2020春季学期C++整理8.0

一.可变长数组的实现（Vector）

~~PS:其实可以概括为一句话：`cin >> a[i++]`~~

话不多说上代码：

```
#include<bits/stdc++.h>
using namespace std;
class CArray
{
    int size;
    int *ptr;
    public:
        CArray(int s = 0);
        CArray(CArray &a);
        ~CArray();
        void push_back(int v);
        CArray &operator = (const CArray & a);
        int length(){
            return size;
        }
        int & CArray :: operator [] (int i){
            return ptr[i];
        }
};
CArray :: CArray(int s) : size(s)
{
    if(s == 0) ptr = NULL;
    else ptr = new = int[s];
}
CArray :: CArray(CArray & a)
{
    if(!a.ptr){
        ptr = NULL;
        size = 0;
        return ;
    }
    ptr = new int[a.size];
    memcpy(ptr,a.ptr,sizeof(int) * a.size);
    size = a.size;
}
CArray :: ~CArray()
{
    if(ptr) delete [] ptr;
}
CArray & CArray :: operator = (const CArray & a)
{
    if(ptr == a.ptr) return *this;
    if(a.ptr == NULL){
        if(ptr) delete [] ptr;
        ptr = NULL;
```

```
46          size = 0;
47          return *this;
48      }
49      if(size < a.size){
50          if(ptr) delete [] ptr;
51          ptr = new int[a.size];
52      }
53      memcpy(ptr,a.ptr,sizeof(int) * a.size);
54      size = a.size;
55      return * this;
56  }
57  void CArray :: push_back(int v)
58  {
59      if(ptr){
60          int *tmpPtr = new int[size + 1];
61          memcpy(tmpPtr.ptr,sizeof(int) * a.size);
62          delete [] ptr;
63          ptr = tmpPtr;
64      }
65      else ptr = new int[1];
66      ptr[size++] = v;
67  }
68  int main()
69  {
70      CArray a;
71      for(int i = 0;i < 5;i++) a.push_back(i);
72      CArray a2,a3;
73      a2 = a;
74      for(int i = 0;i < a.length();i++) cout << a2[i] << ' ';
75      a2 = a3;
76      for(int i = 0;i < a2.length();i++) cout << a2[i] << ' ';
77      cout << endl;
78      a[3] = 100;
79      CArray a4(a);
80      for(int i = 0;i < a4.length();i++) cout << a4[i] << ' ';
81      return 0;
82  }
```

上面这个代码就可以理解为手写`vector`，需要注意下面几点：

- **73行的`a2 = a`中的等于号需要进行重载因为我们在对`size`复制的时候也希望把`ptr`也赋值给a2，所以一般的复制构造函数就行不通了。**
- **复制过程中，我们希望a,a2,a3,a4,都可以指向单独的一片空间，不然可能在释放空间的时候发生多次释放产生危险。所以每一次都要重新申请一片新的空间，见40-56行代码。**
- **输入函数`push_back()`中注意对新空间申请的时候，需要在原有基础上+1。为了减少时间复杂度，我们可以每次申请32（或64）的倍数的空间储存。这样空间复杂度可以降至`O(lgn)`左右（大概是这样）**

注：STL中提供`vector`容器。可以直接调用头文件`#include<vector>`使用

二.数据间类型的转换

1.转换构造函数

转换构造函数只有一个形参：

```
Complex(double x)
{
    real = x;
    imag = 0;
}
```

其作用是将`double`类型的参数x转换为`Complex`类的对象，将x作为复数的实部，虚部为0。

在类体中，可以有转换构造函数，也可以没有，视需要而定。假设`Complex`中定义了上述转换构造函数，在`Complex`作用域中：

```
Complex c1(3,5); //建立对象c1，由于只有一个参数，系统调用转换构造函数
```

它的作用就是将一个`double`型的常数转换成一个名为c1的`Complex`类的对象。

也可以建立无名对象：

```
Complex(3.6); //允许但是无法使用
```

也可以在表达式中应用：

```
c1 = Complex(3.6); //假设c1已经定义了
```

其作用是先建立一个无名对象，然后将他的值赋给c1，c1最后的值就是：`(3.6 + 0*i)`

在应用中，如果已对`+`进行了重载，使其可以对类对象相加。

```
c = c1 + 3.6; //非法！不能将一个类的对象和一个浮点数相加
```

怎么解决呢？就需要用到无名对象了：

```
c = c1 + Complex(3.6); //相当于一次强制类型转换
```

转换构造函数也是构造函数的一种，它遵循构造函数的一般规则。通常把含有一个参数的构造函数用作类型转换。所以称为转换构造函数。

注意：转换构造函数只有一个参数！！如果有多个参数，就不是转换构造函数~~（原因其实是显然的，如果提供了多个参数，那么系统就不知道到底需要转换那个参数了）~~

总结转换构造函数的使用过程：

1. **声明一个类。**
2. **在这个类中定义一个只有一个参数的转换构造函数。**
3. **利用如下形式：**

   `类名（指定数据的类型）`

2.类型转换函数

前面可以实现对一个整形或者浮点型操作转换成类的对象，但是如何将类的对象转换成其他类型？

C++提供了**类型转换函数。类型转换函数的作用是将一个类的对象转换成另一类型的数据**

如果已经声明了类`Complex`，那么在类中可以这样定义函数：

```
operator double()
{
    return real;
}
```

函数返回`double`型变量`real`的值。他的作用是将`Complex`的对象转换成一个`double`类型的数据，其值是类中成员`real`的值。注意：函数名是`operator double`这一点和运算符重载时的规律一致。

类型转换函数在函数名前不指定函数类型，函数没有参数，其返回值的类型是又函数名中指定的类型名来决定的。**类型转换函数只能作为成员函数，因为转换的主题是本类的对象，不能做为友元或者普通函数。**

~~提示：在这里可以理解为一个外国人拥有双重国籍。既`double`不仅有原始的含义，也有了重载后的含义，但是在不同情况下表现出不同的用处。就像一个人有双重国籍，在不同场合下以不同身份出现。~~

转换构造函数和类型转换函数都有一个共同的功能：当需要的时候，系统会自动调用这些函数，建立一个临时的无名对象或者临时变量。比如，我们定义了d1,d2为`double`，c1,c2为类的对象，若在程序中有一下表达式：

```
d1 = d2 + c1;
```

如果我们没有对`+`进行重载，那么系统就会检查有无类型转换函数，结果发现有对`double`的重载函数，就调用它，把对象c1转换成`double`型数据，建立一个临时的`double`数据，并和d2计算，最后赋值给d1.

如果程序中定义了转换构造函数，并且对`+`进行了重载。若有一下表达式：

```
c2 = c1 + d1;
```

编译系统寻找有无对`+`的重载，然后发现有`operator +`的存在，然后建立一个无名对象`Complex(d1)`，、再调用`operator+`函数，相当于执行表达式：

```
c2 = c1 + Complex(d2);
```

三.重载流插入运算符和流提取运算符

我们已经知道了C++中的输入输出流格式：`>>`和`<<`

但是，用户自己定义的数据类型，比如类对象，是无法直接这样输出输入的，必须对其进行重载：

对`>>`和`<<`的重载形式如下：

```
istream & operator >> ();
ostream & operator << ();
```

注意：只能将重载`>>`和`<<`的函数作为友元函数，不能将其定义为成员函数

~~解释一下：一般输入输出的格式都是`cout <<`和`cin >>`，所以真正接受流的是`cout`和`cin`。如果采用成员函数，函数的第一个参数就不是`istream&`或者`ostream&`了，而是需要进行操作的类。这样就无法实现插入和提取流的作用。~~

1.重载流插入运算符

```
```

```cpp
#include<bits/stdc++.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);
        friend ostream& operator <<(ostream& ,Complex&);
    private:
        double real;
        double imag;
};
Complex Complex::operator +(Complex& c2)
{
    return Complex(real + c2.real,imag + c2.imag);
}
ostream & operator <<(ostream& output,Complex& c)
{
    output << '(' << c.real << '+' << c.imag << 'i' << ')' << endl;;
    return output;
}
int main()
{
    Complex c1(2,4),c2(6,10),c3;
    c3 = c1 + c2;
    cout << c3;
    return 0;
}
```

说明：

- 重载运算符`<<`，`cout`是头文件中声明的`ostream`类的对象，`<<`右边是c3，是`Complex`类的对象。所以这里编译系统把`cout << c3`解释为

```cpp
operator << (cout,c3);
```

调用函数时，形参`output`成为了`cout`的引用，形参c成为c3的引用。

- 关于`return output`的作用：

  能连续向输出流插入信息。`output`是实参`cout`的引用，或者说是他的别名。因此，`return output`就是`return cout`将输出流的现状返回。

2.重载流提取运算符

```cpp
#include<bits/stdc++.h>
using namespace std;
class Complex
{
    public:
```

```
        friend istream& operator >>(istream&,Complex&);
        friend ostream& operator <<(ostream& ,Complex&);
    private:
        double real;
        double imag;
};
ostream & operator <<(ostream& output,Complex& c)
{
    output << '(' << c.real << '+' << c.imag << 'i' << ')' << endl;;
    return output;
}
istream & operator >>(istream& input,Complex& c)
{
    input >> c.real >> c.imag;
    return input;
}
int main()
{
    Complex c1,c2;
    cin >> c1 >> c2;
    cout << c1 << endl;
    cout << c2 << endl;
    return 0;
}
```

说明：

- 运算符`>>`重载函数中的形参`input`是`istream`类的对象`cin`的引用。
- `cin`和`>>`可以连续从输入流中提取数据给程序中的`Complex`对象。