声明

下面所写的一些知识观点，还有各种经典案例，全部都是个人平时的学习知识积累，经过一番辛苦整理所得的劳动成果，真心希望可以帮到初学 C++的同学们，让我们一起来分享知识吧！

作者签名：陈小玲

日期： 2012 年 10 月 1 0 日

目录

1.C++经典案例分享

1.1 + 运算符重载

#include "iostream.h"//当出现函数有 **friend** 时就需要改为 **.h** 形式

```
class Point
{
public:
    Point()
    {
        x=0;
        y=0;
    }
    Point(double r,double i)
    {
        x=r;
        y=i;
```

```
}
friend Point operator+(Point &p1,Point &p2)
{
    Point pp;
    pp.x=p1.x+p2.x;
    pp.y=p1.y+p2.y;
    return pp;
}
void disp()
{
    cout<<"("<<x<<","<<y<<")"<<endl;
    }
private:
    double x;
    double y;
};

void main()
{
    Point p1(1,2),p2(3,4),p3;
    p3=p1+p2;
    p1.disp();
```

```
    p2.disp();

    p3.disp();

}
```

1.2　+运算符重载练习

// 将“ **+** ”运算符重载为一般的成员函数,　　该运算符实现两个点坐标的相加运算。

// 理解：两个点坐标　**p1(1,2),p2(3,4)** 是类的两个对象

// 两点坐标的距离为　**:p1+p2=((1+3),(2+4))**

// 输出格式为：　**(x,y)**

```
#include "iostream"

using namespace std;


class Point

{

public:

private:

    int x,y;
```

```
};


void main()
{
    Point p1(1,2),p2(3,4),p3;
    p3=p1.distance(p2);//
    p1.disp();
    p2.disp();
    p3.disp();
}
```

1.3　2 个类的友元函数

```
// 教师：工号、姓名、工资
// 学生：学号、姓名、成绩
#include "iostream"
using namespace std;
class Date;//1，提前声明某个类
class Time
{
public:
    Time(int h,int m,int s);
```

```
    void disp(Date &dx);//disp()是 Time 类的函数
private:
    int hour;
    int minute;
    int sec;
};

class Date//
{
public:
    Date(int y,int m,int d);
    friend void Time::disp(Date&dx);// 将 Time 类中的 disp函数作为
Date 类的朋友
private:
    int year;
    int month;
    int day;
};

Time::Time(int h,int m,int s)
{
    hour=h;
```

```
    minute=m;

    sec=s;

}

void Time::disp(Date &dx)//

{

    cout<<dx.year<<"/"<<dx.month<<"/"<<dx.day<<endl;

    cout<<hour<<":"<<minute<<":"<<sec<<endl;

}


Date::Date(int y,int m,int d)

{

    year=y;

    month=m;

    day=d;

}


void main()

{

    Date date(2011,9,28);
```

```
    Time time(11,20,35);

    time.disp(date);

}
```

1.4　and 重载

```
// 重载 && || !

#include "iostream.h"

class A

{

public:

A(int x1)

{

    x=x1;

}

bool operator!()

{

    if(x!=0) return 0;

    else return 1;

}

friend bool operator&&(A &w,A &dx)

{
```

```
if(w.x!=0)
{
    if(dx.x!=0) return 1;
    else return 0;
        }
        else
            return 0;


    }
private:
    int x;
};
void main()
{
    A a(0),b(-2),c(25);
    cout<<'输出 !结果 "<<endl;
    cout<<(!a)<<endl;//1
    cout<<(!b)<<endl;//0
    cout<<'输出 &&结果 "<<endl;
    cout<<(a&&b)<<endl;//0
    cout<<(b&&c)<<endl;//1
```

```
}
```

1.5 student

```
#include "iostream"

#include "string"

using namespace std;


class student//父类

{

private:

    int num;

protected:

    string name;

public:

    char sex;

    student(int x1,string y1,char z1)

{

    num=x1;

    name=y1;

    sex=z1;
```

```
    }
};

class student1:public student//子类---继承方式:public---a,b,c,y(protected),z(public)
{

private:
    int chinese;
protected:
    int math;
public:
    int english;
    student1(int x1,string y1,char z1,int a1,int b1,int c1):student(x1,y1,z1)//只写子类
    {
        chinese=a1;
        math=b1;
        english=c1;

    }
void disp()
```

```
{
    cout<<"name:"<<name<<endl;
    cout<<"sex:"<<sex<<endl;
    cout<<"chinese:"<<chinese<<endl;
    cout<<"math:"<<math<<endl;
    cout<<"english:"<<english<<endl;
  }

};

void main()
{
    student1 stu1(200118,"she",'F',78,66,99);//先构造父，再构造子类
    stu1.disp();
}
```

错误！文档中没有指定样式的文字。

1.6 成员函数运算符重载练习

```
// 将“ + ”运算符重载为一般的成员函数，  该运算符实现两个点坐标的相加运算。
// 理解：两个点坐标  p1(1,2),p2(3,4) 是类的两个对象
// 两点坐标的距离为  :p1+p2=((1+3),(2+4))
// 输出格式为：  (x,y)
```

```
#include "iostream"

using namespace std;


class Point
{
public:
    Point()
    {
        x=0;
        y=0;
    }
    Point(double r,double i)
    {
        x=r;
        y=i;
    }
    Point operator+(Point &p2)
    {
        Point pp;
        pp.x=x+p2.x;
        pp.y=y+p2.y;
        return pp;
```

```
}
void disp()
{
    cout<<"("<<x<<","<<y<<")"<<endl;
    }
private:
    double x;
    double y;
};


void main()
{
    Point p1(1,2),p2(3,4),p3;
    p3=p1+p2;
    p1.disp();
    p2.disp();
    p3.disp();
}
```

1.7 抽象类 shape

```
#include<iostream>
```

```
using namespace std;

class shape

{

   protected:double x,y;

   public:void set(double i=0,double j=0)

                {

                  x=i;

                y=j;

              }

   virtual void area()=0;

};

class triangle:public shape

{

   public:void area()

                {

                  cout<<"三角形的面积是  :"<<0.5*x*y<<endl;

              }

};

class circle:public shape

{

public:void area()

              {
```

```
                cout<<"圆的面积是 :"<<3.14*x*x<<endl;
        }
};
class rectangle:public shape
{
    public:void area()
            {
            cout<<"矩形的面积是  :"<<x*y<<endl;
          }
};
void main()
{
    shape *p;
    triangle t;
    circle c;
    rectangle r;
    p=&t;
    p->set(5,10);
    p->area();
    p=&c;
    p->set(5);
    p->area();
```

```
p=&r;

p->set(5,10);

p->area();
```

```
}
```

1.8　多重继承

```
#include<iostream>

using namespace std;

class B1

{

   protected:

                    int b1;

  public:B1(int i)

                {

                     b1=i;

                   cout<<'基类 B1的构造函数被调用  "<<endl;

}

~B1()

{
```

```
                cout<<"基类 B1的析出函数被调用  "<<endl;
        }
};
class B2
{
   protected:
                int b2;
  public:B2(int i)
        {
                b2=i;
                cout<<"基类 B2的构造函数被调用  "<<endl;
        }
        ~B2()
        {

                cout<<"基类 B2的析出函数被调用  "<<endl;
        }
};
class D:public B1,public B2
{
    protected:int d;
    public:D(int i,int j,int k):B1(i),B2(j)
```

```
{
    d=k;
    cout<<"派生类 D 的构造函数被调用  "<<endl;
 }
                ~D()
               {


                  cout<<"派生类 D 的析出函数被调用  "<<endl;
                }
};
void main()
{
        D dobj(10 ,20,30);
}
```

1.9　函数模板排序

```
#include<iostream>
using namespace std;
template<class T>
void input(T array[],int m)
```

```
{
    int i;
    for(i=0;i<=m-1;i++)
    cin>>array[i];
}
template<class T>
void sort(T array[],int m)
{
    int i,j;
    T temp;
    for(i=0;i<=m-1;i++)
{
        for(j=i+1;j=m-1;j++)
        if(array[i]>array[j])
        {
            temp=array[j];
            array[j]=array[i];
            array[i]=temp;
        }
    }
}
template<class T>
```

```
void output(T array[],int m)
{
    int i;
    for(i=0;i<m-1;i++)
    cout<<array[i]<<" ";
    cout<<endl;


}
void main()
{
     int ai[10],n,m;
char ac[10];
cout<<'输出 int 型数组的元素个数  ";
cin>>n;
  cout<<'输入 "<<n<<"个 int  型数据 :";
  input(ai,n);
cout<<'排序前  int 型数组为 :";
  output(ai,n);
sort(ai,n);
cout<<'排序后  int 型数组为 :";
   output(ai,5);
cout<<endl;
```

```
  cout<<'输入 char型数组的元素个数为  :";
cin>>m;
  cout<<'输入"<<m<<"个 char 型数据 :";
  input(ac,m);
        cout<<'排序前  char型数组为 :";
          output(ac,m);
            sort(ac,m);
          cout<<'排序后  char型数组为 :";
            output(ac,m);


}
```

1.10　继承 3

```
#include <iostream>

using namespace std;

class  Manmal

{

    public:
```

```
void disp()

{

    cout<<"你爷爷结婚的时候抬轿   "<<endl;

        }

    private:int x;

};


class Dog :public Manmal

{

    public:

       void disp()//同名的时候，用   Manmal::disp();

      {

          Manmal::disp();

         cout<<"你爸爸结婚的时候开三轮   "<<endl;


       }

private:int y;

};

class D :public Dog

{

    public:
```

```
    void disp()//同名的时候，用  Manmal::disp();
{
    Dog::disp();
     cout<<"你结婚的时候开宝马  "<<endl;
                }
private:int z;
};

void main()
{
D dog;
dog.disp();
}
```

1.11 静态成员案例

```
#include "iostream"

using namespace std;


class Box

{

public:

    // 第 1 种

    Box(int width,int length)

    {


        this->width=width;

        this->length=length;

    }

    /*void volume()// 非静态成员函数， 直接可以引用静态和非静态的
数据成员

        {

            cout<<height*width*length<<endl;

        }*/
```

static void volume(Box&dx)// 静态成员函数 **,**可以直接使用静态的数据成员，而不能直接使用非静态的成员

```
    {
        //cout<<height<<endl;//静态成员 ,合法
       //cout<<width<<endl;// 非静态成员 ,不合法
       cout<<height*dx.width*dx.length<<endl;
    }
private:
    static int height;//静态数据成员 ----是在类外直接赋初值   ---公有
    int width;// 非静态数据成员
    int length;//非静态数据成员
};
int Box::height=10;
void main()
{
    Box box(2,3);
    box.volume(box);
}
```

// 凡是私有的数据成员都要用公有的成员函数使用

1.12　拷贝构造函数

```
#include<iostream>

#include<string>

using namespace std;

class person

{

    private:

                int num;

            char name[10];

    public:person(int n,char *str)

       {

         num=n;

             strcpy(name,str);

   }

   person(const person &x)

   {

         num=x.num;

      strcpy(name,x.name);

   }

   void show()

   {
```

```
            cout<<"num="<<num<<"name"<<name<<endl;
        }
};
void main()
{
    person per1(01,"？");
    per1.show();
    person per2(per1);
    per2.show();
}
```

1.13 类的继承

```
#include "iostream"
using namespace std;
class A//父类
{
private:
```

```
    int x;
protected:
    int y;
public:
    int z;
    A(int x1,int y1,int z1)
    {
        x=x1;
        y=y1;
        z=z1;
    }
};

class B:public A//子类 ---继承方式 :public---a,b,c,y(protected),z(public)
{

private:
    int a;
protected:
    int b;
public:
    int c;
```

```
B(int x1,int y1,int z1,int a1,int b1,int c1):A(x1,y1,z1)//写子类
{
    a=a1;
    b=b1;
        c=c1;

    }

};

void main()
{
    B b1(10,20,30,1,2,3);//先构造父，再构造子类

}
```

2. C++部分知识总结：

2.1 重载的 mean ：+将原来的意思表达出来

方法如下：

1. 数据类型 Operator+ （类名， &别名）

（2 个友元函数） friend 数据类型 operator+ （类名， &别名）

“ 用成员函数重载运算符 ”

格式如下：

<类型><类名：：>operator < 运算符 >(形参表)

{

函数体；

}

“ 用友元函数重载运算符 ”

格式如下：

Friend< 函数类型 >operator < 运算符 >(形参表)

{

函数体；

}

2.2 同名可以用指针实现，不同名的只能用对象进行实现。

同名实现的三种方法总结：

1. 用指针实现

2. 用全局函数 --- 静态绑定

3. 设置对象直接实现

不同名实现俩种方法总结：

1. 用对象来实现

2. 设置一个新的对象实现（引用）

Eg: （ fish f;

f.animal::eata();

f.eatf();

）

2.3 非静态成员函数与静态成员区别

1. 非静态成员函数，直接可以引用静态和非静态的数据成员

2. 静态成员函数 ,可以直接使用静态的数据成员，而不能直接使用非静态的成员

3. 静态数据成员 ---- 是在类外直接赋初值 --- 公有

4. 凡是私有的数据成员都要用公有的成员函数使用

注意：

1.1　　静态成员，合法

1.2　　非静态成员，不合法

2.4 继承：

1. 单继承概念：

2. 成员的访问（ P264)

派生类的构造函数的调用顺序是：

11. 基类的构造函数

12. 子对象类的构造函数

13. 派生类的构造函数

纯函数概念理解：

2.5 纯虚函数和抽象类

1）　　基类指针

2）　　纯虚函数，必须在基类中定义

Vitual void disp()=0;

3)　纯虚函数所在的类是抽象类，抽象类不能构造对象。

4) 纯虚函数必须在它的子类中去实现，不实现是构造不了对象的

2.5.1 纯虚函数

纯虚函数是一个在基类中说明的虚函数，　　它在该基类中只有一个函数

声明，并没有具体函数功能的实现。

格式如下：

Virtual < 函数类型 ><函数名 >（参数列表） =0；

纯虚函数与一般虚函数在书写形式上的不同之处在于其后面有 “ =0 ”，表明在基类中无须定义该函数， 它的实现部分， 即函数体将在各派生类完成。

2.5.2 抽象类

包含一个或多个纯虚函数的类称为抽象类。 由于抽象类中的纯虚函数没有具体的函数实现，因此不能定义抽象类的对象。

抽象类的重要用途是提供一个接口，而不提示任何实现细节。

如何一个派生类继承自抽象类， 但是并没有重新定义抽象类中的纯虚函数，则该派生类仍然是一个抽象类。 只有当派生类中所继承的所有纯虚函数都被实现时，它才不是抽象类，此时的派生类称作具体类。

[eg1]

```
#include "iostream"
using namespace std;


class Animal
{
```

```
public:
    virtual void sleep()=0;


};
class Fish:public Animal
{
public:
    void sleep()
    {
        cout<<"Fish sleep."<<endl;
    }


};
void main()
{
    Fish f,*pf;
    pf=&f;
    pf->sleep();
}
```

虚基类：

Class< 派生类 >:virtual[　　继承方式]< 基类名 >

虚函数

格式如下：

virtual<　函数类型 ><函数名 >(形参名)

2.6 异常处理的的语法：

在 C++程序中，任何需要检测异常的语句都必须在　try　语句块中执行，异常必须由紧跟着　try　语句后面的　catch　语句来捕获并处理，因而 try　与 catch 总是结合使用的。

throw< 表达式 >；

try

{

　　//try　　语句块

}

catch(类型 1 参数 1)

{

　　// 针对类型 1 的异常处理；

}

catch(类型 2 参数 2)

{

　　// 针对类型 2 的异常处理；

}

，　.

包含一个或多个纯虚函数的类称为抽象类

2.7 构造函数的特点：

1. 构造函数名与类名相同

2. 构造函数需要指定函数返回值类型，　它有隐含的返回值，　该值在系统内部使用

3. 程序不能直接调用构造函数，在创建对象时需要由系统自动调用

4. 构造函数可以重载

5. 构造函数的实现部分即可以写在类体内，也可以写在类体外

6. 构造函数与一般函数和成员函数一样可以带默认参数

构造函数的一般格式为：

1.< 类名 >::< 构造函数名 >（参数表）

{

　　函数体；

}

2.< 类名 >::< 默认构造函数名 >（）

{

}

2.8 复制构造函数的特点：

1. 函数名与类名相同 （复制构造函数也是一种构造函数， 并且该函数不指定返回值类型）

2. 复制构造函数只有一个参数，是对某个对象引用

3. 每个类都必须有一个复制构造函数

复制构造函数的一般格式为：

<类名 >::< 复制构造函数名 >（const< 类名 >&<引用名 >）

析出函数一般格式为：

对象名 . 类名 :: 析构函数名

静态特点：上一次的值会保留下来

多态特点：指向谁的东西就输出谁的东西

函数类型有：构造函数（可以重载） 、析出函数、友元函数（可以重载）