

类型转换

2022年3月16日 9:59

1. 为什么需要进行类型转换

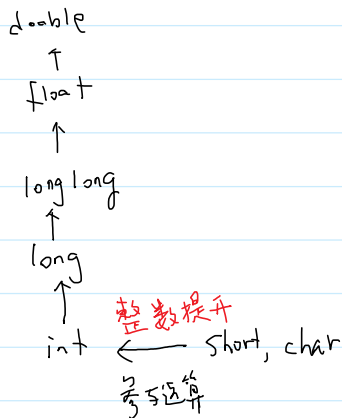
计算机硬件只能对相同类型的数据运算。

2. 何时会发生类型转换。

给定的数据类型与需要的数据类型不匹配

3. 如何进行类型转换。

隐式类型转换：编译器做了类型转换



```
char ch = 'A';
short s = 1;
int i = 10;
long l = 100L;
long long ll = 1000LL;
```

```
float f = 3.14f;
double d = 2.67;
```

f + f	6.28000021	float
d + d	5.3399999999999999	double
f + d	5.8100001049041747	double
l + f	1003.14001	float
ll + d	1002.6700000000000	double

```
char ch = 'A';
short s = 1;
int i = 10;
long l = 100L;
long long ll = 1000LL;
```

ch + ch	130	int
s + s	2	int
i + i	20	int
l + l	200	long
ll + ll	2000	_int64
i + l	110	long
l + ll	1100	_int64

signed → unsigned

```
int i = -1;
unsigned int u = 100;
if (i < u) {
    printf("i is less than u\n");
} else {
    printf("i is greater than u\n");
}
```

Output: i is greater than u

i: 0xFFFFFFFF

u: 0x00000064

注意事项：不要将带符号整数和无符号整数进行运算！

显示转换（强制类型转换）：可以让程序员更精确地控制类型转换

格式: (type-name) expression;

```
// 1.计算浮点数的小数部分
//double d = 3.14, fraction;
//fraction = d - (int)d;

// 2. 表明肯定会发生的转换，提高代码的可读性。
//float f = 3.14;
//// ...
//int i = (int)f;

// 3. 对类型转换进行更精确地控制。
//int dividend = 4, divisor = 3;
//double quotient;
//quotient = (double)dividend / divisor;

// 4. 可以避免溢出
long long millisPerDay = 24 * 60 * 60 * 1000;
long long nanosPerDay = (long long)24 * 60 * 60 * 1000 * 1000 * 1000;
printf("%lld\n", nanosPerDay / millisPerDay);
```

typedef

2022年3月16日 14:43

我们可以使用typedef给类型起别名。

格式: typedef type_name alias;

i.e. typedef int Bool;

Q1: typedef和宏定义之间的区别。

宏定义是在预处理阶段进行处理的(简单的文本替换),编译器是不能够识别宏定义的。因此编译器不能给出一些友好的提示。

编译器能够识别typedef定义的别名,如发生错误,就能够给出一些友好的提示。

定义类型: 请使用typedef而不要使用宏定义。

Q2, 给类型起别名有什么好处。

① 增加代码的可读性

② 增加代码的可移植性。

sizeof 运算符

2022年3月16日 14:55

语法: `sizeof(type_name)`

作用: 计算某一类型的数据所占内存空间的大小(以字节为单位)

<code>sizeof(int)</code>	4
<code>sizeof(i)</code>	4
<code>sizeof(3)</code>	4
<code>sizeof(i + 3)</code>	4

注意事项: `sizeof` 运算符是在编译期进行计算的, 因此它是一个常量表达式, 可以表示数组的长度.

```
int i = 3;  
  
// int arr[i];  
int arr[sizeof(i)];
```

表达式

2022年3月16日 15:03

表达式: 计算某个值的公式. 最简单的表达式: 变量, 常量.

运算符: 连接表达式, 创建更复杂的表达式.

运算符 {
 算术运算符
 赋值运算符
 关系运算符
 比较运算符
 逻辑运算符
 位运算符 (~~xxx~~)
 ...

运算符有两个重要的属性要关注一下: 优先级, 结合性.

Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(type) { list }	Compound literal(C99)	
2	++ --	Prefix increment and decrement ^[note 1]	Right-to-left
	+ -	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(type)	Cast	
	*	Indirection (dereference)	
	&	Address-of	
	sizeof	Size-of ^[note 2]	
	_Alignof	Alignment requirement(C11)	
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10		Bitwise OR (inclusive or)	
11	&&	Logical AND	
12		Logical OR	
13	?:	Ternary conditional ^[note 3]	Right-to-left
14 ^[note 4]	=	Simple assignment	
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= =	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

算术运算符





2022年3月16日 15:12

$+$ $-$ $*$ $/$ $\%$

注意事项: ① $+$ $-$ $*$ $/$ 可以用于浮点数, 但 $\%$ 要求两个操作数都是整数

② 两个整数相除, 其结果为整数. (rounding to zero)

③ $i \% j$ 的结果可能为负, 符号与 i 的符号相同. 满足: $i \% j = i - (i/j) * j$.

 $4 \% 3$	1
 $4 \% -3$	1
 $-4 \% 3$	-1
 $-4 \% -3$	-1

$$4 \% 3 = 4 - (4/3) * 3 = 1$$

$$4 \% -3 = 4 - (4/-3) * (-3) = 1$$

$$-4 \% 3 = -4 - (-4/3) * 3 = -1$$

$$-4 \% -3 = -4 - (-4/-3) * (-3) = -1$$

练习: 判断一个整数的奇偶性.

```
bool is_odd(int n) {  
    return n % 2 == 1;  $\rightarrow$  Error!  
}
```

```
bool is_odd(int n) {  
    return n % 2 != 0;  
}
```

```
bool is_odd(int n) {  
    return n & 1;  
}
```