Лабораторная работа N 2

Файлы-потоки.

Использование файлового ввода-вывода языка C++

1. Введение

Файловый и консольный ввод-вывод тесно взаимосвязаны. Фактически файловый ввод-вывод поддерживается той же иерархией классов, что и консольный. Таким образом, то, что мы рассмотрели в первой работе применимо и для файлов. Однако есть некоторые отличия. В C++ файл открывается посредством связывания с потоком. Имеется три вида потоков ввода, вывода и ввода-вывода. Перед тем как открыть файл нужно сначала создать поток например объявив переменную типа **ifstream** (поток чтения) или **ofstream** (поток записи).

Целью лабораторной работы является получение практических навыков при работе с библиотекой файлового ввода-вывода языка C++. В работе рассматривается как форматируемый, так и неформатируемый двоичный ввод-вывод.

2. Теоретические сведения.
2.1 Форматированный файловый ввод-вывод.

Для ввода-вывода с использованием файлов имеются специальные классы, описанные в <fstream.h>:

- **ifstream** (производный от istream -чтение),
- **ofstream** (производный от ostream -запись),
- **fstream** (производный от ifstream и ofstream - чтение и запись)

Общими для этих классов являются следующие функции:

- **open** (открыть файл),
- **close** (закрыть файл),
- **good** (возвращает не нуль, если нет ошибок),
- **fail** (возвращает не нуль, если возникла ошибка),
- **eof** (конец файла),
- **!** (оператор для определения состояния ошибки).

При открытии файла режим работы с ним определяется вторым параметром функции **open**. Этот параметр может иметь следующие значения:

- **ios::in** (для ввода),
- **ios::out** (для вывода),
- **ios::ate** (перейти в конец файла),
- **ios::app**(для добавления),
- **ios::trunc** (очистить файл),
- **ios::nocreate** (не открывать, если файл не существует),
- **ios::noreplace** (не открывать, если файл существует),
- **ios::binary** (двоичный режим).

При необходимости изменить способ открытия или применения файла можно при создании файлового потока использовать два флага или более флага:
ios::app|ios::noreplace.

Для открытия файла одновременно на чтение и запись можно использовать объекты класса fstream:
fstream object(filename, ios::in|ios::app);

Перед завершением программа проверяет, находятся ли потоки в приемлемом состоянии. При завершении программы открытые файлы неявно закрываются. Для явного закрытия объектов файловых потоков применяется метод close():

object.close();

Для ввода/вывода в потоковые объекты можно применять методы **put(), get()**

Для позиционирования в файле используются функции

– **seekg** (при чтении);

– **seekp** (при записи),

для определения текущей позиции –

– **tellg** (при чтении) и

– **tellp** (при записи).

При вызове **seekg** () и **seekp()** с одним аргументом функции перемещают указатель на заданное место, а при вызове с двумя аргументами вычисляется относительная позиция от

– **ios::beg**, начала файла;

– **ios::cur** текущей позиции или;

– **ios::end** от конца файла.

Пример 1. Использования объектов fstream:

```
#include <fstream.h>
#include <iomanip.h>
void main ()
{fstream f;
int value;
//Открыть файл для чтения
 f.open ("data1.txt",ios::in);
 if (!f) //Ошибка?
{ cerr<<endl<<"Не могу открыть data1.txt для чтения!";
return;}
f.seekg (0,ios::beg); //Перейти в начало файла при чтении
 f >> value; // Чтение из файла
 f.close(); // Закрытие файла
//Открыть файл для записи и при этом очистить его
 f.open ("data2.txt",ios::out|ios::trunc);
 if (!f) //Ошибка?
 { cerr<<endl<<"Не могу открыть data2.txt для записи!";
return;}
 f.seekp (0,ios::beg); //Перейти в начало файла при записи
f << " Значение, прочитанное из файла data1.txt : " << value;
 f.close();
}
```

2.2 Неформатированный файловый ввод-вывод с использованием read, gcount и write

Неформатированный ввод-вывод выполняется с помощью функций-элементов **read** и **write**. Каждая из них вводит или выводит некоторое число байтов в символьный массив в памяти или из него. Эти байты не подвергаются какому-либо форматированию. Они просто вводятся или выводятся в качестве сырых байтов данных. Например, вызов

```
char buffer[ ] = "ПОЗДРАВЛЯЕМ С ДНЕМ РОЖДЕНИЯ";
cout.write(buffer, 12);
```

выводит первые 12 байтов символьного массива buffer (включая нулевые символы, которые могут быть выведены в cout и завершить операцию «). Поскольку символьная строка указывает на адрес своего первого символа, то вызов

```
cout.write("ABCDEFGYIJKLMNOPQRSTUVWXYZ", 10);
```

отобразит на экране первые 10 символов алфавита.

Функция-элемент **read** вводит в символьный массив указанное число символов. Если

считывается меньшее количество символов, то устанавливается флаг failbit. Позже мы увидим, каким образом определять, установлен ли флаг failbit (см. раздел 11.8).

Функция-элемент **gcount** сообщает о количестве символов, прочитанных последней операцией ввода.

Пример 2 показывает работу функций-элементов **read** и **gcount** класса **istream** и функции-элемента **write** класса **ostream**. Программа вводит 20 символов (из более длинной входной последовательности) в массив символов buffer с помощью функции-элемента **read**, определяет число введенных символов с помощью **gcount** и выводит символьный массив buffer с помощью **write**.

Пример 2. Неформатированный ввод-вывод с помощью функций-элементов **read**, **gcount** и **write**.

```
#include  <iostream.h>
const  int  SIZE  =  80;
main ( ) {
char buffer[SIZE] ;
cout  «  "Введите  предложение:"  «  endl;
cin.read(buffer,  20);
cout  «  endl  «  "Введенное  предложение:  "  «  endl;
cout . write (buffer,  cin . gcount ( ) ) ;
cout  «  endl;
return  0;
}
```

Результат работы

```
Введите предложение :
Использование функций-элементов read, write и gcount
Введенное предложение: Использование функци
```

В следующей программе для записи строки и числа типа **double** в файл **test** используется функция **write():**

Пример 3. Использование функции write

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
int main()
{
ofstream out("test", ios::out | ios::binary);
if(!out) {
cout « "Файл открыть невозможной";
return 1;
}
double num = 100.45;
char str[] = "Это проверка";
out.write((char *) &num, sizeof(double));
out.write(str, strlen(str));
out.closeQ;
return 0;
}
```

В следующей программе для считывания из файла, созданного в программе примера 3, используется функция **read():**

Пример 4. Использование функции read

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
ifstream in("test", ios::in | ios::binary);
if(!in) {
cout « "Файл открыть невозможной"; return 1;
}
double num;
charstr[80];
in.read((char *) &num, sizeof(double));
in.read(str, 15);
str[14]=”0”;
cout « num « " « str;
in.close();
return 0;
}
```

2.3 Контроль состояния ввода/вывода и обработка ошибок

В системе ввода/вывода C++ поддерживается информация о состоянии после каждой операции ввода/вывода. Текущее состояние потока ввода/вывода, которое хранится в объекте типа **iostate,** является перечислением, определенным в классе ios и содержащим следующие члены:

Название	Значение
goodbit	Ошибок нет
eofbit	Достигнут конец файла
failbit	Имеет место нефатальная ошибка
badbit	Имеет место фатальная ошибка

Имеются два способа получения информации о состоянии ввода/вывода. Во-первых, можно вызвать функцию **rdstateQ,** являющуюся членом класса **ios.** Прототип этой функции:

```
iostate rdstate();
```

Функция возвращает текущее состояние флагов ошибки. Функция **rdstate()** возвращает флаг **goodbit** при отсутствии какой бы то ни было ошибки. В противном случае она возвращает флаг ошибки.

Другим способом определения того, имела ли место ошибка, является использование одной или нескольких следующих функций - членов класса **ios:**

bool bad(); bool eof(); bool fail(); bool good();

Функция **eof()** уже обсуждалась.

Функция **bad()** возвращает истину, если установлен флаг **badbit.**

Функция **fail()** возвращает истину, если установлен флаг **failbit.**

Функция **good()** возвращает истину при отсутствии ошибок. В противном случае функции возвращают ложь.

После появления ошибки может возникнуть необходимость сбросить это состояние перед тем, как продолжить выполнение программы. Для этого используется функция **c!eaг(),** являющаяся членом класса **ios.** Ниже приведен прототип этой функции:

```
void clear(iostate флаги = ios::goodbit);
```

Если параметр ***флаги*** равен **goodbit** (значение по умолчанию), то сбрасываются флаги всех ошибок. В противном случае переменной ***флаги*** присваиваются значения тех флагов, которые вы хотите сбросить.

В следующей программе иллюстрируется выполнение функции **rdstateQ.** Программа выводит на экран содержимое текстового файла. При наличии ошибки функция сообщает об этом с помощью функции **checkstatus().**

Пример 5. Контроль состояния ввода/вывода

```
#include <iostream>
#include <fstream>
using namespace std;
void checkstatus(ifstream &in);
int main(int argc, char *argv[]) {
if(argc!=2) {
cout« "Содержимое: <имя_файла>\n"; return 1;
}
ifstream in(argv[l]);
if(!in) {
cout « "Файл открыть невозможной"; return 1;
}
char c; while(in.get(c)) {
cout« c;
checkstatus(in);
}
checkstatus(in); // контроль финального состояния
in.close();
return 0;
}
void checkstatus(ifstream &in)
{
ios::iostate i;
i = in.rdstate();
if(i & ios::eofbit) cout« "Достигнут EOF\n";
else if(i & ios::failbit) cout « "Нефатальная ошибка ввода/вывода\n";
else if(i & ios::badbit) cout « "Фатальная ошибка ввода/вывода\n";
}
```

Эта программа всегда будет выводить сообщение по крайней мере об одной ошибке. После окончания цикла **while** последний вызов функции **checkstatus**, как и ожидается, выдаст сообщение о достижении конца файла (символа EOF).

3. Дополнительная литература

1. Герберт Шилдт. Самоучитель C++: Пер. с англ. – 3-е изд. – СПб.: БХВ-Петербург, 2003 (глава 9)

2. Харви Дейтел, Пол Дейтел Как программировать на C++: пер. с англ. -М: ЗАО "Издательство Бином", 1999. (глава 14)

2. Бьярн Страуструп. Язык программирование Си++. – М.: Бином, 2005.(глава 10)

4. Порядок выполнения работы

1. Создайте новый проект в среде MS Visual C++.

2. Для полученного варианта задания, руководствуясь описанием настоящей лабораторной работы, напишите программы на языке C++. Откомпилируйте, отладьте и выполните свою программу.

3. Подготовьте отчет о выполнении лабораторной работы

Для успешной сдачи лабораторной работы необходимо:

– представить преподавателю работающую программу для указанного варианта задания;

– ответить на вопросы по тексту программы.

5. Порядок оформления отчета

Отчет о выполнении лабораторной работы должен содержать:

1) титульный лист;
2) вариант задания;
3) текст программы;
4) результаты работы программы.

6. Варианты заданий

Для допуска к экзамену достаточно выполнить любое одно задание. Для студентов, претендующих на освобождение от экзаменов необходимо выполнить оба задания.

Вариант 1

1. Написать программу копирования файлов. Чтение происходит блоками..
2. Написать программу вычисления количества символов пробела в файле. Обработать ошибки.

Вариант 2

1. Написать программу копирования текстового файла с добавлением двойного интервала между строками.
2. Написать программу вычисления количества символов в файле. Чтение происходит блоками. Обработать ошибки.

Вариант 3

1. Написать программу копирования файла в обратном порядке. Чтение происходит блоками.
2. Написать программу вычисления количества цифр в файле. Обработать ошибки.

Вариант 4

1. Написать программу копирования файла первая половина собственно файл, вторая половина он же в обратном порядке. Чтение происходит блоками.
2. Написать программу вычисления количества символов нижнего регистра в файле. Обработать ошибки.

Вариант 5

1. Написать программу копирования файла в двойном экземпляре. Чтение происходит блоками.
2. Написать программу вычисления количества символов верхнего регистра в файле. Обработать ошибки.

Вариант 6

1. Написать программу копирования файла в двойном экземпляре. Чтение происходит блоками.
2. Написать программу вычисления количества печатаемых символов в файле. Обработать ошибки.

Вариант 7

1. Написать программу копирования файла с удалением лишних пробелов.
2. Написать программу вычисления количества символов или цифр в файле. Чтение происходит блоками. Обработать ошибки.

Вариант 8

1. Написать программу копирования файла с удвоением пробелов. Чтение происходит блоками.
2. Написать программу вычисления количества символов пунктуации в файле. Чтение происходит блоками. Обработать ошибки.

Вариант 9

1. Написать программу записи в файл строк, введенных с клавиатуры. Записывать блоком.
2. Написать программу вычисления количества символов нижнего регистра в файле. Обработать ошибки.

Вариант 10

1. Написать программу копирования файла с заменой пробелов на символ |. Чтение происходит блоками.
2. Написать программу вычисления количества символов верхнего регистра в файле. Обработать ошибки.

Вариант 11

1. Написать программу записи заголовка в файл данных (*.dat). Запись блоками, но поэлементно.
2. Написать программу вычисления количества символа ':' в файле. Обработать ошибки.

Вариант 12

3. Написать программу копирования файла с удалением пробелов. Чтение происходит блоками.
4. Написать программу вычисления количества символов перевода строки в файле. Чтение происходит блоками. Обработать ошибки.

Вариант 13

3. Написать программу копирования файлов. Чтение происходит блоками.
4. Написать программу вычисления количества символов перевода строки в файле. Чтение происходит блоками. Обработать ошибки.

Вариант 14

3. Написать программу копирования текстового файла с добавлением двойного интервала между строками.

4. Написать программу вычисления количества символа ':' в файле. Обработать ошибки.

Вариант 15

3. Написать программу копирования файла в обратном порядке. Чтение происходит блоками.
4. Написать программу вычисления количества символов верхнего регистра в файле. Обработать ошибки.

Вариант 16

3. Написать программу копирования файла первая половина собственно файл, вторая половина он же в обратном порядке. Чтение происходит блоками.
4. Написать программу вычисления количества символов нижнего регистра в файле. Обработать ошибки.

Вариант 17

3. Написать программу копирования файла в двойном экземпляре. Чтение происходит блоками.

5. Написать программу вычисления количества символов пунктуации в файле. Чтение происходит блоками. Обработать ошибки.

Вариант 18

3. Написать программу копирования файла в двойном экземпляре. Чтение происходит блоками.
4. Написать программу вычисления количества символов или цифр в файле. Чтение происходит блоками. Обработать ошибки.

Вариант 19

3. Написать программу копирования файла с удалением лишних пробелов.
4. Написать программу вычисления количества печатаемых символов в файле. Обработать ошибки.

Вариант 20

4. Написать программу копирования файла с удвоением пробелов. Чтение происходит блоками.
5. Написать программу вычисления количества символов верхнего регистра в файле. Обработать ошибки.

Вариант 21

6. Написать программу записи в файл строк, введенных с клавиатуры. Записывать блоком.
7. Написать программу вычисления количества символов нижнего регистра в файле. Обработать ошибки.

Вариант 22

1. Написать программу копирования файла с заменой пробелов на символ |. Чтение происходит блоками.
2. Написать программу вычисления количества цифр в файле. Обработать ошибки.

Вариант 23

1. Написать программу записи заголовка в файл данных (*.dat). Запись блоками, но поэлементно.
2. Написать программу вычисления количества символов в файле. Чтение происходит блоками. Обработать ошибки.

Вариант 24

3. Написать программу копирования файла с удалением пробелов. Чтение происходит блоками.
4. Написать программу вычисления количества символов пробела в файле. Обработать ошибки.

Вариант 25

6. Написать программу копирования файла с заменой пробела на знак _. Чтение происходит блоками.
7. Написать программу вычисления количества непечатаемых символов в файле. Обработать ошибки.