1. 用运算符重载实现对复数的自加，即++A和A++，并在主函数中验证。
2. 对以下程序，用自定义拷贝构造函数和赋值运算符重载两种方法实现深拷贝。

#include <iostream>

#include <cstdlib>

using namespace std;

class **RandNumber**{ // 类定义

public:

**RandNumber**(int n=1); // 构造函数

**RandNumber**(const **RandNumber** &rn); // 拷贝构造函数

**~RandNumber**(); // 析构函数

int get(int i) const; // 取第i个随机数

int getNum() const; // 取随机数个数

void set(int i, int val); // 设置第i个随机数

void print() const; // 打印随机数

private:

int \*vec; // 存放随机数

int num; // 随机数个数

};

// 成员函数实现

**RandNumber::RandNumber**(int n): num(n) { // 构造函数

vec = new int[num];

for(int i=0; i<num; i++)

vec[i] = rand()%100;

}

**RandNumber::~RandNumber**(){ // 析构函数

if(vec!=NULL)

delete [ ] vec;

}

int **RandNumber::**get(int i) const { // 取第i个随机数

return vec[i];

}

int **RandNumber::**getNum() const { // 取随机数个数

return num;

}

void **RandNumber::**set(int i, int val){ // 设置第i个随机数

vec[i] = val;

}

void **RandNumber::**print() const { // 打印随机数

for(int i=0; i<num; i++)

cout<<vec[i]<<“\t”;

cout<<endl;

}

int main(){

**RandNumber** rn1(5);

**RandNumber** rn2; // 利用默认参数构造对象

**RandNumber** rn3=rn1; // 需要用自定义拷贝构造函数实现深拷贝

rn1.print();

rn2.print();

rn3.print();

rn2 = rn1; // 需要用赋值运算符重载实现深拷贝

rn1.print();

rn2.print();

rn3.print();

rn1.set(2, 100); // 设置rn1的随机数的值

rn1.print();

rn2.print();

rn3.print();

return 0;

}

3、用友元函数实现输入输出运算符的重载，并在主函数中调用。

#include<iostream>

using namespace std;

class Complex{

public:

**Complex**(double r=0.0, double img=0.0): real(r), image(img) { }

double getReal() const { return real; }

double getImage() const { return image; }

**friend istream& operator>>**(**istream** &in, **Complex** &cp) ; // 输入

**friend** **ostream&** **operator<<**(**ostream** &out, const **Complex** &cp); // 输出

private:

double real;

double image;

};

**istream& operator>>**(**istream** &in, **Complex** &cp){

in>>cp.real>>cp.image;

return in; // 返回输入流对象的引用

}

**ostream&** **operator<<**(**ostream** &out, const **Complex** &cp){

out<<cp.real<<“+”<<cp.image<<“i”;

return out; // 返回输出流对象的引用

}

int main()

{

Complex c1;

cin>>c1;

cout<<c1<<endl;

}

4、定义复数类的乘法与除法，使之能执行下列运算：

**Complex** a(2.0, 5.0), b(7.0, 8.0), c(0.0, 0.0);

c = a**\***b;

c = a**\***2.0;

c = 2.0**\***a;

c = a**/**b;

c = a**/**2.0;

c = 2.0**/**a;

【参考代码】

2、#include <iostream>

#include <cstdlib>

using namespace std;

class **RandNumber**{ // 类定义

public:

**RandNumber**(int n=1); // 构造函数

**RandNumber**(const **RandNumber** &rn); // 拷贝构造函数

**~RandNumber**(); // 析构函数

**RandNumber**& operator=(const **RandNumber** &rn); // 重载赋

// 值运算

int get(int i) const; // 取第i个随机数

int getNum() const; // 取随机数个数

void set(int i, int val); // 设置第i个随机数

void print() const; // 打印随机数

private:

int \*vec; // 存放随机数

int num; // 随机数个数

};

// 成员函数实现

**RandNumber::RandNumber**(int n): num(n) { // 构造函数

vec = new int[num];

for(int i=0; i<num; i++)

vec[i] = rand()%100;

}

**RandNumber::RandNumber**(const **RandNumber** &rn){ // 拷贝构造函数

num = rn.num;

vec = new int[num]; // 重新申请一个堆空间

for(int i=0; i<num; i++) // 依次值拷贝

vec[i] = rn.vec[i];

}

**RandNumber::~RandNumber**(){ // 析构函数

if(vec!=NULL)

delete [ ] vec;

}

**RandNumber**& **RandNumber::**operator=(const **RandNumber** &rn)｛// 重载赋值运算符

num = rn.num;

delete [ ] vec; // 先释放原有堆空间

vec = new int[num]; // 重新申请一个堆空间

for(int i=0; i<num; i++) // 依次值拷贝

vec[i] = rn.vec[i];

return **\*this**; // 返回自身类的引用

｝

int **RandNumber::**get(int i) const { // 取第i个随机数

return vec[i];

}

int **RandNumber::**getNum() const { // 取随机数个数

return num;

}

void **RandNumber::**set(int i, int val){ // 设置第i个随机数

vec[i] = val;

}

void **RandNumber::**print() const { // 打印随机数

for(int i=0; i<num; i++)

cout<<vec[i]<<“\t”;

cout<<endl;

}

int main(){

**RandNumber** rn1(5);

**RandNumber** rn2; // 利用默认参数构造对象

rn1.print();

rn2.print();

rn2 = rn1; // 使用重载赋值运算符

rn1.print();

rn2.print();

rn1.set(2, 100); // 设置rn1的随机数的值

rn1.print();

rn2.print();

return 0;

}

class **Complex**{

public:

**Complex**(double r=0.0, double img=0.0): real(r), image(img) { }

**Complex& operator=**(const **Complex** &cp ) { // 重载 赋值 运算符

real = cp.real;

image = cp.image;

return **\*this**; // 返回自身类的引用

}

**Complex& operator=**(double cp) { // 重载 赋值 运算符

real = cp;

image = 0.0;

return **\*this**; // 返回自身类的引用

}

double getReal() const { return real; }

double getImage() const { return image; }

private:

double real;

double image;

};