**08.05.2024 года Файловый ввод/вывод. Текстовые файлы**

**Использование файлов в программах. Ввод/вывод текстовых файлов – использование операций << и >>; посимвольный ввод/вывод; построчный ввод/вывод. Примеры программ.**

**Использование файлов в программах**

**Файловый ввод/вывод** в языке программирования C++ – это часть общей системы ввода/вывода. **Файл –** это совокупность данных на внешнем носителе. В **С*++***обмен информацией с файлами реализуетсячерез***потоки***(потоковые классы).

Для организации работы с файлами в программе **С++** необходимо подключение заголовочного файла <**fstream**>, в котором определены три ***класса* *файловых потоков*:**

**ifstream** для ввода данных из файла;

**ofstream** для вывода данных в файл; к

**fstream** для ввода–вывода данных в файл.

Эти классы являются ***производными*** от классов **istream**, **ostream,** **iostream** соответственно**,** и поэтому также имеют доступ ко всем ***операциям ввода/вывода***, которые определены классом **ios**.

**По способу доступа** файлы разделяют ***на последовательные***, чтение/запись данных в которых производится последовательно, и ***файлы с произвольным доступом***, допускающие чтение/запись данных в *произвольном порядке*.

**По способу организации** различают ***текстовые* и *двоичные***файлы.

Обмен информацией с текстовыми файлами *обычно* реализуется через ***текстовые потоки***, а с двоичными файлами – через ***двоичные потоки***.

**Обмен информацией** с файлом предполагает выполнение следующих действий в программе:

* cоздание потока (объекта);
* открытие файла (связь файла с потоком);
* обмен (ввод/вывод) информацией с файлом;
* закрытие файла (разрыв файла с потоком)*.*

**Создание потока.**По умолчанию стандартная библиотека **C++** для ввода/вывода данных предоставляет стандартные потоки **cout** и **cin,** которые сразу же можно использовать. Файловые потоки должны быть **явно установлены** программистом:

**ifstream** fin; // входной поток (объект) **fin**

**ofstream** fout; // выходной поток (объект) **fout**

**fstream** fio; // поток ввода/вывода (объект) **fio**

**Открытие файла.**Файл можно открыть с помощью функции **ореn():**

ofstream **fout**; // создание потока (объекта) **fout**

fout.**open**("vix.txt", ios::out); // открытие файла **vix.dat**

То есть, чтобы открыть файл **vix.dat** для записи, нужно создать объект класса **ofstream**, вызвать для него функцию **ореn(),** указав имя файла (и путь, если файл создаётся не в папке текущего проекта) в качестве аргумента. При открытии файла с ним связывается **поток ввода/вывода**.

В большинстве же случаев для ***открытия*** файла используют **конструкторы** соответствующих классов:

ofstream **fout**("vix.txt", ios::out); // открытие файла **vix.tx** для вывода

ifstream **fin**("vx.txt", ios::in); // открытие файла **vx.dat** для ввода

fstream **fio**("vvx.txt", ios::in|ios::out); // открытие **vvx.dat** для **в/ы**

Конструкторы классов **создают** **объекты** соответствующих классов файлового ввода/вывода, **открывают** **файл** с указанным именем и ***связывают файл с потоком*** (объектом). Первый аргумент у конструкторов – ***имя файла***, и это обязательный аргумент. Второй аргумент задаёт *режим доступа*к файлу.

Режим доступа к файлу служит для описания характера использования файла – для чтения, записи и т. д. В классе **ios** определены следующие константы для указания режима доступа к файлу:

**ios::in** открыть файл только для чтения;

**ios::out** открыть файл только для записи;

**ios::app** открыть файл для добавления данных в конец файла;

**ios::trunc** если файл существует, удалить его;

**ios::binary** открыть файл в двоичном режиме.

Операция **|** (или) позволяет составить режим с любым сочетанием флагов.

По умолчанию объекты класса **ofstream** открыты для вывода, а класса **ifstream** – для ввода, поэтому режимы **out** и **in** можно опускать. По умолчанию все файлы открываются ***как текстовые***. Если файл не был ранее создан, то класс **fstream** использовать нельзя.

Прежде чем начать обмен данными с файлом, следует убедиться, была ли *операция открытия файла успешной.* Например, проверить правильность открытия файла **vx**.**dat** можно следующим образом:

if(!fin){cout<<" No file open for Input\n"; exit(1);}

**или** if (!fin){cerr<<"No file open for Input\n"; exit(1);}

Аргумент функции **exit()**, отличный от нуля, показывает, что программа прекратила выполнение из-за ошибки, например, неверно указано имя файла.

Следует отметить, что поток, посланный в объект **cerr**, немедленно выводится на экран, минуя буферизацию. Этим **cerr** отличается от **cout.**

**Обмен** информацией с файлом можно выполнять только после того, как поток (объект) успешно соединён с файлом (*т.е. файл открыт*).

**Для закрытия файла** (*т.е. отсоединения потока от файла*) используется функция **close(),** которая не имеет параметров и возвращаемого значения:

fin**.close(); //** закрытие файла **vx**.**txt**

**Для определения конца файла** можно использовать функцию **eof().** Функция возвращает значение **true*,*** если был достигнут конец файла, в противном случае функция возвращает значение **false**.

**Ввод/вывод текстовых файлов**

Ввод/вывод текстовых файлов реализуется через ***текстовые потоки***, которые обеспечивают ***форматированный ввод/вывод*** (*в/ы на высоком уровне*).

При форматированном вводе/выводе числа хранятся в виде последовательности символов. Например, целочисленные значения занимают в памяти компьютера **4** байта, тогда как текстовая версия числа, например «**12345**», занимает **5** байтов. Значения типа **float** также всегда занимают **4** байта. А форматированная версия **«6.02314е13»** занимает **10** байтов.

Файловый ввод/вывод (**в/ы**) аналогичен стандартному вводу/выводу**,** единственное отличие – **в/ы** выполнятся не на консоль, **а в файл**. Для организации файлового **в/ы** **необходимо создать** объекты, которые можно использовать аналогично стандартным. Текстовые файлы целесообразно использовать в случае записи/чтения в файл небольшой по объёму информации.

**Использование операций << и >>.** После открытия файла чтение и запись данных можно выполнить, используя операции >> и <<, которые для *файлов* выполняются так же*,* как и для *стандартных потоков ввода/вывода* **cin** и **cout***.*

// **Пример** **1.** В текстовый файл **date** записать **n** случайных чисел.

// Считать содержимое файла и для контроля вывести на экран.

// Добавить в конец файла **date** **n** случайных чисел.

// Считать содержимое файла и для контроля вывести на экран.

#include <iostream>

using namespace std;

#include<fstream> // подключение **<fstream>**

int **main()** {

int n, num;

cout << "Enter n: "; cin >> n; // **5**

cout << "Write to file ";

srand(n);

ofstream **fout**("date", ios::out); // открытие файла **"date"** для записи

if (!fout) { cout << "No file open\n"; exit(1); }

for (int i = 1; i <= n; i++) {

num = rand() % 15 - 5;

fout << num<< ' '; // запись числа в файл **"date"**

}

cout << "size file = " << fout.**tellp**() << endl; // размер файла

fout.close(); // закрытие файла "**date**"

cout << "Read file date" << endl;

ifstream **fin**("date", ios::in); // открытие файла **"date"** для чтения

if (!fin) { cerr << "No file open\n"; exit(1); }

while (!fin.eof()) { // чтение **"date",** пока не конец

fin >> num;

if (!fin.eof()) cout << num << ' ';

}

fin.close();

cout << "\nEnter n: "; cin >> n; // **3**

cout << "\nAdd to end of the file ";

num = 55;

ofstream **fout1**("date", ios::out | ios::app); // добавление в конец файла **"date"**

if (!fout1) { cerr << "No file open\n"; exit(1); }

for (int i = 1; i <= n; i++)

fout1 << num<< ' ';

**fout1**.close(); // закрытие файла **"date"**

cout << "Read file date" << endl;

ifstream **fin1**("date", ios::in); // открытие файла **"date"** для чтения

if (!fin1) { cerr << "No file open\n"; exit(1); }

while (!fin1.eof()) { // чтение **"date"**, пока не конец

fin1 >> num;

if (!fin1.eof()) cout << num << ' ';

}

fin1.close();

}

**Результат:**

Enter n: 5Write to file size file = 14

Read file date 7 11 -2 12 -2

Enter n: **3** Add to end of the file

Read file date 7 11 -2 12 -2 55 55 55

**Посимвольный ввод/вывод.** Функции **get()** и **put()** являются членами **всех потоковых классов** соответственно и используются для **в/ы** символов:

istream& **get**(char &ch);

ostream& **put**(char ch);

Функция **get()** считывает ***один символ*** из соответствующего потока и помещает его значение в переменную **ch**. Она возвращает ***ссылку*** на поток, связанный с предварительно открытым файлом. Функция **put()** записывает ***символ*** **ch** в поток и возвращает ***ссылку*** на этот поток.

**// Пример 2.** Посимвольный **ввод/вывод** файла.

// Использование функций **put(), get()**.

#include <iostream>

using namespace std;

#include<fstream> // подключение **<fstream>**

int **main(){**

**const** char \* p = "**Vsem privet**";

ofstream **fout** ("date", ios::out); // открытие файла **"date**" для вывода

if (!fout){ cout <<"No file open\n"; exit(1);}

while(\*p)

**fout.put**(\*p++); // запись в файл посимвольно

cout << "size file = " << fout.tellp() << endl; // размер файла =**11**

fout.put('\n');

fout.put('\*');

fout.put('\n');

for(int i = 0; i < 26; i++)

fout.put((char)(65+i));

cout << "size file = " << fout.tellp() << endl; // размер файла -= 42

fout.close();

**char** simv;

ifstream **fin**("date", ios::in); // открытие файла для чтения

if (!fin){cerr <<"No open\n"; exit(1);}

while(!fin.eof()){ // проверка на конец файла,

fin.get(simv); // чтение файла и вывод на экран

if(!fin.eof()) cout<<simv;

}

/\* while(fin){ // пока есть поток

fin.get(simv);

if(fin)cout<<simv;

} /\*

/\* while(fin){

fin.get(simv);

if(fin.eof()) break;

cout<<simv;

} \*/

cout<<endl; fin.close();

}

**Результат:**

**size file = 11**

**size file = 42**

Vsem privet

\*

ABCDEFGHIJKLMNOPQRSTUVWXYZ

**Построчный ввод/вывод.** Обычно построчное чтение и запись файлов работают быстрее посимвольных действий. Для чтения символьного массива (строки) из файла используется функция **getline()** класса **ifstream.** Функция читает строку, пока не встретит *символ ‘***\n’**, и помещает её в буфер. Имя буфера передаётся функции как первый аргумент. Максимальный размер буфера задаётся как второй аргумент функции.

**// Пример 3.** Построчный **ввод/вывод** файла.

// Использование функции **getline()** для чтения файла.

#include <iostream>

using namespace std;

#include<fstream> // подключение <**fstream**>

int **main(){**

const int len = 40;

char buf[len];

ofstream **fout** ("date", ios::out); // открытие файла **"date"** для вывода

if (!fout){cerr<<"No file open\n"; exit(1);}

// **Запись** строк-констант в файл. Строки можно вводить как обычно,

// функцией **gets\_s()**

fout<<"1111 1111\n";

fout<<"22222 22222\n";

fout<<"33333333333\n";

fout.close();

ifstream **fin**("date", ios::in); // открытие **"date"** для чтения

if(!fin){ cerr <<"No file open\n"; exit(1);}

while(!fin.eof()){ // чтение, пока не конец файла

**fin.**getline**(**buf, len**);**

if(!fin.eof()) cout<<buf<<endl; // вывод на экран

}

**/\***while(fin){ // чтение файла, пока есть поток

**fin**.getline(buf, len);

if(fin)cout<<buf<<endl;

}***\*/***

fin.close();

}

**Результат:**

1111 1111

22222 22222

33333333333

// **Пример 4.** В структуре содержится информация о студентах факультета:

// фамилия студента, курс, группа, Создать в любом редакторе текстовый файл

// **"struct"**, содержащий информацию о студентах факультета. Считать файл

// **"struct"** и создать текстовый файл **"struct1"**, записав в него фамилию

// студента и курс. Для контроля файл **"struct1"** вывести на экран.

#include<iostream>

#include <iomanip>

#include<fstream> // подключение библиотеки <**fstream**>

using namespace std;

struct Stud{

char fam[15]; // **указатель нельзя!!!!**

int kurs, group;

};

int **main()**{

Stud st; int i = 0;

ifstream fin("**struct.txt**", ios::in);

if(!fin){ cerr <<"Error"<<endl; exit(1);}

ofstream fout("**struct1.txt**", ios::out);

if(!fout){ cerr <<"Error"<<endl; exit(1);}

while(!fin.eof()){ // чтение файла **"struct.txt"**, пока не конец

fin>>st.fam>>st.kurs>>st.group;

if(!fin.eof()){

fout<<st.fam <<' '<<st.kurs<<' ';

i++;

}

}

fin.close(); fout.close();

cout<<"i = "<<i<<endl;

cout<<"**File** **struct1:** "<<endl;

ifstream in("**struct1.txt**", ios::in);

if(!in){ cerr <<"Error"<<endl; exit(1);}

while(in){

in>>st.fam>>st.kurs;

if(in.eof()) break;

cout<<setw(6)<<st.fam<<setw(5)<<st.kurs<<endl;

}

in.close();

}

**!!!** Текстовый файл структур должен быть создан в любом редакторе и до выполнения программы с именем **struct** добавлен (просто скопирован) в директорию текущего проекта **(**где хранится файл исходного кода программы c расширением **.cpp**).

aaa 1 5

sssss 1 3

bbbb 1 3

fffff 2 7

**Результат:**

i = 4

**File struct1:**

aaa 1

sssss 1

bbbb 1

fffff 2

// **Пример 5.** Ввести матрицу и её размеры из **текстового файла**

**//** и транспонировать её. Результат записать **в текстовый файл**.

// Предполагается, что матрица квадратная.

#include <iostream>

#include <fstream>

using namespace std;

void printMatrix(double\*\* a, int n, int m);

void transpose(double\*\* a, int n, int m);

double\*\* inputMatrix(char\* fileName, int& n, int& m);

void outputMatrix(char\* fileName, double\*\* matr, int n, int m);

void del(double\*\* matr, int n);

int **main()**{

char fileName[40];

cout << "Input the name of input file: "; cin >> fileName;

double\*\* matr;

int n = 0, m = 0;

matr = **inputMatrix**(fileName, n, m);

**printMatrix**(matr, n, m);

**transpose**(matr, n, m);

cout << endl << endl << "Transpose matrix" << endl << endl;

**printMatrix**(matr, n, m);

char nameOutputFile[40];

cout << "Input the name of output file: ";

cin >> nameOutputFile;

**outputMatrix**(nameOutputFile, matr, n, m);

**del**(matr, n);

}

//выделение памяти и заполнение матрицы.

// размер матрицы и её элементы считываются с файла

double\*\* **inputMatrix**(char \* fileName, int& n, int& m){

ifstream fin(fileName);

if (!fin){ cerr << "Cannot open file.\n"; exit(1);}

double\*\* matr;

fin >> n; fin >> m; // считываем **n** и **m**

matr = new double\*[n];

if (!matr) {cout << "Error memory allocation.\n"; exit(1);}

for (int i = 0; i < n; i++) {

matr[i] = new double[m];

if (!matr[i]) {cout << "Error memory allocation.\n"; exit(1);}

}

for (int i = 0; i < n; i++) // считываем матрицу

for (int j = 0; j < m; j++)

fin >> matr[i][j];

fin.close();

return matr;

}

// вывод матрицы

void **printMatrix**(double\*\* a, int n, int m){

cout.setf(ios::fixed);

cout.precision(2);

for (int i = 0; i < n; i++){

for (int j = 0; j < m; j++){

cout.width(7);

cout << a[i][j];

}

cout << endl;

}

}

// транспонирование матрицы

void **transpose**(double\*\* a, int n, int m){

for (int i = 0; i < n; i++){

for (int j = 0; j < i; j++){

double temp = a[i][j];

a[i][j] = a[j][i];

a[j][i] = temp;

}

}

}

// запись матрицы в файл

void **outputMatrix**(char\* fileName, double\*\* matr, int n, int m){

ofstream fout(fileName);

if (!fout){ cerr << "Cannot open file.\n"; exit(1);}

fout << n << " " << m << endl;

for (int i = 0; i<n; i++){

for (int j = 0; j<m; j++){

fout.width(8);

fout << matr[i][j];

}

fout << endl;

}

fout.close();

}

// освобождение динамической памяти

void **del**(double\*\* matr, int n){

for (int i = 0; i < n; i++){

delete[] matr[i];

}

delete[] matr;

}

Текстовый файл с исходной матрицей **3x3** для данной программы должен выглядеть, например, так:

3 3

1.1 2.2 -3.33

2.7 -6.9 8.9

1 2 4.67

Файл должен быть создан в любом редакторе и записан в текущую директорию проекта с именем, например, **input** до выполнения программы **(где хранится файл исходного кода программы с расширением .cpp).**

**Результат:**

Input the name of file: **input.txt !!!** расширение **.txt** указывать

1.10 2.20 -3.33

2.70 -6.90 8.90

1.00 2.00 4.67

Transpose matrix

1.10 2.70 1.00

2.20 -6.90 2.00

-3.33 8.90 4.67

Input the name of output file: **output.txt !!!** расширение **.txt** указывать