Лабораторная работа №4

Работа с файловыми потоками в С++.  
Двоичные файлы

**Цель работы:** получить практические навыки решения задач с использованием двоичных файлов на языке С++ с использованием принципов объектно-ориентированного и визуального программирования.

**Порядок выполнения лабораторной работы**

1. Изучить теоретические сведения. Разобрать представленные примеры. Запустить первый, второй и пятый примеры, продемонстрировать их преподавателю, если примеры не были показаны на занятии, результаты их работы должны быть включены в отчет.
2. Получить вариант индивидуального задания у преподавателя. Предполается реализация двух задач: первая задача представлена в пункте лабораторное задание, вторая повторяется из предыдущей работы (вторая группа заданий), только теперь необходима реализация данной задачи при помощи двоичных файлов (объявить структуру и записывать/читать ее из двоичного файла). При необходимости перегрузить операторы << и >> для работы со стандартными потоковыми классами.
3. Реализовать полученные задания. Для выбора решения задачи пользователем реализовать меню. Любое общение с пользователем должно сопровождаться поясняющимся текстом. Также в программе должно быть реализовано минимум 2 исключительных ситуации. Исключительные ситуации должны обрабатываться в отдельном классе. Обязательна оконная (формочная) реализация программы с использованием элементов управления.
4. Показать результат работы программы преподавателю.
5. Защитить лабораторную работу.

**Требования к отчету**

Отчет должен содержать:

1. Цель работы.

2. Задание.

3. Словесное описание исключительных ситуаций.

4. UML диаграмма классов.

5. Блок-схема программы.

6. Текст программы на языке C++ с комментариями.

7. Тесты (копии выходных форм программы/сформированных файлов).

8. Выводы.

## Теоретические сведения

**Cохранение данных в двоичных файлах**

*Сохранение в двоичных файлах данных стандартных типов.*

Для того, чтобы открыть двоичный файл, необходимо задать режим доступа ios::binary (в некоторых компиляторах С++ - ios::bin).

Двоичные файлы более компактны и в некоторых случаях более удобны для обработки.

Для создания выходного файла создают объект

ofstream out\_fil (”Outfil.dat”,ios::out | ios::binary);

if (! out\_fil) { cerr<<”Error: Outfil.dat”<<endl;

exit(1);

}

Для того, чтобы открыть существующий двоичный файл для чтения, нужно создать объект

ifstream in\_fil (”Infil.dat”, ios::in | ios::binary);

if (! in\_fil) { cerr<<”Error: Infil.dat”<<endl;

exit(2);

}

К сожалению, созданные объекты in\_fil и out\_fil не слишком приспособлены для работы с двоичными файлами и требуют некоторых дополнительных действий, необходимых для корректной работы.

**Пример 1.** Запись значения типа double в двоичный файл.

# include <fstream>

# include <iostream>

# include <stdlib.h>

using namespace std;

class bin\_outstream :public ofstream

{

public:

bin\_outstream(const char \*fn) : ofstream(fn, ios::out | ios::binary){}

void writeOurDate(const void\*, int);

ofstream &operator<<(double d) {

writeOurDate(&d, sizeof(d));

return \*this;

}

};

int main()

{

bin\_outstream bin\_out("B\_out.dat");

if (!bin\_out)

{

cerr << "Unable to write to B\_out.dat" << endl;

system("pause");

exit(1);

}

double d = 5.252;

bin\_out << d;

bin\_out << d\*d;

d = 5.2E-5;

bin\_out << d;

system("pause");

return 0;

}

void bin\_outstream::writeOurDate(const void \*Ptr, int len)

{

if (!Ptr) return;

if (len <= 0) return;

write((char\*)Ptr, len);

}

**Пример 2.** Чтение значений типа double из двоичного файла.

# include <fstream>

# include <iostream>

# include <stdlib.h>

using namespace std;

class bin\_instream : public ifstream

{

public:

bin\_instream(const char \*fn) : ifstream(fn, ios::in | ios::binary){}

void readOurDate(void\*, int);

bin\_instream &operator>>(double &d) {

readOurDate(&d, sizeof(d));

return \*this;

}

};

int main()

{

bin\_instream bin\_in("B\_in.dat");

if (!bin\_in)

{

cerr << "Unable to open B\_in.dat" << endl;

system("pause");

exit(1);

}

double d;

long count = 0;

bin\_in >> d;

while (!bin\_in.eof())

{

cout << ++count << ':' << d << endl;

bin\_in >> d;

}

system("pause");

return 0;

}

void bin\_instream::readOurDate(void \*p, int len)

{

if (!p) return;

if (len <= 0) return;

read((char\*)p, len);

}

Для работы с файловыми потоками любого из стандартных типов, нужно перегрузить операторы ввода и вывода под требуемый тип данных или воспользоваться шаблоном класса, задаваемым с помощью ключевого слова template.

Сохранение в двоичных файлах данных, имеющих тип, создаваемый пользователем.

Иногда возникает необходимость сохранить в файле данные, структура которых задается программистом. В этих случаях задают класс, содержащий данные и функции, перегруженные операторы ввода и вывода под эти данные.

**Пример 3.**

Объявим структуру

struct mountine {

char name[20]; //название горы

int altitude; //высота над уровнем моря

int complicate; //сложность

};

mountine mount;

Для сохранения информации в двоичном файле выполняют следующее:

ofstream fil\_out(”mountines.txt”, ios\_base::app);

fil\_out << mount.name << ” ” << mount.altitude << ’ ’ << mount.complicate <<”\n”;

Для сохранения той же информации в двоичном файле выполняют следующее:

ofstrem fil\_out(”mountines.dat”, ios\_base::app | ios\_base::bynary);

fil\_out.write((char \*) &mount, sizeof(mountine));

Метод write копирует указанное число байтов (в данном случае – sizeof(mountine)) в файл из памяти ЭВМ. Несмотря на то, что сохранение данных происходит в двоичном файле, адрес переменной преобразуется к указателю на тип char.

Для чтения данных из двоичного файла используют метод read:

ifstream fil\_in(”mountines.dat”, ios\_base::binary);

fil\_in.read((char \*) &mount, sizeof(mountine));

При записи или чтении классов, не содержащих виртуальных функций, можно использовать тот же самый подход. Чтобы сделать класс потоковым, нужно перегрузить операторы << и >>:

friend ostream &operator<<(ostream &, AnyClass &);

friend istream &operator>>(istream &, AnyClass &);

**Пример 4.** Чтение и запись структуры данных в файл.

//перечисляемый тип с оценками

enum mark {very\_bad = 1, bad = 2, satisfactory = 3, good = 4, excellent = 5};

struct rec //структура данных

{ char name[10];

char subname[15];

int clas;

mark marks[4];

};

//открытие "типизированного" файла для записи

void WriteFile(int n, rec &wrec)

{ FILE \* fo;

fo = fopen("type.txt","wb"); //открываем (создаем) файл для записи

if (fo == NULL)

{ cout << "Ошибка открытия файла для записи" << endl;

exit(EXIT\_FAILURE);

}

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

{ int temp=0; //временная переменная для оценок

cout << "Введите имя ученика: "; cin >> wrec.name;

cout << "Введите фамилию ученика: "; cin >> wrec.subname;

cout << "Введите класс, где обучается ученик: "; cin >> wrec.clas;

cout << "Введите оценки ученика (4 штуки от 1 до 5)" << endl;

for(int j = 0; j < 4; j++)

{ cout << "Введите " << j+1 << " оценку: "; cin >> temp;

wrec.marks[j] = (mark)temp;

}

WriteElement(wrec, fo); //запись одной структуры в файл

cout << endl;

}

fclose(fo); //закрываем файл

cout << endl;

}

//запись структуры в файл

void WriteElement(rec &wrec, FILE \* f1)

{

fwrite(&wrec,sizeof(rec),1,f1); //записываем одну структуру

}

//чтение структуры из файла

void ReadElement(rec & rrec, FILE \* f1)

{

fread(&rrec, sizeof(rec), 1, f1); //читаем одну структуру

}

**Произвольный доступ к элементам файлов**

*Файловый указатель.*

Каждый файл имеет два связанных с ним значения: указатель чтения и указатель записи, по-другому называемые файловым указателем или текущей позицией.

При последовательном доступе к элементам файлов перемещение файлового указателя происходит автоматически. Но иногда бывает нужно контролировать его состояние. Для этого используются следующие функции:

* seekg() – установить текущий указатель чтения;
* tellg() – проверить текущий указатель чтения;
* seekp() – установить текущий указатель записи;
* tellp() – проверить текущий указатель записи.

Организация доступа к элементам двоичных файлов.

Благодаря наличию файлового указателя, в двоичных файлах допустим произвольный доступ к их элементам, который можно реализовать с помощью перегруженных функций – элементов, унаследованных из класса istream:

istream &seekg(streampos) или

istream &seekg(streamoff, ios::seek\_dir);

Типы данных streampos и streamoff эквивалентны значениям типа long, но использовать long в явном виде не рекомендуется из-за неоднозначности работы различных компиляторов. Поэтому их определяют как

typedef long streampos;

typedef long streamoff;

Первая из перегруженных форм функции seekg позиционирует входной поток на заданном байте, вторая – на смещении относительно одной из трех позиций, определенных значением константы ios::seek\_dir (табл. 1)

### Таблица 1

|  |  |  |
| --- | --- | --- |
| **Константа** | **Значение** | **Описание** |
| beg | 0 | поиск от начала файла |
| cur | 1 | поиск от текущей позиции файла |
| end | 2 | поиск от конца файла |

Для позиционирования внутреннего указателя файла для выходных потоков используют перегруженные функции выходных файловых потоков, унаследованных из класса ostream:

ostream &seekp(streampos);

ostream &seekp(streamoff, ios::seek\_dir);

**Пример 5.** В двоичном файле, содержащем целые числа, заменить максимальное значение файла суммой его четных элементов.

#include <fstream>

#include <iostream>

using namespace std;

class bin\_stream :public fstream {

public:

bin\_stream(const char \*fn) : fstream(fn, ios::out | ios::in | ios::binary) {}

void doneOurDate(const void\*, int, int);

bin\_stream &operator<<(int d) {

doneOurDate(&d, sizeof(d), 0);

return \*this;

}

bin\_stream &operator>>(int &d) {

doneOurDate(&d, sizeof(d), 1);

return \*this;

}

};

int main()

{

int i, d, max, i\_max, sum\_even = 0;

bin\_stream bin\_out("Bin.dat");

if (!bin\_out) {

cerr << "Unable to write to Bin.dat" << endl;

system("pause");

exit(1);

}

for (i = 0; i < 10; i++) {

d = rand() % 100;

bin\_out << d;

if (d % 2 == 0) sum\_even += d;

}

bin\_out.seekp(0, ios::beg);

bin\_out >> max;

i\_max = 0;

for (i = 1; i < 10; i++) {

bin\_out >> d;

if (d > max) { max = d; i\_max = i; }

}

bin\_out.seekp(sizeof(int)\* i\_max, ios::beg);

bin\_out << sum\_even;

bin\_out.seekp(0, ios::beg);

for (i = 0; i < 10; i++) {

bin\_out >> d;

cout << d << ' ';

}

bin\_out.close();

cout << endl;

system("pause");

return 0;

}

void bin\_stream::doneOurDate(const void \*Ptr, int len, int sign)

{

if (!Ptr)

return;

if (len <= 0)

return;

if (sign == 0)

write((char\*)Ptr, len);

else

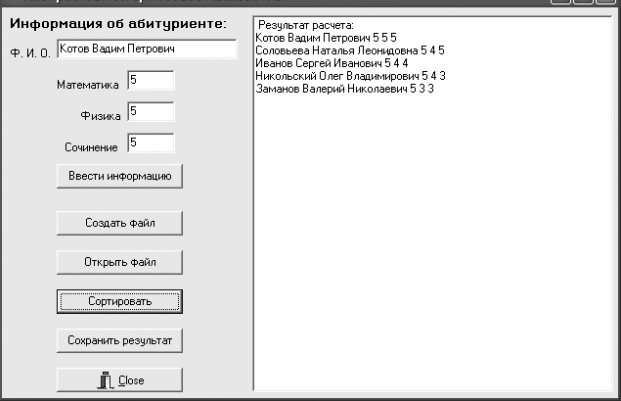
read((char\*)Ptr, len);

}

**Контрольные вопросы**

1. Что такое поток?
2. Особенности работы с двоичными файлами.
3. Что представляет собой файловый указатель?
4. Как организовать доступ к произвольному месту двоичного файла?

**Пример компоновки формы программы к лабораторной работе**



**Лабораторное задание**

Для выполнения лабораторной работы необходимо составить программу согласно своему варианту задания.

# Варианты заданий

| **Номер варианта** | **Задание** |
| --- | --- |
| 1 | В двоичном файле целого типа заменить максимальный элемент суммой предыдущих элементов, минимальный – суммой последующих элементов. |
| 2 | В конец двоичного файла целого типа дописать четные элементы этого файла |
| 3 | В начало двоичного файла целого типа дописать нечетные элементы этого файла. |
| 4 | В середину двоичного файла целого типа поместить элементы этого файла, кратные пяти. |
| 5 | В двоичном файле целого типа поменять местами элементы, стоящие на четных местах с элементами, стоящими на нечетных местах. |
| 6 | В начало двоичного файла целого типа дописать его минимальное значение, в середину – максимальное. |
| 7 | В начало двоичного файла целого типа записать элементы, являющиеся делителями максимального элемента этого файла. |
| 8 | В середину двоичного файла целого типа записать элементы этого файла, меньшие числа, введенного с клавиатуры. |
| 9 | Даны двоичные файлы *f* и *g* целого типа. Записать в начало файла *f* положительные компоненты файла *g*, а в конец файла g – отрицательные компоненты файла *f* с сохранением порядка их следования. |
| 10 | Дан двоичный файл с целыми числами. Удалить из него число, записанное после первого нуля (принять, что нули в файле имеются). Результат записать в другой файл. |
| 11 | Дан двоичный файл с целыми числами. Все его четные элементы заменить нулями. Рассмотреть 2 варианта:   * исходный файл содержит 13 чисел; * размер исходного файла неизвестен. |
| 12 | Дан двоичный файл с целыми числами. Заменить все его элементы, порядковый номер которых кратен 7, на новые значения, которые вводятся с клавиатуры. Рассмотреть 2 варианта:   * исходный файл содержит 20 чисел; * размер исходного файла неизвестен. |
| 13 | Дан двоичный файл с положительными и отрицательными целыми числами. Записать в другой файл сначала отрицательные элементы, а затем положительные. |
| 14 | В двоичном файле целого типа заменить каждый элемент суммой предыдущих элементов. В конце дописать общую сумму всех элементов. |
| 15 | В конец двоичного файла целого типа дописать все его элементы кратные 16. |