

2020春季学期C++整理5.0

一. this 指针

1. 引入

从c语言到c++语言得翻译：

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  class Cchar{
4      public:
5          int price;
6          void Setprice(int p);
7  };
8  void Cchar::Setprice(int p)
9  {
10     price = p;
11 }
12 int main()
13 {
14     Cchar car;
15     car.Setprice(200000);
16     return 0;
17 }
```

上述这段代码用c语言描述就变成了：

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  struct Cchar{
4      int price;
5      void Setprice(int p);
6  };
7  void Setprice(struct Cchar * this, int p)//成元函数这边我们使用全局函数代替
8  {
9      this->price = p;
10 }
11 int main()
12 {
13     struct Cchar car;
14     Setprice(&car, 200000);
15     return 0;
16 }
```

2. this 指针的作用

一句话概括就是：指向成员函数所作用的**对象**

- **非静态的成员函数**中可以直接使用 `this` 来代表指向该函数作用的对象指针。

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  class Cchar{
4      public:
5          int price;
6          void Print(){
7              cout << price << endl;
8          }
9          Cchar(int price){    }
10         Cchar Addone(){
11             this->price++;
12             this->Print();
13             return *this;
14         }
15     };
16     int main()
17     {
18         Cchar car1(2),car2(1);
19         car2 = car1.Addone(); //Addone函数中的this指针其实就是指向的对象car1，于是就可
//访问到car1的变量
20         return 0;
21     }
22     //输出3

```

- 一种特殊情况:

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  class A{
4      int i;
5      public:
6          void Hello(){
7              cout << "hello " << endl;
8          }
9  };
10 int main()
11 {
12     A *p = NULL;
13     p->Hello();
14     return 0;
15 }
16 //结果输出"hello "

```

所以应该怎么解释呢?? 明明我们定义了p指针为 NULL, 为什么还可以正确执行成员函数?

我们翻译成c语言的形式来描述:

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  class A{
4      int i;
5  };
6  void Hello(A * this)
7  {
8      cout << "hello " << endl;
9  }
10 int main()

```

```

11 {
12     A *p = NULL;
13     Hello(p);
14     return 0;
15 }

```

也就是说，`this` 尽管是一个空指针，但是不影响Hello函数的进行。

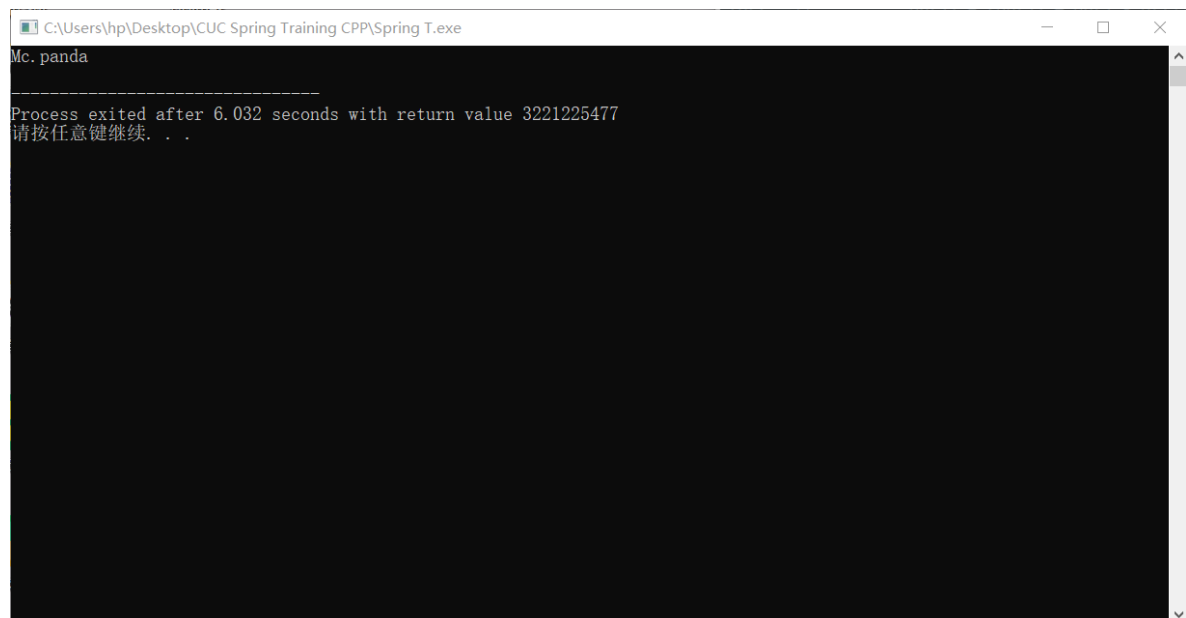
但是如果Hello函数写成这个样子：

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  class A{
4      int i;
5      public:
6          void Hello(){
7              cout << "Mc.panda" << endl; //输出中间变量验证
8              cout << "hello " << i << endl;
9          }
10 };
11 int main()
12 {
13     A *p = NULL;
14     p->Hello();
15     return 0;
16 }

```

此时就会出现错误,输出的情况见下：



```

C:\Users\hjp\Desktop\CUC Spring Training CPP\Spring T.exe
Mc. panda
-----
Process exited after 6.032 seconds with return value 3221225477
请按任意键继续. . .

```

注意：

类的非静态成员函数，其**真实的参数比所写的总是多1**（`this` 指针）

自动窗口		
搜索(Ctrl+E) 🔍 搜索深度: 3 🔼 🔽		
名称	值	类型
price	1	int
this	0x010ffb5c {price=-858993460 }	Cchar *
price	-858993460	int

上图很好得说明了这一问题

二.静态数据成员

我们在学习C语言时已经知道：全局变量可以实现数据共享。但是考虑到全局变量的不稳定性，一般我们不采用全局变量，而是选用类中的静态数据成员。

静态数据成员是一种特殊的数据成员。它以关键字 `static` 开头。比如：

```

1  class Box{
2      public:
3          int volume();
4      private:
5          static int height; //我们把height定义为静态数据成员
6          int width,length;
7  };

```

如果希望各对象中的数据成员的值是一样的，就可以把它定义为静态数据成员，这样他就为各个对象所共有，而不只是属于某个对象的成员，所有的对象都可以引用他。静态的数据成员在内存中只占一份存储空间，**而不是每一个对象都分别为他保留一份空间**。每个对象都可以引用这个静态数据成员。静态数据成员的值对所有对象都是一致得。如果改变它的值，那就会导致各个对象中这个数据成员的值都改变了。这样做可以节约空间，提高效率。

说明：

- 如果只声明了类但是没有定义对象，则类的一般数据成员是不占内存空间的，只有在定义对象时才为对象的数据成员分配空间。但静态数据成员不属于某一个对象，**在为对象所分配的空间中不包括静态数据成员所占的空间**。静态数据成员是在所有对象之外单独开辟空间。**只要在类中指定了静态数据成员，即使不定义对象，系统也会为静态数据成员开辟空间。**
- 静态数据成员实在程序编译时被分配空间的，直到程序结束时才释放空间。
- 静态成员可以进行初始化，但是必须在类体外进行。如：

```
1  int Box :: height = 0;
```

其一般形式：

数据类型 类名 :: 静态数据成员名 = 初值

注意：不能用参数初始化表对静态成员变量初始化，比如：

```
1  Box(int h,int w,int len) : height(h){} //Error!!!!
```

如果没有对静态数据成员初始化，**系统将默认为0**

- 静态数据成员**既可以通过对象名引用，也可以通过类名引用。**

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  class Box{
4      public:
5          Box(int ,int);
6          int volume();
7          static int height;
8          int width;
9          int length;
10 };
11 Box::Box(int w,int len)
12 {
13     width = w;
14     length = len;
15 }
16 int Box::volume()
17 {
18     return height * width * length;
19 }
20 int Box::height = 0; //初始化
21 int main()
22 {
23     Box a(15,20),b(20,30);
24     cout << a.height << endl; //通过对象名引用静态数据成员
25     cout << b.height << endl;
26     cout << Box::height << endl; //通过类名引用
27     cout << a.volume() << endl;
28     return 0;
29 }

```

注：上述代码说明：静态数据成员不属于某一个对象，而是属于类的，但类的对象可以引用他

三.静态成员函数

成员函数也可以是静态的，比如：

```
1 | static int volume()
```

和静态数据成员一样，其是类的一部分而不是某个对象的一部分。在类外调用要使用类名和域运算符 `::`

```
1 | Box :: volume();
```

其实也可以用对象名调用：

```
1 | a.volume();
```

前面我们指出：调用一个非静态成员函数时，系统会把该对象的起始地址赋给成员函数的 `this` 指针。而静态函数则不同，**静态成员函数没有 `this` 指针**。无法对一个对象中的非静态成员进行默认访问。

可以这么说：静态成员函数和非静态成员函数之间的根本区别在于：**是否有 `this` 指针**。由此决定了静态成员函数不能访问本类中的非静态成员。

静态成员函数可以直接引用本类中的静态成员，因为静态成员同样是属于类的，可以直接引用。在C++程序中，静态成员函数用来访问静态数据成员，而不访问非静态数据成员。加入在一个静态成员函数中有这样的语句：

```
1 cout << height << endl; //Correct
2 cout << width << endl; //Error
```

当然，**不能默认访问不是不能访问**，如果一定要访问的话，需要用对象名引用

```
1 cout << a.width << endl;
```

当然你只要不嫌混乱

举个例子：

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 class Box{
4     public:
5         Box(int ,int);
6         int width;
7         int length;
8         static int height;
9         static int volume();
10 };
11 Box::Box(int w,int len)
12 {
13     width = w;
14     length = len;
15 }
16 int Box::volume()
17 {
18     return height;
19 }
20 int Box::height = 10;
21 int main()
22 {
23     Box a(15,20),b(20,30);
24     cout << a.height << endl;
25     cout << b.height << endl;
26     cout << Box::height << endl; //类名调用静态数据成员函数
27     cout << a.volume() << endl; //对象调用静态成员函数
28     cout << Box::volume() << endl; //类名调用静态成员函数
29     return 0;
30 }
```