БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ РАДИОФИЗИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ
КАФЕДРА ИНФОРМАТИКИ И КОМПЬЮТЕРНЫХ СИСТЕМ

Н.В. Серикова

ПРАКТИЧЕСКОЕ РУКОВОДСТВО

к лабораторному практикуму

«ФАЙЛЫ»

ПО КУРСУ

«ОСНОВЫ ООП»

2021
МИНСК

Практическое руководство к лабораторному практикуму «ФАЙЛЫ» по курсу «ООП» предназначено для студентов, изучающих базовый курс программирования на языке C++, специальностей «Компьютерная безопасность», «Прикладная информатика».

Руководство содержит некоторый справочный материал, примеры решения типовых задач с комментариями.

Автор будет признателен всем, кто поделится своими соображениями по совершенствованию данного пособия.

Возможные предложения и замечания можно присылать по адресу:

E-mail: Serikova@bsu.by,

ОГЛАВЛЕНИЕ

Файлы **4**
Потоки **5**
Класс istream **6**
Класс ostream **7**
Организация работы с файлами в C++ **8**
Текстовые файлы **11**
Бинарные файлы **12**
ПРИМЕР 1. Связывание логического и физического файлов 13
ПРИМЕР 2. Открытие файла для чтения 14
ПРИМЕР 3. Открытие файла для записи 15
ПРИМЕР 4. Открытие файла для добавления информации 16
ПРИМЕР 5. Чтение информации из файла прямого доступа 17
ПРИМЕР 6. Чтение информации из текстового файла. Оператор >> 18
ПРИМЕР 7. Чтение информации из текстового файла. Метод getline 19
ПРИМЕР 8. Запись информации в файл прямого доступа 20
ПРИМЕР 9. Запись информации в текстовый файл 21
ПРИМЕР 10. Определение текущей позиции указателя 22
ПРИМЕР 11. Смещение указателя в указанную позицию. Запись в позицию 23
ПРИМЕР 12. Смещение указателя в указанную позицию. Чтение из позиции 24
ПРИМЕР 13. Определение числа компонент в файле 25
ПРИМЕР 14. Удаление файла 26
ПРИМЕР 15. Уничтожение информации от текущей позиции указателя до конца 27
ПРИМЕР 16. Работа с бинарным файлом 28
ПРИМЕР 17. Работа с текстовым файлом 31
Словарь понятий, используемых в заданиях **34**

ФАЙЛЫ

Хранение данных в переменных и массивах является временным (до конца выполнения программы). Для постоянного хранения предназначены **файлы**. Посредством файлов обрабатываемые программой данные могут быть получены извне, а результаты сохранены для последующего использования.

Файлом называют именованную структуру данных, представляющую собой последовательность элементов одного типа, хранящихся на внешнем носителе информации.

Физический файл – файл с точки зрения операционной системы.

Логический файл – файл с точки зрения языка программирования.

У любого файла есть **имя**, что дает возможность работать одновременно с несколькими файлами.
В файле могут содержаться данные любых типов, за исключением файлов.

Под **доступом к файлу** понимают запись и чтение данных из файла.

По способу доступа файлы делятся на файлы **последовательного доступа** и файлы **произвольного (прямого) доступа.**

Файл может быть открыт

- **для чтения (входной файл),**
- **для записи (выходной файл),**
- **для чтения и записи (двунаправленный обмен).**

Операция **вывода данных в файл** означает пересылку их из рабочей области памяти в файл, а операция **ввода** – заполнение ячеек памяти данными, полученными из файла.

Различают **текстовые** и **двоичные (бинарные)** файлы.

При извлечении информации из файла программа должна уметь определять, когда данные в файле закончились. В любом языке программирование существует функция определения конца файла. Функция определяет конец файла либо через размер файла, читая его из служебной структуры файловой системы, либо через маркер конца файла EOF (End Of File, ASCII-код равен 26, комбинация клавиш <Ctrl><Z>), записываемый в конце файлов (конкретный способ зависит от типа используемой файловой системы).

Текущая позиция – место в файле, с которого будет выполняться следующая операция доступа к файлу.

ПОТОКИ

Поток (**stream**) – абстрактное понятие, относящееся к любому переносу данных от источника к приемнику.

Чтение данных из потока называется **извлечением**, а вывод в поток – **помещением**.

Обмен с потоком производится через специальную область – **буфер**.

По направлению потоки делятся на **входные** (данные вводятся в память), **выходные** (данные выводятся из памяти) и **двунаправленные**.

По виду устройств, с которым работает поток, потоки делятся на **стандартные, файловые и строковые.** Характер поведения всех потоков одинаков, несмотря на различные физические устройства, с которыми они связываются.

Существует два вида потоков: **текстовый и двоичный**.

Текстовый используется для ввода-вывода символов, при этом могут происходить некоторые преобразования символов.

Двоичный поток можно использовать с данными любого типа, причем преобразования символов не выполняется: между тем, что посылается в поток , и тем, что реально содержится в файле существует взаимно-однозначное соответствие.

В C++ поток представляет собой объект некоторого класса.

Для поддержки потоков библиотека C++ имеет два базовых класса :

- **ios –** для ввода-вывода;
- **streambuf –** для обеспечения буферизации потоков и их взаимодействия с физическими устройствами.

Классы **istream** и **ostream** являются наследниками **ios** и предназначены **для ввода и вывода** соответственно. Класс **iostream** – наследник одновременно классов **istream** и **ostream** (пример множественного наследования).

КЛАСС ISTREAM

Класс **istream** выполняет действия по вводу данных – извлечение, содержит следующие функции:

>> - форматированное извлечение данных всех основных (и перегружаемых) типов из потока;

для неформатированного чтения из потока:

get() – возвращает код извлеченного из потока символа или EOF;

get(ch) – извлекает один символ в ch и возвращает ссылку на поток;

get(str) – извлекает символы в символьный массив str до ограничителя '\n';

get(str, MAX) – извлекает до MAX числа символов в символьный массив str;

get(str, DELIM) – извлекает символы в символьный массив str до указанного ограничителя (обычно '\n'); оставляет ограничитель в потоке;

get(str, MAX, DELIM) – извлекает в символьный массив str до MAX символов или до символа DELIM; оставляет ограничитель в потоке;

getline(str, MAX, DELIM) – извлекает в символьный массив str до MAX символов или до символа DELIM; извлекает ограничитель из потока;

ignore(MAX, DELIM) – извлекает и удаляет до MAX числа символов до ограничителя включительно (обычно '\n'); с извлеченными данными ничего не делает;

peek () – возвращает следующий символ, оставляя его в потоке, или EOF, если достигнут конец файла;

peek (ch) – читает следующий символ, оставляя его в потоке;

putback (ch) – вставляет во входной поток символ, который становится текущим при извлечении из потока;

gcount() – возвращает число символов, считанных с помощью последнего вызова функ ций неформатированного ввода get(), getline(), read();

(наиболее часто применяемые для неформатированного чтения из файлов:)

read (str, MAX) – извлекает в символьный массив str MAX символов (или все символы до конца файла, если их меньше MAX);

seekg(pos) – устанавливает расстояние (в байтах) от начала файла до файлового указателя (т.е. устанавливает текущую позицию чтения в значение pos);

seekg (pos, seek_dir) – перемещает текущую позицию чтения на pos байтов, считая от одной из трех позиций, определяемых параметром seek_dir: ios::beg (от начала файла), ios::cur (от текущей позиции), ios::end (от конца файла);

tellg() – возвращает позицию (в байтах) указателя файла от начала файла.

tellg(pos) – возвращает позицию (в байтах) указателя файла от начала файла.

КЛАСС OSTREAM

Класс **ostream** предназначен для вывода (вставки в поток) данных, содержит функции:

>> - форматированная вставка в поток данных всех основных (и перегружаемых) типов из потока;

для неформатированного вывода в поток:

put(ch) – выводит в поток один символ ch и возвращает ссылку на поток;

flush() – записывает содержимое потока вывода на физическое устройство (очистка буфера);

(наиболее часто применяемые для неформатированного вывода в файл:)

write(str, SIZE) – записывает SIZE символов из массива str в файл;

seekp(pos) – устанавливает текущую позицию записи в значение pos относительно начала файла;

seekp(pos, seek_dir) – перемещает текущую позицию файлового указателя на pos байтов, считая от одной из трех позиций, определяемых параметром seek_dir: ios::beg (от начала файла), ios::cur (от текущей позиции), ios::end (от конца файла);

tellp() – возвращает позицию указателя файла (в байтах).

ОРГАНИЗАЦИЯ РАБОТЫ С ФАЙЛАМИ В С++

В С++ файл рассматривается как **последовательный поток байтов**.

От классов istream, ostream и iostream наследуются три класса **для работы с файлами**:

- **ifstream** – входной поток;
- **ofstream** – выходной поток;
- **fstream** – поток ввода-вывода.

Каждому файлу операционной системы в языке программирования ставится в соответствие **файловая переменная** определенного типа.

Чтобы иметь возможность работать с файлом, необходимо связать переменную-поток с этим файлом (процедура заключается в связывании объявленного потока с именем существующего или вновь создаваемого файла, а также в указании способа обмена информацией: чтение из файла, запись в него, чтение и запись). Обычно эта связь задается либо при открытии файла, либо при создании потока.

При закрытии файла связь разрывается (т.е. разрывается связь только переменной-потока с внешним файлом, сама переменная продолжает «жить» в соответствии с объявлением и может быть снова связана с тем же или с другим файлом). При закрытии файла буфер освобождается – выводится в файл на внешнее устройство. Если программа заканчивается аварийно, файлы остаются незакрытыми, и последняя порция информации на диск не попадает.

После создания потока, одним из способов связать его с файлом является компонентная функция **open()**.

Прототипы функции для каждого класса выглядят так:

void ifstream :: open (const char* fileName, open_mode mode = ios::in);
void ofstream:: open (const char* fileName, open_mode mode = ios::out | ios::trunc);
void fstream :: open (const char* fileName, open_mode mode = ios::in | ios::out);

здесь fileName – имя файла, в которое может входить в спецификатор пути;
mode – режим – целая статическая константа размером в 1 бит; режимы можно объединять битовой операцией ‘ | ‘.

Режимы обращения к файлам:

ios::in	открыть файл для ввода
ios::out	открыть файл для вывода
ios::app	записать данные в конец файла
ios::ate	переместиться в конец исходного открытого файла, данные могут быть записаны в любое место файла
ios::trunc	удалить содержимое файла, если он существует (по умолчанию это делается и для ios::out)
ios::nocreate	если файл не существует, то операция его открытия не выполняется
ios::noreplace	если файл существует, то операция его открытия не выполняется
ios::binary	открыть файл в двоичном режиме (по умолчанию файлы открываются в текстовом режиме)

Режимы работы с файлами:

- открыть файл для чтения;
- открыть файл для записи:
  - создание нового файла,
  - модификация существующего файла,
  - добавления информации в конец файла.

Чтение из файла

Операция **чтения из файла** означает заполнение ячеек памяти данными, полученными из файла.

Для чтения информации из файла необходимо:

- объявить файловую переменную,
- связать файловую переменную с физическим файлом или устройством,
- открыть файл для чтения,
- переместиться к той позиции в файле, с которой должно начаться считывание (определяется способом доступа к файлу: произвольный или последовательный),
- определить данные какого размера и в какую переменную будут считаны,
- вызвать функцию чтения данных из файла.

Запись в файл

Операция **записи данных в файл** означает пересылку их из рабочей области памяти в файл.

Для записи информации в файл необходимо:

- объявить файловую переменную,
- связать файловую переменную с физическим файлом или устройством,
- открыть файл для записи с указанием того будет ли файл перезаписан или изменен.
- переместиться к месту начала записи (определяется способом доступа к файлу: произвольный или последовательный),
- вызвать функцию записи данных в файл.

Добавление данных в файл

При добавлении информации данные будут записаны в конец файла.

Для добавления информации в файл следует:

- объявить файловую переменную,
- связать файловую переменную с физическим файлом или устройством,
- открыть файл для добавления (при этом указатель автоматически переместится в конец файла),
- вызвать функцию записи информации в файл.

ТЕКСТОВЫЕ ФАЙЛЫ

Под доступом к файлу понимают запись и чтение данных из файла.

Текстовый файл является **файлом последовательного доступа**, т.е. любой элемент файла может быть прочитан или записан, только если перед ним был прочитан или записан элемент файла, непосредственно ему предшествующий.

Текстовые файлы предназначены для хранения текстовой информации.

Текстовый файл рассматривается как последовательность символов, разбитая на строки. Каждая строка завершается символом новой строки. В программе на C++ этот символ обозначается как ‘\n’.

Строки текстового файла могут иметь разную длину. Доступ к каждой строке возможен лишь последовательно, начиная с первой. Именно в файлах такого типа хранятся исходные тексты программ.

Целые и вещественные значения в текстовом файле имеют внешнее (текстовое) представление и отделяются друг от друга пробельным символом (чтобы узнать, где кончается одно число и начинается другое, при выводе в файл их надо их разделять пробельными символами). При чтении чисел из файла, они автоматически преобразуются из текстового представления во внутреннее, машинное. При выводе чисел в текстовый файл, они преобразуются из внутреннего представления в символьный (текстовый) вид.

В каждый момент времени позиции в файле, откуда выполняется чтение и куда производится запись, определяются значениями указателей позиций записи и чтения файла. Позиционирование указателей записи и чтения (т.е. установка на нужные байты) выполняется либо автоматически, либо за счет явного управления их положением.

Текстовые файлы могут создаваться с помощью тестового редактора и содержать данные разных типов.

Длина создаваемого файла никак не оговаривается при его объявлении и ограничивается только ёмкостью устройств внешней памяти.

БИНАРНЫЕ ФАЙЛЫ

Под доступом к файлу понимают запись и чтение данных из файла.

Произвольный (прямой) доступ к файлу означает, что чтение и/или запись данных может производиться в любую указанную позицию.

В файлах произвольного доступа должен быть механизм вычисления местонахождения любого элемента файла (например, если все элементы имеют одинаковую фиксированную длину). Элементы могут вставляться в файл без разрушения других элементов файла. Элементы, которые в нем хранятся, могут быть изменены или удалены.

В C++ **двоичные (или бинарные) файлы** являются файлами прямого доступа.

Произвольный доступ выполняется только для файлов, открытых в двоичном режиме, тогда файлы **- бинарные или двоичные.**

При работе с файлами необходимо знать:

- элементы двоичных файлов хранятся во внутреннем представлении и поэтому должны создаваться программно;
- двоичные файлы также могут содержать элементы разных типов, для большей надежности их следует формировать из элементов только определенного типа;
- для последовательной обработки однотипных данных двоичный файл предпочтительнее текстового как в отношении надежности, так и в отношении экономичности (скорости выполнения операций);
- благодаря возможности прямого доступа к своим элементам двоичные файлы представляются привлекательными при чередовании операций чтения и записи данных.

ПРИМЕР 1. Связывание логического и физического файлов

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

void main()
{
    // в каждом из трех  потоковых классов связывание
    // логического и физического файлов можно выполнить
    // либо при создании потока,либо при открытии файла

                          // при создании потоков
    // создать объект ofstream
    // открыть физический файл file.bin с програмным
    // именем  outfile  в режиме записи
    ofstream outfile("file.bin");

    //  создать объект ifstream
     // открыть физический файл file.txt с программным
     // именем infile  в режиме считывания
    ifstream infile("file.txt");

    // создать объект fstream
    //  открыть физический файл name.txt с программным
    //  именем filename  и  для записи и для чтения
    fstream filename ("name.txt",ios::in | ios::out);


                             //  при открытии файлов
    ofstream out_file;    // создать объект ofstream
    //  открыть физический файл file.bin с програмным
    // именем  utfile  в режиме записи
    out_file.open("file.bin");

    ifstream in_file;       //  создать объект ifstream
      // открыть физический файл file.txt с программным
      // именем unfile  в режиме считывания
    in_file.open("file.txt");

    fstream file_name;       // создать объект fstream
    //  открыть физический файл name.txt с программным
    //   именем filename   и  для записи и для чтения
    file_name.open("name.txt",ios::in | ios::out);
}
```

ПРИМЕР 2. Открытие файла для чтения

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

int main()
{
    // Режимы открытия файлов
    // ios::in              открыть файла для ввода
    // ios::nocreate        если файл не существует, то
    //                операция его открытия не выполняется
    // ios::binary       открыть файл в двоичном режиме

                                                  // неявно
    //  открыть файл file.txt с программным именем infile
    //   в режиме считывания  ios::in - по умолчанию

    ifstream infile("file.txt");
                 // проверка на успешность выполнения
    if (!infile)
             { cout<<"File error"<<endl;
               return 1;
             }

                                                  // явно
    ifstream infile1;          // создать объект ifstream

    //  открыть файл file1.txt с программным именем infile1
    infile1.open("file1.txt");

    //   функцией is_open  можно проверить открыт ли файл
    if (infile1.is_open())
        cout<<"file open"<<endl;
    return 0;
}
```

ПРИМЕР 3. Открытие файла для записи

```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{    // Режимы открытия файлов
     // ios::out    открыть файл для вывода
     // ios::app    записать данные в конец файла
     // ios::ate    переместиться в конец исходного
     //             открытого файла данные могут быть записаны
     //             в любое место файла
     // ios::trunc  удалить содержимое файла, если он
     // существует по умолчанию это делается и для ios::out)
     // ios::_Nocreate          если файл не существует,
     //  то операция его открытия  не выполняется
     // ios::noreplace   если файл существует, то операция
     //                       его открытия   не выполняется
     // ios::binary          открыть файл в двоичном режиме

                                 //  1  неявно
     //  открыть файл file.bin с программным именем outfile
     //  в режиме записи    ios::out - по умолчанию
     ofstream outfile("file.bin", ios::binary);
                  // проверка на успешность выполнения
     if (!outfile)
               {  cout<<"File error"<<endl;
                   return 1;
               }
                                 //  2 или так  неявно
     //  открыть файл file1.txt с программным именем outfile1
     //  при повторном обращении новая запись прибавляется
     //   к существующей
     ofstream outfile1("d:\\file1.txt", ios::app );
                  // проверка на успешность выполнения
     if (!outfile1)
             {  cout<<"File error"<<endl;
                return 2;
             }
                                 // 3   явно
     ofstream outfile2;      // создать объект ofstream
     //  открыть файл file1.txt с программным именем
     //  outfile1 в режиме записи   ios::out - по умолчанию
     Outfile2.open("file1.txt");
     return 0;
}
```

ПРИМЕР 4. Открытие файла для добавления информации

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

int main()
{
                                        //  1  неявно
    //  открыть файл name.txt с программным именем filename
    //  для записи и для чтения
    fstream filename ("name.txt",ios::in | ios::out);
             // проверка на успешность выполнения
    if (!filename)
            { cout<<"File error"<<endl;
               return 1;
               }

                                       // 2   явно
    fstream filename1;                // создать объект fstream
    //  открыть файл name.txt с программным именем filename1
    //  для записи и для чтения
    filename1.open("name.txt",ios::in | ios::out);
    return 0;
}
```

ПРИМЕР 5. Чтение информации из файла прямого доступа

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

// Для считывания и записи блоков двоичных данных в C++
// используются функции read() и write(), которые являются
// членами потоковых классов для ввода и вывода. Прототипы:
//   istream &read (char *str, streamsize  count);
//   ostream &write (const char *str, streamsize  count);

                    // вывод на экран компонент файла
void read_file(ifstream &infile);

int main()
{
 // нужно открыть файл file.bin с программным именем unfile
 //   в режиме считывания      ios::in - по умолчанию
 ifstream infile ("file.bin", ios::binary | ios::_Nocreate);

  if (!infile)
    {  cout<<"error2"<<endl;
       return 1;
     }

    read_file(infile);// вывод на экран значений из file.bin
    infile.close();

    return 0;
}

                   // вывод на экран компонент файла
void read_file(ifstream &infile)
{
   int v;
   infile.read(reinterpret_cast <char*> (&v), sizeof (int));
   while( !infile.eof() )   //  до достижения конца файла
    {
        cout<<v<<" "<<"\n";
            //  чтение элемента из файла  infile.read
    infile.read(reinterpret_cast <char*>(&v), sizeof (int));
    }
}
```

ПРИМЕР 6. Чтение информации из текстового файла. Оператор >>

```
#include <fstream>
#include <iostream>
using namespace std;
                            // вывод на экран строк файла
void read_file(ifstream &infile);

int main()
{
// нужно открыть файл file.txt с программным именем unfile
//   в режиме считывания  ios::in - по умолчанию
   ifstream infile ("file.txt");

   if (!infile)
     { cout<<"error2"<<endl;
       return 1;
     }

    read_file(infile); //вывод на экран значений из file.txt
    infile.close();

    return 0;
}

                                  // вывод на экран строк файла
void read_file(ifstream &infile)
{
    const unsigned MAX = 80;
    char v[MAX];
    infile>>v;                                          //  >>;
    while( !infile.eof() )//  до достижения конца файла
//  можно  while(infile) !!!
//  можно  while( !infile.fail() )
//       более корректно для текстового файла
    {
        cout<<v<<endl;
        infile>>v;       //  чтение строки из файла  infile
    }

}
```

ПРИМЕР 7. Чтение информации из текстового файла. Метод getline

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

                  // вывод на экран строк файла
void read_file(ifstream &infile);

int main()
{
     ifstream infile ("file.txt");
// нужно открыть файл file.txt с программным именем unfile
//    в режиме считывания  ios::in - по умолчанию

     if (!infile)
       {  cout<<"error2"<<endl;
          return 1;
       }

     read_file(infile); //вывод на экран значений из file.txt
     infile.close();
     return 0;
}


                        // вывод на экран строк файла
void read_file(ifstream &infile)
{
     const unsigned MAX = 80;
     char v[MAX];

     infile.getline(v, MAX);
              //  можно infile.get(v[i])  можно  infile>>v;
     while( !infile.eof() ) //  до достижения конца файла
     {
          cout<<v<<endl;
                     //  чтение строки из файла  infile
          infile.getline(v, MAX);
     }
}
```

ПРИМЕР 8. Запись информации в файл прямого доступа

```
#include <stdlib.h>
#include <time.h>                       // for time()
#include <fstream>
#include <iostream>
using namespace std;

// Для считывания и записи блоков двоичных данных в C++
// используются  функции read() и write(), которые являются
// членами потоковых классов для ввода и для вывода.
//   istream &read (char *str, streamsize  count);
//   ostream &write (const char *str, streamsize  count);

                  // создание файла случайных  целых чисел
void create(ofstream &outfile, int n);

int main()
{
     srand((unsigned)time(NULL));
     int n;
     cout<<"enter n"<<endl;
     cin>>n;                // количество чисел файла

                            // создадим файл сл. чисел file.bin
    ofstream outfile("file.bin", ios::binary);
     if (!outfile)
       { cout<<"error1"<<endl;
         return 1;
       }

     create(outfile, n);
     outfile.close();
     return 0;
}
                // создание файла случайных целых чисел
void create(ofstream &outfile, int n)
{
 for (int i = 1; i <= n; i++)
 {
  int  v = rand() % 100;
  outfile.write(reinterpret_cast <char*>(&v), sizeof (int));
  }
}
```

ПРИМЕР 9. Запись информации в текстовый файл

```
#include <stdlib.h>
#include <time.h>                    // for time()
#include <fstream>
#include <iostream>
using namespace std;

                    // создание файла случайных  целых чисел
void create(ofstream &outfile, int n);

int main()
{
     srand((unsigned)time(NULL));
     int n;
     cout<<"enter n"<<endl;
     cin>>n;                  // количество чисел файла

                         // создадим текстовый файл сл. чисел
          ofstream outfile("file.txt");
     if (!outfile)
        {cout<<"error1"<<endl;
         return 1;
        }

     create(outfile, n);
     outfile.close();

     return 0;
}


          // создание текстового файла случайных целых чисел
void create(ofstream &outfile, int n)
{
     for (int i = 1; i <= n;  i++)
     {
          int v = rand()%100;
          outfile<<v<<endl;     // запись целых чисел в файл
                                // каждое число в новой строке
     }
}
```

ПРИМЕР 10. Определение текущей позиции указателя

```
// Только для двоичных файлов
// Для потока istream - функция pos_type tellg()
// Для потока ostream - функция pos_type tellp ()
// тип pos_type позволяет хранить самое большое допустимое
// значение, которое может возвратить любая из этих функций

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

                        // определение текущей позиции
int main()
{
     int n, k, v;

     ifstream filename ("file.bin", ios::binary | ios::in);
     if (!filename)
       {cout<<"error1"<<endl;
        return 1;
       }

     cout<<" k= "<<endl;
     cin>>k;

     for (int i = 0; i < k; i++)
      filename.read(reinterpret_cast<char*>(&v),sizeof(int));

     n = filename.tellg(); //определяем номер текущей позиции
     cout<<" n= "<<n<<endl;

     filename.close();

     return 0;
}
```

ПРИМЕР 11. Смещение указателя в указанную позицию. Запись в позицию

```
//  Только для двоичных файлов
//  Прототипы функций для произвольного доступа  к файлу
//  istream &seekg (off_type offset, seekdir origin);
//  ostream &seekp (off_type offset, seekdir origin);
//  тип off_set позволяет хранить самое большое допустимое
//  значение для  параметра offset
//  тип seekdir может иметь три значения
//  ios::beg   начало файла
//  ios::cur    текущая позиция
//  ios::end    конец файла
//  Для потока istream - функция seekg () перемещает текущий
//  указатель на offset  байт  относительно позиции,
//  заданной параметром origin .
//  Для потока ostream - функция seekp () перемещает текущий
//  указатель на offset  байт  относительно позиции,
//   заданной параметром origin .
                        // запись в n позицию файла числа v
#include <fstream>
#include <iostream>
using namespace std;

int main()
{   int n, v;
// нужно открыть файл file.bin с программным именем filename
//  для записи и для чтения
 fstream filename ("file.bin",ios::binary|ios::in|ios::out);
 if ( !filename)
   { cout<<"error1"<<endl;
     return 1;
    }

    cout<<" n = "<<endl;
    cin>>n;                                  // номер позиции
    cout<<" number = "<<endl;
    cin>>v;                                   // новое число
                      // устанавливаем указатель на позицию
                      // n*sizeof(тип компоненты)
    filename.seekp(n * sizeof(int), ios::beg);
                                          // запись числа
    filename.write(reinterpret_cast<char*>(&v),sizeof(int));

    filename.close();
    return 0;
}
```

ПРИМЕР 12. Смещение указателя в указанную позицию. Чтение из позиции

```
//  Только для двоичных файлов
//  Прототипы функций для произвольного доступа  к файлу
//  istream &seekg (off_type offset, seekdir origin);
//  ostream &seekp (off_type offset, seekdir origin);
//  тип off_set позволяет хранить самое большое допустимое
//  значение для параметра offset
//  тип seekdir может иметь три значения
//  ios::beg   начало файла
//  ios::cur    текущая позиция
//  ios::end    конец файла
//  Для потока istream - функция seekg () перемещает текущий
//  указатель на offset  байт  относительно позиции,
//  заданной параметром origin .
//  Для потока ostream - функция seekp () перемещает текущий
//  указатель  на offset  байт  относительно позиции,
//   заданной параметром origin .

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

int main()
{
    int n, v;

    ifstream filename ("file.bin", ios::binary | ios::in);
    if (!filename)
      { cout<<"error1"<<endl;
        return 1;
      }

    cout<<" n = "<<endl;
    cin>>n;

                //  устанавливаем указатель на позицию
                // n*sizeof(тип компоненты)
    filename.seekg(n*sizeof(int),ios::beg);
                                    // чтение числа
    filename.read(reinterpret_cast<char*>(&v),sizeof (int));

    cout<<" number= "<<v<<endl;
    filename.close();
    return 0;
}
```

ПРИМЕР 13. Определение числа компонент в файле

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

int main()
{

     ifstream infile ("file.txt", ios::binary);
     if ( !infile)
       { cout<<"error1"<<endl;
         return 1;
       }

     infile.seekg(0, ios::end);  //  указатель в конец файла
                                 //  определяем текущую позицию указателя
     int n = infile.tellg();
     n = n / sizeof(int);      //  количество компонент
                               //  sizeof(тип компонент файла)

     cout<<n<<endl;
     infile.close();

     return 0;
}
```

ПРИМЕР 14. Удаление файла

```
#include <stdio.h>                              // for remove()
#include <iostream>
using namespace std;


void main( void )
{
              //  Удаление файла
              //  функция  int remove( const char *path );

     if( remove( "file.txt" ) == -1 )
          cout<<"Could not delete 'file.txt'"<<endl;
     else
          cout<<"Deleted 'file.txt'"<<endl;
}
```

ПРИМЕР 15. Уничтожение информации от текущей позиции указателя до конца

```
#include <stdio.h>          //  for remove()  rename()
#include <fstream>
#include <iostream>
using namespace std;

int main()
{    int v, rezult;
     unsigned n;
    // открыть файл file.bin с именем infile для чтения
     ifstream infile ("file.bin", ios::binary);
     if (!infile)
       { cout<<"error1"<<endl;
         return 1;
        }
     cout<<" n = "<<endl; cin>>n;       // номер позиции
        //открыть временный файл file1.bin  для записи
     ofstream outfile ("file1.bin",ios::binary);
     if (!outfile)
       { cout<<"error1"<<endl;
         return 2;
       }
  //  чтение элемента из файла infile до необходимой позиции
    while(!infile.eof() && infile.tellg() <= n * sizeof(int))
    {
     infile.read(reinterpret_cast<char*>(&v),sizeof (int));
                      // запись элементов в outfile
     outfile.write(reinterpret_cast<char*>(&v),sizeof (int));
     }
     infile.close();
     outfile.close();

     rezult=remove( "file.bin");  // удаление исходного файла
     if (rezult != -1)
     {    // переименование временного файла в исходный файл
          rezult = rename("file1.bin", "file.bin");
          if (rezult)   {cout<<"error3"<<endl; return 3;}
     }
     else
        { cout<<"error4"<<endl;
          return 4;
    }
     return 0;
}
```

ПРИМЕР 16. Работа с бинарным файлом

```
//   Задача: создание бинарного файла случайных целых чисел,
//   вывод чисел на экран, создание нового файла только из
//   четных чисел исходного файла

#include <ctime>
#include <stdlib.h>
#include <fstream>
#include <iostream>
using namespace std;

void create(ofstream&,int);       // создание файла сл. чисел
void read_file(ifstream &);       // вывод на экран
          // создание файла четных чисел из файла исходного
void create_chet(ofstream &, ifstream &);

int main()
{    srand((unsigned)time(NULL));
     int n;
     cout<<"enter n"<<endl; cin>>n; // количество чисел файла
     // открыть file.bin  с именем outfile  для записи
     ofstream outfile("file.bin",ios::binary);
     if (!outfile)
       { cout<<"error1"<<endl;
         return 1;
       }
                // создание файла сл. чисел  file.bin
     create(outfile,n);
     outfile.flush();

          // открыть file.bin  с именем infile  для чтения
  ifstream infile ("file.bin", ios::binary);
  if (!infile)
    { cout<<"error2"<<endl;
      return 2;
     }

     read_file(infile);       // вывод на экран file.bin
     infile.close();
 // открыть file1.bin  с  именем outfile_new  для записи
     ofstream outfile_new ("file1.bin", ios::binary);
     if (!outfile_new)
       { cout<<"error3"<<endl;
         return 3;
        }
 // открыть file.bin с программным именем infile  для чтения
```

```
    infile.clear();
    infile.open("file.bin", ios::binary);
    if (!infile)
      { cout<<"error4"<<endl;
        return 4;
       }
                  // создание файла четных чисел file1.bin
                  // из файла исходного file.bin
    create_chet(outfile_new,infile);
    outfile_new.flush();        // outfile_new.close();
    infile.close();
//открыть file1.bin  с программным именем infile  для чтения
    infile.clear();
    infile.open("file1.bin", ios::binary);
    if (!infile)
      { cout<<"error5"<<endl;
        return 5;
      }
            // вывод на экран файла четных чисел file1.bin
    read_file(infile);
    infile.close();
    return 0;
}

                     // создание файла из n сл. чисел
void create(ofstream &outfile, int   n)
{
    for (int i = 1; i <= n;  i++)
    {
        int v = rand() % 100;
 outfile.write(reinterpret_cast <char*> (&v), sizeof (int));
    }
}
                     // вывод на экран чисел файла
void read_file(ifstream &infile)
{
    int v;
    infile.read(reinterpret_cast<char*>(&v),sizeof (int));
    while( !infile.eof())
    {
        cout<<v<<" ";
    infile.read(reinterpret_cast<char*>(&v),sizeof (int));
    }
    cout<<endl;
}
         // создание файла четных чисел из файла исходного
```

```
void create_chet(ofstream &outfile_new,  ifstream &infile)
{
     int ch;

     infile.read(reinterpret_cast<char*>(&ch), sizeof(int));

     while (infile)
     {
          if(ch % 2 == 0)
         outfile_new.write(reinterpret_cast<char*>(&ch),
                           sizeof (int));
          infile.read(reinterpret_cast<char*>(&ch),
                       sizeof(int));
     }
}
```

ПРИМЕР 17. Работа с текстовым файлом

```
//  Задача: создание текстового файла случайных целых чисел,
//  вывод чисел на экран, создание нового файла только из
//  четных чисел исходного файла

#include <ctime>
#include <stdlib.h>
#include <fstream>
#include <iostream>
using namespace std;

void create(ofstream&,int);    // создание файла сл. чисел
void read_file(ifstream &);  // вывод на экран
          // создание файла четных чисел из файла исходного
void create_chet(ofstream &, ifstream &);

int main()
{    srand((unsigned)time(NULL));
     int n;
     cout<<"enter n"<<endl; cin>>n;// количество чисел файла

// открыть file.txt с программным именем outfile  для записи
     ofstream outfile("file.txt");
     if (!outfile)
       { cout<<"error1"<<endl;
         return 1;
        }
                // создание файла сл. чисел  file.txt
     create(outfile,n);
     outfile.flush();
// открыть file.txt  с программным именем infile  для чтения
     ifstream infile ("file.txt");
     if (!infile)
       { cout<<"error2"<<endl;
         return 2;
       }
     read_file(infile);
                             // вывод на экран file.txt
     infile.close();
// открыть file1.txt  с  именем outfile_new  для записи
     ofstream outfile_new ("file1.txt");
     if (!outfile_new)
       { cout<<"error3"<<endl;
          return 3;
        }
```

```
// открыть file.txt  с программным именем infile  для чтения
     infile.clear();
     infile.open("file.txt");
     if (!infile)
       { cout<<"error4"<<endl;
         return 4;
        }
     // создание файла четных чисел file1.txt  из файла
     // исходного file.txt
     create_chet(outfile_new,infile);
     outfile_new.flush();              // outfile_new.close();
     infile.close();
 // открыть file1.txt  с  именем infile  для чтения
     infile.clear();
     infile.open("file1.txt");
     if (!infile)
       { cout<<"error5"<<endl;
         return 5;
        }
            // вывод на экран файла четных чисел file1.txt
     read_file(infile);
     infile.close();
     return 0;
}
                          // создание файла n сл. чисел
void create(ofstream &outfile,  int   n)
{
     for (int i = 1; i <= n; i++)
     {
          int v = rand() % 100;
          outfile<<v<<endl;
     }
}

                  // вывод на экран чисел файла
void read_file(ifstream &infile)
{
     int v;
     infile>>v;
     while( !infile.fail())  //while( !infile.eof())
     {
          cout<<v<<" ";
          infile>>v;
     }
     cout<<endl;
}
```

```
        // создание файла четных чисел из файла исходного
void create_chet(ofstream &outfile_new, ifstream &infile)
{
     int v;
     infile>>v;
     while (!infile.eof())
     {
          if(v % 2 == 0)
               outfile_new<<v<<endl;
          infile>>v;
     }
}
```

СЛОВАРЬ ПОНЯТИЙ, ИСПОЛЬЗУЕМЫХ В ЗАДАНИЯХ