

三、C++学习笔记—核心编程

本阶段，将对C++面向对象编程技术做详细学习，深入C++中的核心和精髓

3.4.4 友元

生活中你的家有客厅(Public)，有你的卧室(Private)

客厅所有来的客人都可以进去，但是你的卧室是私有的，也就是说只有你能进去

但是呢，你也可以允许你的好闺蜜好基友进去。

在程序里，有些私有属性 也想让类外特殊的一些函数或者类进行访问，就需要用到友元的技术

友元的目的就是让一个函数或者类 访问另一个类中私有成员

友元的关键字为 **friend**

友元的三种实现：

- 全局函数做友元
- 类做友元
- 成员函数做友元

3.4.4.1 全局函数做友元

```
1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  //建筑物类
6  class Building
7  {
8      //goodGay全局函数是 Building好朋友，可以访问Building中私有成员--只要写到类上面即可
9      friend void goodGay(Building *building); // 相当于全局函数的声明
10
11  public:
12      Building()
13      {
14          m_SittingRoom = "客厅";
15          m_BedRoom = "卧室";
16      }
17
18
19  public:
20
21      string m_SittingRoom; //客厅
22
23  private:
24
25      string m_BedRoom; //卧室
```

```

26 };
27
28 //全局函数
29 void goodGay(Building *building)
30 {
31     cout << "好基友全局函数 正在访问: " << building->m_SittingRoom << endl;
32     cout << "好基友全局函数 正在访问: " << building->m_BedRoom << endl;
33 }
34
35 void test01()
36 {
37     Building building; //实例化一个对象
38     goodGay(&building);
39
40 }
41
42 int main()
43 {
44
45     test01();
46
47     system("pause");
48     return 0;
49 }

```

3.4.4.2 类做友元

一个类可以访问另一个类中的私有成员

```

1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  //类做有元
6  class Building;
7  class GoodGay
8  {
9  public:
10
11     GoodGay();
12     void visit(); //参观函数, 访问Building中的属性
13
14
15     Building *building;
16 };
17
18 class Building
19 {
20     // GoodGay类是本类的好朋友, 可以访问本类中的私有成员
21     friend class GoodGay;
22 public:

```

```

23     Building(); //构造函数
24
25
26 public:
27     string m_SittingRoom; // 客厅
28
29 private:
30     string m_BedRoom; //卧室
31 };
32
33
34
35 //类外写成员函数
36 Building::Building()
37 {
38     m_SittingRoom = "客厅";
39     m_BedRoom = "卧室";
40 }
41
42 GoodGay::GoodGay()
43 {
44     //创建建筑物对象
45     building = new Building;
46
47 }
48
49
50 void GoodGay::visit()
51 {
52     cout << "好基友类正在访问: " << building->m_SittingRoom << endl;
53     cout << "好基友类正在访问: " << building->m_BedRoom << endl;
54 }
55
56 //测试函数
57 void test01()
58 {
59     GoodGay gg; //实例化一个对象
60     gg.visit();
61 }
62
63
64 int main()
65 {
66
67     test01();
68
69     system("pause");
70     return 0;
71 }

```

3.4.4.3 成员函数做友元

```

1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  //
6  class Building; //先告诉编译器会写这个类，但是先别报错
7  class GoodGay
8  {
9  public:
10
11      GoodGay(); // 构造函数
12
13      Building * building;//
14
15      void visit(); //成员函数 让visit函数可以访问Building中私有成员
16      void visit2(); //让visit2函数不可以访问Building中私有成员
17
18
19 };
20
21 //建筑物类
22 class Building
23 {
24     //告诉编译器，GoodGay类下的visit成员函数作为本类的好朋友，可以访问私有成员
25     friend void GoodGay::visit(); //全局函数
26
27 public:
28     Building(); //构造函数
29
30 public:
31     string m_SittingRoom; //客厅
32
33 private:
34
35     string m_BedRoom; //卧室
36
37 };
38
39
40 //类外实现成员函数——内部属性实现赋初值的操作
41 Building::Building()
42 {
43     m_SittingRoom = "客厅";
44     m_BedRoom = "卧室";
45 }
46
47
48 GoodGay::GoodGay()
49 {
50     building = new Building; //创建了一个Building堆区，并且用指针维护这个建筑物吧
51
52 }
53

```

```

54 void GoodGay::visit()
55 {
56     cout << "visit函数正在访问: " << building->m_SittingRoom << endl;
57     cout << "visit函数正在访问: " << building->m_BedRoom << endl;
58 }
59
60 void GoodGay::visit2()
61 {
62     cout << "visit2函数正在访问: " << building->m_SittingRoom << endl;
63     //cout << "visit2函数正在访问: " << building->BedRoom << endl;
64
65 }
66
67
68 //测试函数
69 void test01()
70 {
71     GoodGay gg;
72     gg.visit();
73     gg.visit2();
74 }
75
76 int main()
77 {
78
79     test01();
80
81     system("pause");
82     return 0;
83 }

```

3.4.5 运算符重载

运算符重载概念：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

3.4.5.1 加号运算符重载

对于内置数据类型，编译器知道如何进行运算

```
int a = 10;
int b = 10;
int c = a + b;
```

```
class Person
{
public:

    int m_A;
    int m_B;
}
```

```
Person p1;
p1.m_A = 10;
p1.m_B = 10;
```

```
Person p2;
p2.m_A = 10;
p2.m_B = 10;
```

```
Person p3 = p1 + p2;
```

通过自己写成员函数，实现两个对象相加属性后返回新的对象

```
Person PersonAddPerson(Person &p)
{
    Person temp;
    temp.m_A = this->m_A + p.m_A;
    temp.m_B = this->m_B + p.m_B;
    return temp;
}
```

编译器给起了一个通用名称

通过成员函数重载+号

```
Person operator+ (Person &p)
{
    Person temp;
    temp.m_A = this->m_A + p.m_A;
    temp.m_B = this->m_B + p.m_B;
    return temp;
}
```

```
Person p3 = p1.operator+(p2);
简化为
Person p3 = p1 + p2;
```

通过全局函数重载+

```
Person operator+ (Person &p1, Person &p2)
{
    Person temp;
    temp.m_A = p1.m_A + p2.m_A;
    temp.m_B = p1.m_B + p2.m_B;
    return temp;
}
```

```
Person p3 = operator+ (p1, p2)
简化为
Person p3 = p1 + p2;
```

作用：实现两个自定义数据类型相加的运算

```
1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  //加号运算符重载
6  class Person
7  {
8
9  public:
10     //1、成员函数重载+号
11     //Person operator+(Person &p)
12     //{
13     //    Person temp;
14     //    temp.m_A = this->m_A + p.m_A;
15     //    temp.m_B = this->m_B + p.m_B;
16     //    return temp;
17     //}
18
19 public:
20     int m_A;
21     int m_B;
22
23 private:
24
25 };
26
27 //2、全局函数重载+号
28 Person operator+(Person &p1, Person &p2)
29 {
30     Person temp;
31     temp.m_A = p1.m_A + p2.m_A;
```

```

32     temp.m_B = p1.m_B + p2.m_B;
33     return temp;
34 }
35
36 //函数重载的版本
37 Person operator+(Person &p1, int num)
38 {
39     Person temp;
40     temp.m_A = p1.m_A + num;
41     temp.m_B = p1.m_B + num;
42     return temp;
43 }
44 }
45
46
47
48 //测试函数
49 void test01()
50 {
51     Person p1;
52     p1.m_A = 10;
53     p1.m_B = 10;
54
55     Person p2;
56     p2.m_A = 10;
57     p2.m_B = 10;
58     //成员函数本质的调用
59     //Person p3 = p1.operator+(p2);
60
61     //全局函数重载本质调用
62     //Person p3 = operator+(p1, p2);
63     Person p3 = p1+p2;
64     //运算符重载 也可以发生函数重载
65
66     Person p4 = p1 + 100; //Person +int
67
68     cout << "p3.m_A=" << p3.m_A << endl;
69     cout << "p3.m_B=" << p3.m_B << endl;
70
71     cout << "p4.m_A=" << p4.m_A << endl;
72     cout << "p4.m_B=" << p4.m_B << endl;
73
74 }
75
76 int main()
77 {
78
79     test01();
80
81     system("pause");
82     return 0;
83 }

```

总结1：对于内置的数据类型的表达式的运算符是不可能改变的

总结2：不要滥用运算符重载(加法写成减法，减法写成加法)

3.4.5.2 左移运算符重载

作用：可以输出自定义数据类型

```
1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  //左移运算符重载
6  class Person
7  {
8
9      friend ostream & operator<<(ostream &cout, Person p); //可以访问私有变量
10
11  public:
12      Person(int a, int b)
13      {
14          m_A = a;
15          m_B = b;
16      }
17
18  public:
19
20      //利用成员函数重载，左移运算符 p.operator<<(cout) 简化版本p<<cout
21      //不会利用成员函数重载<<运算符，因为无法实现cout在左侧
22      /*void operator<<(Person &p)
23      {
24
25      }*/
26
27  private:
28      int m_A;
29      int m_B;
30
31
32  };
33
34  //只能利用全局函数重载左移运算符
35  ostream & operator<<(ostream &cout, Person p) //本质 operator<<(cout p) 简化cout<<p
36  {
37      cout << "m_A=" << p.m_A << " m_B=" << p.m_B;
38      return cout;
39  }
40
41
42  void test01()
43  {
44      Person p(10,10);
```



```
45     //p.m_A = 10;
46     //p.m_B = 10;
47
48     cout << p << endl;
49
50 }
51
52 int main()
53 {
54
55     test01();
56
57     system("pause");
58     return 0;
59 }
```

总结：重载左移运算符配合友元可以实现输出自定义数据类型