

知识点1【运算符重载】（重要）

思路：

- 1、重载<<运算符（全局函数实现）
- 2、重载输入>>运算符
- 3、可以重载的运算符
- 4、重载加法运算符+（全局函数实现）
- 5、重载加法运算符+（成员函数实现 推荐）
- 6、重载==运算符（成员函数实现 推荐）
- 7、重载++运算符

案例1：重载后置++

案例2：重载前置++

知识点2【string类】（了解）

- 1、构造和析构函数
- 2、重载输入输出
- 3、重载中括号运算符
- 4、重载+运算符
- 5、如果有指针成员 必须重载=赋值运算符（深拷贝）
- 6、重载>运算符
- 7、完整代码

知识点3【重载函数调用运算符()】（了解）

知识点4【智能指针】（了解）

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）

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 class Person
5 {
6     friend ostream & operator<<(ostream &out, Person &ob);
7 private:
8     int num;
9     string name;
10    float score;
11 public:
12    Person(){}
13    Person(int num, string name, float score):num(num),name(name),score(score){}
14
15 };
16 ostream & operator<<(ostream &out, Person &ob)
17 {
18     out<<ob.num<<" "<<ob.name<<" "<<ob.score<<endl;
19     return out;
20 }
21
22 int main(int argc, char *argv[])
23 {
24     Person lucy(100,"lucy", 99.8f);
25     Person bob(101,"bob", 88.8f);
26     cout<<lucy<<bob<<lucy<<bob;//operator<<(cout, lucy);
27     return 0;
28 }
29
```

2、重载输入>>运算符

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 class Person
5 {
6     friend ostream & operator<<(ostream &out, Person &ob);
7     friend istream &operator>>(istream &in, Person &ob);
8 private:
9     int num;
10    string name;
11    float score;
12 public:
13    Person(){}
14    Person(int num, string name, float score):num(num),name(name),score(score){}
15
16 };
17 ostream & operator<<(ostream &out, Person &ob)
18 {
19     out<<ob.num<<" "<<ob.name<<" "<<ob.score<<endl;
20     return out;
21 }
22 istream &operator>>(istream &in, Person &ob)
23 {
24     in>>ob.num>>ob.name>>ob.score;
25     return in;
26 }
27
28 int main(int argc, char *argv[])
29 {
30     Person lucy;
31     Person bob;
32
33     cin>>lucy>>bob;
34
35     cout<<lucy<<bob<<endl;
36     return 0;
37 }
38
```

```
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[]			

不能重载的算符

. :: .* ?: sizeof

尽量别重载
无法完成 它们 短路特性

“ ←

4、重载加法运算符+（全局函数实现）

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 class Person
5 {
6     friend ostream & operator<<(ostream &out, Person ob);
7     friend istream & operator>>(istream &in, Person &ob);
8     friend Person operator+(Person ob1, Person ob2);
9 private:
10     int num;
11     string name;
12     float score;
13 public:
14     Person(){}
15     Person(int num, string name, float score):num(num),name(name),score(score){}
```

```

16
17 };
18 ostream & operator<<(ostream &out, Person ob)
19 {
20     out<<ob.num<<" "<<ob.name<<" "<<ob.score<<endl;
21     return out;
22 }
23 istream &operator>>(istream &in, Person &ob)
24 {
25     in>>ob.num>>ob.name>>ob.score;
26     return in;
27 }
28 Person operator+(Person ob1, Person ob2)
29 {
30     Person tmp;
31     tmp.num = ob1.num+ob2.num;
32     tmp.name = ob1.name+ob2.name;
33     tmp.score = ob1.score+ob2.score;
34
35     return tmp;
36 }
37
38 int main(int argc, char *argv[])
39 {
40     Person lucy(100,"lucy", 88.8f);
41     Person bob(101,"bob", 99.9f);
42     Person tom(102,"tom", 77.7f);
43
44     cout<<lucy+bob+tom<<endl;
45     return 0;
46 }

```

303 lucybobtom 266.4

5、重载加法运算符+ (成员函数实现 推荐)

```

1 #include <iostream>
2 #include <string>

```

```

3 using namespace std;
4 class Person
5 {
6     friend ostream & operator<<(ostream &out, Person ob);
7     friend istream &operator>>(istream &in, Person &ob);
8 private:
9     int num;
10    string name;
11    float score;
12 public:
13     Person(){}
14     Person(int num, string name, float score):num(num),name(name),score(sco
re){}
15     //成员函数重载+
16     Person operator+(Person ob)
17     {
18         Person tmp;
19         tmp.num = this->num + ob.num;
20         tmp.name = this->name + ob.name;
21         tmp.score = this->score + ob.score;
22         return tmp;
23     }
24
25 };
26 ostream & operator<<(ostream &out, Person ob)
27 {
28     out<<ob.num<<" "<<ob.name<<" "<<ob.score<<endl;
29     return out;
30 }
31 istream &operator>>(istream &in, Person &ob)
32 {
33     in>>ob.num>>ob.name>>ob.score;
34     return in;
35 }
36
37 int main(int argc, char *argv[])
38 {
39     Person lucy(100,"lucy", 88.8f);
40     Person bob(101,"bob", 99.9f);
41     Person tom(102,"tom", 77.7f);

```

```

42 //lucy+bob;//lucy.operator+(bob)
43 cout<<lucy+bob+tom<<endl;
44 return 0;
45 }

```

303 lucybobtom 266.4

6、重载==运算符 (成员函数实现 推荐)

```

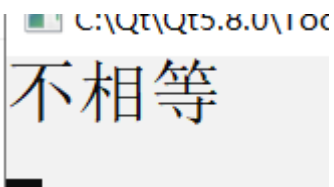
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 class Person
5 {
6     friend ostream & operator<<(ostream &out, Person ob);
7     friend istream & operator>>(istream &in, Person &ob);
8 private:
9     int num;
10    string name;
11    float score;
12 public:
13    Person(){}
14    Person(int num, string name, float score):num(num),name(name),score(score){}
15    //成员函数重载+
16    Person operator+(Person ob)
17    {
18        Person tmp;
19        tmp.num = this->num + ob.num;
20        tmp.name = this->name + ob.name;
21        tmp.score = this->score + ob.score;
22        return tmp;
23    }
24    //成员函数重载==
25    bool operator==(Person &ob)
26    {
27        if(num == ob.num && name==ob.name && score==ob.score)
28            return true;
29        return false;

```

```

30  }
31
32  };
33  ostream & operator<<(ostream &out, Person ob)
34  {
35      out<<ob.num<<" "<<ob.name<<" "<<ob.score<<endl;
36      return out;
37  }
38  istream &operator>>(istream &in, Person &ob)
39  {
40      in>>ob.num>>ob.name>>ob.score;
41      return in;
42  }
43
44  int main(int argc, char *argv[])
45  {
46      Person lucy(100,"lucy", 88.8f);
47      Person bob(101,"bob", 99.9f);
48      if(lucy == bob)
49      {
50          cout<<"相等"<<endl;
51      }
52      else
53      {
54          cout<<"不相等"<<endl;
55      }
56      return 0;
57  }

```



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++

```

1 类名称 operator++(int)
2  {

```



```
3 //先保存 旧的值
4 //自增++
5 return old;//返回旧值
6 }
```

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 class Person
5 {
6     friend ostream & operator<<(ostream &out, Person ob);
7     friend istream & operator>>(istream &in, Person &ob);
8 private:
9     int num;
10    string name;
11    float score;
12 public:
13    Person(){}
14    Person(int num, string name, float score):num(num),name(name),score(score){}
15    //成员函数重载+
16    Person operator+(Person ob)
17    {
18        Person tmp;
19        tmp.num = this->num + ob.num;
20        tmp.name = this->name + ob.name;
21        tmp.score = this->score + ob.score;
22        return tmp;
23    }
24    //成员函数重载==
25    bool operator==(Person &ob)
26    {
27        if(num == ob.num && name==ob.name && score==ob.score)
28            return true;
29        return false;
30    }
31    //重载后置++ operator++(a,int)
32    Person operator++(int)
33    {
34        //先保存 旧的值
```

```

35  Person old = *this; /*this == lucy
36
37  //lucy++ ==> lucy = lucy+1
38  this->num = this->num +1;
39  this->name = this->name+this->name; // (自定义操作)
40  this->score = this->score+1;
41
42  return old; //返回旧值
43  }
44
45
46 };
47 ostream & operator<<(ostream &out, Person ob)
48 {
49  out<<ob.num<<" "<<ob.name<<" "<<ob.score<<endl;
50  return out;
51 }
52 istream &operator>>(istream &in, Person &ob)
53 {
54  in>>ob.num>>ob.name>>ob.score;
55  return in;
56 }
57
58 int main(int argc, char *argv[])
59 {
60  Person lucy(100, "lucy", 88.8f);
61  Person bob;
62  //先使用 后++
63  bob = lucy++; //
64  cout<<bob<<endl;
65  cout<<lucy<<endl;
66  return 0;
67 }
68

```

```
100 lucy 88.8
```

```
101 lucylucy 89.8
```

案例2：重载前置++

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  class Person
5  {
6  friend ostream & operator<<(ostream &out, Person ob);
7  friend istream &operator>>(istream &in, Person &ob);
8  private:
9  int num;
10 string name;
11 float score;
12 public:
13 Person(){}
14 Person(int num, string name, float score):num(num),name(name),score(score){}
15 //成员函数重载+
16 Person operator+(Person ob)
17 {
18 Person tmp;
19 tmp.num = this->num + ob.num;
20 tmp.name = this->name + ob.name;
21 tmp.score = this->score + ob.score;
22 return tmp;
23 }
24 //成员函数重载==
25 bool operator==(Person &ob)
26 {
27 if(num == ob.num && name==ob.name && score==ob.score)
28 return true;
29 return false;
30 }
31 //重载后置++ operator++(a,int)
```

```

32  Person operator++(int)
33  {
34      //先保存 旧的值
35      Person old = *this; // *this == lucy
36
37      //lucy++ ==> lucy = lucy+1
38      this->num = this->num +1;
39      this->name = this->name+this->name; // (自定义操作)
40      this->score = this->score+1;
41
42      return old; //返回旧值
43  }
44      //重载前置++ operator++(a)
45      Person operator++()
46      {
47          //先++
48          this->num = this->num +1;
49          this->name = this->name+this->name; // (自定义操作)
50          this->score = this->score+1;
51
52          //后使用
53          return *this;
54      }
55
56  };
57  ostream & operator<<(ostream &out, Person ob)
58  {
59      out<<ob.num<<" "<<ob.name<<" "<<ob.score<<endl;
60      return out;
61  }
62  istream &operator>>(istream &in, Person &ob)
63  {
64      in>>ob.num>>ob.name>>ob.score;
65      return in;
66  }
67
68  int main(int argc, char *argv[])
69  {
70      Person lucy(100, "lucy", 88.8f);
71      Person bob;

```

```

72 //先++ 后使用
73 bob = ++lucy;//
74 cout<<bob<<endl;
75 cout<<lucy<<endl;
76 return 0;
77 }

```

```

101 lucylucy 89.8

```

```

101 lucylucy 89.8

```

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

知识点2【string类】（了解）

1、构造和析构函数

```

1 MyString::MyString()
2 {
3     str=NULL;
4     size=0;
5 }
6
7 MyString::MyString(char *str)
8 {
9     size = strlen(str);
10    this->str = new char[size+1];
11    memset(this->str, 0, size+1);
12

```

```

13  strcpy(this->str, str);
14  }
15
16  MyString::MyString(const MyString &ob)
17  {
18      size = ob.size;
19      str = new char[size+1];
20      memset(str, 0, size+1);
21      strcpy(str, ob.str);
22  }
23
24  MyString::~MyString()
25  {
26      if(str != NULL)
27      {
28          delete [] str;
29          str=NULL;
30      }
31  }

```

2、重载输入输出

```

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

```

```

1  //全局函数实现 <<重载
2  ostream& operator<<(ostream &out, MyString ob)
3  {
4      out<<ob.str;
5      return out;
6  }
7  //全局函数实现 >>重载
8  istream& operator>>(istream &in, MyString &ob)
9  {
10     char buf[1024]="";
11     cin>>buf;
12
13     if(ob.str != NULL)//ob已经有字符串
14     {
15         delete [] ob.str;
16         ob.str = NULL;

```

```

17  }
18
19  ob.size = strlen(buf);
20  ob.str = new char[ob.size+1];
21  memset(ob.str, 0, ob.size+1);
22  strcpy(ob.str, buf);
23
24  return in;
25  }

```

3、重载中括号运算符

```

1  char& MyString::operator[](int pos)
2  {
3      if(pos<0 || pos>=size)
4      {
5          cout<<"元素位置不合法"<<endl;
6          exit(-1);
7      }
8
9      return str[pos];
10 }

```

```

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

```

1  MyString MyString::operator+(MyString ob)
2  {
3      MyString tmp;
4      tmp.size = size+ob.size;
5      tmp.str = new char[tmp.size+1];
6      memset(tmp.str, 0, tmp.size+1);

```

```

7
8  strcpy(tmp.str, str);
9  strcat(tmp.str, ob.str);
10
11  return tmp;
12 }
13
14 MyString MyString::operator+(char *str)
15 {
16  MyString tmp;
17  tmp.size = size+strlen(str);
18  tmp.str = new char[tmp.size+1];
19  memset(tmp.str, 0, tmp.size+1);
20
21  strcpy(tmp.str, this->str);
22  strcat(tmp.str, str);
23
24  return tmp;
25 }
26

```

```

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;
}

```

```

$ ./a.out
helloworld
helloworld
helloworldxixixi

```


5、如果有指针成员 必须重载=赋值运算符（深拷贝）

```
1  MyString &MyString::operator=(MyString ob)
2  {
3      //str2 = str1;
4      if(this->str != NULL)
5      {
6          delete [] this->str;
7          this->str = NULL;
8      }
9
10     this->size = ob.size;
11     this->str = new char[this->size+1];
12     memset(this->str, 0, this->size+1);
13     strcpy(this->str, ob.str);
14
15     return *this;
16 }
17
18 MyString &MyString::operator=(char *str)
19 {
20     //str2 = str1;
21     if(this->str != NULL)
22     {
23         delete [] this->str;
24         this->str = NULL;
25     }
26
27     this->size = strlen(str);
28     this->str = new char[this->size+1];
29     memset(this->str, 0, this->size+1);
30     strcpy(this->str, str);
31
32     return *this;
33 }
```

```

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

```

1  bool MyString::operator>(MyString ob)
2  {
3      if(str==NULL || ob.str == NULL)
4      {
5          exit(-1);
6      }
7      if(strcmp(this->str, ob.str) > 0)
8      {
9          return true;
10     }
11     return false;
12 }
13
14 bool MyString::operator>(char *str)
15 {
16     if(this->str==NULL || str == NULL)
17     {
18         exit(-1);
19     }
20     if(strcmp(this->str, str) > 0)
21     {
22         return true;
23     }
24     return false;
25 }
26

```

```

#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

```

1  #ifndef MYSTRING_H
2  #define MYSTRING_H
3  #include <iostream>
4  using namespace std;
5
6  class MyString
7  {
8  friend ostream& operator<<(ostream &out, MyString ob);
9  friend istream& operator>>(istream &in, MyString &ob);
10 private:
11     char *str;

```

```

12  int size;
13  public:
14  MyString();
15  MyString(char *str);
16  MyString(const MyString &ob);
17  ~MyString();
18  int getSize() const;
19  //成员函数重载[]
20  char& operator[](int pos);
21  MyString operator+(MyString ob);
22  MyString operator+(char *str);
23
24  MyString& operator=(MyString ob);
25  MyString& operator=(char *str);
26
27  bool operator>(MyString ob);
28  bool operator>(char *str);
29
30  // bool operator<(MyString ob);
31  // bool operator<(char *str);
32  // bool operator==(MyString ob);
33  // bool operator==(char *str);
34  // bool operator!=(MyString ob);
35  // bool operator!=(char *str);
36  };
37
38 #endif // MYSTRING_H
39

```

mystring.cpp

```

1  #include "mystring.h"
2  #include <string.h>
3
4  int MyString::getSize() const
5  {
6  return size;
7  }
8
9  char& MyString::operator[](int pos)
10 {
11  if(pos<0 || pos>=size)

```

```

12  {
13  cout<<"元素位置不合法"<<endl;
14  exit(-1);
15  }
16
17  return str[pos];
18  }
19
20  MyString MyString::operator+(MyString ob)
21  {
22  MyString tmp;
23  tmp.size = size+ob.size;
24  tmp.str = new char[tmp.size+1];
25  memset(tmp.str, 0, tmp.size+1);
26
27  strcpy(tmp.str, str);
28  strcat(tmp.str, ob.str);
29
30  return tmp;
31  }
32
33  MyString MyString::operator+(char *str)
34  {
35  MyString tmp;
36  tmp.size = size+strlen(str);
37  tmp.str = new char[tmp.size+1];
38  memset(tmp.str, 0, tmp.size+1);
39
40  strcpy(tmp.str, this->str);
41  strcat(tmp.str, str);
42
43  return tmp;
44  }
45
46  MyString &MyString::operator=(MyString ob)
47  {
48  //str2 = str1;
49  if(this->str != NULL)
50  {
51  delete [] this->str;
52  this->str = NULL;

```

```

53     }
54
55     this->size = ob.size;
56     this->str = new char[this->size+1];
57     memset(this->str, 0, this->size+1);
58     strcpy(this->str, ob.str);
59
60     return *this;
61 }
62
63 MyString &MyString::operator=(char *str)
64 {
65     //str2 = str1;
66     if(this->str != NULL)
67     {
68         delete [] this->str;
69         this->str = NULL;
70     }
71
72     this->size = strlen(str);
73     this->str = new char[this->size+1];
74     memset(this->str, 0, this->size+1);
75     strcpy(this->str, str);
76
77     return *this;
78 }
79
80 bool MyString::operator>(MyString ob)
81 {
82     if(str==NULL || ob.str == NULL)
83     {
84         exit(-1);
85     }
86     if(strcmp(this->str, ob.str) > 0)
87     {
88         return true;
89     }
90     return false;
91 }
92
93 bool MyString::operator>(char *str)

```

```

94 {
95     if(this->str==NULL || str == NULL)
96     {
97         exit(-1);
98     }
99     if(strcmp(this->str, str) > 0)
100     {
101         return true;
102     }
103     return false;
104 }
105
106 MyString::MyString()
107 {
108     str=NULL;
109     size=0;
110 }
111
112 MyString::MyString(char *str)
113 {
114     size = strlen(str);
115     this->str = new char[size+1];
116     memset(this->str, 0, size+1);
117
118     strcpy(this->str, str);
119 }
120
121 MyString::MyString(const MyString &ob)
122 {
123     size = ob.size;
124     str = new char[size+1];
125     memset(str, 0, size+1);
126     strcpy(str, ob.str);
127 }
128
129 MyString::~MyString()
130 {
131     if(str != NULL)
132     {
133         delete [] str;

```

```

134  str=NULL;
135  }
136  }
137
138  //全局函数实现 <<重载
139  ostream& operator<<(ostream &out, MyString ob)
140  {
141      out<<ob.str;
142      return out;
143  }
144  //全局函数实现 >>重载
145  istream& operator>>(istream &in, MyString &ob)
146  {
147      char buf[1024]="";
148      cin>>buf;
149
150      if(ob.str != NULL)//ob已经有字符串
151      {
152          delete [] ob.str;
153          ob.str = NULL;
154      }
155
156      ob.size = strlen(buf);
157      ob.str = new char[ob.size+1];
158      memset(ob.str, 0, ob.size+1);
159      strcpy(ob.str, buf);
160
161      return in;
162  }

```

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。


```

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;
    //像函数调用 ==> 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：

```

1  class Data
2  {
3  public:
4      Data()
5      {
6          cout<<"无参构造"<<endl;
7      }
8      ~Data()
9      {
10         cout<<"析构函数"<<endl;
11     }
12     void func()
13     {
14         cout<<"func函数"<<endl;
15     }
16 };
17 class SmartPointer
18 {

```

```
19 private:
20     Data *p;
21 public:
22     SmartPointer(){}
23     SmartPointer(Data *p)
24     {
25         this->p = p;
26     }
27     ~SmartPointer()
28     {
29         delete p;
30     }
31     Data& operator*()
32     {
33         return *p;
34     }
35     Data* operator->()
36     {
37         return p;
38     }
39 };
40
41 void test02()
42 {
43     SmartPointer ob(new Data);
44     ob.operator *().func();
45     (*ob).func();
46
47     ob.operator ->()->func();
48     ob->func();
49 }
```

