

知识点1【const修饰成员函数】

- 2、const修饰对象 叫常对象

知识点2【友元】

- 1、普通全局函数 作为 类的友元
- 2、类的某个成员函数 作为 另一个类的友元

终极代码：

- 3、一个类整体 作为 另一个类的友元

知识点3【封装电视机 和遥控器的类】

- 1、封装电视机的类
- 2、设置遥控器的类2-1

知识点4【强化类的封装】

知识点5【运算符重载】2-1

- 1、重载运算符的概述
- 2、运算符<<的重载
- 2、重载+运算符：全局函数作为友元 完成运算符重载+
- 3、重载+运算符：成员函数 完成运算符重载+

知识点1【const修饰成员函数】

用const修饰的成员函数时，const修饰this指针指向的内存区域，成员函数体内不可以修改本类中的任何普通成员变量，当成员变量类型符前用mutable修饰时例外。

```
1 int myFun(void) const //const修饰的是成员函数
2 {}//函数内部不能修改 普通成员变量 mutable修饰时例外
```

```
1 class Data
2 {
```

```

3 private:
4     int data;
5
6     mutable int num;
7 public:
8     //遍历 成员的函数 不会去修改成员的值
9     //如果函数不会更改成员数据 就让编译器知道 这是一个const函数
10    void myPrintData(void) const
11    {
12        //data =10000;//err const修饰函数 函数不能操作普通成员变量
13        cout<<"this->data"<<endl;
14        //cout<<data<<endl;
15
16        //mutable修饰的成员变量 可以修改
17        num = 200;
18    }
19
20    Data()
21    {
22        cout<<"无参构造"<<endl;
23    }
24    Data(int data)
25    {
26        this->data =data;
27        cout<<"有参构造"<<endl;
28    }
29    Data(const Data &ob)
30    {
31        this->data = ob.data;
32        cout<<"拷贝构造"<<endl;
33    }
34    ~Data()
35    {
36        cout<<"析构函数"<<endl;
37    }
38 };
39 void test02()
40 {
41     Data ob1(100);
42     ob1.myPrintData();

```

2、const修饰对象 叫常对象

const int num = 10; //系统不会给num开辟空间 num被放入符号表中 如果后期对num
这时系统才会给num开辟空间

```

1  class Data
2  {
3  private:
4      int data;
5
6      mutable int num;
7  public:
8      //遍历 成员的函数 不会去修改成员的值
9      //如果函数不会更改成员数据 就让编译器知道 这是一个const函数
10     void myPrintData(void) const
11     {
12         //data =10000; //err const修饰函数 函数不能操作普通成员变量
13         cout<<"this->data"<<endl;
14         //cout<<data<<endl;
15
16         //mutable修饰的成员变量 可以修改
17         num = 200;
18     }
19
20     //编译器认为 普通成员函数 存在修改成员变量 可能
21     void setData(int data) const
22     {
23         //this->data = data;
24         return;
25     }
26     Data()
27     {
28         cout<<"无参构造"<<endl;
29     }
30     Data(int data)
31     {
32         this->data =data;
33         cout<<"有参构造"<<endl;

```

```
34  }
35  Data(const Data &ob)
36  {
37      this->data = ob.data;
38      cout<<"拷贝构造"<<endl;
39  }
40  ~Data()
41  {
42      cout<<"析构函数"<<endl;
43  }
44  };
45  void test03()
46  {
47      //常对象
48      const Data ob1(200);
49
50      //常对象 只能调用const修饰的函数 遍历成员数据
51      ob1.setData(20000);
52
53      ob1.myPrintData();
54  }
```

运行结果：

有参构造 200 析构函数

知识点2【友元】

c++允许 友元 访问 私有数据。

友元的语法：

friend关键字只出现在声明处 其他类、类成员函数、全局函数都可声明为友元 友元函数不是类的成员，不带this指针 友元函数可访问对象任意成员属性，包括私有属性。

1、普通全局函数 作为 类的友元

```
1 //房间类
2 class Room
3 {
4     //将goodGayVisit作为类的友元函数
5     //goodGayVisit 访问 类中所有数据 但是 它不是类的成员
6     friend void goodGayVisit(Room &room);
7 private:
8     string bedRoom;//卧室
9 public:
10    string sittingRoom;//客厅
11 public:
```

```

12  Room()
13  {
14      this->bedRoom = "卧室";
15      this->sittingRoom="客厅";
16  }
17  };
18
19  // 普通全局函数 作为 类的友元
20  //好基友 访问 我的房间
21  void goodGayVisit(Room &room)
22  {
23      cout<<"好基友访问了你的"<<room.sittingRoom<<endl;
24      cout<<"好基友访问了你的"<<room.bedRoom<<endl;//ok
25  }
26  void test01()
27  {
28      Room myRoom;
29      goodGayVisit(myRoom);
30  }

```

运行结果：

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

好基友访问了你的客厅
好基友访问了你的卧室

2、类的某个成员函数 作为 另一个类的友元

问题1：

```

3 using namespace std;
4 class GoodGay
5 {
6 public:
7     void visit1(Room &room)
8     {
9         cout<<"好基友visit1访问了你的"<<room.sittingRoom<<endl;
10        cout<<"好基友visit1访问了你的"<<room.bedRoom<<endl;|
11    }
12 };
13 class Room
14 {
15 private:
16     string bedRoom;//卧室
17 public:
18     string sittingRoom;//客厅
19 public:
20     Room()
21     {
22         this->bedRoom = "卧室";

```

问题1: 不是别 Room类名称
解决办法: 将类名称 提前声明 (向前声明)

问题2:

```

3 using namespace std;
4 class Room;//Room向前声明
5 class GoodGay
6 {
7 public:
8     void visit1(Room &room)
9     {
10        cout<<"好基友visit1访问了你的"<<room.sittingRoom<<endl;
11        cout<<"好基友visit1访问了你的"<<room.bedRoom<<endl;
12    }
13 };
14 class Room
15 {
16 private:
17     string bedRoom;//卧室
18 public:
19     string sittingRoom;//客厅
20 public:
21     Room()

```

问题1: 不识别 sittingRoom和bedRoom
注意: 向前声明 class Room;只能说明Room这个类存在
但不能描述 Room有哪些成员

通过

不知道有哪些成员

解决办法: visit1函数定义 放在所有类的下方 别再类中定义

问题3: visit1成员函数内 不能访问 Room的私有数据

```

21     string sittingRoom; // 客厅
18 public:
19     Room()
20     {
21         this->bedRoom = "卧室";
22         this->sittingRoom = "客厅";
23     }
24 };
25
26 void GoodGay::visit1(Room &room)
27 {
28     cout<<"好基友visit1访问了你的"<<room.sittingRoom<<endl;
29     cout<<"好基友visit1访问了你的"<<room.bedRoom<<endl;
30 }
31
32 int main(int argc, char *argv[])
33 {
34

```

放在所有类的下方

公有数据

私有数据

终极代码：

```

1 #include <iostream>
2
3 using namespace std;
4 class Room; //Room向前声明
5 class GoodGay
6 {
7 public:
8     void visit1(Room &room); //此处的Room 被上方 class Room
9     void visit2(Room &room);
10 };
11
12
13 class Room
14 {
15     //如果想方 visit2作为Room类的友元 那么Visit2就可以访问 Room的私有数据
16     //一定要记得 加类作用域
17     friend void GoodGay::visit2(Room &room);
18 private:
19     string bedRoom; //卧室
20 public:
21     string sittingRoom; //客厅
22 public:
23     Room()
24     {
25         this->bedRoom = "卧室";
26         this->sittingRoom = "客厅";

```

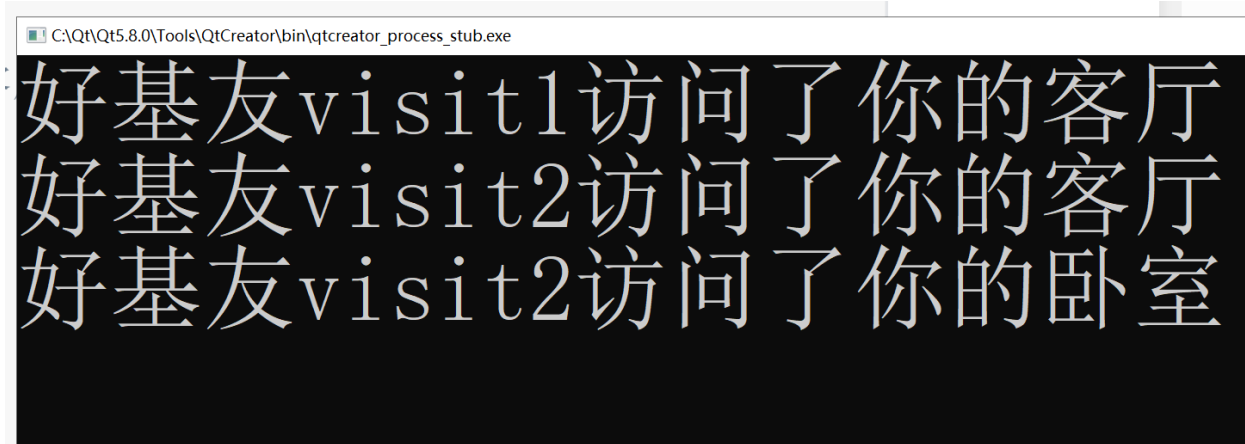


```

27  }
28  };
29
30  void GoodGay::visit1(Room &room)
31  {
32      cout<<"好基友visit1访问了你的"<<room.sittingRoom<<endl;
33      //cout<<"好基友visit1访问了你的"<<room.bedRoom<<endl;//不能访问 Room私有数据
34  }
35
36  void GoodGay::visit2(Room &room)
37  {
38      cout<<"好基友visit2访问了你的"<<room.sittingRoom<<endl;
39      cout<<"好基友visit2访问了你的"<<room.bedRoom<<endl;
40  }
41
42  int main(int argc, char *argv[])
43  {
44      Room myRoom;
45      GoodGay goodGay;
46
47      goodGay.visit1(myRoom);//只能访问客厅
48      goodGay.visit2(myRoom);//客厅 卧室 都可以访问
49
50      return 0;
51  }
52

```

运行结果：



```

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
好基友visit1访问了你的客厅
好基友visit2访问了你的客厅
好基友visit2访问了你的卧室

```

3、一个类整体 作为 另一个类的友元

一个类的所有成员函数 访问 另一个类的私有数据

```
1 #include <iostream>
2
3 using namespace std;
4 class Room; //Room向前声明
5 class GoodGay
6 {
7 public:
8     void visit1(Room &room); //此处的Room 被上方 class Room
9     void visit2(Room &room);
10 };
11
12
13 class Room
14 {
15     //将GoodGay作为Room的友元
16     //GoodGay 所有成员函数 都可以访问 Room私有数据
17     friend class GoodGay;
18 private:
19     string bedRoom; //卧室
20 public:
21     string sittingRoom; //客厅
22 public:
23     Room()
24     {
25         this->bedRoom = "卧室";
26         this->sittingRoom = "客厅";
27     }
28 };
29
30 void GoodGay::visit1(Room &room)
31 {
32     cout<<"好基友visit1访问了你的"<<room.sittingRoom<<endl;
33     cout<<"好基友visit1访问了你的"<<room.bedRoom<<endl;
34 }
35
36 void GoodGay::visit2(Room &room)
37 {
38     cout<<"好基友visit2访问了你的"<<room.sittingRoom<<endl;
39     cout<<"好基友visit2访问了你的"<<room.bedRoom<<endl;
40 }
```

```

41
42 int main(int argc, char *argv[])
43 {
44     Room myRoom;
45     GoodGay goodGay;
46
47     goodGay.visit1(myRoom);
48     goodGay.visit2(myRoom);
49
50     return 0;
51 }
52

```

运行结果：

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

```

好基友visit1访问了你的客厅
好基友visit1访问了你的卧室
好基友visit2访问了你的客厅
好基友visit2访问了你的卧室

```

知识点3 【封装电视机 和遥控器的类】

1、封装电视机的类

```

1  class TV
2  {
3      enum{ On,Off }; //电视状态
4      enum{ minVol,maxVol = 100 }; //音量从0到100
5      enum{ minChannel = 1,maxChannel = 255 }; //频道从1到255
6  private:
7      int mState; //电视状态，开机，还是关机
8      int mVolume; //电视机音量
9      int mChannel; //电视频道
10 public:
11     TV()
12     {

```

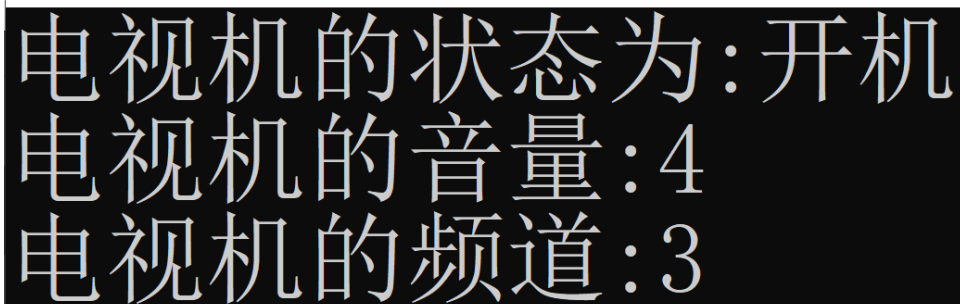
```
13  this->mState = Off;//默认关机
14  this->mVolume = minVol;
15  this->mChannel = minChannel;
16  }
17  void onOrOff(void)
18  {
19  this->mState = (this->mState == On ? Off:On);
20  }
21
22  //加大音量
23  void volumeUp(void)
24  {
25  if(this->mVolume >= maxVol)
26  return;
27
28  this->mVolume++;
29  }
30  //减小音量
31  void volumeDown(void)
32  {
33  if(this->mVolume <= minVol)
34  return;
35  this->mVolume--;
36  }
37
38  //增加频道
39  void channelUp(void)
40  {
41  if(this->mChannel >= maxChannel)
42  return;
43  this->mChannel++;
44  }
45  //减小频道
46  void channelDown(void)
47  {
48  if(this->mChannel <= minChannel)
49  return;
50  this->mChannel--;
51  }
52
```

```

53 //显示电视机的状态
54 void showTVState(void)
55 {
56     cout<<"电视机的状态为:"<< (this->mState==On ? "开机":"关机") <<endl;
57     cout<<"电视机的音量:"<<this->mVolume<<endl;
58     cout<<"电视机的频道:"<<this->mChannel<<endl;
59 }
60 };
61 void test01()
62 {
63     TV tv;
64     tv.onOrOff();//开机
65
66     tv.volumeUp();//调四次音量
67     tv.volumeUp();
68     tv.volumeUp();
69     tv.volumeUp();
70
71     tv.channelUp();//调三次频道
72     tv.channelUp();
73
74     tv.showTVState();
75 }

```

运行结果：



电视机的状态为:开机
电视机的音量:4
电视机的频道:3

2、设置遥控器的类2-1

```

1 class TV
2 {
3     friend class Remote;

```

```

4 //默认为私有
5 enum{ On,Off }; //电视状态
6 enum{ minVol,maxVol = 100 }; //音量从0到100
7 enum{ minChannel = 1,maxChannel = 255 }; //频道从1到255
8 private:
9 int mState; //电视状态, 开机, 还是关机
10 int mVolume; //电视机音量
11 int mChannel; //电视频道
12 public:
13 TV()
14 {
15     this->mState = Off;//默认关机
16     this->mVolume = minVol;
17     this->mChannel = minChannel;
18 }
19 void onOrOff(void)
20 {
21     this->mState = (this->mState == On ? Off:On);
22 }
23
24 //加大音量
25 void volumeUp(void)
26 {
27     if(this->mVolume >= maxVol)
28         return;
29
30     this->mVolume++;
31 }
32 //减小音量
33 void volumeDown(void)
34 {
35     if(this->mVolume <= minVol)
36         return;
37     this->mVolume--;
38 }
39
40 //增加频道
41 void channelUp(void)
42 {
43     if(this->mChannel >= maxChannel)

```

```
44     return;
45     this->mChannel++;
46 }
47 //减小频道
48 void channelDown(void)
49 {
50     if(this->mChannel <= minChannel)
51         return;
52     this->mChannel--;
53 }
54
55 //显示电视机的状态
56 void showTVState(void)
57 {
58     cout<<"电视机的状态为:"<< (this->mState==On ? "开机":"关机") <<endl;
59     cout<<"电视机的音量:"<<this->mVolume<<endl;
60     cout<<"电视机的频道:"<<this->mChannel<<endl;
61 }
62 };
63
64 //遥控器类
65 class Remote
66 {
67 private:
68     TV *pTv;
69 public:
70     Remote(TV *pTv)
71     {
72         this->pTv = pTv;
73     }
74     //音量的加减
75     void volumeUp(void)
76     {
77         //调节的电视机的音量
78         this->pTv->volumeUp();
79     }
80     void volumeDown(void)
81     {
82         this->pTv->volumeDown();
83     }
```

```
84
85 //频道的加减
86 void channelUp(void)
87 {
88     this->pTv->channelUp();
89 }
90 void channelDown(void)
91 {
92     this->pTv->channelDown();
93 }
94
95 //电视开关
96 void onOrOff(void)
97 {
98     this->pTv->onOrOff();
99 }
100
101 //遥控器设置频道设置
102 void setChannel(int num)
103 {
104     //判断 频道 是否有效
105     if(num >= TV::minChannel && num<= TV::maxChannel )
106     {
107         this->pTv->mChannel = num;
108     }
109 }
110
111 void showTVState(void)
112 {
113     this->pTv->showTVState();
114 }
115
116 };
117
118 void test02()
119 {
120     TV tv;
121     Remote remote(&tv);
122
123     remote.onOrOff();
124     remote.volumeUp();
```



```
125 remote.volumeUp();
126 remote.volumeUp();
127 remote.volumeUp();
128
129 remote.channelUp();
130 remote.channelUp();
131
132 remote.showTVState();
133
134 remote.setChannel(75);
135 remote.showTVState();
136 }
```

运行结果：

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

电视机的状态为:开机
电视机的音量:4
电视机的频道:3
电视机的状态为:开机
电视机的音量:4
电视机的频道:75

知识点4 【强化类的封装】

myarray.h

```
1 #ifndef MYARRAY_H
2 #define MYARRAY_H
3
```

```

4
5 class MyArray
6 {
7 private:
8     int capacity;//数组总共能存放的元素个数
9     int size;//数组实际存放的元素个数
10    int *addr;//数组首元素地址
11 public:
12    MyArray();
13    MyArray(int capacity);
14    ~MyArray();
15
16    //往数组的尾部插入数据
17    void pushBack(int data);
18    //获得指定位置的数据
19    int getData(int pos);
20    //修改指定位置的值
21    void setData(int pos,int data);
22    //获取数组的容量（能存放的最大元素个数）
23    int getCapacity(void);
24    //获取数组的实际大小(实际元素的个数)
25    int getSize(void);
26    void printMyArray(void);
27 };
28
29 #endif // MYARRAY_H
30

```

myarray.cpp

```

1  #include "myarray.h"
2  #include<iostream>
3  using namespace std;
4  MyArray::MyArray()
5  {
6      //假如数组的容量为100
7      this->capacity = 100;
8      //数组的size 为0
9      this->size = 0;
10     //根据容量 给数组申请空间
11     this->addr = new int[this->capacity];

```

```
12 }
13
14 MyArray::MyArray(int capacity)
15 {
16     this->capacity = capacity;
17     this->size = 0;
18     this->addr = new int[this->capacity];
19 }
20
21 MyArray::~MyArray()
22 {
23     if(this->addr != NULL)
24     {
25         delete [] this->addr;
26         this->addr = NULL;
27     }
28 }
29
30 void MyArray::pushBack(int data)
31 {
32     if(this->size >= this->capacity)//数组已满
33     {
34         cout<<"数组已满"<<endl;
35         return;
36     }
37     addr[this->size] = data;
38     this->size++;
39 }
40
41 int MyArray::getData(int pos)
42 {
43     if(pos >= this->size || pos < 0)
44     {
45         cout<<"位置无效"<<endl;
46         return -1;
47     }
48
49     return addr[pos];
50 }
51
```

```

52 void MyArray::setData(int pos, int data)
53 {
54     if(pos >= this->size || pos < 0)
55     {
56         cout<<"位置过大"<<endl;
57         return;
58     }
59     addr[pos] = data;
60     return;
61 }
62
63 int MyArray::getCapacity()
64 {
65     return this->capacity;
66 }
67
68 int MyArray::getSize()
69 {
70     return this->size;
71 }
72
73 void MyArray::printMyArray()
74 {
75     int i=0;
76     for(i=0;i<this->size; i++)
77     {
78         cout<<this->addr[i]<<" ";
79     }
80     cout<<endl;
81 }
82

```

main.cpp

```

1  #include <iostream>
2  #include"myarray.h"
3  using namespace std;
4
5  int main(int argc, char *argv[])
6  {
7      //实例化一个数组对象

```

```

8  MyArray arr1;
9  cout<<"容量:"<<arr1.getCapacity()<<endl;
10 cout<<"大小:"<<arr1.getSize()<<endl;
11
12 MyArray arr2(50);
13 cout<<"容量:"<<arr2.getCapacity()<<endl;
14 cout<<"大小:"<<arr2.getSize()<<endl;
15
16 //往数组中插入20个数据
17 int i=0;
18 for(i=0;i<20;i++)
19 {
20 arr2.pushBack(i);
21 }
22 cout<<"容量:"<<arr2.getCapacity()<<endl;
23 cout<<"大小:"<<arr2.getSize()<<endl;
24
25 //遍历数组
26 arr2.printMyArray();
27
28 //更改pos=9的值 2000
29 arr2.setData(9,2000);
30 arr2.printMyArray();
31
32 //得到下标为9的值
33 cout<<arr2.getData(9)<<endl;//
34 return 0;
35 }
36

```

知识点5 【运算符重载】 2-1

1、重载运算符的概述

运算符重载，就是对**已有的运算符**重新进行**定义**，赋予其另一种功能，以适应不同的数据类型

运算符重载的目的：简化操作 让已有的运算符 适应不同的数据类型。

语法：函数的名字由关键字**operator**及其紧跟的**运算符**组成

比如：重载+运算符 ==> **operator+** 重载=号运算 ==> **operator=**

注意：重载运算符 不要更改 运算符的本质操作（+是数据的相加 不要重载成相减）

2、运算符<<的重载

```

1  #include <iostream>
2  #include<string.h>
3  using namespace std;
4  class Person
5  {
6  private:
7      char *name;
8      int num;
9  public:
10     Person(char *name, int num)
11     {
12         this->name = new char[strlen(name)+1];
13         strcpy(this->name,name);
14         this->num = num;
15         cout<<"有参构造"<<endl;
16     }
17     //普通的成员函数
18     void printPerson(void)
19     {
20         cout<<"name = "<<name<<" , num = "<<num<<endl;
21     }
22     ~Person()
23     {
24         if(this->name != NULL)
25         {
26             delete [] this->name;
27             this->name = NULL;
28         }
29         cout<<"析构函数"<<endl;
30     }
31 };
32 int main(int argc, char *argv[])
33 {
34     Person ob1("lucy",18);
35     //普通的成员函数 遍历信息
36     //ob1.printPerson();
37
38     //cout默认输出方式 无法识别 自定义对象 输出格式
39     //cout<<ob1<<endl;//err
40

```

```

41     return 0;
42 }
43

```

截图：

```

37
38     //cout默认输出方式 无法识别 自定义对象 输出格式
39     cout<<ob1<<endl;
40
41     return 0;
42 }
43

```

解决上述问题 需要重载<<

```

32 void operator<<(ostream &out, Person &ob)//out=cout, ob =ob1
33 {
34     //重新实现 输出格式
35     out<<ob.name<<"", "<<ob.num;
36 }
37 int main(int argc, char *argv[])
38 {
39     Person ob1("lucy",18);

```

name 和 num 是私有的
解决办法：??

解决办法：将operator<<设置成友元：

完整代码：

```

1 #include <iostream>
2 #include<string.h>
3 using namespace std;
4 class Person
5 {
6     //设置成友元函数 在函数内 访问Person类中的所有数据
7     friend ostream& operator<<(ostream &out, Person &ob);
8 private:
9     char *name;
10    int num;
11 public:
12    Person(char *name, int num)
13    {
14        this->name = new char[strlen(name)+1];
15        strcpy(this->name,name);
16        this->num = num;
17        cout<<"有参构造"<<endl;
18    }
19    //普通的成员函数
20    void printPerson(void)
21    {

```

```

22  cout<<"name = "<<name<<", num = "<<num<<endl;
23  }
24  ~Person()
25  {
26  if(this->name != NULL)
27  {
28  delete [] this->name;
29  this->name = NULL;
30  }
31  cout<<"析构函数"<<endl;
32  }
33  };
34
35  ostream& operator<<(ostream &out, Person &ob)//out=cout, ob =ob1
36  {
37  //重新实现 输出格式
38  out<<ob.name<<", "<<ob.num;
39
40  //每次执行为 返回值得到cout
41  return out;
42  }
43  int main(int argc, char *argv[])
44  {
45  Person ob1("lucy",18);
46  //普通的成员函数 遍历信息
47  //ob1.printPerson();
48
49  //cout默认输出方式 无法识别 自定义对象 输出格式
50  //cout<<ob1<<endl;//err
51
52  //运算符重载的调用方式1:
53  operator<<(cout, ob1)<<endl;
54  //运算符重载的调用方式2:
55  //对方法1 进行优化 去掉operator,第一个参数 放在运算符<<的左边 第二个参数 放在
  运算符<<的右边
56  cout<<ob1<<endl;//等价operator<<(cout, ob1);
57
58  Person ob2("bob",19);
59  cout<<ob1<<" "<<ob2<<endl;
60
61  return 0;

```



```
62  }  
63
```

运行结果：

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

```
有参构造  
lucy, 18  
lucy, 18  
有参构造  
lucy, 18 bob, 19  
析构造函数  
析构造函数
```

2、重载+运算符：全局函数作为友元 完成运算符重载+

1+1 这种情况下 不需要重载 编译器能够计算

```
1  #include <iostream>  
2  #include<string.h>  
3  using namespace std;  
4  class Person  
5  {  
6      //设置成友元函数 在函数内 访问Person类中的所有数据  
7      friend ostream& operator<<(ostream &out, Person &ob);  
8      friend Person operator+(Person &ob1, Person &ob2);  
9  private:  
10     char *name;  
11     int num;  
12 public:  
13     Person()
```

```

14 {
15     this->name = NULL;
16     this->num = 0;
17     cout<<"无参构造"<<endl;
18 }
19 Person(char *name, int num)
20 {
21     this->name = new char[strlen(name)+1];
22     strcpy(this->name, name);
23     this->num = num;
24     cout<<"有参构造"<<endl;
25 }
26 //普通的成员函数
27 void printPerson(void)
28 {
29     cout<<"name = "<<name<<", num = "<<num<<endl;
30 }
31 ~Person()
32 {
33     if(this->name != NULL)
34     {
35         delete [] this->name;
36         this->name = NULL;
37     }
38     cout<<"析构函数"<<endl;
39 }
40 };
41
42 //全局函数作为友元 完成运算符重载<<
43 ostream& operator<<(ostream &out, Person &ob)//out=cout, ob =ob1
44 {
45     //重新实现 输出格式
46     out<<ob.name<<", "<<ob.num;
47
48     //每次执行为 返回值得到cout
49     return out;
50 }
51 //全局函数作为友元 完成运算符重载+
52 Person operator+(Person &ob1, Person &ob2)//ob1 ob2
53 {

```

```

54 //name+name(字符串追加)
55 char *tmp_name = new char[strlen(ob1.name)+strlen(ob2.name)+1];
56 strcpy(tmp_name,ob1.name);
57 strcat(tmp_name,ob2.name);
58
59 //num+num (数值相加)
60 int tmp_num = ob1.num+ob2.num;
61 Person tmp(tmp_name, tmp_num);
62
63 //释放tmp_name的空间
64 if(tmp_name != NULL)
65 {
66 delete [] tmp_name;
67 tmp_name = NULL;
68 }
69
70 return tmp;
71 }
72 void test01()
73 {
74 Person ob1("lucy",18);
75 //普通的成员函数 遍历信息
76 //ob1.printPerson();
77
78 //cout默认输出方式 无法识别 自定义对象 输出格式
79 //cout<<ob1<<endl;//err
80
81 //运算符重载的调用方式1:
82 operator<<(cout, ob1)<<endl;
83 //运算符重载的调用方式2:
84 //对方法1 进行优化 去掉operator,第一个参数 放在运算符<<的左边 第二个参数 放在
运算符<<的右边
85 cout<<ob1<<endl;//等价operator<<(cout, ob1);
86
87 Person ob2("bob",19);
88 cout<<ob1<<" "<<ob2<<endl;
89 }
90
91 void test02()
92 {
93 Person ob1("lucy",18);

```

```
94  Person ob2("bob", 19);
95
96  cout<<ob1<<endl;
97  cout<<ob2<<endl;
98  //Person ob3 = operator+(ob1,ob2);
99  Person ob3 = ob1+ob2;
100  cout<<ob3<<endl;
101  }
102  int main(int argc, char *argv[])
103  {
104      test02();
105
106      return 0;
107  }
108
```

运行结果：

有参构造
有参构造
lucy, 18
bob, 19
有参构造
lucybob, 37
析构函数
析构函数
析构函数

3、重载+运算符：成员函数 完成运算符重载+

```
1 #include <iostream>
2 #include<string.h>
3 using namespace std;
4 class Person
5 {
```

```
6 //设置成友元函数 在函数内 访问Person类中的所有数据
7 friend ostream& operator<<(ostream &out, Person &ob);
8 private:
9 char *name;
10 int num;
11 public:
12 Person()
13 {
14     this->name = NULL;
15     this->num = 0;
16     cout<<"无参构造"<<endl;
17 }
18 Person(char *name, int num)
19 {
20     this->name = new char[strlen(name)+1];
21     strcpy(this->name,name);
22     this->num = num;
23     cout<<"有参构造"<<endl;
24 }
25 //成员函数 完成运算符重载 ob1用this代替 ob2用参数ob代替
26 Person operator+(Person &ob)
27 {
28     //this ==> &ob1
29
30     //name+name(字符串追加)
31     char *tmp_name = new char[strlen(this->name)+strlen(ob.name)+1];
32     strcpy(tmp_name,this->name);
33     strcat(tmp_name,ob.name);
34
35     //num+num(数值相加)
36     int tmp_num = this->num+ob.num;
37     Person tmp(tmp_name, tmp_num);
38
39     //释放tmp_name的空间
40     if(tmp_name != NULL)
41     {
42         delete [] tmp_name;
43         tmp_name = NULL;
44     }
45
```

```

46     return tmp;
47 }
48
49 //普通的成员函数
50 void printPerson(void)
51 {
52     cout<<"name = "<<name<<" , num = "<<num<<endl;
53 }
54 ~Person()
55 {
56     if(this->name != NULL)
57     {
58         delete [] this->name;
59         this->name = NULL;
60     }
61     cout<<"析构函数"<<endl;
62 }
63 };
64
65 //全局函数作为友元 完成运算符重载<<
66 ostream& operator<<(ostream &out, Person &ob)//out=cout, ob =ob1
67 {
68     //重新实现 输出格式
69     out<<ob.name<<" , "<<ob.num;
70
71     //每次执行为 返回值得到cout
72     return out;
73 }
74
75 void test03()
76 {
77     Person ob1("lucy",18);
78     Person ob2("bob", 19);
79
80     //Person ob3 = ob1.operator+(ob2);
81     Person ob3 = ob1+ob2;
82     cout<<ob3<<endl;
83 }
84 int main(int argc, char *argv[])
85 {

```

```
86  test03();  
87  
88  return 0;  
89  }  
90
```

运行结果：

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

有参构造
有参构造
有参构造
lucybob, 37
析构造函数
析构造函数
析构造函数

运算符的第一个参数：如果是对象 基本用 成员函数 重载运算符