|  |  |
| --- | --- |
| 实验名称 | 实验四 多态与虚函数程序设计 |
| 实  验  目  的  及  要  求 | 1. 理解静态联编与动态联编。 2. 理解静态多态与动态多态。 3. 掌握虚函数的作用与特点。 4. 理解虚函数对多态性的支持。 5. 理解纯虚函数与抽象类的作用。 6. 理解虚析构函数的作用。 7. 掌握虚函数和纯虚函数的概念。 8. 理解纯虚函数与抽象类的作用。 9. 掌握运算符重载的规则与方法。 |
| 实验  环境 | 操作系统：Windows 10 loT企业版LTSC 版本21H2(操作系统内部版本19044.1645)  开发环境：Clion 2021.3.4 |
| 实  验  内  容 | 1. 运行程序，分析结果。   #include <iostream>  using namespace std;  class B  {  public:  virtual void f1(double x)  {  cout<<"B::f1(double)"<<x<<endl;  }  void f2(double x)  {  cout<<"B::f2(double)"<<2\*x<<endl;  }  void f3(double x)  {  cout<<"B::f3(double)"<<3\*x<<endl;  }  };  class D:public B  {  public:  virtual void f1(double x)  {  cout<<"D::f1(double)"<<x<<endl;  }  void f2(double x)  {  cout<<"D::f2(double)"<<2\*x<<endl;  }  void f3(double x)  {  cout<<"D::f3(double)"<<3\*x<<endl;  }  };  int main()  { D d;  B\* pb=&d;  D\* pd=&d;  pb->f1(1.23);  pd->f1(1.23);  pb->f2(2.23);  pd->f2(2.23);  pb->f3(3.23);  pd->f3(3.23);  return 0;  }     1. 运行下面的程序，分析运行结果；再把基类的析构函数定义为virtual，再运行程序，分析运行结果，体会将基类的析构函数定义为虚析构函数的作用。   #include <iostream>  using namespace std;  class BaseClass  {  public:  BaseClass(){cout<<"BaseClass()"<<endl;}  ~BaseClass() { cout<<"~BaseClass()"<<endl;}  };  class DerivedClass:public BaseClass  {  public:  DerivedClass(){cout<<"DerivedClass()"<<endl;}  ~DerivedClass(){ cout<<"~DerivedClass()"<<endl;}  };  int main()  {  BaseClass \*bp=new DerivedClass();  delete bp;  return 0;  }     1. 编写程序，计算三角形、正方形和圆形这3种图形的周长和面积，并在main函数中使用相关数据进行测试。要求：   （1）抽象出一个基类Shape类，从其派生出Triangle类（三角形类）、Square（正方形类）、Circle（圆形类）。   1. Shape类中定义3个protected数据成员供派生类使用，派生类中无需定义数据成员。 2. Shape类中定义求面积getArea()和求周长getcircumference()成员函数为虚函数。其派生类中分别重定义各自的求面积和求周长函数。   #include <iostream>  #include<cmath>  using namespace std;  class Shape{  public:  Shape(double a=0,double b=0,double c=0)  {  data1=a;  data2=b;  data3=c;  }  virtual double getArea()  {  cout<<"area is:";  return 0;  }  protected:  double data1;  double data2;  double data3;  };  class Triangle: public Shape{  public:  Triangle(double a=0,double b=0,double c=0):Shape(a,b,c){}  double s=(data1+data2+data3)/2;  double getArea(){  cout<<"Triangle area is:"<<endl;  return sqrt(s\*(s-data1)\*(s-data2)\*(s-data3));  }  };  class Square: public Shape{  public:  Square(double a=0,double b=0): Shape(a,b){}  double getArea()  {  cout<<"Square area is:"<<endl;  return data1\*data2;  }  };  class Circle: public Shape{  public:  Circle(double a=0): Shape(a){};  double getArea()  {  cout<<"Circle area is:"<<endl;  return 3.1416\*data1\*data1;  }  };  int main() {  Triangle tri1(4,5,6.1);  Square squ1(5.1,6);  Circle cir1(2.000);  cout<<squ1.getArea()<<endl;  cout<<tri1.getArea()<<endl;  cout<<cir1.getArea()<<endl;  return 0;  }   1. 一个游戏中有多种怪物(Monster)，怪物之间可能要发生战斗(fight)，每场战斗都是一个怪物与另一怪物之间的一对一战斗。每个怪物都有自己的生命值(hitpoint)、攻击力值(damage)和防御力值(defense)，每种怪物都有各自特有的攻击(attack)方式，产生相应的攻击效果；战斗时，两个怪物依次攻击对方，即怪物a首先攻击怪物b, 然后轮到怪物b攻击怪物a, 之后，怪物a再次攻击怪物b，…, 直到一方生命值为0。   请根据你对上述描述的理解：  （1）定义并实现怪物类Monster，成员的设计可以包括数据成员hitpoint、damage和defense，以及其他任意需要的成员函数。要求包括一个纯虚函数attack，作为派生类对象各自实现攻击细节的接口；要求定义一个非虚的成员函数fight，用来描述与另外一个怪物进行战斗的过程，该函数的实现可为Monster类的任意派生类所复用（派生类不需重新定义及实现）。不必考虑怪物的生命值减少至0后如何处理。   |  |  | | --- | --- | | 猫进攻导致对方的生命值减少量： | 狗进攻导致对方的生命值减少量： | | （猫的攻击力值 \* 2 — 对方的防御力值）  若上式小于1，则取1 | (狗的攻击力值 — 对方的防御力值 ＋5 )\*2  若上式小于2，则取2 |  1. 作为怪物的特例，猫和狗的攻击效果如下表所示。在Monster的基础上，以继承手段定义并实现这两个类。   （3）再增加其他派生类，如Cock类，自行定义进攻对方的生命值减少量。  自行设计并实现游戏的测试函数，输出相互攻击的实时生命力值，并给出胜负结果。  #include<iostream>  #include<cmath>  #include<string.h>  using namespace std;  class Monster  {  public:  Monster(int hit,int dam,int def)  {  hitpoint=hit;  damage=dam;  defense=def;  }  bool fight(Monster &other)  {  while(true)  {  attack(other);  if(other.GetHP()<=0)  return true;  other.attack(\*this);  if(hitpoint<=0)return false;  }  }  virtual void attack(Monster &other)=0;  void ReduceHP(int harm)  {  hitpoint=hitpoint-harm;  if(hitpoint<0)  hitpoint=0;  }  int GetHP()const  {  return hitpoint;  }  int GetDefense()const  {  return defense;  }  protected:  int hitpoint;//生命值  int damage;//攻击力值  int defense;//防御力值  };  class Cat:public Monster  {  public:  Cat(int hit,int dam,int def):Monster(hit,dam,def)  {}  virtual void attack(Monster &other)  {  int harm=damage\*2-other.GetDefense();//harm代表生命值减少量  if(harm<1) harm=1;  other.ReduceHP(harm);  cout<<"Cat HP:"<<hitpoint<<"|"<<endl;  }  };  class Dog:public Monster  {  public:  Dog(int hit,int dam,int def):Monster(hit,dam,def)  {}  virtual void attack(Monster &other)  {  int harm=(damage-other.GetDefense()+5)\*2;  if(harm<2) harm=2;  other.ReduceHP(harm);  cout<<"Dog HP:"<<hitpoint<<"|"<<endl;  }  };  class Cock:public Monster  {  public:  Cock(int hit,int dam,int def):Monster(hit,dam,def)  {}  virtual void attack(Monster &other)  {  int harm=10;  other.ReduceHP(harm);  cout<<"Cock HP:"<<hitpoint<<"|"<<endl;  }  };  int main()  {  Dog d(100,10,7);  Cat c(120,8,9);  cout<<"DOG VS CAT"<<endl;  cout<<"Cat:"<<c.GetHP()<<endl;  cout<<"Dog:"<<d.GetHP()<<endl<<endl;  if(d.fight(c))  cout<<"DOG WIN"<<endl;  else cout<<"DOG LOSE"<<endl;  cout<<endl;  Cock e(150,5,3);  cout<<"Cock VS DOG"<<endl;  if(e.fight(d))  cout<<"Cock WIN!"<<endl;  else cout<<"Cock LOSE!"<<endl;  }  5.设计一个复数Complex类，实现复数的加、减运算，并实现复数的输入、输出。已知测试main函数如下：  #include <iostream>  using namespace std;  class Complex{  public:  Complex(double a=0,double b=0){  num1 = a;  num2 = b;  }  friend ostream &operator<<( ostream &output, const Complex &D )  {  output << "real:" << D.num1 << "vitural:" << D.num2<<endl;  return output;  }  friend istream &operator>>( istream &input, Complex &D )  {  input >> D.num1 >> D.num2;  return input;  }  Complex operator+(const Complex& temp)  {  Complex shi;  shi.num1 = this->num1 + temp.num1;  shi.num2 = this->num2 + temp.num2;  return shi;  }  friend Complex operator+(const int add, const Complex origin)  {  Complex sum;  sum.num1 = add + origin.num1;  sum.num2 = origin.num2;  return sum;  }  friend Complex operator-(const Complex origin,const int add)  {  Complex sum;  sum.num1 = origin.num1 - add;  sum.num2 = origin.num2;  return sum;  }  private:  double num1;  double num2;  };  int main( )  {  Complex c1(5.0,10.0); //5+10i  Complex c2(3.0,-2.5); //3-2.5i  Complex c3,c4,c5=20;  c3=c1+c2;  c4=c1-5;  cout<<c1<<c2<<c3<<c4;  // Complex c5;  cout<<"input a complex , with a space in it"<<endl;  cin>>c5;  cout<<"the complex you input is"<<endl<<c5<<endl;  c4=5+c5;  cout<<c4<<endl;  return 0;  }  6.设计一个三角形类Triangle,包含三角形三条边长的私有数据成员，另有一个重载运算符“+",以实现求两个三角形对象的面积之和，进一步实现3个或更多的三角形对象相加的面积之和。  分析提示：在Triangle类中设计一个友元函数double operator+(Triangle t1,Triangle t2),它重载运算符“+",返回t1和t2两个三角形的面积之和。  #include <iostream>  #include<cmath>  using namespace std;  class Triangle{  private:  double line1;  double line2;  double line3;  double s;  public:  Triangle(double a=0, double b=0,double c=0){  line1=a;  line2=b;  line3=c;  }  double area(){  double s;  s=(line1+line2+line3)/2;  return sqrt(s\*(s-line3)\*(s-line2)\*(s-line1));  }  /\*operator double()  {  return this->area();  }\*/  friend double operator+(Triangle t1,Triangle t2){  return t1.area()+t2.area();  }  friend double operator+(double t,Triangle t1)  {  return t+t1.area();  }  };  int main()  {  Triangle tri1(3,4,5),tri2(6,8,10);  double area1=tri1+tri2;  cout<<area1<<endl;  Triangle tri3(9,12,15);  double area2=tri1+tri2+tri3;  cout<<area2<<endl;  return 0;  } |
| 调  试  过  程  及  实  验  结  果 | 1. 运算符重载的时候 两个类相加可以直接写 2. 类+int 需要注意左加右加的写法不同 必要的时候使用友元函数 |
| 总  结 | 菜鸟教程的样例代码是个好东西  https://www.runoob.com/cplusplus/cpp-tutorial.html |