

## 本小节内容

浮点数 IEEE754 标准解析及实战计算演示

## 1 浮点数 IEEE754 标准解析及实战计算演示(组成要考,选择题)

在 C 语言中,要使用 float 关键字或 double 关键字定义浮点型变量。float 型变量占用的内存空间为 4 字节,double 型变量占用的内存空间为 8 字节。与整型数据的存储方式不同,浮点



图 1 浮点型数据的组成

型数据是按照指数形式存储的。系统把一个浮点型数据分成小数部分(用 M 表示)和指数部分(用 E 表示)并分别存放。指数部分采用规范化的指数形式,指数也分正、负(符号位,用 S 表示),如图 1 所示。

数符(即符号位)占1位,是0时代表正数,是1时代表负数。表1是IEEE-754浮点型变量存储标准。

格式	SEEEEEE	ЕМММММММ	МММММММ	ММММММММ		
二进制数	01000000	10010000	00000000	00000000		
十六进制数	40	90	00	00		

表 1 IEEE-754 浮点型变量存储标准

- S: S是符号位, 用来表示正、负, 是1时代表负数, 是0时代表正数。
- **E**: **E** 代表指数部分 (指数部分的值规定只能是 1 到 254, 不能是全 0, 全 1) , 指数部分 运算前都要减去 127 (这是 IEEE-754 的规定) , 因为还要表示负指数。这里的 10000001 转换 为十进制数为 129, 129 127 = 2, 即实际指数部分为 2。

上面表 1 可以变为如下表格:

S	阶码	尾数
0	10000001	0010 0000 0000 0000 0000 000

接下来我们通过下面实例来演示:

#include <stdio.h>

int main()



```
float f = 4.5;
  float f1 = 1.456;
   return 0;
}
    执行方法如下图所示:
             #include <stdio.h>
             int main()
                 float f = 4.5; f: 4.5
                 float f1 = 1.456; f1: 1.45599997
                 return 0;
            f main
在这里打断点并点击小虫子调试按钮
                        选择内存视图,首先在这里输入&f,然后再输入&f1
     内存视图
           转到:
                &f1
                                                        0x000000000061fe18
                                                                  这个是f的内存样子
   061fe00
                    08 00 00 00
                                  00 00 00 00
                                                 3a 00 00 00
                                                                00 00 00 00
   061fe10
                    00 15 cb 00
                                   00 00 00 00
                                                 35 5e ba 3f
                                                                00 00 90 40
   061fe20
                    c0 14 cb 00
                                  00 00 00 00
                                                 c7 13 40 00
                                                                00 00 00 00
                    00 00 00 00
   061fe30
                                   00 00 00 00
                                                 3a 00 00 00
                                                                00 00 00 00
```

在上图中我们可以看到 4.5, 也就是变量 f 的内存是 00 00 90 40, 因为是小端存储, 所以实际值是 0x40900000。

接着看指数部分,计算机并不能直接计算 10 的幂次,**f 的指数部分是表 2 中的 EEEEEEEE 所对应的部分,也就是 1000 0001**,其十进制值为 129,129 – 127 = 2,即实际指数部分为 2 指数值为 2,代表 2 的 2 次幂。因此将 1.001 向左移动 2 位即可,也就是 100.1;然后转换为十进制数,整数部分是 4,小数部分是  $2^{-1}$ ,刚好等于 0.5,因此十进制数为 4.5。**浮点数的小数部分是通过 2^{-1}+2^{-2}+2^{-3}+\cdots来近似一个浮点数的。** 

可能你会疑惑,不应该是小数部分乘以指数部分么,怎么变成左移 2 位,其实是等价的,对于小数部分 1.001,其十进制值为  $2^{\circ}$  +  $2^{-3}$  = 1.125,那么 1.125\*指数部分,也就会 4,就是 1.125\*4=4.5,也就是等于 4.5。

表 2 浮点数 4.5 的存储效果



格式	SEEEEEE	EMMMMMMM	МММММММ	MMMMMMMM
二进制数	01000000	10010000	00000000	00000000
十六进制数	40	90	00	00

在上图中我们可以看到 1.456, 也就是变量 f1 的内存是 35 5e ba 3f, 因为是小端存储, 所以实际值是 0x3fba5e35。

首先先看 f1 的小数部分,如下表 3 所示, M (灰色) 代表小数部分,这里为 011 1010 0101 1110 0011 0101,总计 23 位。底数左边省略存储了一个 1 (这是 IEEE-754 的规定),使用的实际底数表示为 1.011 1010 0101 1110 0011 0101。

接着看指数部分, 计算机并不能直接计算 10 的幂次, **f1 的指数部分是表 3 中的 EEEEEEEE 所对应的部分, 也就是 0**111 1111, 其十进制值为 127, 127 – 127 = 0, 即实际指数部分为 0 指数值为 0, 代表 2 的 0 次幂。因此 1.011 1010 0101 1110 0011 0101 无需做移动。 **浮点数的小数部分是通过 2<sup>0</sup> + 2<sup>-2</sup> + 2<sup>-3</sup> +2<sup>-4</sup> + 2<sup>-6</sup>····来近似一个浮点数的, 1+0.25+0.125+0.0625+0.015625=1.453125。** 

表 3 浮点数 1.456 的存储效果

格式	SEEEEEE	EMMMMMMM	MMMMMMM	MMMMMMMM	
二进制数	00111111	10111010	01011110	00110101	
十六进制数	3f	ba	5e	35	