C 常量 C 运算符

C 存储类

存储类定义 C 程序中变量/函数的范围（可见性）和生命周期。这些说明符放置在它们所修饰的类型之前。下面列出 C 程序中可用的存储类：

- auto
- register
- static
- extern

auto 存储类

auto 存储类是所有局部变量默认的存储类。

```
{
   int mount;
   auto int month;
}
```

上面的实例定义了两个带有相同存储类的变量，auto 只能用在函数内，即 auto 只能修饰局部变量。

register 存储类

register 存储类用于定义存储在寄存器中而不是 RAM 中的局部变量。这意味着变量的最大尺寸等于寄存器的大小（通常是一个词），且不能对它应用一元的 '&' 运算符（因为它没有内存位置）。

```
{
   register int  miles;
}
```

寄存器只用于需要快速访问的变量，比如计数器。还应注意的是，定义 'register' 并不意味着变量将被存储在寄存器中，它意味着变量可能存储在寄存器中，这取决于硬件和实现的限制。

static 存储类

static 存储类指示编译器在程序的生命周期内保持局部变量的存在，而不需要在每次它进入和离开作用域时进行创建和销毁。因此，使用 static 修饰局部变量可以在函数调用之间保持局部变量的值。

static 修饰符也可以应用于全局变量。当 static 修饰全局变量时，会使变量的作用域限制在声明它的文件内。

全局声明的一个 static 变量或方法可以被任何函数或方法调用，只要这些方法出现在跟 static 变量或方法同一个文件中。

以下实例演示了 static 修饰全局变量和局部变量的应用：

实例

```
#include <stdio.h>
/* 函数声明 */
void func1(void);
static int count=10; /* 全局变量 - static 是默认的 */
int main()
{
while (count--) {
func1();
}
return 0;
}
void func1(void)
{
/* 'thingy' 是 'func1' 的局部变量 - 只初始化一次
* 每次调用函数 'func1' 'thingy' 值不会被重置。
*/
static int thingy=5;
thingy++;
printf(" thingy 为 %d ，  count 为 %d\n", thingy, count);
}
```

实例中 count 作为全局变量可以在函数内使用，thingy 使用 static 修饰后，不会在每次调用时重置。

可能您现在还无法理解这个实例，因为我已经使用了函数和全局变量，这两个概念目前为止还没进行讲解。即使您现在不能完全理解，也没有关系，后续的章节我们会详细讲解。当上面的代码被编译和执行时，它会产生下列结果：

```
  thingy 为 6 ，  count 为 9
  thingy 为 7 ，  count 为 8
  thingy 为 8 ，  count 为 7
  thingy 为 9 ，  count 为 6
  thingy 为 10 ，  count 为 5
  thingy 为 11 ，  count 为 4
  thingy 为 12 ，  count 为 3
  thingy 为 13 ，  count 为 2
  thingy 为 14 ，  count 为 1
  thingy 为 15 ，  count 为 0
```

extern 存储类

extern 存储类用于提供一个全局变量的引用，全局变量对所有的程序文件都是可见的。当您使用 'extern' 时，对于无法初始化的变量，会把变量名指向一个之前定义过的存储位置。

当您有多个文件且定义了一个可以在其他文件中使用的全局变量或函数时，可以在其他文件中使用 *extern* 来得到已定义的变量或函数的引用。可以这么理解，*extern* 是用来在另一个文件中声明一个全局变量或函数。

extern 修饰符通常用于当有两个或多个文件共享相同的全局变量或函数的时候，如下所示：

第一个文件：main.c

实例

```
#include <stdio.h>
int count ;
extern void write_extern();
int main()
{
count = 5;
write_extern();
}
```

第二个文件：support.c

实例

```
#include <stdio.h>
extern int count;
void write_extern(void)
{
printf("count is %d\n", count);
}
```

在这里，第二个文件中的 *extern* 关键字用于声明已经在第一个文件 main.c 中定义的 *count*。现在 ，编译这两个文件，如下所示：

```
$ gcc main.c support.c
```

这会产生 **a.out** 可执行程序，当程序被执行时，它会产生下列结果：

```
count is 5
```

C 常量 C 运算符