

知识点1【类的概述】（了解）

- 1、定义一个类 关键字class

知识点2【课堂练习】（了解）

练习1：请设计一个Person类

练习2：设置立方体的类

案例3：点和圆的关系

知识点3【成员函数在类外实现】（了解）

知识点4【类在其他文件实现】（了解）

知识点5【构造函数】（重要）

- 1、构造函数的概述
- 2、构造函数的定义（重要,公有方式创建）
- 3、构造函数的调用时机

知识点6【析构函数】（重要）

知识点7【拷贝构造函数】（重要）

- 1、拷贝构造的定义
- 2、拷贝构造 和 无参构造 有参构造的关系
- 3、拷贝构造几种调用形式（了解）

1、旧对象给新对象初始化 调用拷贝构造

2、给对象取别名 不会调用拷贝构造

3、普通对象作为函数参数 调用函数时 会发生拷贝构造

4、函数返回值普通对象（Visual Studio会发生拷贝构造）（Qtcreator,linux不会发生）

知识点8【拷贝构造的浅拷贝和深拷贝】（重要）

知识点9【初始化列表】（重要）

知识点10【对象数组】（重要）

知识点11【explicit关键字】（重要）

知识点1【类的概述】（了解）

类将数据和方法封装在一起，加以权限区分，用户只能通过公共方法 访问私有数据。

1、定义一个类 关键字class

类的权限分为：private、protected、public。但是在类的内部 不存在 权限之分。只是对类外有效。

如果类不涉及到继承，private、protected没有区别，都是私有属性。

```
1 #include <iostream>
2
3 using namespace std;
4
5 //类Data1 是一个类型
6 class Data1
7 {
8     //类中 默认为私有
9     private:
10     int a;//不要给类中成员 初始化
11     protected://保护
12     int b;
13     public://公共
14     int c;
15     //在类的内部 不存在 权限之分
16     void showData(void)
17     {
18         cout<<a<<" "<<b<<" "<<c<<endl;
19     }
20 };
21 void test01()
22 {
23     //类实例化一个对象
24     Data1 ob;
25     //类外不能直接访问 类的私有和保护数据
26     //cout<<ob.a <<endl;
```

```

27 //cout<<ob.b <<endl;
28 cout<<ob.c <<endl;
29
30 //类中的成员函数 需要对象调用
31 ob.showData();
32 }

```

访问属性	属性	对象内部	对象外部
public	公有	可访问	可访问
protected	保护	可访问	不可访问
private	私有	可访问	不可访问

建议：

将数据设置成私有，将方法设置成公有。

知识点2 【课堂练习】（了解）

练习1：请设计一个Person类

请设计一个Person类，Person类具有name和age属性，提供初始化函数 (Init)，并提供对name和age的读写函数(set, get)，但必须确保age的赋值在有效范围内(0-100),超出有效范围，则拒绝赋值，并提供方法输出姓名和年龄

```

1 #include <iostream>
2
3 using namespace std;
4
5 //类Data1 是一个类型
6 class Data1
7 {
8     //类中 默认为私有
9     private:
10     int a;//不要给类中成员 初始化
11     protected://保护
12     int b;
13     public://公共
14     int c;
15     //在类的内部 不存在 权限之分
16     void showData(void)
17     {
18     cout<<a<<" "<<b<<" "<<c<<endl;

```

```
19  }
20  };
21  void test01()
22  {
23      //类实例化一个对象
24      Data1 ob;
25      //类外不能直接访问 类的私有和保护数据
26      //cout<<ob.a <<endl;
27      //cout<<ob.b <<endl;
28      cout<<ob.c <<endl;
29
30      //类中的成员函数 需要对象调用
31      ob.showData();
32  }
33  #include<string.h>
34  class Person
35  {
36  private:
37      char mName[32];
38      int mAge;
39  public:
40      //初始化成员
41      void init(char *name, int age)
42      {
43          strcpy(mName, name);
44          if(age>=0 && age<=100)
45          {
46              mAge = age;
47          }
48          else
49          {
50              cout<<"年龄无效"<<endl;
51          }
52          return;
53      }
54      //设置name
55      void setName(char *name)
56      {
57          strcpy(mName, name);
58      }
```

```
59 //获取name
60 char *getName(void)
61 {
62     return mName;
63 }
64
65 //设置age
66 void setAge(int age)
67 {
68     if(age>=0 && age<=100)
69     {
70         mAge = age;
71     }
72     else
73     {
74         cout<<"年龄无效"<<endl;
75     }
76 }
77 //得到age
78 int getAge(void)
79 {
80     return mAge;
81 }
82
83 //显示所有数据
84 void showPerson(void)
85 {
86     cout<<mName<<" "<<mAge<<endl;
87 }
88 };
89
90 void test02()
91 {
92     Person ob1;
93
94     ob1.init("lucy", 18);
95     ob1.showPerson();
96
97     ob1.setName("bob");
98     cout<<"年龄:"<<ob1.getAge()<<endl;
```

```

99  ob1.showPerson();
100 }
101

```

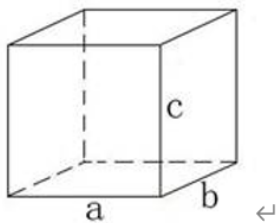
```

lucy 18
年龄:18
bob 18

```

练习2：设置立方体的类

设计立方体类(Cube)，求出立方体的面积($2ab + 2ac + 2bc$)和体积($a * b * c$)，分别用全局函数和成员函数判断两个立方体是否相等。↵



```

1  #include <iostream>
2
3  using namespace std;
4
5  //类Data1 是一个类型
6  class Data1
7  {
8      //类中 默认为私有
9      private:
10     int a;//不要给类中成员 初始化
11     protected://保护
12     int b;
13     public://公共
14     int c;
15     //在类的内部 不存在 权限之分
16     void showData(void)
17     {
18         cout<<a<<" "<<b<<" "<<c<<endl;
19     }
20 };

```

```
21 void test01()
22 {
23     //类实例化一个对象
24     Data1 ob;
25     //类外不能直接访问 类的私有和保护数据
26     //cout<<ob.a <<endl;
27     //cout<<ob.b <<endl;
28     cout<<ob.c <<endl;
29
30     //类中的成员函数 需要对象调用
31     ob.showData();
32 }
33 #include<string.h>
34 class Person
35 {
36 private:
37     char mName[32];
38     int mAge;
39 public:
40     //初始化成员
41     void init(char *name, int age)
42     {
43         strcpy(mName, name);
44         if(age>=0 && age<=100)
45         {
46             mAge = age;
47         }
48         else
49         {
50             cout<<"年龄无效"<<endl;
51         }
52         return;
53     }
54     //设置name
55     void setName(char *name)
56     {
57         strcpy(mName, name);
58     }
59     //获取name
60     char *getName(void)
```

```
61 {
62     return mName;
63 }
64
65 //设置age
66 void setAge(int age)
67 {
68     if(age>=0 && age<=100)
69     {
70         mAge = age;
71     }
72     else
73     {
74         cout<<"年龄无效"<<endl;
75     }
76 }
77 //得到age
78 int getAge(void)
79 {
80     return mAge;
81 }
82
83 //显示所有数据
84 void showPerson(void)
85 {
86     cout<<mName<<" "<<mAge<<endl;
87 }
88 };
89
90 void test02()
91 {
92     Person ob1;
93
94     ob1.init("lucy", 18);
95     ob1.showPerson();
96
97     ob1.setName("bob");
98     cout<<"年龄:"<<ob1.getAge()<<endl;
99     ob1.showPerson();
100 }
```



```
101
102 class Cube
103 {
104 private:
105     int mA;
106     int mB;
107     int mC;
108 public:
109     void setA(int a)
110     {
111         mA = a;
112     }
113     int getA(void)
114     {
115         return mA;
116     }
117     void setB(int b)
118     {
119         mB = b;
120     }
121     int getB(void)
122     {
123         return mB;
124     }
125     void setC(int c)
126     {
127         mC = c;
128     }
129     int getC(void)
130     {
131         return mC;
132     }
133
134     //获取面积
135     int getS(void)
136     {
137         return (mA*mB+mB*mC+mC*mA)*2;
138     }
139     //获取体积
140     int getV(void)
```

```
141 {
142     return mA*mB*mC;
143 }
144 //成员函数实现
145 bool compareCube02(Cube &ob)
146 {
147     if(mA==ob.mA && mB ==ob.mB && mC == ob.mC)
148     {
149         return true;
150     }
151     return false;
152 }
153 };
154
155 //全局函数 比较两个立方体是否先等
156 bool compareCube01(Cube &ob1, Cube &ob2)
157 {
158     if(ob1.getA()==ob2.getA() && ob1.getB() ==ob2.getB() && ob1.getC() == c
b2.getC())
159     {
160         return true;
161     }
162
163     return false;
164 }
165
166 void test03()
167 {
168     Cube ob1;
169     ob1.setA(10);
170     ob1.setB(20);
171     ob1.setC(30);
172
173     cout<<"面积:"<<ob1.getS()<<endl;
174     cout<<"体积:"<<ob1.getV()<<endl;
175
176     Cube ob2;
177     ob2.setA(10);
178     ob2.setB(20);
179     ob2.setC(30);
180
```

```

181 // if(compareCube01(ob1, ob2))
182 if(ob1.compareCube02(ob2))
183 {
184     cout<<"相等"<<endl;
185 }
186 else
187 {
188     cout<<"不相等"<<endl;
189 }
190 }
191
192 int main(int argc, char *argv[])
193 {
194     test03();
195     return 0;
196 }

```

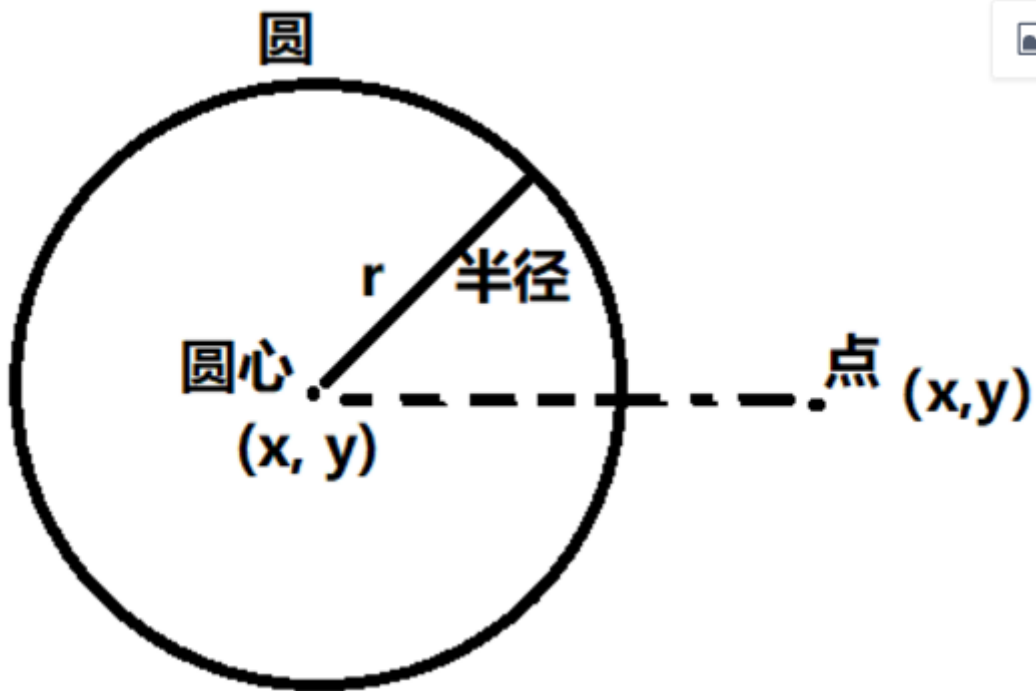
面积:2200
体积:6000
相等

案例3：点和圆的关系

设计一个圆形类 (AdvCircle) , 和一个点类 (Point) , 计算点和圆的关系。

假如圆心坐标为 x_0, y_0 , 半径为 r , 点的坐标为 x_1, y_1 :

- 1) 点在圆上: $(x_1 - x_0)(x_1 - x_0) + (y_1 - y_0)(y_1 - y_0) == r * r$
- 2) 点在圆内: $(x_1 - x_0)(x_1 - x_0) + (y_1 - y_0)(y_1 - y_0) < r * r$
- 3) 点在圆外: $(x_1 - x_0)(x_1 - x_0) + (y_1 - y_0)(y_1 - y_0) > r * r$




```
1 class Point
2 {
3 private:
4     int mX;
5     int mY;
6 public:
7     void setX(int x)
8     {
9         mX = x;
10    }
11    int getX(void)
12    {
13        return mX;
14    }
15    void setY(int y)
16    {
17        mY = y;
18    }
19    int getY(void)
20    {
21        return mY;
22    }
23 };
24
25 class Circle
```

```
26 {
27 private:
28     Point p;//对象作为类的成员变量
29     int mR;
30 public:
31     void setPoint(int x, int y)
32     {
33         p.setX(x);
34         p.setY(y);
35     }
36     Point getPoint(void)
37     {
38         return p;
39     }
40     void setR(int r)
41     {
42         mR = r;
43     }
44     int getR(void)
45     {
46         return mR;
47     }
48
49     //判断点 在圆的位置
50     int pointIsOnCircle(Point &ob)
51     {
52         int len = (ob.getX()-p.getX())*(ob.getX()-p.getX())+\
53         (ob.getY()-p.getY())*(ob.getY()-p.getY());
54         if(len == mR*mR)
55         {
56             return 0;
57         }
58         else if(len > mR*mR)
59         {
60             return 1;
61         }
62         else if(len < mR*mR)
63         {
64             return -1;
65         }
```

```

66  }
67  };
68
69  void test04()
70  {
71      //实例化一个点的对象
72      Point p;
73      p.setX(5);
74      p.setY(5);
75
76      //实例化一个圆的对象
77      Circle cir;
78      cir.setPoint(2,2);
79      cir.setR(5);
80      if(cir.pointIsOnCircle(p) == 0)
81      {
82          cout<<"点在圆上"<<endl;
83      }
84      else if(cir.pointIsOnCircle(p) > 0)
85      {
86          cout<<"点在圆外"<<endl;
87      }
88      else if(cir.pointIsOnCircle(p) < 0)
89      {
90          cout<<"点在圆内"<<endl;
91      }
92  }

```

 C:\Qt\Qt5.8.0\Tools\

点在圆内

知识点3 【成员函数在类外实现】（了解）

```

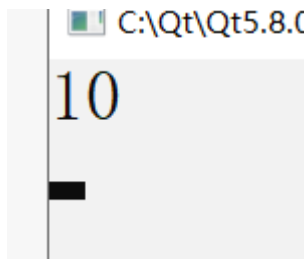
1  class Data2
2  {
3  private:
4      int mA;
5  public:
6      void setA(int a);
7      int getA(void);

```

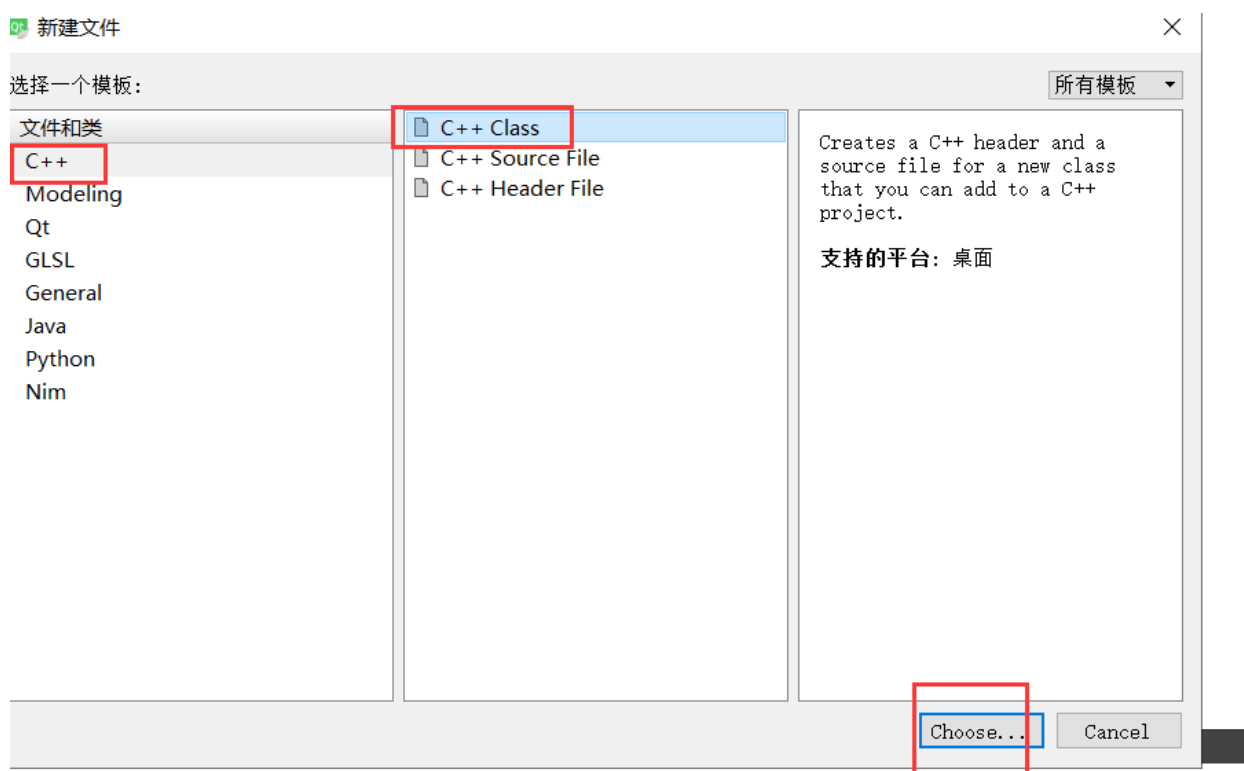
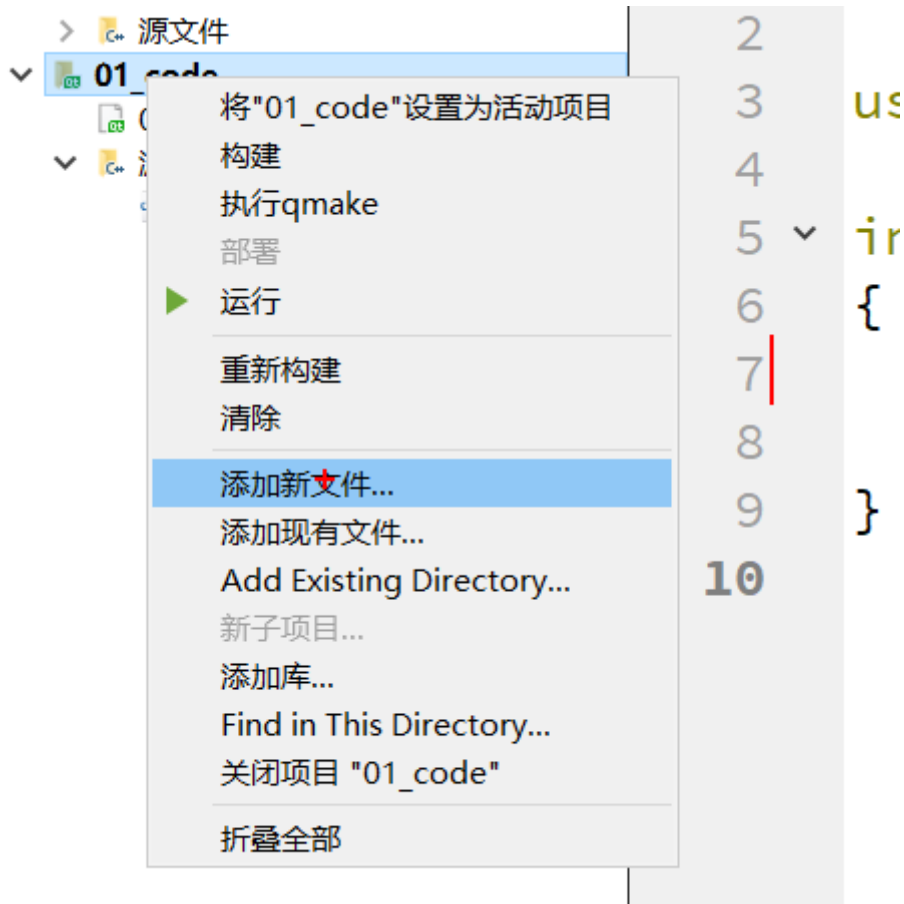
```

8  };
9  void test05()
10 {
11     Data2 ob;
12     ob.setA(10);
13     cout<<ob.getA()<<endl;
14 }
15 int main(int argc, char *argv[])
16 {
17     test05();
18     return 0;
19 }
20
21 void Data2::setA(int a)
22 {
23     mA = a;
24 }
25
26 int Data2::getA()
27 {
28     return mA;
29 }

```



知识点4 【类在其他文件实现】（了解）





C++ Class

Details
Summary

Define Class

类名称

Class name: Data

Base class: <Qt::Item>

- ☐ Include QObject
- ☐ Include QWidget
- ☐ Include QMainWindow
- ☐ Include QDeclarativeItem - Qt Quick 1
- ☐ Include QQuickItem - Qt Quick 2
- ☐ Include QSharedData

Header file: data.h

自动完成

Source file: data.cpp

Path: D:\work\bk2103\code\c++\day02\01_code

浏览...

下一步(N)

取消

头文件定义类，cpp实现类的成员函数

data.h

```
1 #ifndef DATA_H
2 #define DATA_H
3
4
5 class Data
6 {
7 private:
8     int mA;
9 public:
10     int getA(void);
11     void setA(int a);
12 };
13
14 #endif // DATA_H
```

data.cpp

```
1 #include "data.h"
2
3
4 int Data::getA()
5 {
6     return mA;
7 }
```

```

8
9 void Data::setA(int a)
10 {
11     mA = a;
12 }
13

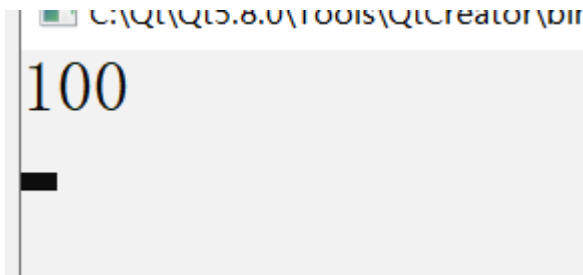
```

main.cpp

```

1 #include <iostream>
2 #include "data.h"
3 using namespace std;
4
5 int main(int argc, char *argv[])
6 {
7     Data ob;
8     ob.setA(100);
9     cout<<ob.getA()<<endl;
10    return 0;
11 }
12

```



知识点5【构造函数】（重要）

1、构造函数的概述

类实例化对象的时候 系统自动调用构造函数 完成对象的初始化。

如果用户不提供构造函数 编译器 会自动添加一个默认的构造函数（空函数）

2、构造函数的定义（重要,公有方式创建）

构造函数名 和 类名相同，没有返回值类型（连void都不可以），可以有参数（可以重载）

先给对象开辟空间（实例化） 然后调用构造函数（初始化）

```

1 class Data1
2 {
3 public:
4     int mA;
5 public:

```

```
6 //无参构造函数
7 Data1()
8 {
9     mA=0;
10    cout<<"无参构造函数"<<endl;
11 }
12 //有参构造函数
13 Data1(int a)
14 {
15     mA=a;
16     cout<<"有参构造函数 mA="<<mA<<endl;
17 }
18 };
19 void test01()
20 {
21     //隐式调用无参构造函数（推荐）
22     Data1 ob1;
23
24     //显示调用无参构造函数
25     Data1 ob2 = Data1();
26
27     //隐式调用有参构造函数（推荐）
28     Data1 ob3(10);
29
30     //显示调用有参构造函数
31     Data1 ob4 = Data1(10);
32
33     //匿名对象(无参) 当前语句技术 立即释放
34     Data1();
35     Data1(20);
36
37     //构造函数隐式转换（类中只有一个数据成员）
38     Data1 ob5 = 100;
39 }
```

```
无参构造函数
无参构造函数
有参构造函数 mA=10
有参构造函数 mA=10
无参构造函数
有参构造函数 mA=20
有参构造函数 mA=100
```

3、构造函数的调用时机

如果用户不提供任何构造函数 编译器默认提供一个空的无参构造。

如果用户定义了构造函数（不管是有参、无参），编译器不再提供默认构造函数。

知识点6 【析构函数】（重要）

当对象生命周期结束的时候 系统自动调用析构函数。

函数名和类名称相同，在函数名前加~，没有返回值类型，没有函数形参。（不能被重载）

先调用析构函数 再释放对象的空间。

```
1  class Data1
2  {
3  public:
4      int mA;
5  public:
6      //无参构造函数
7      Data1()
8      {
9          mA=0;
10         cout<<"无参构造函数"<<endl;
11     }
12
13
14     //有参构造函数
15     Data1(int a)
16     {
17         mA=a;
```

```

18  cout<<"有参构造函数 mA="<<mA<<endl;
19  }
20
21  //析构函数
22  ~Data1()
23  {
24  cout<<"析构函数 mA="<<mA<<endl;
25  }
26  };

```

```

Data1 ob1(10);
void test02()
{
    Data1 ob2(20);
    {
        Data1 ob3(30);
    }
    Data1 ob4(40);
}

```

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

```

有参构造函数 mA=10
有参构造函数 mA=20
有参构造函数 mA=30
析构函数 mA=30
有参构造函数 mA=40
析构函数 mA=40
析构函数 mA=20
析构函数 mA=10

```

一般情况下，空的析构函数就足够。但是如果一个类有指针成员，这个类必须写析构函数，释放指针成员所指向空间。

```

1  #include<stdlib.h>
2  #include<string.h>
3  class Data2
4  {
5  public:
6  char *name;//指针成员
7  public:
8  Data2()
9  {
10  name=NULL;
11  }
12
13  Data2(char *str)
14  {
15  name = (char *)calloc(1, strlen(str)+1);
16  strcpy(name, str);
17  cout<<"有参构造 name="<<name<<endl;
18  }

```

```

19 ~Data2()
20 {
21     cout<<"析构函数name = "<<name<<endl;
22     if(name != NULL)
23     {
24         free(name);
25         name=NULL;
26     }
27 }
28 };
29 void test03()
30 {
31     Data2 ob1("hello world");
32 }

```

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

```

有参构造 name=hello world
析构函数name = hello world

```

知识点7 【拷贝构造函数】（重要）

1、拷贝构造的定义

拷贝构造：本质是构造函数

拷贝构造的调用时机：**旧对象** 初始化 **新对象** 才会调用**拷贝构造**。

```

1 Data4(const Data4 &ob)//ob就是旧对象的别名
2 {
3     //一旦实现 了拷贝构造 必须完成赋值动作
4     mA = ob.mA;
5     cout<<"拷贝构造函数"<<endl;
6 }

```

```

1 class Data4
2 {
3 public:
4     int mA;
5 public:
6     Data4()

```

```

7  {
8  mA = 0;
9  cout<<"无参构造 mA = "<<mA<<endl;
10 }
11 Data4(int a)
12 {
13 mA = a;
14 cout<<"有参构造 mA = "<<mA<<endl;
15 }
16 #if 1
17 Data4(const Data4 &ob)//ob就是旧对象的别名
18 {
19 //一旦实现 了拷贝构造 必须完成赋值动作
20 mA = ob.mA;
21 cout<<"拷贝构造函数"<<endl;
22 }
23 #endif
24 ~Data4()
25 {
26 cout<<"析构函数 mA = "<<mA<<endl;
27 }
28 };
29
30 void test04()
31 {
32 Data4 ob1(10);
33 Data4 ob2 = ob1;
34 cout<<ob2.mA <<endl;
35 }

```

如果用户不提供拷贝构造 编译器会自动提供一个默认的拷贝构造（完成赋值动作--浅拷贝）

2、拷贝构造 和 无参构造 有参构造的关系

如果用户定义了 拷贝构造或者有参构造 都会屏蔽无参构造。

如果用户定义了 无参构造或者有参构造 不会屏蔽拷贝构造。

3、拷贝构造几种调用形式（了解）

1、旧对象给新对象初始化 调用拷贝构造

```

1 Data4 ob1(10);
2 Data4 ob2 = ob1;//调用拷贝构造

```

2、给对象取别名 不会调用拷贝构造

```
1 Data4 ob1(10);  
2 Data4 &ob2 = ob1; //不会调用拷贝构造
```

3、普通对象作为函数参数 调用函数时 会发生拷贝构造

```
void fun01(Data4 ob) //Data4 ob = ob1;  
{  
    cout<<ob.mA<<endl;  
}  
  
void test04()  
{  
    Data4 ob1(100);  
  
    fun01(ob1);  
}
```

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

有参构造 mA = 100
拷贝构造函数
100
析构函数 mA = 100
析构函数 mA = 100

4、函数返回值普通对象 (Visual Studio会发生拷贝构造) (Qtcreator,linux不会发生)

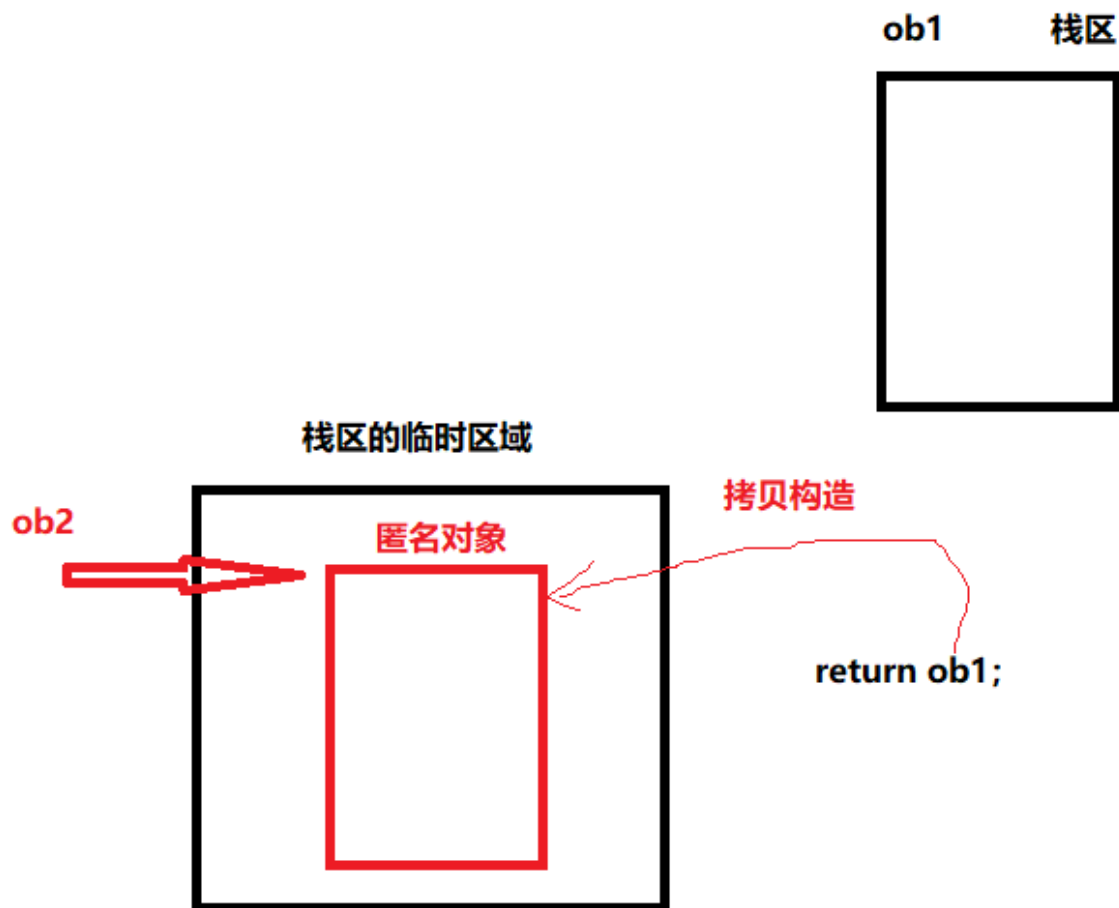
Visual Studio会发生拷贝构造

```
Data4 getObject(void)  
{  
    Data4 ob1(10);  
    return ob1;  
}  
  
void test04()  
{  
    Data4 ob2 = getObject();  
}  
  
int main(int argc, char* argv[])  
{  
    test04();  
    return 0;  
}
```

Microsoft Visual Studio 调试控制台

有参构造 mA = 10
拷贝构造函数
析构函数 mA = 10
析构函数 mA = 10

D:\work\bk2103\code
要在调试停止时自动
按任意键关闭此窗口.



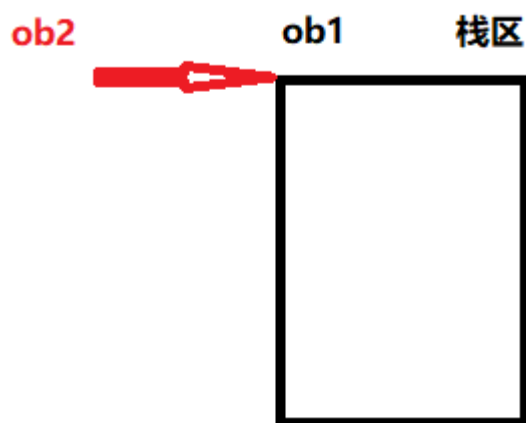
Qtcreator,linux不会发生

```
Data4 getObject(void)
{
    Data4 ob1(10);
    return ob1;
}

void test04()
{
    Data4 ob2 = getObject();
}
```

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

有参构造 mA = 10
析构函数 mA = 10



知识点8 【拷贝构造的浅拷贝和深拷贝】（重要）

默认的拷贝构造 都是浅拷贝。

如果类中**没有指针成员**，不用实现拷贝构造和析构函数。

如果类中**有指针成员**，必须实现析构函数释放指针成员指向的堆区空间，必须实现拷贝构造完成深拷贝动作。

```
1 #include<iostream>
2 #include<string.h>
3 using namespace std;
4 class Data5
5 {
6 public:
7     char* name;
8 public:
9     Data5()
10    {
11        name = NULL;
12    }
13     Data5(char* str)
14    {
15        name = (char*)calloc(1, strlen(str) + 1);
16        strcpy(name, str);
17        cout << "有参构造 name=" << name << endl;
18    }
19     Data5(const Data5& ob)//深拷贝
20    {
21        //为对象的指针成员申请独立的空间
22        name = (char*)calloc(1, strlen(ob.name) + 1);
```

```

23  strcpy(name, ob.name);
24  cout << "拷贝构造函数" << endl;
25  }
26  ~Data5()
27  {
28  cout << "析构函数name = " << name << endl;
29  if (name != NULL)
30  {
31  free(name);
32  name = NULL;
33  }
34  }
35  };
36  void test05()
37  {
38  Data5 ob1((char *)"hello world\n");
39  Data5 ob2 = ob1;
40  }

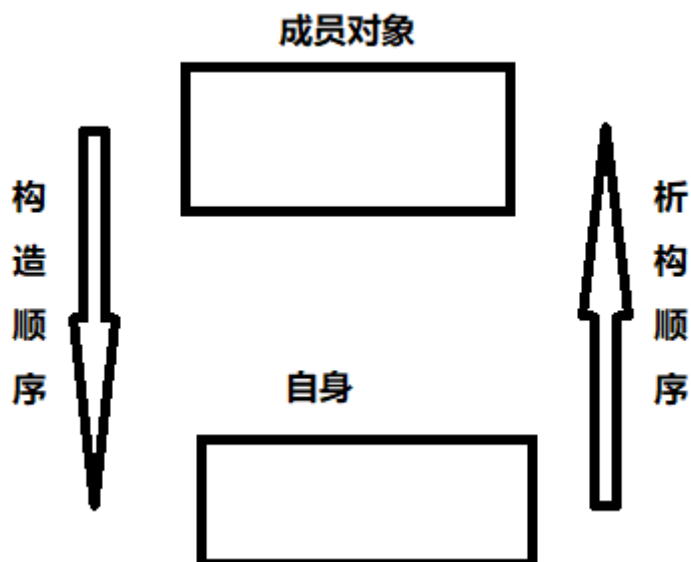
```

设计一个类：无参构造、有参构造、析构函数、拷贝构造

知识点9【初始化列表】（重要）

一个类的对象 作为另一个类的成员：成员对象

如果类中想调用成员对象的有参构造 必须使用初始化列表。



类会自动调用成员对象的无参构造。

类想调用成员对象 有参构造 必须使用初始化列表。

```

1  B(int a, int b):ob(a)
2  {

```

```
3
4 }
```

```
1 class A
2 {
3 public:
4     int mA;
5 public:
6     A()
7     {
8         mA = 0;
9         cout<<"A的无参构造"<<endl;
10    }
11    A(int a)
12    {
13        mA = a;
14        cout<<"A的有参构造"<<endl;
15    }
16    ~A()
17    {
18        cout<<"A的析构函数"<<endl;
19    }
20 };
```

```
1 class B
2 {
3 public:
4     int mB;
5     A ob; //成员对象
6 public:
7     B()
8     {
9         cout<<"B类的无参构造"<<endl;
10    }
11    //初始化列表 成员对象 必须使用对象名+（） 重要
12    B(int a, int b):ob(a)
13    {
14        mB = b;
15        cout<<"B类的有参构造"<<endl;
16    }
```

```

17  ~B()
18  {
19      cout<<"B的析构函数"<<endl;
20  }
21  };

```

```

1  int main(int argc, char *argv[])
2  {
3      B ob1(10,20);
4      cout<<"mA ="<<ob1.ob.mA<<" , mB ="<<ob1.mB<<endl;
5      return 0;
6  }

```

知识点10【对象数组】（重要）

对象数组：本质是数组 数组的每个元素是对象

```

1  class A
2  {
3  public:
4      int mA;
5  public:
6      A()
7      {
8          mA = 0;
9          cout<<"A的无参构造 mA="<<mA<<endl;
10     }
11     A(int a)
12     {
13         mA = a;
14         cout<<"A的有参构造mA="<<mA<<endl;
15     }
16     ~A()
17     {
18         cout<<"A的析构函数 mA = "<<mA<<endl;
19     }
20 };

```

```

1  void test02()
2  {
3      //对象数组 每个元素都会自动调用构造和析构函数

```

```
4 //对象数组不初始化 每个元素 调用无参构造
5 A arr1[5];
6
7 //对象数组的初始化 必须显示使用有参构造 逐个元素初始化
8 A arr2[5]={A(10),A(20),A(30),A(40),A(50) };
9 int n =sizeof(arr2)/sizeof(arr2[0]);
10 int i=0;
11 for(i=0;i<n;i++)
12 {
13     cout<<arr2[i].mA<<" ";
14 }
15 cout<<endl;
16 }
```

```
A的无参构造 mA=0
A的无参构造 mA=0
A的无参构造 mA=0
A的无参构造 mA=0
A的无参构造 mA=0
A的有参构造 mA=10
A的有参构造 mA=20
A的有参构造 mA=30
A的有参构造 mA=40
A的有参构造 mA=50
10 20 30 40 50
A的析构函数 mA = 50
A的析构函数 mA = 40
A的析构函数 mA = 30
A的析构函数 mA = 20
A的析构函数 mA = 10
A的析构函数 mA = 0
A的析构函数 mA = 0
A的析构函数 mA = 0
A的析构函数 mA = 0
A的析构函数 mA = 0
```

知识点11 【explicit关键字】（重要）

explicit防止构造函数隐式转换

explicit修饰构造函数

```
1 //允许有参构造 隐式转换
2 A(int a)
3 {
4     mA = a;
```

```
5  cout<<"A的有参构造mA="<<mA<<endl;  
6  }
```

```
1  //构造函数隐式转换（类中只有一个数据成员）  
2  A ob1=100;//ok
```

```
1  //防止有参构造 隐式转换  
2  explicit A(int a)  
3  {  
4      mA = a;  
5      cout<<"A的有参构造mA="<<mA<<endl;  
6  }
```

```
1  //构造函数隐式转换（类中只有一个数据成员）  
2  A ob1=100;//err 转换失败
```