**Министерство образования и науки Российской Федерации**

**Пермское федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский Национальный Исследовательский Политехнический Университет»**

**Электротехнический факультет**

**Кафедра «Информационные технологии и автоматизированные системы»**

**ОТЧЁТ**

По лабораторной работе №18.10 на тему

«Сохранение данных в файле с использованием потоков»

Вариант №11

Выполнил студент группы РИС-20-1б

Шумилов Лев Сергеевич

Проверил доцент кафедры ИТАС

Полякова О.А.

Пермь 2021

**Цель работы** –научиться работать с файловыми потоками ввода-вывода и с потоками ввода-вывода на консоли в классах посредством специальных методов и friend-операторов.

**Постановка задачи**

Задача – реализовать на языке программирования C++ программу, которая содержит классы с реализованными потоками ввода и вывода, работающую с некоторым файлом, продемонстрировать полученный файл.

**Исходные данные для варианта №11:**

* Класс-обработчик файлов;
* Класс-элемент: класс «Деньги» из предыдущих лабораторных работ;

**Анализ задачи**

Для решения задачи были использованы следующие средства:

1. Язык программирования C++ (Microsoft Visual C++)
2. Текстовый редактор Microsoft Visual Studio Code
3. Система контроля версий Git

Для работы с файлом и элементами классов, реализован класс FileManager, который будет содержать все необходимые метода для:

* Работы с «деньгами»;
* Добавление/удаление «денег»;
* Чтение «денег» с файла;
* Запись «денег» в файл;
* Ввод «денег» с клавиатуры;
* Вывод «денег» на экран;
* Обработка «денег» такая, как:
  + Деление всех денег пополам;
  + Удалить деньги, значение которых выше, чем заданное число;
  + Вставить новые деньги в массив денег;

Весь необходимый функционал объявлен в заголовочном файле:

#pragma once

#include "Money.h"

#include <vector>

using namespace std;

class FileManager

{

Money\* MoneyCollection;

int \_moneyCount = 0;

public:

string FileName;

FileManager();

FileManager(string fileName);

void MakeRandomMoneyCollection(int itemCount);

void AddNewMoney(Money newMoney);

void DeleteLastMoney();

void DeleteIfValueMoreThan(Money& comparedMoney);

void DivideByTwoEveryItem();

void InsertNewMoney(int index, int count);

void WriteToFile();

void ReadFromFile();

void DeleteFile();

void PrintCollection();

};

Также модернизации следует подвергнуть класс «Деньги», а именно:

* Добавить операторы ввода-вывода данных класса в:
  + Файл;
  + На экран;

Таким образом, объявление «Денег» будет выглядеть следующим образом:

#pragma once

#include <iostream>

#include <fstream>

using namespace std;

class Money

{

private:

long \_dollars;

int \_cents;

public:

int Dollars() { return \_dollars; }

void SetDollars(long dollars);

int Cents() { return \_cents; }

void SetCents(int cents);

void PrintMoney();

friend bool operator > (Money& first, Money& second);

friend bool operator < (Money& first, Money& second);

friend istream& operator >> (istream& in, Money& money);

friend ostream& operator << (ostream& out,Money& money);

friend fstream& operator >> (fstream& fin, Money& money);

friend fstream& operator << (fstream& fout, Money& money);

Money& operator++ ();

Money operator++ (int);

Money();

Money(long dollars, int cents);

Money(Money& parentMoney);

~Money();

};

Реализация операторов ввода и вывода:

istream &operator>>(istream &in, Money &money)

{

long dollars;

int cents;

cout << "dollars: ";

cin >> dollars;

cout << "cents";

cin >> cents;

money.SetDollars(dollars);

money.SetCents(cents);

return in;

}

ostream &operator<<(ostream &out, Money &money)

{

money.PrintMoney();

return out;

}

fstream &operator>>(fstream &fin, Money &money)

{

long dollars;

int cents;

fin >> dollars >> cents;

money.SetDollars(dollars);

money.SetCents(cents);

return fin;

}

fstream &operator<<(fstream &fout, Money &money)

{

fout << money.Dollars() << endl << money.Cents() << endl;

return fout;

}

Реализация файлового менеджера:

Запись данных в файл:

void FileManager::WriteToFile()

{

fstream streamWriter(FileName, ios::out | ios::trunc);

if (!streamWriter)

{

cerr << "Error: file opening error;\n";

return;

}

for (int i = 0; i < \_moneyCount; i++)

streamWriter << MoneyCollection[i] << endl;

streamWriter.close();

}

Чтение данных с файла:

void FileManager::ReadFromFile()

{

fstream streamReader(FileName, ios::in);

if (!streamReader)

{

cerr << "Error: file opening error;\n";

return;

}

Money money;

while (streamReader >> money)

AddNewMoney(money);

}

Удаление файла:

void FileManager::DeleteFile()

{

remove(FileName.c\_str());

}

Печать данных на экран:

void FileManager::PrintCollection()

{

for (int i = 0; i < \_moneyCount; i++)

{

cout << "Money #" << i + 1 << ":\n";

cout << MoneyCollection[i] << endl;

}

}

Конструкторы:

FileManager::FileManager()

{

FileName = "";

}

FileManager::FileManager(string fileName)

{

FileName = fileName;

}

Добавление «денег» в конец массива:

void FileManager::AddNewMoney(Money newMoney)

{

\_moneyCount++;

Money\* moreMonies = new Money[\_moneyCount];

for (int i = 0; i < \_moneyCount - 1; i++)

moreMonies[i] = MoneyCollection[i];

moreMonies[\_moneyCount - 1] = newMoney;

if (\_moneyCount > 1)

delete[] MoneyCollection;

MoneyCollection = moreMonies;

}

Удаление «денег» из массива:

void FileManager::DeleteLastMoney()

{

if (\_moneyCount > 0)

{

\_moneyCount--;

Money\* lessMonies = new Money[\_moneyCount];

for (int i = 0; i < \_moneyCount; i++)

lessMonies[i] = MoneyCollection[i];

delete[] MoneyCollection;

MoneyCollection = lessMonies;

}

else

cout << "Count of money cannot be < 0;\n";

}

Удаление «денег» при значении больше, чем нужное:

void FileManager::DeleteIfValueMoreThan(Money &comparedMoney)

{

int count = 0;

for (int i = 0; i < \_moneyCount; i++)

if (MoneyCollection[i] < comparedMoney)

count++;

Money\* suitMoney = new Money[count];

int pointer = -1;

for (int i = 0; i < \_moneyCount; i++)

if (MoneyCollection[i] < comparedMoney)

{

pointer++;

suitMoney[pointer] = MoneyCollection[i];

}

\_moneyCount = count;

delete[] MoneyCollection;

MoneyCollection = suitMoney;

}

Деление всею «денег» пополам:

void FileManager::DivideByTwoEveryItem()

{

for (int i = 0; i < \_moneyCount; i++)

{

long dollars = MoneyCollection[i].Dollars();

int cents = MoneyCollection[i].Cents();

MoneyCollection[i].SetDollars(dollars / 2);

if (dollars % 2 == 0)

MoneyCollection[i].SetCents(cents / 2);

else

MoneyCollection[i].SetCents((cents + 100) / 2);

}

}

Вставка «денег» в массив под нужным номером:

void FileManager::InsertNewMoney(int index, int count)

{

Money\* moreMoney = new Money[\_moneyCount + count];

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

moreMoney[i] = MoneyCollection[i];

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

moreMoney[index + i] = Money(111, 11);

for (int i = index + count; i < \_moneyCount + count; i++)

moreMoney[i] = MoneyCollection[i - count];

delete[] MoneyCollection;

MoneyCollection = moreMoney;

}

Создание случайной коллекции «денег»:

void FileManager::MakeRandomMoneyCollection(int itemCount)

{

\_moneyCount = itemCount;

MoneyCollection = new Money[\_moneyCount];

for (int i = 0; i < \_moneyCount; i++)

MoneyCollection[i] = Money(rand(), rand() % 100);

}

Главная программа, которая демонстрирует работу потоков ввода-вывода:

#include <iostream>

#include "FileManager.h"

using namespace std;

int main()

{

FileManager manager("F1.txt");

Money newMoney(4000, 00);

cout << "New random money collection:\n";

manager.MakeRandomMoneyCollection(40);

manager.PrintCollection();

cout << "Every money value has been divided by 2:\n";

manager.DivideByTwoEveryItem();

manager.PrintCollection();

cout << "Delete money if the value is more than 4000.00$:\n";

manager.DeleteIfValueMoreThan(newMoney);

manager.PrintCollection();

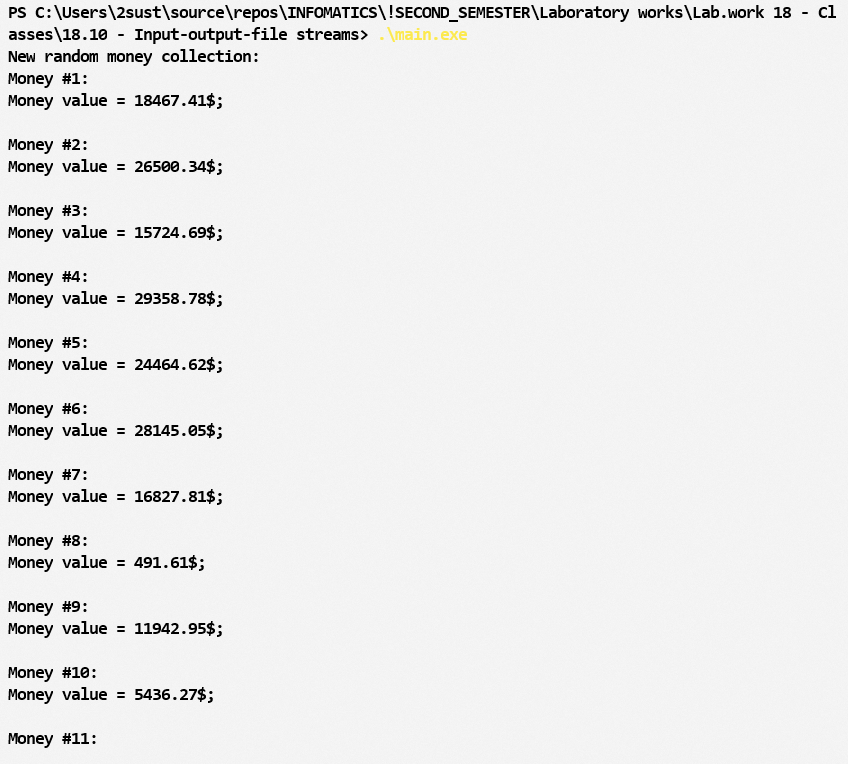
cout << "Write the last money data into F1.txt:\n";

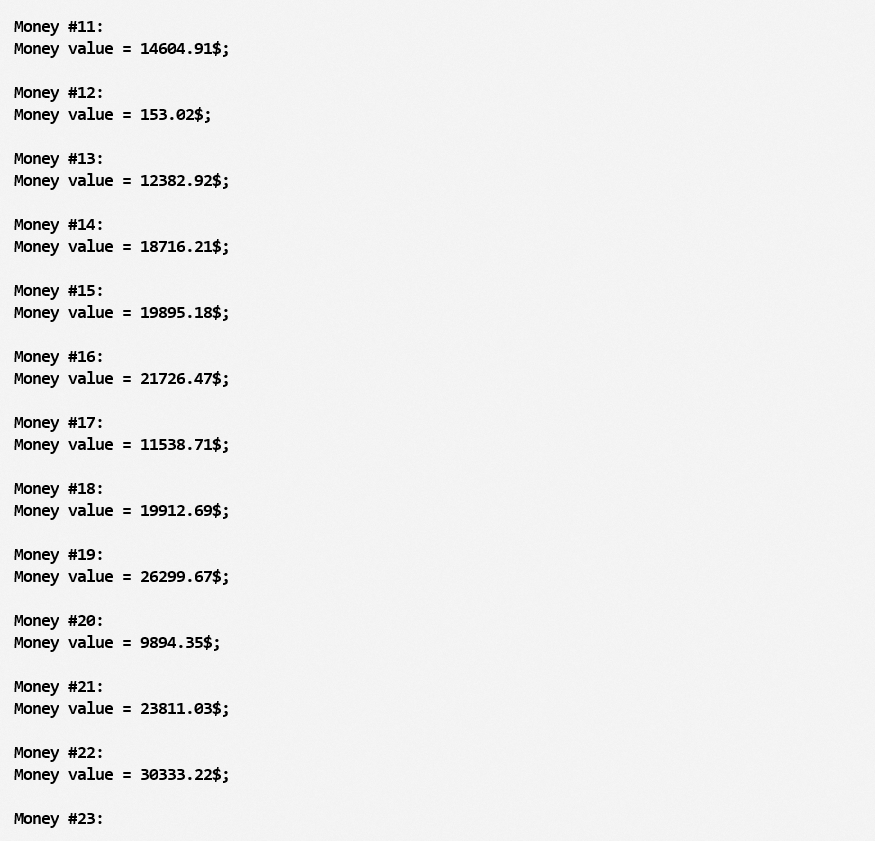
manager.WriteToFile();

return 0;

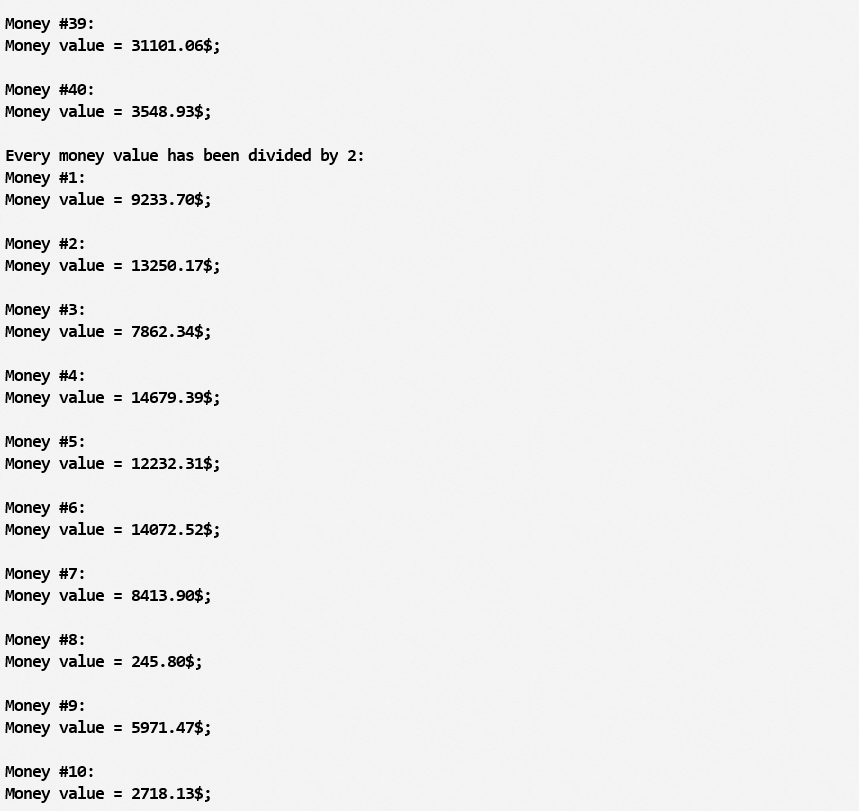
}

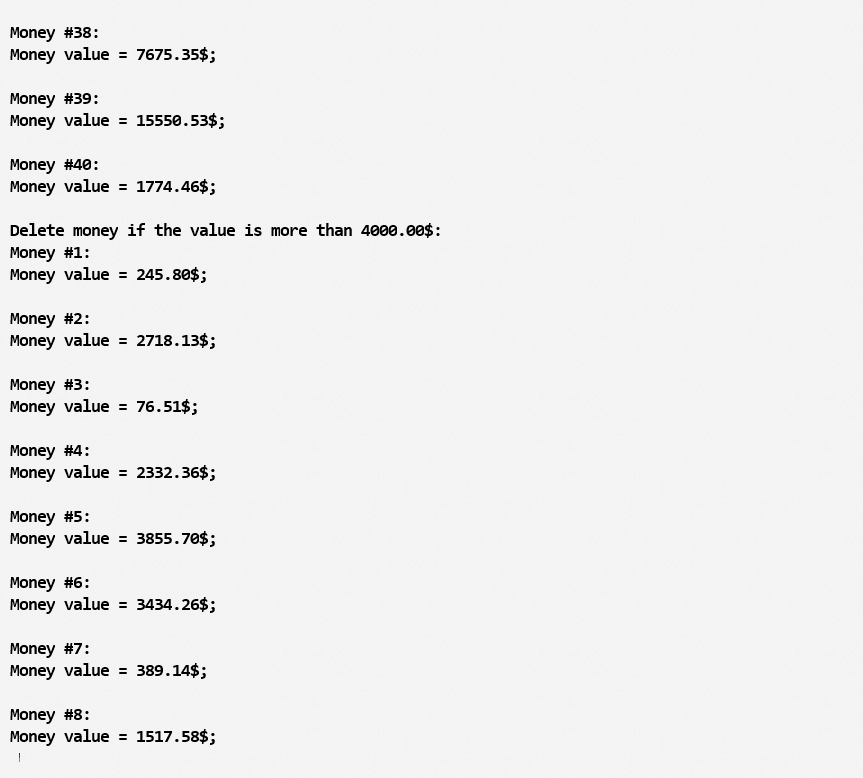
**Скриншоты выполненной программы:**

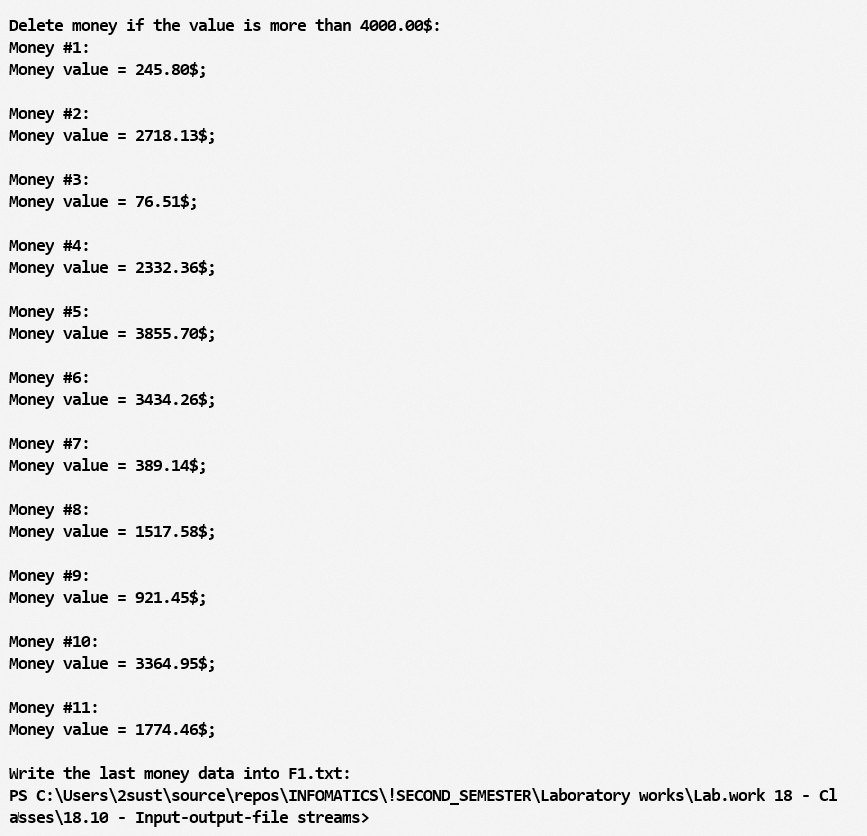


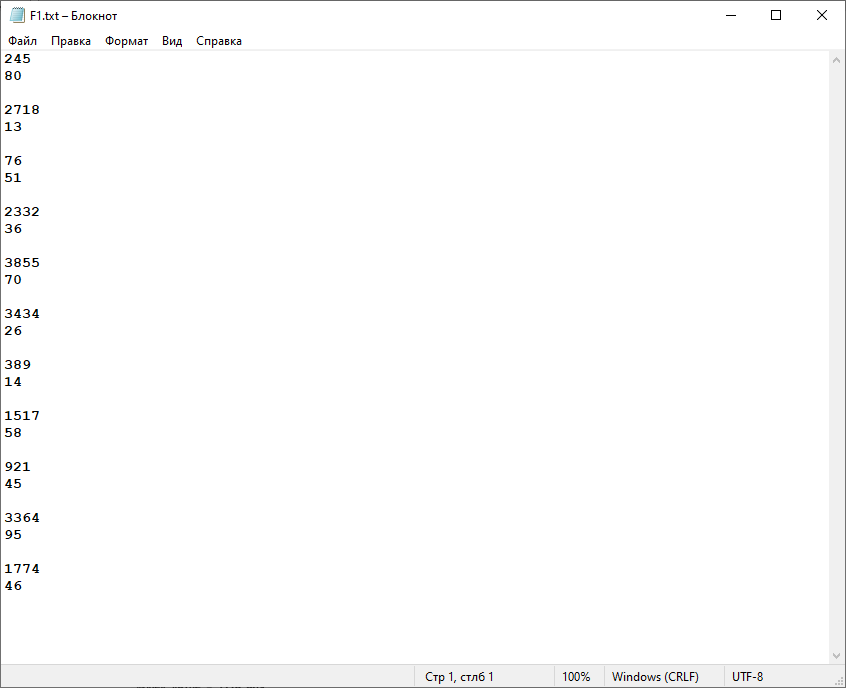












**Ответы на вопросы:**

**Что такое поток?**

Поток - определяется как последовательность байтов и не зависит от конкретного устройства, с которым производится обмен.

**Какие типы потоков существуют?**

Потоки бывают

* Стандартные: только однонаправленные, либо входные, либо выходные.
* Строковые: могут быть и однонаправленными и двунаправленными
* Файловые: могут быть и однонаправленными и двунаправленными.

**Какую библиотеку надо подключить при использовании стандартных потоков?**

#include <iostream>

**Какую библиотеку надо подключить при использовании файловых потоков?**

#include <fstream>

**Какую библиотеку надо подключить при использовании строковых потоков?**

#include <sstream>

**Какая операция используется при выводе в форматированный поток?**

operator<<

**Какая операция используется при вводе из форматированных потоков?**

operator>>

**Какие методы используются при выводе в форматированный поток?**

|  |  |  |
| --- | --- | --- |
| ostream& put(char c) | char c=’a’;  stream.put(c); | Записывает в поток  stream символ c |
| ostream& write(const char\* buf, int size) | char c=’a’; stream.write(&c); | Записывает в поток stream символ c |
| ostream& write(const char\* buf, int size) | char s[]=”string1”; stream.write(s,strlen  (s)); | Записывает в поток stream строку символов |

**Какие методы используется при вводе из форматированного потока?**

|  |  |  |
| --- | --- | --- |
| Прототип | Пример |  |
| int\_type get();  istream& get(char &c) | c= stream.get();  stream.get(c) | ввод из потока одного  символа |
| istream&read(char\*buf,int  size) | stream.read(&c,1) | ввод из потока одного  символа |

Ввод символьных массивов и строк выполняется с помощью операции >> до первого символа-разделителя (обычно пробела). Для ввод строк с пробелами используют методы get() и getline().

|  |  |  |
| --- | --- | --- |
| Прототип | Пример |  |
| istream& get(char\*str,  streamsize count) | stream.get(s,50) | ввод из потока строки с  пробелами (пока не встретиться признак конца файла eof или не будет введено 50 символов) |
| istream& get(char\*str,  streamsize count, char lim) | stream.get(s,50,  ’;’) | ввод из потока строки с  пробелами (пока не встретиться признак конца файла eof, или не будет введено 50 символов, или не встретится ; ) |
| istream&  getline(char\*str, streamsize count) | stream.getline(s,  50) | Работает также как get,  но удаляет из входного потока символ ‘\n’ |
| istream& get(char\*str,  streamsize count, char lim) | stream.getline(s,  50,’;’) |

Ввод в двоичные файлы производится методом read:

istream&read(char\*buf, streamsize size)

Метод читает size символов в массив buf. Символы разделители на ввод не влияют.

**Какие режимы для открытия файловых потоков существуют?**

Режимы открытия потока

|  |  |
| --- | --- |
| Режим | Описание |
| in | открыть поток для чтения (по умолчанию для ifstream) |
| out | (по умолчанию для оfstream) |
| trunk | удалить старое содержимое файла (по умолчанию для оfstream) |
| app | открыть поток для записи в конец файла |
| ate | открыть поток для чтения и/или записи и встать в конец файла |
| binary | открыть поток в двоичном режиме |

**Какой режим используется для добавления записей в файл?**

in.

**Какой режим (комбинация режимов) используется в конструкторе ifstream** **file(“f.txt”)?**

in.

**Какой режим (комбинация режимов) используется в конструкторе fstream file(“f.txt”)?**

In|out

**Какой режим (комбинация режимов) используется в конструкторе ofstream file(“f.txt”)?**

Out.

**Каким образом открывается поток в режиме ios::out|ios::app?**

fstream stream (”d:/files/number.txt”, ios::out|ios::app);

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

fstream stream (”d:/files/number.txt”, ios::out|ios::trunc);

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

fstream stream (”d:/files/number.txt”, ios::out | ios::in| ios::trunk);

**Каким образом можно открыть файл для чтения?**

ifstream stream (”d:/files/number.txt”);

**Каким образом можно открыть файл для записи?**

ofstream stream (”d:/files/number.txt”);

**Привести примеры открытия файловых потоков в различных режимах.**

ofstream stream (”d:/files/number.txt”);

ifstream stream;

stream.open(”c:/files/number.txt”);

ofstream stream (”d:/files/number.txt”, std::ios::app);

**Привести примеры чтения объектов из потока.**

int nl=0;

while(stream.get(c))

{

if (c==’\n’) nl++;

}

**Привести примеры записи объектов в поток.**

stream<<”smth”;

**Полный исходный код программы на языке программирования C++:**

FileManager.h

#pragma once

#include "Money.h"

#include <vector>

using namespace std;

class FileManager

{

Money\* MoneyCollection;

int \_moneyCount = 0;

public:

string FileName;

FileManager();

FileManager(string fileName);

void MakeRandomMoneyCollection(int itemCount);

void AddNewMoney(Money newMoney);

void DeleteLastMoney();

void DeleteIfValueMoreThan(Money& comparedMoney);

void DivideByTwoEveryItem();

void InsertNewMoney(int index, int count);

void WriteToFile();

void ReadFromFile();

void DeleteFile();

void PrintCollection();

};

FileManager.cpp

#include "FileManager.h"

#include <iostream>

#include <string>

using namespace std;

void FileManager::WriteToFile()

{

fstream streamWriter(FileName, ios::out | ios::trunc);

if (!streamWriter)

{

cerr << "Error: file opening error;\n";

return;

}

for (int i = 0; i < \_moneyCount; i++)

streamWriter << MoneyCollection[i] << endl;

streamWriter.close();

}

void FileManager::ReadFromFile()

{

fstream streamReader(FileName, ios::in);

if (!streamReader)

{

cerr << "Error: file opening error;\n";

return;

}

Money money;

while (streamReader >> money)

AddNewMoney(money);

}

void FileManager::DeleteFile()

{

remove(FileName.c\_str());

}

void FileManager::PrintCollection()

{

for (int i = 0; i < \_moneyCount; i++)

{

cout << "Money #" << i + 1 << ":\n";

cout << MoneyCollection[i] << endl;

}

}

FileManager::FileManager()

{

FileName = "";

}

FileManager::FileManager(string fileName)

{

FileName = fileName;

}

void FileManager::AddNewMoney(Money newMoney)

{

\_moneyCount++;

Money\* moreMonies = new Money[\_moneyCount];

for (int i = 0; i < \_moneyCount - 1; i++)

moreMonies[i] = MoneyCollection[i];

moreMonies[\_moneyCount - 1] = newMoney;

if (\_moneyCount > 1)

delete[] MoneyCollection;

MoneyCollection = moreMonies;

}

void FileManager::DeleteLastMoney()

{

if (\_moneyCount > 0)

{

\_moneyCount--;

Money\* lessMonies = new Money[\_moneyCount];

for (int i = 0; i < \_moneyCount; i++)

lessMonies[i] = MoneyCollection[i];

delete[] MoneyCollection;

MoneyCollection = lessMonies;

}

else

cout << "Count of money cannot be < 0;\n";

}

void FileManager::DeleteIfValueMoreThan(Money &comparedMoney)

{

int count = 0;

for (int i = 0; i < \_moneyCount; i++)

if (MoneyCollection[i] < comparedMoney)

count++;

Money\* suitMoney = new Money[count];

int pointer = -1;

for (int i = 0; i < \_moneyCount; i++)

if (MoneyCollection[i] < comparedMoney)

{

pointer++;

suitMoney[pointer] = MoneyCollection[i];

}

\_moneyCount = count;

delete[] MoneyCollection;

MoneyCollection = suitMoney;

}

void FileManager::DivideByTwoEveryItem()

{

for (int i = 0; i < \_moneyCount; i++)

{

long dollars = MoneyCollection[i].Dollars();

int cents = MoneyCollection[i].Cents();

MoneyCollection[i].SetDollars(dollars / 2);

if (dollars % 2 == 0)

MoneyCollection[i].SetCents(cents / 2);

else

MoneyCollection[i].SetCents((cents + 100) / 2);

}

}

void FileManager::InsertNewMoney(int index, int count)

{

Money\* moreMoney = new Money[\_moneyCount + count];

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

moreMoney[i] = MoneyCollection[i];

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

moreMoney[index + i] = Money(111, 11);

for (int i = index + count; i < \_moneyCount + count; i++)

moreMoney[i] = MoneyCollection[i - count];

delete[] MoneyCollection;

MoneyCollection = moreMoney;

}

void FileManager::MakeRandomMoneyCollection(int itemCount)

{

\_moneyCount = itemCount;

MoneyCollection = new Money[\_moneyCount];

for (int i = 0; i < \_moneyCount; i++)

MoneyCollection[i] = Money(rand(), rand() % 100);

}

Money.h

#pragma once

#include <iostream>

#include <fstream>

using namespace std;

class Money

{

private:

long \_dollars;

int \_cents;

public:

int Dollars() { return \_dollars; }

void SetDollars(long dollars);

int Cents() { return \_cents; }

void SetCents(int cents);

void PrintMoney();

friend bool operator > (Money& first, Money& second);

friend bool operator < (Money& first, Money& second);

friend istream& operator >> (istream& in, Money& money);

friend ostream& operator << (ostream& out,Money& money);

friend fstream& operator >> (fstream& fin, Money& money);

friend fstream& operator << (fstream& fout, Money& money);

Money& operator++ ();

Money operator++ (int);

Money();

Money(long dollars, int cents);

Money(Money& parentMoney);

~Money();

};

Money.cpp

#include "Money.h"

#include <string>

#include <iostream>

using namespace std;

void Money::SetDollars(long dollars)

{

if (dollars < 0)

cout << "Error: dollars less than 0;\n";

else

\_dollars = dollars;

}

void Money::SetCents(int cents)

{

if (cents < 0 || cents >= 100)

cout << "Error: cents less than 0 or bigger than 100;\n";

else

\_cents = cents;

}

void Money::PrintMoney()

{

string centStr = (\_cents < 10) ? '0' + to\_string(\_cents) : to\_string(\_cents);

cout << "Money value = " << \_dollars << '.' << centStr << "$;\n";

}

Money& Money::operator++()

{

\_cents++;

if (\_cents == 100)

{

\_cents = 0;

\_dollars++;

}

return \*this;

}

Money Money::operator++(int)

{

Money temp(\_dollars, \_cents);

++(\*this);

return temp;

}

Money::Money()

{

\_dollars = 0;

\_cents = 0;

}

Money::Money(long dollars, int cents)

{

SetDollars(dollars);

SetCents(cents);

}

Money::Money(Money& parentMoney)

{

\_dollars = parentMoney.\_dollars;

\_cents = parentMoney.\_cents;

}

Money::~Money() {}

bool operator>(Money& first, Money& second)

{

if (first.Dollars() == second.Dollars())

return first.Cents() > second.Cents();

else

return first.Dollars() > second.Dollars();

}

bool operator<(Money& first, Money& second)

{

if (first.Dollars() == second.Dollars())

return first.Cents() < second.Cents();

else

return first.Dollars() < second.Dollars();

}

istream &operator>>(istream &in, Money &money)

{

long dollars;

int cents;

cout << "dollars: ";

cin >> dollars;

cout << "cents";

cin >> cents;

money.SetDollars(dollars);

money.SetCents(cents);

return in;

}

ostream &operator<<(ostream &out, Money &money)

{

money.PrintMoney();

return out;

}

fstream &operator>>(fstream &fin, Money &money)

{

long dollars;

int cents;

fin >> dollars >> cents;

money.SetDollars(dollars);

money.SetCents(cents);

return fin;

}

fstream &operator<<(fstream &fout, Money &money)

{

fout << money.Dollars() << endl << money.Cents() << endl;

return fout;

}

main.cpp

#include <iostream>

#include "FileManager.h"

using namespace std;

int main()

{

FileManager manager("F1.txt");

Money newMoney(4000, 00);

cout << "New random money collection:\n";

manager.MakeRandomMoneyCollection(40);

manager.PrintCollection();

cout << "Every money value has been divided by 2:\n";

manager.DivideByTwoEveryItem();

manager.PrintCollection();

cout << "Delete money if the value is more than 4000.00$:\n";

manager.DeleteIfValueMoreThan(newMoney);

manager.PrintCollection();

cout << "Write the last money data into F1.txt:\n";

manager.WriteToFile();

return 0;

}