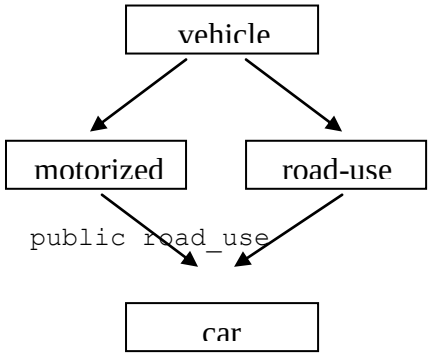
5. Виртуални функции и виртуални класове

1. Виртуални базови класове.
При директното наследяване на множество базови класове е възможно един производен клас да наследи многократно даден базов клас. Нека класа **vehicle** има два производни класа **motorized** и **road_use**, които от своя страна са базови за класа **car**:




```
class vehicle
{ int range; ... }
class motorized : public vehicle
{ ... };
class road_use : public vehicle
{ ... };
class car : public motorized,

{ ... };
```

В този случай класът **car** наследява двукратно класа **vehicle** – първо се наследява чрез **motorized**, а след това чрез **road_use**. Това води до дублиране на член-променливи и обект от класа **car** ще наследи двукратно член-променливата **range** декларирана в класа **vehicle**.
Многократното наследяване може да се избегне чрез наследяване на базовия клас като **виртуален**. Това предотвратява присъствието на 2(или повече) копия на базовия клас във всеки следващ индиректен производен клас. Декларацията в този случай ще има вида:

```
class vehicle
{ int range; ... }
class motorized : virtual public vehicle
{ ... };
class road_use : virtual public vehicle
{ ... };
class car : public motorized, public road_use
{ ... };
```

Задача 1. В следната програма е дефиниран базов клас за превозно средство – **vehicle**. В производните му класове – **motorized** и **road_use**, той е деклариран като *виртуален*. От своя страна **motorized** и **road_use** са базови класове за клас **car**. Да се дефинира обект от клас **car** и да се изведе дефинираният обект.

```
#include<iostream.h>
//Базов клас за различни превозни средства
class vehicle {
       int num_wheels;
       int range;
    public:
       vehicle(int w, int r)
       {
         num_wheels = w; range = r;
       }
       void showv()
       {
         cout << ”Wheels:” << num_wheels << ’\n’;
```

```
        cout << ”Range:” << range << ’\n’;
      }
};
enum motor {gas, electric, diesel};

class motorized : virtual public vehicle {
      enum motor mtr;
    public:
      motorized(enum motor m, int w, int r) : vehicle(w, r)
      {
        mtr = m;
      }
      void showm() {
        cout << ”Motor:”;
        switch(mtr) {
          case gas : cout << ”Gas\n”;
            break;
          case electric : cout << ”Electric\n”;
            break;
          case diesel : cout << ”Diesel\n”;
            break;
       }
      }
};

class road_use : virtual public vehicle {
      int passengers;
    public:
      road_use(int p, int w, int r) : vehicle(w, r)
      {
        passengers = p;
      }
      void showr()
      {
        cout << ”Passengers:” << passengers << ’\n’;
      }
};

enum steering {power, rack_pinion, manual};

class car : public motorized, public road_use {
      enum steering strng;
    public:
      car(enum steering s, enum motor m, int p, int w, int r) :
          motorized(m, w, r), road_use(p, w, r), vehicle(w, r)
      {
        strng = s;
      }
      void show() {
        showv(); showr(); showm();
        cout << ”Steering: ”;
        switch(strng) {
          case power : cout << ”Power\n”;
            break;
          case rack_pinion : cout << ”Rack and pinion\n”;
            break;
          case manual : cout << ”Manual\n”;
            break;
       }
      }
};
```

```
int main()
{
  car c(power, gas, 5, 4, 500);

  c.show();
  return 0;
}
```

2. Виртуални функции.

Виртуалната функция представлява член-функция, която се дефинира в базовия клас и се предефинира в производния клас. За да се създаде виртуална функция, преди декларацията на функцията трябва да се добави ключовата дума **virtual**. Когато виртуална функция се предефинира в производен клас, ключовата дума **virtual** не е необходима.

Чрез виртуалните функции се постига *полиморфизъм по време на изпълнение*. За целта виртуалната функция трябва да бъде извикана посредством указател. Когато указател към базов клас сочи към обект от производен клас, съдържащ виртуална функция, и тази функция се извика посредством указател, C++ решава коя версия на функцията ще бъде извикана въз основа на *типа на обекта*, сочен от указателя. А това решение се взема по време на изпълнение.

Задача 2. Следната програма създава общ базов клас, наречен **area**, който съдържа двата размера на една фигура. Той също така декларира чисто виртуална функция **getarea()**, която е предефинирана от производните класове, и връща лицето на типа фигура, дефинирана от производния клас.

```
#include<iostream.h>

class area {
       double dim1, dim2;  //размери на фигурата
    public:
       void setarea(double d1, double d2)
       {
         dim1 = d1;
         dim2 = d2;
       }
       void getdim(double &d1, double &d2)
       {
         d1 = dim1;
         d2 = dim2;
       }
       virtual double getarea() = 0;  //чисто виртуална функция
};
class rectangle : public area {
    public:
       double getarea()
       { double d1, d2;
         getdim(d1, d2);
         return d1*d2;
       }
};
class triangle : public area {
    public:
       double getarea()
       { double d1, d2;
         getdim(d1, d2);
         return 0.5*d1*d2;
       }
};
int main()
```

```
{
  area *p;
  rectangle r;
  triangle t;
  r.setarea(3.3, 4.5);
  t.setarea(4.0, 5.0);
  p = &r;
  cout << ”Rectangle has area: ” << p->getarea() << ’\n’;
  p = &t;
  cout << ”Triangle has area: ” << p->getarea() << ’\n’;
  return 0;
}
```

Тъй като за виртуалната функция **getarea()** в базовия клас не съществува смислено действие, което тя да извършва, всеки от производните класове трябва да предефинира тази функция. За тази цел тя е декларирана като чисто виртуална функция. Задаването на стойност 0 за функцията казва на компилатора, че тази функция няма тяло в базовия клас.

Задачи за самостоятелна работа:

1. Разгледайте Exam5.cpp. Обяснете си наследяването и виртуалните методи. Какво ще се разпечати на екрана? Пуснете програмата и проверете правилността на отговорите си.
2. Добавете и имплементирайте класове за Тигър и Слон по аналогия на примера.