[**C++ 带有指针成员的类处理方式**](https://www.cnblogs.com/lucy-lizhi/p/6551308.html)

在一个类中，如果类没有指针成员，一切方便，因为默认合成的析构函数会自动处理所有的内存。但是如果一个类带了指针成员，那么需要我们自己来写一个析构函数来管理内存。在<<c++ primer>> 中写到，如果一个类需要我们自己写析构函数，那么这个类，也会需要我们自己写拷贝构造函数和拷贝赋值函数。

**析构函数：**

我们这里定义一个类HasPtr，这个类中包含一个int 类型的指针。然后定义一个析构函数，这个函数打印一句话。

HasPtr.h 类的头文件

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 #pragma once

2 #ifndef \_\_HASPTR\_\_

3 #define \_\_HASPTR\_\_

4

5 class HasPtr

6 {

7 public:

8 HasPtr(int i,int \*p);

9 //HasPtr& operator=(HasPtr&);

10 //HasPtr(const HasPtr&);

11 ~HasPtr();

12 int get\_ptr\_value();

13 void set\_ptr\_value(int \*p);

14 int get\_val();

15 void set\_val(int v);

16 private:

17 int val;

18 int \*ptr;

19 };

20

21 #endif // !\_\_HASPTR\_\_

[复制代码](javascript:void(0);)

HasPtr.cpp 类的实现

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 #include "stdafx.h"

2

3 #include <iostream>

4 #include "HasPtr.h"

5

6 using namespace std;

7

8 HasPtr::HasPtr(int i, int \*p)

9 {

10 val = i;

11 ptr = p;

12 }

13

14 int HasPtr::get\_ptr\_value()

15 {

16 return \*ptr;

17 }

18

19 void HasPtr::set\_ptr\_value(int \*p)

20 {

21 ptr = p;

22 }

23

24 int HasPtr::get\_val()

25 {

26 return val;

27 }

28

29 void HasPtr::set\_val(int v)

30 {

31 val = v;

32 }

33

34 HasPtr::~HasPtr()

35 {

36 cout << "destructor of HasPtr " << endl;

37 }

[复制代码](javascript:void(0);)

ClassWithPointer 类，包含main入口，HasPtr在stack上。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 // ClassWithPointer.cpp : 定义控制台应用程序的入口点。

2 //

3

4 #include "stdafx.h"

5 #include <iostream>

6 #include "HasPtr.h"

7 using namespace std;

8

9 int main()

10 {

11 int temp = 100;

12 HasPtr ptr(2,&temp);

13 cout << ptr.get\_ptr\_value() << endl;

14 cout << ptr.get\_val() << endl;

15 system("PAUSE");

16 system("PAUSE");

17 return 0;

18 }

[复制代码](javascript:void(0);)

执行该入口方法，发现最后还是打印了析构函数这句话，OK，在main 方法中，stack上定义了一个HasPtr，在main方法退出前，析构函数自动调用了。

如果将HasPtr改为动态对象，也就是放在堆上呢？

ClassWithPointer 类，包含main入口，HasPtr在heap上。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 // ClassWithPointer.cpp : 定义控制台应用程序的入口点。

2 //

3

4 #include "stdafx.h"

5 #include <iostream>

6 #include "HasPtr.h"

7 using namespace std;

8

9 int main()

10 {

11 int temp = 100;

12 //HasPtr ptr(2,&temp);

13 HasPtr \*ptr = new HasPtr(2,&temp);

14 cout << ptr->get\_ptr\_value() << endl;

15 cout << ptr->get\_val() << endl;

16 system("PAUSE");

17 return 0;

18 }

[复制代码](javascript:void(0);)

执行一下，发现析构函数没有调用。OK，我们在return 0前面添加一个delete ptr; 析构函数执行了。

所以，这里有两个结论：

1. 当一个对象在stack 上时，析构函数自动调用。
2. 当一个对象在heap上时，需要调用delete 语句，析构函数才会被执行。

现在在析构函数中调用delete 语句来删除指针成员。

头文件不变，HasPtr.cpp 文件代码如下：

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 #include "stdafx.h"

2

3 #include <iostream>

4 #include "HasPtr.h"

5

6 using namespace std;

7

8 HasPtr::HasPtr(int i, int \*p)

9 {

10 val = i;

11 ptr = p;

12 }

13

14 int HasPtr::get\_ptr\_value()

15 {

16 return \*ptr;

17 }

18

19 void HasPtr::set\_ptr\_value(int \*p)

20 {

21 ptr = p;

22 }

23

24 int HasPtr::get\_val()

25 {

26 return val;

27 }

28

29 void HasPtr::set\_val(int v)

30 {

31 val = v;

32 }

33

34 HasPtr::~HasPtr()

35 {

36 cout << "destructor of HasPtr " << endl;

37 delete ptr;

38 }

[复制代码](javascript:void(0);)

 ClassWithPointer 代码如下：

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 // ClassWithPointer.cpp : 定义控制台应用程序的入口点。

2 //

3

4 #include "stdafx.h"

5 #include <iostream>

6 #include "HasPtr.h"

7 using namespace std;

8

9 int main()

10 {

11 int temp = 100;

12 HasPtr ptr(2,&temp);

13 cout << ptr.get\_ptr\_value() << endl;

14 cout << ptr.get\_val() << endl;

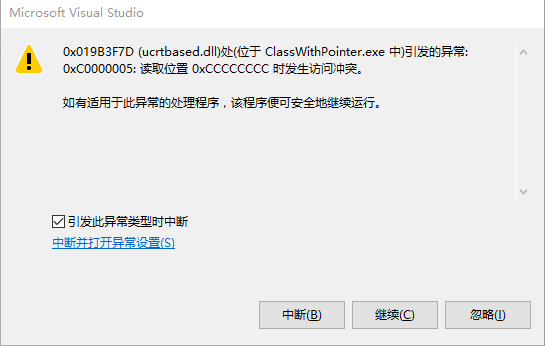
15 system("PAUSE");

16 return 0;

17 }

[复制代码](javascript:void(0);)

执行一下，正常打印结束后，抛出错误：



这里说明delete 不能删除stack 上的指针值。

现在在ClassWithPointer传入一个动态指针来测试一下。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 // ClassWithPointer.cpp : 定义控制台应用程序的入口点。

2 //

3

4 #include "stdafx.h"

5 #include <iostream>

6 #include "HasPtr.h"

7 using namespace std;

8

9 int main()

10 {

11 int \*temp = new int(100);

12 HasPtr ptr(2,temp);

13 cout << ptr.get\_ptr\_value() << endl;

14 cout << ptr.get\_val() << endl;

15 system("PAUSE");

16 return 0;

17 }

[复制代码](javascript:void(0);)

执行后析构函数正常运行。所以这里有两个结论：

1. delete 语句不能删除stack 上的指针值。
2. delete 语句只能删除heap上的指针值，也就是new 出来的对象。

默认拷贝构造函数和默认赋值操作：

这里我们调用默认的构造函数和默认的赋值操作，看看会出现什么，为了方便查看，我在析构函数中打印了当前对象的地址，以及在main方法中打印了对象地址，这样就可以看到哪个对象调用了析构函数：

HasPtr.cpp 代码如下：

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 #include "stdafx.h"

2

3 #include <iostream>

4 #include "HasPtr.h"

5

6 using namespace std;

7

8 HasPtr::HasPtr(int i, int \*p)

9 {

10 val = i;

11 ptr = p;

12 }

13

14 int HasPtr::get\_ptr\_value()

15 {

16 return \*ptr;

17 }

18

19 void HasPtr::set\_ptr\_value(int \*p)

20 {

21 ptr = p;

22 }

23

24 int HasPtr::get\_val()

25 {

26 return val;

27 }

28

29 void HasPtr::set\_val(int v)

30 {

31 val = v;

32 }

33

34 HasPtr::~HasPtr()

35 {

36 cout << "destructor of HasPtr " << this << endl;

37 delete ptr;

38 }

[复制代码](javascript:void(0);)

ClassWithPointer 代码如下：

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 // ClassWithPointer.cpp : 定义控制台应用程序的入口点。

2 //

3

4 #include "stdafx.h"

5 #include <iostream>

6 #include "HasPtr.h"

7 using namespace std;

8

9 int main()

10 {

11 int \*temp = new int(100);

12 HasPtr ptr(2,temp);

13 cout << "ptr-------------->" << &ptr << endl;

14 cout << ptr.get\_ptr\_value() << endl;

15 cout << ptr.get\_val() << endl;

16

17 HasPtr ptr2(ptr);

18 cout << "ptr2-------------->" << &ptr2 << endl;

19 cout << ptr2.get\_ptr\_value() << endl;

20 cout << ptr2.get\_val() << endl;

21

22 HasPtr ptr3 = ptr;

23 cout << "ptr3-------------->" << &ptr3 << endl;

24 cout << ptr3.get\_ptr\_value() << endl;

25 cout << ptr3.get\_val() << endl;

26

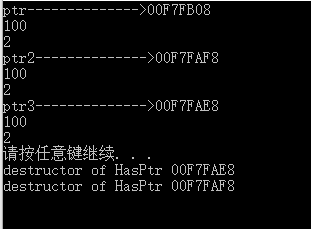
27 system("PAUSE");

28 return 0;

29 }

[复制代码](javascript:void(0);)

运行结果如下，最后还是报错了：



其实程序运行到第二个析构函数时，报错了。报错原因是，ptr 其实已经是pending指针了，因为这个ptr 指针所指向的地址已经被delete了。

不过我们这里最起码可以知道默认的拷贝构造函数和赋值操作，也是会直接复制指针值的，不是指针所指向的值。是指针变量的值，也就是地址。

所以这里引申出来的问题是：如何管理对象中指针成员的内存？ 这个是一个核心问题。

上面的例子，就是默认的方式，但是管理失败了，因为析构函数到最后会删除pending 指针，导致异常发生。

智能指针：

引入一个类U\_Ptr，用来管理我们需要在业务对象中需要的指针变量，假设为int \*p。头文件如下：

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 #pragma once

2 #ifndef \_\_UPTR\_\_

3 #define \_\_UPTR\_\_

4 #include "HasPtr.h"

5 #include <iostream>

6

7 using namespace std;

8 class U\_Ptr

9 {

10 friend class HasPtr;

11 int \*ip;

12 size\_t use;

13

14 U\_Ptr(int \*p):ip(p),use(1) {}

15 ~U\_Ptr()

16 {

17 cout << "destruction:"<< \*ip << endl;

18 delete ip;

19 }

20 };

21 #endif // !\_\_UPTR\_\_

[复制代码](javascript:void(0);)

现在我们的业务对象还是HasPtr。头文件如下：

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 #pragma once

2 #ifndef \_\_HASPTR\_\_

3 #define \_\_HASPTR\_\_

4 #include "U\_Ptr.h"

5 class HasPtr

6 {

7 public:

8 HasPtr(int \*p, int i):ptr(new U\_Ptr(p)),val(i){}

9

10 HasPtr(const HasPtr &orgi) :ptr(orgi.ptr), val(orgi.val)

11 {

12 ++ptr->use;

13 cout << "coming into copy construction:" << ptr->use << endl;

14 }

15

16 HasPtr& operator=(const HasPtr &rhs);

17

18 ~HasPtr();

19

20 int get\_ptr\_value() const;

21 int get\_int() const;

22 void set\_ptr(int \*p);

23 void set\_int(int i);

24 private:

25 U\_Ptr \*ptr;

26 int val;

27 };

28

29 #endif // !\_\_HASPTR\_\_

[复制代码](javascript:void(0);)

HasPtr.cpp 实现如下：

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 #include "stdafx.h"

2 #include "HasPtr.h"

3 #include <iostream>

4

5 using namespace std;

6

7 HasPtr& HasPtr::operator=(const HasPtr &rhs)

8 {

9 ++rhs.ptr->use;

10 if (--ptr->use == 0)

11 {

12 delete ptr;

13 }

14 ptr = rhs.ptr;

15 val = rhs.val;

16 return \*this;

17 }

18

19 HasPtr::~HasPtr()

20 {

21 cout << "destruction:" << ptr->use << endl;

22 if (--ptr->use == 0)

23 {

24 delete ptr;

25 }

26 }

27

28 int HasPtr::get\_ptr\_value() const

29 {

30 return \*ptr->ip;

31 }

32 int HasPtr::get\_int() const

33 {

34 return val;

35 }

36 void HasPtr::set\_ptr(int \*p)

37 {

38 ptr->ip = p;

39 }

40 void HasPtr::set\_int(int i)

41 {

42 val = i;

43 }

[复制代码](javascript:void(0);)

测试类如下：

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 // SmartPointer.cpp : 定义控制台应用程序的入口点。

2 //

3

4 #include "stdafx.h"

5 #include "HasPtr.h"

6 #include <iostream>

7

8 using namespace std;

9

10

11 int main()

12 {

13 int \*temp = new int(100);

14 HasPtr ptr(temp,22);

15 cout << "ptr------------>" << endl;

16 cout << ptr.get\_ptr\_value() << endl;

17 cout << ptr.get\_int() << endl;

18 HasPtr ptr2(ptr);

19 cout << "ptr2------------>" << endl;

20 cout << ptr2.get\_ptr\_value() << endl;

21 cout << ptr2.get\_int() << endl;

22 system("PAUSE");

23 return 0;

24 }

[复制代码](javascript:void(0);)

我们把U\_Ptr 叫做智能指针，用于帮我们管理需要的指针成员。我们的业务对象HasPtr对象包含一个智能指针，这个指针在HasPtr 对象创建时创建，智能指针的use 变量用来记录业务对象HasPtr对象被复制了多少次，也就是说，有多少个相同的指针指向了ptr所指向的地方。如果要记录HasPtr对象一共有多少个一样的，那么就需要在拷贝构造函数和赋值操作处进行对use变量加一操作，在析构函数处进行减一操作。当减到0时，删除指针。