## 知识点1【运算符重载】（重要）

运算符重载 是对已有的运算符 指定新功能。不能创建新运算。

运算符重载关键字operator

**思路：**

1、弄懂运算符的运算对象的个数。 （个数决定了 重载函数的参数个数）

2、识别运算符左边的运算对象 是类的对象 还是其他.

类的对象：全局函数实现（不推荐） 成员函数实现（推荐，少一个参数）

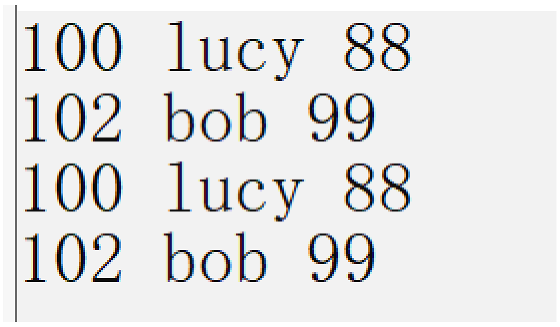
其他：只能是全局函数实现

## 1、重载<<运算符（全局函数实现）

```cpp
#include <iostream>
#include <string>
using namespace std;
class Person
{
 friend ostream & operator<<(ostream &out, Person &ob);
private:
 int num;
 string name;
  float score;
 public:
  Person(){}
  Person(int num, string name, float score):num(num),name(name),score(sco
re){}

 };
 ostream & operator<<(ostream &out, Person &ob)
 {
  out<<ob.num<<" "<<ob.name<<" "<<ob.score<<endl;
  return out;
 }

 int main(int argc, char *argv[])
 {
  Person lucy(100,"lucy", 99.8f);
  Person bob(101,"bob", 88.8f);
  cout<<lucy<<bob<<lucy<<bob;//operator<<(cout, lucy);
  return 0;
 }

```

## 2、重载输入>>运算符

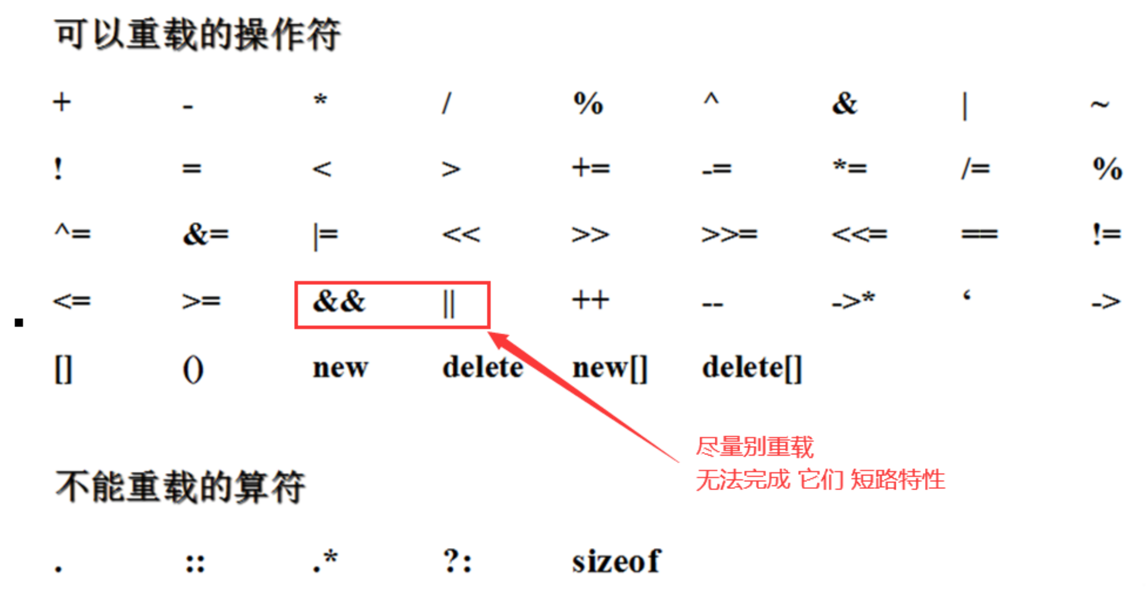
```
#include <iostream>
#include <string>
using namespace std;
class Person
{
 friend ostream & operator<<(ostream &out, Person &ob);
 friend istream &operator>>(istream &in, Person &ob);
private:
 int num;
  string name;
  float score;
public:
  Person(){}
  Person(int num, string name, float score):num(num),name(name),score(sco
re){}

 };
 ostream & operator<<(ostream &out, Person &ob)
 {
  out<<ob.num<<" "<<ob.name<<" "<<ob.score<<endl;
  return out;
 }
 istream &operator>>(istream &in, Person &ob)
 {
  in>>ob.num>>ob.name>>ob.score;
  return in;
 }

 int main(int argc, char *argv[])
 {
  Person lucy;
  Person bob;

  cin>>lucy>>bob;

  cout<<lucy<<bob<<endl;
  return 0;
 }

```



```
100 lucy 88
102 bob 99
100 lucy 88
102 bob 99
```

如果使用全局函数 重载运算符 必须将全局函数设置成友元。

## 3、可以重载的运算符



可以重载的操作符

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

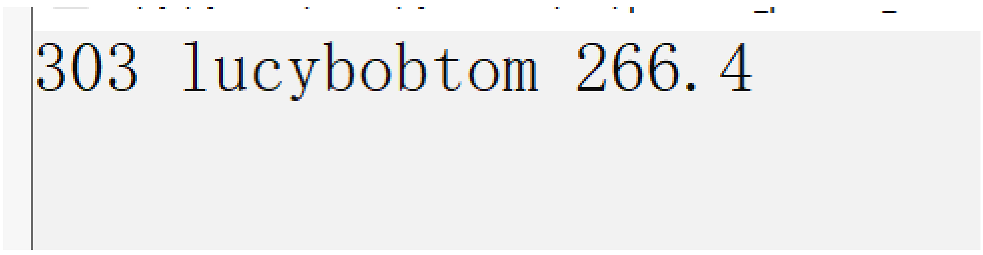
尽量别重载
无法完成 它们 短路特性

不能重载的算符

|  |  |  |  |  |
|---|---|---|---|---|
| . | :: | .* | ?: | sizeof |

## 4、重载加法运算符+（全局函数实现）

```cpp
#include <iostream>
#include <string>
using namespace std;
class Person
{
 friend ostream & operator<<(ostream &out, Person ob);
 friend istream &operator>>(istream &in, Person &ob);
 friend Person operator+(Person ob1, Person ob2);
private:
  int num;
  string name;
  float score;
public:
  Person(){}
  Person(int num, string name, float score):num(num),name(name),score(score){}

};
ostream & operator<<(ostream &out, Person ob)
{
 out<<ob.num<<" "<<ob.name<<" "<<ob.score<<endl;
 return out;
}
istream &operator>>(istream &in, Person &ob)
{
 in>>ob.num>>ob.name>>ob.score;
 return in;
}
Person operator+(Person ob1, Person ob2)
{
 Person tmp;
 tmp.num = ob1.num+ob2.num;
 tmp.name = ob1.name+ob2.name;
 tmp.score = ob1.score+ob2.score;

 return tmp;
}

int main(int argc, char *argv[])
{
 Person lucy(100,"lucy", 88.8f);
 Person bob(101,"bob", 99.9f);
 Person tom(102,"tom", 77.7f);

 cout<<lucy+bob+tom<<endl;
 return 0;
}
```

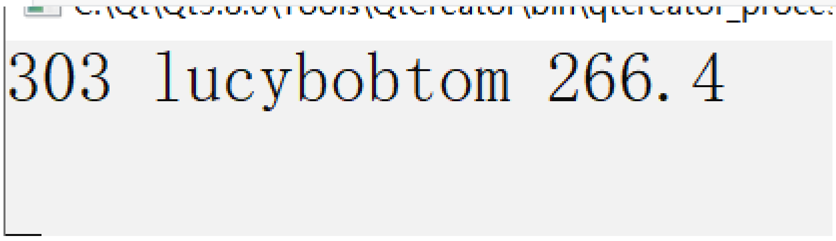


```
303 lucybobtom 266.4
```

## 5、重载加法运算符+（成员函数实现 推荐）

```cpp
#include <iostream>
#include <string>
```

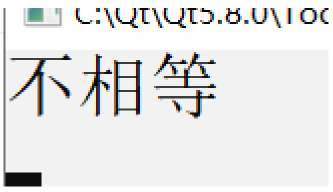
```cpp
using namespace std;
class Person
{
 friend ostream & operator<<(ostream &out, Person ob);
 friend istream &operator>>(istream &in, Person &ob);
private:
 int num;
  string name;
  float score;
 public:
  Person(){}
  Person(int num, string name, float score):num(num),name(name),score(score){}
  //成员函数重载+
  Person operator+(Person ob)
  {
  Person tmp;
  tmp.num = this->num + ob.num;
  tmp.name = this->name + ob.name;
  tmp.score = this->score + ob.score;
  return tmp;
  }

 };
 ostream & operator<<(ostream &out, Person ob)
 {
  out<<ob.num<<" "<<ob.name<<" "<<ob.score<<endl;
  return out;
 }
 istream &operator>>(istream &in, Person &ob)
 {
  in>>ob.num>>ob.name>>ob.score;
  return in;
 }

 int main(int argc, char *argv[])
 {
  Person lucy(100,"lucy", 88.8f);
  Person bob(101,"bob", 99.9f);
  Person tom(102,"tom", 77.7f);
  //lucy+bob;//lucy.operator+(bob)
  cout<<lucy+bob+tom<<endl;
  return 0;
 }
```



```
303 lucybobtom 266.4
```

## 6、重载==运算符（成员函数实现 推荐）

```
#include <iostream>
#include <string>
using namespace std;
class Person
{
 friend ostream & operator<<(ostream &out, Person ob);
 friend istream &operator>>(istream &in, Person &ob);
private:
 int num;
  string name;
  float score;
 public:
  Person(){}
  Person(int num, string name, float score):num(num),name(name),score(sco
re){}
  //成员函数重载+
  Person operator+(Person ob)
  {
  Person tmp;
  tmp.num = this->num + ob.num;
  tmp.name = this->name + ob.name;
  tmp.score = this->score + ob.score;
  return tmp;
  }
  //成员函数重载==
  bool operator==(Person &ob)
  {
  if(num == ob.num && name==ob.name && score==ob.score)
  return true;
  return false;
  }

 };
 ostream & operator<<(ostream &out, Person ob)
 {
  out<<ob.num<<" "<<ob.name<<" "<<ob.score<<endl;
  return out;
 }
 istream &operator>>(istream &in, Person &ob)
 {
  in>>ob.num>>ob.name>>ob.score;
  return in;
 }

 int main(int argc, char *argv[])
 {
  Person lucy(100,"lucy", 88.8f);
  Person bob(101,"bob", 99.9f);
  if(lucy == bob)
  {
  cout<<"相等"<<endl;
  }
  else
  {
  cout<<"不相等"<<endl;
  }
  return 0;
 }
```


```
不相等
```

## 7、重载++运算符

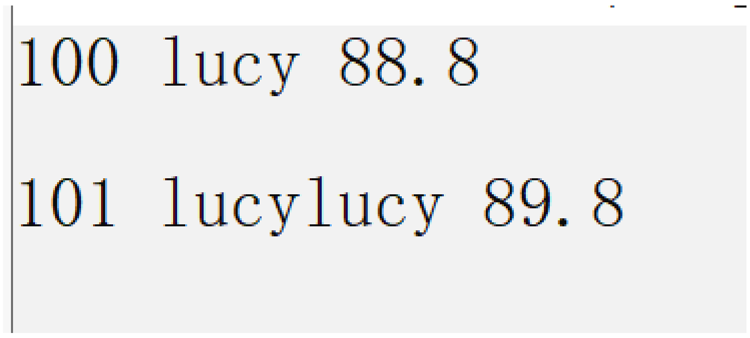
++a(前置++)，它就调用operator++(a),

a++ （后置++），它就会去调用operator++(a,int)

**案例1：重载后置++**

```
类名称 operator++(int)
{
 //先保存 旧的值
 //自增++
 return old;//返回旧值
}
```

```
#include <iostream>
#include <string>
using namespace std;
class Person
{
 friend ostream & operator<<(ostream &out, Person ob);
 friend istream &operator>>(istream &in, Person &ob);
private:
 int num;
  string name;
  float score;
 public:
  Person(){}
  Person(int num, string name, float score):num(num),name(name),score(score){}
  //成员函数重载+
  Person operator+(Person ob)
  {
  Person tmp;
  tmp.num = this->num + ob.num;
  tmp.name = this->name + ob.name;
  tmp.score = this->score + ob.score;
  return tmp;
  }
  //成员函数重载==
  bool operator==(Person &ob)
  {
  if(num == ob.num && name==ob.name && score==ob.score)
  return true;
  return false;
  }
  //重载后置++ operator++(a,int)
  Person operator++(int)
  {
  //先保存 旧的值
  Person old = *this;//*this == lucy

  //lucy++ ==> lucy = lucy+1
  this->num = this->num +1;
  this->name = this->name+this->name;//（自定义操作）
  this->score = this->score+1;

  return old;//返回旧值
  }


 };
 ostream & operator<<(ostream &out, Person ob)
 {
  out<<ob.num<<" "<<ob.name<<" "<<ob.score<<endl;
  return out;
 }
 istream &operator>>(istream &in, Person &ob)
 {
  in>>ob.num>>ob.name>>ob.score;
  return in;
 }

 int main(int argc, char *argv[])
 {
  Person lucy(100,"lucy", 88.8f);
  Person bob;
  //先使用 后++
  bob = lucy++;//
  cout<<bob<<endl;
  cout<<lucy<<endl;
  return 0;
 }

```

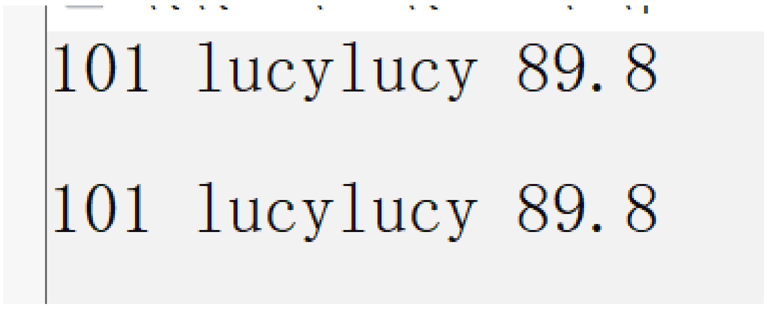


```
100 lucy 88.8

101 lucylucy 89.8
```

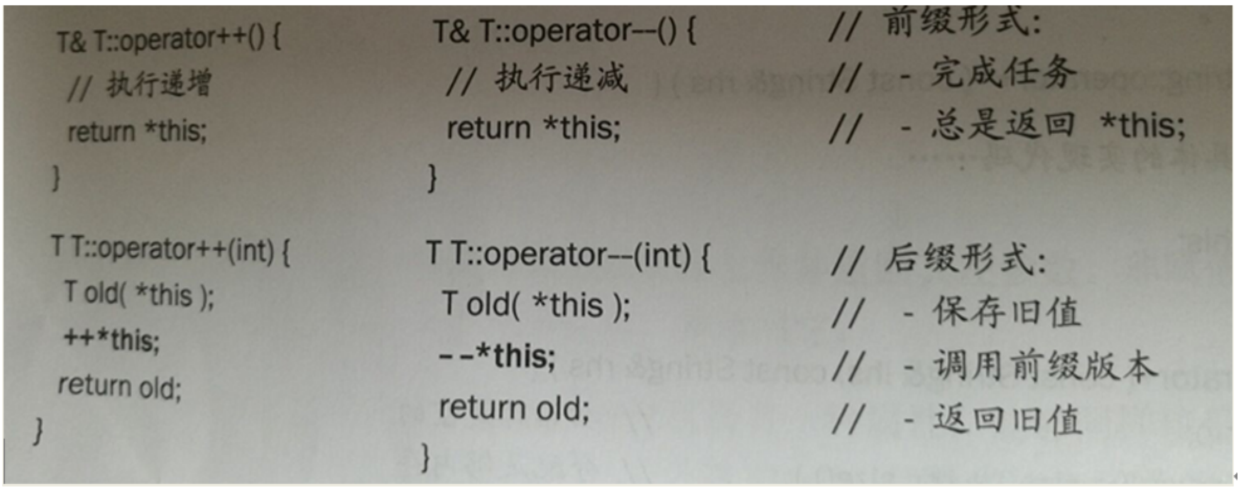
**案例2：重载前置++**

```cpp
#include <iostream>
#include <string>
using namespace std;
class Person
{
 friend ostream & operator<<(ostream &out, Person ob);
 friend istream &operator>>(istream &in, Person &ob);
private:
 int num;
 string name;
 float score;
public:
 Person(){}
 Person(int num, string name, float score):num(num),name(name),score(score){}
 //成员函数重载+
 Person operator+(Person ob)
 {
 Person tmp;
 tmp.num = this->num + ob.num;
 tmp.name = this->name + ob.name;
 tmp.score = this->score + ob.score;
 return tmp;
 }
 //成员函数重载==
 bool operator==(Person &ob)
 {
 if(num == ob.num && name==ob.name && score==ob.score)
 return true;
 return false;
 }
 //重载后置++ operator++(a,int)
 Person operator++(int)
 {
 //先保存 旧的值
 Person old = *this;//*this == lucy

 //lucy++ ==> lucy = lucy+1
 this->num = this->num +1;
 this->name = this->name+this->name;//（自定义操作）
 this->score = this->score+1;

 return old;//返回旧值
 }
 //重载前置++ operator++(a)
 Person operator++()
 {
 //先++
 this->num = this->num +1;
 this->name = this->name+this->name;//（自定义操作）
 this->score = this->score+1;

 //后使用
 return *this;
 }

};
ostream & operator<<(ostream &out, Person ob)
{
 out<<ob.num<<" "<<ob.name<<" "<<ob.score<<endl;
 return out;
}
istream &operator>>(istream &in, Person &ob)
{
 in>>ob.num>>ob.name>>ob.score;
 return in;
}

int main(int argc, char *argv[])
{
 Person lucy(100,"lucy", 88.8f);
 Person bob;
```

```
//先++ 后使用
bob = ++lucy;//
cout<<bob<<endl;
cout<<lucy<<endl;
return 0;
}
```



```
101 lucylucy 89.8

101 lucylucy 89.8
```



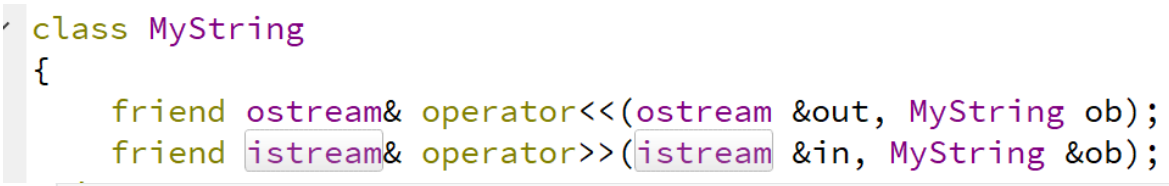
```
T& T::operator++() {          T& T::operator--() {          // 前缀形式:
  // 执行递增                    // 执行递减                    //  - 完成任务
  return *this;                 return *this;                //  - 总是返回 *this;
}                             }

T T::operator++(int) {        T T::operator--(int) {        // 后缀形式:
  T old( *this );               T old( *this );              //  - 保存旧值
  ++*this;                      --*this;                     //  - 调用前缀版本
  return old;                   return old;                  //  - 返回旧值
}                             }
```

## 知识点2【string类】（了解）

### 1、构造和析构函数

```
MyString::MyString()
{
 str=NULL;
 size=0;
}

MyString::MyString(char *str)
{
 size = strlen(str);
 this->str = new char[size+1];
 memset(this->str, 0, size+1);

 strcpy(this->str, str);
}

MyString::MyString(const MyString &ob)
{
 size = ob.size;
 str = new char[size+1];
 memset(str, 0, size+1);
 strcpy(str, ob.str);
}

MyString::~MyString()
{
 if(str != NULL)
 {
 delete [] str;
 str=NULL;
 }
}
```

## 2、重载输入输出

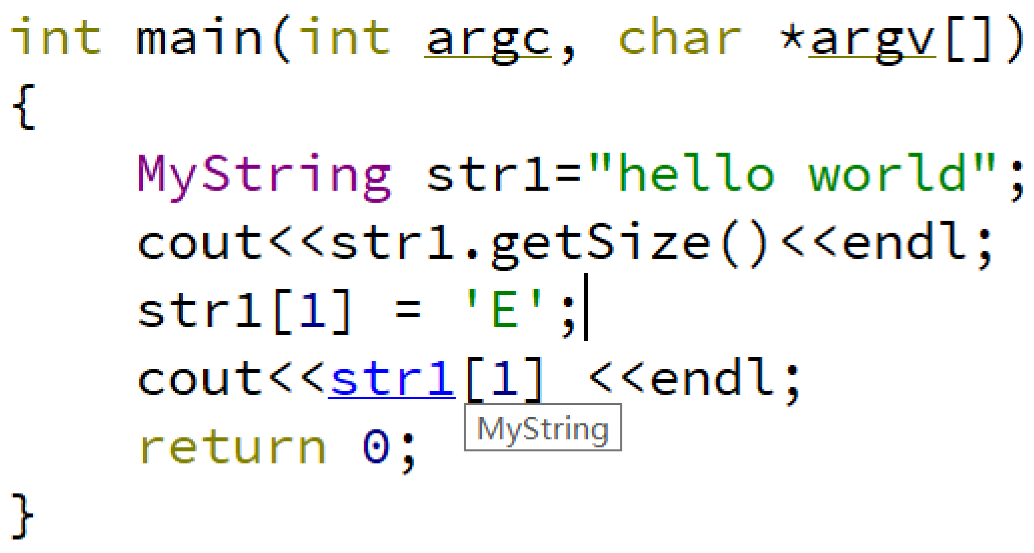


```
class MyString
{
    friend ostream& operator<<(ostream &out, MyString ob);
    friend istream& operator>>(istream &in, MyString &ob);
```

```
//全局函数实现 <<重载
ostream& operator<<(ostream &out, MyString ob)
{
 out<<ob.str;
 return out;
}
//全局函数实现 >>重载
istream& operator>>(istream &in, MyString &ob)
{
 char buf[1024]="";
 cin>>buf;

 if(ob.str != NULL)//ob已经有字符串
 {
 delete [] ob.str;
 ob.str = NULL;
 }

 ob.size = strlen(buf);
 ob.str = new char[ob.size+1];
 memset(ob.str, 0,ob.size+1);
 strcpy(ob.str, buf);

 return in;
}
```

## 3、重载中括号运算符

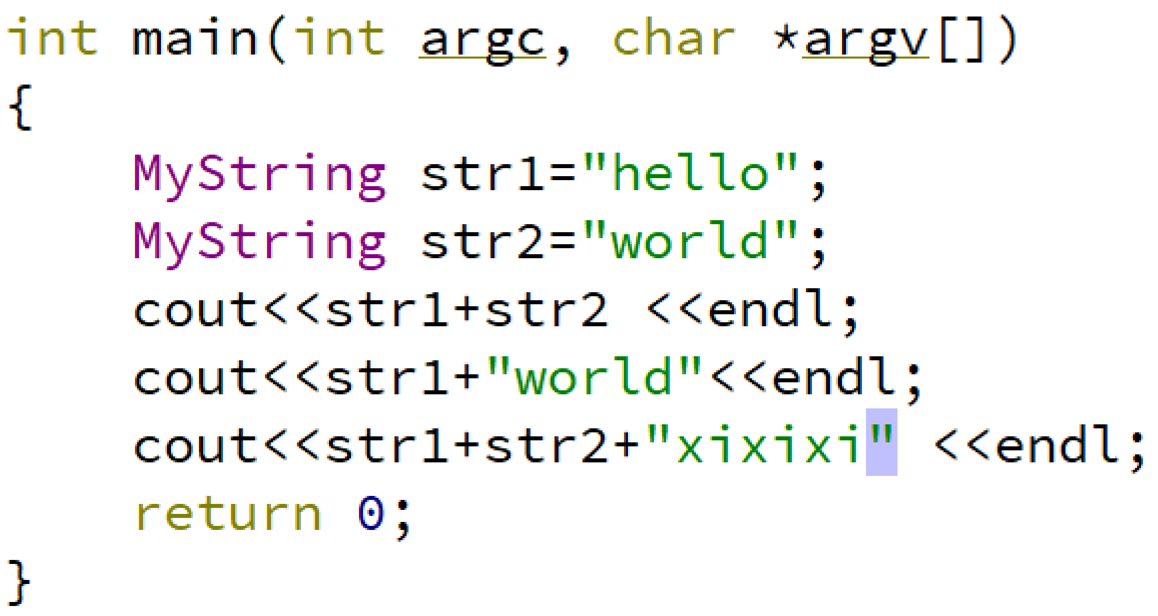
```
char& MyString::operator[](int pos)
{
 if(pos<0 || pos>=size)
 {
 cout<<"元素位置不合法"<<endl;
 exit(-1);
 }

 return str[pos];
}
```



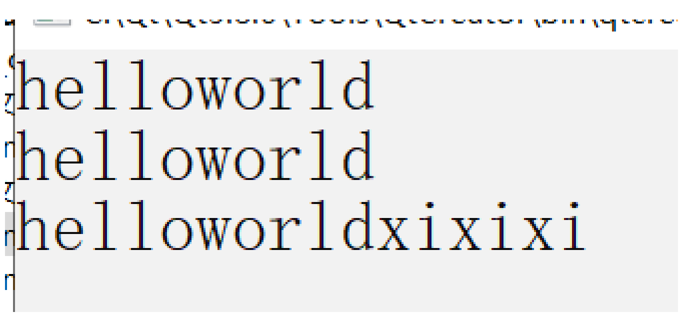
```
int main(int argc, char *argv[])
{
    MyString str1="hello world";
    cout<<str1.getSize()<<endl;
    str1[1] = 'E';
    cout<<str1[1] <<endl;
    return 0;
}
```

## 4、重载+运算符

```
MyString MyString::operator+(MyString ob)
{
 MyString tmp;
 tmp.size = size+ob.size;
 tmp.str = new char[tmp.size+1];
 memset(tmp.str, 0, tmp.size+1);

strcpy(tmp.str, str);
strcat(tmp.str, ob.str);

 return tmp;
}

MyString MyString::operator+(char *str)
{
 MyString tmp;
 tmp.size = size+strlen(str);
 tmp.str = new char[tmp.size+1];
 memset(tmp.str, 0, tmp.size+1);

 strcpy(tmp.str, this->str);
 strcat(tmp.str, str);

 return tmp;
}

```



```
int main(int argc, char *argv[])
{
    MyString str1="hello";
    MyString str2="world";
    cout<<str1+str2 <<endl;
    cout<<str1+"world"<<endl;
    cout<<str1+str2+"xixixi" <<endl;
    return 0;
}
```



```
helloworld
helloworld
helloworldxixixi
```

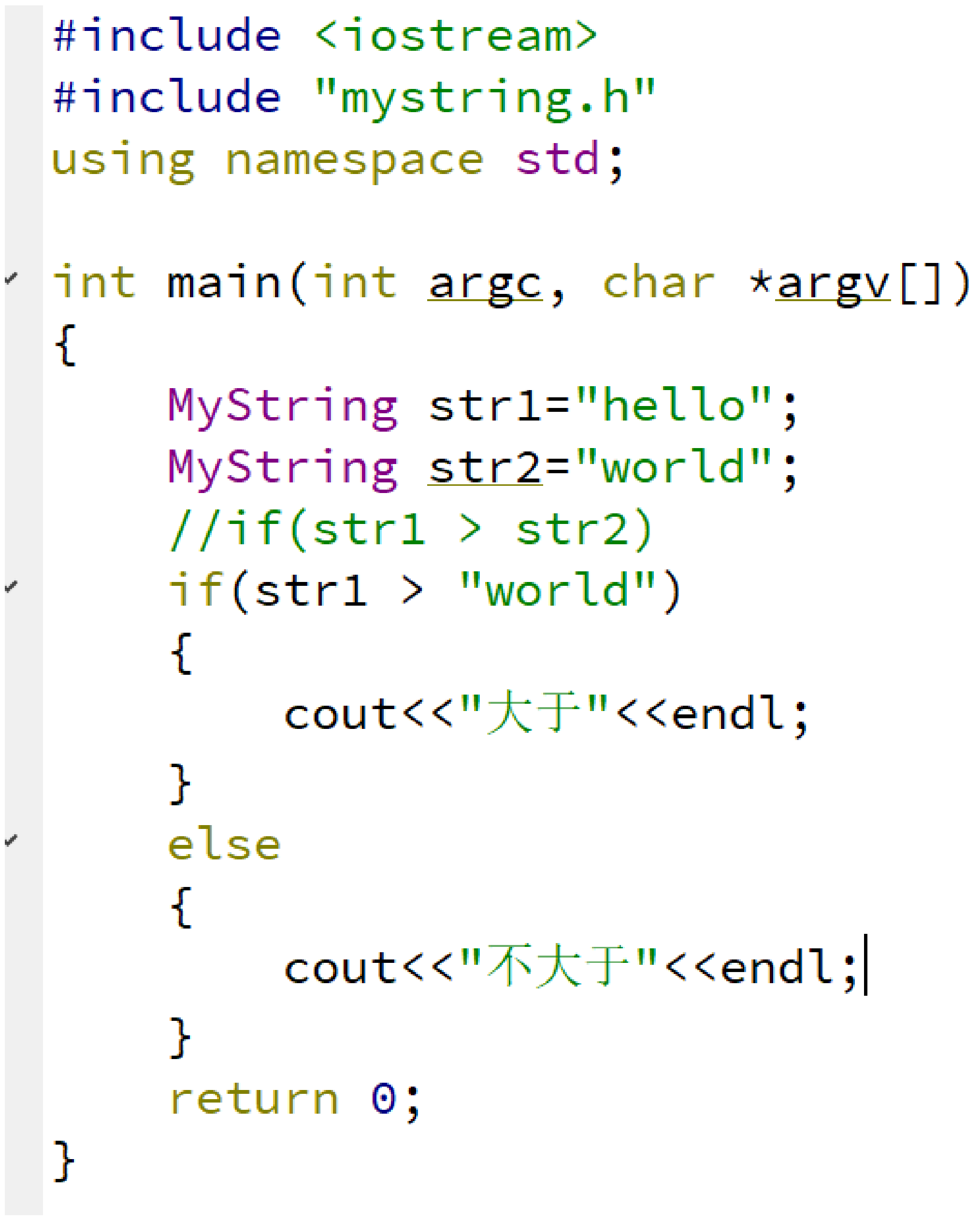
## 5、如果有指针成员 必须重载=赋值运算符（深拷贝）

```cpp
MyString &MyString::operator=(MyString ob)
{
 //str2 = str1;
 if(this->str != NULL)
 {
 delete [] this->str;
 this->str = NULL;
 }

  this->size = ob.size;
  this->str = new char[this->size+1];
  memset(this->str, 0, this->size+1);
  strcpy(this->str, ob.str);

  return *this;
 }

 MyString &MyString::operator=(char *str)
 {
  //str2 = str1;
  if(this->str != NULL)
  {
  delete [] this->str;
  this->str = NULL;
  }

  this->size = strlen(str);
  this->str = new char[this->size+1];
  memset(this->str, 0, this->size+1);
  strcpy(this->str, str);

  return *this;
 }
```

```
int main(int argc, char *argv[])
{
    MyString str1="hello";
    MyString str2;
    str2 = str1;
    str2 = "world";
    cout<<str2 <<endl;
    return 0;
}
```

## 6、重载>运算符

```
bool MyString::operator>(MyString ob)
{
 if(str==NULL || ob.str == NULL)
 {
 exit(-1);
 }
 if(strcmp(this->str, ob.str) > 0)
 {
 return true;
  }
  return false;
}

 bool MyString::operator>(char *str)
 {
  if(this->str==NULL || str == NULL)
  {
  exit(-1);
  }
  if(strcmp(this->str, str) > 0)
  {
  return true;
  }
  return false;
 }

```



```cpp
#include <iostream>
#include "mystring.h"
using namespace std;

int main(int argc, char *argv[])
{
    MyString str1="hello";
    MyString str2="world";
    //if(str1 > str2)
    if(str1 > "world")
    {
        cout<<"大于"<<endl;
    }
    else
    {
        cout<<"不大于"<<endl;
    }
    return 0;
}
```

## 7、完整代码

mystring.h

```cpp
#ifndef MYSTRING_H
#define MYSTRING_H
#include <iostream>
using namespace std;

class MyString
{
 friend ostream& operator<<(ostream &out, MyString ob);
 friend istream& operator>>(istream &in, MyString &ob);
private:
  char *str;
 int size;
public:
 MyString();
 MyString(char *str);
 MyString(const MyString &ob);
 ~MyString();
 int getSize() const;
 //成员函数重载[]
 char& operator[](int pos);
 MyString operator+(MyString ob);
 MyString operator+(char *str);

 MyString& operator=(MyString ob);
 MyString& operator=(char *str);

 bool operator>(MyString ob);
 bool operator>(char *str);

// bool operator<(MyString ob);
// bool operator<(char *str);
// bool operator==(MyString ob);
// bool operator==(char *str);
// bool operator!=(MyString ob);
// bool operator!=(char *str);
};

#endif // MYSTRING_H

```

mystring.cpp

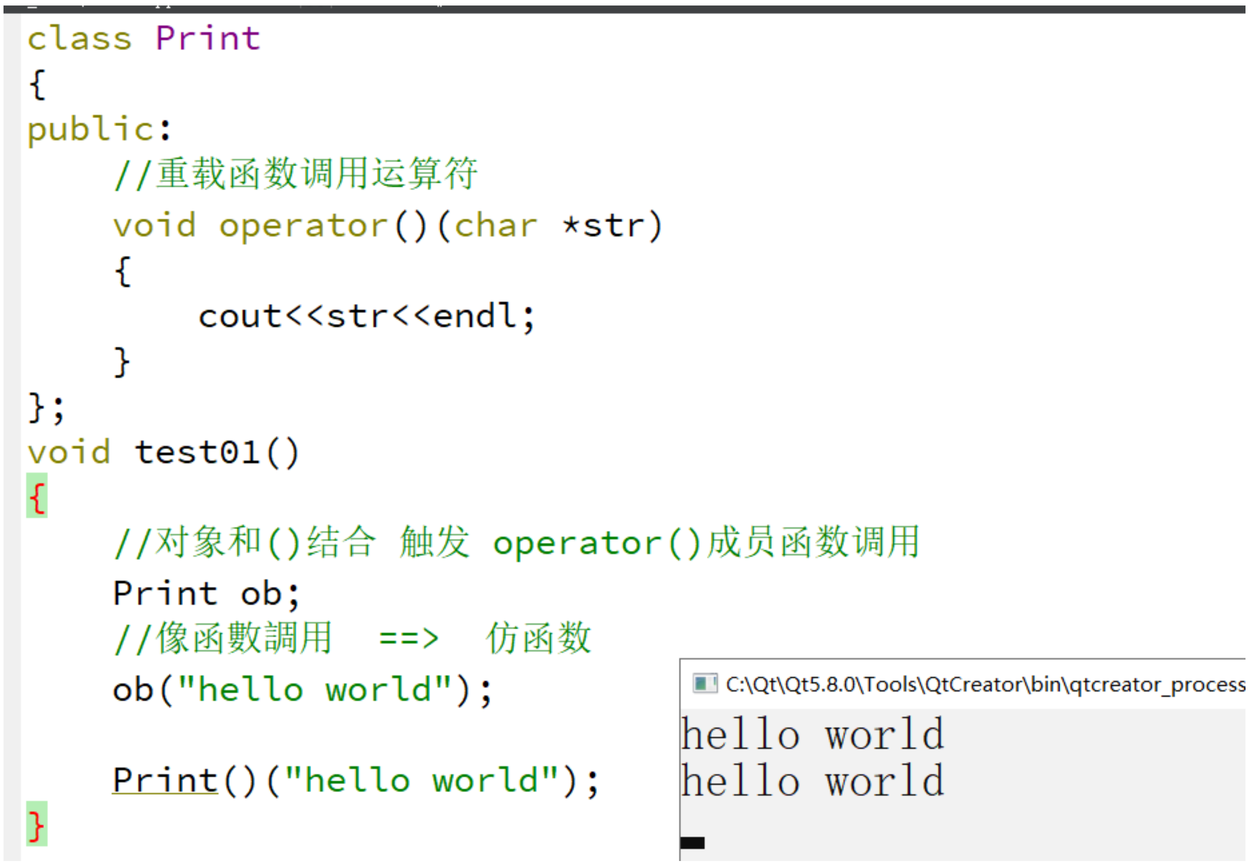
```
#include "mystring.h"
#include <string.h>

int MyString::getSize() const
{
 return size;
}

char& MyString::operator[](int pos)
 {
  if(pos<0 || pos>=size)
```

```cpp
 {
 cout<<"元素位置不合法"<<endl;
 exit(-1);
 }

 return str[pos];
}

MyString MyString::operator+(MyString ob)
{
 MyString tmp;
 tmp.size = size+ob.size;
 tmp.str = new char[tmp.size+1];
 memset(tmp.str, 0, tmp.size+1);

 strcpy(tmp.str, str);
 strcat(tmp.str, ob.str);

 return tmp;
}

MyString MyString::operator+(char *str)
{
 MyString tmp;
 tmp.size = size+strlen(str);
 tmp.str = new char[tmp.size+1];
 memset(tmp.str, 0, tmp.size+1);

 strcpy(tmp.str, this->str);
 strcat(tmp.str, str);

 return tmp;
}

MyString &MyString::operator=(MyString ob)
{
 //str2 = str1;
 if(this->str != NULL)
 {
 delete [] this->str;
 this->str = NULL;
 }

 this->size = ob.size;
 this->str = new char[this->size+1];
 memset(this->str, 0, this->size+1);
 strcpy(this->str, ob.str);

 return *this;
}

MyString &MyString::operator=(char *str)
{
 //str2 = str1;
 if(this->str != NULL)
 {
 delete [] this->str;
 this->str = NULL;
 }

 this->size = strlen(str);
 this->str = new char[this->size+1];
 memset(this->str, 0, this->size+1);
 strcpy(this->str, str);

 return *this;
}

bool MyString::operator>(MyString ob)
{
 if(str==NULL || ob.str == NULL)
 {
 exit(-1);
 }
 if(strcmp(this->str, ob.str) > 0)
 {
 return true;
 }
 return false;
}

bool MyString::operator>(char *str)
```

```
{
 if(this->str==NULL || str == NULL)
 {
 exit(-1);
 }
 if(strcmp(this->str, str) > 0)
  {
  return true;
  }
  return false;
 }

MyString::MyString()
 {
  str=NULL;
  size=0;
 }

MyString::MyString(char *str)
 {
  size = strlen(str);
  this->str = new char[size+1];
  memset(this->str, 0, size+1);

  strcpy(this->str, str);
 }

MyString::MyString(const MyString &ob)
 {
  size = ob.size;
  str = new char[size+1];
  memset(str, 0, size+1);
  strcpy(str, ob.str);
 }

MyString::~MyString()
 {
  if(str != NULL)
  {
  delete [] str;
  str=NULL;
  }
 }

 //全局函数实现 <<重载
 ostream& operator<<(ostream &out, MyString ob)
 {
  out<<ob.str;
  return out;
 }
 //全局函数实现 >>重载
 istream& operator>>(istream &in, MyString &ob)
 {
  char buf[1024]="";
  cin>>buf;

  if(ob.str != NULL)//ob已经有字符串
  {
  delete [] ob.str;
  ob.str = NULL;
  }

  ob.size = strlen(buf);
  ob.str = new char[ob.size+1];
  memset(ob.str, 0,ob.size+1);
  strcpy(ob.str, buf);

  return in;
 }
```

## 知识点3【重载函数调用运算符()】（了解）

重载（）运算符　一般用于 为算法 提供策略。



```cpp
class Print
{
public:
    //重载函数调用运算符
    void operator()(char *str)
    {
        cout<<str<<endl;
    }
};
void test01()
{
    //对象和()结合 触发 operator()成员函数调用
    Print ob;
    //像函數調用  ==>  仿函数
    ob("hello world");

    Print()("hello world");
}
```

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process

```
hello world
hello world
```

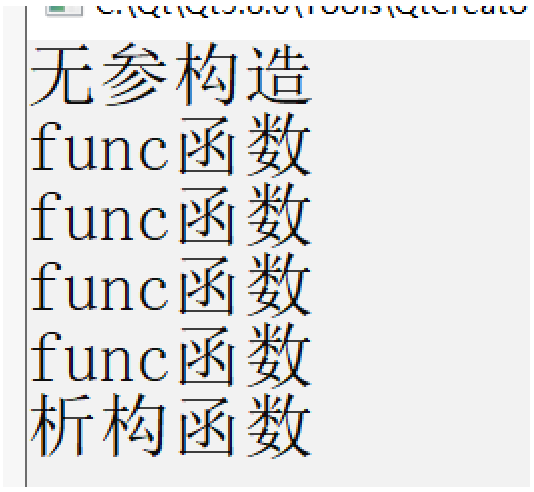
## 知识点4【智能指针】（了解）

智能指针：解决 堆区空间的对象 释放问题

重载* 运算符：

```cpp
class Data
{
public:
 Data()
 {
 cout<<"无参构造"<<endl;
 }
 ~Data()
 {
  cout<<"析构函数"<<endl;
  }
  void func()
  {
  cout<<"func函数"<<endl;
  }
 };
 class SmartPointer
 {
```

```cpp
private:
 Data *p;
public:
 SmartPointer(){}
 SmartPointer(Data *p)
 {
 this->p = p;
 }
 ~SmartPointer()
 {
 delete p;
 }
 Data& operator*()
 {
 return *p;
 }
 Data* operator->()
 {
 return p;
 }
};

void test02()
{
 SmartPointer ob(new Data);
 ob.operator *().func();
 (*ob).func();

 ob.operator ->()->func();
 ob->func();
}
```



```
无参构造
func函数
func函数
func函数
func函数
析构函数
```