1 Файловые потоки в C++

В C++ аппарат работы с файлами основан на объектах классов, хранящихся в файлах ostream и istream, fstream.

Класс файловых потоков – это иерархия, базовым классов этой иерархии является абстрактный класс с именем ios, который находится на вершине иерархии. Он используется только в качестве базового класса для остальных потоков ввода-вывода.

Поток – это логический интерфейс к файлу.

Класс ios содержит единственный открытый конструктор потока, для использования которого требуется иметь предварительно созданный объект типа streambuf, обеспечивающий возможности буферизованного ввода или вывода. В большинстве случаев намного удобнее воспользоваться производными от ios классами, которые обеспечивают более простой интерфейс для создания потоковых объектов и предоставляют дополнительную функциональность, адаптированную к задачам ввода или вывода данных.

При работе с потоковой библиотекой ввода-вывода программист обычно достаточно активно использует следующие классы:

- istrstream - класс входных строковых потоков;
- ostrstream - класс выходных строковых потоков;
- strstream - класс двунаправленных строковых потоков (ввода-вывода);
- ifstream - класс входных файловых потоков;
- ofstream - класс выходных файловых потоков;
- fstream - класс двунаправленных файловых потоков (ввода-вывода);
- constream - класс консольных выходных потоков.

2 Управление текстовым файлом

Тестовый файл – это последовательность символов, сохраненная на внешнем носителе. Текстовый файл может быть создан текстовым редактором кодировки ASCII и средствами программы.

2.1 Создание потока

Создание потока конструктором без параметров. Такой поток представляет переменную, которая не связана с каким – то физическим файлом.

```
ifstream if; // if  поток для ввода данных в файл
```

ofstream of; // of для чтения данных из файла

fstream ff; // ff для чтения и записи

2.2 Операции над текстовым файлом

2.2.1 Связывание потока с физическим файлом

Создание, открытие и связывание потока с файлом при инициализации объекта:

ifstream if (“A.txt”);//поток if указывает на файл A.txt, который расположен в папке проекта. Поток if связан с файлом, открытым для чтения

ofstream of(“A.txt”);// Поток of связан с файлом, открытым для записи

fstream ff(“A.txt”); // Поток ff связан с файлом, открытым для чтения и записи

Метод открытия файла потока – open. Так как поток — это объект класса, то открытие можно выполнить методом open.

Формат метода:

open([имя файла**,]** [способ открытия файла])

имя файла – строковое значение или переменная типа строка, которое задает имя физического файла;

способ открытия файла – режим открытия файла, задается перечисляемой переменной enum *open_mode*(app, binary, in, out, trunc, ate)**,** которая определена в базовом классе ***ios.*** Так как классы ifstream, ofstream, fstream являются производными от класса ios, то при определении экземпляра одного из потоковых классов, обращение к значениям перечисяемой переменной должно идти с указанием класса родителя: ***ios::app, или ios::in, или ios:out и т.д.***

Назначения констант перечисления:

app – открыть существующий файл для записи в конец (указатель в конец);

out - открыть файл для создания и записи в начало (указатель на первую запись);

in - открыть файл для чтения из файла с первой записи (указатель на первую запись);

trunc – очистить файл, если он существует;

ate - переместить указатель на конец файла.

При открытии файла можно задавать несколько режимов открытия в одной строке, в этом случае режимы разделяются символом | .

f.open(“F.txt”, ios::out | ios::binary | ios::trunc) // открыть двоичный файл для записи и если он существует очистить содержимое файла

Пример 1. Открытие файла методом open для чтения данных из файла. Открыть текстовый фай A.txt для записи:

1) **при** создании файлового потока конструктором:

```
ofstream f.open("A.txt"); //по умолчанию для записи
ofstream f.open("A.txt", ios::in); //явно, с указанием соответствующего режима
```

2) **после** создания файлового потока

```
ofstream f;
f.open("A.txt");
```

Пример 2. Создание текстового файла программой. Открытие файла методом open для чтения данных из файла:

1) **при** создании файлового потока конструктором

```
ifstream f.open("A.txt"); //по умолчанию для записи
ifstream f.open("A.txt", ios::out | ios::trunc); //явно, с указанием соответствующего режима
```

2) **после** создания файлового потока

```
ifstream f;
f.open("A.txt", ios::trunc);
```

Пример 3. Модификация текстового файла программой. Требования: файл должен существовать. Открытие файла методом open для **добавления данных в конец файла**:

1) **при** создании файлового потока конструктором

```
ofstream f.open("A.txt", ios::app); //явно, с указанием соответствующего режима
```

2) **после** создания файлового потока

```
ofstream f;
f.open("A.txt", ios::app);
```

2.2.2 Закрытие файлового потока

При закрытии файлового потока прекращается доступ к данным. Закрытие файла осуществляет метод осуществляется методом ***close()***.

Формат вызова метода закрытия файла:

Имя потока.close()

Пример. Открытие файлового потока для чтения и закрытие.

```
int main()
```

```
{
 ofstream f;
 f.open(“A.txt”);
 // обработка файла
 f.close();
}
```

2.2.3 Проверка существования файла

Биты ошибок при обработке файла:

- badbit – устанавливается в случае катастрофической ошибки;
- failbit - устанавливается в случае восстанавливаемой ошибки;
- eofbit – устанавливается, если достигнут конец файла (это не обязательно ошибочная ситуация).

Функции получения значений состояний

- bool bad() const - возвращает true если установлен badbit;
- bool eof() const - возвращает true если установлен eofbit;
- bool fail() const - возвращает true если установлен failbit;
- bool good() const - возвращает true если не установлен ни один бит;
- iostate rdstate() const - возвращает битовую маску, ассоциированную с потоком.

Примеры использования битов потоков для проверки существования файлов и их состояния

Способ 1. (Чаще всего применяется). Применение операции ***!****имяпотока* к файловому потоку вернет результат вызова функции fail(), которая проверяет состояние бита failbit, в который записывается результат открытия файла.

```
int main()
{
 ofstream f(“A.txt”);
 if (!f)              //проверка открытия потока
 {
        cout << ”файл не открыт”;
        return1;
 }
 // обработка файла
 f.close();
}
```

Пример 1. Помещение записи в поток осуществляется операцией помещения в поток <<. Аналогично помещению в поток cout.

```
ofstream o(“A.txt”, ios::trunc);
int x;
o << x;
```

Пример 2. Создание текстового файла с именем A.txt и запись в текстовый (форматированный файл). Запись в файл двух целых чисел, числа располагаются в отдельных строках.

```
#include "stdafx.h"
#include <fstream>
#include <iostream>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
 ofstream f;
 f.open("A.txt");
 if (!f.is_open())//проверка существования файла с именем A.txt
 {
        cout << "файл не открыт";
        return 1;
 }
 //запись в файл двух строк
 f << "Текст записывается в файл A.txt" << endl;
 f << 123 << ' ' << 125 << '\n';
 // обработка файла
 f.close();
 return 0;
}
```

2.3 Методы для управления ошибками потоков

В базовом классе ios определено поле state, которое хранит состояние потока в виде совокупности битов. Каждая совокупность отвечает за определенное состояние. Для доступа к состоянию используются методы потока (табл. 13.1).

Таблица 13.1. Методы потока

Прототип метода	Описание действия
int rdstate()	возвращает текущее состояние потока
int eof()	результат 0, если не достигнут конец файла
int fail()	результат 0, если ошибка: при форматировании (флаг failbit=1), или от неисправности оборудования (hardfail), или серьезная ошибка в потоке(badbit=1)
int bad()	результат 0, если ошибка: при форматировании или от неисправности оборудования
int good()	проверка результатов операций ввода-вывода. результат 0, если флаги ошибок failbit, badbit, hardfail сброшены в 0
void cleare(int =0)	очистить флаг состояния
operator void*()	результат NULL, если установлен хотя бы один бит ошибки
operator !()	результат not NULL, если установлен хотя бы один бит ошибки

2.3.1 Функции и методы чтения данных из файла

Для чтения даных из файла используются следующие функции:

– Фунция – getline.

Формат метода:

```
int getline(поток чтения, переменная)
```

– Перегруженная операция извлечения из потока >>

Пример чтения из файла, созданного текстовым редактором

Пусть текстовый файл подготовлен текстовым редактором и хранит информацию на трех отдельных строках в файле (в конце строки стоит символ конца строки):

```
Герберт Шилдт.
C++ методика программирования Шилдта
2009
```

Задача:

– прочитать *Герберт Шилдт* в переменную *Fam*;
– прочитать C++ методика программирования Шилдта в переменную NameBook;
– прочитать *2009* в переменную числового типа *year*.

Код программы:

```
#include "stdafx.h"
#include "iostream"
```

```
#include "istream"
#include <string>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
 ifstream fin("B.txt");
 if (!fin)
 {
        cout << "No open file";
        return 1;
 }
 string Fam;
 string NameBook;
 int year;
 getline(fin, Fam);//чтение текста с пробелами
 getline(fin, NameBook);//чтение текста с пробелами
 fin >> year; //чтение числа
 return 0;
}
```

2.3.2 Использование функций проверки ошибок потоков

Проследим на примере возможности функций проверки ошибок потоков.

Пример использования методов управления ошибками: методы good(), eof(). Запись данных в не открытый файл.

```
int _tmain(int argc, _TCHAR* argv[])
{
 ifstream fin("C.txt");
 if (!fin)
 {
        cout << "No open file";
        return 1;
 }
 int N;
 string str;
 char ch;
 double x;
```

```
    while (!fin.eof())
    {
        fin >> N;
        fin >> str;
        fin >> ch;
        fin >> x;
        cout << N << ' ' << str << ' ' << ch << ' ' << x << endl;
    }
    ofstream of;
    of << "file no open" << endl;
    if (!of.good())
    {
        cout << "No open file";
        system("pause");
        return 1;
    }
    ifstream fin1("C.txt", ios::out | ios::trunc);//очистили файл
    if (fin1)
    {
        fin1 >> str;
    }
    if (!fin1.good())
    {
        cout << "file is FULL";
        system("pause");
        return 1;
    }
    fin1.close();
    return 0;
}
```

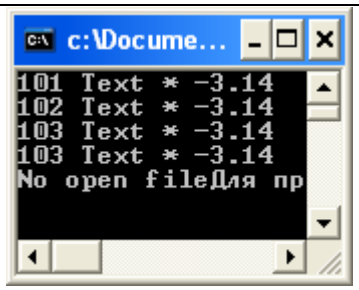

Рис. 13.2. Результат выполнения программы

2.3.3 Чтение данных из текстового файла

Текстовый файл хранит информацию в понятном человеку формате, в отличие от двоичных файлов. Поэтому их называют форматированными.

Чтение осуществляется:

- операцией выбора из потока >>;
- функцией getline текста с пробелами между словами;
- методами потока: get().

Методы для чтения из потока класса ifstream представлены в табл. 13.2.

Таблица 13.2. Методы для чтения из потока класса ifstream

Метод	Описание
gcount()	Возврат количества символов, прочитанных во время последней операции ввода
get(char &c)	Чтение символа в переменную с
get(char *buf, streamsize num) Чтение строковых данных из файла	Чтение символа в массив символов buf, пока не будет прочитано num-1 символов, или найден символ новой строки ли конец файла. Символ новой строки из потока **не извлекается.**
getline(char *buf, streamsize num) Чтение строковых данных из файла	Чтение символа в массив символов buf, пока не будет прочитано num-1 символов, или найден символ новой строки ли конец файла. Символ новой строки из потока извлекается, но в buf не записывается. Символ новой строки остается в буфере до следующего чтения.
ignore(buf, num, lim='\n')	Считывает и пропускает num символов или встретит символ, указанный параметром lim.
peek()	Возвращает следующий символ, без удаления его из потока, либо EOF

2.3.4 Особенности формирования текстового файла для чтения данных

1. Числовые данные должны быть разделены пробелом или символом \n.
2. Символы должны следовать последовательно друг за другом.

3. Если в одной строке подготовлены данные числовые, символьные, строковые (это смешанный ввод), то нужно помнить, о разделителях между данными. Потоковый ввод >> читает данные до пробела, поэтому для чисел это хорошо.

Пример 3. На чтение числовых данных из текстового данных. В текстовом файле хранится N чисел.

На первой строке хранится число - N количество чисел в файле. Со второй строки расположены N чисел по несколько чисел на строке, по сколько не знаем. Записать прочитанные данные в массив из N чисел.

Пример содержимого файла:

```
11
1 2
3 4 5 6
7 8
9
10
```

Код программы:

```
ifstream fin("C.txt");
if (!fin)
{       cout << "No open file";
 return 1;
}
int N;
fin >> N;
int *x = new int[N], i = 0;
while (!fin.eof())//до конца файла fin
{       fin >> x[i]; i++;
}
fin.close();
for (int i = 0; i<N; i++) cout << x[i] << ' ';
fin.clse();
```

Пример 4. Чтение данных разного типа подготовленных в одной строке и таких строк несколько.

Пусть в файле C.txt на нескольких строках подготовлены данные:

```
101 Текст * -3.14
102 Текст * -3.14
```

103 Текст * -3.14

Код программы:

```
ifstream fin("C.txt");
if (!fin)
{
 cout << "No open file";
 return 1;
}
int N;
string str;
char ch;
double x;
while (!fin.eof())//до конца файла fin
{
 fin >> N;
 fin >> str;
 fin >> ch;
 fin >> x;
 cout << N << ' ' << str << ' ' << ch << ' ' << x << endl;
}
fin.close();
}
```

2.3.5 Пример. Операции с текстовым файлом.

Рассмотрим работу с текстовым файлом на примере.

```
#include "stdafx.h"
#include "fstream"
#include "iostream"
using namespace std;
//создание текстового файла
//Структура файла: в строке одно число,
//строка завершается символом конца строки
void inpfiletxt(ofstream &fout, char *namefile)
{
 fout.open(namefile, ios::out | ios::trunc);
 for (int x = 1; x <= 10; x++)
```

```
 {
        fout << x << endl;
 }
 fout.close();
}
//вывод содержимого текстового файла
//чтение числа и символа конца строки
//чтобы обработать до конца файла
void outfiletxt(char *namefile)
{
 int x;
 ifstream fin;
 cout << endl;
 fin.open(namefile, ios::in);
 while (!fin.eof())
 {
        fin >> x;
        fin.get();
        cout << x;
 }
 fin.close();
}
//добавление записи в конец файла
void appfiletxt(ofstream &fout, char *namefile, int x)
{
 fout.open(namefile, ios::out | ios::app);
 fout << x;
 fout.close();
}
//прочитать запись по заданному номеру
void seekNtextfile(char *namefile, int n)
{
 int x;
 ifstream fin;
 fin.open(namefile, ios::in);
 int i;
```

```
    for (i = 1; (i<n && (!fin.eof())); i++)
    {
        fin >> x;
        fin.get();
    }
    cout << endl;
    while (!fin.eof() && (i == n))
    {
        fin >> x;
        fin.get();
        cout << x;
        i++;
    }
    fin.close();
}
int _tmain(int argc, _TCHAR* argv[])
{
    ofstream fout;
    inpfiletxt(fout, "A.txt");
    ifstream fin;
    outfiletxt("A.txt");
    appfiletxt(fout, "A.txt", 100);
    outfiletxt("A.txt");
    seekNtextfile("A.txt", 4);
    cin.get();
    return 0;
}
```

2.4 Управление двоичным (бинарным) файлом

Двоичные файлы используются для более компактного хранения информации. Хранят данные в машинном формате, т.е. в том виде как они представлены в оперативной памяти. Например, значение вещественной переменной типа float будет записано в файл в формате с плавающей точкой. Под это значение будет отведено столько байт, сколько требуется переменной формата float.

Логически файл состоит из записей фиксированной длины, минимальная длина один байт, максимальная соответствует размеру записанной при создании файла записи. Поэтому из такого файла удобно читать записи: *каким форматом записали, таким и будем читать.*

Создается файл программным путем. Как и текстовый файл имеет последовательную организацию, т.е. новые записи добавляются в конец файла. Обрабатывается файл так же последовательно от первой записи к последней. Для двоичных файлов с записями фиксированной длины применяется метод произвольного доступа, при котором можно обрабатывать отдельную запись: прочитать, изменить и записать на старое место.

Для применения в программе двоичного файла необходимо определить поток, поддерживающий двоичные операции.

2.4.1 Создание потока для двоичного файла

Рассмотрим операции:

1. Создание потока для создания двоичного файла.
   1.1.Без связывания с физическим файлом.

```
ofstream fb;
fb.open("data.dat",ios::out|ios::binary);
```

Выделенные параметры режима открытия файла обязательны для создания двоичного потока. Параметр ios::out – открытие потока для записи в файл.

Параметр ios::binary – открытие двоичного потока.

   1.2.Связывание открытие файла при объявлении.

```
ifstream fb("data.dat",ios::in|ios::binary);
```

2. Создание потока для чтения данных из двоичного файла
   2.1.Без связывания с физическим файлом.

```
ofstream fb;
fb.open("data.dat",ios::in|ios::binary);
```

   2.2.Связывание открытие файла при объявлении

```
ifstream fb("data.dat",ios::in|ios::binary);
```

3. Создание потока для добавления данных в двоичный файл
   3.1.Без связывания с физическим файлом.

```
ofstream fb;
fb.open("data.dat",ios::in|ios::binary|ios::app);
```

   3.2.Связывание открытие файла при объявлении

```
ifstream fb("data.dat",ios::in|ios::binary|ios::app);
```

2.4.2 Организация записи данных в двоичный файл

Запись данных в двоичный файл обеспечивает метод **write**

Формат метода:

ostream& write(const char *buf, streamsize num);

buf – указатель на блок памяти, значение которого будет записываться в файл;

num – количество байт в блоке памяти, на который указывает buf.

Хотя *buf* представлена в функции как *char **, можно записывать в файл данные любого типа. Просто надо указатель на данные привести к char * и указать нужную длину блока в байтах.

Формат двоичного файла представлен на рис. 13.3.

Запись 1	Запись 2	Запись 3	Запись 4			Запись N

Рис. 13.3. Формат двоичного файла

Записи это блоки размера num. Они ничем не отделяются друг от друга.

Пример 1. Создание файла с 10 записями типа double:

```
int main()
{
 fout.open("DD.dat", ios::binary | ios::trunc);
 if (!fout)
 {
        cout << "No open file";
        return 1;
 }
 double val;
 for (int i = 1; i <= 10; i++)
 {
        fout.write((char*)&val, sizeof(double));
 }
 fout.close();
 return 0;
}
```

Пример 2. Создание двоичного файла из записей – массивов.

```
int main()
{
 fout.open("DD.dat", ios::binary | ios::trunc);
```

```
    if (!fout)
    {
        cout << "file not open";
        return 1;
    }
    int x[3] = { 1,2,3 };
    //пишем массив из трех чисел
    fout.write((char*)x, 3 * sizeof(int));
    int y[3] = { 5,6,7 };
    fout.write((char*)y, 3 * sizeof(int));
    if (!fout.good())//контроль ошибок ввода-вывода
    {
        cout << "Error vvoda";
        return 1;
    }
    fout.close();
    ifstream fio("DD.dat", ios::in | ios::binary);//открыли для чтения
    while (!fio.eof())
    {   //читаем массив из трех чисел
        fio.read((char*)x, 3 * sizeof(int));
        for (int i = 0; i<3; i++) cout << x[i] << endl;
    }
    fio.close();
    int a;
    ifstream fii;
    fii.open("DD.dat", ios::in | ios::binary);
    while (!fii.eof())
    {   //читаем по одному числу
        fii.read((char*)&a, sizeof(int));
        cout << a << endl;
    }
    return 0;
}
```

Пример 3. Создание файла из записей (или свойств полей объекта).

В текстовом файле хранятся данные о книгах фонда библиотеки.

Структура записи о книге:

```
struct book
{
 char Fam[30];
 char Name[15];
 int year;
};
```

Сформировать двоичный файл из записей текстового файла, в котором построчно хранятся данные:

```
Шилдт
C++ Методики
2015
Страуструп
Язык C++
2015
Иванова Г.И.
ООП
2012
```

Код программы:

```
#include "stdafx.h"
#include "fstream"
#include "iostream"
#include "istream"
#include <string>
using namespace std;
struct book
{
 char Fam[30];
 char Name[15];
 int year;
};
//чтение из текстового файла записи типа book, формирование и запись в двоичный
//функция записи в файл значения типа book
void create_bin_file(ifstream &ft, ofstream &fb)
{
 book x;
```

```
 while (!ft.eof())
 {
        getline(ft, x.Fam);    //заполнение X
        getline(ft, x.Name);
        ft >> x.year;
        ft.get();
        //Запись X в дв.файл
        fb.write((char *)&x, sizeof(book));
 }
 ft.close();
 fb.close();
}
//вывод записей двоичного файла
void out_bin_file(ifstream &fb)
{
 book x;
 //чтение из файла всей записи
 fb.read((char *)&x, sizeof(book));
 while (!fb.eof())
 {
        cout << x.Fam;
        cout << x.Name;
        cout << x.year;
        fb.read((char *)&x, sizeof(book));
 }
 fb.close();
}
int main()
{
 ifstream ft;
 ofstream fb;
 char fnameText[30], fnameBin[30];
 cout << "Name for Text"; cin >> fnameText;
 cout << "Name for Text"; cin >> fnameBin;
 ft.open(fnameText, ios::out);
 fb.open(fnameBin, ios::out | ios::binary);
```

```
 if (!ft || !fb)
 {
        cout << "файл не открыт";
        return 1;
 }
 create_bin_file(ft, fb);
 ifstream fbb(fnameBin, ios::in | ios::binary);
 out_bin_file(fbb);
 return 0;
}
```

Пример 4. Добавление новой записи в конец двоичного файла.

```
#include "stdafx.h"
#include "fstream"
#include "iostream"
#include "istream"
#include <string>
using namespace std;
struct book
{
 string Fam;
 string Name;
 int year;
};
//вывод записей двоичного файла
void out_bin_file(ifstream &fb)
{
 book x;
 //чтение из файла всей запис сразу
 fb.read((char *)&x, sizeof(book));
 while (!fb.eof())
 {
        cout << x.Fam;
        cout << x.Name;
        cout << x.year;
        fb.read((char *)&x, sizeof(book));
 }
```

```
 fb.close();
}
void add_bin_file(char *fnameBin)
{
 book x;
 ofstream fadd(fnameBin, ios::in | ios::binary | ios::app);
 cin >> x.Fam;
 cin >> x.Name;
 cin >> x.year;
 fadd.write((char *)&x, sizeof(book));
 fadd.close();
}
int main()
{
 char fnameText[30], fnameBin[30];
 cout << "Name for Text"; cin >> fnameBin;
 fb.open(fnameBin, ios::in | ios::binary);
 if (!fb)
 {
        cout << "файл не открыт";
        return 1;
 }
 out_bin_file(fb);
 add_bin_file(fnameBin);
 out_bin_file(fb);
 return 0;
}
```

2.4.3 Чтение данных из двоичного файла

Из двоичного файла можно читать данные блоками по несколько байт. Размер блока должен быть к моменту чтения определен. Определение потока с операциями чтения данных двоичного файла - это объект типа ofstream. Чтение данных из двоичного файла выполняет метод класса istream.

Формат метода ***read()***

istream& read(char *buf, streamsize num);

buf – указатель на блок памяти (как массив), в котором будет сохранено прочитанное из файла значение в размере num;

num – количество байт – размер прочитанного блока. Значение пред загрузкой в buf может быть приведено к этому типу char *.

Пример 1. Чтение двоичного файла, содержащего записи о книге.

При создании файла с записями о книгах, записывали записи фиксированной длины – размер структуры. Чтение будем выполнять такими же блоками, в переменную типа book. Так как размер записи в файле равен размеру переменной типа book, то при чтении записи значения полей разместятся в соответствующих полях переменной.

Код программы:

```
#include "stdafx.h"
#include "fstream"
#include "iostream"
#include "istream"
#include <string>
using namespace std;
struct book
{
 string Fam;
 string Name;
 int year;
};
void create_bin_file(ifstream &ft, ofstream &fb)
{
 book x;
 while (!ft.eof())
 {
        getline(ft, x.Fam);
        getline(ft, x.Name);
        ft >> x.year;
        ft.get();
        fb.write((char *)&x, sizeof(book));
        fb.clear();
 }
 ft.close();
```

```
 fb.close();
}
void out_bin_file(ifstream &fb)
{
 book x;
 fb.read((char *)&x, sizeof(book));
 while (!fb.eof())
 {
        cout << x.Fam;
        cout << x.Name;
        cout << x.year;
        fb.read((char *)&x, sizeof(book));
 }
 fb.close();
}
void add_bin_file(char *fnameBin)
{
 book x;
 ofstream fadd(fnameBin, ios::in | ios::binary | ios::app);
 cin >> x.Fam;
 cin >> x.Name;
 cin >> x.year;
 fadd.write((char *)&x, sizeof(book));
 fadd.close();
}
int main()
{
 ifstream ft;
 ofstream fb;
 char fnameText[30], fnameBin[30];
 cout << "Name for Text"; cin >> fnameText;
 cout << "Name for Text"; cin >> fnameBin;
 ft.open(fnameText, ios::out);
 fb.open(fnameBin, ios::out | ios::binary);
 ifstream fbb(fnameBin, ios::in | ios::binary);
 if (!ft || !fb)
```

```
        {
                cout << "файл не открыт";
                return 1;
        }
        int num;
        while (1)
        {
                cout << " Operation for files" << endl;
                cout << " 1. Create for TextFile inBinFile"
                        << endl;
                cout << " 2. OUT for BinFile" << endl;
                cout << " 3. Add record in BinFile" << endl;
                cout << " 4. EXIT" << endl;
                cout << "numPunkt="; cin >> num;
                switch (num)
                {
                        case 1: create_bin_file(ft, fb); break;
                        case 2: {ifstream fbb(fnameBin, ios::in | ios::binary);
out_bin_file(fbb);    break; }
                        case 3: add_bin_file(fnameBin); break;
                        case 4: exit(0);
                }
                If(!fbb.good())
                {
                        cout << ”Ошибка ввода” << endl;
                        return 1;
                }
        }
        return 0;
      }
```

2.4.4 Закрытие файла

Закрытие двоичного файла осуществляется аналогично текстовым айлам:

1. При выполнени деструктора.
2. При вызве метода close.

2.4.5 Метод контроля ошибок потока

Для контроля ошибок работы с двоичным файлом используется метод:

```
bool good()
```

Если флаги ошибок не установлены, то результат true.

2.5 Поток класса fstream

2.5.1 Использование одного потока для чтения и записи данных в файл

Поток fstream позволяет открывать файлы для операций чтения и записи. Это требуется если необходимо модифицировать записи файла. Особенно этот поток удобен для обработки файлов с записями фиксированной длины, так как позволяет изменить запись, не переписывая весь файл.

Например, чтобы изменить значение поля year в файле BOOKS.txt, необходимо создать другой файл, переписать в него все записи до нужной, прочитать, изменить нужную и записать ее в новый файл, а далее переписать все остальные.

Этот поток предоставляет возможность заменить данные в файле значений при последовательном проходе по файлу. При чтении и записи могут использоваться методы: put(), get(), read(), write().

Для некоторых компиляторов вывод в файл можно выполнить, используя метод flush(), который перепишет данные, размещенные в буфере файла в файл, замещая находящиеся там значения. Для некоторых компиляторов нужно использовать методы seekg() или seekp().

Пример. Запись и чтение данных из файла, открыв его один раз для операций ввода и вывода.

Создадим текстовым редактором файл с текстом:

```
Сессия 2017 года, заменим на Сессия 2018 года.
```

Код программы:

```
#include "stdafx.h"
#include "fstream"
#include "iostream"
//#include "istream"
#include <string>
using namespace std;
int main()
{
 char ch;
```

```
fstream ff("A.txt");
if (!ff)
{
	cout << "No open";
	system("pause");
	return 1;
}
//пропустить слово сессия
ff.get(ch);
while (ch != ' ')
{
	cout << ch;
	ff.get(ch);
}
//заменить 2017 на 2018
ff.put('2'); ff.put('0'); ff.put('1'); ff.put('8');//запись в буфер
if (!ff.good())
{
	cout << "ERROR input" << endl;
	system("pause");
	return 1;
}
//вытолкнуть из буфера в файл
ff.flush();
//Прочтать остальные смволы файла,должно выть year
ff.get(ch);
while (!ff.eof())
{
	cout << ch;
	ff.get(ch);
}
cout << endl; ff.close();
if (!ff.good())
{
	cout << "ERROR output" << endl;
	system("pause");
```

```
            return 2;
        }
        return 0;
    }
```

2.5.2 Организация прямого доступа к записям двоичного файла

Двоичный файл с записями фиксированной длины можно рассматривать как массив. Для таких файлов реализован аппарат прямого доступа к элементам, подобно индексу в массивах.

С такими двоичными файлами связано понятие ***указателя***. При выполнении операции ввода или вывода указатель перемещается вперед. Указатель ввода и вывода. При чтении или записи одной записи указатель увеличивается на размер одной записи.

Используя функции произвольного доступа

istream &seekg(offset, origin) ;

ostream &seekp(offset, origin)

можно смещаться по файлу на определенное количество записей, от позиции в файле: от начала файла, от текущей позиции, от конца файла.

Значения параметров:

- ***offset*** – количество байтов на которое надо сместиться от позиции origin, вперед или назад по записям файла;
- ***origin*** – позиция, от которой начинается смещение, определяется значениями:
    - ios::beg – от начала файла;
    - ios::cur – от текущей позиции;
    - ios::end – от конца файла.

Функция ***seekg()*** перемещает ассоциированный с файлом указатель чтения на offset записей или символов. Функция ***seekg()*** объявлена в istream . И istream и ostream наследуются классом fstream, который позволяет выполнять операции ввода и вывода.

Функция ***seekp()*** перемещает ассоциированный с файлом указатель записи на offset записей или символов. Функция ***seekp()*** определена в ostream и наследована ostream.

Пример. Изменение записи с заданным номером в двустороннем потоке

```
void ChengeRecord(int pos, char *namefile)
```

```
{
 fstream fdirect(namefile, ios::binary | ios::out | ios::in);
 record r;
 fdirect.seekg(pos * sizeof(r), ios::beg);
 fdirect.read((char *)&r, sizeof r);
 fdirect.seekg(-sizeof(r), ios::cur);
 strcpy(r.info, "****");
 fdirect.write((char *)&r, sizeof r);
}
```

2.5.3 Пример применение прямого доступа для перемещения по файлу

Пример. Создать поток для прямого доступа к двоичному файлу, содержащему три записи с данными:

```
Шилдт Методики 2015
Страуструп Язык С++ 2015
Иванова Г.И. ООП 2012
```

Выполнить операции:

1) Открыть поток для чтения и записи.
2) Изменить фамилию Страуструп на Straustrup. Над найти запись с фамилией Страуструп. Она будет в процессе поиска прочитана в переменную X, после ее чтения указатель за этой записью.
3) Изменить прочитанную запись в переменной X.
4) Записать измененную запись на ее место в файле, для этого:

– переместить указатель на одну запись назад;
– выполнить запись в файл значения переменной X.

5) Показать содержание файла. Для этого надо установить указатель перед первой записью.

Код программы:

```
#include "stdafx.h"
#include <fstream>
#include <iostream>
#include <istream>
#include <string>
using namespace std;
struct book
{
```

```
    string Fam;
    string Name;
    int year;
};
void out_bin_file(fstream &fb)
{
    book x;
    fb.read((char *)&x, sizeof(book));
    while (!fb.eof())
    {
        cout << x.Fam << endl;
        cout << x.Name << endl;
        cout << x.year << endl;
        fb.read((char *)&x, sizeof(book));
    }
    fb.close();
}
//поиск записи в файле и изменения
void cheng_bin_file(fstream &fbpr, string newfnam, string oldfnam)
{
    book x;
    int i = 0;
    do
    {
        fbpr.read((char *)&x, sizeof(book));
        if (x.Fam == oldfnam)
        {
            //заменитть фамилию в записи на новую
            x.Fam = newfnam;
            //смещение назад на одну запись (на старое место)
            fbpr.seekg(sizeof(book)*(i - 1), ios::cur);
            //запись X в файл
            fbpr.write((char*)&x, sizeof(book));
            return;
        }
```

```
    } while (!fbpr.eof());
}
int _tmain(int argc, _TCHAR* argv[])
{
 setlocale(LC_ALL, "rus");
 char fnameBin[30] = "B.dat";
 book x;
 fstream fbpr;
 fbpr.open(fnameBin, ios::out | ios::in | ios::binary);
 if!fbpr)
 {
        cout << "файл не открыт";
        return 1;
 }
 string oldfnam = "Страуструп", newfnam = "Straustrup";
 cheng_bin_file(fbpr, newfnam, oldfnam);
 //установить указатель перед первой записью
 //чтобы вывести записи файла и не открывать его снова
 fbpr.seekg(sizeof(book) * 0, ios::beg);
 out_bin_file(fbpr);
 fbpr.close();
 return 0;
}
```

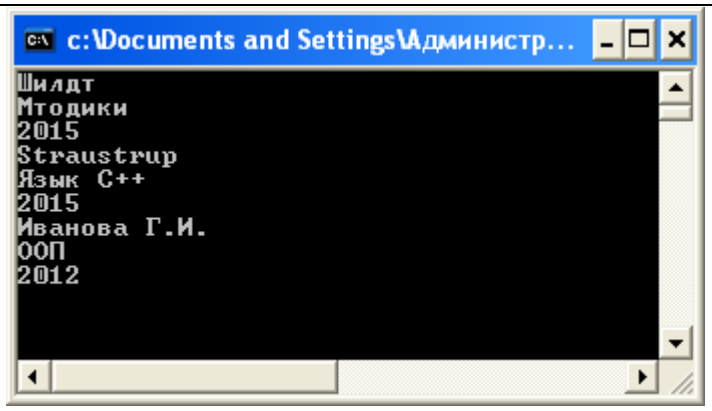

Рис. 13.4. Результат выполнения программы

Создание хеш-таблицы с открытым адресом

```
// ConsoleApplication9.cpp : Этот файл содержит функцию "main". Здесь начинается и
заканчивается выполнение программы.
//

#include <iostream>
using namespace std;
```

```
struct typeitem {
    int key=0;
    int offset=0;
    bool openORclose=true;     //свободна
    bool deletedORnot=false;    //не удалялась

};
struct HeshTable {
    int L = 19;
    typeitem *T;// таблица
    int insertedcount;//количество вставленных ключей
    int deletedcount; //количество удаленных ключей
    void createHeshTable() {
        T = new typeitem[L];
        insertedcount = 0;
        deletedcount=0;
    }
};
//хеш-функция
int hesh(int key,int L) {
    return key % L;
 }
//вставка без рехеширования
int insertInHeshTable(int key,int offset, HeshTable& t) {
    int i = hesh(key, t.L);//если key=9 i=9%19=9
    //разрешение коллизии
    while (i<t.L && t.T[i].openORclose == false)
            i++;
    if (i < t.L)
    {
        t.T[i].key = key; t.T[i].offset = offset; t.T[i].openORclose = false;
        t.insertedcount++;
        return 0;
    }
    else
        return 1;
}
void outTable(HeshTable& t) {
    for (int i = 0; i < t.L; i++)
        cout << i << ' ' << t.T[i].key << "    " << t.T[i].offset << " " <<
        t.T[i].openORclose << "   " << t.T[i].deletedORnot << '\n';

}
int search(HeshTable& t, int key);

//9  123  false false
//10  9   true true
//11  28  false false

int search(HeshTable& t, int key) {
    int i = hesh(key, t.L);
    //ищем по кластеру
    while (i < t.L && ((t.T[i].openORclose == false && t.T[i].deletedORnot==false)
        || (t.T[i].openORclose == true && t.T[i].deletedORnot == true))
        && t.T[i].key != key)
        i++;
    if (t.T[i].openORclose == true && t.T[i].deletedORnot == false) {
        return -1;
    }
    return i;
}
//удаление
int deletedFromHeshTable(HeshTable& t, int key) {
    int i = search(t, key);
```

```
    if (i == -1) return 1;//нет такой записи в таблице
    t.T[i].deletedORnot = true;
    t.T[i].openORclose = true;
    t.deletedcount++;
    return 0;
}
//Формирование хеш таблицы(key,offset<...) по ключам файла
//Поиск записи с ключем в файле:
//1) поиск ключа в хеш таблице и получение offset
//2) fb.seekg(offset, ios::beg)
//fb.read(rfile, sizeof(rfile));
//3) обработать запись rfile

int main()
{
    HeshTable T;
    T.createHeshTable();
    insertInHeshTable(123, 0, T);  // 9
    insertInHeshTable(12, 3, T);   //12
    insertInHeshTable(19, 1, T);   //0
    insertInHeshTable(9, 4, T);    //(9) 10 коллизия
    insertInHeshTable(28, 2, T);   //(9) 11 коллизия
    outTable(T);
    typeitem r;
    int i= search(T, 28);
    if (i!=-1){
        r = T.T[i];
        cout << r.key << ' ' << r.offset << endl;
    }
    else
        cout << "record is not" << '\n';
    i= deletedFromHeshTable(T, 9);
    if (i == 0)cout << "record is deleted";
    else
        cout << "record is not" << '\n';
    outTable(T);
    i = search(T, 28);
    if (i != -1) {
        r = T.T[i];
        cout << r.key << ' ' << r.offset << endl;
    }
    else
        cout << "record is not" << '\n';



    std::cout << "Hello World!\n";
}
```