. 324402 | | |  | | Цевелюк А.И. |
| Проверила | | |  | | Купрейчик А.С. |

Минск 2024

**1 Цель работы**

Цель работы:изучение и практическое освоение механизмов файлового ввода/вывода данных в языке программирования C++. Лабораторная работа направлена на понимание работы с потоками данных, использование классов для обработки файлов (таких как ifstream, ofstream и fstream), реализацию основных операций записи и чтения данных в файлы и из файлов, а также обработку ошибок при работе с файлами.

**2 Теоретические сведения по лабораторной работе**

Работа с файлами в языке C++ осуществляется через файловые потоки, которые представляют собой каналы для ввода и вывода данных в файлы. Основными классами, используемыми для работы с файлами, являются ifstream (input file stream) для чтения данных из файлов, ofstream (output file stream) для записи данных в файлы и fstream (file stream) для выполнения как чтения, так и записи. Эти классы являются наследниками соответствующих потоковых классов: basic\_ifstream — наследник basic\_istream, basic\_ofstream — наследник basic\_ostream, а basic\_fstream — наследник basic\_iostream. Такие иерархии обеспечивают использование методов, общих для ввода/вывода, при работе с файлами.

Запись и чтение данных в файлы:

Для записи данных в файл используется объект класса ofstream. Пример кода:

ofstream ofs("example.txt");

if (ofs.is\_open()) {

ofs << "Hello, world!" << endl;

ofs.close();

}

Этот код создает файл example.txt и записывает в него строку "Hello, world!". Если файл уже существует, он будет перезаписан.

Для чтения данных из файла используется объект класса ifstream. Пример кода:

ifstream ifs("example.txt");

string line;

if (ifs.is\_open()) {

while (getline(ifs, line)) {

cout << line << endl;

}

ifs.close();

}

Этот код читает строки из файла example.txt и выводит их на экран.

Бинарный ввод/вывод данных:

Помимо текстового ввода/вывода, C++ поддерживает бинарный режим, который используется для более эффективного хранения данных, таких как числа и структуры. Запись и чтение в бинарном режиме осуществляется с использованием методов write() и read() классов basic\_ostream и basic\_istream.

Обработка ошибок:

Для обработки ошибок при работе с файлами в C++ используются методы и флаги потоковых классов, такие как is\_open(), eof(), fail(), bad(), и good(). Эти методы позволяют проверять успешность выполнения операций ввода/вывода и корректно реагировать на ошибки, такие как невозможность открытия файла или его повреждение.

**3 Формулировка индивидуального задания**

На основе разработанной иерархии классов, реализованной влабораторной работе «Использование потоков ввода/вывода д анных», реализовать программу работы с данными, используя потоки файлового ввода-вывода. Реализовать функций добавления данных в файл, удаления данных из файла, редактирования данных в файле, просмотра данных из файла, поиска данных по необходимым параметрам в файле. Использовать функции open(), is\_open(), bad(), good(), close(), eof(), fail(), перегрузить операторы вывода в поток << и считывания из потока >>, seekg(), seekp(), tellg().

**4 Код решения индивидуального задания**

// ReSharper disable CppClangTidyCertErr33C

// ReSharper disable CppClangTidyConcurrencyMtUnsafe

#include <iostream>

#include <fstream>

#include <iomanip>

#include <vector>

#include <string>

#include <algorithm>

#include <limits>

class item

{

std::string name\_;

double price\_;

public:

explicit item(std::string n = "", const double p = 0.0) : name\_(std::move(n)), price\_(p) {}

void set\_name(const std::string& n) { name\_ = n; }

void set\_price(const double p) { price\_ = p; }

std::string get\_name() const { return name\_; }

double get\_price() const { return price\_; }

friend std::ostream& operator<<(std::ostream& os, const item& item);

friend std::istream& operator>>(std::istream& is, item& item);

};

std::ostream& operator<<(std::ostream& os, const item& item)

{

os << std::setw(20) << std::left << item.name\_ << std::setw(10) << std::fixed << std::setprecision(2) << std::right << item.price\_;

return os;

}

std::istream& operator>>(std::istream& is, item& item)

{

std::getline(is, item.name\_, ',');

is >> item.price\_;

is.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

return is;

}

class order

{

std::vector<item> items\_;

public:

void add\_item(const item& item)

{

items\_.push\_back(item);

}

void remove\_item(const std::string& name)

{

const auto it = remove\_if(items\_.begin(), items\_.end(), [&name](const item& item) { return item.get\_name() == name; });

if (it != items\_.end())

{

items\_.erase(it, items\_.end());

std::cout << "Item removed successfully!\n";

}

else

std::cout << "Item not found.\n";

}

void edit\_item(const std::string& name, const item& new\_item)

{

for (auto& item : items\_)

{

if (item.get\_name() == name)

{

item = new\_item;

std::cout << "Item edited successfully!\n";

return;

}

}

std::cout << "Item not found.\n";

}

void sort\_items()

{

std::sort(items\_.begin(), items\_.end(), [](const item& a, const item& b) { return a.get\_name() < b.get\_name(); });

}

void display\_items(std::ostream& os) const

{

os << std::setw(20) << std::left << "Name" << std::setw(10) << std::right << "Price\n";

os << std::setfill('-') << std::setw(30) << "" << std::setfill(' ') << '\n';

for (const auto& item : items\_)

os << item << '\n';

}

void find\_item(const std::string& name) const

{

const auto it = std::find\_if(items\_.begin(), items\_.end(), [&name](const item& item) { return item.get\_name() == name; });

if (it != items\_.end())

std::cout << "Item found: " << \*it << '\n';

else

std::cout << "Item not found.\n";

}

void load\_from\_file(const std::string& filename)

{

std::ifstream file(filename);

if (!file.is\_open())

{

std::cerr << "Error opening file for reading!\n";

return;

}

item new\_item;

file.seekg(0);

while (file >> new\_item)

{

if (file.bad())

{

std::cerr << "Critical error while reading the file.\n";

file.clear();

break;

}

add\_item(new\_item);

}

file.close();

}

void save\_to\_file(const std::string& filename) const

{

std::ofstream file(filename);

if (!file.is\_open())

{

std::cerr << "Error opening file for writing!\n";

return;

}

file.seekp(0);

for (const auto& item : items\_)

{

file << item.get\_name() << ',' << item.get\_price() << '\n';

if (file.fail())

{

std::cerr << "Error writing to file.\n";

file.clear();

}

}

file.close();

}

static void remove\_item\_from\_file(const std::string& filename, const std::string& item\_name)

{

std::ifstream file(filename);

if (!file.is\_open())

{

std::cerr << "Error opening file for reading!\n";

return;

}

std::ofstream temp\_file("temp.txt");

if (!temp\_file.is\_open())

{

std::cerr << "Error opening temporary file for writing!\n";

return;

}

item current\_item;

std::streampos pos;

while (file >> current\_item)

{

pos = file.tellg();

if (current\_item.get\_name() != item\_name)

temp\_file << current\_item.get\_name() << ',' << current\_item.get\_price() << '\n';

else

std::cout << "Removing item at position: " << pos << "\n";

}

file.close();

temp\_file.close();

std::remove(filename.c\_str());

std::rename("temp.txt", filename.c\_str());

}

};

class shop // NOLINT(cppcoreguidelines-special-member-functions)

{

order\* order\_;

public:

shop() : order\_(new order()) {}

~shop() { delete order\_; }

void add\_order\_item(const item& item) const

{

order\_->add\_item(item);

}

void remove\_order\_item(const std::string& name) const

{

order\_->remove\_item(name);

}

void edit\_order\_item(const std::string& name, const item& new\_item) const

{

order\_->edit\_item(name, new\_item);

}

void sort\_order\_items() const

{

order\_->sort\_items();

}

void display\_order() const

{

order\_->display\_items(std::cout);

}

void find\_order\_item(const std::string& name) const

{

order\_->find\_item(name);

}

void load\_order(const std::string& filename) const

{

order\_->load\_from\_file(filename);

}

void save\_order(const std::string& filename) const

{

order\_->save\_to\_file(filename);

}

static void remove\_item\_from\_file(const std::string& filename, const std::string& item\_name)

{

order::remove\_item\_from\_file(filename, item\_name);

}

};

void end()

{

std::cout << "\n";

system("pause");

system("cls");

}

int main()

{

// ReSharper disable once CppTooWideScope

const shop shop;

int choice = 0;

std::string name;

double price;

do

{

if (choice) end();

std::cout << "\n===== Dish Shop =====\n";

std::cout << "1. Add Item\n";

std::cout << "2. Remove Item\n";

std::cout << "3. Edit Item\n";

std::cout << "4. Sort Items\n";

std::cout << "5. Display Order\n";

std::cout << "6. Load Order from File\n";

std::cout << "7. Save Order to File\n";

std::cout << "8. Find Item by Name\n";

std::cout << "9. Remove Item from File\n";

std::cout << "10. Exit\n";

std::cout << "Enter your choice: ";

std::cin >> choice;

switch (choice)

{

case 1:

{

std::cout << "Enter item name: ";

std::cin >> name;

std::cout << "Enter item price: ";

std::cin >> price;

shop.add\_order\_item(item(name, price));

break;

}

case 2:

{

std::cout << "Enter item name to remove: ";

std::cin >> name;

shop.remove\_order\_item(name);

break;

}

case 3:

{

std::cout << "Enter item name to edit: ";

std::cin >> name;

std::cout << "Enter new item name: ";

std::cin >> name;

std::cout << "Enter new item price: ";

std::cin >> price;

shop.edit\_order\_item(name, item(name, price));

break;

}

case 4:

shop.sort\_order\_items();

break;

case 5:

shop.display\_order();

break;

case 6:

{

std::cout << "Enter file name to load: ";

std::cin >> name;

shop.load\_order(name);

break;

}

case 7:

{

std::cout << "Enter file name to save: ";

std::cin >> name;

shop.save\_order(name);

break;

}

case 8:

{

std::cout << "Enter item name to find: ";

std::cin >> name;

shop.find\_order\_item(name);

break;

}

case 9:

{

std::string item\_name;

std::cout << "Enter file name: ";

std::cin >> name;

std::cout << "Enter item name to remove: ";

std::cin >> item\_name;

shop::remove\_item\_from\_file(name, item\_name);

break;

}

case 10:

std::cout << "Have a nice day!\n";

break;

default:

std::cout << "Invalid choice. Please try again.\n";

break;

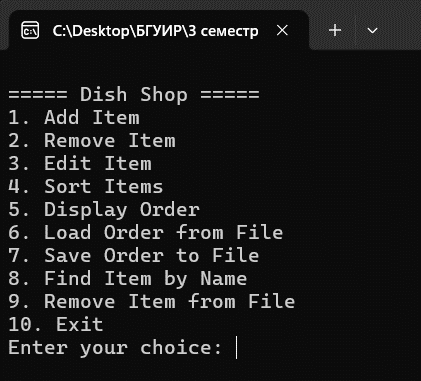
}

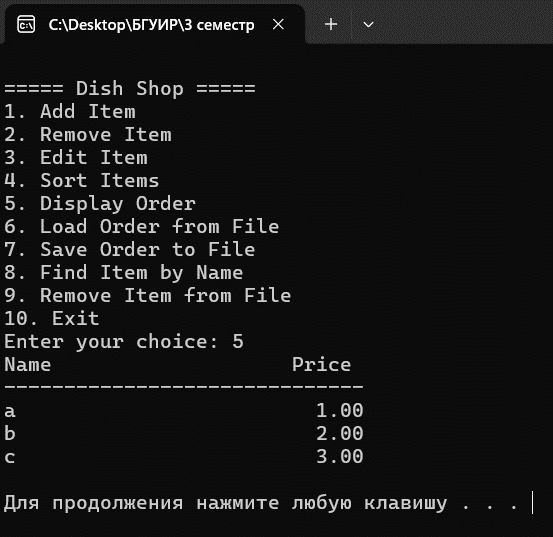
} while (choice != 10);

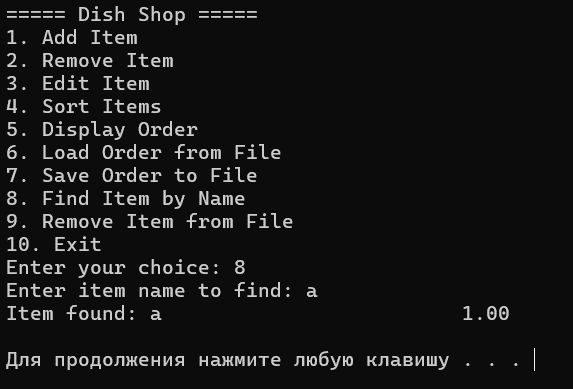
return 0;

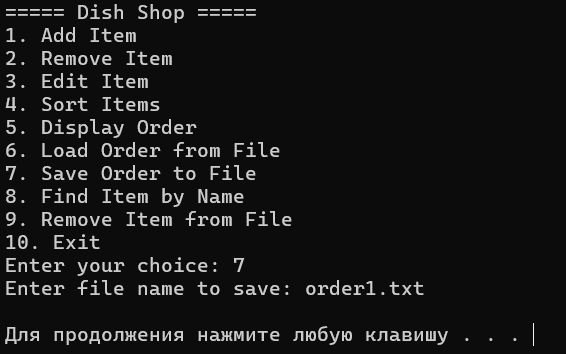
}

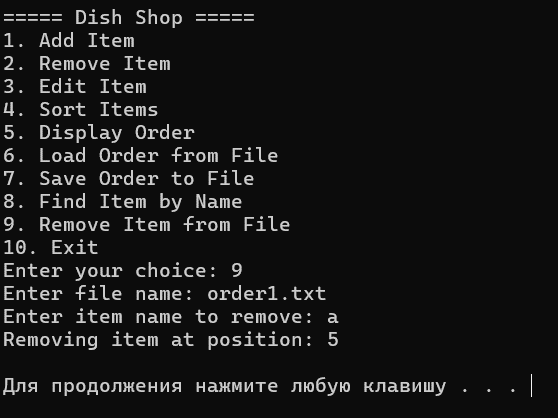
**5 Скриншоты выполнения индивидуального задания**

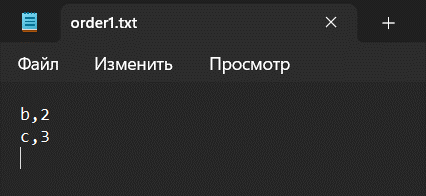


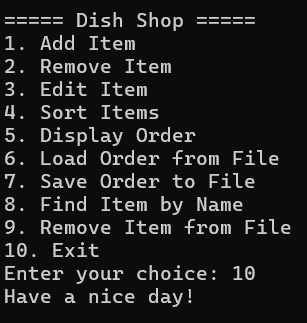












**6 Ответы на контрольные вопросы**

1. Назовите потоковые классы, предназначенные для файлового ввода/вывода.

Потоковые классы, предназначенные для файлового ввода/вывода в C++, это:

ifstream (input file stream) — используется для чтения данных из файла.

ofstream (output file stream) — используется для записи данных в файл.

fstream (file stream) — используется как для чтения, так и для записи данных в файл.

2. Для чего предназначена функция fopen()? Напишите её прототип.

Функция fopen() используется в языке C для открытия файла и связывания его с файловым указателем. Эта функция открывает файл с указанным именем в указанном режиме (чтение, запись, добавление и т. д.).

Прототип функции:

FILE \*fopen(const char \*filename, const char \*mode);

3. Для чего предназначена функция fseek()? Напишите её прототип.

Функция fseek() используется для установки позиции в потоке файла на заданное смещение. Это позволяет перемещать указатель чтения или записи на нужное место в файле.

Прототип функции:

int fseek(FILE \*stream, long int offset, int whence);

Параметры:

stream — указатель на файл.

offset — смещение в байтах.

whence — позиция, от которой осуществляется смещение (начало файла, текущая позиция или конец файла).

4. Напишите выражение, создающее объект saleFile класса ofstream, и ассоциируйте его с файлом Sales.txt.

ofstream saleFile("Sales.txt");

5. Напишите оператор if, определяющий, достиг ли объект типа ifstream под названием saleFile конца файла или же возникла ошибка.

if (saleFile.eof() || saleFile.fail()) {

// Обработка ошибки или конца файла

}

6. Напишите оператор, записывающий единичный символ в объект fileOut класса ofstream.

fileOut.put('A'); // Запись символа 'A' в файл

7. Напишите операторы, позволяющие считать все содержимое объекта iFile класса ifstream в массив buff.

iFile.read(buff, sizeof(buff));

8. Какие утверждения про биты режимов app и ate являются верными:

а) Определяются в классе ios — верно

б) Могут устанавливаться для чтения или для записи — неверно (только для записи)

в) Работают с функциями put() и get() — неверно

г) Устанавливают режимы открытия файлов — верно

9. Что такое «текущая позиция» в файле?

Текущая позиция в файле — это место в потоке файла, откуда будет выполняться следующая операция чтения или записи. Она определяется количеством байтов от начала файла до текущего указателя чтения или записи. Для управления текущей позицией используются функции seekg() и seekp() (для чтения и записи соответственно).

**7 Выводы по лабораторной работе**

Вывод: я изучил и освоил механизмы файлового ввода/вывода данных в языке программирования C++. Я разобрался в том, как работают с потоками данных, научился использовать классы для обработки файлов (таких как ifstream, ofstream и fstream), реализовывать основные операции записи и чтения данных в файлы и из файлов, а также обрабатывать ошибки при работе с файлами.