实验四　运算符重载

一、定义一个复数类 Complex，重载运算符“+”，使之能用于复数的加法运算。参加运算的两个运算量可以都是类对象，也可以有一个是整数，顺序任意。重载运算符“++”，可以实现前增量和后增量（要求用成员函数和友元函数二种实现方法）。

```
#include <iostream.h>
//using namespace std;
class Complex
{public:
    Complex(){real=0;imag=0;}
    Complex(double r,double i){real=r;imag=i;}
    /*Complex operator + (Complex &c2)              //成员函数实现重载+
    {return Complex(real+c2.real,imag+c2.imag);}*/
    Complex(double r){real=r;imag=0;}     //复数与一数值相加时会调用此转换构造函数

    friend Complex& operator++(Complex& cc);
    friend Complex& operator++(Complex& cc,int);
    friend Complex operator+(Complex c1,Complex c2);//重载+，友元形式
//要实现复数+数值及数值+复数，所以 operator+只能重载定义为友元，且参数不能为引用
    friend ostream& operator<<(ostream& out,Complex c);
private:
    double real,imag;
 };
Complex& operator++(Complex& cc)          //重载前++运算符
{     ++cc.real;
      ++cc.imag;
      return cc;                    //返回实部与虚部各加 1 后的对象
}
Complex& operator++(Complex& cc,int)     //重载后++运算符
{
      Complex temp(cc);
      cc.real ++;cc.imag ++;
      return temp;
}
Complex operator+(Complex c1,Complex c2)
{
      return Complex((c1.real +c2.real),(c1.imag +c2.imag ));
}
ostream& operator<<(ostream& out,Complex c)
{
      return out<<c.real<<(c.imag>=0?"+":"")<<c.imag<<"i"<<endl;
}
int main()
{
```

```
    Complex c1(3,4),c2(5,-10),c3;
    c3=c1+c2;              //+重载定义后可以直接用+
    cout<<"c1="<<c1;
    cout<<"c2="<<c2;
    cout<<"c1+c2="<<c3;
    c3=c1+6.3;          //复数+数值
    cout<<c3;
    c3=6.3+c1;        //数值+复数
    cout<<c3;
    return 0;
}
```

二、有两个矩阵 a 和 b，均为 3 行 5 列。求两个矩阵之和。重载插入运算符“+”，使之能用于矩阵相加，重载插入运算符“<<”和流提取运算符“>>”，使之能用于该矩阵的输入与输出。

```
#include<iostream.h>
//using namespace std;
class Matrix
{
public:
    Matrix()                    //无参构造函数
    {
        for(int i=0;i<3;i++)
            for(int j=0;j<5;j++)
                p[i][j]=0;
    }
    Matrix(int (*a)[5])                    //无参构造函数
    {
        for(int i=0;i<3;i++)
            for(int j=0;j<5;j++)
                p[i][j]=a[i][j];
    }
    Matrix operator+(Matrix m2)
    {
        for(int i=0;i<3;i++)
            for(int j=0;j<5;j++)
            {
                p[i][j]+=m2.p[i][j];
            }
            return    *this;
    }
    friend istream& operator>>(istream& in,Matrix &m1);
    friend ostream& operator<<(ostream& out,Matrix m1);
private:
```

```
    int p[3][5];
};
istream& operator>>(istream& in,Matrix &m1)
{
    cout<<"请输入"<<"3*5"<<"矩阵的元素"<<endl;
    for(int i=0;i<3;i++)
        for(int j=0;j<5;j++)
        {
            in>>m1.p[i][j];
        }
        return in;
}
ostream& operator<<(ostream& out,Matrix m1)
{
    for(int i=0;i<3;i++)
    {   for(int j=0;j<5;j++)
        {
            out<<m1.p[i][j]<<" ";
        }
        out<<endl;
    }
    out<<"------------------------"<<endl;
    return out;
}
int main()
{
    int m1[3][5]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
    int m2[3][5]={15,14,13,12,11,10,9,8,7,6,5,4,3,2,1};
    Matrix a1(m1),a2(m2),a3;
    a3=a1+a2;
    cout<<a3;
}
```

三、如果矩阵的行数和列数不固定，实现上题的功能。

```
#include<iostream>
using namespace std;
class Matrix
{
public:
    Matrix()                    //无参构造函数
    {
        p=NULL;m=0;n=0;
    }
    Matrix(int a,int b)         //有参构造函数
    {
```

```
        m=a;
        n=b;
        p=new int*[m];
        for(int i=0;i<m;i++)
            p[i]=new int[n];
    }
    Matrix(Matrix &mat)          //拷贝构造函数，为什么要定义这个函数知道吗？
    {   int i,j;
        m=mat.m;
        n=mat.n;
        p=new int*[mat.m];
        for(i=0;i<m;i++)
            p[i]=new int[n];
        for(i=0;i<m;i++)
          for(j=0;j<n;j++)
            {
              p[i][j]=mat.p[i][j];
            }
        cout<<"copying"<<endl;
    }
    Matrix& operator=(Matrix mat)//重载赋值运算符=，为什么要定义这个函数知道吗？
    {   int i,j;
        m=mat.m;
        n=mat.n;
        p=new int*[mat.m];
        for(i=0;i<m;i++)
            p[i]=new int[n];
        for(i=0;i<m;i++)
          for(j=0;j<n;j++)
            {
              p[i][j]=mat.p[i][j];
            }
          return *this;
    }
    Matrix operator+(Matrix m2)   //重载加法运算符+
    {
        for(int i=0;i<m2.m;i++)
            for(int j=0;j<m2.n;j++)
            {
                    p[i][j]+=m2.p[i][j];
            }
            return   *this;
    }
    friend istream& operator>>(istream& in,Matrix &m1);
```

```
        friend ostream& operator<<(ostream& out,Matrix &m1);
        ~Matrix()                       //析构函数
        {
            for(int i=0;i<m;i++)
                delete []p[i];
            delete p;
        }
    private:
        int m,n,**p;
    };
    istream& operator>>(istream& in,Matrix &m1)
    {
        cout<<"请输入"<<m1.m<<"*"<<m1.n<<"矩阵的元素"<<endl;
        for(int i=0;i<m1.m;i++)
            for(int j=0;j<m1.n;j++)
            {
                in>>m1.p[i][j];
            }
            return in;
    }
    ostream& operator<<(ostream& out,Matrix &m1)
    {
        for(int i=0;i<m1.m;i++)
        {   for(int j=0;j<m1.n;j++)
            {
                out<<m1.p[i][j]<<" ";
            }
            out<<endl;
        }
        out<<"----------------------"<<endl;
        return out;
    }
    int main()
    {
        Matrix a1(3,5),a2(3,5),a3;
        cin>>a1;
        cin>>a2;
        cout<<a1<<a2;
        a3=a1+a2;    //会调用赋值运算符重载函数
        cout<<a3;
    }
```

四、定义一个字符串类 String，重载运算符+，使之能实现二个字符串的连接。重载插入运算符“<<”和流提取运算符“>>”，使之能用于该字符串的输入与输出。

请查阅资料了解 C++中的类 string 有哪些成员函数？你能自己编程实现哪些？ 请编程实现

```
#include<iostream.h>
#include<string.h>
#include<stdlib.h>
class String
{    char *p;
     int size;                                  //分配的空间大小
public:
     //构造函数与析构函数，构造函数在类外定义
     String();                                  //无参构造函数声明
     String(char *str);                         //有 1 个参数的构造函数声明
     String(const String &o);                   //复制构造函数
     ~String(){delete [] p;}                    //析构函数

   friend ostream& operator<<(ostream& stream,String &o);//重载流输出运算符函数声明
   friend istream& operator>>(istream& stream,String &o);//重载流输入运算符函数声明

     String& operator=(String &o);     //重载赋值运算符声明
     String& operator=(char *s);       //重载赋值运算符声明

     String operator+(String &o);      //重载加法运算符声明
     String operator+(char *s);        //重载加法运算符声明

     char& operator[](int i);                   //重载下标运算符声明
     const char& operator[](int i)const;     //重载下标运算符声明
     //以下是重载比较运算符的函数定义
     int operator==(String &o){return !strcmp(p,o.p);}
     int operator!=(String &o){return strcmp(p,o.p);}
     int operator<(String &o){return strcmp(p,o.p)<0;}
     int operator>(String &o){return strcmp(p,o.p)>0;}
     int operator<=(String &o){return strcmp(p,o.p)<=0;}
     int operator>=(String &o){return strcmp(p,o.p)>=0;}
     int operator==(char *s){return !strcmp(p,s);}
     int operator!=(char *s){return strcmp(p,s);}
     int operator<(char *s){return strcmp(p,s)<0;}
     int operator>(char *s){return strcmp(p,s)>0;}
     int operator<=(char *s){return strcmp(p,s)<=0;}
     int operator>=(char *s){return strcmp(p,s)>=0;}

     int _size(){return size-1;}          //返回字符串对象长度减 1
     int length(){return strlen(p);} //返回字符串对象保存的字符个数
};
String::String()                          //无参构造函数
```

```
{     size=1;
          p=new char[size];                    //申请堆内存空间
      if(!p)
      {     cout<<"Allocation error\n";
          exit(1);
      }
      *p='\0';                                 //创建一个空字符串
}
String::String(char *str)              //有 1 个参数的构造函数
{     size=strlen(str)+1;
          p=new char[size];                //申请堆内存空间
      if(!p)
      {     cout<<"Allocation error\n";
          exit(1);
      }
      strcpy(p,str);                           //将 str 指向的字符串赋给当前对象
}
String::String(const String &o)     //拷贝构造函数
{     size=o.size;
      p=new char[size];                        //申请堆内存空间
      if(!p)
      {     cout<<"Allocation error\n";   }
      strcpy(p,o.p);                     //将 o.p 指向的字符串赋给当前对象
}
String & String::operator=( String &s)
{
      size=s.size;
      p=new char[s.size ];
      strcpy(p,s.p);
      return *this;
}
String& String::operator=(char *s)        //重载赋值运算符声明
{
      size=strlen(s)+1;
      p=new char[size ];
      strcpy(p,s);
      return *this;
}
char& String::operator[](int i)
{
      if(i>=0&&i<=size-1)     //判断下标是否超出范围
          return p[i];        //返回下标为 index 的 data 数组元素引用
      else
      {     cout<<"下标超出范围"<<endl;
```

```
            exit(1);
        }
    }
    String String::operator+(String &o)
    {
        String temp;
        temp.size =temp.size +o.size-1;
        temp=new char[size]; //重新分配空间了，原有的数据在前面要保存
        strcpy(temp.p,p);
        strcat(temp.p,o.p);
        return temp;
    }
    String String::operator+(char *s)
    {
        char *temp=new char[size];
        strcpy(temp,p);
        size +=strlen(s) ;
        p=new char[size]; //重新分配空间了，原有的数据在前面要保存
        strcpy(p,temp);
        strcat(p,s);
        return *this;
    }

    ostream& operator<<(ostream& stream,String &o)
    {
        return stream<<o.p<<endl;
    }
    main()
    {
        String s1,s2("abce"),s3(s2);    //三种不同的构造
        s1=s2+s3;                       // +的操作
        cout<<s1;                       //<<的操作
        s1=s1+"ab";                     //另一种+的操作
        cout<<s1;
    }
```