ГОСУДАРСТВЕННЫЙ КОМИТЕТ СВЯЗИ, ИНФОРМАТИЗАЦИИ И ТЕЛЕКОММУНИКАЦИОННЫХ ТЕХНОЛОГИИ РЕСПУБЛИКИ УЗБЕКИСТАН

НУКУССКИЙ ФИЛИАЛ ТАШКЕНТСКОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИИ

Факультет: Компьютер инжиниринг

КУРСОВАЯ РАБОТА

По предмету:Программирование на языкеС++

На тему:Работа с файлами

Выполнила:студентка 2в курса Компьютер инжинирингаТилепова А.

Принял:Ядгаров Ш.

НУКУС 2015

План:

Введение

Основная часть

1. **Организация работы с файлами средствами C++**
2. **Режимы открытия файлов**
3. **Обработка двоичных файлов**
4. **Блокировка**

Прикладная работа

Заключение

Список использованной литературы

Введение

Большинство компьютерных программ работают с файлами, и поэтому возникает необходимость создавать, удалять, записывать читать, открывать файлы. Что же такое файл? **Файл** – именованный набор байтов, который может быть сохранен на некотором накопителе. Файлом называют способ хранения информации на физическом устройстве. Ну, теперь ясно, что под файлом понимается некоторая последовательность байтов, которая имеет своё, уникальное имя, например **файл.txt**. В одной директории не могут находиться файлы с одинаковыми именами. Под именем файла понимается не только его название, но и расширение, например: **file.txt** и **file.dat -** разные файлы, хоть и имеют одинаковые названия. Существует такое понятие, как полное имя файлов – это полный адрес к директории файла с указанием имени файла, например: **D:\docs\file.txt**. Важно понимать эти базовые понятия, иначе сложно будет работать с файлами.В **C++** отсутствуют операторы для работы с файлами. Все необходимые действия выполняются с помощью функций, включенных в стандартную библиотеку. Они позволяют работать с различными устройствами, такими, как диски, принтер, коммуникационные каналы и т.д. Эти устройства сильно отличаются друг от друга. Однако файловая система преобразует их в единое абстрактноелогическое устройство, называемое **потоком**.

Текстовый поток — это последовательность символов. При передаче символов из потока на экран, часть из них не выводится (например, символ возврата каретки, перевода строки).

Двоичный поток — это последовательность байтов, которые однозначно соответствуют тому, что находится на внешнем устройстве.

Организация работы с файлами средствами C++

Файловый ввод-вывод с использованием потоков:

-Библиотека потокового ввода-вывода **fstream**

-Связь файла с потоком вывода:**ofstream** имя логического файла;

-Связь файла с потоком ввода:**ifstream** имя логического файла;

-Открытие файла имя логического **файла.open**(имя физического файла);

-Закрытие файла имя логического **файла.close();**

Режимы открытия файлов

Режимы открытия файлов устанавливают характер использования файлов. Для установки режима в классе **ios_base** предусмотрены константы, которые определяют режим открытия файлов (см. Таблица 1).

Таблица 1 — режимы открытия файлов

онстанта	**Описание**
ios_base::in	открыть файл для чтения
ios_base::out	открыть файл для записи
ios_base::ate	при открытии переместить указатель в конец файла
ios_base::app	открыть файл для записи в конец файла
ios_base::trunc	удалить содержимое файла, если он существует
ios_base::binary	открытие файла в двоичном режиме

Режимы открытия файлов можно устанавливать непосредственно при создании объекта или при вызове функции **open().**

```
ofstreamfout("cppstudio.txt", ios_base::app); // открываемфайл
1 для добавления информации к концу файла
2 fout.open("cppstudio.txt", ios_base::app); // открываем файлдля
  добавления информации к концу файла
```

Режимы открытия файлов можно комбинировать с помощью поразрядной логической операции **или** |, например: ios_base::out | ios_base::trunc - открытие файла для записи, предварительно очистив его.

Объекты класса **ofstream**, при связке с файлами по умолчанию содержат режимы открытия файлов ios_base::out | ios_base::trunc**.** То есть файл будет создан, если не существует. Если же файл существует, то его содержимое будет удалено, а сам файл будет готов к записи. Объекты класса ifstream связываясь с файлом, имеют по умолчанию режим открытия файла ios_base::in - файл открыт только для чтения. Режим открытия файла ещё называют — флаг, для удобочитаемости в дальнейшем будем использовать именно этот термин. В таблице 1 перечислены далеко не все флаги, но для начала этих должно хватить.

Обратите внимание на то, что флаги ate и app по описанию очень похожи, они оба перемещают указатель в конец файла, но флаг app позволяет производить запись, только в конец файла, а флаг ate просто переставляет флаг в конец файла и не ограничивает места записи.

Разработаем программу, которая, используя операцию sizeof(), будет вычислять характеристики основных типов данных в C++ и записывать их в файл. Характеристики:

1. число байт, отводимое под тип данных

2. максимальное значение, которое может хранить определённый тип данных.

Запись в файл должна выполняться в таком формате:

```
1  /*  data type          byte     max value
2  bool                  =  1     255.00
3  char                  =  1     255.00
4  short int             =  2     32767.00
5  unsigned short int =  2      65535.00
6  int                   =  4     2147483647.00
7  unsigned int          =  4     4294967295.00
8  long int              =  4     2147483647.00
9  unsigned long int     =  4     4294967295.00
10float                  =  4     2147483647.00
11long float             =  8     92233720368547758O0.00
12double                 =  8     92233720368547758O0.00  */
```

Такая программа уже разрабатывалась ранее в разделе Типы данных C++ , но там вся информация о типах данных выводилась на стандартное устройство вывода, а нам необходимо программу переделать так, чтобы информация записывалась в файл. Для этого необходимо открыть файл в режиме записи, с предварительным усечением текущей информации файла (**строка 14**). Как только файл создан и успешно открыт (строки 16 — 20), вместо оператора cout, в **строке 22** используем объект fout. таким образом, вместо экрана информация о типах данных запишется в файл.

ПРОВЕРКА ОШИБОК ПРИ ВЫПОЛНЕНИИ ФАЙЛОВЫХ ОПЕРАЦИЙ

Программы, представленные до настоящего момента, предполагали, что во время файловых операций В/В не происходят ошибки. К сожалению, это сбывается не всегда. Например, если вы открываете файл для ввода, ваши программы должны проверить, что файл существует. Аналогично, если ваша программа пишет данные в файл, вам необходимо убедиться, что операция прошла успешно (к примеру, отсутствие места на диске, скорее всего, помешает записи данных). Чтобы помочь вашим программам следить за ошибками, вы можете использовать функцию **fail** файлового объекта. Если в процессе файловой операции ошибок не было, функция возвратит ложь (0). Однако, если встретилась ошибка, функция **fail** возвратит истину. Например, если программа открывает файл, ей следует использовать функцию **fail**, чтобы определить, произошла ли ошибка, как это показано ниже:

```
ifstreaminput_file("FILENAME.DAT");
if (input_file.fail())
{
  cerr<< "Ошибкаоткрытия FILENAME.EXT" <<endl;
  exit(1);
}
```

Таким образом, программы должны убедиться, что операции чтения и записи прошли успешно. Следующая программа **TEST_ALL.CPP** использует функцию **fail**для проверки различных ошибочных ситуаций:

```
#include <iostream.h>
#include <fstream.h>
void main(void)
```

```
{
  char line[256] ;
  ifstreaminput_file("BOOKINFO.DAT") ;
  if (input_file.fail()) cerr<< "Ошибкаоткрытия BOOKINFO.DAT" <<endl;
  else
  {
    while ((! input_file.eof()) && (! input_file.fail()))
    {
       input_file.getline(line, sizeof(line)) ;
      if (! input_file.fail()) cout<< line <<endl;
    }
  }
}
```

ОПРЕДЕЛЕНИЕ КОНЦА ФАЙЛА

Обычной файловой операцией в ваших программах является чтение содержимого файла, пока не встретится конец файла. Чтобы определить конец файла, ваши программы могут использовать функцию **eof** потокового объекта. Эта функция возвращает значение 0, если конец файла еще не встретился, и 1, если встретился конец файла. Используя цикл ***while***, ваши программы могут непрерывно читать содержимое файла, пока не найдут конец файла, как показано ниже:

```
while (! input_file.eof())
{
  // Операторы
}
```

В данном случае программа будет продолжать выполнять цикл, пока функция **eof**возвращает ложь (0). Следующая программа TEST_EOF.CPP использует функцию **eof** длячтения содержимого файла BOOKINFO.DAT, пока не достигнет конца файла:

```
#include <iostream.h>

#include <fstream.h>

void main (void)

{
  ifstreaminput_file("BOOKINFO.DAT");
  char line[64];
  while (! input_file.eof())

  {
    input_file.getline(line, sizeof(line));
    cout<< line <<endl;
  }
}
```

Аналогично, следующая программа **WORD_EOF.CPP** читает содержимое файла по одному слову за один раз, пока не встретится конец файла:

```
#include <iostream.h>

#include <fstream.h>

void main(void)

{
 ifstreaminput_file("BOOKINFO.DAT");
 char word[64] ;
 while (! input_file.eof())

 {
 input_file>> word;
 cout<< word <<endl;
  }
}
```

И наконец, следующая программа CHAR_EOF.CPP читает содержимое файла по одному символу за один раз, используя функцию **get**, пока не встретит конец файла:

```
#include <iostream.h>
```

```
#include <fstream.h>

void main(void)

{
  ifstreaminput_file("BOOKINFO.DAT");
  char letter;
  while (! input_file.eof())

  {
    letter = input_file.get();
    cout<< letter;
  }
}
```

ЗАКРЫТИЕ ФАЙЛА, ЕСЛИ ОН БОЛЬШЕ НЕ НУЖЕН

При завершении вашей программы операционная система закроет открытые ею файлы. Однако, как правило, если вашей программе файл больше не нужен, она должна его закрыть. Для закрытия файла ваша программа должна использовать функцию **close**, как показано ниже:

input_file.close ();

Когда вы закрываете файл, все данные, которые ваша программа писала в этот файл, сбрасываются на диск, и обновляется запись каталога для этого файла.

ВЫПОЛНЕНИЕ ОПЕРАЦИЙ ЧТЕНИЯ И ЗАПИСИ

Все программы, представленные в данном уроке, выполняли файловые операции над символьными строками. По мере усложнения ваших программ, возможно, вам понадобится читать и писать массивы и

структуры. Для этого ваши программы могут использовать функции **read**и **write**. При использовании функций **read**и **write**вы должны указать буфер данных, в который данные будут читаться или из которого они будут записываться, а также длину буфера в байтах, как показано ниже:

```
input_file.read(buffer, sizeof(buffer)) ;
output_file.write(buffer, sizeof(buffer));
```

Например, следующая программа STRU_OUT.CPP использует функцию **write**для вывода содержимого структуры в файл EMPLOYEE.DAT:

```
#include <iostream.h>

#include <fstream.h>

void main(void)

{
 struct employee

{
    char name[64];
    int age;
    float salary;
 } worker = { "ДжонДой", 33, 25000.0 };

ofstreamemp_file("EMPLOYEE.DAT") ;
emp_file.write((char *) &worker, sizeof(employee));
}
```

Функция **write**обычно получает указатель на символьную строку. Символы *(char *)* представляют собой оператор приведения типов, который информирует компилятор, что вы передаете указатель на другой тип. Подобным образом следующая программа STRU_IN.CPP использует метод **read***для* чтения из файла информации о служащем:

```
#include <iostream.h>

#include <fstream.h>
```

```
void main(void)
{
struct employee
{
    char name [6 4] ;
    int age;
    float salary;
 } worker = { "ДжонДой", 33, 25000.0 };
 ifstreamemp_file("EMPLOYEE.DAT");
 emp_file.read((char *) &worker, sizeof(employee));
 cout<< worker.name <<endl;
 cout<<worker.age<<endl;
  cout<<worker.salary<<endl;
}
```

1. Заголовочный файл **fstream.h**определяет классы **ifstream**и **ofstream**, с помощью которых ваша программа может выполнять операции файлового ввода и вывода.

2. Для открытия файла на ввод или вывод вы должны объявить объект типа **ifstream**или **ofstream**, передавая конструктору этого объекта имя требуемого файла.

3. После того как ваша программа открыла файл для ввода или вывода, она может читать или писать данные, используя операторы извлечения (>>) и вставки (<<).

4. Ваши программы могут выполнять ввод или вывод символов в файл или из файла, используя функции **get**и **put**.

5. Ваши программы могут читать из файла целую строку, используя функцию **getline**.

6. Большинство программ читают содержимое файла, пока не встретится конец файла. Ваши программы могут определить конец файла с помощью функции **eof**.

7. Когда ваши программы выполняют файловые операции, они должны проверять состояние всех операций, чтобы убедиться, что операции выполнены успешно. Для проверки ошибок ваши программы могут использовать функцию **fail.**

8. Если вашим программам необходимо вводить или выводить такие данные, как структуры или массивы, они могут использовать методы **read**и **write.**

9. Если ваша программа завершила работу с файлом, его следует закрыть с помощью функции **close.**

Обработка двоичных файлов

При записи информации в двоичный файл символы и числа записываются в виде последовательности байт.

Для того чтобы записать данные в двоичный файл, необходимо:

1. описать файловую переменную типа **FAIL *** с помощью оператора **FILE *filename**, здесь **filename** — имя переменной, где будет храниться указатель на файл.
2. открыть файл с помощью функции **fopen**
3. записать информацию в файл с помощью функции **fwrite**
4. закрыть файл с помощью функции **fclose**

Для того чтобы считать данные из двоичного файла, необходимо:

1. описать переменную типа **FILE ***
2. открыть файл с помощью функции **fopen**
3. считать необходимую информацию из файла с помощью функции **fread**, при этом следить за тем достигнут ли конец файла.
4. закрыть файл с помощью функции **fclose**

Рассмотрим основные функции, необходимые для работы с двоичными файлами.

Для открытия файла предназначена функция **fopen**.

FILE *fopen(const *filename, const char *mode)

Здесь **filename** — строка, в которой хранится полное имя открываемого файла, **mode** — строка, определяющая режим работы с файлом; возможны следующие значения:

- «**rb**» — открываем двоичный файл в режиме чтения;
- «**wb**» — создаем двоичный файл для записи; если он существует, то его содержимое очищается;
- «**ab**» — создаем или открываем двоичный файл для дозаписи в конец файла;
- «**rb+**» — открываем существующий двоичный файл в режиме чтения и записи;
- «**wb+**» — открываем двоичный файл в режиме чтения и записи, существующий файл очищается;
- «**ab+**» — двоичный файл открывается или создается для исправления существующий информации и добавления новой в конец файла.

Функция возвращает в файловой переменной **f** значение **NULL** в случае неудачного открытия файла. После открытия файла доступен 0-й его байт, указатель файла равен 0, значение которого по мере чтения или записи смещается на считанное (записанное) количество байт. Текущие значение указателя файла — номер байта, начиная с которого будет происходить операция чтения или записи.Для закрытия файла предназначена функция **fclose**:**intfclose(FILE *filename);**

Она возвращает 0 при успешном закрытие файла и **EOF** в противном случае.Функция**remove** предназначена для удаления файлов:

int remove(const char *filename);

Эта функция удаляет с диска файл с именем filenema. Удаляемый файл должен быть закрыт. Функция возвращает ненулевое значение, если файл не удалось удалить.Для переименования файлов предназначена функция **rename**:**intrename(constchar *oldfilename, constchar *newfilename);**

Первый параметр — старое имя файла, второй — новое. Возвращает 0 при удачном завершении программы.Чтение из двоичного файла осуществляется с помощью функции **fread**:**fread(void *ptr, size, n, FILE *filename);**

Функция **fread** считывает из файла **filename** в массив **ptr n** элементов размера **size**. Функция возвращает количество считанных элементов. После чтения из файла его указатель смещается на **n*size** байт.Запись в двоичный файл осуществляется с помощью функции **fwrite**:

fwrite(const void *ptr, size, n, FILE *filename);

Функция **fwrite** записывает в файл **filename** из массива **ptr n** элементов размера **size**. Функция возвращает количество записанных элементов. После записи информации в файл указатель смещается на **n*size** байт.Для контроля достижения конца файла есть функция **feof**:

intfeof(FILE *filename);

Она возвращает ненулевое значение если достигнут конец файла.Для более точного усвоения материала предлагаю рассмотреть пару стандартных задач.

Задача 1

Создать двоичный файл **D:\\game\\noobs.dat** и записать в него целое число n и n вещественных чисел.

Решение:

```
#include "stdafx.h"
#include<iostream>
usingnamespacestd;
intmain()
{
setlocale (LC_ALL, "RUS");
intn, i;
doublea;
FILE *f; //описываем файловую переменную
//создаем двоичный файл в режиме записи
f=fopen("D:\\game\\noobs.dat", "wb");
//ввод числа n
cout<<"n="; cin>>n;
fwrite(&n, sizeof(int), 1, f);
//цикл для ввода n вещественных чисел
for (i=0; i<n; i++)
{
//ввод очередного вещественного числа
cout<<"a=";
cin>>a;
//запись вешественного числа в двоичный файл
fwrite(&a, sizeof(double), 1, f);
}
//закрываем файл
fclose(f);
system("pause");
```

return 0;

}

Задача 2

В текстовом файле D:\\game\\accounts.txt хранятся вещественные числа, вывести их на экран и вычислить их количество.

Решение:

```
#include "stdafx.h"
#include<iostream>
#include<fstream>
#include<iomanip>
#include<stdlib.h>
using namespace std;
int main()
{
setlocale (LC_ALL, "RUS");
int n=0;
float a;
fstream F;
//открываем файл в режиме чтения
F.open("D:\\sites\\accounts.txt");
//если открытие файла прошло корректно, то
if (F)
{
//цикл для чтения значений из файла;выполнение цикла прервется,
//когда достигнем конца файла, в этом случае F.eof() вернет истину.
while (!F.eof())
{
//чтение очередного значения из потока F в переменную a
F>>a;
//вывод значения переменной a на экран
cout<<a<<"\t";
//увеличение количества считанных чисел
n++;
}
```

```
29 //закрытие потока
30 F.close();
31 //вовод на экран количества считанных чисел
32 cout<<"n="<<n<<endl;
33 }
34 //если открытие файла прошло некорректно, то вывод
35 //сообщения об отсутствии такого файла
36 else cout<<" Файл не существует"<<endl;
37 system("pause");
38 return 0;
39 }
```

Блокировка

В состав класса включены методы LockRange и UnlockRange, позволяющие заблокировать один или несколько фрагментов данных файла для доступа из других процессов. Если приложение пытается повторно блокировать данные, уже заблокированные раньше этим или другим приложением, вызывается исключение. Блокировка представляет собой один из механизмов, позволяющих нескольким приложениям или процессам одновременно работать с одним файлом, не мешая друг другу.Установить блокировку можно с помощью метода **LockRange**. Чтобы снять установленные блокировки, надо воспользоваться методом **UnlockRange**. Если в одном файле установлены несколько блокировок, то каждая из них должна сниматься отдельным вызовом метода **UnlockRange**.

Позиционирование

Чтобы переместить указатель текущей позиции файла в новое положение, можно воспользоваться одним из следующих методов класса **CFile** - **Seek, SeekToBegin, SeekToEnd.** В состав класса **CFile** также входят методы, позволяющие установить и изменить длину файла, - **GetLength, SetLength**.

При открытии файла указатель текущей позиции файла находится в самом начале файла. Когда порция данных прочитана или записана, то указатель текущей позиции перемещается в сторону конца файла и указывает на данные, которые будут читаться или записываться очередной операцией чтения или записи в файл.Чтобы переместить указатель текущей позиции файла в любое место, можно воспользоваться универсальным методом **Seek**. Он позволяет переместить указатель на определенное число байт относительно начала, конца или текущей позиции указателя. Чтобы переместить указатель в начало или конец файла, наиболее удобно использовать специальные методы. Метод **SeekToBegin** перемещает указатель в начало файла, а метод **SeekToEnd** - в его

конец.Но для определения длины открытого файла совсем необязательно перемещать его указатель. Можно воспользоваться методом **GetLength**. Этот метод также возвращает длину открытого файла в байтах. Метод **SetLength** позволяет изменить длину открытого файла. Если при помощи этого метода размер файла увеличивается, то значение последних байт не определено.Текущую позицию указателя файла можно определить с помощью метода**GetPosition**. Возвращаемое методом **GetPosition** 32-разрядное значение определяет смещение указателя от начала файла.

Характеристики открытого файла

Чтобы определить расположение открытого файла на диске, надо вызвать метод **GetFilePath**. Этот метод возвращает объект класса **CString**, в котором содержится полный путь файла, включая имя диска, каталоги, имя файла и его расширение.Если требуется определить только имя и расширение открытого файла, можно воспользоваться методом **GetFileName**. Он возвращает объект класса CString, в котором находится имя файла. В случае, когда нужно узнать только имя открытого файла без расширения, пользуются методом **GetFileTitle**.Следующий метод класса CFile позволяет установить путь файла. Это метод не создает, не копирует и не изменяет имени файла, он только заполняет соответствующий элемент данных в объекте класса CFile.

Класс CMemFile

В библиотеку MFC входит класс CMemFile, наследуемый от базового класса CFile. Класс CMemFile представляет файл, размещенный, в оперативной памяти. С объектами класса CMemFile так же, как и с объектами класса CFile. Отличие заключается в том, что файл, связанный с объектом CMemFile, расположен не на диске, а в оперативной памяти компьютера. За счет этого операции с таким файлом происходят значительно быстрее, чем собычнымифайлами.Работая с объектами класса CMemFile, можно

использовать практически все методы класса CFile, которые были описаны выше. Можно записывать данные в такой файл или считывать их. Кроме этих методов в состав класса **CMemFile** включены дополнительные методы.Для создания объектов класса CMemFile предназначено два различных конструктора. Первый конструктор CMemFile имеет всего один необязательный параметр nGrowBytes:

```
CMemFile(UINT nGrowBytes=1024);
```

Этот конструктор создает в оперативной памяти пустой файл. После создания файл автоматически открывается (не нужно вызывать метод Open).Когда начинается запись в такой файл, автоматически выделяется блок памяти. Для получения памяти методы класса **CMemFile** вызывают стандартные функции **malloc, realloc** и **free**. Если выделенного блока памяти недостаточно, его размер увеличивается. Увеличение блока памяти файла происходит по частям по nGrowBytes байт. После удаления объекта класса **CMemFile** используемая память автоматически возвращается системе.Второй конструктор класса CMemFile имеет более сложный прототип. Это конструктор используется в тех случаях, когда программист сам выделяет память для файла:

```
CMemFile(BYTE* lpBuffer, UINT nBufferSize, UINT nGrowBytes=0);
```

Параметр lpBuffer указывает на буфер, который будет использоваться для файла. Размер буфера определяется параметром nBufferSize.Необязательный параметр nGrowBytes используется более комплексно, чем в первом конструкторе класса. Если nGrowBytes содержит нуль, то созданный файл будет содержать данные из буфера lpBuffer. Длина такого файла будет равнanBufferSize.Если nGrowBytes больше нуля, то содержимое буфера lpBuffer игнорируется. Кроме того, если в такой файл записывается больше данных, чем помещается в отведенном буфере, то его размер автоматически увеличивается. Увеличение блока памяти файла происходит по частям по nGrowBytes байт.Класс **CMemFile** позволяет получить указатель на область

памяти, используемую файлом. Через этот указатель можно непосредственно работать с содержимым файла, не ограничивая себя методами класса **CFile**. Для получения указателя на буфер файла можно воспользоваться методом Detach. Перед этим полезно определить длину файла (и соответственно размер буфера памяти), вызвав метод **GetLength**.Метод **Detach** закрывает данный файл и возвращает указатель на используемый им блок памяти. Если опять потребуется открыть файл и связать с ним оперативный блок памяти, нужно вызвать метод **Attach**.Нужно отметить, что для управления буфером файла класс**CMemFile** вызывает стандартные функции **malloc, realloc** и **free**. Поэтому, чтобы не нарушать механизм управления памятью, буфер lpBuffer должен быть создан функциями **malloc** или **calloc**.

Класс CStdioFile

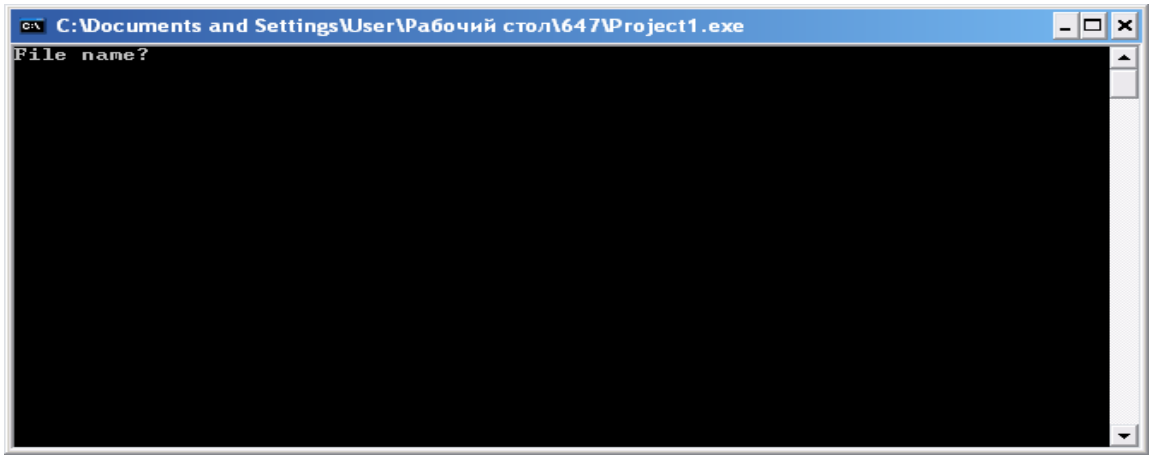
Тем, кто привык пользоваться функциями потокового ввода/вывода из стандартной библиотеки C и C++, следует обратить внимание на класс **CStdioFile**, наследованный от базового класса **CFile**. Этот класс позволяет выполнять буферизированный ввод/вывод в текстовом и двоичном режиме. Для объектов класса **CStdioFile** можно вызывать практически все методы класса CFile.В класс **CStdioFile** входит элемент данных m_pStream, который содержит указатель на открытый файл. Если объект класса **CStdioFile** создан, но файл еще не открыт, либо закрыт, то m_pStream содержит константу NULL.Класс **CStdioFile** имеет три различных конструктора. Первый конструктор класса CStdioFile не имеет параметров. Этот конструктор только создает объект класса, но не открывает никаких файлов. Чтобы открыть файл, надо вызвать метод **Open** базового класса **CFile**.Второй конструктор класса **CStdioFile** можно вызвать, если файл уже открыт и нужно создать новый объект класса CStdioFile и связать с ним открытый файл. Этот конструктор можно использовать, если файл был открыт стандартной функцией **fopen**. Параметр метода должен содержать указатель на файл, полученный вызовом стандартной функции **fopen**.Третий конструктор можно использовать, если

надо создать объект класса **CStdioFile**, открыть новый файл и связать его с только что созданным объектом.Для чтения и записи в текстовый файл класс CStdioFile включает два новых метода: **ReadString**и **WriteString**. Первый метод позволяет прочитать из файла строку символов, а второй метод - записать.

Прикладная работа

Пример 1

```
#include<cstdlib>
#include <iostream>
using namespace std;
int main()
{
 FILE *f; intdat;
srand(time(0));
int n=rand()%30 + 1;
cout<< "aygul ";
char s[20];
cin.getline(s, 20);
 f=fopen(s, "wb");
for (inti=1; i<=n; i++)
{ dat = rand()%101 - 50;
cout<<dat<< " ";
fwrite(&dat, sizeof(int), 1, f);
 }
cout<<endl;
fclose(f);
system("PAUSE");
return EXIT_SUCCESS;
}
```



Пример 2

Запись текстовой информации в файл.

```
#include<iostream>

#include<fstream>
int main()
{
char str_file[]=”Строкадляфайла”;
FILE* fp = fopen(“my_file.txt”,”w”);
if(fp != NULL)
{
printf(“Идетзаписьинформациивфайл…\n”);
for(inti=0;i <strlen(str_file);i++)
putc(str_file[i],fp);
}
else printf(“Невозможнооткрытьфайлназапись.\n”);
fclose(fp);
return 0;
}
```

В данном примере задается специализированный указатель fp типа FILE, который инициализируется функцией fopen(). Функция fopen() в качестве первого аргумента принимает строку, в которой задан путь и имя файла. Вторым параметром определяется способ обработки файла, в данном случае, значение “w”, которое означает открытие файла на запись с удалением всей прежней информации из него. Если файл открыт успешно, то указатель fp не будет равен NULL и с ним возможна работа. В этом

случае с помощью функции putc() выполняется запись символов в файл, на который указывает указатель fp. Перед завершением программы открытый файл следует закрыть во избежание в нем потери данных. Это достигается функцией fclose(), которая принимает указатель на файл и возвращает значение 0 при успешном закрытии файла, иначе значение EOF.

Пример 3

Считывание текстовой информации из файла.

```
#include<iostream>

#include<fstream>
int main()
{
char str_file[100];
FILE* fp = fopen(“my_file.txt”,”r”);
if(fp != NULL)
{
inti=0;
char ch;
while((ch = getc(fp)) != EOF)
str_file[i++]=ch;
str_file[i] = ‘\0’;
printf(str_file);
}
else printf(“Невозможнооткрытьфайлначтение.\n”);
fclose(fp);
return 0;
}
```

В приведенном примере функция fopen() открывает файл на чтение, что определяется значением второго аргумента равного «r». Это значит, что в него невозможно произвести запись данных, а только считывание. Сначала выполняется цикл while, в котором из файла считывается символ с помощью функции getc() и выполняется проверка: если считанное значение не равно символу конца файла EOF, то значение переменной ch записывается в массив str_file. Данный цикл будет выполняться до тех пор, пока не будут считаны все символы из файла, т.е. пока не будет достигнут символ EOF. После завершения цикла формируется строка str_file, которая выводится на экран с помощью функции printf(). Перед завершением программы также выполняется функция закрытия файла fclose().

Работа с текстовыми файлами через функции putc и getc не всегда удобна. Например, если необходимо записать или считать строку целиком, то желательно иметь функции, выполняющие эту работу. В качестве таковых можно воспользоваться функциями fputs() и fgets() для работы со строками. Перепишем предыдущие примеры с использованием данных функций.

Пример 4

Использование функций fpust() и fgets().

```
#include<iostream>

#include<fstream>
int main()
{
char str_file[]=”Строкадляфайла”;
FILE* fp = fopen(“my_file.txt”,”w”);
if(fp != NULL) fputs(str_file,fp);
else printf(“Невозможнооткрытьфайлназапись.\n”);
fclose(fp);
fp = fopen(“my_file.txt”,”r”);
if(fp != NULL)
{
fgets(str_file,sizeof(str_file),fp);
printf(str_file);
}
fclose(fp);
return 0;
}
```

Аналогичные действия по записи данных в файл и считывания информации из него можно выполнить и с помощью функций fprintf() и fscanf(). Однако эти функции предоставляют большую гибкость в обработке данных файла. Продемонстрируем это на следующем примере. Допустим, имеется структура, хранящая информацию о книге: название, автор, год издания. Необходимо написать программу сохранения этой информации в текстовый файл и их считывания.

Пример 5

Использование функций fseek() и ftell().

```
#include<iostream>
```

```
#include<fstream>
int main(void)
{
FILE* fp = fopen("my_file.txt","w");
if(fp != NULL)
{
fprintf(fp,"It is an example using fseek and ftell functions.");
}
fclose(fp);
fp = fopen("my_file.txt","r");
if(fp != NULL)
{
char ch;
fseek(fp,0L,SEEK_END);
long length = ftell(fp);
printf("length = %ld\n",length);
for(inti = 1;i <= length;i++)
{
fseek(fp,-i,SEEK_END);
ch = getc(fp);
putchar(ch);
}
}
fclose(fp);

system("PAUSE");


return 0;
}
```

Вданномпримересначаласоздаетсяфайл, вкоторыйзаписываетсястрока “It is an example using fseek and ftell functions.”. Затем этот файл открывается на чтение и с помощью функции fseek(fp,0L,SEEK_END) указатель позиции помещается в конец файла. Это достигается за счет установки флага SEEK_END, который перемещает позицию в конец файла при нулевом смещении. В результате функция ftell(fp) возвратит число символов в открытом файле. В цикле функция fseek(fp,-i,SEEK_END) смещает указатель позиции на –i символов относительно конца файла, после чего считывается символ функцией getc(), стоящий на i-й позиции с конца. Так как переменная i пробегает значения от 1 до length, то на экран будут выведены символы из файла в обратном порядке.

Пример 6

Файл содержит несколько строк, в каждой из которых записано единственное выражение вида a#b (без ошибок), где a, b - целочисленные величины, # - операция +, -, /, *. Вывести каждое из выражений и их значения.

```
/* Dev-C++ */

#include<cstdlib>

#include <iostream>

#include <fstream>

using namespace std;

int main()

{

long a, b;  char s[256], c;  inti;

cout<< "File name? "; cin>> s;

ifstream f;  f.open(s);

while (!f.eof())

   {    f.getline(s, 256);

    i=0; a=0;

while (s[i]>='0'&&s[i]<='9')

     {

         a=a*10+s[i]-'0';

i++;
```

```
    }
   c=s[i++];    b=0;
while (s[i]>='0'&&s[i]<='9')
    {
        b=b*10+s[i]-'0';
i++;
    }
switch (c){
case '+': a+=b; break;
case '-': a-=b; break;
case '/': a/=b; break;
case '*': a*=b; break;}
cout<< s << " = " << a <<endl;   }
f.close();
system("PAUSE");
return EXIT_SUCCESS;
}
```

Пример 7

```
/* В заданном файле целых чисел посчитать количество компонент,
кратных 3. */
/* Dev-C++ */
#include <cstdlib>
#include <iostream>
#include <fstream>
using namespace std;
int main()
```

```
{intr,ch;
ifstream f;
f.open("CH_Z.TXT");
ch=0;
for (;f.peek()!=EOF;)
    {f>>r;
cout<< r << " ";
if (r%3==0) ch++ ;
     }
f.close();
cout<<endl<< "Answer: " <<ch;
system("PAUSE");
return EXIT_SUCCESS;
}
```

Пример 8

Заполнить файл некоторым количеством целых случайных чисел.

```
/* Заполнить файл некоторым количеством целых случайных чисел. */


/* Dev-C++ */
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
 FILE *f; intdat;
srand(time(0));
```

```
int n=rand()%30 + 1;
cout<< "File name? ";
char s[20];
cin.getline(s, 20);
 f=fopen(s, "wb");
for (inti=1; i<=n; i++)
{ dat = rand()%101 - 50;
cout<<dat<< " ";
fwrite(&dat, sizeof(int), 1, f);
 }
cout<<endl;
fclose(f);
system("PAUSE");
return EXIT_SUCCESS;
}
```

Пример 9. Найти сумму и количество целых чисел, записанных в бинарный файл.

```
/* Найти сумму и количество целых чисел, записанных в бинарный файл. */
/* Dev-C++ */
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
 FILE *f;
```

```
intdat, n=0, sum=0;
cout<< "File name? ";
char s[20];
cin.getline(s, 20);
 f=fopen(s, "rb");
while (fread(&dat, sizeof(int), 1, f))
      {n++;
cout<<dat<< " ";
sum+=dat;
      }
cout<<endl;
cout<< "sum: " << sum << "; number: " << n <<endl;
fclose(f);
system("PAUSE");
return EXIT_SUCCESS;
}
```

Заключение

Файлы позволяют пользователю считывать большие объемы данных непосредственно с диска, не вводя их с клавиатуры. Существуют два основных типа файлов: текстовые и двоичные.

Текстовыми называются файлы, состоящие из любых символов. Они организуются по строкам, каждая из которых заканчивается символом «конца строки». Конец самого файла обозначается символом «конца файла». При записи информации в текстовый файл, просмотреть который можно с помощью любого текстового редактора, все данные преобразуются к символьному типу и хранятся в символьном виде.В двоичных файлах информация считывается и записывается в виде блоков определенного размера, в которых могут храниться данные любого вида и структуры.В программе может быть открыт не один файл. В таком случае каждый файл должен быть связан со своим файловым указателем. Однако если программа сначала работает с одним файлом, потом закрывает его, то указатель можно использовать для открытия второго файла.

С файлом можно работать не как с последовательностью символов, а как с последовательностью байтов. В принципе, с нетекстовыми файлами работать по-другому не возможно. Однако так можно читать и писать и в текстовые файлы. Преимущество такого способа доступа к файлу заключается в скорости чтения-записи: за одно обращение можно считать/записать существенный блок информации.

Список использованной литературы:

1.Основы программирования на языке С++: Учебное пособие/Сост. С. М. Наместников. – Ульяновск: УлГТУ, 2007.

2.Подбельский *В. В.* Глава 8. Препроцессорные средства // Язык Си++ / рец. Дадаев Ю. Г.. — 4. — М.: Финансы и статистика, 2003. — С. 263-280. — 560 с.

Источники:

1.http://www.helloworld.ru/texts/comp/lang/visualc/vc2/9.htm

2. http://cppstudio.com/post/446/

3.http://kvodo.ru/urok-10-1-rabota-s-tekstovyimi-faylami-v-c.html

4. http://sernam.ru/c_51.php

5. http://programmersclub.ru/37/

6. cppstudio.com/post/446/