知识点1【类的空间大小】

知识点2【在类内声明 类外定义 成员函数】

知识点3【类的定义在头文件 成员函数 是在cpp文件中实现】（重要）

知识点4【构造和析构函数的概述】

知识点5【构造和析构函数定义】(重要)

构造函数语法：

析构函数语法：

知识点6【构造函数的分类以及调用】

1、构造函数分类：

按参数类型：分为无参构造函数和有参构造函数

按类型分类：普通构造函数和拷贝构造函数(复制构造函数)

2、构造函数的调用

注意：在同一作用域 构造和析构的顺序相反

3、拷贝构造函数（系统提供一个拷贝构造函数 赋值操作）

4、拷贝构造函数的注意事项：

1、不能调用拷贝构造函数去初始化匿名对象,也就是说以下代码不正确

2、对象作为函数的参数 如果实参与形参 都是普通对象 那么就会调用拷贝构造

3、函数返回局部对象 在qt中会被优化 从而调用不了拷贝构造

知识点7【构造函数的调用规则】（重要）

系统会对任何一个类提供3个函数成员函数：

1、如果用户提供了有参构造 将屏蔽 系统的默认构造函数。

2、如果用户提供了有参构造 不会屏蔽 系统的默认拷贝构造函数。

3、如果用户提供了拷贝构造函数 将屏蔽 系统的默认构造函数、默认拷贝构造函数。

总结：

知识点8【深拷贝与浅拷贝】

知识点9【初始化列表】

注意：初始化成员列表(参数列表)只能在构造函数使用

知识点10【类的对象作为另一个类的成员】

知识点11【explicit关键字】

知识点12【new和delete】从堆区申请空间

1、和malloc calloc realloc比较

1.1、malloc返回一个void指针，c++不允许将void赋值给其他任何指针，必须强转。

1.2、malloc可能申请内存失败，所以必须判断返回值来确保内存分配成功。

1.3、malloc 不会调用构造函数。free 不会调用析构函数（重要的）

知识点1【类的空间大小】

```
using namespace std;
class Data
{
private:
 //成员数据 占类的空间大小
 int num;//4B
public:

 //成员函数 不占类的空间大小
  void setNum(int data)
  {
  num = data;
  }
  int getNum(void)
  {
  return num;
  }
```

```
};
void test01()
{
 printf("%d\n",sizeof(Data));//4B
}
```

知识点2【在类内声明 类外定义 成员函数】

```
#include <iostream>

using namespace std;
class Data
{
private:
 //成员数据 占类的空间大小
 int num;//4B
public:

 //成员函数 在类内 声明 类外定义
 void setNum(int data);
 int getNum(void);
};

//Data类的成员函数
void Data::setNum(int data)
{
 num = data;
}

int Data::getNum()
{
 return num;
}

void test01()
{
 printf("%d\n",sizeof(Data));//4B

 Data ob;
 ob.setNum(100);
 cout<<"num = "<<ob.getNum()<<endl;
```

```
}
int main(int argc, char *argv[])
{
 test01();
 return 0;
}

```

知识点3【类的定义在头文件 成员函数 是在cpp文件中实现】（重要）

data.h

```
#ifndef DATA_H
#define DATA_H
class Data
{
private:
 int num;
public:
 //设置num
 void setNum(int n);
  int getNum(void);
};

#endif // DATA_H

```

data.cpp

```
#include "data.h"
void Data::setNum(int n)
{
 num = n;
}
int Data::getNum(void)
{
 return num;
}

```

main.cpp

```
#include <iostream>
#include"data.h"
```

```
using namespace std;

int main(int argc, char *argv[])
{
 Data ob;
 ob.setNum(200);
 cout<<"num = "<<ob.getNum()<<endl;
  return 0;
}

```

知识点4【构造和析构函数的概述】

构造函数和析构函数，这两个函数将会被编译器自动调用，构造函数完成对象的初始化动作，析构函数在对象结束的时候完成清理工作。

注意：对象的初始化和清理工作是编译器强制我们要做的事情，即使你不提供初始化操作和清理操作，编译器也会给你增加默认的操作，只是这个默认初始化操作不会做任何事。

构造函数：实例化对象的时候系统自动调用

析构函数：对象释放的时候系统自动调用

知识点5【构造和析构函数定义】(重要)

构造函数语法：

构造函数函数名和类名相同，没有返回类型，连void都不可以，但可以有参数，可以重载

析构函数语法：

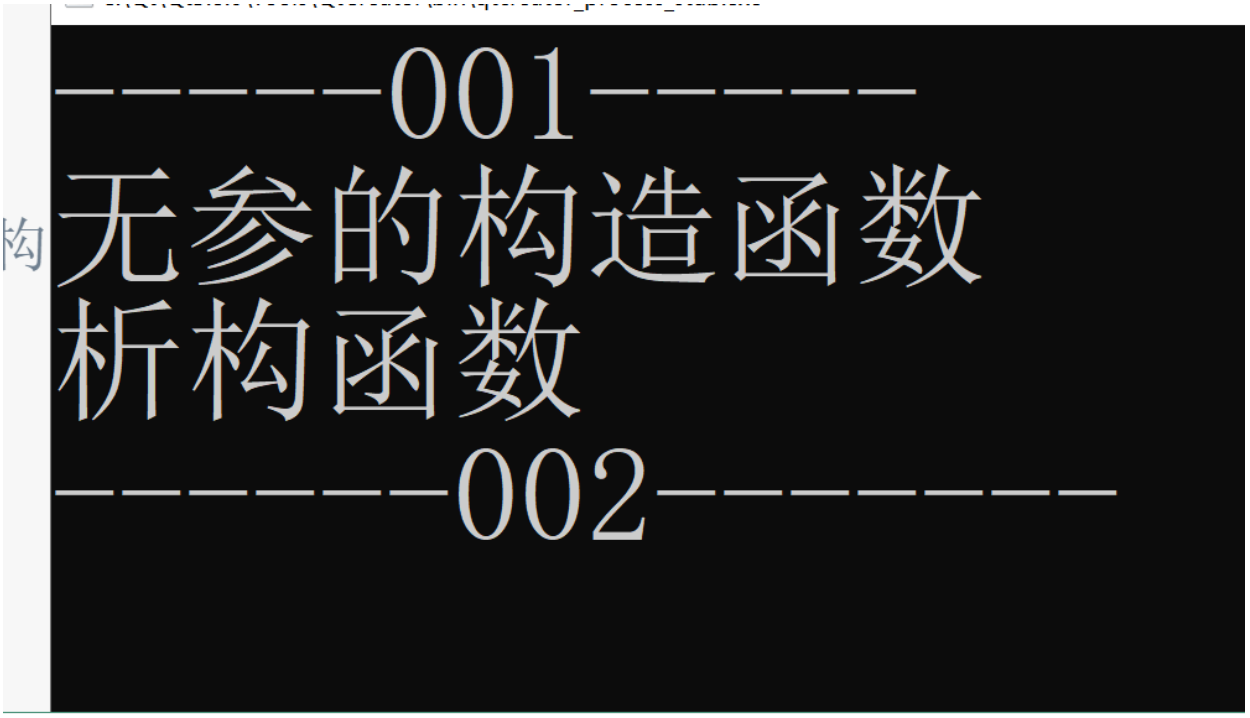
析构函数函数名是在类名前面加”~”组成,没有返回类型，连void都不可以,不能有参数，不能重载

案例：

```
class Data
{
public:
 int num;
public:
 //构造函数(无参的构造)
 Data()
 {
 num = 0;
  cout<<"无参的构造函数"<<endl;
  }
```

```
12  //构造函数（有参的构造）
13  Data(int n)
14  {
15  num = n;
16  cout<<"有参的构造函数"<<endl;
17  }
18
19  //析构函数
20  ~Data()
21  {
22  cout<<"析构函数"<<endl;
23  }
24
25 };
26 void test01()
27 {
28  //类实例化对象 系统自定调用构造函数
29  Data ob;
30
31  //函数结束的时候 局部对象ob 被释放 系统自动调用析构函数
32 }
33 int main(int argc, char *argv[])
34 {
35  cout<<"-----001-----"<<endl;
36  test01();
37  cout<<"------002-------"<<endl;
38  return 0;
39 }
```

运行结果;



知识点6【构造函数的分类以及调用】

1、构造函数分类：

按参数类型：分为无参构造函数和有参构造函数

按类型分类：普通构造函数和拷贝构造函数(复制构造函数)

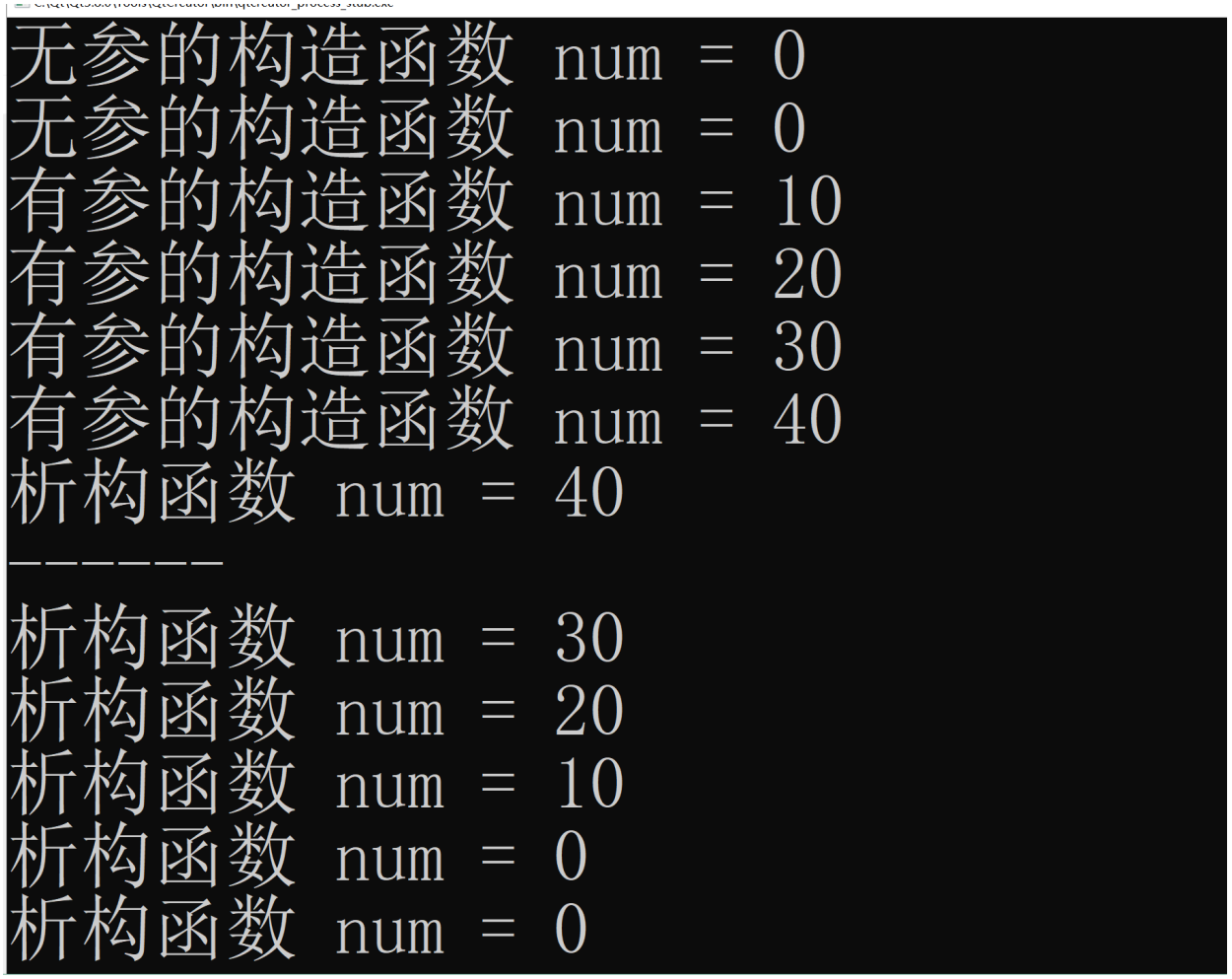
2、构造函数的调用

```
class Data
{
public:
 int num;
public:
 //构造函数(无参的构造)
 Data()
 {
 num = 0;
  cout<<"无参的构造函数 num = "<<num<<endl;
  }
  //构造函数（有参的构造）
  Data(int n)
  {
  num = n;
  cout<<"有参的构造函数 num = "<<num<<endl;
  }
```

```
18
19  //析构函数（没有返回值类型 没有参数 不能重载）
20  ~Data()
21  {
22  cout<<"析构函数 num = "<<num<<endl;
23  }
24
25 };
26 void test02()
27 {
28  //调用无参 或 默认构造 (隐式调用)
29  Data ob1;
30  //调用无参构造 （显示调用）
31  Data ob2 = Data();
32
33  //调用有参构造（隐式调用）
34  Data ob3(10);
35  //调用有参构造（显示调用）
36  Data ob4 = Data(20);
37
38  //隐式转换的方式 调用有参构造（针对于 只有一个数据成员）（尽量别用)
39  Data ob5 = 30;//转化成Data ob5(30)
40
41  //匿名对象(当前语句结束 匿名对象立即释放)
42  Data(40);
43  cout<<"------"<<endl;
44
45  //千万不要 用一下方式调用 无参构造
46
47 }
```

```
void test08()
{
    //不会 认为是实例化对象ob1 而是看成函数ob1声明
    Data ob1();
}
```

运行结果：



注意：在同一作用域 构造和析构的顺序相反

3、拷贝构造函数（系统提供一个拷贝构造函数 赋值操作）

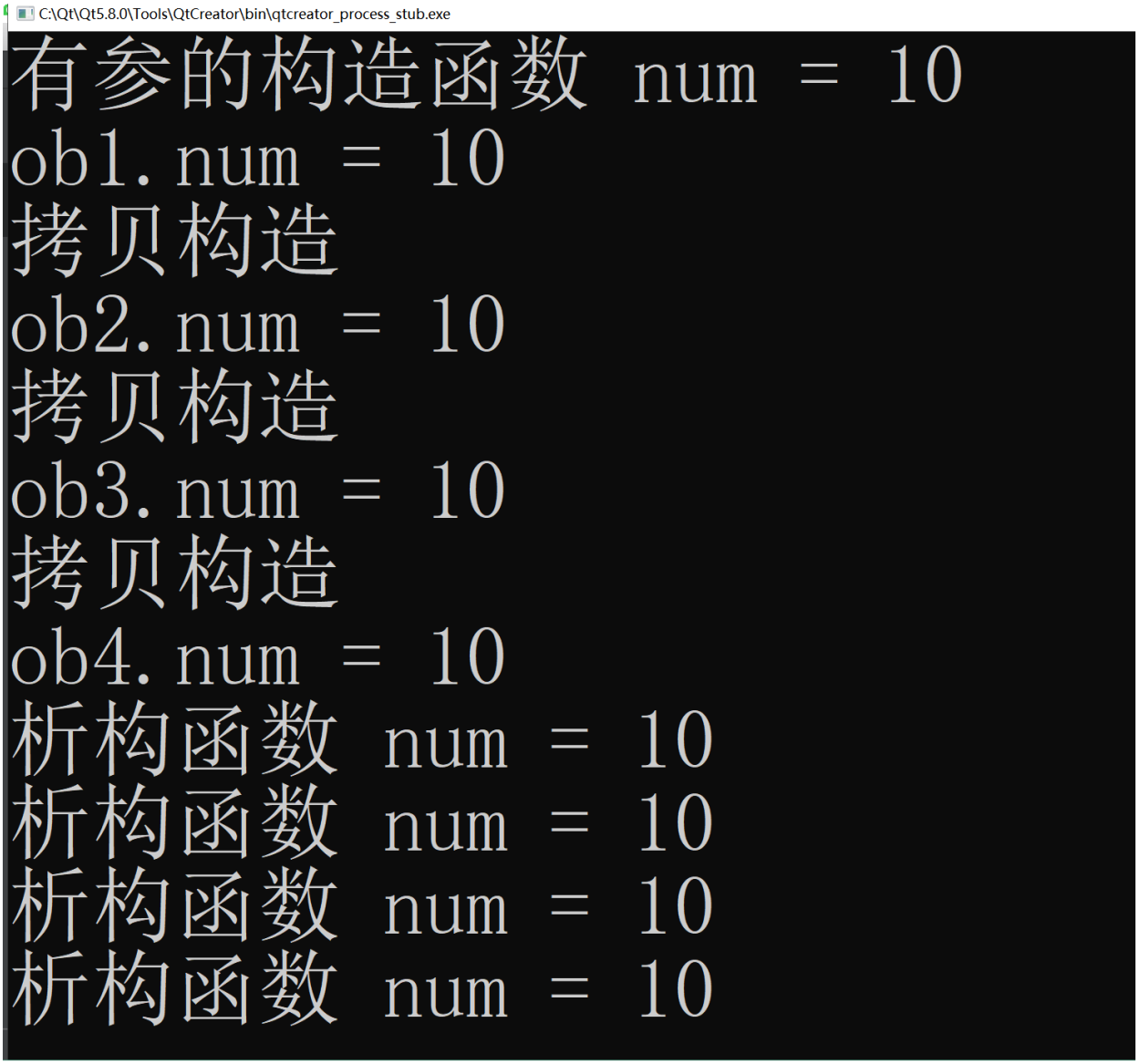
```
//拷贝构造函数
Data(const Data &ob)//const Data &ob = ob1
{
 //拷贝构造函数 是ob2调用 num就是ob2的num
 //ob2.num = ob1.num
 num = ob.num;
 cout<<"拷贝构造"<<endl;
}
```

```
void test03()
{
 Data ob1(10);
 cout<<"ob1.num = "<<ob1.num<<endl;

 //调用拷贝构造函数（如果用户 不实现拷贝构造 系统将调用默认的拷贝构造）
 //默认的拷贝构造:单纯的整体赋值（浅拷贝）
 //如果用户实现了 拷贝构造 系统将调用用户实现的拷贝构造
```

```
9
10  Data ob2(ob1);//隐式调用拷贝构造函数
11  cout<<"ob2.num = "<<ob2.num<<endl;
12
13  Data ob3 = Data(ob1);//显示调用拷贝构造函数
14  cout<<"ob3.num = "<<ob3.num<<endl;
15
16  Data ob4 = ob1;//=隐式转换调用
17  cout<<"ob4.num = "<<ob4.num<<endl;
18 }
```

运行结果：



记住一句话：旧对象 初始化 新对象 才会调用拷贝构造函数。

```
1 Data ob1(10);
```

```
2
3 Data ob2(ob1);//拷贝构造
4 Data ob3 = Data(ob1);//拷贝构造
5 Data ob4 = ob1;//拷贝构造函数
```

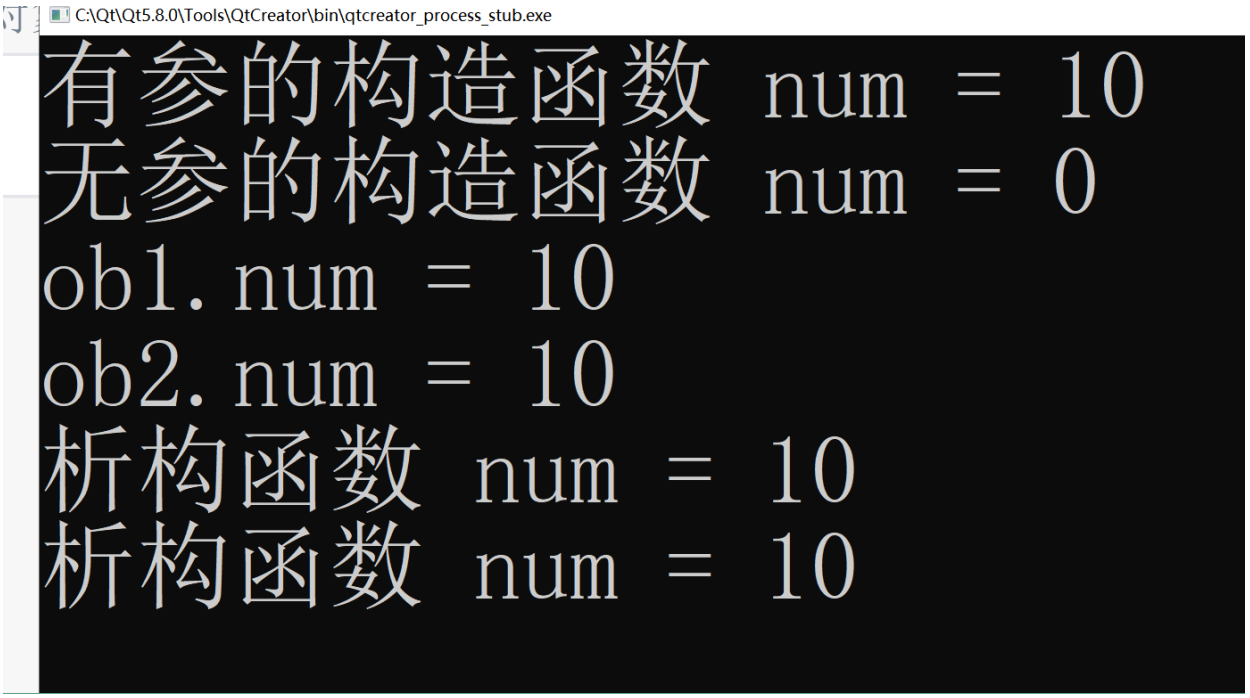
注意：下方的就不会调用拷贝构造

```
1 Data ob1(10);
2 Data ob2;
3 ob2 = ob1;//不会调用拷贝构造 单纯对象 赋值操作
```

案例：

```
1  void test04()
2  {
3   Data ob1(10);//调用有参构造
4   Data ob2;//调用无参构造
5
6   ob2 = ob1;//对象的赋值
7
8   cout<<"ob1.num = "<<ob1.num<<endl;
9   cout<<"ob2.num = "<<ob2.num<<endl;
10 }
```

运行结果：



4、拷贝构造函数的注意事项：

1、不能调用拷贝构造函数去初始化匿名对象,也就是说以下代码不正确

```
void test05()
{
    Data ob1(10);//有参构造

    //调用不了拷贝构造 错
    Data(ob1);//Data(ob1) ==> Data ob1; 造成ob1重定义
}
```

2、对象作为函数的参数 如果实参与形参 都是普通对象 那么就会调用拷贝构造

```
//函数的形参 是在函数调用的时候 开辟空间
//此处的ob 就会调用拷贝构造
void myPrintData(Data ob)//Data ob = ob1;
{
    cout<<" num = "<<ob.num<<endl;
}
void test06()
{
    Data ob1(10);

    myPrintData(ob1);
}
```

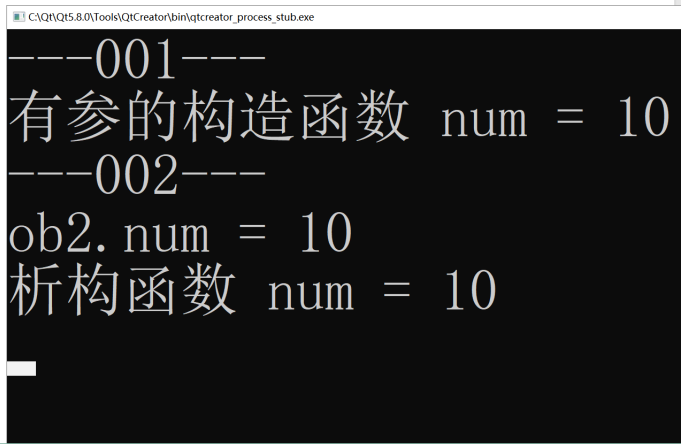
3、函数返回局部对象 在qt中会被优化 从而调用不了拷贝构造

```
Data returnData(void)
{
    cout<<"---001---"<<endl;
    Data ob1(10);//有参构造
    cout<<"---002---"<<endl;

    return ob1;
}

void test07()
{
    Data ob2 = returnData();
    cout<<"ob2.num = "<<ob2.num<<endl;

}
```



知识点7【构造函数的调用规则】（重要）

系统会对任何一个类提供3个函数成员函数：

默认构造函数（空） 默认析构函数（空） 默认拷贝构造函数（浅拷贝）

1、如果用户提供了有参构造 将屏蔽 系统的默认构造函数。

```
1 Data ob1;//err
```

2、如果用户提供了有参构造 不会屏蔽 系统的默认拷贝构造函数。

```
Data ob1(10);
Data ob2 = ob1;
ob2.num == 10
```

3、如果用户提供了拷贝构造函数 将屏蔽 系统的默认构造函数、默认拷贝构造函数。

```
Data ob1;//err
```

总结：

对于构造函数：用户一般要实现：无参构造、有参构造、拷贝构造、析构。

知识点8【深拷贝与浅拷贝】

浅拷贝：

```
class Person
{
private:
 char *m_name;
 int m_num;
public:
 Person()
 {
 m_name = NULL;
  m_num = 0;
  cout<<"无参构造"<<endl;
  }
  Person(char *name,int num)
  {
  //为m_name申请空间
  m_name = (char *)calloc(1,strlen(name)+1);
  if(m_name == NULL)
  {
  cout<<"构造失败"<<endl;
  }
  cout<<" 已经申请好空间"<<endl;
  strcpy(m_name,name);
  m_num = num;
```

```
 cout<<"有参构造"<<endl;
 }

 Person(const Person &ob)//ob==>lucy
 {
 cout<<"拷贝构造函数"<<endl;
 m_name = (char *)calloc(1, strlen(ob.m_name)+1);
 cout<<"空间已被申请"<<endl;
 strcpy(m_name, ob.m_name);

 m_num = ob.m_num;
 }

 ~Person()
 {
 if(m_name != NULL)
 {
 cout<<"空间已被释放"<<endl;
 free(m_name);
 m_name = NULL;
 }
 cout<<"析构函数"<<endl;
 }

 void showPerson(void)
 {
 cout<<"m_name = "<<m_name<<", m_num = "<<m_num<<endl;
 }
};

void test01()
{
 Person lucy("lucy",100);
 lucy.showPerson();

 //浅拷贝的问题（多次释放同一块堆区空间）
 //通过自定义 拷贝构造函数 完成深拷贝动作
 Person bob = lucy;//调用系统的默认拷贝构造（单纯的值拷贝）
}
```

如果类中的成员 指向了堆区空间 一定要记得在析构函数中 释放该空间
如果用户 不实现 拷贝构造 系统就会提供默认拷贝构造
而默认拷贝构造 只是单纯的赋值 容易造成浅拷贝问题
用户记得 要实现：无参构造（初始化数据）、有参构造（赋参数）、拷贝构造（深拷贝）、析构函数（释放空间）

知识点9【初始化列表】

注意：初始化成员列表(参数列表)只能在构造函数使用

```
class Data
{
private:
 int m_a;
 int m_b;
 int m_c;
public:
 //成员名(形参名)
 Data(int a,int b,int c):m_a(a),m_b(b),m_c(c)
 {
 //m_a = a;
 // m_b = b;
 //m_c = c;
 cout<<"有参构造"<<endl;
 }
 ~Data()
 {
 cout<<"析构函数"<<endl;
 }
 void showData(void)
 {
 cout<<m_a<<" "<<m_b<<" "<<m_c<<endl;
 }
};
void test01()
{
 Data ob(10,20,30);
 ob.showData();
}
```

运行结果：

知识点10【类的对象作为另一个类的成员】

```
class A
{
private:
 int m_a;
public:
 A()
 {
 cout<<"A无参构造函数"<<endl;
 }
  A(int a)
  {
  m_a = a;
  cout<<"A有参构造函数"<<endl;
  }
  ~A()
  {
```

```
 cout<<"A析构函数"<<endl;
 }
};
class B
{
private:
 int m_b;
public:
 B()
 {
 cout<<"B无参构造函数"<<endl;
 }
 B(int b)
 {
 m_b = b;
 cout<<"B有参构造函数"<<endl;
 }
 ~B()
 {
 cout<<"B析构函数"<<endl;
 }
};

class Data
{
private:
 A oba;//对象成员
 B obb;//对象成员
 int data;//基本类型成员
public:
 Data()
 {
 cout<<"Data无参构造"<<endl;
 }

 //初始化列表:对象名+() 显示调用 调用对象成员的构造函数
 Data(int a, int b, int c):oba(a),obb(b),data(c)
 {
 //data =c;
 cout<<"有参构造"<<endl;
```

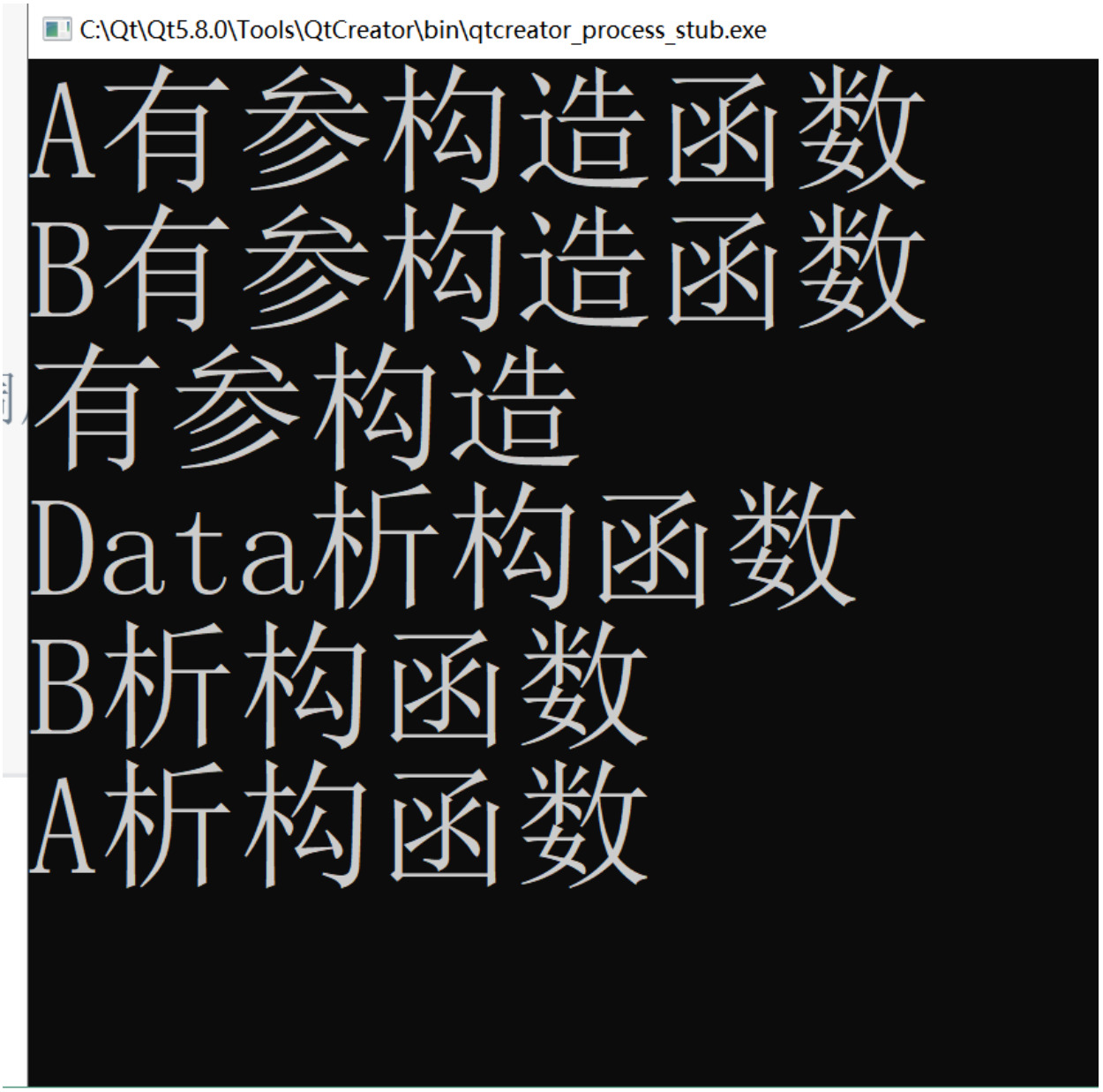
```
 }
 ~Data()
 {
 cout<<"Data析构函数"<<endl;
 }

};
void test01()
{
 //先调用 对象成员的构造-->自己的构造函数-->析构自己--->析构对象成员
 //Data ob1;

 //系统默认调用的是 对象成员的无参构造
 //必须在Data的构造函数中 使用初始化列表 使其对象成员 调用有参构造
 Data ob2(10,20,30);

}
```

运行结果;



注意：

1、按各对象成员在类定义中的顺序（和参数列表的顺序无关）依次调用它们的构造函数

2、先调用对象成员的构造函数，再调用本身的构造函数。 析构函数和构造函数调用顺序相反，先构造，后析构

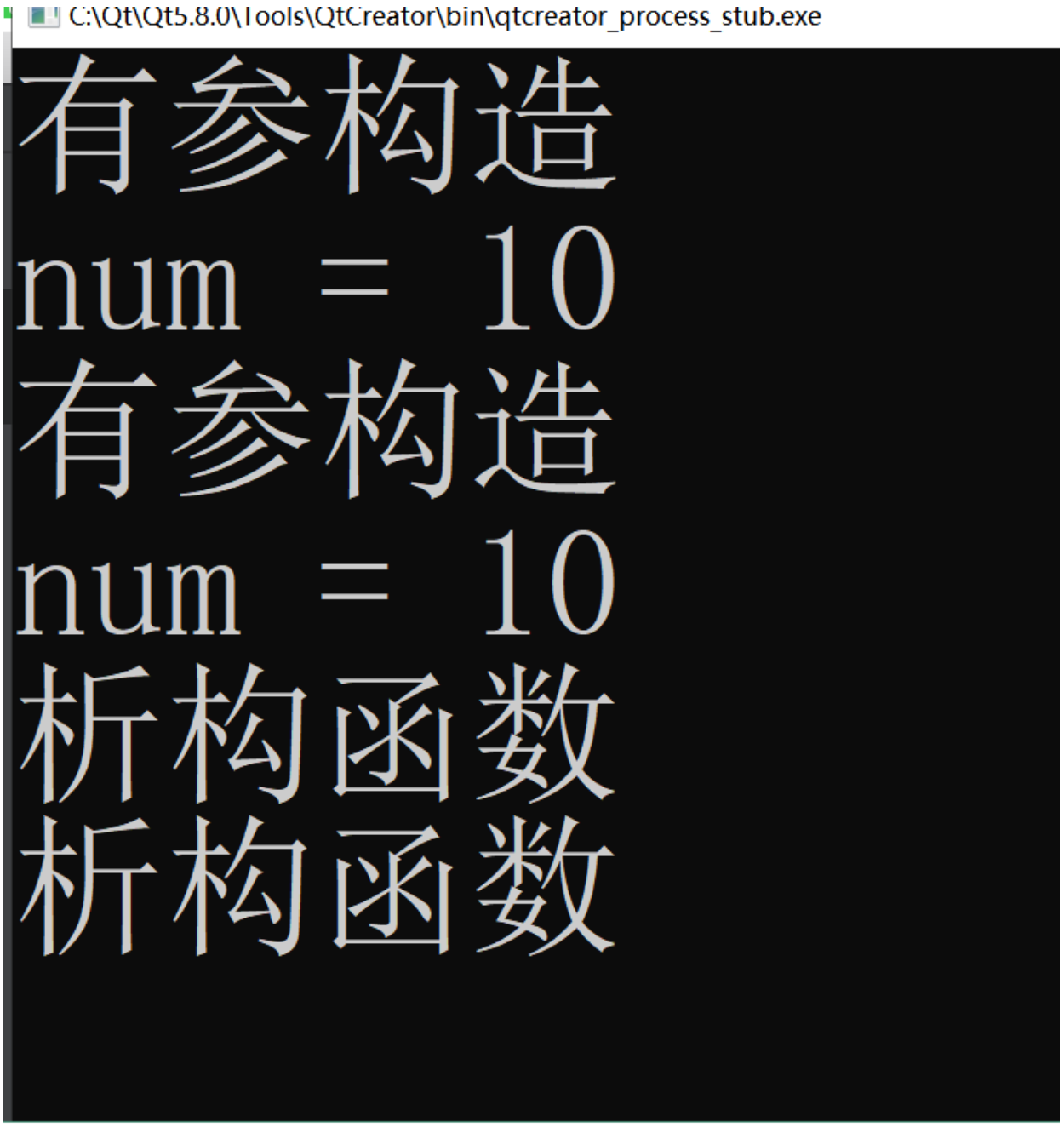
知识点11【explicit关键字】

C++提供了关键字explicit，禁止通过构造函数进行的隐式转换。声明为explicit的构造函数不能在隐式转换中使用

```
1 class Data
2 {
3 private:
4  int num;
5 public:
```

```
6  //explicit 该有参构造函数 不允许 隐式转换
7  explicit Data(int n):num(n)
8  {
9  cout<<"有参构造"<<endl;
10  }
11  ~Data()
12  {
13  cout<<"析构函数"<<endl;
14  }
15  void showNum(void)
16  {
17  cout<<"num = "<<num<<endl;
18  }
19 };
20 int main(int argc, char *argv[])
21 {
22  //隐私转换
23  //Data data = 10;//explicit 该有参构造函数 不允许 隐式转换
24  Data data(10);//ok
25  data.showNum();
26
27  Data ob=Data(10);//ok
28  ob.showNum();
29  return 0;
30 }
```

运行结果：



知识点12【new和delete】从堆区申请空间

1、和malloc calloc realloc比较

1.1、malloc返回一个void指针，c++不允许将void赋值给其他任何指针，必须强转。

1.2、malloc可能申请内存失败，所以必须判断返回值来确保内存分配成功。

1.3、malloc 不会调用构造函数。free 不会调用析构函数（重要的）