**Тема. Файловый ввод/вывод. Двоичные файлы.**

**Двоичные файлы. Ввод/вывод файлов. Функции put(), get(). Функции** **read(),** **write(). Произвольный доступ к bin-файлу. Объектный ввод/вывод bin-файлов. Флаги ошибок. О записи указателей в файл.**

**Двоичные (binary) файлы**. Текстовые файлы целесообразно использовать в случае записи/чтения в файл небольшой по объёму информации. В противном случае, конечно, гораздо эффективнее использовать **двоичный** ввод/вывод, при котором числа хранятся таким же образом, как в **ОП** компьютера.

***Двоичный файл*** – это последовательность **байтов**. Обмен информацией с двоичными файлами реализуется через *двоичные потоки*.

**Ввод/вывод файлов**. Для записи данных в файл создаётся объект класса **ofstream,** для чтения данных изфайла – объект класса **ifstream,** а для **в/ы** данных – объект класса **fstream.** Для тогочтобы открытьфайл как двоичный, необходимо задать режим доступа к файлу **ios::binary.**

Для записи данных в двоичный файл и их считывания можно использовать *операции >> и <<*. Например, записать значения типа **double** в файл и их считать можно следующим образом:

double d = 5.555, z;

ofstream **fout**("test", ios::out | **ios::binary**); // открыть для вывода

fout<<d<<endl;

ifstream **fin**("test", ios::in | **ios::binary**); // открыть для ввода

fin>>z;

cout<<"z = "<<z<<endl; // z = **5.555**

Язык **C++** также предоставляет широкий ***набор функций*** для работы с *двоичными файлами*, которые дают возможность точно реализовывать процессы считывания информации из файлов и записи в файлы.

**Функции** **get(), put().** При произвольном доступе к файлу преобразования символов, которые могут происходить в ***текстовых файлах*** (например, символ ***новой строки*** может быть некоторыми библиотечными функциями заменен парой символов **— *возврата каретки и перевода строки*)**, 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.*

С помощью функции **put()** можно *записать байт*, а с помощью функции **get() –** *считать* *байт*.

// **Пример 1.** Произвольный доступ к bin-файлу.

// Использование функций **seekg()** и **seekp()**.

#include <iostream>

#include <fstream>

using namespace std;

int **main()**{

const char\* p = "Vsem privet111";

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

}

fout.put('\n'); fout.put((char)65); // запись символа 'A'

fout.put((char)66); // запись символа 'B'

fout.seekp(11, ios::beg);

fout.put('\*'); // запись символа '\*'

fout.close();

char simv;

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

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

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

fin.get(simv); // чтение символа и вывод

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

}

cout << endl; fin.close();

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

out.seekp(2, ios::beg);

out.put('X'); out.put('Z'); // запись **'X**' и **Z** в файл, позиции – 2, 3

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

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

out.get(simv);

if (out)cout << simv;

}

cout << endl; out.close();

}

**Результат:**

Vsem privet\*11

AB

VsXZ privet\*11

AB

**Функции** **read()** и **write().** Для считывания и записи *группы* *байтов* (*блоков двоичных данных*) используют функции**,** прототипы которых имеют следующий вид:

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

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

Функции **read()/write()** соответственнокласса **ifstream**/**ofstream** вводит/выводит из заданного потока/в заданный поток в область памяти (**buf)** фиксированное число байтов **(num)**, начиная с указанного адреса. Если поток связан с файлом, то данные считываются/записываются, начиная с позиции в файле, определяемой функцией позиционирования указателя файла **seekg()**/**seekp().**

Тип **type** должен быть определён как некоторая разновидность *целочисленного типа*.

Этим функциям всё равно, как организованы данные, что они собой представляют, – они просто переносят байты из буфера в файл и обратно. Параметрами этих функций являются **адрес буфера** и его длина.

Если при считывании данных конец файла достигнут до того, как было считано **num** символов, выполнение функции **read()** просто прекращается.

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

Приведение типа к **(char\*)** в функциях **read()** и **write()** необходимо, если буфер **в/ы** не определён как символьный массив, т.к. в **С++** указатель на один тип *не преобразуется автоматически* в указатель на другой тип.

**// Пример 2.** Запись в двоичный файл целых чисел.

#include <iostream>

#include <fstream>

using namespace std;

int **main()**{

const int MAX = 100; // размер буфера

int buf[MAX]; // буфер для целых чисел

for(int j = 0; j < MAX; j++) // заполнить буфер данными

buf[j] = j; // (0, 1, 2, …

ofstream **out**("data", ios::binary); // создать выходной поток

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

out.write((char\*) buf, MAX\*sizeof(int) ); out.close(); // закрыть поток

for(int j = 0; j < MAX; j++) // стереть буфер

buf[j] = 0;

ifstream **in**("data", ios::binary); // создать входной поток

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

in.read((char\*) buf, MAX\*sizeof(int) ); // читать поток

for(int j = 0; j < MAX; j++) // проверка данных

if( buf[j] != j ) { cout << "Nocorrect dannie!\n"; return 1; }

cout << " Correct dannie \n";

in.close();

}

**Результат:**

Correct Dannie

**// Пример 3.** Запись строк в двоичный файл .

#include <iostream>

#include <cstring>

#include <fstream>

using namespace std;

int main() {

char buf[30];

ofstream out("data", ios::binary); // создать выходной поток

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

for (int i = 0; i < 3; i++) { // заполнить буфер

cout << "Enter stroku: "; gets\_s(buf);

out.write(buf, sizeof(buf));

}

out.close(); // закрыть поток

ifstream in("data", ios::binary); // создать входной поток

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

for (int i = 0; i < 3; i++) {

in.read(buf, sizeof(buf)); // читать поток

cout << buf << endl;

}

in.close();

}

**Результат:**

Enter stroku: 1111 2222

Enter stroku: 33333 44

Enter stroku: 55 6666 7777

1111 2222

33333 44

55 6666 7777

**// Пример 4.** Запись в двоичный файл целых чисел. Чтение

// созданного файла. Добавление данных в конец файла.

#include<iostream>

#include<fstream>**>**

using namespace std;

int **main()**{

int n, a;

cout<<"Vvedi n "; cin>>n;

ofstream **fout**("**masBin**", ios::binary);

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

srand(n);

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

a = rand() % 20 - 10;

**fout**.write((char\*) &a, 4); // запись в файл

}

fout.close();

ifstream **fin**("**masBin**", ios::binary);

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

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

fin.read((char\*)&a, 4);

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

}

cout<<endl;

fin.close();

ofstream **fout1**("masBin", ios::binary | ios::**app**);

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

int dop = 55;

**fout1**.seekp(8, ios::beg); // всё равно добавление в конец файла

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

**fout1**.write((char\*) &dop, 4);

fout1.close();

ifstream **fin1**("**masBin**", ios::binary);

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

while(fin1){ // чтение, пока есть поток

**fin1**.read((char\*)&a, 4);

if(fin1) cout<<a<<' '; // вывод для контроля на экран

}

cout<<endl;

fin1.close();

}

**Результат:**

Vvedi n 6

8 3 9 -5 -4 -1

8 3 9 -5 -4 -1 55 55 55 55 55 55

**// Пример 5.** Запись/считывание в файл/из файла массива чисел.

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

#include <iostream>

#include <fstream>

using namespace std;

int **main()** {

const int n = 5;

ofstream fout("test", ios::out | ios::binary); // открытие файла

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

int mas[] = { 1, 2, 3, 4, 5 };

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

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

int z = 7;

fout.write((char\*)&z, sizeof(z)); // запись числа **7** с позиции **4**.

fout.**seekp**(8, ios::beg);

fout.put('A'); // запись символа 'A'

fout.close();

int a;

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

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

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

fin.read((char\*)&a, 4);

if (fin) cout << a << ' '; // **1 7 65 4 5**

}

fin.close();

}

// **Пример 6.** Поиск в файле числа с максимальным значением и его

// порядкового номера, и замена числа нулём.

#include <iostream>

#include <fstream>

using namespace std;

int **main()**{

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

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

int num, n;

cout<<"Vvedi kolichstvo chisel: "; cin>>n;

srand(n); // для получения разных чисел

cout<<"Write file"<<endl;

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

num = rand() % 20 - 10;

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

}

fout.close();

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

ifstream **fin1**("**date**", ios::binary);

if(!fin1){cout<<"File no open\n"; exit(1);}

**int x;**

while(!fin1.eof()){

fin1.read((char \*)&x, sizeof(x));

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

}

cout<<endl;

fin1.close();

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

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

cout<<"Read file"<<endl;

int max, ind; int k = 0;

fin.read((char \*) &max, sizeof(max)); // чтение первого числа

cout<<max<<' ';

ind = 0;

while(!fin.eof()){ // поиск **max** и индекса

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

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

k++;

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

}

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

fin.close();

int z = 0;

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

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

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

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

cout<<"Read file"<<endl;

while(1){

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

if(fio.eof()) break;

cout<<num<<' ';

}

cout<<endl;

fio.close();

}

**Результат:**

Vvedi kolichstvo chisel: **7** Write file

Read file:

-9 -8 5 -10 -9 6 2

Read file

-9 -8 5 -10 -9 **6** 2 **max = 6 ind = 5**

Read file

-9 -8 5 -10 -9 **0** 2

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

// Перестановка структур в файле. Сортировка структур в файле.

// Исходный файл структур "**struct**"надо создать в редакторе, например,

// Блокноте и поместить в папку проекта, 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 **sozdFileBin**(); // функция создания файла

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

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

int **main()**{

sozdFileBin();

print();

perestanovka();

print();

}

void **sozdFileBin**(){ // создание **binary**-файла структур

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

ifstream **fin**("struct.txt");

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

ofstream **fout**("structBin", ios::binary);

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

Stud st;

while(fin){

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

if(!fin)break;

fout.write((char\*)&st, sizeof(Stud));

}

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

}

void **perestanovka(){**

cout<<"fun **Perestanonka**: \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()**{

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 sozdFileBin t:**

fun **Print:**

aaa 1 5

sssss 1 3

bbbb 1 3

fffff 2 7

fun **Perestanonka:**

fun **Print:**

sssss 1 3

aaa 1 5

bbbb 1 3

fffff 2 7

**Объектный ввод/вывод в файл**

При записи объектов в файл следует использовать бинарный режим. При этом в файл на диск записывается та же битовая конфигурация, что хранится в памяти.

// **Пример** 8**.** .Запись объектов в файл. Добавление

// объектов в конец файла. Считывание объектов из файла.

#include <fstream>

#include <iostream>

using namespace std;

class Person{

char name[30]; // имя человека

short age; // возраст

**public:**

void getData();

void showData();

};

void Person::getData(){ // функция ввода данных

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

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

}

void Person::showData() { // функция вывода данных

cout << "Name: " << name << endl;

cout << "Age: " << age << endl;

}

int **main()** {

// запись одного объекта в файл

Person ob; // создание объекта

cout << "Enter dannie o persone:\n";

ob.getData(); // получение данных

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

fout.**write**((char\*)&ob, sizeof(ob)); //запись объекта

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

fout.close();

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

char ch;

// добавление объектов в конец файла

Person pers; // создать объект **Person**

**fstream** file; // создать поток **в/ы**

file.open("person", ios::app | ios::out | ios::in | ios::binary );

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

do{

cout << "Enter dannie o persone:\n";

pers.getData(); // ввод данных

file.write((char\*) &pers, sizeof(pers) ); //запись в файл

cout << "Continue vvod (y/n)? "; cin >> ch;

} while(ch == 'y'); // выход по '**n**'

**file.seekg(0);** // указатель на начало файла

cout << "Spisok Person:\n";

file.read((char\*)&pers, sizeof(pers) ); // чтение

while( !file.eof() ) {

pers.showData(); //вывод данных на консоль

file.read((char\*)&pers, sizeof(pers));

}

cout << endl;

}

**Результат:**

Enter dannie o persone:

Enter name: AAAA Enter age: 35

Name: AAAA

Age: 35

Enter dannie o persone:

Enter name: CCCC Enter age: 55

Continue vvod (y/n)? **y**

Enter dannie o persone:

Enter name: VVVV Enter age: 40

Continue vvod (y/n)? **n**

**Spisok Person:**

Name: AAAA

Age: 35

Name: CCCC

Age: 55

Name: VVVV

Age: 40

Следует обратить внимание на тот факт, новые данные об объектах дописываются **в конец** файла"**person**" (т.е. информация накапливается), так как режим доступа к данному файлу **ios::app**. Отметим также, что используется класс **fstream**, который позволяет и записывать данные в файл, и считывать их из файла.

Обычно принято для записи/чтения объектов файл использовать разные программы или классы. Причём, классы в программах должны иметь одни и те же *поля данных*, а функции у них могут быть совершенно *разные*  – в одном классе функция ввода исходных данных для объекта, а в другом классе – функция вывода полей объекта. Не имеет значения, какие используются функции в классах, – они в файл вместе с данными не записываются. Это для данных важен единый формат. Впрочем, это утверждение справедливо только для ***обычных классов***, в которых не используются ***виртуальные*** функции.

**Флаги ошибок.** Объекты классов, наследуемых от **ios** классов, содержат ***флаги статуса ошибок***, с помощью которых можно проверить результат выполнения операций. Например, при чтении файла рано или поздно будет прочитан конец файла. Сигнал **EOF** посылается в программу операционной системой, когда больше нет данных для чтения, например:

while(!fin.eof()) // пока не **EOF**

Проверяя конкретный флаг признака окончания файла, надо учитывать, что, мы не проверяем все остальные ***флаги ошибок***. Чтобы проверять всё, что можно, следует изменить условие цикла:

while(fin.good()) // пока нет ошибок

Можно также проверять поток напрямую, как в некоторых программах и делалось. Любой *потоковый объект*, например **fin** может тестироваться на выявления ошибок, включая **EOF**. Если какое-либо из условий ошибки имеет значение **true**, объект возвращает **ноль**, иначе, объект возвращает *ненулевое* значение. Поэтому следующий вариант цикла **while** также будет правильным:

while(fin) // пока нет ошибок

**О записи указателей в файлы.** Хотя записывать переменные в файл достаточно просто, всё становится немного сложнее, когда программист начинает работать с указателями. Известно, что указатели содержат лишь адреса переменных, на которые они указывают. 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 объектов, полями которых являются указатели, т.к. значения указателей не будут корректными при чтении объекта в другую область памяти.

**Правило**: Не следует сохранять адреса переменных в файлах. Переменные, которые изначально были по одним адресам, при повторном запуске программы могут находиться уже по другим адресам.