# Наследование и композиция

## 1. Порядок вызова конструкторов и деструкторов

Необходимость в явном вызове конструкторов в списке инициализирующих значений возникает достаточно часто, но вам никогда не понадобится явно вызывать деструкторы, поскольку в каждом классе может быть только один деструктор, и он вызывается без аргументов.   
Тем не менее компилятор по-прежнему гарантирует вызов всех деструкторов, то есть всех деструкторов во всей иерархии наследования, начиная с последних классов в цепочке наследования и до корневого базового класса.   
Конструкторы и деструкторы принципиально отличаются от других функций: они вызываются все от начала и до конца иерархии, тогда как при вызове обычной функции вызывается только эта функция, но не ее версии из базовых классов.   
 Если вы хотите вызвать базовую версию обычной *переопределенной* функции, это придется сделать явно.   
Давайте разберёмся в порядке вызова конструкторов и деструкторов при большом количестве подобъектов. Знание порядка вызовов конструкторов и деструкторов может помочь при определении причин «непонятных» ошибок в программе и причин «утечки» памяти.   
Чтобы точно понять, как это происходит, выполните в пошаговом режиме с «заходом» в вызываемые функции, следующий пример:

//L9:Order.срр

// Порядок вызова конструкторов и деструкторов

#include <fstream>

using namespace std;

ofstream out("order.out");

template <class T>class T{

public:

int m;

T(int n):m(n) {out <<typeid(this).name()<< " constructor\n";}

~T(){out <<typeid(this).name()<< " destructor\n";}

};

class Base1;

class Member1;

class Member2;

class Member3;

class Member4;

class Derived1 : public T<Base1> {

T<Member1> m1;

T<Member2> m2;

public:

Derived1(int n1=1,int n2=2) : m2(n1), m1(n2), T<Base1>(3) {

out << "Derived1 constructor\n";

}

~Derived1() {

out <<" Derived1 destructor\n";

}

};

class Derived2 :public Derived1 {

T<Member3> m3;

T<Member4> m4;

public: Derived2() : m3(1), Derived1(2), m4(3) {

out << "Derived2 constructor\n";

}

~Derived2() {

out << "Derived2 destructor\n";

}

};

int main() {

Derived2 d2;

}

//===================================  
Справка. Оператор *typeid* .  
В языке C++ оператор *typeid* возвращает ссылку на объект *type\_info*, описывающий тип объекта, адрес которого указывается в круглых скобках.   
Общая форма записи оператора *typeid* такова:   
typeid(&*объект*)

Оператор *typeid* поддерживает в языке C++ возможность идентификации динамической информации о типе (RTTI - run-time type information).

Класс *type\_info* определяет следующие открытые члены.

bool operator == (const type\_info &ob) const;

bool operator != (const type\_info &ob) const;

const char \*name() const;

Перегруженные операторы == и != служат для сравнения типов. Функция name() возвращает указатель на имя типа, *установленного компилятором*.

Если оператор typeid применяется к указателю полиморфного класса, он автоматически возвращает тип объекта, на который он указывает (полиморфный класс — это класс который содержит хотя-бы одну виртуальную функцию.) Следовательно, оператор typeid можно использовать для определения типа объекта, адресуемого указателем на базовый класс.  
//====================================

Сначала мы создаем объект ofstream для записи всех результатов в файл. Затем для уменьшения объема работы мы определяем шаблон для построения классов, которые затем будет использоваться при наследовании и композиции.

Каждый конструктор и деструктор выводит сообщение о своем вызове в файл. Обратите внимание: конструкторы не являются конструкторами по умолчанию; у них имеется аргумент типа int. Этот аргумент используется ещё и для того, чтобы заставить нас явно вызывать *перегруженные* конструкторы в списке инициализирующих значений. (Если мы перегрузили хотя бы один конструктор, то необходимо создать и конструктор «по умолчанию», иначе его не будет).  
 Если в базовом классе отсутствует конструктор «по умолчанию», и вы явно не вызываете конструктор базового класса в списке инициализации конструктора производного класса, то подобъект базового класса в производном классе не будет инициализирован.   
 Если в классе есть конструктор «по умолчанию», то он вызовется «неявно».

Объясните результат и последовательность выполнения программы в пошаговом режиме.



2. Разработать глобальную функцию  
bool comparator1(DBType type, void \*obj1, Condition condition, void \*obj2).

Библиотека dbmsLib, помимо методов классов DBTable, DBTableTxt, DBTableBin, DBTableSet, DBDate, содержит

Это должна быть обобщённая функция, которая могла бы сравнивать два объекта любого из типов, определённых в перечислении DBType, по различным условиям, определённым в перечислении Condition.   
Функция будет использоваться в методе SelfRows() для выборки из таблицы строк, в которых значение поля в указанном столбце удовлетворяет заданным условиям.

Указание.   
1. При разработке метода   
bool comparator1(DBType type, void \*obj1, Condition condition, void \*obj2)   
использовать приём «двойная диспетчеризация», который реализуется с помощью вложенных операторов switch следующим образом:   
switch (type){  
case Int32: switch (condition){  
 case Equal: return \*(int\*)obj1 == \*(int\*)obj2 ;  
 case NotEqual: return \*(int\*)obj1 != \*(int\*)obj2 ;  
 . . .  
 default: cout<<"Недопустимая операция сравнения\n"<<endl;  
 return false;  
 }

case Double: switch (condition){  
 case Equal: return … ;  
 case NotEqual: return … ;

. . .  
 default: cout<<"Недопустимая операция сравнения\n"<<endl;  
 return false;  
  
 ………  
 }

…

}

default: cout<<"Недопустимый тип данных\n"<<endl;  
 return false;

}

2. При сравнении строк исключать незначащие ведущие пробелы.   
Для упрощения переопределения виртуальных функций интерфейса в классе DBTableBin, данные, имеющие тип String, должны заканчиваться байтом 00. Например, бинарный файл с таблицей Books, в начало которого в поля размером по 24 байта записаны имя таблицы и имя первичного ключа, должен начинаться так:  
18 00 00 00 42 6F 6F 6B 73 00 FE FE FE FE FE FE |....Books.......  
FE FE FE FE FE FE FE FE FE FE FE FE 18 00 00 00 |................   
42 6F 6F 6B 49 44 00 FE FE FE FE FE FE FE FE FE |BookID..........  
FE FE FE FE FE FE FE FE |........

Байты между 00 и 24 (от начала поля) не используются и могут хранить любые значения, не обязательно FE. Поле, хранящее BookID, начинается с байта, порядковый номер которого в файле равен 4 + 24 + 4 = 32.

## 3. Переопределить виртуальную функцию DBTable\* SelfRows(string colName, Condition cond, void\* value) для классов DBTableTxt и DBTableBin

При переопределении виртуальной функции SelfRows() в классе DBTableTxt вызов функции bool comparator1(DBType type, void \*obj1, Condition condition, void \*obj2)  
 из SelfRows() будет выглядеть так:

DBTableTxt\* tab=new DBTableTxt(tableName);

tab->columnHeaders=columnHeaders;

for(unsigned int i=0; i<data.size(); i++)   
 if(comparator(columnHeaders[colName].colType,data[i][colName],cond, value))

tab->data.push\_back(data[i]);

A при переопределении виртуальной функции SelfRows() в классе DBTableBin вызов функции bool comparator1(DBType type, void \*obj1, Condition condition, void \*obj2)   
из SelfRows() будет выглядеть так:  
 Header hdr=GetHeader();

DBTableBin\* tab=new DBTableBin(tName,hdr,primaryKey;

for(unsigned int i=0; i<nRows; i++){  
 if(comparator(hdr[colName].colType,

GetValue(data[i]+FieldPosition(colName),colName,hdr),cond, value)){

Row row=\*new Row(GetRow(i));

tab->AddRow(row,tab->GetSize());  
 }  
 }