深入理解取整、取余与取模问题

原创 鹿九丸 于 2021-11-15 23:04:05 发布 5546 收藏 15 版权

分类专栏： C语言系列 文章标签： c语言

C语言系列 专栏收录该内容 4 订阅 23 篇文章 订阅专栏

目录

---

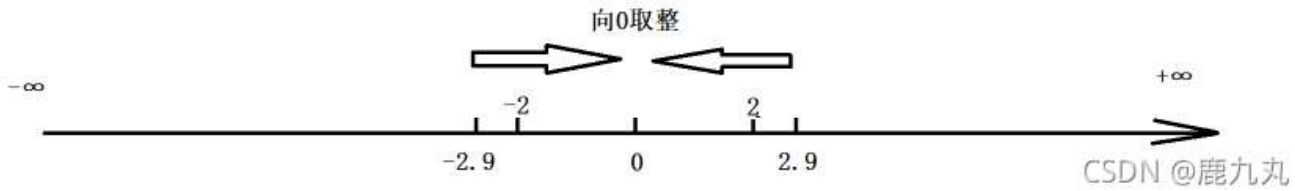
1. 取整问题

1.0向取整（C语言默认的取整方案）

```
#include<stdio.h>
#include<windows.h>
int main()
{
    //本质是向0取整
   //trunc()函数也有这种作用，不过返回值是浮点数，而且必须引用math.h头文件
    int i = -2.9;
    int j = 2.9;
    printf("%d\n", i); //结果是：-2
```

```
    printf("%d\n", j); //结果是：2
    return 0;
}
```

```
    system("pause");
```



2.地板取整（向负无穷的方向取整）

```
#include <stdio.h>
#include <math.h> //因为使用了floor函数，需要添加该头文件
#include <windows.h>
int main()
{
    //本质是向-∞取整，注意输出格式要不然看不到结果
    printf("%.1f\n", floor(-2.9)); //-3
    printf("%.1f\n", floor(-2.1)); //-3
    printf("%.1f\n", floor(2.9)); //2
    printf("%.1f\n", floor(2.1)); //2
    system("pause");
    return 0;
}
```

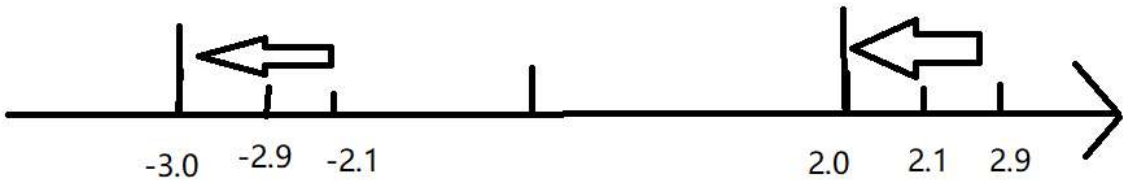
注意：使用floor()函数需要引头文件，参数为double 类型。返回值也同样为double类型。同时不要忘了引math.h头文件。

floor

Calculates the floor of a value.

double floor(double *x*);

Function	Required Header	Compatibility
floor	<math.h>	ANSI, Win 95, Win NT



注意：python默认的取整方案就是地板取整，后面也正是因为这个原因，才出现了取模的不同！

3.天花板取整（向+无穷的方向取整）

```c
#include <stdio.h>
#include <math.h>
#include <windows.h>
int main()
{
    //本质是向+∞取整，注意输出格式要不然看不到结果
    printf("%.1f\n", ceil(-2.9)); //-2
    printf("%.1f\n", ceil(-2.1)); //-2
    printf("%.1f\n", ceil(2.9)); //3
    printf("%.1f\n", ceil(2.1)); //3
    system("pause");
    return 0;
}
```

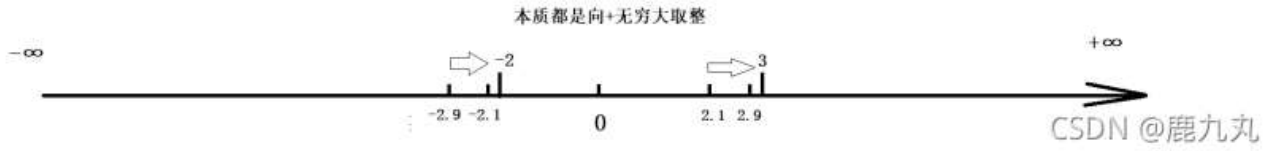
注意：使用ceil()函数需要引头文件，参数为double 类型。返回值也同样为double类型。同时不要忘了引math.h头文件。

ceil

Calculates the ceiling of a value.

double ceil(double *x*);

Routine	Required Header	Compatibility
ceil	<math.h>	ANSI, Win 95, Win NT



4.四舍五入取整

```c
#include <stdio.h>
#include <math.h>
#include <windows.h>
int main()
{
//本质是四舍五入
printf("%.1f\n", round(2.1));
printf("%.1f\n", round(2.9));
printf("%.1f\n", round(-2.1));
printf("%.1f\n", round(-2.9));
system("pause");
return 0;
}
```

注意：使用round()函数需要引头文件，参数为double 类型。返回值也同样为double类型。同时不要忘了引math.h头文件。

汇总例子

```c
#include <stdio.h>
#include <math.h>
#include <windows.h>
int main()
{
    const char* format = "%.1f \t%.1f \t%.1f \t%.1f \t%.1f\n";
    printf("value\tround\tfloor\tceil\ttrunc\n");
    printf("-----\t-----\t-----\t----\t-----\n");
    printf(format, 2.3, round(2.3), floor(2.3), ceil(2.3), trunc(2.3));
    printf(format, 3.8, round(3.8), floor(3.8), ceil(3.8), trunc(3.8));
    printf(format, 5.5, round(5.5), floor(5.5), ceil(5.5), trunc(5.5));
    printf(format, -2.3, round(-2.3), floor(-2.3), ceil(-2.3), trunc(-2.3));
    printf(format, -3.8, round(-3.8), floor(-3.8), ceil(-3.8), trunc(-3.8));
    printf(format, -5.5, round(-5.5), floor(-5.5), ceil(-5.5), trunc(-5.5));
    system("pause");
    return 0;
}
```

```
value   round   floor   ceil    trunc
-----   -----   -----   ----    -----
2.3     2.0     2.0     3.0     2.0
3.8     4.0     3.0     4.0     3.0
5.5     6.0     5.0     6.0     5.0
-2.3    -2.0    -3.0    -2.0    -2.0
-3.8    -4.0    -4.0    -3.0    -3.0
-5.5    -6.0    -6.0    -5.0    -5.0
请按任意键继续. . .                CSDN @鹿九丸
```

2.取模问题

1.余数的定义

余数的定义：如果a和d是两个自然数，d非零，可以证明存在两个唯一的整数 q 和 r，满足 a = q*d + r , q 为整数，且0 ≤ |r|< |d|。其中，q 被称为商，r 被称为余数。

注意：余数并不一定都是正数，大家一定要牢记这个概念！

2.两种余数

> 由定义可知：
>
> **-10%3=-1------>-10/3=-3------->3*(-3)+(-1)=(-10)（C语言中是这样的）**
>
> **-10%3=2------->-10/3=-4------->4*(-3)+ 2=(-10)（python环境中是这样的）**
>
> 解释C: -10 = (-3) * 3 + (-1)**(负余数)**
>
> 解释Python：-10 = （?）* 3 + 2,其中，可以推到出来,'?'必须是-4,即-10 = （-4）* 3 + 2，才能满足定义。 **（正余数）**
>
> **所以，在不同语言，同一个计算表达式，负数“取模”结果是不同的。我们可以称之为分别叫做正余数和负余数。**

3.为什么会有这种现象?

> 由上面的例子可以看出，具体余数r的大小，本质是取决于**商q**的。
>
> 而商，又取决谁呢？取决于除法计算的时候，**取整规则。**
>
> **C语言中默认是 0 向取整，python中默认是-无穷的方向取整。**

3.区分取余与取模

1.取余与与取模的本质区别

> **取余：尽可能让商，进行向0取整。**
>
> **取模：尽可能让商，向负无穷方向取整。**
>
> 所以：
>
> **C中%,本质其实是取余。**
>
> **Python中%，本质其实是取模。**

2.理解链

> **对任何一个大于0的数，对其进行0向取整和负无穷取整，取整方向是一致的。**故**取模等价于取余。**其实这也是为什么我们常常会认为取模以取余是一码事的原因所在。
>
> **对任何一个小于0的数，对其进行0向取整和负无穷取整，取整方向是相反的。**故**取模不等价于取余。**

3.同符号与不同符号

1.同符号：

> **同符号数据相除，得到的商，一定是正数，即大于0！ 故，在对其商进行取整的时候，取模等价于取余。（倘若从数学上理解，就是简单的在负数的前面加一个绝对值即可）**

2.不同符号

```c
#include<stdio.h>
#include <windows.h>
int main()
{
    printf("%d\n", -10 / 3); //结果：-3
    printf("%d\n\n", -10 % 3); //结果：-1 为什么？ -10=(-3)*3+(-1)
    printf("%d\n", 10 / -3); //结果：-3
    printf("%d\n\n", 10 % -3); //结果：1 为什么？10=(-3)*(-3)+1
    system("pause");
    return 0;
}
```

从上面可以看出：

被除数为负数时，取余后为负号。

除数为负数时，取余后为正数。

不同符号在C语言中虽然也有一定的规律，但我并不希望大家利用这个规律，而是**利用定义老老实实的计算**，毕竟这这是针对C语言的结论，在python中就不适用了，因为二者的取整方式是不同的。