Лекция 8.Организация ввода-вывода в C++

8.1. Форматированный ввод-вывод в C++

Для управления вводом-выводом в C++ используются:

- флаги форматного ввода-вывода;
- манипуляторы форматирования.

8.1.1. Использование флагов форматного ввода-вывода

Флаги позволяют включить или выключить один из параметров вывода на экран. Для установки флага вывода используется следующая конструкция языка C++.

cout.setf(ios::flag)

Для снятия флага используется конструкция

cout.unsetf(ios::flag)

здесь flag – имя конкретного флага.

Если при выводе необходимо установить несколько флагов, то можно воспользоваться логической операцией «или» (|). В этом случае конструкция языка C++ будет такой.

cout.setf(ios::flag1|ios::flag2| ios::flag3)

В данном случае flag1, flag2, flag3 – имена устанавливаемых флагов вывода.

Основные флаги форматирования с примерами приведены в таблице 8.1.

8.1.2. Использование флагов форматного ввода-вывода

Основные манипуляторы с примерами приведены в табл. 8.2

Еще один способ управления шириной поля вывода с помощью операторов:

- cout.width(n) – устанавливает ширину поля вывода – n позиций;
- cout.presicion(m) - определяет m цифр в дробной части числа.

Таблица 8.1. Флаги форматирования

Флаг	Описание	Пример использования	Результат

right	Выравнивание по правой границе	`int r=-25;`
		`cout.setf(ios::right);`
		`cout.width(15);`
		`cout<<"r="<<r<<endl;`
			`r=-25`

left	Выравнивание по левой границе (по умолчанию)	`double r=-25.45;`
		`cout.setf(ios::left);`
		`cout.width(50);`
		`cout<<"r="<<r<<endl;`
			`r=-25.45`

boolalpha	Вывод логических величин в текстовом виде (true, false)	`bool a=true;`
		`cout<<a<<endl;`
		`cout.setf(ios::boolalpha);`
		`cout<<a<<endl;`
			`1`
			`true`

dec	Вывод величин в десятичной системе счисления (по умолчанию)	`int r=-25;` `cout<<"r="<<r<<endl;`	`r=-25`

oct	Вывод величин в восьмеричной системе счисления (для этого надо снять флаг вывод в десятичной)	`int p=23;`
		`cout.unsetf(ios::dec);`
		`cout.setf(ios::oct);`
		`cout<<"p="<<p<<endl;`
			`p=27`

hex	Вывод величин в шестнадцатеричной системе счисления (для этого надо снять флаг вывод в	`int p=23;`
		`cout.unsetf(ios::dec);`
		`cout.setf(ios::hex);`
		`cout<<"p="<<p<<endl;`
			`p=17`

Флаг	Описание	Пример использования	Результат
	десятичной)		

showbase	Выводить индикатор основания системы счисления	`int p=23;`
		`cout.unsetf(ios::dec);`
		`cout.setf(ios::hex|ios::showbase);`
		`cout<<"p="<<p<<endl;`
			`p=0x17`

uppercase	Использовать прописные буквы в шестнадцатеричных цифрах	`int p=29;`
		`cout.unsetf(ios::dec);`
		`cout.setf(ios::hex|ios::uppercase);`
		`cout<<"p="<<p<<endl;`
			`p=1D`

showpos	Выводить знак «+» для положительных чисел	`int p=29;`
		`cout.setf(ios::showpos);`
		`cout<<"p="<<p<<endl;`
			`p=+29`

scientific	Экспоненциальная форма вывода вещественных чисел	`double p=146.673;`
		`cout.setf(ios::scientific);`
		`cout<<"p="<<p<<endl;`
			`p=1.466730e+002`

fixed	Фиксированная форма вывода вещественных чисел (по умолчанию)	`double p=146.673;`
		`cout.setf(ios::fixed);`
		`cout<<"p="<<p<<endl;`
			`p=146.673`

Таблица 8.2. Манипуляторы форматирования

Манипулятор	Описание	Пример использования	Результат примера

setw(n)	Определяет ширину поля вывода в n символов	`int r=253;`
		`cout.setf(ios::fixed);`
		`cout<<"r="<<setw(10)<<r<<endl;`
			`r=       253`

setprecisi-on(n)	Определяет количество цифр (n-1) в дробной части числа	`double h=1234.6578;`
		`cout.setf(ios::fixed);`
		`cout<<"h="<<setw(15)<<`
		`setprecision(3)<<h<<endl;`
			`h=        1234.658`

dec	Перевод числа в десятичную систему (по умолчанию)	`int r=0253;` `cout<<"r="<<dec<<r<<endl;`	`r=171`
oct	Перевод числа в восьмеричную систему	`int r=253;` `cout<<"r="<<oct<<r<<endl;`	`r=375`
hex	Перевод числа в шестнадцати-ричную систему	`int r=253;` `cout<<"r="<<hex<<r<<endl`	`p=fd`

right	Выравнивание по правой границе	`int r=-25;`
		`cout.width(15);`
		`cout<<"r="<<setw(15)<<right<<r<<endl;`
			`r          =-25`

left	Выравнивание по левой границе (по умолчанию)	`int r=-25;`
		`cout.width(15);`
		`cout<<"r="<<setw(15)<<left<<r<<endl;`[1]
			`r=-25`

boolalpha	Вывод логических величин в текстовом виде (true, false)	`bool a=true;` `cout<<boolalpha<<a<<endl;`	`true`

1 Еще один пример приведен при использовании манипулятора setfill

Манипулятор	Описание	Пример использования	Результат примера
noboolalpha	Вывод логических величин в числовом виде (1, 0)	`bool a=true;` `cout<<noboolalpha<<a<<endl;`	`1`
showpos	Выводить знак «+» для положительных чисел	`int p=29;` `cout<<"p="<<showpos<<p<<endl;`	`p=+29`
noshowpos	Не выводить знак «+» для положительных чисел	`int p=29;` `cout<<"p="<<noshowpos<<p<<endl;`	`p=29`
uppercase	Использовать прописные буквы в шестнадцатеричных цифрах	`int p=253;` `cout<<"p="<<hex<<uppercase<<p<<endl;`	`p=FD`

nouppercase	Использовать строчные буквы в шестнадцатеричных цифрах	`int p=253;`
		`cout<<"p="<<hex<<`
		`nouppercase<<p<<endl;`
			`p=fd`

showbase	Выводить индикатор основания системы счисления	`int p=253;`
		`cout<<"p="<<hex<<uppercase`
		`<<showbase<<p<<endl;`
			`p=0XFD`

noshowbase	Не выводить индикатор основания системы счисления	`int p=253;`
		`cout<<"p="<<hex<<uppercase`
		`<<noshowbase<<p<<endl;`
			`p=FD`

showpos	Выводить знак «+» для положительных чисел	`int p=28;`
		`cout.setf(ios::showpos);`
		`cout<<"p="<<p<<endl;`
			`p=+29`

setfill(c)	Установить символ с как заполнитель	`cout<<"x="<<right<<setw(10`
		`<<setprecision(4)<<`
		`setfill('!')<<`
			`x=!!!!0.1429`

Манипулятор	Описание	Пример использования	Результат примера
		`(float) 1/7<<endl;`	
		`cout<<"x="<<left<<setw(10)<<setprecision(4)<<setfill('!')<<(float) 1/7<<endl;`	`x=0.1429!!!!`
scientific	Экспоненциальная форма вывода вещественных чисел	`double p=146.673;` `cout<<"p="<<scientific<<p<<endl;`	`p=1.466730e+002`
fixed	Фиксированная форма вывода вещественных чисел (по умолчанию)	`cout<<"p="<<fixed<<p<<endl;`	`p=146.673`

8.2. Операции с файлами в C++

Файлы делятся на:

1. Текстовые
2. Двоичными

8.2.1. Операции с текстовыми файлами

Текстовый файл — файл, в котором каждый символ из используемого набора символов хранится в виде одного байта (кода, соответствующего символу). Текстовые файлы разбиваются на несколько строк с помощью специального символа «конец строки». Текстовый файл заканчивается специальным символом «конец строки».

В C++ операции с файлами можно осуществлять с помощью файловых указателей и с помощью потоков.

Работа с текстовыми файлами с помощью файловых указателей

Для записи данных в файл нужно выполнить

1. Описать указатель на файл **FILE *filename**;
2. Открыть файл (функция **fopen**)

Описание функции

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

filename – строка, в которой хранится полное имя открываемого файла.

mode – строка, которая определяет режим работы с файлом; возможны следующие значения:

- «**r**» – открываем текстовый файл в режиме чтения;
- «**w**» – создаем текстовый файл;
- «**a**» – создаем или открываем текстовый файл для дозаписи в конец файла;
- «**r+**» – открываем текстовый файл в режиме чтения и записи;
- «**w+**» – открываем текстовый файл для исправления, старое содержимое выбрасывается;
- «**a+**» текстовый файл открывается или создается для исправления существующей информации и добавления новой в конец файла;

Функция возвращает указатель на файловую переменную или NULL при неудачном открытии файла.

3. Записать данных в файл (функция **fprintf**)

Функция **fprintf** аналогична функции **printf**, единственным отличием является первый параметр – указатель на файл. С помощью этой функции вывод осуществляется не на экран, а в файл.

4. Закрыть файл (функция **fclose**)

int fclose(FILE *filename);

Возвращает 0 при успешном закрытии файла и NULL в противном случае.

Кроме этих функций для работы с файлами есть еще две:

int remove(const char *filename);

Эта функция удаляет с диска файл, указатель на который хранится в файловой переменной **filename**. Функция возвращает ненулевое значение, если файл не удалось удалить.

int rename(const char *oldfilename, const char *newfilename);

Функция переименовывает файл; первый параметр – старое имя файла, второй – новое. Возвращает 0 при неудачном завершении программы.

Для чтения данных из файла нужно выполнить:

1. Описать указатель на файл **FILE *filename**;
2. Открыть файл (функция **fopen**)
3. Считать данные из файла (функция **fscanf**)

Функция **fscanf** аналогична функции **scanf**, единственным отличием является первый параметр – указатель на файл. С помощью этой функции вывод осуществляется не на экран, а в файл. В случае необходимости при чтении надо контролировать возможность чтения очередного компонента с помощью функции **feof**ю[2].

4. Закрыть файл (функция **fclose**)

ЗАДАЧА 8.1. В файле **abc.txt** хранятся матрицы **A(N,M)** и **B(M,K)**. Пусть структура файла следующая: в первой строке хранятся числа **n** и **m**, затем построчно матрица **A**, за тем строка, в которой хранится **m** и **k**. Затем – построчно матрица **B**. Найти матрицу **C=A·B** и записать ее в файл **rez.txt**.

```
#include <stdio.h>
#include <alloc.h>
int main()
{
int i,j,n,m,l,k;
float *b,*c,*a,s,temp;
FILE *f;
f=fopen("abc.txt","r");
fscanf(f,"%d%d",&n,&m);
a=(float *)calloc(n*m,sizeof(float));
for(i=0;i<n;i++)
    for(j=0;j<m;j++)
    {   fscanf(f,"%g",&temp);     *(a+i*m+j)=temp;      }
fscanf(f,"%d%d",&m,&l);
b=(float *)calloc(m*l,sizeof(float));
c=(float *)calloc(n*l,sizeof(float));
for(i=0;i<m;i++)
    for(j=0;j<l;j++)
    {   fscanf(f,"%g",&temp);     *(b+i*l+j)=temp;      }
fclose(f);
f=fopen("rez.txt","w");
for(i=0;i<n;i++)
    for(j=0;j<l;*(c+i*l+j)=s,j++)
```

2 Подробно функция описана в разделе, посвященном двоичным файлам.

```
        for(s=0,k=0;k<m;k++)
            s+=*(a+i*m+k)**(b+k*l+j);
fprintf(f,"Matrica C\n");
for(i=0;i<n;fprintf(f,"\n"),i++)
    for(j=0;j<l;j++)
        fprintf(f,"%g\t",*(c+i*l+j));
fclose(f);
free(a);     free(b);          free(c);
}
```

Работа с текстовыми файлами с помощью файловых потоков

Для работы с файлами используются специальные типы данные, называемые потоками. Поток ifstream служит для работы с файлами в режиме чтения. Поток ofstream служит для работы с файлами в режиме записи. Для работы с файлами в режиме, как чтения, так и записи служит поток fstream.

В программах на C++ при работе с текстовыми файлами необходимо подключать библиотеки iostream и fstream.

Для того, чтобы записывать данные в текстовый файл необходимо:

1. Описать переменную типа ofstream
2. Отрыть файл с помощью функции open.
3. Вывести информацию в файл с помощью cout.
4. Обязательно закрыть файл.

Для того, чтобы считывать данные из текстового файл необходимо:

1. Описать переменную типа ifstream
2. Отрыть файл с помощью функции open.
3. Считать информацию из файла с помощью cin, при считывании каждой порции данных необходимо проверять, что чтение возможно.
4. Закрыть файл.

Запись информации в текстовый файл

Для того, чтобы начать работать с текстовым файлом необходимо описать переменную типа ofstream. Например, с помощью оператора

ofstream F;

будет создана переменная F для записи информации в файл.

На следующем этапе файл необходимо открыть для записи. В общем случае оператор открытия файла будет иметь вид:

F.open("file", mode);

Здесь F – переменная, описанная как ofstream,
file – полное имя файла на диске, например, **D:\\STUDENT\\abc.txt**.
mode – режим работы с открываемым файлом:

- ios::in – открыть файл в режиме чтения данных, этот режим является режимом по умолчанию для потоков ifstream;
- ios::out – открыть файл в режиме записи данных, этот режим является режимом по умолчанию для потоков ofstream;
- ios::app – открыть файл в режиме записи данных в конец файла;
- ios::ate – передвинуться в конец уже открытого файла;

- ios::trunc – очистить файл, это же происходит в режиме ios::out;
- ios::nocreate – не выполнять операцию открытия файл, если он не существует[3];
- ios::noreplace – не открывать существующий файл.
- ios::binary – открыть двоичный файл

Параметр mode может отсутствовать, в этом случае файл открывается в режиме по умолчанию для данного потока

ios::in – для потоков ifstream,
ios::out – для потоков ofstream.

После удачного открытия файла (в любом режиме) в переменной F будет храниться true, в противном случае false. Это позволит проверять корректность операции открытия файла.

Открыть файл в режиме записи можно одним из следующих способов:

```
//Первый способ.
ofstream F;
F.open("D:\\STUDENT\\abc.txt", ios::out);
//Второй способ, режим ios::out
// является режимом по умолчанию для потока  ofstream.
ofstream F;
F.open("D:\\STUDENT\\abc.txt");
//Третий способ объединяет
//описание переменной типа поток
//и открытие файла в одном операторе.
ofstream F("D:\\STUDENT\\abc.txt", ios::out);
```

После открытия файла в режиме записи, будет создан пустой файл, в который можно будет записывать информацию. Если Вы хотите открыть существующий файл, то в качестве режима следует использовать значение ios::app.

После открытия файла в режиме записи, в него можно писать точно также, как и на экран, только вместо стандартного устройства вывода cout необходимо указать имя открытого для записи файла.

Например, для записи в поток F переменной a, оператор вывода будет иметь вид:

```
F<<a;
```

Для последовательного вывода в поток G переменных b, c и d оператор вывода станет таким:

```
G<<b<<c<<d;
```

В качестве примера рассмотрим следующую задачу.

ЗАДАЧА 8.2. Создать текстовый файл **D:\abc.txt** и записать туда n вещественных чисел.

```
#include <iostream>
#include <fstream>
#include <iomanip>
```

3При открытии файла в режиме ios::in происходит как раз обратное, если файл не существует, он создается

```
using namespace std;
int main()
{
    int i, n;
    double a;
//Описывает поток для записи данных в файл.
    ofstream f;
//Открываем файл в режиме записи,
// режим ios::out устанавливается по умолчанию.
    f.open("D:\\abc.txt");
    cout<<"n=";
    cin>>n;
    for(i=0;i<n;i++)
    {
        cout<<"a=";
//Ввод очередного числа.
        cin>>a;
//Запись в файл очередного числа и символа табуляции.
        f<<a<<"\t";
    }
    f.close();
    return 0;
}
```

Чтение информации из текстового файла

Для того чтобы прочитать информацию из текстового файла необходимо описать переменную типа **ifstream**. После этого необходимо открыть для чтения с помощью оператора open. Если переменную назвать F, то первые два оператора будут такими:

```
ifstream F;
F.open("D:\\STUDENT\\abc.txt", ios::in);
```

После открытия файла в режиме чтения, из него можно считывать информацию точно так же, как и клавиатуры, только вместо стандартного устройства ввода cin необходимо указать имя открытого для чтения файла.

Например, для ввода из потока F в переменную a, оператор ввода будет иметь вид:

```
F>>a;
```

Для последовательного ввода из потока G в переменные b, c и d оператор ввода станет таким:

```
G>>b>>c>>d;
```

Два числа в текстовом файле считаются разделенными, если между ними есть хотя бы один из символов: провел, табуляция, символ конца строки.

Хорошо если программисту заранее известно, сколько и каких значений хранится в текстовом файле. Однако часто просто известен тип значений, хранящихся в файле, при этом количество значений в файле может быть

различным. При решении подобной проблемы необходимо считывать значения из файла по одному, а перед каждым считыванием проверять достигнут ли конец файла. Для проверки достигнут или нет конец файла, служит функция

```
F.eof()
```

Здесь F – имя потока, функция возвращает логическое значение: true – если достигнут конец файла, если не достигнут функция возвращает значение false.

Цикл для чтения содержимого всего файла можно записать так:

```
while (!F.eof())
{
F>>a;
обработка значения переменной a
}
```

Рассмотрим следующую задачу.

ЗАДАЧА 8.3. В текстовом файле **D:\abc.txt** хранятся вещественные числа, вывести их на экран и вычислить их количество

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <iomanip>
using namespace std;
int main()
{
    ifstream f;
    float a;
    int n=0;
//Открываем файл в режиме чтения.
    f.open("D:\\abc.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 f в переменную a.
        f>>a;
//Вывод значения переменной a на экран.
        cout<<a<<"\t";
//Увеличение количества считанных чисел.
        n++;
    }
//Закрытие файла.
    f.close();
```

```
//Вывод на экран количества считанных чисел.
    cout<<"n="<<n<<endl;
    }
//Если открытие файла прошло некорректно, то вывод
// сообщение, об отсутствии такого файла.
    else cout<<"File not found"<<endl;
    return 0;
}
```

Существует возможность открывать файл с данными таким образом, чтобы в него можно было дописывать информацию. Рассмотрим эту возможность на примере решения следующей задачи.

ЗАДАЧА 8.4. В файле **D:\abc.txt** (см. рис. 8.1) хранится массив вещественных чисел, дописать в файл этот же массив, упорядочив его по возрастанию.

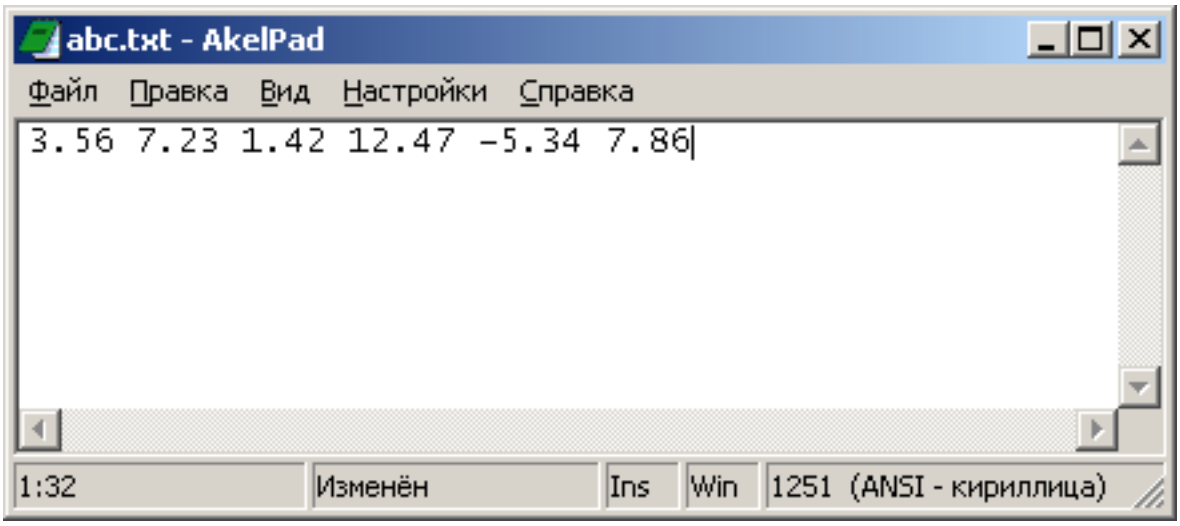

Рис. 8.1. Содержимое файла **D:\abc.txt**

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <iomanip>
using namespace std;
int main()
{
//Поток для чтения.
    ifstream f;
//Поток для записи.
    ofstream g;
    float *a,b;
    a=new float[100];
    int i,j,n=0;
//Открываем файл в режиме чтения.
    f.open("D:\\abc.txt",ios::in);
//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 очередной
// элемент массива a.
        f>>a[n];
//Вывод элемента массива a на экран.
        cout<<a[n]<<"\t";
//Увеличение количества считанных чисел.
        n++;
    }
//Вывод на экран количества считанных чисел.
    cout<<"n="<<n<<endl;
//Упорядочение массива
    for(i=0;i<n-1;i++)
        for(j=0;j<n-i-1;j++)
            if (a[j]>a[j+1])
            {
                b=a[j];
                a[j]=a[j+1];
                a[j+1]=b;
            }
//Закрываем поток для чтения.
    f.close();
//Открываем поток в режиме дозаписи.
    g.open("D:\\abc.txt",ios::app);
//Запись в файл символа табуляции для разделения
//последнего символа исходного файла и
// первого элемента дозаписываемого в массив.
    g<<"\t";
// Запись в файл элемента массива и символа табуляции.
    for(i=0;i<n;i++)
        g<<a[i]<< "\t";
//Закрытие файла
    g.close();
    }
//Если открытие файла прошло некорректно, то вывод
// сообщения об отсутствии такого файла.
    else cout<<
    "File not found"<<endl;
```

```
    delete [] a;
    return 0;}
```

При запуске программы на экране в консольном режиме появится окно, подобное представленному на рис. 8.2. В результате работы программы изменится содержимое файла **D:\abc.txt** (см. рис. 8.3).

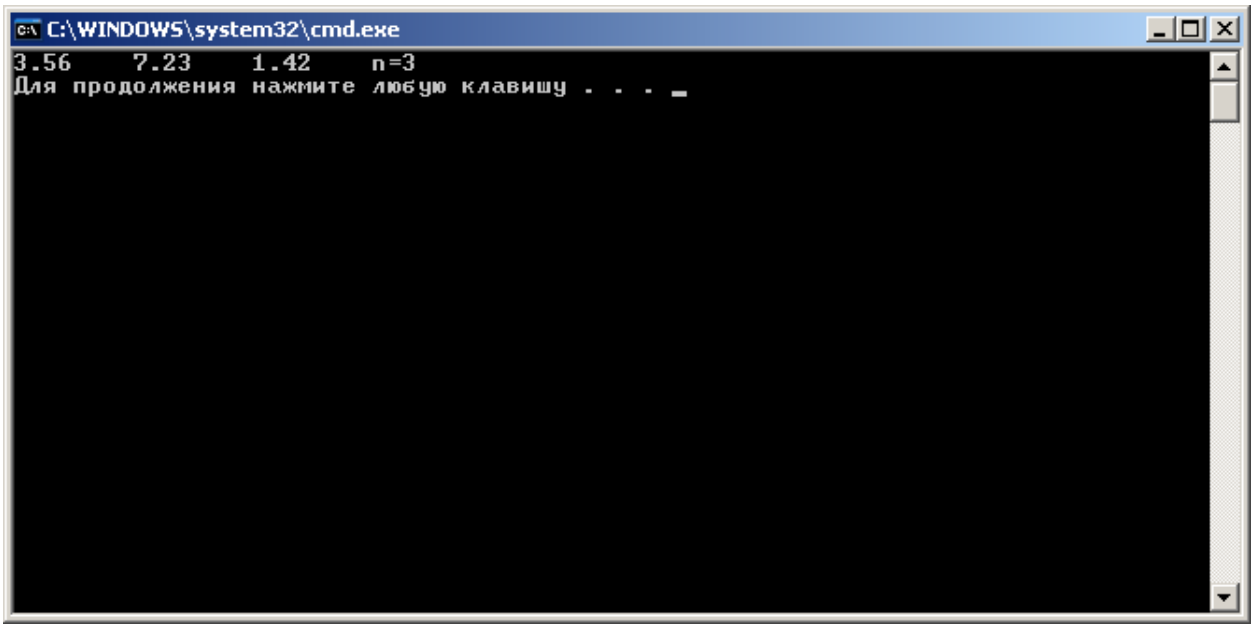

Рис.8.2. Результаты работы программы задачи 8.4

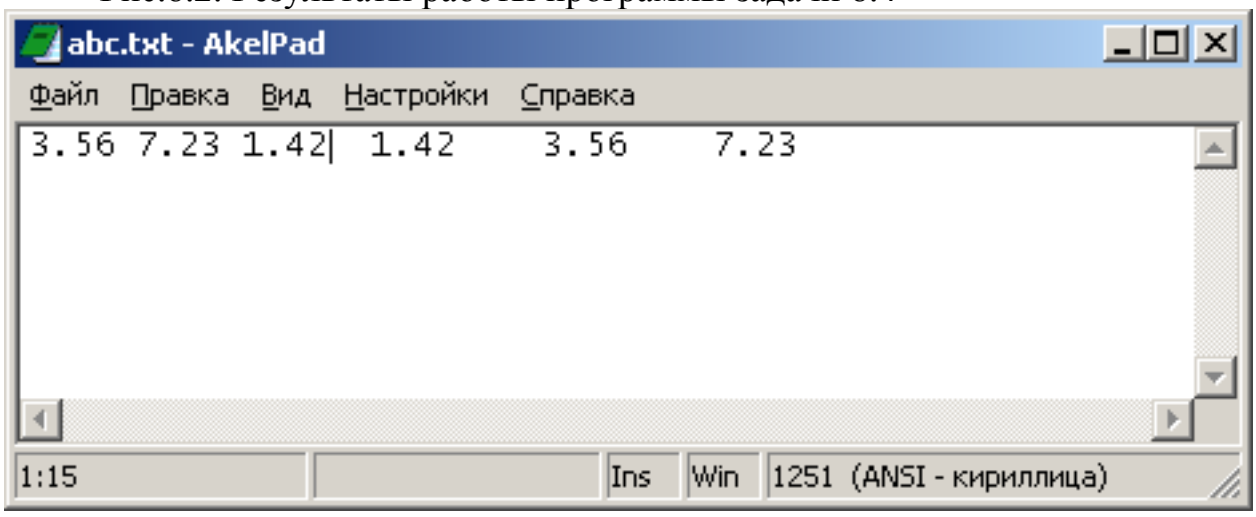

Рис.8.3. Содержимое файла **D:\abc.txt** после запуска программы

Но если повторно запустить программу уже над обновленным файлом **D:\abc.txt**, то на экране можно будет увидеть следующее (см. рис. 8.4) и в файл **D:\abc.txt** станет таким (см. рис. 8.5).

Дело в том, что при первом запуске программы в файл **D:\abc.txt** после последнего элемента массива был записан символ табуляции. Таким образом, в конце файла после последнего числа находится символ табуляции. При повторном чтении информации из файла происходит следующее: программа нормально считывает последнее число, записывает его в массив. Происходит возврат к началу цикла while, программа проверяет, достигнут ли конец файла. А так как после числа есть символ табуляции (это мог быть и символ пробел, перехода на

новую строку или их комбинация), то конец файла не достигнут, происходит вход в цикл. После этого оставшиеся до конца файла символы считываются и интерпретируются, как вещественное число типа float. В результате этого получается дополнительное ненужное число. На рис. 8.4-8.5 – это значение -4.31602E+008.

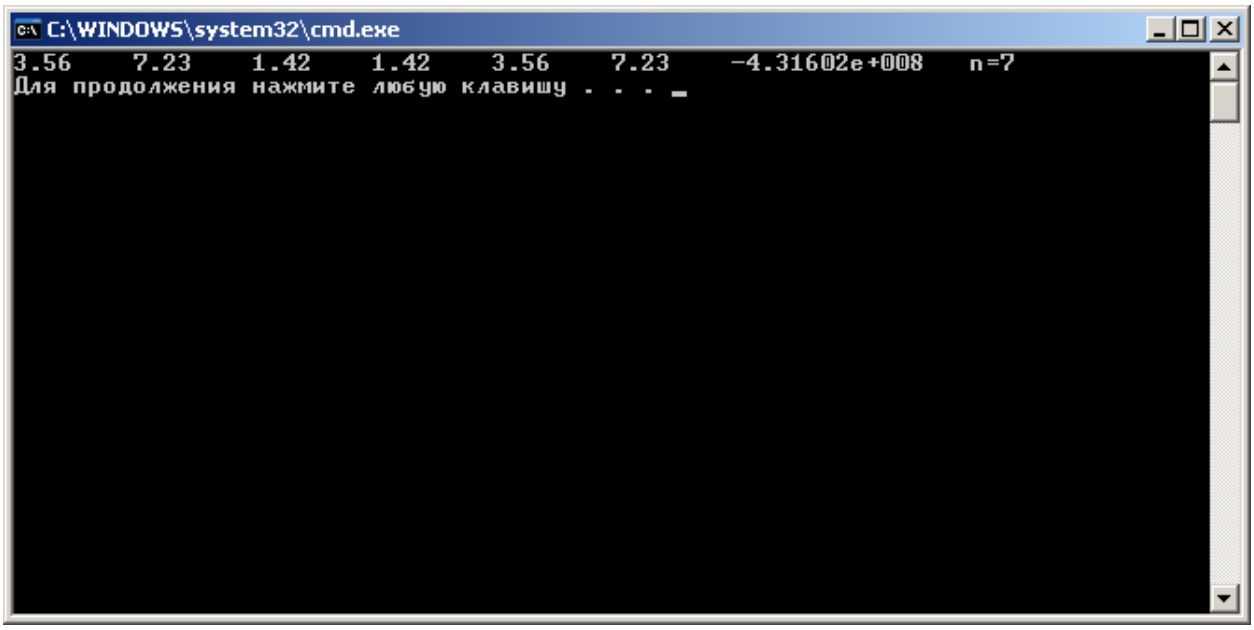

Рис.8.4. Результаты работы программы после повторного запуска

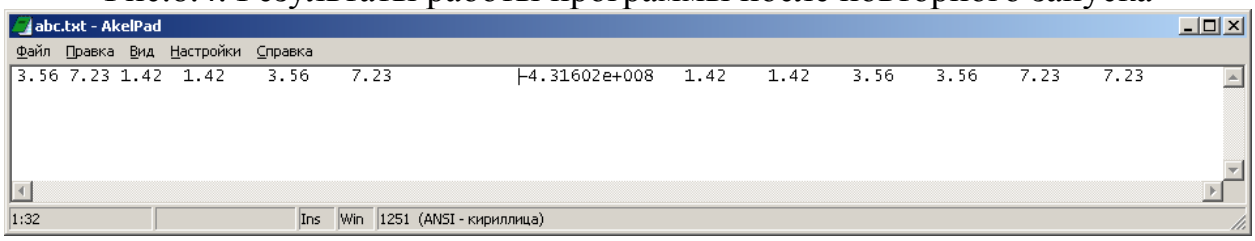

Рис.8.5. Содержимое файла **D:\abc.txt** после повторного запуска программы

В этом причина неправильной работы программы при повторном запуске. Причину нашли, но как исправить работу программы? В этом конкретном случае это не сложно, надо добиться того, чтобы программа не записывала в файл символ табуляции после последнего числа. Для этого изменим последний цикл for следующим образом.

```
for(i=0;i<n;i++)
     if (i<n-1) g<<a[i]<<"\t";
     elseg<<a[i];
```

После исправлений программа будет работать правильно (рис.8.6, 8.7).

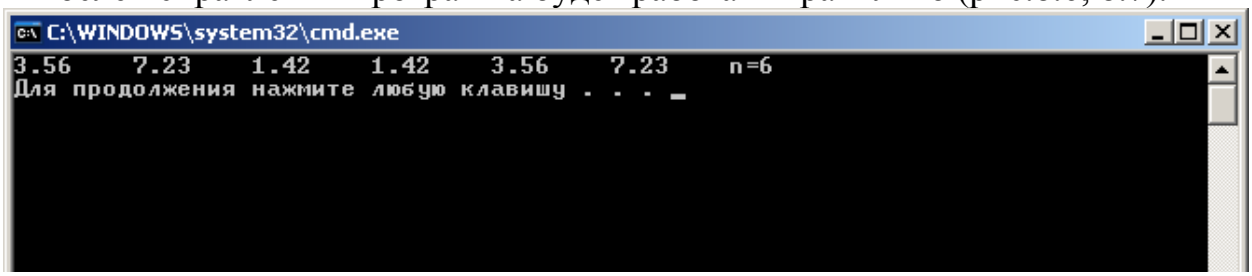

Рис. 8.6. Результат повторного запуска исправленной программы для решения задачи 8.3

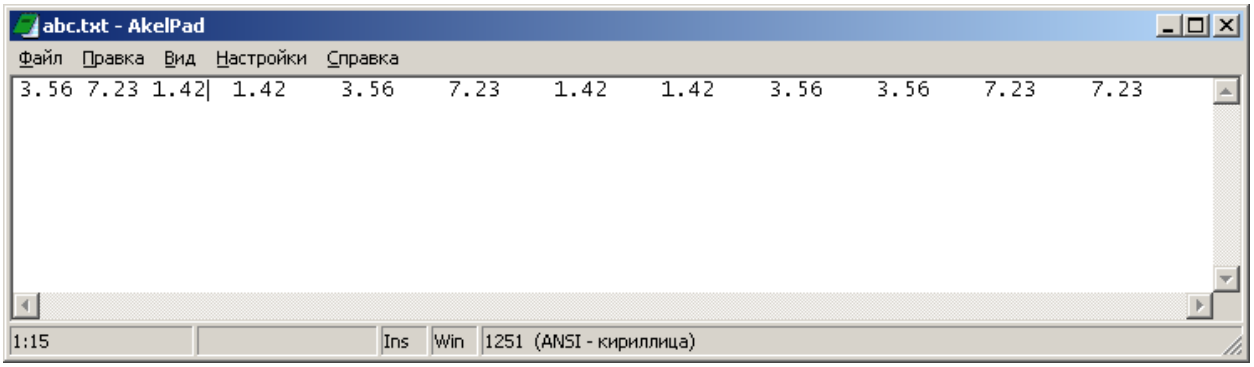

Рис. 8.7 Содержимое файла **D:\abc.txt** после повторного запуска исправленной программы

Внимание!!! Если в текстовом файле хранятся числовые значения, то после последнего числа не должно быть пробелов символов табуляции и конца строки.

Пусть есть файл **D:\abc4.txt**, в котором в первой строке хранится количество значений, а далее сами значения (см. рис. 8.8).

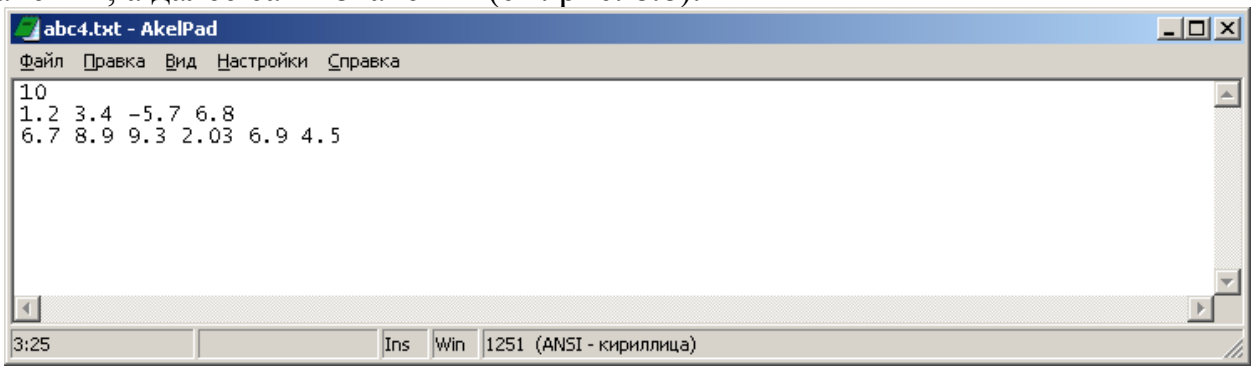

Рис. 8.8. Содержимое файла **D:\abc4.txt**

```
#include <iostream>
#include <fstream.h>
using namespace std;
int main()
{
//Поток для чтения данных
     ifstream f;
     int i, n;
     float *a;
//Открываем файл
     f.open("D:\\abc4.txt");
//Считываем из файла в переменную
// n количество вещественных чисел
     f>>n;
//Выделяем память для массива вещественных чисел
     a=new float[n];
//В цикле последовательно считываем значения
     for(i=0;i<n;i++)
     {
```

```
//Считывание значение из файла в массив
        f>>a[i];
//Вывод элемента массива на экран
        cout<<a[i]<<"\t";
    }
//Закрываем файл
    f.close();
//Освобождаем память
    delete []a;
    return 0;
}
```

8.2.2. Обработка двоичных файлов

При записи в двоичный файл символы и числа записываются в виде последовательности байт (в своем внутреннем двоичном представлении в памяти компьютера).

Порядок работы с двоичными и тестовыми файлами аналогичен.

Для того, чтобы записать данные двоичный файл необходимо:

1. Описать переменную типа **FILE*** с помощью оператора

FILE *filename;

filename – имя переменной, где будет храниться указатель на файл.

2. Отрыть файл с помощью функции **fopen.**
3. Записать информацию в файл с помощью функции **fwrite.**
4. Закрыть файл с помощью функции **fclose.**

Для того, чтобы считывать данные из двоичного файл необходимо:

1. Описать переменную типа **FILE ***
2. Отрыть файл с помощью функции **fopen**.
3. Считать необходимую информацию из файла с помощью функции **fread,** при считывании информации следить за тем, достигнут ли конец файла.
4. Закрыть файл с помощью функции **fclose**

Для открытия файла предназначена функция fopen.

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

filename – строка, в которой хранится полное имя открываемого файла, **mode –** строка, которая определяет режим работы с файлом; возможны следующие значения:

- «**rb**» – открываем двоичный файл в режиме чтения;
- «**wb**» – создаем двоичный файл для записи, если файл 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 существующей информации и добавления новой в конец файла.

Функция возвращает указатель на файловую переменную или NULL при неудачном открытии файла.

После открытия файла, указатель файла указывает на 0-й байт файла, а по мере чтения или записи смещается на считанное (записанное) количество байт. Текущее значение указателя файла – номер байта, начиная с которого будет происходить операция чтения или записи.

Для закрытия файла предназначена функция fclose

int fclose(FILE *filename);

Возвращает 0 при успешном закрытии файла и NULL в противном случае.
Функция remove предназначена для удаления файлов.

int remove(const char *filename);

Эта функция удаляет с диска файл с именем filename. Удаляемый файл должен быть закрыт. Функция возвращает ненулевое значение, если файл не удалось удалить.

Для переименования файлов предназначена функция rename.

int rename(const char *oldfilename, const char *newfilename);

Функция переименовывает файл; первый параметр – старое имя файла, второй – новое. Возвращает 0 при удачном завершении программы.

Чтение из двоичного файла осуществляется с помощью функций 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 указатель файла 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 файла смещается на n*size байт.

Для контроля достижения конца файла есть функция feof.

int feof(FILE * filename);

функция возвращает ненулевое значение, если достигнут конец файла.

Рассмотрим использование двоичных файлов на примере решения двух стандартных задач.

ЗАДАЧА 8.5. Создать двоичный файл **E:\STUDENT\abc.dat**, куда записать целое число n и n вещественных чисел.

```
#include <iostream>
using namespace std;
int main()
{
//Описываем файловую переменную.
    FILE *f;
    int i, n;
    double a;
//Создаем двоичный файл в режиме записи.
    f=fopen("E:\\STUDENT\\abc.dat", "wb");
```

```
//Ввод числа n.
    cout<<"n=";      cin>>n;
// Цикл для ввода n вещественных чисел.
    for(i=0;i<n;i++)
    {
//Ввод очередного вещественного числа.
        cout<<"a=";      cin>>a;
//Запись вещественного числа в двоичный файл.
fwrite(&a,sizeof(double),1,f);
    }
//Закрываем файл.
    fclose(f);
    return 0;
}
```

ЗАДАЧА 8.6. Вывести на экран содержимое созданного в предыдущей задаче двоичного файла **E:\STUDENT\abc.dat**.

```
#include <iostream>
using namespace std;
int main()
{
//Создаем двоичный файл в режиме записи.
    FILE *f;
    int i,n;
    double *a;
//Отрываем существующий двоичный файл в режиме чтения.
    f=fopen("E:\\STUDENT\\abc.dat", "rb");
//Считываем из файл одно целое число в переменную n.
    fread(&n,sizeof(int),1,f);
//Вывод n на экран.
    cout<<"n="<<n<<"\n";
// Выделение памяти для массива из n чисел.
    a=new double[n];
//Чтение n вещественных чисел из файла в массив a.
    fread(a,sizeof(double),n,f);
//Вывод массива на экран.
    for(i=0;i<n;i++)
        cout<<a[i]<<"\t";
    cout<<endl;
//Закрытие файла.
    fclose(f);
    return 0;
}
```

Двоичный файл – последовательная структура данных, после открытия файла доступен первый байт. Можно последовательно считывать или записывать данные

из файла. Допустим, необходимо считать пятнадцатое, а затем первое число, хранящееся в файле. С помощью последовательного доступа это можно сделать следующим образом:

```
FILE *f;
int i,n;
double a;
f=fopen("file.dat","rb");
for(i=0;i<15;i++
fread(&a,sizeof(double),1,f);
fclose(f);
f=fopen("file.dat","rb");
fread(&a,sizeof(double),1,f);
fclose(f);
```

Для произвольного перемещения внутри файла служит функция fseek.

int fseek(FILE *F, long int offset, int origin);

Функция устанавливает указатель текущей позиции файла F, в соответствии со значениями начала отсчета origin и смешения offset. Параметр offset равен количеству байтов, на которые будет смешен указатель файла относительно начала отсчета, заданного параметром origin. В качестве значения для параметра origin должен быть взят одно из следующих значений, определенных в заголовке stdio.h.

- SEEK_SET – отсчет смещения offset вести с начала файла;
- SEEK_CUR – отсчет смещения offset вести с текущей позиции файла;
- SEEK_END – отсчет смещения offset вести с конца файла.

Функция возвращает нулевое значение при успешном выполнении операции, ненулевое – при возникновении сбоя при выполнении смещения.

Функция fseek фактически реализует прямой доступ к любому значению в файле. Необходимо только знать месторасположение (номер байта) значения в файле. Рассмотрим использование прямого доступа в двоичных файлах на примере решения следующей задачи.

ЗАДАЧА 8.7. В созданном в задаче 8.4 двоичном файле **E:\STUDENT\abc.dat**, поменять местами наибольшее и наименьшее из вещественных чисел.

Алгоритм решения задачи состоит из следующих этапов:

1. Чтение вещественных чисел из файла в массив a.
2. Поиск в массиве a максимального (max) и минимального (min) значения и их номеров (imax, imin).
3. Перемещение указателя файла к максимальному значению и запись min.
4. Перемещение указателя файла к минимальному значению и запись max.

```
#include <iostream>
using namespace std;
int main()
{
//Описание файловой переменной.
    FILE *f;
```

```
    int i,n,imax, imin;
    double *a, max,min;
//Открытие файла в режиме чтения и записи.
f=fopen("E:\\STUDENT\\abc.dat", "rb+");
//Считываем из файла в переменную n количество
// вещественных чисел в файле.
    fread(&n,sizeof(int),1,f);
    cout<<"n="<<n<<"\n";
//Выделяем память для хранения вещественных чисел,
// эти числа будут хранится в массиве a.
    a=new double[n];
//Считываем из файла в массив a вещественные числа.
    fread(a,sizeof(double),n,f);
//Поиск максимального, минимального элемента в
//массиве a, и их индексов.
    for(imax=imin=0, max=min=a[0],i=1;i<n;i++)
    {
        if (a[i]>max)
        {
            max=a[i];
            imax=i;
        }
        if (a[i]<min)
        {
            min=a[i];
            imin=i;
        }
    }
//Перемещение указателя к максимальному элементу.
    fseek(f,sizeof(int)+imax*sizeof(double),SEEK_SET);
//Запись min вместо максимального элемента файла.
fwrite(&min,sizeof(double),1,f);
//Перемещение указателя к минимальному элементу.
    fseek(f,sizeof(int)+imin*sizeof(double),SEEK_SET);
//Запись max вместо минимального элемента файла.
fwrite(&max,sizeof(double),1,f);
//Закрытие файла.
    fclose(f);
//Освобождение памяти, выделенной под массив a.
    delete []a;
    return 0;
}
```