

C Basic

C language

<<

<< 左移：右边空出的位上补0，左边的位将从字头挤掉，左移n位，其值相当于乘 2^n

需要注意的一个问题是int类型最左端的符号位和移位移出去的情况。我们知道，int是有符号的整形数，最左端的1位是符号位，即0正1负，那么移位的时候就会出现溢出，例如：

```
int i = 0x40000000; //16进制的40000000,为2进制的01000000...0000  
i = i << 1;
```

那么，i在左移1位之后就会变成0x80000000，也就是2进制的100000...0000，符号位被置1，其他位全是0，变成了int类型所能表示的最小值，32位的int这个值是-2147483648，溢出。如果再接着把i左移1位会出现什么情况呢？在C语言中采用了丢弃最高位的处理方法，丢弃了1之后，i的值变成了0。

>>

>> 右移：右边的位被挤掉了，对于左边移出的空位，如果是正数则空位补0，若为负数，可能补0或1，这取决于不同的计算系统。

```
int i=1;  
i<<2;
```

右移对符号位的处理和左移不同，对于有符号整数来说，比如int类型，右移会保持符号位不变，例如：

```
int i = 0x80000000;  
i = i >> 1; //i的值不会变成0x40000000,而会变成0xc0000000
```

就是说，符号位向右移动后，正数的话补0，负数补1，也就是汇编语言中的算术右移。同样当移动的位数超过类型的长度时，会取余数，然后移动余数个位。

负数10100110 >>5(假设字长为8位)，则得到的是 11111101

总之，在C中，左移是逻辑/算术左移（两者完全相同），右移是算术右移，会保持符号位不变。实际应用中可以根据情况用左/右移做快速的乘/除运算，这样会比循环效率高很多。

>>>

>>> 运算符：右边的位被挤掉，对于左边移出的空位一概补上0；

位与

&

位异或 XOR/exclusive or

位操作符：

位操作直接将两个操作数按照二进制对应进行操作；

例：`0xaa&`（位与）`0xf0=0xa0;`

逻辑操作是 两个操作数整体来操作；

例：`0xaa&&`（逻辑与）`0xf0=1;`

嵌入式的移位操作针对于无符号数：左移时右侧补0，右移时左侧补0，相当于逻辑移位

对于有符号数：左移右侧补0，右移左侧补符号位（正数补0，负数补1），相当于算术移位

运算优先级：

算术运算符 优先于 位移位运算符 优先于 关系运算符

条件编译

一般情况下，源程序中所有的行都参加编译。但是有时希望对其中一部分内容只在满足一定条件才进行编译，也就是对一部分内容指定编译的条件，这就是“条件编译”。有时，希望当满足某条件时对一组语句进行编译，而当条件不满足时则编译另一组语句。

`#if` 编译预处理中的条件命令，相当于C语法中的`if`语句

`#ifdef` 判断某个宏是否被定义，若已定义，执行随后的语句

`#ifndef` 与`#ifdef`相反，判断某个宏是否未被定义

`#elif` 若`#if`, `#ifdef`, `#ifndef` 或前面的`#elif`条件不满足，则执行`#elif`之后的语句，相当于C语法中的`else-if`

`#else` 与`#if`, `#ifdef`, `#ifndef` 对应，若这些条件不满足，则执行`#else`之后的语句，相当于C语法中的`else`

`#endif` `#if`, `#ifdef`, `#ifndef` 这些条件命令的结束标志

`defined` 与`#if`, `#elif` 配合使用，判断某个宏是否被定义

```
#ifdef语句1
```

```
//程序2  
#endif
```

如果宏定义了语句1则程序2。

作用：我们可以用它区隔一些与特定头文件、程序库和其他文件版本有关的代码。

```
#ifdef Program1  
//  
#else
```

它的作用是：当标识符已经被定义过(一般是用`#define`命令定义)，则对程序段1进行编译，否则编译程序段2。

优点

条件编译可以提高C源程序的通用性

例如，在调试程序时，常常希望输出一些所需的信息，而在调试完成后不再输出这些信息。可以在源程序中插入以下的条件编译段：

```
#ifdef DEBUG  
print("device_open(%p)\n",file);  
#endif
```

如果在它的前面有以下命令行：`#define DEBUG`

则在程序运行时输出file指针的值，以便调试分析。调试完成后只需将这个**define**命令行删除即可。有人可能觉得不用条件编译也可达此目的，即在调试时加一批**printf**语句，调试后一一将**printf**语句删除去。的确，这是可以的。但是，当调试时加的**printf**语句比较多时，修改的工作量是很大的。用条件编译，则不必一一删改**printf**语句，只需删除前面的一条“#define **DEBUG**”命令即可，这时所有的用**DEBUG**作标识符的条件编译段都使其中的**printf**语句不起作用，即起统一控制的作用，如同一个“开关”一样。有人会问：不用条件编译命令而直接用**if**语句也能达到要求，用条件编译命令有什么好处呢？的确，此问题完全可以不用条件编译处理，但那样做目标程序长（因为所有语句都编译），而采用条件编译，可以减少被编译的语句，从而减少目标的长度。当条件编译段比较多时，目标程序长度可以大大减少。

bits shift opt. can be found in following links:

<https://blog.csdn.net/21aspnet/article/details/160037>