**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 №8 на тему

«Блоковый ввод-вывод»

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 двоичными файлами, организовывать ввод-вывод структурированной информации на внешних носителях.

**Постановка задачи**

Задача – реализовать программу, в которой выполняются следующие задачи:

1. Сформировать двоичный файл из элементов, заданной в варианте структуры;
2. Распечатать содержимое сформированного файла;
3. Выполнить удаление и добавление элементов в соответствии со своим вариантом;

При этом должны соблюдаться следующие условия:

1. Формирование, печать, добавление и удаление файлов оформить в виде функций;
2. Предусмотреть сообщения об ошибках при открытии файла и выполнении операций ввода-вывода;

Исходные данные для **варианта №11**:

Структура «Видеокассета», которая содержит следующие поля:

* Название фильма
* Режиссёр
* Продолжительность
* Цена кассеты

Задачи:

1. Удалить все элементы с ценой выше заданной;
2. Добавить 3 элемента в конец файла;

**Анализ задачи**

Для того, чтобы решить задачу, нужно написать программу, которая выполняет следующие действия:

1. Принимает данные, введённые пользователем с клавиатуры;
2. Открывает файл для записи данных в него;
3. Записывает введённые данные в файл;
4. Извлекает данные с этого же файла и записывает их в массив;
5. Удаляет из массива элементы по условию, который вводит пользователь;
6. Добавляет в конец файла 3 новых элемента, данные которого вводит пользователь, в конец файла;

Для решения задачи были использованы следующие средства:

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

В первую очередь, программа будет работать с элементами структур «Видеокассета», в C++ данная структура объявляется следующим образом:

struct Videotape

{

char filmName[MAX\_NAME\_LENGTH];

char producer[MAX\_NAME\_LENGTH];

int timeDuration;

int cost;

};

Где MAX\_NAME\_LENGTH – это максимальная длина строки, она объявлена в виде директивы #define. Точно таким же образом заданы следующие параметры:

1. Максимальная длина массива структур;
2. Количество элементов, которые нужно добавить после удаления элементов;
3. Сообщения об успешном выполнении и об ошибках;

#define MAX\_LENGTH 100

#define MAX\_NAME\_LENGTH 30

#define ADD\_COUNT 3

#define OPEN\_FILE\_ERROR "Error: the file is not open!!!"

#define WRITE\_FILE\_ERROR "Error: writing to file is failed!!!"

#define SUCCESS "All file operations was successfully done!!!"

Для того, чтобы программа могла работать с файлами и выполнять операции ввода-вывода на консоль, были подключены следующие заголовочные файлы:

1. stdio.h – для работы с файлами
2. iostream – для ввода с клавиатуры и вывода на консоль

#include <iostream>

#include <stdio.h>

using namespace std;

Чтобы вводить **целые натуральные** числа с клавиатуры (в том числе попросить новый ввод, если введено не натуральное число), была реализована функция InputNaturalNum.

int InputNaturalNum()

{

int number;

do

cin >> number;

while (number < 1);

return number;

}

Для того чтобы отлавливать ошибки во время выполнения программы, была добавлена глобальная переменная \_IsProcessSuccessful, которая в начале выполнения равна true. Если во время выполнения операций чтения данных из файла или ввода данных в файл произойдёт ошибка, то значение этой переменной будет изменено на false.

bool \_IsProcessSuccessful = true;

Также каждая функция, отвечающая за работу с файлами, будет возвращать значения типа bool для того, чтобы отлавливать ошибки сразу после их выполнения (если функция успешно выполнена, то возвращается true, если произошла ошибка, то возвращается false).

bool WriteVideotape(FILE\* fileStream, Videotape\* tape)

{

if (fileStream != NULL)

{

fwrite(tape, sizeof(Videotape), 1, fileStream);

return (ferror == NULL)? false : true;

}

else

return false;

}

bool ReadVideotape(FILE\* fileStream, Videotape\* tape)

{

if ((fileStream != NULL) && !feof(fileStream))

{

fread(tape, sizeof(Videotape), 1, fileStream);

return true;

}

else

return false;

}

Переменная типа FILE\* необходима для того, чтобы открыть файл в программе и задать режим работы с файлом (wb – двоичная запись, rb – двоичное чтения, ab – добавление двоичной записи в файл).

Функции fread и fwrite отвечают за непосредственное файловое чтение и запись, соответственно: в качестве параметров они принимают:

* указатель на элемент, который нужно записать;
* количество байтов, элемента, который записывается;
* количество элементов;
* переменная типа FILE\* – файл, с которым будет работать функция;

Затем идёт проверка на ошибки и возвращается соответствующее значение типа bool.

Для того, чтобы организовать ввод и вывод подобных структур, были реализованы соответствующие функции: InputVideotape и PrintVideotapes. InputVideotape позволяет ввести с клавиатуры данные одной структуры, а PrintVideotapes – вывод всех структур в массиве. Чтобы вводить строки (в том числе пробелы), используется функция cin.getline:

Videotape InputVideotape()

{

Videotape input;

cin.ignore();

cout << "name: ";

cin.getline(input.filmName, MAX\_NAME\_LENGTH, '\n');

cout << "producer: ";

cin.getline(input.producer, MAX\_NAME\_LENGTH, '\n');

cout << "duration: ";

input.timeDuration = InputNaturalNum();

cout << "cost: ";

input.cost = InputNaturalNum();

cout << endl;

return input;

}

void PrintVideotapes(Videotape\* tapes, int tapesCount)

{

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

{

cout << "Videotype #" << i + 1 << ": " << endl;

cout << " film's name - " << (tapes + i)->filmName << endl;

cout << " film's producer - " << (tapes + i)->producer << endl;

cout << " film's duration - " << (tapes + i)->timeDuration

<< " minutes" << endl;

cout << " videotape's cost - " << (tapes + i)->cost << " dollars"

<< endl << endl;

}

}

Также, необходимо реализовать функцию для удаления элемента из массива. Алгоритм удаления элемента прост:

1. Начиная с номера элемента, который нужно удалить присваивается значение элемента с текущим индексом на значение элемента со следующим индексом (сдвиг элементов массива)
2. Удаляется последний элемент массива (элементам присваиваем значение 0 и пустую строку
3. Уменьшаем длину рабочего массива на 1

void DeleteVideotape(Videotape\* tapes, int deleteIndex, int& tapesCount)

{

for (int i = deleteIndex; i < tapesCount - 1; i++)

\*(tapes + i) = \*(tapes + i + 1);

\*(tapes + tapesCount - 1) = Videotape { "", "", 0, 0 };

tapesCount--;

}

Реализация необходимых функций готова, теперь можно приступать к написанию основной программы в методе main().

В начале выполнения программа сразу обращается к файлу F.dat (для удобства файл был помещён в отдельную папку Work files) и проверяет успешно ли файл открыт. Если открытие произошло успешно, то выводятся сообщения о просьбе ввода данных с клавиатуры, происходит непосредственный приём данных.

После каждой попытки записи проходит проверка на ошибки во время выполнения записи в файл после ввода данных каждого элемента с клавиатуры (используется цикл while, в котором идёт проверка на количество введённых элементов и на наличие ошибок). Если такие присутствуют, выводится соответствующее сообщение об ошибке и выполнение программы завершается.

FILE\* file = fopen("Work files\\F.dat", "wb");

if (file != NULL)

{

Videotape tapes[MAX\_LENGTH];

cout << "Input count of videotapes (1 to 100):" << endl;

int tapesCount = InputNaturalNum();

cout << "Input data for " << tapesCount << " new videotapes:"

<< endl;

int i = 0;

while ((i < tapesCount) && \_IsProcessSuccessful)

{

Videotape workVideotape = InputVideotape();

\_IsProcessSuccessful = WriteVideotape(file, &workVideotape);

i++;

}

fclose(file);

if (\_IsProcessSuccessfil)

{

//the next code will be here!!!

}

else

cout << WRITE\_FILE\_ERROR << endl;

}

else

cout << OPEN\_FILE\_ERROR << endl;

После успешной работы с файлом, необходимо корректно закрыть файловый поток, воспользовавшись функцией fclose.

После того, как файл успешно сформирован, из него нужно извлечь данные: сначала открывается файл в режиме для чтения (для этого можно присвоить переменной file новое значение), затем происходит запись в массив с помощью функции ReadVideotape, описанной выше.

file = fopen("Work files\\F.dat", "rb");

i = 0;

while ((i < tapesCount) && \_IsProcessSuccessful)

{

\_IsProcessSuccessful = ReadVideotape(file, &tapes[i]);

i++;

}

fclose(file);

if (\_IsProcessSuccessful)

{

//the next code will be here!!!

}

else

cout << OPEN\_FILE\_ERROR << endl;

После того, как данные были успешно извлечены из файла, можно вывести содержимое файла при помощи описанной выше функции PrintVideotapes.

PrintVideotapes(tapes, tapesCount);

Далее стоит задача удалить те элементы массива видеокассет (с помощью функции DeleteVideotape, которая была описана выше), в которых указана цена, которая выше, чем число, введённое пользователем.

Чтобы случайно не перескочить через один элемент, было принято решение реализовать **перебор элементов из конца в начало**: в таком случае, направление перебора и сдвиг элементов после каждого удаление идёт в одну сторону, таким образом, программа не пропустит ни один элемент при проверке. Затем идёт печать оставшихся элементов.

cout << "Input the cost of videotape, which will not be exceeded:"

<< endl;

int maxCost = InputNaturalNum();

for (int i = tapesCount - 1; i >= 0; i--)

{

if (tapes[i].cost > maxCost)

DeleteVideotape(tapes, i, tapesCount);

}

cout << "Left videotapes:" << endl;

PrintVideotapes(tapes, tapesCount);

Теперь остаётся решить последнюю задачу: добавить 3 новых элемента в конец файла. Поскольку функция WriteVideotape в качестве одного из элементов принимает переменную типа FILE\*, то есть сам файловый поток, то можно без проблем воспользоваться этой же функцией, предварительно присвоив переменной file режим добавления (ab).

Таким же вышеописанным образом (через цикл while) идёт запись в файл и проверка на наличие ошибок. После того, как цикл завершён, выводится сообщение об успешном выполнении всех операций или об ошибке, при их наличии.

cout << "Input data for " << ADD\_COUNT << " new videotapes:" << endl;

file = fopen("Work files\\F.dat", "ab");

i = 0;

while ((i < ADD\_COUNT) && \_IsProcessSuccessful)

{

tapes[tapesCount] = InputVideotape();

\_IsProcessSuccessful = WriteVideotape(file, &tapes[tapesCount]);

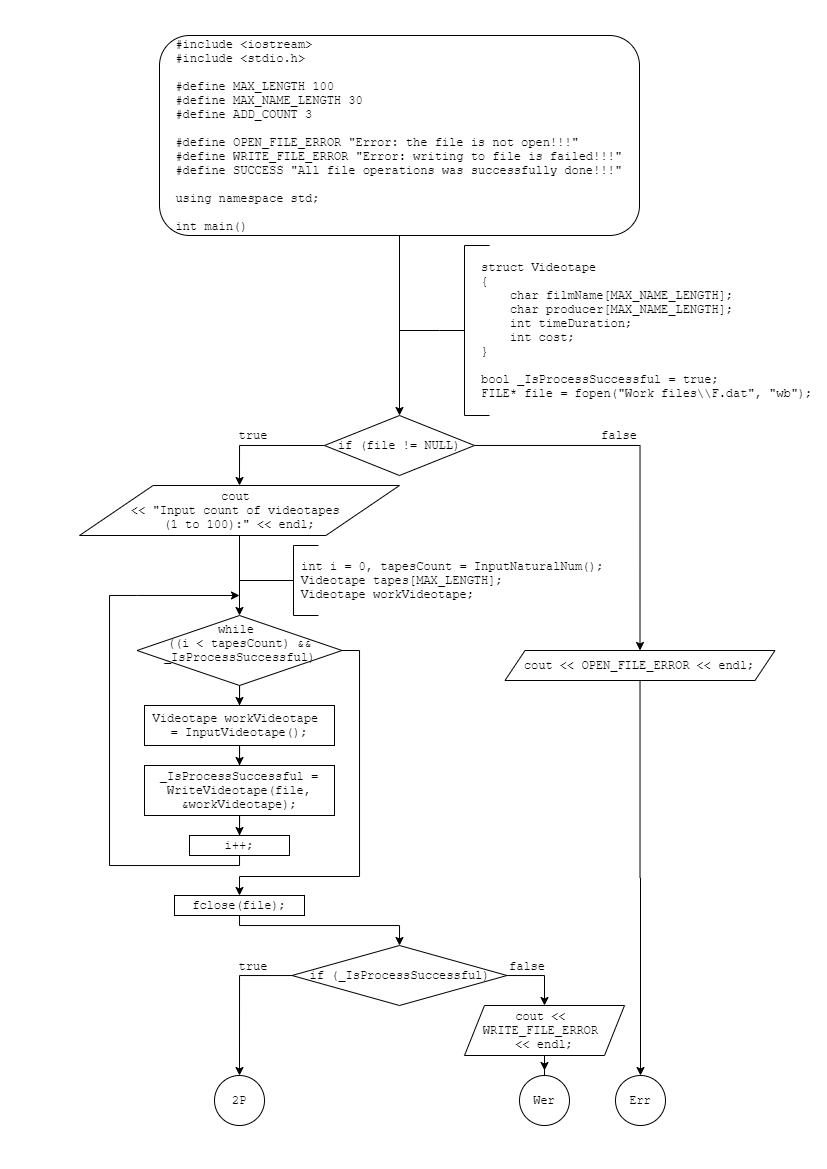
tapesCount++;

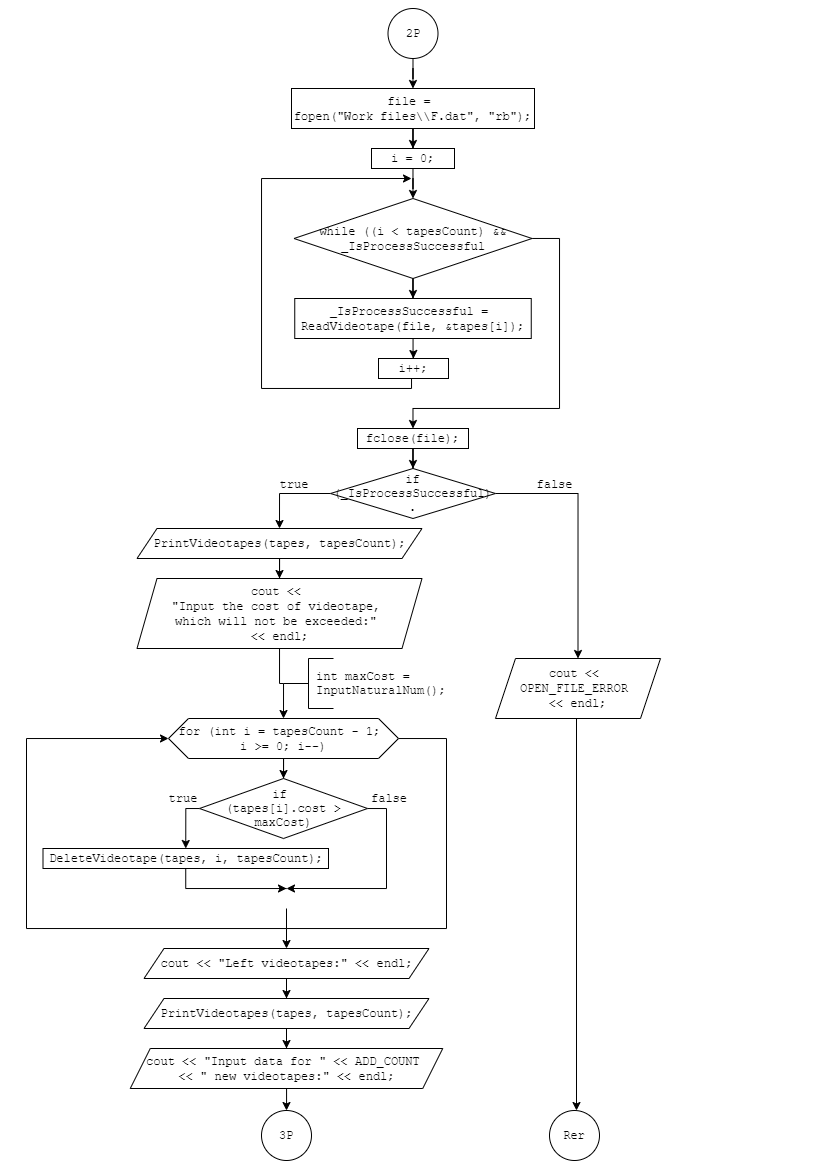
i++;

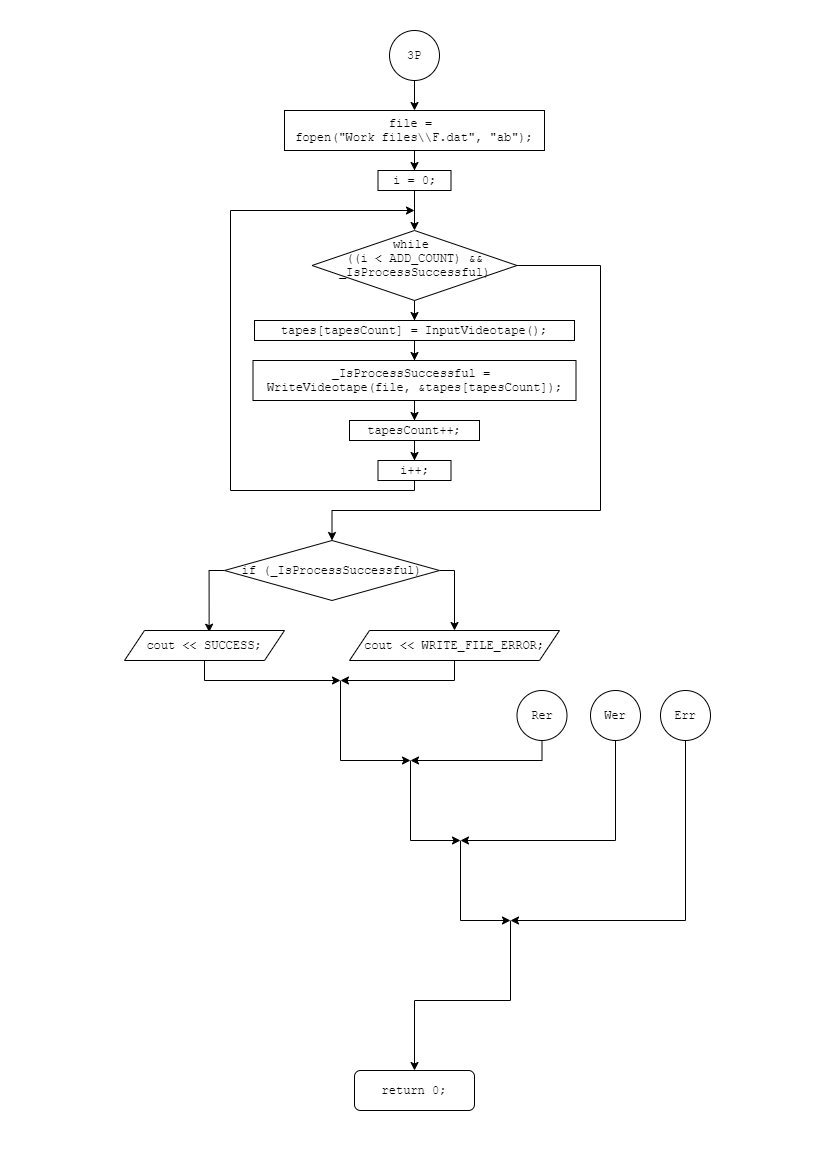
}

cout << (\_IsProcessSuccessful)? SUCCESS : WRITE\_FILE\_ERROR;

**Блок-схема** решённой задачи:







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

#include <iostream>

#include <stdio.h>

#define MAX\_LENGTH 100

#define MAX\_NAME\_LENGTH 30

#define ADD\_COUNT 3

#define OPEN\_FILE\_ERROR "Error: the file is not open!!!"

#define WRITE\_FILE\_ERROR "Error: writing to file is failed!!!"

#define SUCCESS "All file operations was successfully done!!!"

using namespace std;

bool \_IsProcessSuccessful = true;

struct Videotape

{

char filmName[MAX\_NAME\_LENGTH];

char producer[MAX\_NAME\_LENGTH];

int timeDuration;

int cost;

};

int InputNaturalNum()

{

int number;

do

cin >> number;

while (number < 1);

return number;

}

Videotape InputVideotape()

{

Videotape input;

cin.ignore();

cout << "name: ";

cin.getline(input.filmName, MAX\_NAME\_LENGTH, '\n');

cout << "producer: ";

cin.getline(input.producer, MAX\_NAME\_LENGTH, '\n');

cout << "duration: ";

input.timeDuration = InputNaturalNum();

cout << "cost: ";

input.cost = InputNaturalNum();

cout << endl;

return input;

}

bool WriteVideotape(FILE\* fileStream, Videotape\* tape)

{

if (fileStream != NULL)

{

fwrite(tape, sizeof(Videotape), 1, fileStream);

return (ferror == NULL)? false : true;

}

else

return false;

}

bool ReadVideotape(FILE\* fileStream, Videotape\* tape)

{

if ((fileStream != NULL) && !feof(fileStream))

{

fread(tape, sizeof(Videotape), 1, fileStream);

return true;

}

else

return false;

}

void PrintVideotapes(Videotape\* tapes, int tapesCount)

{

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

{

cout << "Videotype #" << i + 1 << ": " << endl;

cout << " film's name - " << (tapes + i)->filmName << endl;

cout << " film's producer - " << (tapes + i)->producer << endl;

cout << " film's duration - " << (tapes + i)->timeDuration << " minutes" << endl;

cout << " videotape's cost - " << (tapes + i)->cost << " dollars" << endl << endl;

}

}

void DeleteVideotape(Videotape\* tapes, int deleteIndex, int& tapesCount)

{

for (int i = deleteIndex; i < tapesCount - 1; i++)

\*(tapes + i) = \*(tapes + i + 1);

\*(tapes + tapesCount - 1) = Videotape { "", "", 0, 0 };

tapesCount--;

}

int main()

{

FILE\* file = fopen("Work files\\F.dat", "wb");

if (file != NULL)

{

Videotape tapes[MAX\_LENGTH];

cout << "Input count of videotapes (1 to 100):" << endl;

int tapesCount = InputNaturalNum();

cout << "Input data for " << tapesCount << " new videotapes:" << endl;

int i = 0;

while ((i < tapesCount) && \_IsProcessSuccessful)

{

Videotape workVideotape = InputVideotape();

\_IsProcessSuccessful = WriteVideotape(file, &workVideotape);

i++;

}

fclose(file);

if (\_IsProcessSuccessful)

{

file = fopen("Work files\\F.dat", "rb");

i = 0;

while ((i < tapesCount) && \_IsProcessSuccessful)

{

\_IsProcessSuccessful = ReadVideotape(file, &tapes[i]);

i++;

}

fclose(file);

if (\_IsProcessSuccessful)

{

PrintVideotapes(tapes, tapesCount);

cout << "Input the cost of videotape, which will not be exceeded:" << endl;

int maxCost = InputNaturalNum();

for (int i = tapesCount - 1; i >= 0; i--)

{

if (tapes[i].cost > maxCost)

DeleteVideotape(tapes, i, tapesCount);

}

cout << "Left videotapes:" << endl;

PrintVideotapes(tapes, tapesCount);

cout << "Input data for " << ADD\_COUNT << " new videotapes:" << endl;

file = fopen("Work files\\F.dat", "ab");

i = 0;

while ((i < ADD\_COUNT) && \_IsProcessSuccessful)

{

tapes[tapesCount] = InputVideotape();

\_IsProcessSuccessful = WriteVideotape(file, &tapes[tapesCount]);

tapesCount++;

i++;

}

cout << (\_IsProcessSuccessful) ? SUCCESS : WRITE\_FILE\_ERROR;

}

else

cout << OPEN\_FILE\_ERROR << endl;

}

else

cout << WRITE\_FILE\_ERROR << endl;

}

else

cout << OPEN\_FILE\_ERROR << endl;

system("pause");

return 0;

}

**Скриншоты** выполненной программы:

