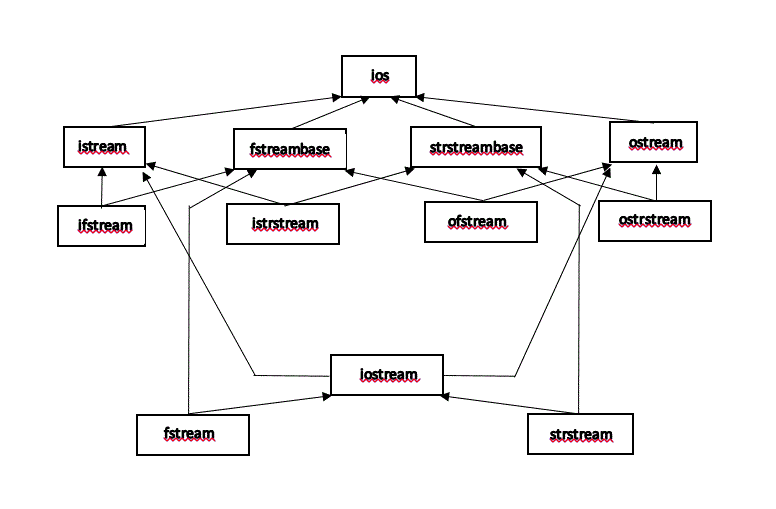
Лекция 6. Потоки и файлы

**Потоковые классы**

**Поток** - это объект класса, предназначенный для переноса информации от источника к приёмнику. Потоковые классы имеют сложную иерархическую структуру. Базовым классом является ios, который предназначен для организации ввода-вывода и содержит общие для различных операций ввода-вывода константы и методы. Названия производных классов формируются с использованием следующих символов:

 - Направление передачи данных (i - ввод, o - вывод, io - ввод-вывод);

- Вид источника данных (f - файл, str - строка символов);



Флаги форматирования позволяют включить или выключить один из параметров ввода/вывода. Чтобы установить флаг ввода/вывода, необходимо вызвать функцию **setf()**, если необходимо отключить флаг вывода, то используется функция  **unsetf()**. Далее показаны конструкции установки и снятия флагов вывода. Доступ к функциям оператора вывода выполняется через операцию точка.

// установка флага вывода

cout.setf( ios::/\*name\_flag\*/ );// устанавливает флаг и имеет 2 перегруженные версии: 1-ая версия имеет 1 параметр - флаг, а 2-ая имеет 2 параметра - флаг и маску

// где name\_flag - это имя флага

// снятие флага вывода

cout.unsetf( ios::/\*name\_flag\*/ );// сброс флагов

// где name\_flag - это имя флага

**Класс ios**

Все классы, предназначеные для работы с потоками, являются потомками класса ios. Данный класс содержит константы и методы, определяющие режимы работы, флаги форматирования и ошибок.

**Флаги форматирования**

|  |  |
| --- | --- |
| Флаг | Значение |
| skipws | Пропуск пробелов при вводе |
| left | Выравнивание по левому краю |
| right | Выравнивание по правому краю |
| internal | Заполнение между знаком или основанием числа и самим числом |
| dec | Перевод в десятичную систему счисления |
| oct | Перевод в восьмеричную систему счисления |
| hex | Перевод в шестнадцатеричную систему счисления |
| boolalpha | Перевод логического 0 и 1 соответственно в "false" и "true" |
| snowbase | Выводить индикатор основания системы счисления (0 для восьмеричной, 0x для шестнадцатеричной) |
| snowpoint | Показывать десятичную точку при выводе |
| uppercase | Переводить в верхний регистр буквы X, E и буквы шестнадцатеричной системы счисления (ABCDEF) (по умолчанию - в нижнем регистре) |
| showpos | Показывать "+" перед положительными целыми числами |
| scientific | Экспоненциальный вывод чисел с плавающей запятой |
| fixed | Фиксированный вывод чисел с плавающей запятой |
| unitbuf | Сброс потоков после вставки |
| stdio | Сброс stdout, sterror после вставки |

Для установки флагов к ним обращаются с использованием имени класса.

**Маски**

маска adjustfield для файла выравнивания;

маска basefield для флага системы счисления;

маска floatfield для флагов с плавающей точкой.

Пример. Использование флагов.

#include <iostream>

using namespace std;

int main(){

int x=100;

double y=178,34;

cout.setf(ios::hex ios::basefield);

cout.setf(ios::uppercase);

cout<<x<<endl;//64

cout.setf(ios::snowbase);

cout<<x<<endl;//0X64

cout<<x<<endl;//100

cout.unsetf(ios::hex);

cout<<x<<endl;//100

cout.setf(ios::snowpos);

cout.setf(ios::scientific, ios::floatfield);

cout<<y<<endl;//+1.783400E+002

cout.unsetf(ios::scientific);

cout<<y<<endl;//+178.34

return 0;

}

**Манипуляторы** - это инструкции форматирования, передаваемые непосредственно в поток. Манипулятор определяет форматирование только для одного данного, находящегося за ним в потоке.

**iomanip** - библиотека, в которой находятся манипуляторы.

|  |  |
| --- | --- |
| Манипуляторы | Назначение |
| ws | Удаление из входного потока пробельных символов (пробел, символы табуляции, перевода строки, возврата каретки, символ перевода страницы) |
| dec | Перевод в десятичную систему счисления |
| oct | Перевод в восьмеричную систему счисления |
| hex | Перевод в шестнадцатеричную систему счисления |
| setfill (символ заполения) | Устанавливает символ заполнения (по умолчанию пробел) |
| setprecision (точность) | Устанавливает точность (число выводимых знаков) |
| setiosflags (флаги форматирования) | Устанавливает указанные флаги форматирования |
| resetiosflags (флаги форматирования) | Сбрасывает указанные флаги форматирования |

Пример. Создание собственных манипуляторов.

#include <iostream>

#include <iomanip>

using namespace std;

ostream& Wrt(ostream &p){

p<<setioflags(ios::fixed)<<setw(8)<<setprecision(3);

return p;

}

class myx{

double x;

public:

myx(double a):x(a) {}

friend ostream& operator<<(ostream &p, myx ob)

{p<<ob.x<<endl;

return p;

}

};

int main(){

myx x=12.21389;//12.214

cout<<Wrt<<x;

return 0;

}

**Класс istream**

Класс предназначен для извлечения данных (чтение из оперативной памяти). Большинство методов предназначены для работы с потоком cin. Некоторые методы класса:

|  |  |
| --- | --- |
| >> | Операция извлечения данных потока |
| get(ch) | Чтение одного символа (имеются перегруженные функции)  Пример:  cin.get(ch) // или (cin>>ws).get(ch);  ch=cin.get() // или ch=(cin>>ws).get(); |
| get(str, n, ch) | Помещает в нультерминальную строку str n-1 символов строки или до символа ch (имеются перегруженные функции)  Пример:  cin.get(str, 7);//Ввод в str 6 символов  cin.get(str, 10, '\n');//Ввод строки до символа "n" (нажатия на клавишу Enter)  cin.get(str, 10, '0');//Ввод строки до символа "0" или 9 символов |
| ignore(n, ch) | Удаляет символы до ограничителя ch или до n символа включительно |
| peek(ch) | Читает один символ, оставляя его в потоке  Пример:  ch=cin.peek(); |
| putback(ch) | Вставляет обратно во входной поток последний прочитанный символ  Пример:  cin.putback(ch); |
| gcount() | Возвращает число символов, считанных с помощью последнего вызова функций get(), getline(), read();  Пример:  n=cin.gcount(); |
| << | Операция вставки в поток |
| put(ch) | Вставка символа ch в поток  Пример: cout.put(ch); |
| flush() | Очистка буфера  Пример:  cout.flush(); |

**Класс ostream**

Предназначен для вывода данных (помещения в поток).

|  |  |
| --- | --- |
| << | Операция вставки в поток |
| put(ch) | Вставка символа ch в поток  Пример: cout.put(ch); |
| flush() | Очистка буфера  Пример:  cout.flush(); |

**Потоковые объекты**

Используются 4 стандартных потоковые объекта:

|  |  |
| --- | --- |
| сin | ввод с клавиатуры; |
| сout | вывод на экран; |
| сerr | вывод на экран, минуя этап буферизации; данный поток нельзя перенаправлять; используется для вывода сообщений об ошибках; |
| сlog | вывод на экран; не может быть перенаправлен, но проходит буферизацию; используется для ведения журналов. |

Пример. Определение операторов ввода-вывода для пользовательского класса.

class Cls{

public:

int s;

};

istream& operator>>(istream& vvod,Cls& m)

{

cout<<"Enter value:";

vvod>>m.s;

return vvod;

}

ostream& operator<<(ostream& vivod, Cls& m)

{

vivod<<"value: "<<m.s<<endl;

return vivod;

}

void main(){

Cls x;

cin>>x;

cout<<x;

}

**Контроль исключительных ситуаций ввода-вывода**

Для получения информации о состоянии потока используется функция int rdstate();

Возможные состояния потока:

Обращение флагов осуществляется посредством ссылок на базовый класс:

ios::goodbit

|  |  |  |
| --- | --- | --- |
| Флаг | Значение | Описание |
| goodbit | 0x00 | Ошибок нет |
| eofbit | 0x01 | Достигнут конец файла |
| failbit | 0x02 | Операция не выполнена (без потери символов) |
| badbit | 0x04 | Недопустимая операция (потеря данных) |

**Функции проверки флагов ошибок**

|  |  |
| --- | --- |
| Функция | Возвращаемый результат |
| int eof() | Возвращает true, если достигнут конец файла (флаг eofbit) |
| int fail() | Возвращает true в случае неудачной операции (флаги failbit, badbit) |
| int bad() | Возвращает true, если имеется ошибка (флаги badbit) |
| int good() | Возвращает true, если ошибок не было |
| clear() | Снимает все флаги ошибок (при использовании без аргументов) или устанавливает указанный флаг (например, clear (ios::failbit)) |

После возникновения любой ошибки операции над потоком игнорируются до сброса флагов ошибок.

Наличие ошибок можно проверить с использованием конструкции: if(cin.fail()) cout<<"Error";

Пример. Ввод данных с контролем исключительных ситуаций.

int main(){

int i;

while(true)

{

cout<<endl<<"Enter an integer:";

cin>>i;

if(cin.good()) break;//Операция прошло без ошибок

cout<<"Data input error:";

switch(cin.rdstate()){//Получение информации о состоянии потока

case ios::failbit:

cout<<"The operation failed"; break;//Операция не выполнена

case ios::badbit:

cout<<"Error due to the failure"; break;//Сбой

}

cin.clear();//Сброс битов ошибок

cin.ignore(20,'\n');//Очистка строки

}

cout<<"The number entered:"<<i<<endl;

return 0:}

**Связанные потоки (tied ostream)**

С потоком ввода может быть ассоциирован связанный поток вывода, который позволяет выполнять логически корректные операции ввода-вывода. При связавании потоков их буферы синхронизируются таким образом, что содержимое буфера выходного потока будет очищаться перед другой операцией в другом потоке данных. Каждый поток ввода в любой момент времени может быть связан только с одним потоком вывода. Но поток вывода может быть связан с несколькими потоками ввода.

Управление связывание реализуется с помощью функций:

1) ostream\* tie() - Возвращает указатель на выходной поток, связанный с входным потоком.

2) ostream\* tie(ostream\*) - Связывает поток с заданным входным потоком и возвращает указатель на предыдущий связанный входной поток.

По умолчанию стандартный поток ввода связан со стандартным потоком вывода:

cin.tie(&cout);

Такая связь гарантирует, что в случае такого ввода данных:

cout<<"Vvedite x:";

//flush();

cin>>x;

- автоматически будет добавлена функция очистки буфера: flush();

Вызов функции с нулевым параметром cin.tie(0) приводит к разрыву связей между потоками. В этом случае у потоков устанавливается асинхронный режим обмена данными, который ускоряет работу программы, так как отменяется операция очистки буфера.

**Потоковый ввод-вывод файлов**

**fstream** - ввод-вывод из/в файл.

**ifstream** - чтение данных из файла.

**ostream** - запись данных в файл.

Объекты данных классов могут создавать свои потоки.

имя\_объекта() - Создание потока без соединения с файлом. Для соединения с файлом требуется использовать метод open.

имя\_объекта(const char \*имя\_файла, int режим\_открытия) - Создание потока, связанного с указанным файлом, и задание режима открытия.

Конструктор потока или метод open содержит следующее значение по умолчанию для различных потоков:

ofstream(const char \*имя\_файла, int mode=ios::out);

ifstream(const char \*имя\_файла, int mode=ios::in | ios::out);

fstream(const char \*имя\_файла, int mode=ios::in | ios::out);

**Режим открытия**

|  |  |
| --- | --- |
| ios :: in | Создаётся поток для чтения данных из файла |
| ios :: out | Создаётся поток для записи данных в файл (режим по умолчанию) |
| ios :: ate | При создании потока указатель перемещается в конец файла, запись ведётся в текущую позицию |
| ios :: app | При записи данные добавляются в конец файла |
| ios :: trunc | При создании файла его содержимое уничтожается (режим по умолчанию) |
| ios :: binary | Двоичный режим работы (по умолчанию текстовый режим работы) |

Для задания нескольких режимов используется операция **или (побитовая)**. Если файл создан удачно, то объект возвращает значение true, либо можно использовать метод bool is\_open().

Поток - объект класса.

Пример. Ввод строки в файл и чтение строки из файла.

#include <iostream>

#include <fstream>

using namespace std;

int main()

{

ofstream ofl;//Поток для ввода данных в файл

ofl.open("MyText.txt");//Создание файла

if(!ifl.is\_open())//Если файл не создан

{

cout<<"Error creating file"<<endl;

return 1;

}

ofl<<"File created";//Ввод данных в файл

ofl.close();//Закрытие файла

ifstream ifl;//Поток для вывода данных из файла

ofl.open("MyText.txt");//Открытие файла

if(!ifl)//Для разнообразия

{

cout<<"Error reading file"<<endl;

return 2;

}

char str1[30], str2[30];

ifl>>str1>>str2;//Чтение стоки из файла

cout<<str1<<" "<<str2<<endl;

ifl.close();

return 0;

}

**Методы класса ifstream**

|  |  |
| --- | --- |
| Метод | Описание |
| getline (str, n, ch) | Читает символьную строку из текстового файл или данные из бинарного файла до указанного n количества элементов или до символа-ограничителя ch. |
| read (str, n) | Считывает из файла в нультерминальную строку str до заданного n числа символов или до конца файла. |
| eof() | Возвращает ненулевое значение (true), если указатель потока достиг конца файла. |
| seekg(pos) | Устанавливает указатель в позицию pos от начала файла. |
| seekg (pos, seek\_dir) | Перемещает текущую позицию в файле на pos байтов от позиции, определяемой параметром seek\_dir: ios::beg (от начала файла), ios::cur (от текущей позиции), ios::end (от конца файла). |
| tellg (pos) | Возвращает текущую позицию указателя от начала файла. |

**Методы класса ofstream**

|  |  |
| --- | --- |
| Метод | Описание |
| write (str, n) | Записывает n символов из строки str в файл. |
| seekp (pos) | Устанавливает указатель в позицию pos от начала файла. |
| seekp (pos, seek\_dir) | Перемещает текущую позицию в файле на pos байтов от позиции, определяемой параметром seek\_dir: ios::beg (от начала файла), ios::cur (от текущей позиции), ios::end (от конца файла). |
| tellp() | Возвращает текущую позицию указателя от начала файла. |

**Текстовый режим работы**

Особенности:

1) Операция помещения в поток позволяет записывать в файл любые стандартные типы данных.

2) Числа записываются в виде последовательности символов, а при чтении приводятся к своему обычному представлению.

3) Текстовый режим менее эффективен при хранении чисел.

4) При записи нескольких чисел в одну строку их надо отделять пробелами.

5) При чтении операция извлечения данных из файла читает строку до первого пробела, поэтому при записи следует устанавливать символ-ограничитель (обычно переход на новую строку), а при чтении - использовать метод getline().

Пример. Ввести в файл и вывести на экран данные различных типов.

#include <iostream>

#include <fstream>

#include <ctype.h>//Макросы для определения типа

using namespace std;

int main()

{

ofstream ofl;//Создание потока для ввода данных в файл

ofl.open("MyText.txt");//Создание файла

if(!ofl) return 1;//Если файл не создан

ofl<<"Two words"<<234<<'w'<<55<<endl;

ofl.close();//Закрытие файла

ifstream ifl;//Создание потока для вывода данных из файла

ifl.open("MyText.txt");//Открытие файла

if(!ifl) return 2;//Если файл не открыт

char str[50];

ifl>>str;//Чтение строки из файла

cout<<str<<endl;

ifl.seeg(0);//Возврат указателя в начало файла

ifl.getline(str, 50, "\n");

cout<<str<<endl;//Twowords234w55

ifl.seekg(0);

char ch, \*uk;

while(!ifl.eof())

{

uk=str;

ch=ifl.peek();//Чтение одного символа без извлечения из потока

if(isdigit(ch))//Если прочитанный символ - цифра

{

while(isdigit(\*uk=ifl.get()) && !ifl.eof()) uk++;

ifl.putback(\*uk);

\*uk='\0';

cout<<str<<endl;//Вывод числа

}

else//Если не цифра

{

int k=1;

str[0]=ifl.get();

while(!isdigit(ifl.peek()) && !ifl.eof()) str[k++]=ifl.get();

str[k]='\0';

cout<<str<<endl;

}

}

ifl.close();

return 0;

}

**Двоичный режим работы**

read - \*char

write - const \*char

Функция read в качестве первого аргумента использует значение типа char, а write - типа const char. Поэтому необходимо преобразование указателей на используемые типы данных. Для этого используется оператор явного преобразования типов reinterpret\_cast:

istream& istream::read(reinterpret\_cast<char\*>(&st), streamsize n);

ostream& ostream::write(reinterpret\_cast<const char\*>(&st), streamsize n);

Пример. Создать класс для хранения и сортировки информации в файле.

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct Tstud{//Информация о студенте

char fio[30];//ФИО студента

int ngr;//Номер группы

int oop;//Оценка по ООП

Tstud vvod(){

Tstud st;

cout<<endl<<"Information about the students (fio, ngr, otc): ";

cin>>st.fio>>st.ngr>>st.oop;

return st;

}

}stud;

int nz=sizeof(Tstud);//Размер одной структуры

class CStud

{

char flname[20];//Имя файла

fstream \*fl;//Указатель на поток

public:

CStud (char s[]){

strcpy(flname, s);

fl=new fstream();//Создание потока

}

~CStud{

fl->close();

delete fl;

}

bool Newfl();//Создание файла

boolReadfl (Tstud &);//Чтение одного элемента из файла

void Rewfl();//Переход в начало файла

void Addfl(Tstud);//Запись 1 элемента в файл

void Sortfl();//Сортировка по баллу

const char\* Getflname();//Чтение имени текущего файла

};

bool CStud::Newfl()//Создание нового файла для ввода-вывода

{

fl->open(flname, ios::in | ios::out |ios::binary | ios::trunc);//Для чтения, записи, бинарный, новый

if(!fl) return false;

return true;

}

bool CStud::Readfl(Tstud &p)//Чтение одного элемента из текущей позиции файла

{

fl->read(reinterpret\_cast<char\*>(&p), nz);//Чтение

if(fl->good()) return true;

fl->clear();//Сброс битов ошибок

return false;

}

void CStud::Rewfl()//Перемещение указателей в начало потока

{

fl->seekg(0);//Перемещение указателя в начало файла (для операции чтения)

fl->seekp(0);//Перемещение указателz в начало файла (для операции записи)

}

void CStud::Addfl(Tstud stud)//Запись 1 элемента в файл

{

fl->write(reinterpret\_cast<char\*>(&stud), nz);

fl->flush(0);//Очистка буфера

}

void CStud::Sortfl(Tstud stud)//Сортировка в файле

{

fl->seekp(0, ios\_base::end);//Переход в конец файла

int nb=fl->tellp();//Определение размера файла

int n=nb/nz;//Количество студентов

Tstud stud01, stud02;

for(int i=0; i<n-1; i++)

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

{

fl->seekg(i\*nz, ios::beg);//Сдвиг позиции

fl->read(reinterpret\_cast<char\*>(&stud01), nz);

fl->seekg(j\*nz, ios::beg);//Сдвиг позиции

fl->read(reinterpret\_cast<char\*>(&stud02), nz);

if(stud01.opp>stud02.oop)

{

fl->seekp(i\*nz, ios::beg);

fl->write(reinterpret\_cast<char\*>(&stud02), nz);

fl->seekp(j\*nz, ios::beg);

fl->write(reinterpret\_cast<char\*>(&stud01), nz);

}

}

fl->flush();//Выгрузка потока в файл

}

const char\* CStud::Getflname()//Определение имени текущего файла

{

return this->flname;

}

ostream& operator<<(ostream & vivod, CStud &s)

{

Tstud p;

vivod<<"File"<<s.Getflname()<<endl;

s.Rewfl();//Помещение в начало файла

while(s.Readfl(p))

vivod<<p.fio<<''<<p.ngr<<''<<p.oop<<endl;

return vivod;

}

int main()

{

CStud myfile("stud.dat");

if(!myfile.Newfl())

{

cout<<"Error creating file";

return 1;

}

int n;

cout<<endl<<"Enter the number of students: ";

cin>>n;

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

myfile.Addfl(stud.vvod());//Ввод в файл

cout<<myfile<<endl;//Вывод файла

myfile.Sortfl();//Сортировка

cout<<myfile<<endl;

}

Разрешено использовать методы, записывающие в файл объекты класса. Каждый объект находится в своей области памяти, поэтому при чтении и записи следует использовать указатель this. Для хранения множества объектов обычно используются статические функции, которые применяются к базовому классу и наследуемым классам.

Для определения размера объекта во время выполнения программы используется оператор **typeid (объект)**, находящийся в библиотеке typeinfo.lib.

Перегруженные операторы == и != обеспечивают сравнение типов объектов. Если оператор typeid применяется к указателю на объект базового класса, она автоматически учитывает тип объекта, в том числе при работе с производными классами.

Пример. Написать класс, хранящий в файле свои объекты и объекты порождённых классов. Адреса объектов должны хранится в массиве указателей, который является статической переменной.

\*

#include <iostream>

#include <fstream>

#include <typeinfo>

using namespace std;

enum Clstype{Class01, Class02};

class BCls

{

private:

double x;

static int n;//Количество объектов

static BCls\* umas[];//Массив указателей на объекты

public:

BCls(){n=0;}

virtual void vvod()

{

cout<<"Enter x: ";

cin>>x;

}

virtual void vivod()

{

cout<<endl<<"x="<<x;

}

static void adds();//Добавить элемент

static void shows();//Вывести данные

static void readfl();//Чтение из файла

static void addfl();//Запись в файл

virtual Clstype gettype();//Получить тип

};

int BCls::n=0;

BCls\* BCls::umas[20];

class Cls01:public BCls

{

private:

double y;

public:

virtual void vvod()

{

BCls::vvod();

cout<<"Enter y: ";

cin>>y;

}

virtual void vivod()

{

BCls::vivod();

cout<<endl<<"y="<<y;

}

};

class Cls02:public Cls01

{

private:

double z;

public:

void vvod()

{

Cls01::vvod();

cout<<"Enter z: ";

cin>>z;

}

};

void BCls::adds()

{

char ch;

cout<<"\n 'y' to enter x, y:"

"\n 'z' to enter x, y, z:"

"\n SELECT:";

cin>>ch;

switch(ch)

{//Создать объект указанного типа

case 'y': umas[n]=new Cls01; break;

case 'z': umas[n]=new Cls02; break;

default: cout<<endl<<"Error"; return;

}

umas[n++]->vvod();//Ввести данные

}

Clstype BCls::gettype()//Определение типа объекта

{

if (typeid(\*this)==typeid(Cls01)) return Class01;

else if (typeid(\*this)==typeid(Cls02)) return Class02;

else{cerr<<"Error"<<endl; exit(1);}

}

void BCls::shows()

{

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

{

switch(umas[i]->gettype())//Вывести тип

{

case Class01:cout<<"Class01:"; break;

case Class02:cout<<"Class02:";break;

default: cout<<"Error";

}

umas[i]->vivod();//Вывод данных

cout<<endl;

}

}

void BCls::addfl()//Запись в файл объектов

{

int size;

cout<<"Writing into file"<<endl;

ofstream ouf;

Clstype ctype;//Тип объекта

ouf.open("myfile.dat", ios::binary | ios::trunc);

if(!ouf)

{

cout<<"Error creating file"; return;

}

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

{

ctype=umas[j]->gettype();//Получить тип объекта

ouf.write(reinterpret\_cast<char\*>(&ctype), sizeof(ctype));

switch(ctype)

{

case Class01:size=sizeof(Cls01); break;

case Class02:size=sizeof(Cls02); break;

}

ouf.write(reinterpret\_cast<char\*>(umas[j]), size);//Запись объекта в файл

if(!ouf)

{

cout<<"Error"<<endl;

return;

}

}

}

void BCls::readfl()//Чтение из файла

{

int size;

Clstype ctype;

ifstream inf;

inf.open("myfile.dat", ios::binary);

if(!inf)

{

cout<<"Error"<<endl; return;

}

n=0;

while(true)

{

inf.read(reinterpret\_cast<char\*>(&ctype), sizeof(ctype));//Определение типа объекта

if(!inf.eof()) break;//Выход, если достигнут конец файла

if(!inf)

{

cout<<"Error"<<endl;

return;

}

switch(ctype)//Выделение памяти для хранения объектов

{

case Class01:umas[n]=new Cls01;

size=sizeof(Cls01); break;

case Class02:umas[n]=new Cls02;

size=sizeof(Cls02); break;

default: cout<<"Error"; return;

}

inf.read(reinterpret\_cast<char\*>(umas[n]), size);

if(!inf)

{

cout<<"Error"<<endl;

return;

}

n++;

}

cout<<"Reading"<<endl;

}

int main()

{

char ch;

while(true)

{

cout<<"\n 'a' - Add"

"\n 'p' - Show"

"\n 'w' - Write into file"

"\n 'r' - Read from file"

"\n 'q' - Exit"

"\n SELECT: ";

cin>>ch; system("cls");

switch(ch)

{

case 'a':BCls::adds();break;

case 'p':BCls::shows();break;

case 'w':BCls::addfl();break;

case 'r':BCls::readfl();break;

case 'q':exit(0);break;

default: cout<<"Error"<<endl;

}

}

return 0;

}