

知识点1【可重载的运算符】（了解）

知识点2【重载自增 或自减 ++ --运算符】

知识点3【指针运算符 * ->】（了解）

知识点4【重载=运算符】（重要）

前提1：类中 没有指针成员 不需要重载=运算符（默认的浅拷贝就可以完成）

运行结果：

前提2：类中 有指针成员 必须重载=运算符

知识点5【等于和不等于(==、!=)运算符重载】

知识点6【函数调用符 () 的重载】（了解）

知识点7【不要重载&&、||】（了解）

不要重载&&、|| 因为 用户无法实现 && ||的短路特性。

知识点8【符号重载的总结】

知识点9【强化训练字符串类String】（重要）

知识点1【可重载的运算符】（了解）

可以重载的操作符

+	-	*	/	%	^	&		~
!	=	<	>	+=	-=	*=	/=	%=
^=	&=	=	<<	>>	>>=	<<=	=	!=
<=	>=	&&		++	--	->*	'	->
[]	()	new	delete	new[]	delete[]			

不能重载的算符

· :: .* ?: sizeof

知识点2 【重载自增 或自减 ++ --运算符】

operator++

编译器看到++a(前置++), 它就调用operator++(a),当编译器看到a++ (后置++), 它就会去调用operator++(a,int).

```
1  #include <iostream>
2
3  using namespace std;
4  class Data
5  {
6  friend ostream& operator<<(ostream &out, Data &ob);
7  private:
8      int a;
9      int b;
10 public:
11     Data()
12     {
13         cout<<"无参的构造函数"<<endl;
14         a = 0;
15         b=0;
16     }
17     Data(int a,int b):a(a),b(b)
18     {
19         cout<<"有参构造"<<endl;
20         //this->a = a;
21         //this->b = b;
22     }
23     void showData(void)
24     {
25         cout<<"a = "<<a<<", b= "<<b<<endl;
26     }
27     ~Data()
28     {
29         cout<<"析构造函数函数"<<endl;
30     }
31
32     //成员函数 重载前置++ ++ob1 (先加 后使用)
33     //编译器 默认识别 operator++(a) //但是a可以用this代替 从而化简 operator++()
34     Data& operator++()//++ob1
35     {
```

```
36 //先加
37 a++; //this->a = this->a +1
38 b++; //this->b = this->b +1
39 //后使用
40 return *this;
41 }
42 //成员函数 重载后置++ ob1++ (先使用 后加)
43 //编译器 默认识别 operator++(a,int) //但是a可以用this代替 从而化简 operator
44 Data& operator++(int)//ob1++
45 {
46 //先使用(备份加之前的值)
47 static Data old=*this;
48
49 //后加
50 a++;
51 b++;
52
53 //返回备份值
54 return old;
55 }
56
57 //重载前置-- --ob3
58 //编译器 默认识别 operator--(a) //但是a可以用this代替 从而化简 operator--()
59 Data& operator--()
60 {
61 //先减
62 a--;
63 b--;
64
65 //后使用(返回)
66 return *this;
67 }
68
69 //重载后-- ob4--
70 //编译器 默认识别 operator--(a,int) //但是a可以用this代替 从而化简 operator
71 Data& operator--(int)
72 {
73 //先使用
74 static Data old=*this;
```

```
75
76 //再减
77 a--;
78 b--;
79
80 return old;
81 }
82
83
84 };
85 //普通全局函数 作为类的友元 重载<<运算符
86 ostream& operator<<(ostream &out, Data &ob)
87 {
88     out<<"a = "<<ob.a<<", b = "<<ob.b;
89     return out;
90 }
91 void test01()
92 {
93     Data ob1(10,20);
94     ob1.showData();
95
96     //重载<<直接输出自定义对象的值
97     //operator<<(cout,ob1);
98     cout<<ob1<<endl;
99
100    //成员函数 重载 ++运算符
101    cout<<++ob1<<endl;
102
103    Data ob2(10,20);
104    cout<<ob2++<<endl;
105    cout<<ob2<<endl;
106
107    //成员函数 重载 --运算符
108    Data ob3(10,20);
109    cout<<"ob3 "<<ob3<<endl;
110    cout<<--ob3<<endl;
111
112    Data ob4(10,20);
113    cout<<"ob4 "<<ob4<<endl;
114    cout<<ob4--<<endl;
115    cout<<"ob4 "<<ob4<<endl;
```

```
116
117
118 }
119 int main(int argc, char *argv[])
120 {
121     test01();
122     return 0;
123 }
124
```

运行结果：

有参构造

a = 10, b = 20

a = 10, b = 20

a = 11, b = 21

有参构造

a = 10, b = 20

a = 11, b = 21

有参构造

ob3 a = 10, b = 20

a = 9, b = 19

有参构造

ob4 a = 10, b = 20

a = 10, b = 20

ob4 a = 9, b = 19

析构造函数

T& T::operator++() { // 执行递增 return *this; }	T& T::operator--() { // 执行递减 return *this; }	// 前缀形式: // - 完成任务 // - 总是返回 *this;
T T::operator++(int) { T old(*this); ++*this; return old; }	T T::operator--(int) { T old(*this); --*this; return old; }	// 后缀形式: // - 保存旧值 // - 调用前缀版本 // - 返回旧值

知识点3 【指针运算符 * ->】 (了解)

```

1  #include <iostream>
2
3  using namespace std;
4  class Person
5  {
6  private:
7      int num;
8  public:
9      Person(int num):num(num)
10     {
11         //this->num = num;
12         cout<<"有参构造num = "<<num<<endl;
13     }
14
15     void showPerson(void)
16     {
17         cout<<"num = "<<num<<endl;
18     }
19     ~Person()
20     {
21         cout<<"析构函数 num = "<<num<<endl;
22     }
23 };
24
25 //设计一个智能指针 解决 Person new出的堆区空间 释放问题
26 class SmartPointer{
27 public:

```

```
28  Person *pPerson;
29  public:
30      SmartPointer(Person *p)
31      {
32          pPerson = p;
33      }
34
35      ~SmartPointer()
36      {
37          if(pPerson != NULL)
38          {
39              delete pPerson;
40              pPerson = NULL;
41          }
42      }
43
44      //成员函数重载->运算符
45      Person* operator->()
46      {
47          return this->pPerson;
48      }
49
50      //成员函数 重载 *运算
51      Person& operator*()
52      {
53          return *(this->pPerson);
54      }
55  };
56  void test01()
57  {
58      Person *p = new Person(100);
59      p->showPerson();
60
61      //假如我忘了 delete p
62      //delete p;
63
64      //需求: 自动的帮我释放 堆区空间 (智能指针的概念)
65      SmartPointer pointer(new Person(200));
66
67      //访问Person类中的showPerson()
```

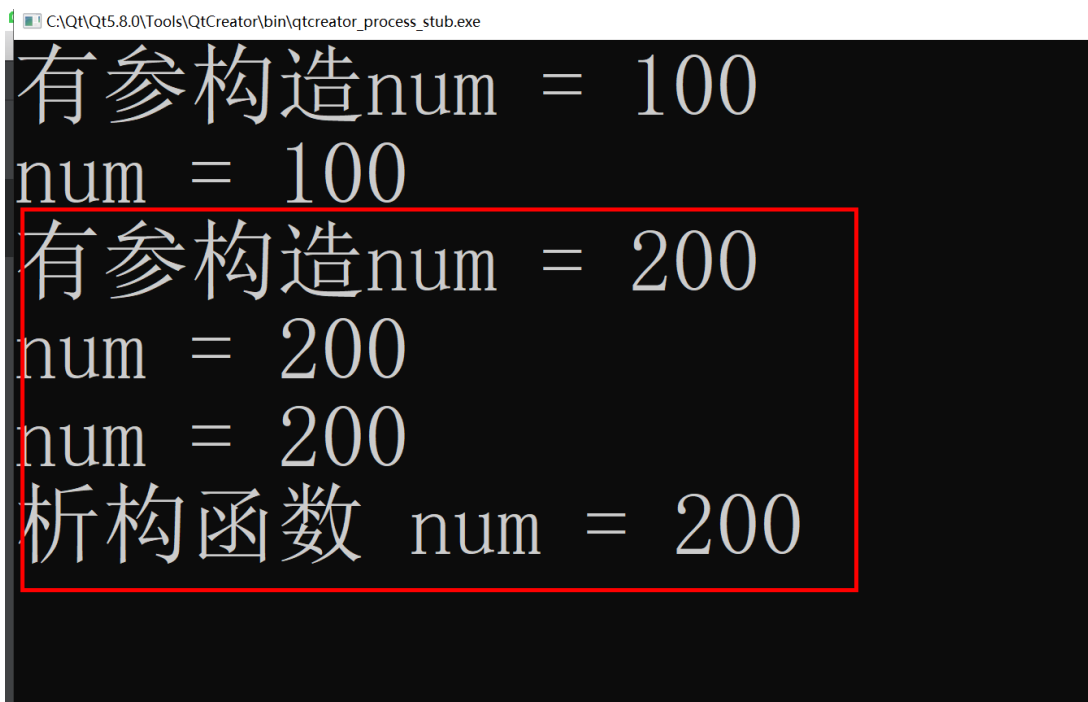


```

68 //pointer.pPerson->showPerson();
69
70 //保证指针的使用
71 //(pointer.operator ->())->showPerson();
72 pointer->showPerson();
73 (*pointer).showPerson();
74
75 }
76 int main(int argc, char *argv[])
77 {
78     test01();
79     return 0;
80 }

```

运行结果：



```

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
有参构造num = 100
num = 100
有参构造num = 200
num = 200
num = 200
析构函数 num = 200

```

知识点4 【重载=运算符】（重要）

前提1：类中 **没有指针成员** 不需要重载=运算符（默认的浅拷贝就可以完成）

```

1 #include <iostream>
2
3 using namespace std;
4 class Person
5 {
6     private:

```

```

7   int a;
8   int b;
9   public:
10  Person():a(0),b(0)
11  {
12      cout<<"无参构造"<<endl;
13  }
14  Person(int a, int b):a(a),b(b)
15  {
16      cout<<"有参构造"<<endl;
17  }
18  void showPerson(void)
19  {
20      cout<<"a = "<<a<<", b = "<<b<<endl;
21  }
22  ~Person()
23  {
24      cout<<"析构函数"<<endl;
25  }
26 };
27 void test01()
28 {
29     Person ob1(10,20);
30     ob1.showPerson();
31
32     //注意 旧对象 给新对象赋值 调用的是拷贝构造（默认拷贝构造就是单纯的赋值）
33     Person ob2 = ob1;//这个地方 可不是调用赋值=运算符
34     ob2.showPerson();
35
36     Person ob3;
37     ob3 = ob1;//此处才是调用的赋值=运算符(默认赋值=运算是浅拷贝)
38     ob3.showPerson();
39 }
40 int main(int argc, char *argv[])
41 {
42     test01();
43     return 0;
44 }

```

运行结果：

有参构造
运 a = 10, b = 20
a = 10, b = 20
无参构造
a = 10, b = 20
析构函数
析构函数
析构函数

前提2：类中 **有指针成员** 必须重载=运算符

指针作为类的成员：

- 1、拷贝构造函数 必须自定义（默认拷贝构造 是浅拷贝）
- 2、必须重载=运算符（默认=号运算符 是浅拷贝）

```
1 #include <iostream>
2 #include<string.h>
3 using namespace std;
4 class Person
5 {
6     private:
7         char *name;//指针成员
8     public:
9         Person()
10    {
```

```
11  name = NULL;
12  cout<<"无参构造"<<endl;
13  }
14  Person(char *name)
15  {
16  //根据实际传入的 参数 给this->name申请空间
17  this->name = new char[strlen(name)+1];
18
19  //将name指向的字符串 拷贝到 this->name指向的空间中
20  strcpy(this->name,name);
21
22  cout<<"有参构造"<<endl;
23  }
24  Person(const Person &ob)//ob代表的就是旧对象
25  {
26  //this代表的是新对象
27  cout<<"拷贝构造函数"<<endl;
28  this->name = new char[strlen(ob.name)+1];
29  strcpy(this->name, ob.name);
30  }
31
32
33  ~Person()
34  {
35  cout<<"析构函数"<<endl;
36  if(this->name != NULL)
37  {
38  delete [] this->name;
39  this->name = NULL;
40  }
41  }
42
43  void showPerson(void)
44  {
45  cout<<"name = "<<name<<endl;
46  }
47  //成员函数 重载=运算符
48  Person& operator=(Person &ob)//ob == ob1
49  {
50  //this ==>&ob3
```

```

51  if(this->name != NULL)//说明this->name 以前有指向(重点)
52  {
53      //释放以前指向的空间
54      delete [] this->name;
55      this->name = NULL;
56  }
57
58      //申请空间
59      this->name = new char[strlen(ob.name)+1];
60      //拷贝内容
61      strcpy(this->name,ob.name);
62
63      return *this;//重点
64  }
65  };
66
67  void test01()
68  {
69      Person ob1("lucy");
70      ob1.showPerson();
71
72      Person ob2 = ob1;//调用拷贝构造
73
74      Person ob3("bob");
75      //不重载 = 默认是浅拷贝
76      ob3 = ob1;
77
78      ob3.showPerson();
79
80      Person ob6,ob5,ob4;
81      ob6 = ob5 = ob4 = ob1;
82      ob6.showPerson();
83  }
84  int main(int argc, char *argv[])
85  {
86      test01();
87      return 0;
88  }
89

```

运行结果:

 C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe

有参构造

```
name = Lucy
```

拷贝构造函数

有参构造

```
name = Lucy
```

无参构造

无参构造

无参构造

```
name = Lucy
```

析构函数

析构函数

析构函数

析构函数

析构函数 析构函数 析构函数

知识点5 【等于和不等于(==、!=)运算符重载】

```
1 //重载== ==出现在判断语句中
2 bool operator==(Person &ob)
3 {
4     if(strcmp(this->name, ob.name) == 0)
5     {
6         return true;
7     }
8     return false;
9 }
10 //重载!= !=出现在判断语句中
11 bool operator!=(Person &ob)
12 {
13     if(strcmp(this->name, ob.name) != 0)
14     {
15         return true;
16     }
17     return false;
18 }
```

```
1 void test02()
2 {
3     Person ob1("lucy");
4     Person ob2("lucy");
5     Person ob3("bob");
6
7     if(ob1 == ob2)
8     {
9         cout<<"ob1 == ob2"<<endl;
10    }
11    else
```

```
12  {  
13  cout<<"ob1 != ob2"<<endl;  
14  }  
15  
16  if(ob1 != ob3)  
17  {  
18  cout<<"ob1 != ob3"<<endl;  
19  }  
20  else  
21  {  
22  cout<<"ob1 == ob3"<<endl;  
23  }  
24  }
```

运行结果:

有参构造
有参构造
有参构造
ob1 == ob2
ob1 != ob3
析构函数
析构函数
析构函数

知识点6【函数调用符（）的重载】（了解）

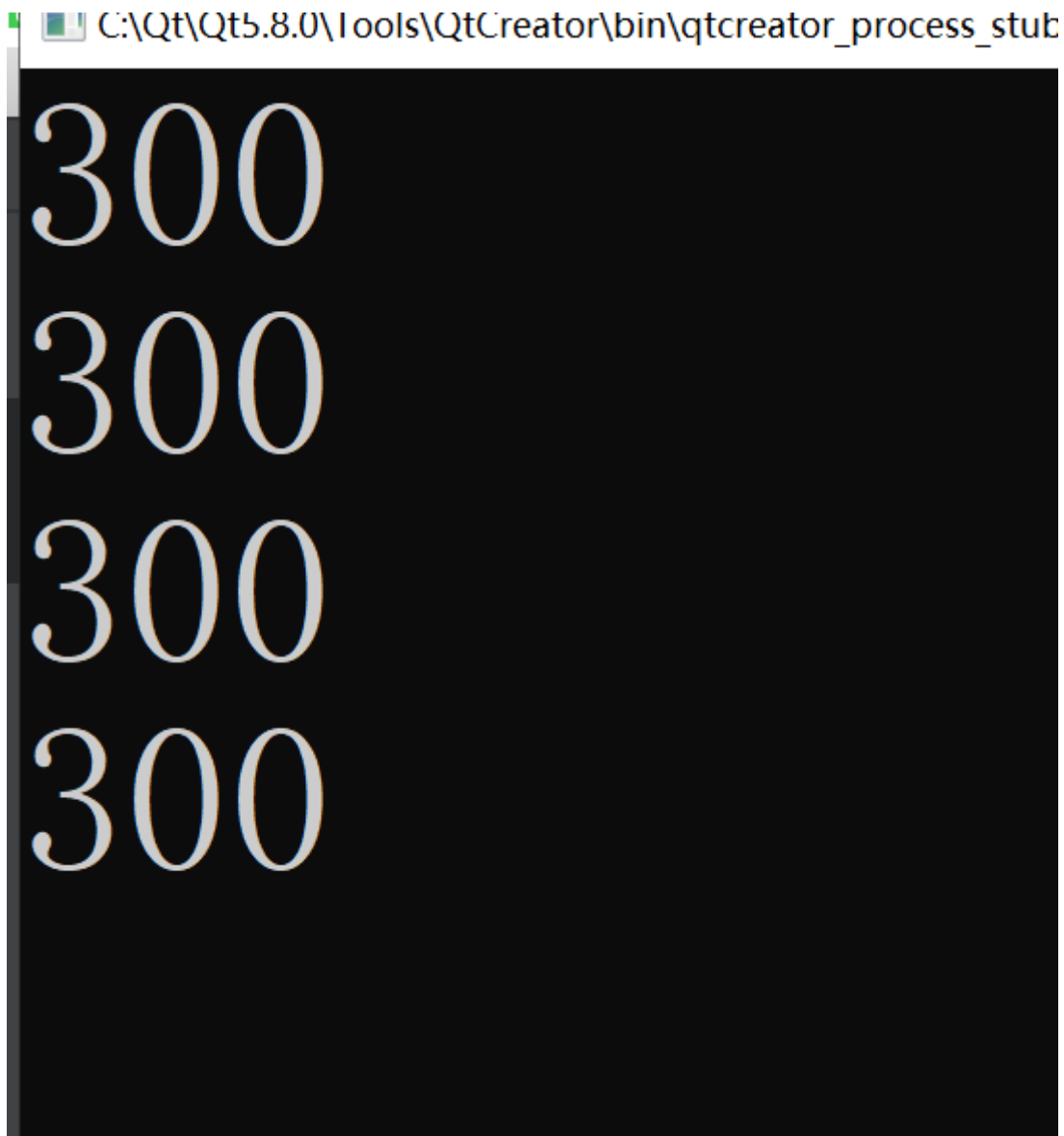
```
1 #include <iostream>  
2
```

```

3 using namespace std;
4 class Fun
5 {
6 public:
7     int my_add(int x,int y)
8     {
9         return x+y;
10    }
11    //重载()
12    //第一个()是重载的符号 第二个()是标明要传参
13    int operator()(int x,int y)
14    {
15        return x+y;
16    }
17 };
18
19 void test01()
20 {
21     Fun fun;
22     cout<<fun.my_add(100,200)<<endl;
23
24     cout<<fun.operator ()(100,200)<<endl;
25     //优化 fun和()结合 就会自动寻找()运算符
26     cout<<fun(100,200)<<endl;
27     //此处 fun(100,200)不是一个真正的函数 仅仅是一个对象名和()结合 调用()重载运算符而已
28     //fun不是函数名 只是fun(100,200)类似一个函数调用 所以将fun(100,200)叫做仿函数
29
30     //此处的Fun是类名称
31     //Fun()匿名对象 Fun()(100,200) 就是匿名对象(100,200)
32     cout<<Fun()(100,200)<<endl;//了解
33
34 }
35 int main(int argc, char *argv[])
36 {
37     test01();
38     return 0;
39 }
40

```

运行结果：



知识点7 【不要重载&&、||】（了解）

不要重载&&、|| 因为 用户无法实现 && ||的短路特性。

&& 短路特性： A && B 如果A为假 B将不会执行

|| 短路特性： A || B 如果A为真 B将不会执行

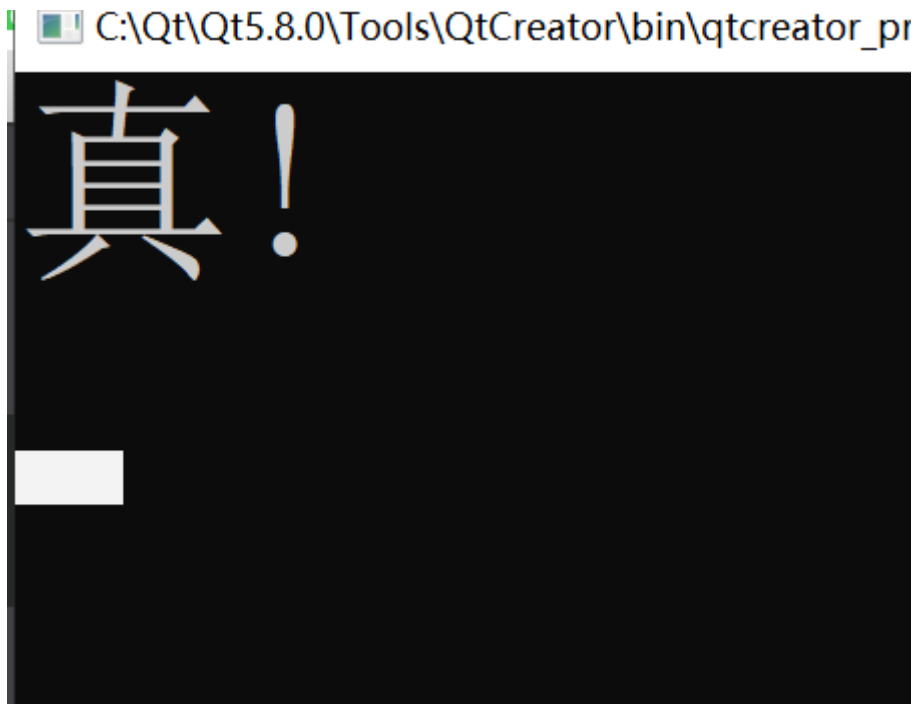
```
1  #include <iostream>
2
3  using namespace std;
4
5  class Complex{
6  public:
7      Complex(int flag){
8          this->flag = flag;
9      }
10     Complex& operator+=(Complex& complex){
```

```

11  this->flag = this->flag + complex.flag;
12  return *this;
13  }
14  bool operator&&(Complex& complex){
15  return this->flag && complex.flag;
16  }
17  public:
18  int flag;
19  };
20  int main(){
21
22  Complex complex1(0); //flag 0
23  Complex complex2(1); //flag 1
24
25  //原来情况，应该从左往右运算，左边为假，则退出运算，结果为假
26  //这边却是，先运算（complex1+complex2），导致，complex1的flag变为complex1+
  complex2的值， complex1.a = 1
27  // 1 && 1
28  //complex1.operator&&(complex1.operator+=(complex2))
29  if (complex1 && (complex1 += complex2)){
30  //complex1.operator+=(complex2)
31  cout << "真!" << endl;
32  }
33  else{
34  cout << "假!" << endl;
35  }
36
37  return EXIT_SUCCESS;
38  }
39

```

运算结果：



&&的短路特性 期望的结果是假，但是程序的结果为真。

知识点8 【符号重载的总结】

4.6.10 符号重载总结

=, [], () 和 -> 操作符只能通过成员函数进行重载 << 和 >> 只能通过全局函数配合友元函数进行重载 不要重载 && 和 || 操作符，因为无法实现短路规则 常规建议

运算符	建议使用
所有的一元运算符	成员
= () [] -> ->*	必须是成员
+= -= /= *= ^= &= != %= >>= <<=	成员
其它二员运算符	非成员

知识点9 【强化训练字符串类String】（重要）

String 包含字符串 以及 字符串的操作。

mystring.h

```
1 #ifndef MYSTRING_H
2 #define MYSTRING_H
3 #include<iostream>
4 using namespace std;
5 class MyString
6 {
7     friend ostream& operator<<(ostream &out, MyString &ob);
```

```

8  friend istream& operator>>(istream &in, MyString &ob);
9  private:
10  char *str;
11  int size;
12  public:
13  MyString();
14  MyString(const char *str);
15  MyString(const MyString &ob);
16  ~MyString();
17  int Size(void);
18
19  //重载[]
20  char& operator[](int index);
21  };
22
23 #endif // MYSTRING_H
24

```

mystring.cpp

```

1  #include "mystring.h"
2  #include<string.h>
3  #include<iostream>
4  using namespace std;
5  MyString::MyString()
6  {
7  this->str = NULL;
8  this->size = 0;
9  cout<<"无参构造"<<endl;
10 }
11
12 MyString::MyString(const char *str)
13 {
14 cout<<"char *构造函数"<<endl;
15 //申请空间
16 this->str = new char[strlen(str)+1];
17 //拷贝字符串
18 strcpy(this->str, str);
19
20 //更新size
21 this->size = strlen(str);
22 }

```

```
23
24 MyString::MyString(const MyString &ob)
25 {
26     cout<<"拷贝构造函数"<<endl;
27     //申请空间
28     this->str = new char[strlen(ob.str)+1];
29     //拷贝字符串
30     strcpy(this->str, ob.str);
31
32     //更新size
33     this->size = ob.size;
34 }
35
36 MyString::~MyString()
37 {
38     cout<<"析构函数"<<endl;
39     if(this->str != NULL)
40     {
41         delete [] this->str;
42         this->str = NULL;
43     }
44 }
45
46 int MyString::Size()
47 {
48     return this->size;
49 }
50
51 char& MyString::operator[](int index)//index表示数组的下标
52 {
53     //判断下标是否合法
54     if(index >=0 && index < this->size)
55     {
56         return this->str[index];
57     }
58     else
59     {
60         cout<<"index无效"<<endl;
61     }
62 }
```

```

63 ostream& operator<<(ostream &out, MyString &ob)
64 {
65     out<<ob.str; //访问了ob中的私有数据 必须设置成友元
66     return out;
67 }
68
69 istream& operator>>(istream &in, MyString &ob)
70 {
71     //记得将原有的数据清楚
72     if(ob.str != NULL)
73     {
74         delete [] ob.str;
75         ob.str = NULL;
76     }
77
78     //获取键盘输入的字符串
79     char buf[1024] = ""; //临时buf
80     in >> buf; //先得到键盘输入的数据 然后根据buf的实际大小 开辟空间
81
82     ob.str = new char[strlen(buf)+1];
83     strcpy(ob.str, buf);
84     ob.size = strlen(buf);
85
86     return in;
87 }
88

```

main.cpp

```

1  #include <iostream>
2  #include "mystring.h"
3  using namespace std;
4
5  int main(int argc, char *argv[])
6  {
7      MyString str1("hehe");
8      //自定义对象 必须重载<< (普通全局友元函数实现)
9      cout<<str1<<endl;
10     cout<<"size = "<<str1.Size()<<endl;
11
12     //自定义对象 必须重载>> (普通全局友元函数实现)
13     cin>>str1;

```



```
14  cout<<str1<<endl;
15  cout<<"size = "<<str1.Size()<<endl;
16
17  MyString str2("hello class");
18  //重载[]运算符
19  cout<<str2[1]<<endl;
20
21  //重载[]运算符 返回值必须是左值 才能写操作
22  //重载[]运算符 的返回值必须是引用
23  str2[1] = 'E';
24  cout<<str2<<endl;
25
26  return 0;
27  }
28
```

运行结果：

char *构造函数

hehe

size = 4

手动输入

hehe

hehe

size = 4

char *构造函数

e

hEllo class

析构造函数

析构造函数

■

