**2023/08/15 14:59**

**操作符**

%,取模操作符只能作用于整数

**基本运算符**

**1赋值运算符： =**

C语言中，=不意味着相等，而是起到一个赋值的作用，将等号右边的值赋值给等号左边的变量。、

用到更规范的名称应该称他们为左值和右值

对于早期C语言，左值意味着：

1、它指定一个对象

2、他可以用在赋值运算符的左侧

但是一旦他被加上const这一关键字后，他就称为具有“**常属性**”的变量，将变得不可被修改，就不能作为左值右值指的是可以给左值赋值的量，它可以是常量或者是常属性的变量，但是左值只能是（可修改的）变量

#include <stdio.h>

int main(void)

{

int a;

int b;

int c;

const int TWO = 2;//const修饰以后它便不可修改，所以要在定义时就对其初始化

a = 1;

b = 2;

c =TWO \* (a + b);

}

(a + b)只是一个临时的运算值，不会占用空间，在计算结束后他就会被销毁。

此外，你还可能遇到这样的情况

#include <stdio.h>

int main(void)

{

int a;

a = 1;

a = a + 1;

}

这种情况意味着将（a + 1）赋值给a本身，相当于a加了1；

他也可以写成如下形式

a +=1；

他们是等效的。

同时他也等效于a++；

和++a；

他们的效果都是给a加上1。

**加法运算符： +**

相加的值（运算对象）可以是变量，也可以是常亮，可以用以下语句来表达。

income = salary + bribes;

在这里三者都是可以修改的左值，都可以被赋值，但是（ salary + bribes）是右值。

**减法运算符： -**

它和加号同时被称为二元运算符，因为他们都需要两个运算对象才能操作。

同样的，减号和赋值运算符也可以联动，

a = a - 1;

等价于

a -= 1;

a--;

--a;

**符号运算符：加和减**

int main(void)

{

rokey = -12;

smokey = -rokey;

//此时smokey的值为12

//C90加入了新的标准加号也可以当做一元运算符使用

dozen = +12;

}

**乘法运算符：\***

二元运算符，C语言中的乘号和数学中的不太一样。我们在这里使用\*

而不是 x ，他同样可以和赋值运算符组合，形成 \*= ；

下面是一个小程序实现打印1到109及其平方。

int main(void)

{

int i = 1;

while(i < 11)

{

printf("%d %d\n",i ,i \* i);

i += 1;

}

}

**除法运算符：/**

1、整数除法和浮点数除法不同。浮点数除法的结果是浮点数，而整数的除法结果是整数。整数是没有小数部分的。所以在面对5除以3这样的情况时，整数除法会使得小数部分丢失，这被称为“截断”。而且不会进行四舍五入，

2、如果混合计算，编译器不会真正使浮点数去除以整数，而是先把他们转化成相同的类型，-3.8经过处理后理想值为-4，但是编译器会直接将小数部分截去，变成-3，我们称这一方法为趋零截断。

**取模运算符：%**

特性——只能用于整数

它的作用就是计算除法运算中的余数，

如5 % 3运算结果就是2；

负数取模运算结果为负数

**递增和递减运算符 ++ 和 --**

这两个运算符之前有提到过，他们的作用我们已经知晓了，当然他们可以放在变量的前面也可以放在变量的后面，此时分别对应称他们为前置运算符和后置运算符，

他们之间的区别就是先对其进行递增或者递减运算还是先使用它们，如果是运算符在前则先进行运算再使用，反之则是先使用再运算。

int main(void)

{

int a = 1;

printf("%d\n",a++);

printf("%d\n",++a);

}

以上运行结果为1和3.

**强制类型转换**

如果想将一个类型强制转化为别的类型，只需要在其之前加上小括号中输入类型名即可，当然，既然是类型转化，则必然有弊端，可能出现意想不到的结果，所以需要谨慎使用。

int main(void)

{

float a =9.2;

print("%d\n",(int)a);

}

**优先级**

**括号和成员选择**

[]

()

.

->

**单目一元操作符和双目一元操作符**

-负号运算符

~ 按位取反

++ 自增

-- 自减

\*取值运算符

&取地址

!逻辑非

(type)强制类型

sizeof长度运算符

**高阶运算符**

/

\*

%

**低阶运算符**

+

-

**移位**

<<

>>

**比值**

>

>=

<

<=

==

!=

**按位和逻辑**

&

^

|

&&

||

**条件**

?:

**赋值**

=

/=

\*=

%=

+=

-=

<<=

>>=

&=

^=

|=

**逗号**

,

N4