## Лабораторная работа № 3

**Тема работы Наследование. Простое и множественное наследование**.

**Цель работы:** изучить принципы и получить практические навыки при использовании наследования классов; разработать иерархию нассследования классов; изучить вызов конструкторов и деструкторов при наследовании классов; дополнительно: рассмотреть случаи, когда необходимо использовать виртуальное наследование.

**Теоретические сведения:** рассмотрены в соответствующих разделах [1, 3−8].

**Наследование.**

**Наследование** – это механизм получения нового класса из существующего. Существующий класс может быть дополнен или изменен для создания производного класса. При создании нового класса вместо написания полностью новых данных и функций программист может указать, что новый класс должен наследовать данные и функции ранее определенного базового класса. Этот новый класс называется производным классом. Каждый производный класс сам является кандидатом на роль базового класса для будущих производных классов. При простом наследовании класс порождается одним базовым классом. При множественном наследовании производный класс наследуется несколькими базовыми классами. Производный класс обычно добавляет свои данные и функции, так что производный класс в общем случае больше своего базового.
Механизм наследования помогает сделать разработку более экономной и читаемой. Совокупность классов можно описать в виде такой иерархической структуры, что, если класс B наследует структуру и поведение класса A, то класс A называется базовым, а класс B – производным (рис. 1).

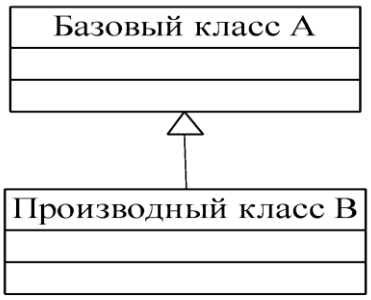


Рис. 1. Диаграмма наследования классов

Различают простое наследование и множественное. В первом случае производный класс имеет один базовый класс (рис. 2).

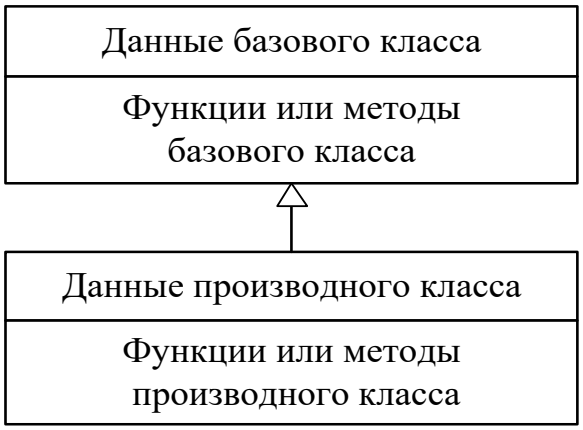


Рис. 2. Простое наследование классов

Шаблон объявления производного класса можно представить следующим образом:

```
ключ_класса имя_производного_класса :
необязательный_модификатор_доступа имя_базового_класса
{
    Тело производного класса
};
```

Пример объявления производного класса.

```
class Location                    // базовый класс
{
   int x,y;
   public:
         . .  .
};
class Point:public Location  // производный класс
{
   Тело класса Point
};
```

Класс Location является базовым и наследуется с атрибутом public. Класс Point – производный класс. Двоеточие (:) отделяет производный класс от базового. Атрибут класса (модификатор прав доступа) может задаваться ключевыми словами public и private. Атрибут может опускаться – в этом случае принимается атрибут по умолчанию (для ключевого слова class – private, для struct – public). Объединение (union) не может быть базовым или производным классом.

Модификатор прав доступа используется для изменения прав доступа к наследуемым элементам класса в соответствии с правилами, представленными в табл. 1.

Таблица 1. Модификаторы прав доступа при наследовании

| Ограничения на доступ в базовом классе | Модификатор наследования прав | Ограничения на доступ в производном классе |
|---|---|---|
| private | private | Нет доступа |
| protected | private | private |
| public | private | private |
| private | public | Нет доступа |
| protected | public | protected |
| public | public | public |

Отметим, что в производных классах права на доступ к элементам базовых классов не могут быть расширены, а только более ограничены.
Пример использования прав доступа к элементам базового класса.

```
#include<iostream>
using namespace std;
#include<string.h>
class A                          // базовый класс
{
   int a1;
   public:
   int a2;
   void f1();
};
class B:A                // производный класс
{
   int b1;
   public:
   void f1()
   {
      a1=1;          // ошибка, a1 – private-переменная класса A и доступна
                       // только для методов и дружественных функций
                       // собственного класса
      b1=0;          // доступ к переменной типа private из метода класса
      a2=1;          // a2 унаследована из класса A с атрибутом доступа private
                       // и поэтому доступна в методе класса
   }
};
int main()
{
   A a_ob1;          // объявление объекта a_ob1 класса A
   B b_ob1;          // объявление объекта  b_ob1 класса B
   b_ob1.a2+=1;  // ошибка, т. к. a2 private
   a_ob1.a2+=1;  // допустимая операция
   return 0;
}
```

Рассмотрим еще пример, демонстрирующий наследование прав доступа к элементам базовых классов.

```
#include<iostream>
using namespace std;
#include<string.h>
#define N 10
class book              // базовый класс
{
protected:
   char naz[20];          // название книги
   int kl;                // количество страниц
public:
   book(char *, int);     // конструктор класса book
   ~book();               // деструктор класса book
};
class avt: public book          // производный класс
{
   char fm[10];              // фамилия автора
   public:
   avt(char *, int, char *); // конструктор класса avt
   ~avt();                   // деструктор класса avt
   void see();
};
enum razd{teh, hyd,uch};   // перечисление

class rzd: public book
{
   razd rz;
   public:
   rzd(char *, int, razd);    // конструктор класса rzd
   ~rzd();                    // деструктор класса rzd
   void see();
};
book::book(char *s1, int i)
{
   cout << "\n Конструктор класса book" << endl;
   strcpy(naz,s1);
   kl=i;
}
book::~book()
{
   cout << "\nДеструктор класса book" << endl;
}
avt::avt(char *s1, int i, char* s2):book(s1,i)
{
   cout << "\n  Конструктор класса avt" << endl;
   strcpy(fm,s2);
```

```
}
 avt::~avt()
{
   cout << "\nДеструктор класса avt" << endl;
}
void avt::see()
{
   cout << "\nНазвание книги  " << naz << endl << "\nКоличество страниц " <<
   kl <<endl;
}
rzd::rzd(char *s1, int i, razd tp):book(s1,i)
{
   cout << "\n Конструктор класса rzd" << endl;
   rz=tp;
}
rzd::~rzd()
{
   cout << "Деструктор класса rzd" << endl;
}
void rzd::see()
{
   switch(rz)
   {
      case teh:  cout << "\n Раздел технической литературы" << endl; break;
      case hyd: cout << "\nРаздел художественной литературы" << endl;
      break;
      case uch: cout << "\n Раздел учебной литературы" << endl; break;
   }
}
int main()
{
   setlocale(LC_ALL,"Russian");
   avt av("Книга 1", 123, "автор 1");
   rzd rz("Книга 1", 123, teh);
   av.see();
   rz.see();
   return 0;
}
```

Если базовый класс имеет конструктор с одним или более аргументами, то любой производный класс должен иметь конструктор. В нашем примере конструктор класса book задан в виде

```
book::book(char *s1, int i)
{
   cout << "\nКонструктор класса book";
   strcpy(naz,s1);
   kl=i;
}
```

Теперь объявление объекта в функции main (либо в другой функции) может осуществляться

```
book my_ob("Дейтел", 1113);
```

В соответствии со сделанными выше замечаниями производный класс avt тоже должен иметь конструктор. В нашем примере он задан следующим образом:

```
avt::avt(char *s1, int i, char s2):book(s1,i)
{
   cout << "\n  Конструктор класса avt";
   strcpy(fm,s2);
}
```

**Конструкторы и деструкторы производных классов.** Если у базового и производного классов имеются конструкторы и деструкторы, то конструкторы выполняются в порядке наследования, а деструкторы − в обратном порядке. Общий синтаксис конструктора производного класса следующий:

```
конструктор_производного_класса(арг) : base(арг)
{
        тело конструктора производного класса
}
```

**Множественное наследование.**

В языке C++ имеется возможность образовывать производный класс от нескольких базовых классов. Общая форма такого наследования имеет вид

**class имя_произв_класса : имя_базового_кл 1,…,имя_базового_кл N**
**{ содержимое класса**
**};**

Иерархическая структура, в которой производный класс наследует от нескольких базовых классов, называется множественным наследованием. В этом случае производный класс, имея собственные компоненты, имеет доступ к protected- и public-компонентам базовых классов.
Конструкторы базовых классов при создании объекта производного класса вызываются в том порядке, в котором они указаны в списке при объявлении производного класса. Ниже приведен пример программы, использующей множественное наследование.

**Множественное наследование. Пример:** Разработать иерархию классов, реализующих множественное наследование. Первый базовый класс Firma содержит название фирмы, производный от него класс Otdel содержит информацию о табельном номере работника и номере отдела. Второй базовый класс Bank – информацию о наименовании банка и величине счета в банке. Класс, производный от этих двух классов, Work содержит фамилию работника. Все классы содержат функции просмотра полей.

```
#include <iostream>
#include <string.h>
#include <locale.h>
using namespace std;
class Firma
{
  protected:
       char naz[20];          //  название фирмы
  public:
      Firma(char *);
      ~Firma() { cout << "деструктор класса A" << endl; }
      void Firma_prnt();
};
class Otdel : public Firma
{
   protected:
       long tn;          // табельный номер
       int nom;          // номер подразделения
```

```
  public:
     Otdel(char*,long ,int);
     ~Otdel() { cout << "деструктор класса B1" << endl; }
     void Otdel_prnt();
};
class Bank
{
 protected:
     char naz[30];       // название банка
     double  schet;      // сумма денег
 public:
     Bank(char *, double);
     ~ Bank (){ cout << "деструктор класса B2" << endl; }
     void Bank_prnt();
};
class Work : public Otdel, public Bank
{
     char *fam;
 public:
     Work(char *,char *,long ,int ,char *,double) ;
     ~Work() { cout << "деструктор класса C" << endl; }
     void Work_prnt();
};
Firma::Firma(char *NAZ) {strcpy(naz,NAZ);}
Otdel::Otdel(char *NAZ, long TN,int NOM): Firma(NAZ),tn(TN),nom(NOM) {}
Bank::Bank(char *NAZ, double SCHET): schet(SCHET) {strcpy(naz,NAZ);}
Work::Work(char *FAM,char *NAZ1,long TN,int NOM,char *NAZ2,double ZP) :
        Otdel(NAZ1,TN,NOM), Bank(NAZ2,ZP)
{
  fam = new char[strlen(FAM)+1];
  strcpy(fam,FAM);
}
void Firma::Firma_prnt() {cout << naz << endl;}
void Otdel::Otdel_prnt()
{
  Firma::Firma_prnt();
  cout << " таб. N " << tn <<"  подразделение = " << nom <<endl;
}
void Bank::Bank_prnt()
{
     cout << " банк   : " << naz << endl;
     cout << " счет   = " << schet << endl;
}
void Work::Work_prnt()
{
  Otdel::Otdel_prnt();
  Bank::Bank_prnt();
  cout << " фамилия " << fam<<endl;
}
```

```
int main()
{
  Work rb("Иванов","предприятие",1234,2,"банк",555.6);
  Work *pt=&rb;  // указатель на созданный объект. Далее можно
                 // использовать для вызова методов либо
                 //  сам объект car, либо
                 // указатель на него pt
  setlocale(LC_ALL,"Russian" );
  rb.Otdel_prnt();  // вызов метода Otdel_prnt, используя объект
  pt->Otdel_prnt(); // вызов метода Otdel_prnt, используя
                    //указатель на объект
  rb.Bank_prnt();
  pt->Bank_prnt();

  rb.Work_prnt();
  pt->Work_prnt();
  return 0;
}
```

При применении множественного наследования возможно возникновение нескольких конфликтных ситуаций. ***Первая*** − конфликт имен методов или атрибутов нескольких базовых классов:

```
class A
{   public: void fun(){}
};
class B
{   public: void fun(){}
};
class C : public A, public B
{  };
int main()
{
   C obj;
   obj.fun();            //  error C::f' is ambiguous
   return 0;
}
```

При таком вызове функции fun() компилятор не может определить, к какой из двух функций классов A и B выполняется обращение. Неоднозначность можно устранить, явно указав, какому из базовых классов принадлежит вызываемая функция:

```
obj.A:: fun();  или  obj.B::fun();
```

***Вторая*** проблема возникает при многократном включении некоторого базового класса:

```
class A
{ public: void fun(){}
};
class B : public A
{    // компоненты класса B
};
class C : public A
{    // компоненты класса C
};
class D : public B, public C
{ };

int main()
{
  D obj;
  obj.fun();            // ambiguous access of 'fun'
  return 0;
}
```

Проблема состоит в том, что при создании объекта класса D создаются два (одинаковых) объекта базового класса A (для объектов класса B и C соответственно). Решение этой проблемы состоит в использовании виртуального наследования:

```
class B : virtual public A
class C : virtual public A.
```

В этом случае при создании объекта класса D происходит однократное создание виртуального объекта класса A. Рассмотрим пример программы.

**Виртуальное наследование. Пример:** Разработать иерархию классов, реализующих виртуальное наследование. Базовый класс Car содержит название марки автомобиля, первый производный класс Color – цвет и число дверей, второй производный класс Power – мощность двигателя и величину расхода топлива. Класс Shop, производный от этих двух классов, содержит название автомагазина. Все классы содержат методы отображения своих компонентов.

```
#include <iostream>
#include <string.h>
#include <locale.h>
#include <iomanip>
using namespace std;
class Car
{
    char *naz;        //  марка автомобиля
  public:
    Car(char *NAZ)
```

```
    {
      naz=new char[strlen(NAZ)+1];
      strcpy(naz,NAZ);
    }
    ~Car()
    {
      delete [] naz;
      cout << "деструктор класса A" << endl;
    }
    void car_prnt(){ cout <<"марка а/м "<<naz<< endl; }
};
class Color : virtual public Car
{
 protected:
     char *cv;        // цвет автомобиля
     int  kol;        // количество дверей
 public:
     Color(char *NAZ,char *CV,int KOL): Car(NAZ),kol(KOL)
     {
       cv=new char[strlen(CV)+1];
       strcpy(cv,CV);
     }
     ~Color()
     {
       delete [] cv; ;
       cout << "деструктор класса B1" << endl;
     }
     void color_prnt()
     {
       Car::car_prnt();
       cout << "цвет а/м "<<cv<<" количество дверей "<<kol<<endl;
     }
};
class Power : virtual public Car
{
  protected:
     int pow;         // мощность двигателя автомобиля
     double  rs;      // расход топлива
  public:
     Power(char *NAZ,int POW,double RS): Car(NAZ),pow(POW),rs(RS) {};
    ~Power(){ cout << "деструктор класса B2" << endl; }
    void power_prnt()
    {
      Car::car_prnt();
      cout <<"мощность двигателя "<<pow<<" расход топлива "<<rs<<endl;
    }
};
class Shop : public Color,public Power
{
```

```
    char *mag;          // название магазина
  public: Shop(char *NAZ,char *CV,int KOL,int POW,double RS,char *MAG):
    Color(NAZ,CV,KOL),Power(NAZ,POW,RS),Car(NAZ)
    {
      mag =new char[strlen(MAG)+1];
      strcpy(mag,MAG);
    }
    ~Shop()
   {
     delete [] mag;
     cout << "деструктор класса C" << endl;
   }
   void shop_prnt()
  {
    Car::car_prnt();
    Color::color_prnt();
    Power::power_prnt();
    cout << " название магазина " <<mag<<endl;
  }
};
int main()
{
  Shop car("BMW","красный",4,140,8.5,"Автомагазин"); // объект класса Shop
  Shop *pt=&car; // указатель на созданный объект. Далее можно использо-
                 //  вать для вызова методов либо сам объект car, либо
                 // указатель на него pt
  setlocale(LC_ALL,"Russian" );
  car.car_prnt();
  pt->car_prnt();
  car.color_prnt();
  pt->color_prnt();
  car.power_prnt();
  pt->power_prnt();
  car.shop_prnt();
  pt->shop_prnt();
  return 0;
}
```

## Контрольные вопросы

1. Опишите модификаторы доступа и наследования. Как изменяются атрибуты доступа элементов класса при наследовании?
2. Как работают конструкторы при наследовании?
3. Как работают деструкторы при наследовании?
4. Какой класс называется производным?
5. Как объявляются производные классы?
6. Для чего используется множественное наследование? Чем оно отличается

от простого наследования?

7. Каков механизм вызова конструкторов при множественном и виртуальном наследовании?

8. Какие проблемы возможны при множественном наследовании?

## Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Ознакомиться с материалами литературных источников.
3. Ответить на контрольные вопросы.
4. Разработать алгоритм программы.
5. Написать, отладить и выполнить программу.