# ЛАБОРАТОРНОЙ РАБОТЫ № 3

По дисциплине: Объектно-ориентированное программирование

Тема работы: **Классы с динамическими полями. Создание, инициализация и уничтожение динамических объектов. Динамические массивы объектов и массивы указателей на объекты.**

Цель работы: Научить работать с динамическими объектами.

Количество часов: 2ч.

## Содержание работы (Задание, Задачи):

**Вариант 1:**

1. Написать программу, создающую массив из пяти динамических объектов класса c), доступных через инициализированный массив указателей на эти объекты и неинициализированный статический объект класса d) (из лаб. работы №1).
2. Построить класс «матриц». Реализовать метод сложения матриц.

**Вариант 2:**

1. Написать программу, создающую динамический массив из трех объектов класса c) и инициализированный статический объект класса d) (из лаб. работы №1).
2. Построить класс «матриц». Реализовать метод умножения матрицы на число.

**Вариант 3:**

1. Написать программу, создающую массив из четырех динамических объектов класса c), доступных через неинициализированный массив указателей на эти объекты и статический неинициализированный объект класса d) (из лаб. р. №1).
2. Построить класс «матриц». Реализовать метод транспонирования матрицы.

**Вариант 4:**

1. Написать программу, создающую неинициализированный динамический объект класса c) и инициализированный статический массив из двух объектов класса d) (из лабораторной работы №1).
2. Построить класс «стек».

**Вариант 5:**

1. Написать программу, создающую неинициализированный статический массив из трех объектов класса c) и инициализированный динамический объект класса d) (из лабораторной работы №1).
2. Построить класс «очередь».

**Методические указания по выполнению:**

## I. Классы с динамическими полями.

По правилам языка C++ объекту некоторого класса память может быть выделена статически – на этапе компиляции или динамически – во время выполнения программы.

Наиболее часто динамическое распределение памяти применяют в классах,

использующих в качестве полей массивы, строки, структуры и их комбинации. В этом случае поле содержит указатель на переменную соответствующего типа.

При создании классов с динамическими полями необходимо учитывать большое количество особенностей. Так конструктор такого класса обычно осуществляет выделение участков памяти требуемого размера под переменные, адреса которых присваиваются соответствующим указателям, и обеспечивает контроль наличия доступной памяти.

Деструктор класса соответственно должен при уничтожении объекта освобождать

распределенную при конструировании память, поскольку автоматически эта память не освобождается.

При этом, если в классе предусмотрен неинициализирующий конструктор и в программе по какой-то причине динамические поля могут быть не созданы, то в деструкторе нужно проверять факт выделения памяти под поля прежде, чем ее освобождать.

. Кроме того, для инициализации полей объекта, созданного с помощью неинициализирующего конструктора, следует предусмотреть метод класса, который выделял бы память под поля такого объекта и инициализировал их.

Дополнительные проблемы могут возникнуть при необходимости создания копии

объекта, например при передаче объекта в подпрограмму в качестве параметра-значения. В этом случае для реализации операции копирования класс с динамическими полями помимо обычного конструктора, в котором будет выделяться память под размещение динамического поля (например см. рис. 1.1, а), должен включать копирующий конструктор для корректного создания копии объекта. Это связано с тем, что при использовании стандартного копирующего конструктора адрес динамического поля просто копируется, и после этого мы получаем два объекта, использующих одни и те же

динамические поля (см. рис. 1.1, б). Соответственно при уничтожении копии объекта деструктором единое для двух объектов динамическое поле освобождается, оставляя основной объект без динамического поля (см. рис. 1.1, в). Теперь при любом обращении к этому полю мы получим сообщение об ошибке.

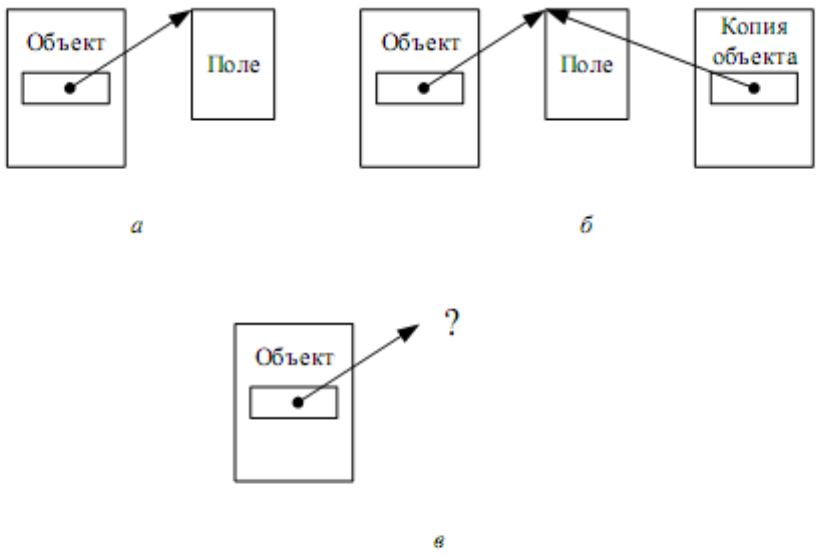


**Рис. 1.1. Объект с динамическим полем:**

*а* – после конструирования; *б* – после некорректного создания копии;

*в* – после удаления неверно созданной копии

***Пример 1. Работа с объектом, включающим динамическое поле.***

```
#include <stdio.h>
#include <conio.h>
class TNum
{
```

```
        public: int *pn;  // указатель для адреса динамического поля

        TNum(int n)                    // инициализирующий конструктор
        {   pn=new int(n);  }

        TNum(const TNum &Obj) // копирующий конструктор
        {   pn=new int(*Obj.pn);     }

        ~TNum()              // деструктор
        {   delete pn;  }
};

void Print(TNum b)         // подпрограмма с параметром объектом
{  printf("%d ",*b.pn); }

void main()
{
   TNum A(1);
   Print(A);
   _getch();
}
```

## II. Создание, инициализация и уничтожение динамических объектов.

При создании/уничтожении динамических объектов выделение и освобождение участков памяти осуществляется при выполнении операций *new* и *delete*.

При выделении памяти для отдельных объектов используют следующие формы обращения к операции *new:*

*<Имя указателя на объект>=new <Имя класса>;*

или

*<Имя указателя на объект>=new <Имя класса>(<Список параметров>);*

Вторую форму применяют при наличии у конструктора списка параметров.

Операция *delete* требует указания только имени объекта:

*delete <Имя указателя на объект>;*

Например:

*Num *a;* // объявление указателя на объект

*a = new Num;* // выделение памяти под неинициализированный объект

*delete a;* // освобождение выделенной памяти

В этом случае класс, под объекты которого выделяется память, должен содержать конструктор, вызываемый без указания аргументов, и при наличии защищенных или скрытых полей – метод инициализации невидимых извне полей.

Для создания инициализированных динамических объектов класс должен содержать инициализирующий конструктор с соответствующими параметрами:

*Num *b;* // объявление указателя на объект

*b = new Num(5);* // выделение памяти под инициализированный объект

*delete b;*

При освобождении памяти автоматически будет вызван деструктор класса, а при его отсутствии автоматически будет сгенерирован «пустой» деструктор вида:

*Num::~Num(){}*

***Пример 2. Использование динамических объектов со статическими полями.***

```
#include <locale.h>
#include <iostream>
#include <iomanip>
using namespace std;
```

```
class TVector
{
 private:    int x,y,z;
 public:
    TVector(){} // неинициализирующий конструктор
    TVector(int ax,int ay,int az) // инициализирующий конструктор
    {  x=ax;y=ay;z=az; }
    ~TVector(){} // деструктор
    void PrintVec();
    void SetVec(int ax,int ay,int az) // инициализирующий метод
    {  x=ax;y=ay;z=az; }
};
void TVector::PrintVec()
{
    cout<<"Значение вектора: "<<setw(5)<<x<<" , ";
    cout<<setw(5)<<y<<" , "<<setw(5)<<z<<"\n";
}
void main()
{
    setlocale(0,"russian");
    TVector *a,*b; //  два указателя на объекты класса
        // выделяем память под динамические объекты класса
    a=new TVector(12,34,23); // инициализированный объект
    b=new TVector;      // неинициализированный объект
    b->SetVec(10,45,56); // инициализация объекта
    a->PrintVec();    //  выводит: 12,  34,  23
    b->PrintVec();    //  выводит: 10,  45,  56
    // освобождаем память, выделенную под динамические объекты класса
    delete a;         // вызывает деструктор
```

```
    delete b;        // вызывает деструктор
    system("pause");
}
```

При работе с динамическими объектами следует помнить, что присваивание одного объекта другому с помощью указателей сводится к копированию адреса: после выполнения операции первый указатель содержит тот же адрес, что и второй. Старое значение в первом указателе стирается, и, если он содержал адрес некоторого объекта, то память, выделенная ранее под этот объект, остается занятой и более недоступной. Такая ошибка получила название «утечка памяти». К ошибке также приведет попытка освободить память по обоим указателям, так как один и тот же участок памяти, выделенный под объект, освобождается дважды.

## III. Динамические массивы объектов и массивы указателей на объекты

По требованию решаемой задачи из динамических объектов могут создаваться массивы. Это можно сделать тремя способами:

• создать динамический массив объектов – память под него выделяют одним непрерывным фрагментом равным объему всех объектов массива (рис. 1.2, а), например:

*B mas[] = new B[n];*

• создать статический массив указателей на объекты и затем динамически выделить память под элементы (рис. 1.2, б), например:

*B *mas[n];* // память под массив указателей выделена статически

*for (i=0; i<n; i++) mas[i]=new B;* // выделение памяти под объекты

• создать динамический массив указателей и затем также динамически выделить память под элементы (рис. 1.2, в), например:

*B **mas=new B *[n];* // память под массив указателей выделена динамически

*for (i=0; i<n; i++) mas[i]=new B;* // выделение памяти под объекты

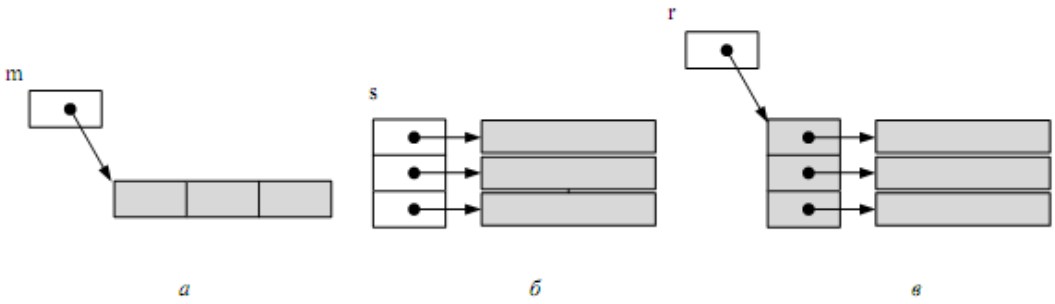


**Рис. 1.2.** Три способа организации динамических массивов объектов (серым выделены элементы структуры, размещенные в динамической памяти):
*а* – динамический массив объектов; *б* – статический массив указателей на динамически размещаемые объекты; *в* – динамический массив указателей на динамически размещаемые объекты

Освобождать выделенную память нужно так, как она была выделена:

- одним фрагментом – для динамического массива объектов:

  *delete[] mas;*

- поэлементно – для массива указателей на объекты:

  *for (i=0; i<n; i++) delete mas[i];*

- одним фрагментом, если память под массив указателей выделялась динамически:

  *delete [] mas;*

Различия между статическими и динамическими, инициализированными и неинициализированными объектами и массивами из них существенно влияют на синтаксис их описания и на работу с ними, поэтому эти различия важно хорошо понимать.

Определим класс Point, включающий два скрытых поля, инициализирующий и неинициализирующий конструкторы, инициализирующий метод и метод вывода содержимого полей на экран:

```
#include <iostream>
class Point
{
    private: int x,y;
    public:  Point(){}
```

```
        Point(int ax,int ay): x(ax),y(ay){}
        void SetPoint(int ax,int ay){  x=ax; y=ay;  }
        void Print(){  std::cout<<x<<" "<<y<<"\n";  }
};
```

*А теперь создадим объекты и массивы объектов различных типов.*

**А. Неинициализированный статический объект**

```
Point a;        // объявление объекта
a.SetPoint(5,10); // инициализация полей
a.Print();      // вывод содержимого полей на экран
```

Поскольку объект – статический память специально выделять и освобождать не надо, эта операция будет выполнена автоматически.

**Б. Инициализированный статический объект**

```
Point b(2,3);   // создание объекта
b.Print();      // вывод содержимого полей на экран
```

Объект сразу создается инициализированным, проблем с памятью также нет.

**В. Неинициализированный динамический объект**

```
Point *e;       // объявление неинициализированного указателя на объект
e=new Point(3,4); // выделение памяти и инициализация полей
e->Print();     // вывод содержимого полей на экран
delete e;       // освобождение памяти
```

Под объект необходимо отдельно выделить память, соответственно ее следует и освободить.

**Г. Инициализированный динамический объект**

```
Point *j=new Point(3,4); /* объявление указателя на объект,
                            выделение памяти и инициализация полей объекта */
j->Print();    // вывод содержимого полей на экран
delete j;              // освобождение памяти
```

**Д. Неинициализированный статический массив объектов**

```
Point c[4];              // объявление массива объектов
for(int i=0;i<4;i++)
{
        c[i].SetPoint(i*i,i-5); // инициализация полей
         c[i].Print();  // вывод содержимого полей на экран
}
```

**Е. Инициализированный статический массив объектов**

```
Point d[2]= {Point(2,4),Point(4,5)}; /* создание массива
             объектов и инициализация их полей */
for(i=0;i<2;i++) d[i].Print();// вывод содержимого полей на экран
```

**Ж. Неинициализированный динамический массив объектов**

```
Point *m=new Point[3];
for(i=0;i<3;i++)
{
        m[i].SetPoint(i,i+1);
        m[i].Print();
}
delete [] m;
```

**З. Неинициализированный статический массив указателей на объекты**

```
Point *s[3];
for(i=0;i<3;i++)
{
        s[i]=new Point(i,i+1);
        s[i]->Print();
}
for(i=0;i<3;i++) delete s[i];
```

**И. Инициализированный статический массив указателей на объекты**

```
Point *q[]= {new Point(2,7),new Point(1,5),new Point(4,2)};
for(i=0;i<3;i++)
{
        q[i]->Print();
}
for(i=0;i<3;i++) delete q[i];
```

## Пособия и инструменты:

Visual Studio 2008

## Вопросы для защиты лабораторной работы:

1. Что такое копирующий конструктор? Назовите случаи, когда использование такого конструктора обязательно.
2. Укажите различия между статическим и динамическим выделением памяти.
3. Посредством, каких операций происходит выделение и освобождение памяти при формировании динамических объектов?
4. Опишите три способа создания массивов динамических объектов.
5. Как следует освобождать память при разрушении динамического массива объектов/статического массива указателей на динамические объекты/динамического массива указателей на динамические объекты?

**Литература:**


1. Скляров В.А. Язык C++ и объектно-ориентированное программирование. – М.: Высшая школа., 1997.
2. Иванова Г.С., Ничушкина Т.Н., Пугачев Е.К. Объектно-ориентированное программирование: Учебник для ВУЗов. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2007.
3. Березин Б.И., Березин С.Б. Начальный курс С и C++. – М.: «Диалог-МИФИ», 1997.
4. Вайнер Р., Пинсон Л. C++ изнутри: Пер. с англ. – Киев: «ДиаСофт», 1993.
5. Дейтел Х., Дейтел П. Как программировать на C++: пер. с англ./ – М.: ЗАО «Издательство БИНОМ», 2005 г. – 1024 с.: ил.