C语言位运算（按位与运算、或运算、异或运算、左移运算、右移运算）

< 上一页 下一页 >

所谓位运算，就是对一个比特（Bit）位进行操作。在《数据在内存中的存储》一节中讲到，比特（Bit）是一个电子元器件，8个比特构成一个字节（Byte），它已经是粒度最小的可操作单元了。

C语言提供了六种位运算符：

运算符	&	|	^	~	<<	>>
说明	按位与	按位或	按位异或	取反	左移	右移

按位与运算（&）

一个比特（Bit）位只有 0 和 1 两个取值，只有参与 & 运算的两个位都为 1 时，结果才为 1，否则为 0。例如 1&1 为 1，0&0 为 0，1&0 也为 0，这和逻辑运算符 && 非常类似。

C语言中不能直接使用二进制，& 两边的操作数可以是十进制、八进制、十六进制，它们在内存中最终都是以二进制形式存储，& 就是对这些内存中的二进制位进行运算。其他的位运算符也是相同的道理。

例如，9 & 5 可以转换成如下的运算：

```
   0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 1001  （9 在内存中的存储）
& 0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 0101  （5 在内存中的存储）
--------------------------------------------------------------------------------
   0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 0001  （1 在内存中的存储）
```

也就是说，按位与运算会对参与运算的两个数的所有二进制位进行 & 运算，9 & 5 的结果为 1。

又如，-9 & 5 可以转换成如下的运算：

```
  1111 1111 -- 1111 1111 -- 1111 1111 -- 1111 0111  （-9 在内存中的存储）
& 0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 0101  （5 在内存中的存储）
--------------------------------------------------------------------------
  0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 0101  （5 在内存中的存储）
```

`-9 & 5` 的结果是 5。

> 关于正数和负数在内存中的存储形式，我们已在VIP教程《整数在内存中是如何存储的，为什么它堪称天才般的设计》中进行了讲解。

再强调一遍，`&` 是根据内存中的二进制位进行运算的，而不是数据的二进制形式；其他位运算符也一样。以 `-9&5` 为例，-9 的在内存中的存储和 -9 的二进制形式截然不同：

```
 1111 1111 -- 1111 1111 -- 1111 1111 -- 1111 0111  （-9 在内存中的存储）
-0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 1001  （-9 的二进制形式，前面多余的 0 可以抹掉）
```

按位与运算通常用来对某些位清 0，或者保留某些位。例如要把 n 的高 16 位清 0，保留低 16 位，可以进行 `n & 0XFFFF` 运算（0XFFFF 在内存中的存储形式为 0000 0000 -- 0000 0000 -- 1111 1111 -- 1111 1111）。

【实例】对上面的分析进行检验。

```
01.  #include <stdio.h>
02.
03.  int main(){
04.      int n = 0X8FA6002D;
05.      printf("%d, %d, %X\n", 9 & 5, -9 & 5, n & 0XFFFF);
06.      return 0;
07.  }
```

运行结果：
1, 5, 2D

按位或运算（|）

参与 `|` 运算的两个二进制位有一个为 1 时，结果就为 1，两个都为 0 时结果才为 0。例如 `1|1` 为 1，`0|0` 为0，`1|0` 为1，这和逻辑运算中的 `||` 非常类似。

例如，`9 | 5` 可以转换成如下的运算：

```
  0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 1001  （9 在内存中的存储）
| 0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 0101  （5 在内存中的存储）
--------------------------------------------------------------------------
  0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 1101  （13 在内存中的存储）
```

`9 | 5` 的结果为 13。

又如，`-9 | 5` 可以转换成如下的运算：

```
  1111 1111 -- 1111 1111 -- 1111 1111 -- 1111 0111  （-9 在内存中的存储）
| 0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 0101  （5 在内存中的存储）
--------------------------------------------------------------------------
  1111 1111 -- 1111 1111 -- 1111 1111 -- 1111 0111  （-9 在内存中的存储）
```

`-9 | 5` 的结果是 -9。

按位或运算可以用来将某些位置 1，或者保留某些位。例如要把 n 的高 16 位置 1，保留低 16 位，可以进行 `n | 0XFFFF0000` 运算（0XFFFF0000 在内存中的存储形式为 1111 1111 -- 1111 1111 -- 0000 0000 -- 0000 0000）。

【实例】对上面的分析进行校验。

```
#include <stdio.h>

int main(){
    int n = 0X2D;
    printf("%d, %d, %X\n", 9 | 5, -9 | 5, n | 0XFFFF0000);
    return 0;
}
```

运行结果：
13, -9, FFFF002D

按位异或运算（^）

参与 `^` 运算两个二进制位不同时，结果为 1，相同时结果为 0。例如 `0^1` 为1，`0^0` 为0，`1^1` 为0。

例如，`9 ^ 5` 可以转换成如下的运算：

```
  0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 1001  （9 在内存中的存储）
^ 0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 0101  （5 在内存中的存储）
```

```
--------------------------------------------------------------------------------
  0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 1100  （12 在内存中的存储）
```

`9 ^ 5` 的结果为 12。

又如，`-9 ^ 5` 可以转换成如下的运算：

```
  1111 1111 -- 1111 1111 -- 1111 1111 -- 1111 0111  （-9 在内存中的存储）
^ 0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 0101  （5 在内存中的存储）
--------------------------------------------------------------------------------
  1111 1111 -- 1111 1111 -- 1111 1111 -- 1111 0010  （-14 在内存中的存储）
```

`-9 ^ 5` 的结果是 -14。

按位异或运算可以用来将某些二进制位反转。例如要把 n 的高 16 位反转，保留低 16 位，可以进行n ^ 0XFFFF0000运算（0XFFFF0000 在内存中的存储形式为 1111 1111 -- 1111 1111 -- 0000 0000 -- 0000 0000）。

【实例】对上面的分析进行校验。

```
01.  #include <stdio.h>
02.
03.  int main(){
04.      unsigned n = 0X0A07002D;
05.      printf("%d, %d, %X\n", 9 ^ 5, -9 ^ 5, n ^ 0XFFFF0000);
06.      return 0;
07.  }
```

运行结果：
12, -14, F5F8002D

取反运算（~）

取反运算符 `~` 为单目运算符，右结合性，作用是对参与运算的二进制位取反。例如 `~1` 为0，`~0` 为1，这和逻辑运算中的 `!` 非常类似。。

例如，`~9` 可以转换为如下的运算：

```
~ 0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 1001  （9 在内存中的存储）
--------------------------------------------------------------------------------
  1111 1111 -- 1111 1111 -- 1111 1111 -- 1111 0110  （-10 在内存中的存储）
```

所以 `~9` 的结果为 -10。

例如，`~-9` 可以转换为如下的运算：

```
~ 1111 1111 -- 1111 1111 -- 1111 1111 -- 1111 0111  （-9 在内存中的存储）
--------------------------------------------------------------------------------
  0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 1000  （9 在内存中的存储）
```

所以 `~-9` 的结果为 8。

【实例】对上面的分析进行校验。

```
01.  #include <stdio.h>
02.
03.  int main(){
04.      printf("%d, %d\n", ~9, ~-9 );
05.      return 0;
06.  }
```

运行结果：
-10, 8

左移运算（<<）

左移运算符 `<<` 用来把操作数的各个二进制位全部左移若干位，高位丢弃，低位补0。

例如，`9<<3` 可以转换为如下的运算：

```
<< 0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 1001  （9 在内存中的存储）
--------------------------------------------------------------------------------
   0000 0000 -- 0000 0000 -- 0000 0000 -- 0100 1000  （72 在内存中的存储）
```

所以 `9<<3` 的结果为 72。

又如，`(-9)<<3` 可以转换为如下的运算：

```
<< 1111 1111 -- 1111 1111 -- 1111 1111 -- 1111 0111  （-9 在内存中的存储）
--------------------------------------------------------------------------------
   1111 1111 -- 1111 1111 -- 1111 1111 -- 1011 1000  （-72 在内存中的存储）
```

所以(-9)<<3的结果为 -72

如果数据较小，被丢弃的高位不包含 1，那么左移 n 位相当于乘以 2 的 n 次方。

【实例】对上面的结果进行校验。

```
01.  #include <stdio.h>
```

```
02.
03.  int main(){
04.      printf("%d, %d\n", 9<<3, (-9)<<3 );
05.      return 0;
06.  }
```

运行结果：
72, -72

右移运算（>>）

右移运算符 `>>` 用来把操作数的各个二进制位全部右移若干位，低位丢弃，高位补 0 或 1。如果数据的最高位是 0，那么就补 0；如果最高位是 1，那么就补 1。

例如，`9>>3` 可以转换为如下的运算：

```
>> 0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 1001  （9 在内存中的存储）
--------------------------------------------------------------------------------
   0000 0000 -- 0000 0000 -- 0000 0000 -- 0000 0001  （1 在内存中的存储）
```

所以 `9>>3` 的结果为 1。

又如，`(-9)>>3` 可以转换为如下的运算：

```
>> 1111 1111 -- 1111 1111 -- 1111 1111 -- 1111 0111  （-9 在内存中的存储）
--------------------------------------------------------------------------------
   1111 1111 -- 1111 1111 -- 1111 1111 -- 1111 1110  （-2 在内存中的存储）
```

所以 `(-9)>>3` 的结果为 -2

如果被丢弃的低位不包含 1，那么右移 n 位相当于除以 2 的 n 次方（但被移除的位中经常会包含 1）。

【实例】对上面的结果进行校验。

```
01.  #include <stdio.h>
02.
03.  int main(){
04.      printf("%d, %d\n", 9>>3, (-9)>>3 );
05.      return 0;
06.  }
```

运行结果：
1, -2

< 上一页 下一页 >

所有教程

socket Python基础教程 C#教程 MySQL MySQL函数 C语言入门
C语言专题 C语言编译器 C语言编程实例 GCC编译器 数据结构 C++教程
OpenCV Qt教程 Unity 3D教程 UE4 STL vi命令 Android教程
Linux入门 Linux命令 Shell脚本 Java教程 设计模式 Java Swing
JSP教程 CSS教程 TensorFlow 区块链 Go语言教程 Docker 编程笔记
资源下载 编程视频 关于我们 汇编语言 大数据 云计算

优秀文章

C语言if语句
Linux whereis命令：查找二进制命令、源文件和帮助文档
Java字节流的使用：字节输入/输出流、文件输入/输出流、字节数组输入/输出流
Linux系统信号与管道系列视频教程（小江老师出品13集）
C语言指针数组（数组每个元素都是指针）详解
Unity 3D ToolTip控件
Shell declare和typeset命令：设置变量属性
Android MediaRecorder录制音频
Docker Swarm服务日志及相关配置
深度优先生成树和广度优先生成树（详解版）