C/C++ 中的static关键字

Arkin，Mechanic . Keep coding.

小目录：

1.（面向对象的）静态成员变量

在类内成员变量的声明前加上关键字static，该数据成员就是类内的静态数据成员。

```
//Example 5
#include <iostream.h>
class Myclass
{
public:
        Myclass(int a,int b,int c);
        void GetSum();
private:
        int a,b,c;
        static int Sum;//声明静态数据成员
};

int Myclass::Sum=0;     //定义并初始化静态数据成员

Myclass::Myclass(int a,int b,int c)
{
        this->a=a;
        this->b=b;
        this->c=c;
        Sum+=a+b+c;
}

void Myclass::GetSum()
{
        cout<<"Sum="<<Sum<<endl;
}

void main()
{
        Myclass M(1,2,3);
        M.GetSum();
        Myclass N(4,5,6);
        N.GetSum();
        M.GetSum();
}
```

静态成员变量有以下特点：

1. 静态成员变量是该类的所有对象所共有的。对于普通成员变量，每个类对象都有自己的一份拷

次内存，由该类的所有对象共享访问。所以，静态数据成员的值对每个对象都是一样的，它的值可以更新；

2. 因为静态数据成员在全局数据区分配内存，由本类的所有对象共享，所以，它不属于特定的类对象，不占用对象的内存，而是在所有对象之外开辟内存，在没有产生类对象时其作用域就可见。因此，在没有类的实例存在时，静态成员变量就已经存在，我们就可以操作它；
3. 静态成员变量存储在全局数据区。static 成员变量的内存空间既不是在声明类时分配，也不是在创建对象时分配，而是在初始化时分配。静态成员变量必须初始化，而且只能在类体外进行。否则，编译能通过，链接不能通过。在Example 5中，语句int Myclass::Sum=0;是定义并初始化静态成员变量。初始化时可以赋初值，也可以不赋值。如果不赋值，那么会被默认初始化，一般是 0。静态数据区的变量都有默认的初始值，而动态数据区（堆区、栈区）的变量默认是垃圾值。
4. static 成员变量和普通 static 变量一样，编译时在静态数据区分配内存，到程序结束时才释放。这就意味着，static 成员变量不随对象的创建而分配内存，也不随对象的销毁而释放内存。而普通成员变量在对象创建时分配内存，在对象销毁时释放内存。
5. 静态数据成员初始化与一般数据成员初始化不同。初始化时可以不加 static，但必须要有数据类型。被 private、protected、public 修饰的 static 成员变量都可以用这种方式初始化。静态数据成员初始化的格式为：＜数据类型＞＜类名＞::＜静态数据成员名＞=＜值＞
6. 类的静态成员变量访问形式1：＜类对象名＞.＜静态数据成员名＞
7. 类的静态成员变量访问形式2：＜类类型名＞::＜静态数据成员名＞，也即，静态成员不需要通过对象就能访问。
8. 静态数据成员和普通数据成员一样遵从public,protected,private访问规则；
9. 如果静态数据成员的访问权限允许的话（即public的成员），可在程序中，按上述格式来引用静态数据成员；
10. sizeof 运算符不会计算 静态成员变量。

```
class CMyclass{
    int n;
    static int s;
};    //则sizeof（CMyclass）等于4
```

何时采用静态数据成员?

设置静态成员（变量和函数）这种机制的目的是将某些和类紧密相关的全局变量和函数写到类里面，看上去像一个整体，易于理解和维护。如果想在同类的多个对象之间实现数据共享，又不要用全局变量，那么就可以使用静态成员变量。也即，静态数据成员主要用在各个对象都有相同的某项属性的时候。比如对于一个存款类，每个实例的利息都是相同的。所以，应该把利息设为存款类的静态数据成员。这有两个好处：

1. 不管定义多少个存款类对象，利息数据成员都共享分配在全局数据区的内存，节省存储空间。
2. 一旦利息需要改变时，只要改变一次，则所有存款类对象的利息全改变过来了。

你也许会问，用全局变量不是也可以达到这个效果吗?

同全局变量相比，使用静态数据成员有两个优势：

1. 静态成员变量没有进入程序的全局命名空间，因此不存在与程序中其它全局命名冲突的可能。
2. 可以实现信息隐藏。静态成员变量可以是private成员，而全局变量不能。

2.（面向对象的） 静态成员函数

与静态成员变量类似，我们也可以声明一个静态成员函数。

静态成员函数为类服务而不是为某一个类的具体对象服务。静态成员函数与静态成员变量一样，都是类的内部实现，属于类定义的一部分。普通成员函数必须具体作用于某个对象，而静态成员函数并不具体作用于某个对象。

普通的成员函数一般都隐含了一个this指针，this指针指向类的对象本身，因为普通成员函数总是具体地属于类的某个具体对象的。当函数被调用时，系统会把当前对象的起始地址赋给 this 指针。通常情况下，this是缺省的。如函数fn()实际上是this->fn()。

用其余的静态成员函数和静态成员变量。从另一个角度来看，由于静态成员函数和静态成员变量在类实例化之前就已经存在可以访问，而此时非静态成员还是不存在的，因此静态成员不能访问非静态成员。

```
//Example 6
#include <iostream>
using namespace std;

class Student{
private:
   char *name;
   int age;
   float score;
   static int num;      //学生人数
   static float total;  //总分
public:
   Student(char *, int, float);
   void say();
   static float getAverage();  //静态成员函数，用来获得平均成绩
};

int Student::num = 0;
float Student::total = 0;

Student::Student(char *name, int age, float score)
{
   this->name = name;
   this->age = age;
   this->score = score;
   num++;
   total += score;
}

void Student::say()
{
   cout<<name<<"的年龄是 "<<age<<"，成绩是 "<<score<<"（当前共"<<num<<"名学生）"<<end
}

float Student::getAverage()
{
   return total / num;
}

int main()
{
   (new Student("小明", 15, 90))->say();
   (new Student("李磊", 16, 80))->say();
   (new Student("张华", 16, 99))->say();
   (new Student("王康", 14, 60))->say();
   cout<<"平均成绩为 "<<Student::getAverage()<<endl;
   return 0;
}

运行结果：
小明的年龄是 15，成绩是 90（当前共1名学生）
李磊的年龄是 16，成绩是 80（当前共2名学生）
张华的年龄是 16，成绩是 99（当前共3名学生）
王康的年龄是 14，成绩是 60（当前共4名学生）
平均成绩为 82.25
```

静态成员函数的特点：

1. 出现在类体外的函数定义不能指定关键字static；
2. 静态成员之间可以相互访问，即静态成员函数（仅）可以访问静态成员变量、静态成员函数；
3. 静态成员函数不能访问非静态成员函数和非静态成员变量；
4. 非静态成员函数可以任意地访问静态成员函数和静态数据成员；
5. 由于没有this指针的额外开销，静态成员函数与类的全局函数相比速度上会稍快；

- 通过成员访问操作符(.)和(->)，也即通过类对象或指向类对象的指针调用静态成员函数。
- 直接通过类来调用静态成员函数。＜类名＞::＜静态成员函数名＞（＜参数表＞）。也即，静态成员不需要通过对象就能访问。

拷贝构造函数的问题

在使用包含静态成员的类时，有时候会调用拷贝构造函数生成临时的隐藏的类对象，而这个临时对象在消亡时会调用析构函数有可能会对静态变量做操作（例如total_num--），可是这些对象在生成时却没有执行构造函数中的total_num++的操作。解决方案是为这个类写一个拷贝构造函数，在该拷贝构造函数中完成total_num++的操作。

3. （面向过程的）静态全局变量

在全局变量前，加上关键字static，该变量就被定义成为一个静态全局变量。

```
//Example 1
#include <iostream.h>

void fn();
static int n; //定义静态全局变量

void main()
{
        n=20;
        cout<<n<<endl;
        fn();
}

void fn()
{
        n++;
        cout<<n<<endl;
}
```

静态全局变量有以下特点：

1. 该变量在全局数据区分配内存；
2. 未经初始化的静态全局变量会被程序自动初始化为0（自动变量的自动初始化值是随机的）；
3. 静态全局变量在声明它的整个文件都是可见的，而在文件之外是不可见的；
4. 静态变量都在全局数据区分配内存，包括后面将要提到的静态局部变量。对于一个完整的程序，在内存中的分布情况如下：【代码区】【全局数据区】【堆区】【栈区】，一般程序的由new产生的动态数据存放在堆区，函数内部的自动变量存放在栈区，静态数据（即使是函数内部的静态局部变量）存放在全局数据区。自动变量一般会随着函数的退出而释放空间，而全局数据区的数据并不会因为函数的退出而释放空间。

Example 1中的代码中将

```
static int n; //定义静态全局变量
```

改为

```
int n; //定义全局变量
```

程序照样正常运行。
定义全局变量就可以实现变量在文件中的共享，但定义静态全局变量还有以下好处：

1. 静态全局变量不能被其它文件所用；
2. 其它文件中可以定义相同名字的变量，不会发生冲突；

将上述示例代码改为如下：

```
//Example 2
//File1
#include <iostream.h>

void fn();
static int n; //定义静态全局变量

void main()
{
        n=20;
        cout<<n<<endl;
        fn();
}

//File2
#include <iostream.h>
extern int n;
void fn()
{
        n++;
        cout<<n<<endl;
}
```

编译并运行Example 2，会发现上述代码可以分别通过编译，但运行时出现错误。 这就是因为静态全局变量不能被其它文件所用，即使在其它文件中使用extern 进行声明也不行。

我们将

```
static int n; //定义静态全局变量
```

改为

```
int n; //定义全局变量
```

再次编译运行程序，程序可正常运行。

因此，在一个文件中，静态全局变量和全局变量功能相同；而在两个文件中，要使用同一个变量，则只能使用全局变量而不能使用静态全局变量。

4.（面向过程的）静态局部变量

在局部变量前，加上关键字static，该变量就被定义成为一个静态局部变量。

```
//Example 3
#include <iostream.h>
void fn();
void main()
{
        fn(); //10
        fn(); //11
        fn(); //12
}

void fn()
{
        static n=10;
        cout<<n<<endl;
        n++;
}
```

通常，在函数体内定义了一个变量，每当程序运行到该语句时都会给该局部变量分配栈内存。但随着程序退出函数体，系统就会收回栈内存，局部变量也相应失效。

但有时候我们需要在两次调用之间对变量的值进行保存。通常的想法是定义一个全局变量来实现。但这样一来，变量已经不再属于函数本身了，不再仅受函数的控制，这给程序的维护带来不便。

静态局部变量正好可以解决这个问题。静态局部变量保存在全局数据区，而不是保存在栈中，每次的值保持到下一次调用，直到下次赋新值。

静态局部变量有以下特点：

1. 静态局部变量在全局数据区分配内存；
2. 静态局部变量在程序执行到该对象的声明处时被首次初始化，即以后的函数调用不再进行初始化；
3. 静态局部变量一般在声明处初始化，如果没有显式初始化，会被程序自动初始化为0；
4. 静态局部变量始终驻留在全局数据区，直到程序运行结束。但其作用域为局部作用域，当定义它的函数或语句块结束时，其作用域随之结束；

5. （面向过程的）静态函数

在函数的返回类型前加上static关键字,函数即被定义为静态函数。静态函数与普通函数不同，它只能在声明它的文件当中可见，不能被其它文件使用。

```
//Example 4
#include <iostream.h>
static void fn();//声明静态函数

void main()
{
        fn();
}

void fn()//定义静态函数
{
        int n=10;
        cout<<n<<endl;
}
```

定义静态函数的好处：（类似于静态全局变量）

1. 静态函数不能被其它文件所用；
2. 其它文件中可以定义相同名字的函数，不会发生冲突；

参考：

发布于 2018-05-29

C++ 中的 static