Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы» направление подготовки: 09.03.04 – «Программная инженерия»

# Лабораторная работа " Сохранение данных в файле с использованием потоков."

Выполнил студент гр. РИС-24-3б

Жиряков Леонид Антонович

Проверил:

Доцент кафедры ИТАС   
Ольга Андреевна Полякова

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (оценка) (подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

г. Пермь, 2024

Вариант 8.

Постановка задачи:

Общая:

1. Создать пользовательский класс с минимальной функциональностью.

2. Написать функцию для создания объектов пользовательского класса (ввод исходной информации с клавиатуры) и сохранения их в потоке (файле).

3. Написать функцию для чтения и просмотра объектов из потока.

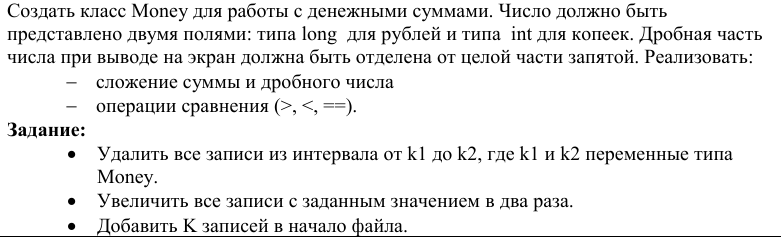
4. Написать функцию для удаления объектов из потока в соответствии с заданием варианта. Для выполнения задания выполнить перегрузку необходимых операций.

5. Написать функцию для добавления объектов в поток в соответствии с заданием варианта. Для выполнения задания выполнить перегрузку необходимых операций.

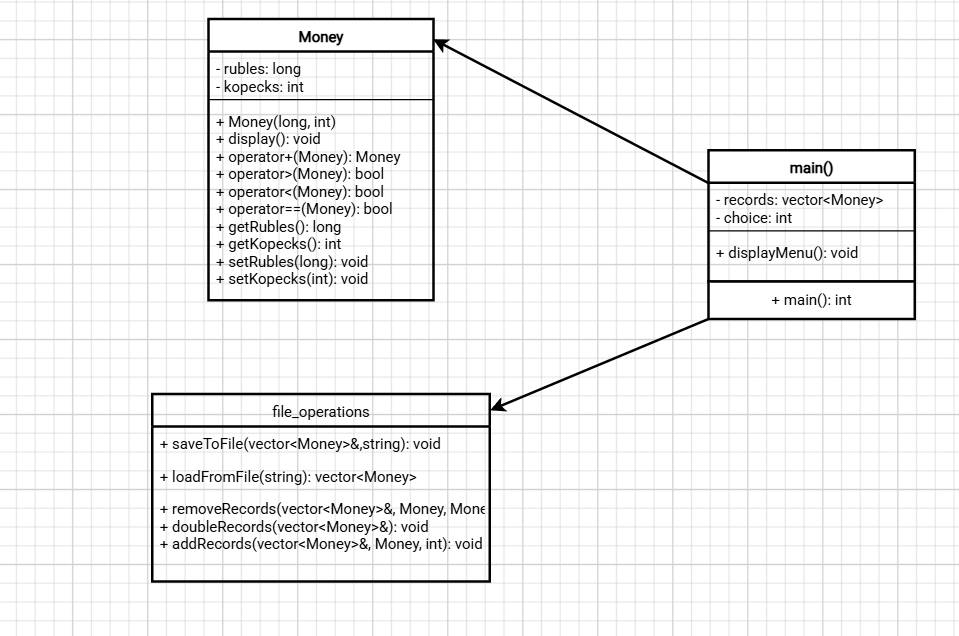
6. Написать функцию для изменения объектов в потоке в соответствии с заданием варианта. Для выполнения задания выполнить перегрузку необходимых операций.

7. Для вызова функций в основной программе предусмотреть меню.

Персональная:



UML – диаграмма:



Программная реализация

Заголовочный файл

**file\_operations.h:**#pragma once

#include "Money.h"

#include <fstream>

#include <vector>

#include <string>

void saveToFile(const std::vector<Money>& records, const std::string& filename);

std::vector<Money> loadFromFile(const std::string& filename);

void removeRecords(std::vector<Money>& records, const Money& k1, const Money& k2);

void doubleRecords(std::vector<Money>& records);

void addRecords(std::vector<Money>& records, const Money& newRecord, int count); **Money.h:**

#pragma once

#include <iostream>

class Money

{

private:

long rubles; // рубли

int kopecks; // копейки

public:

Money(long r = 0, int k = 0) : rubles(r), kopecks(k)

{

if (kopecks >= 100)

{

rubles += kopecks / 100;

kopecks %= 100;

}

}

void display() const;

Money operator+(const Money& m) const;

bool operator>(const Money& m) const;

bool operator<(const Money& m) const;

bool operator==(const Money& m) const;

long getRubles() const;

int getKopecks() const;

void setRubles(long r);

void setKopecks(int k);

};

Реализация методов

**file\_operations.cpp:**

#include "file\_operations.h"

void saveToFile(const std::vector<Money>& records, const std::string& filename)

{

std::ofstream outFile(filename, std::ios::binary);

for (const auto& record : records)

{

outFile.write(reinterpret\_cast<const char\*>(&record), sizeof(record));

}

outFile.close();

}

std::vector<Money> loadFromFile(const std::string& filename)

{

std::vector<Money> records;

std::ifstream inFile(filename, std::ios::binary);

Money record;

while (inFile.read(reinterpret\_cast<char\*>(&record), sizeof(record)))

{

records.push\_back(record);

}

inFile.close();

return records;

}

void removeRecords(std::vector<Money>& records, const Money& k1, const Money& k2)

{

records.erase(std::remove\_if(records.begin(), records.end(), [&](const Money& m)

{

return (m < k1 || m > k2);

}), records.end());

}

void doubleRecords(std::vector<Money>& records)

{

for (auto& record : records)

{

long r = record.getRubles();

int k = record.getKopecks() \* 2;

if (k >= 100)

{

r += k / 100;

k %= 100;

}

record.setRubles(r);

record.setKopecks(k);

}

}

void addRecords(std::vector<Money>& records, const Money& newRecord, int count)

{

for (int i = 0; i < count; ++i)

{

records.insert(records.begin(), newRecord);

}

}

**Money.cpp:**#include "Money.h"

void Money::display() const

{

std::cout << rubles << " рублей " << kopecks << " копеек" << std::endl;

}

Money Money::operator+(const Money& m) const

{

return Money(rubles + m.rubles, kopecks + m.kopecks);

}

bool Money::operator>(const Money& m) const

{

return rubles > m.rubles || (rubles == m.rubles && kopecks > m.kopecks);

}

bool Money::operator<(const Money& m) const

{

return rubles < m.rubles || (rubles == m.rubles && kopecks < m.kopecks);

}

bool Money::operator==(const Money& m) const

{

return rubles == m.rubles && kopecks == m.kopecks;

}

long Money::getRubles() const

{

return rubles;

}

int Money::getKopecks() const

{

return kopecks;

}

void Money::setRubles(long r)

{

rubles = r;

}

void Money::setKopecks(int k)

{

kopecks = k;

}

**Функция main**

#include <iostream>

#include <vector>

#include "Money.h"

#include "file\_operations.h"

void displayMenu()

{

std::cout << "1. Добавить запись\n";

std::cout << "2. Удалить записи\n";

std::cout << "3. Увеличить записи\n";

std::cout << "4. Смотреть записи\n";

std::cout << "5. Выход\n";

}

int main()

{

setlocale(LC\_ALL, "Russian");

std::vector<Money> records;

int choice;

while (true)

{

displayMenu();

std::cin >> choice;

// Проверка на корректность ввода

if (std::cin.fail())

{

std::cin.clear(); // Сбрасываем состояние ошибки

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); // Игнорируем неверный ввод

std::cout << "Неверный ввод! Пожалуйста, введите число от 1 до 5.\n";

continue; // Возвращаемся к началу цикла

}

switch (choice)

{

case 1:

{

long r;

int k;

std::cout << "Введите рубли и копейки: ";

std::cin >> r >> k;

records.emplace\_back(r, k);

saveToFile(records, "money.dat");

break;

}

case 2:

{

Money k1, k2;

long r1, r2;

int k1c, k2c;

std::cout << "Введите диапазон (рубли и копейки) для удаления:\n";

std::cout << "К1: ";

std::cin >> r1 >> k1c;

k1 = Money(r1, k1c);

std::cout << "К2: ";

std::cin >> r2 >> k2c;

k2 = Money(r2, k2c);

removeRecords(records, k1, k2);

saveToFile(records, "money.dat");

break;

}

case 3:

{

doubleRecords(records);

saveToFile(records, "money.dat");

break;

}

case 4:

{

records = loadFromFile("money.dat");

for (const auto& record : records)

{

record.display();

}

break;

}

case 5:

return 0;

default:

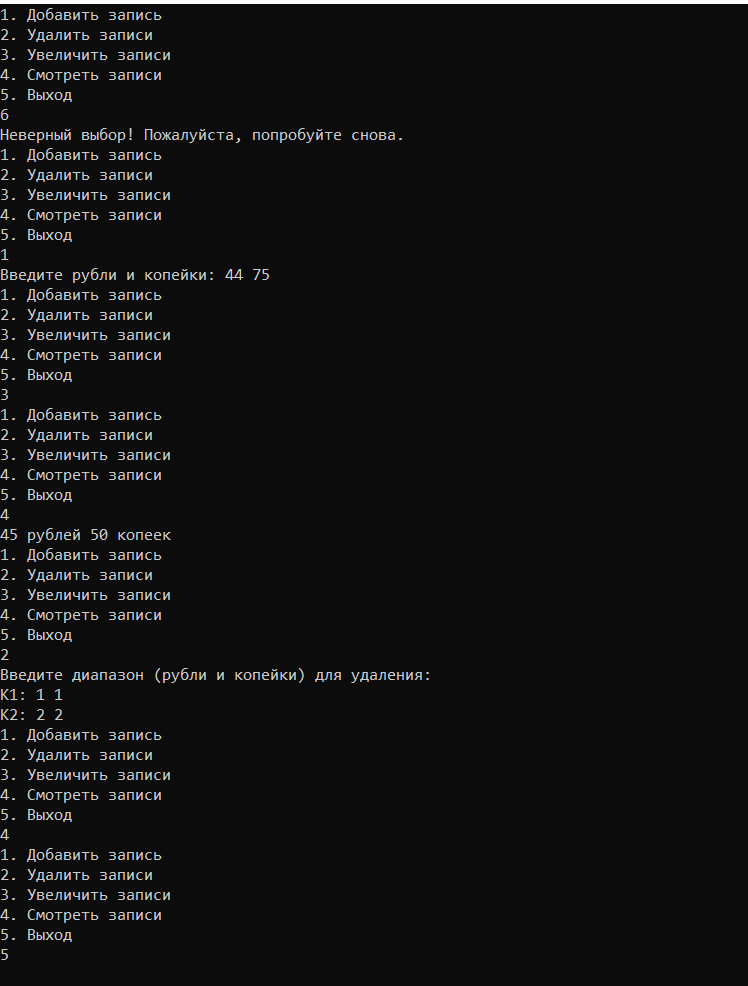
std::cout << "Неверный выбор! Пожалуйста, попробуйте снова.\n";

}

}

}

Результат работы:



Контрольные вопросы:

1. Что такое поток?

2. Какие типы потоков существуют?

3. Какую библиотеку надо подключить при использовании стандартных потоков?

4. Какую библиотеку надо подключить при использовании файловых потоков?

5. Какую библиотеку надо подключить при использовании строковых потоков?

6. Какая операция используется при выводе в форматированный поток?

7. Какая операция используется при вводе из форматированных потоков?

8. Какие методы используются при выводе в форматированный поток?

9. Какие методы используется при вводе из форматированного потока?

10. Какие режимы для открытия файловых потоков существуют?

11. Какой режим используется для добавления записей в файл?

12. Какой режим (комбинация режимов) используется в конструкторе ifstream file("f.txt")?

13. Какой режим (комбинация режимов) используется в конструкторе fstream file("f.txt")?

14. Какой режим (комбинация режимов) используется в конструкторе ofstream file("f.txt")?

15. Каким образом открывается поток в режиме ios::outios::app?

16. Каким образом открывается поток в режиме ios::out |ios::trunc?

17. Каким образом открывается поток в режиме ios::out |ios::in|ios::trunk?

18. Каким образом можно открыть файл для чтения?

19. Каким образом можно открыть файл для записи?

20. Привести примеры открытия файловых потоков в различных режимах.

21. Привести примеры чтения объектов из потока.

22. Привести примеры записи объектов в поток.

23. Сформулировать алгоритм удаления записей из файла.

24. Сформулировать алгоритм добавления записей в файл.

25. Сформулировать алгоритм изменения записей в файле.

Ответы:

1. **Что такое поток?**  
   Поток — это последовательность байтов, не зависящая от конкретного устройства, с которым производится обмен (например, оперативная память, файл на диске, клавиатура или принтер).
2. **Какие типы потоков существуют?**
   * Стандартные потоки (входные и выходные)
   * Строковые потоки (однонаправленные и двунаправленные)
   * Файловые потоки (однонаправленные и двунаправленные)
3. **Какую библиотеку надо подключить при использовании стандартных потоков?**  
   Необходимо подключить библиотеку #include <iostream>.
4. **Какую библиотеку надо подключить при использовании файловых потоков?**  
   Необходимо подключить библиотеку #include <fstream>.
5. **Какую библиотеку надо подключить при использовании строковых потоков?**  
   Необходимо подключить библиотеку #include <sstream>.
6. **Какая операция используется при выводе в форматированный поток?**  
   При выводе используется перегруженная операция сдвига влево operator<<.
7. **Какая операция используется при вводе из форматированных потоков?**  
   При вводе используется перегруженная операция сдвига вправо operator>>.
8. **Какие методы используются при выводе в форматированный поток?**  
   Методы put(char c) и write(const char\* buf, int size).
9. **Какие методы используются при вводе из форматированного потока?**  
   Методы get(char &c), read(char\* buf, int size), getline(char\* str, streamsize count).
10. **Какие режимы для открытия файловых потоков существуют?**  
    Основные режимы: ios::in, ios::out, ios::app, ios::trunc, ios::binary.
11. **Какой режим используется для добавления записей в файл?**  
    Режим ios::app используется для добавления записей в файл.
12. **Какой режим (комбинация режимов) используется в конструкторе ifstream file("f.txt")?**  
    Используется режим ios::in.
13. **Какой режим (комбинация режимов) используется в конструкторе fstream file("f.txt")?**  
    Используется режим ios::in | ios::out.
14. **Какой режим (комбинация режимов) используется в конструкторе ofstream file("f.txt")?**  
    Используется режим ios::out.
15. **Каким образом открывается поток в режиме ios::out | ios::app?**  
    Поток открывается для записи в конец файла, добавляя новые записи.
16. **Каким образом открывается поток в режиме ios::out | ios::trunc?**  
    Поток открывается для записи, при этом содержимое файла будет обрезано (удалено).
17. **Каким образом открывается поток в режиме ios::out | ios::in | ios::trunc?**  
    Поток открывается для чтения и записи с обрезкой содержимого файла.
18. **Каким образом можно открыть файл для чтения?**  
    Файл открывается с использованием ifstream с режимом ios::in.
19. **Каким образом можно открыть файл для записи?**  
    Файл открывается с использованием ofstream с режимом ios::out.

### Примеры открытия файловых потоков в различных режимах.

#include <fstream>

// Открытие входного файлового потока

std::ifstream inputFile("input.txt"); // Входной поток для чтения из файла

// Открытие выходного файлового потока

std::ofstream outputFile("output.txt"); // Выходной поток для записи в файл

// Открытие двунаправленного файлового потока

std::fstream file("data.txt", std::ios::in | std::ios::out); // Чтение и запись в файл

### Примеры чтения объектов из потока.

#include <fstream>

#include <iostream>

int main()

{

std::ifstream inputFile("data.txt");

int a;

double b;

char c;

inputFile >> a; // Чтение целого числа

inputFile >> b; // Чтение вещественного числа

inputFile >> c; // Чтение символа

inputFile.close();

return 0;

}

### Примеры записи объектов в поток.

#include <fstream>

#include <iostream>

int main()

{

std::ofstream outputFile("data.txt");

int a = 10;

double b = 5.67;

char c = 'x';

outputFile << a << '\n'; // Запись целого числа

outputFile << b << '\n'; // Запись вещественного числа

outputFile << c << '\n'; // Запись символа

outputFile.close();

return 0;

}

### 23. Алгоритм удаления записей из файла.

1. Открыть файл в режиме ios::in для чтения.
2. Создать временный файл в режиме ios::out для записи.
3. Считать данные из оригинального файла, пропуская записи, которые нужно удалить.
4. Записать оставшиеся данные во временный файл.
5. Закрыть оба файла.
6. Удалить оригинальный файл.
7. Переименовать временный файл, чтобы он соответствовал оригинальному.

### 24. Алгоритм добавления записей в файл.

1. Открыть файл в режиме ios::out | ios::app для добавления.
2. Записать новые данные в файл.
3. Закрыть файл.

### 25. Алгоритм изменения записей в файле.

1. Открыть файл в режиме ios::in для чтения.
2. Создать временный файл в режиме ios::out для записи.
3. Считать данные из оригинального файла.
4. При необходимости изменить записи во временном файле.
5. Записать измененные данные во временный файл.
6. Закрыть оба файла.
7. Удалить оригинальный файл.
8. Переименовать временный файл, чтобы он соответствовал оригинальному.

Ссылка:

**https://github.com/LeonidZhir/-**