**15.05.2024 года Файловый ввод/вывод. Двоичные файлы**

**Двоичные (binary) файлы. Ввод/вывод (чтение/запись) двоичных файлов. Произвольный доступ к файлу. О записи указателей (адресов) в файл.**

**Двоичные (binary) файлы**

Текстовые файлы целесообразно использовать в случае чтения/записи небольшой по объёму информации. В противном случае гораздо эффективнее использовать **двоичный** ввод/вывод, при котором данные на внешнем носителе хранятся, как и в **ОП** компьютера.

**Двоичный файл** – это последовательность **байтов**. Обмен информацией с двоичными файлами реализуется через **двоичные потоки**. Для работы с двоичными файлами их следует открывать в **двоичном режиме** (**ios::binary),** что предотвращает преобразование символов.

**Ввод/вывод файлов (чтение/запись)**

Для чтения данных из файла создаётся объект класса **ifstream,** для записи данных вфайл – объект класса **ofstream.** Для чтения/записи данных необходимо создать объект класса **fstream,** причём класс **fstream** можно использовать только в том случае, если файл уже был ранее создан.

Для работы с *двоичными файлами* в языке **C++** поддерживаетсяширокий диапазон ***функций*** ввода/вывода. Функции дают возможность точно контролировать процессы считывания информации из файла и записи в файл.

**Функции** **put(), get().** Используя функцию **put()** можно ***записать байт* в файл**, а с помощью функции **get() – *считать байт*** из файла.

При считывании символа конца файла функция**get()**возвращает значение **0.**

**// Пример 1.** Использование функций **put(), get()** для записи/чтения в **bin**-файл.

#include <iostream>

using namespace std;

#include<fstream>

int **main(){**

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

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

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

while(\*p) // запись в файл строки-константы

**fout.put**(\*p++);

fout.put('\n');

**char** simv;

cout<<"Enter simbols. For end -- enter **0**"<<endl;

while(1){ // запись в файл символов

cin.get(simv);

if(simv == '0') break;

fout.put(simv);

}

fout.close();

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

if (!fin){cout <<"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;

}

cout<<endl;

/\*

while (!fin.eof()) {

fin >> simv;

if (!fin.eof()) cout << simv; // Vsemprivetabcdf123

}

\*/

fin.close();

}

**Результат:**

Enter simbols. For end -- enter 0

abcd f 1 2 3 0

Vsem privet

abcd f 1 2 3

Следует обратить внимание, что для считывания символов из потока **cin** в программе используется функция **get()**. Это предотвращает игнорирование пробельных символов.

**Функции** **read() и write().** Являются членами потоковых классов соответственно для ввода и вывода. Прототипы функций имеют следующий вид:

ifstream& **read((**char **\*) buf**, size **num);**

ofstream& **write((** char \*) **buf**, size **num);** где

**buf** – указатель на область памяти (**буфер)**;

**num** – число\_байтов.

Функциям всё равно, как организованы данные, что они собой представляют, они просто переносят байты из буфера в файл и обратно.

Функция **read()** класса **ifstream**считывает с файла **num** байтов и передаёт их в буфер, определённый указателем **buf.** Функция **write()** класса **ofstream** записывает **num** байтов в файл из буфера, заданного указателем **buf**.

Приведение к типу **(char \*)** адреса буфера в функциях необходимо, если буфер ввода/вывода не определён как символьный массив. Поскольку в **С++** указатель на один тип не преобразуется автоматически в указатель на другой тип.

При считывании символа конца файла функция**read()**возвращает значение **0.**

**// Пример 2.** Запись в **bin**-файл целых чисел. Чтение

// созданного файла. Добавление чисел в конец файла.

#include<iostream>

#include<fstream>**>**

using namespace std;

int **main()**{

int n, num;

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

ofstream **fout**("test", ios::binary); // открытие файла для запизи

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

srand(n);

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

num = rand() % 20 - 10;

**fout**.write((char\*) &num, sizeof(int)); // запись в файл

}

cout <<"size file = "<<fout.**tellp()**<<endl; // size file = 20 байтов

fout.close();

ifstream **fin**("test", ios::binary); // открытие файла для чтения

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

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

**fin.**read((char\*) &num, sizeof(int));

if(!fin.eof())cout<<num<<' '; // вывод для контроля на экран

}

cout<<endl;

fin.close();

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

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

int dop = 55;

cout<<"Enter n for add "; cin>>n;

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

**out**.write((char\*) &dop, sizeof(int));

out.close();

ifstream **in**("test", ios::binary);

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

while(in){ // чтение, пока есть поток

**in**.read((char\*)&num, sizeof(int));

if(in) cout<<num<<' '; // вывод для контроля на экран

}

cout<<endl; in.close();

}

**Результат:**

Enter n **6**

4 3 5 -1 -10

size file = 20

Enter n for add **4**

4 3 5 -1 -10 55 55

**// Пример 3.** Запись/чтение **char**-arrays (строк) в **bin-**файл/из **bin-**файла.

#include <iostream>

#include <cstring>

#include <fstream>

using namespace std;

int **main()** {

char buf[30];

int n;

ofstream fout("data", ios::binary); // открытие файла для записи

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

cout<<"Enter n -- number **char**-arrays: "; cin>>n;

cin.ignore();

cout << "Enter "<<n<<" char-arrays:\n";

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

gets\_s(buf);

fout.write(buf, sizeof(buf)); // запись (вывод) в файл строки

}

fout.close(); // закрытие файла

cout<<"Char-arrays: *\*n";

ifstream fin("data", ios::binary); // открытие файла для чтения

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

while(fin){ // пока есть поток

fin.read(buf, sizeof(buf)); // чтение (ввод) из файла строки

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

}

/\*

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

fin.read(buf, sizeof(buf)); // чтение (ввод) из файла

cout << buf << endl;

}

\*/

fin.close();

}

**Результат:**

Enter n -- number char-arrays: 3

Enter 3 char-arrays:

11111

333 444

333 444 555

Char-arrays:

11111 333 444 333 444 555

**Произвольный доступ к файлу**

В отличие от файлов *последовательного доступа*, любая из записей файлов ***произвольного*  *доступа*** может быть доступна ***непосредственно***, без считывания или записи предшествующих записей. Например, можно создать файл с фиксированной длиной записи, что даёт возможность определять расположение записей в файле относительно начала файла.

**Файловые указатели.** Каждый класс файлового **ввода/вывода** содержит файловые указатели – **get***-* и **put***-указатели,* которые определяют, в каком месте файла должна выполниться ***следующая*** операция ввода/вывода (чтения/записи) соответственно.

Любая запись/чтение в файл/ из файла происходит в ***текущем*** местоположении *файлового указателя*. По умолчанию, *при открытии файла* для чтения/записи, файловые указатели находятся в **самом начале** файла. Однако, если файл открывается в режиме ***добавления***, то файловый указатель перемещается в **конец файла**, чтобы пользователь имел возможность добавить данные в файл.

**Функции произвольного доступа к файлу.**В системе ввода/вывода **C++** произвольный доступ к файлу осуществляется путём управления файловыми указателями с помощью следующих функций классов **istream** и **ostream**:

**tellg() –** возвращает текущую позицию чтения;

**seekg(**pos**) –** устанавливает абсолютную позицию **pos** чтения;

**seekg(**offset, rpos**) –** устанавливает относительную позицию чтения;

**tellp() –** возвращает текущую позицию записи;

**seekp(**pos**) –** устанавливает абсолютную позицию **pos** записи;

**seekp(**offset, rpos**) –** устанавливает относительную позицию записи.

Смещение **rpos** задаётся по отношению к трём позициям, для которых определены соответствующие константы:

**ios::beg** – cмещение относительно начала файла (по умолчанию);

**ios::cur –** cмещение относительно текущей позиции указателя;

**ios::end** – смещение относительно конца файла.

Функция **seekg()** перемещает текущий **get**-указатель, а функция **seekр()** перемещает текущий **pu**t-указатель соответствующих файлов. Класс **fstream**, использующийся для записи/чтения, имеет доступ к обеим функциям.

Положительное смещение означает перемещение файлового указателя к концу файла, тогда как отрицательное смещение означает перемещение файлового указателя к началу файла.

При позиционировании необходимо всегда следить за тем, чтобы позиция оставалась внутри файла. Позиционирование перед началом или после конца файла приводит к непредсказуемым последствиям.

Перейти к началу или концу файла очень просто:

inf.seekg(0, ios::beg); // перейти к **началу** файла

inf.seekg(0, ios::end); // перейти в **конец** файла

Используя функции **seekg()** и **seekp(),** можно получить доступ к записям файла в произвольном порядке.

Функции **tellg() и tellp()в**озвращают текущую позицию файлового указателя. Их можно использовать для определения размера файла:

// **Пример 4.** Произвольный доступ к **bin**-файлу.

// Использование функций **seekg(),** **seekp(), tellp(), tellg()**.

#include <iostream>

#include <fstream>

using namespace std;

int **main()**{

const char\* p = "**Vsem privet123**";

**ofstream** fout("date", ios::out | ios::binary);

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

while (\*p)

fout.put(\*p++); // запись в файл

cout <<"size file = "<<fout.**tellp()**<<endl; // size file = **14**  байтов

fout.seekp(5);

fout.put('P');

fout.seekp(0, ios::end); // перейти в конец файла

cout <<"size file = "<<fout.**tellp()**<<endl; // size file = **14**  байтов

fout.seekp(-2, ios::end); fout.put('5'); // запись **'5**'

fout.seekp(2, ios::beg); fout.put('X'); // запись **'X**'

fout.close();

char simv;

**ifstream** fin("date", ios::in | ios::binary);

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

// fin.seekg(0, ios::beg); // переход в начало файла

cout <<"size file = "<< fin.**tellg()**<<endl; // size file = **0** байтов

while (!fin.eof()) {

fin.get(simv);

if (!fin.eof()) cout << simv; // **VsXm Privet153**

}

cout << endl; fin.close();

**fstream** out("date", ios::in | ios::out | ios::binary);

out.seekp(**3**, ios::beg); out.put('Z'); // запись **'Z**'

out.seekp(**-3**, ios::end); out.put('4'); // запись **'4'**

out.seekg(0, ios::beg); // переход **в начало** файла

while (out) { // чтение, пока есть поток

out.get(simv);

if (out)cout << simv; // **VsXZ Privet453**

}

cout << endl; out.close();

}

При произвольном доступе к файлу преобразования символов, которые могут происходить в ***текстовых файлах*,**могут привести к тому, что запрашиваемая позиция файла не будет соответствовать реальной. Также в некоторых компиляторах реализация **seekg(),** **seekp()** при использовании с текстовыми файлами может давать ***ошибки***. Поэтому в **общем случае** произвольный доступ для ***операций ввода/вывода*** должен выполняться *только для файлов,* открытых в двоичном режиме *– двоичных файлов.*

// **Пример 5.** Создать бинарный файл **натуральных** целых чисел. Найти в файле

// число с максимальным значением и заменить его нулём в **самом** файле.

#include <iostream>

#include <fstream>

using namespace std;

int **main()** {

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

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

int num, n;

cout << "Enter n: "; cin >> n;

srand(n);

cout << "Write file:" << endl;

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

num = rand() % 20;

fout.write((char\*) &num, sizeof(int)); // запись в файл

}

fout.close();

cout << "Read file: " << endl;

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

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

while (!fin.eof()) {

fin.read((char\*) &num, sizeof(int)); // вывод на экран

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

}

// Иногда приходится сбросить состояние файла функцией перед тем, перед тем,

// как выполнять с ним операции, включая изменение позиции чтения

**fin.clear();**

fin.seekg(0, ios::beg);

int max = 0, ind = 0; int k = 0, nul = 0;

while (fin) { // поиск **max** и его индекса

fin.read((char\*) &num, sizeof(int));

if (!fin)break;

if (num > max) { max = num; ind = k; }

k++;

}

cout << "\n max = " << max << " ind = " << ind << endl;

fin.close();

**fstream** fio("date", ios::in | ios::out | ios::binary); // для записи/чтения файла

fio.seekp(sizeof(int) \* **ind**, ios::beg);

fio.write((char\*) &**nul**, sizeof(int)); // замена

fio.seekg(0, ios::beg);

cout << "Read file:" << endl;

while (fio) {

fio.read((char\*)&num, sizeof(int)); // чтение файла

if (fio) cout << num << ' ';

}

cout << endl;

fio.close();

}

**Результат:**

Enter n: 7

Write file:

Read file:

1 2 15 0 1 16 12

max = 16 ind = 5

Read file:

1 2 15 0 1 0 12

**// Пример 6.** Запись в **bin-**файл массива чисел одним блоком.

// Изменение содержимого файла (в самом файле).

// Чтение из **bin-**файла массива чисел (блока). Произвольный доступ к **bin**-файлу.

#include <iostream>

#include <fstream>

using namespace std;

int **main()**{

const int n = 10;

int buf[n]; // массив целых чисел

for (int i = 0; i < n; i++) // заполнение массива

buf[i] = i; // (0, 1, 2, …

cout<<"Sourse array: \n";

for (int i = 0; i < n; i++) // вывод на экран массива

cout << buf[i] << ' ';

cout << endl;

ofstream fout("test", ios::binary);

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

cout<<"Writing array to file :\n";

fout.write((char\*) buf, n \* sizeof(int)); // запись в файл массива

// fout.write((char\*) buf, sizeof(buf)); // или так

cout<<"Changing file contents:\n";

fout.**seekp**(4, ios::beg);

int z = 77;

fout.write((char\*) &z, sizeof(z)); // запись числа **77** с позиции **4**.

fout.**seekp**(20, ios::beg);

cout <<fout.**tellp()**<<endl; // **20**

fout.put('A'); // запись символа **'A'** с позиции **20**

fout.close();

ifstream fin("test", ios::in | ios::binary); // открытие файла для чтения

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

cout<<"Reading array from file:\n";

fin.read((char\*) buf, n \* sizeof(int)); // чтение массива из файла "**test**"

// fout.read((char\*) buf, sizeof(buf)); // или так

for (int i = 0; i < n; i++) // и вывод массива на экран

cout << buf[i];

cout << endl;

cout<<"Reading array from file by one number:\n";

fin.seekg(0, ios::beg);

int num;

while (fin) { // чтение массива из файла, пока есть

fin.read((char\*) &num, sizeof(int)); // поток, по одному числу

if (fin) cout << num << ' ';

}

fin.close();

}

**Результат:**

Sourse array:

0 1 2 3 4 5 6 7 8 9

Writing array to file :

Changing file contents:

20

Reading array from file:

0**77**234**65**6789

Reading array from file by one number:

0 **77** 2 3 4 **65** 6 7 8 9

// **Пример 7**. Чтение текстового **файла-структур**. Создание **bin-файла**.

// Перестановка структур в файле.

// Исходный файл структур надо создать в редакторе, например, в Блокноте

// и поместить в папку проекта с именем **struct**, где файл исходного кода программы.

#include<iostream>

#include <iomanip>

#include<fstream>

using namespace std;

struct **Stud**{

char fam[15]; // **указатель нельзя**, т. к. пишем структуру в файл

int kurs, group;

};

void **sozdBin**(); // функция создания файла

void **print()**; // функция вывода файла

void **obmen();** // функция перестановки структур **в файле**

int **main()**{

sozdBin();

print();

obmen();

print();

}

void **sozdBin**(){ // функция **sozdBin**()

cout<<"fun **sozdFileBin**: \n";

ifstream **fin**("struct.txt"); // открытие файла для чтения

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

ofstream **fout**("structBin", ios::binary); // открытие файла для записи

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

Stud st;

while(fin){ // чтение файла-структур

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

if(!fin)break;

fout.write((char\*) &st, sizeof(Stud)); // запись в **bin-файл**

// fout.write((char\*) &st, sizeof(st)); // или так

}

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

}

void **obmen(){** // функция **obmen()**

cout<<"fun **obmen()**\n";

Stud x, y;

int k = 0;

**fstream** fio("**structBin**", ios::binary | ios::in | ios::out);

if(!fio){cout<<"Error"<<endl; exit(1);}

fio.seekg(k, ios::beg);

fio.read((char\*) &**x**, sizeof(Stud));

fio.read((char\*) &y, sizeof(Stud));

fio.seekp(k, ios::beg);

fio.write((char\*) &y, sizeof(Stud));

fio.write((char\*) &x, sizeof(Stud));

fio.close();

}

void **print()**{ // функция **print()**

cout<<"fun **print**: \n";

ifstream fin("structBin", ios::binary);

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

Stud st;

while(fin){

fin.read((char\*) &st, sizeof(Stud));

if(fin)

cout<<setw(10)<<st.fam<<setw(5)<<st.kurs<<setw(5)<<st.group<<endl;

}

cout<<endl; fin.close();

}

**Результат:**

fun **sozdBin:**

fun **print:**

aaa 1 5

sssss 1 3

bbbb 1 3

fffff 2 7

fun **obmen:**

fun **print:**

sssss 1 3

aaa 1 5

bbbb 1 3

fffff 2 7

**О записи указателей (адресов) в файл**

Известно, что указатели содержат адреса данных, например, переменных, И хотя эти адреса можно записывать в файл и считывать их из файла – это может **создать проблемы**, так как адрес одной и той же переменной может **отличаться** при каждом повторном запуске программы.

Аналогично и для ввода/вывода структур, объектов, полями которых являются указатели, т. к. значения указателей не будут корректными при чтении объекта в другую область памяти.

Не следует сохранять адреса данных в файлах. Данные, которые изначально были по одним адресам, при повторном запуске программы могут находиться уже по другим адресам.