

## 赋值运算符

2022年3月17日 9:40

简单赋值: =

$V = E$ , 把表达式  $E$  的值赋值给变量  $V$ , 整个表达式的值为赋值后变量  $V$  的值. (A)

注意事项:

① 赋值过程中可能发生隐式类型转换. i.e. `int i = 3.14;`

② 赋值运算符是从右向左结合. `i = j = k = 3;`  $\Rightarrow$  `i = (j = (k = 3))`

```
float f;
int i;
```

```
f = i = 3.33f;
```

	i	f
3	3	3.00000000

复合赋值运算符: `+=`, `-=`, `*=`, `/=`, ...

$a += b \Leftrightarrow a = a + b$

$a -= b \Leftrightarrow a = a - b$

$a *= b \Leftrightarrow a = a * b$

$a /= b \Leftrightarrow a = a / b$

## 自增和自减运算符

2022年3月17日 9:53

自增:  $i = i + 1$ ,  $i++$

自减:  $i = i - 1$ ,  $i--$

C语言专门提供自增和自减的运算符:  $++$ ,  $--$  (\*\*\*)

$i++$ : 表达式的值为  $i$ , 副作用是  $i$  自增。

$++i$ : 表达式的值为  $i+1$ , 副作用是  $i$  自增。

```
int i = 1;
printf("i = %d\n", i++); // 1
printf("i = %d\n", i);   // 2

printf("i = %d\n", ++i); // 3
printf("i = %d\n", i);   // 3
```

注意事项:  $i = i++$ , (会产生未定义的行为)

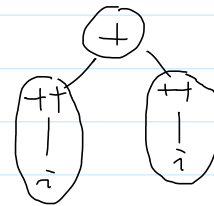
$j = (i++) + (i++);$   $\times$

$a[i] = b[i++];$

$j = i++;$   
 $\downarrow$   
 $temp = i$   
 $j = temp$   $\swarrow$  交换  
 $i = i + 1$

$i = i++$   
 $\downarrow$   
 $temp = i$   
 $i = temp$   
 $i = i + 1$

$(i++) + (i++) \rightarrow$



$i--$ : 表达式的值为  $i$ , 副作用是  $i$  自减

$--i$ : 表达式的值为  $i-1$ , 副作用是  $i$  自减

```
int i = 1;
printf("i = %d\n", i--); // 1
printf("i = %d\n", i);   // 0

printf("i = %d\n", --i); // -1
printf("i = %d\n", i);   // -1
```

练习:

```
int i, j, k;
i = 1;
j = 2;
k = ++i + j++;
printf("i = %d, j = %d, k = %d\n", i, j, k);
```

## 关系运算符

2022年3月17日 10:18

$<, <=, >, >=$  (从左向右)

其运算结果要么为0, 要么为1.

注意事项:  $i < j < k$ . 先计算  $i < j$  这个表达式的值, 结果要么0, 要么为1, 然后再计算  $(0, 1) < k$ .

$j > i \&\& j < k$

# 判等运算符

2022年3月17日 10:21

$==$ ,  $!=$

其运算结果要么为0, 要么为1.

## 逻辑运算符

2022年3月17日 10:22

&&, ||, !

注意事项: && 和 || 运算符会发生短路现象.

$e_1 \&\& e_2$ : 先计算  $e_1$  表达式, 若  $e_1$  为 false, 则不会再计算表达式  $e_2$ .

$e_1 || e_2$ : 先计算表达式  $e_1$ , 若  $e_1$  为 true, 则不会再计算表达式  $e_2$ .

好处: 方便编写程序.

`if (i != 0 && j / i > 1) { ... }` → 利用短路原则.

## 位运算符 (\*\*\*\*)

2022年3月17日 10:52

$\ll, \gg, \&, |, ^, \sim$

### 移位运算符

$i \ll j$ : 将  $i$  左移  $j$  位, 右边补 0.

```
short s = 13;  
printf("s << 2 = %d\n", s << 2);
```

0000 0000 0000 1101  $\rightarrow$  0000 0000 0011 0100

若没有发生溢出, 左移  $j$  位, 相当于乘以  $2^j$

$i \gg j$ : 将  $i$  右移  $j$  位. 若  $i$  为无符号数或非负数, 则右边补 0; 若  $i$  为负数, 它的行为是由实现定义的, 有的左边会补 0, 有的左边补 1.

```
short s = 13;  
printf("s >> 2 = %d", s >> 2);
```

0000 0000 0000 1101  $\rightarrow$  0000 0000 0000 0011

右移  $j$  位, 相当于除以  $2^j$  (向下取整).

为了代码的可移植, 最好不要对带符号整数进行移位运算.

### #2 按位运算符

short  $i = 3$ ,  $j = 4$ ;

$\sim i$ : 0000 0000 0000 0011  $\rightarrow$  1111 1111 1111 1100 (-4)

$i \& j$ : 0000 0000 0000 0011  
 $\&$  0000 0000 0000 0100  
0000 0000 0000 0000 (0)

$i | j$ : 0000 0000 0000 0011  
0000 0000 0000 0100  
0000 0000 0000 0111 (7)

$i ^ j$ : 0000 0000 0000 0011 (相同为 0, 不同为 1)  
 $\wedge$  0000 0000 0000 0100  
加密 0000 1000 0000 0111 (7)

message key

$m ^ k = c$   
 $\uparrow$   
加密

$c ^ k = m$   
 $\uparrow$   
解密 (一次性加密)

$$\begin{aligned} a ^ 0 &= a \\ \Rightarrow a ^ a &= 0 \\ a ^ b &= b ^ a \quad (\text{交换性}) \\ a ^ (b ^ c) &= (a ^ b) ^ c \quad (\text{结合性}) \end{aligned}$$

$$\begin{aligned} c_1 &= m_1 ^ k \\ c_2 &= m_2 ^ k \end{aligned}$$

$$c_1 ^ c_2 = m_1 ^ m_2$$

#1. 如何判断一个数是否奇数?

```
bool is_odd(int n) {  
    return n & 0x1;  
}
```

#2. 如何判断一个整数是否为 2 的幂? 1, 2, 4, 8, ...

```
bool isPowerOf2(unsigned int n) {
    unsigned int i = 1;
    while (i < n) {
        i <<= 1;
    }
    return i == n;
}
```

2 的幂的 = 进制表示有什么特性: 只有 1 个 1。

0000 1000      n  
0000 0111      n-1

0001 1000      n  
0001 0111      n-1

```
bool isPowerOf2(unsigned int n) {
    return (n & n - 1) == 0;
}
```

0 的位数及低于 0 = !

#3 给定一个不为 0 的整数, 找出值为 1 的权重最低的位。

输入: 0011 0100      输出: 4

$((n \wedge n-1) + 1) \gg 1$       ✓

$n \& (-n)$       ✓

0011 0100      n

1100 1100      -n

#4 给定一个整数数组, 里面的数都是成对的, 只有一个数例外, 请找出这个数?

```
int findSingleNumber(int arr[], int n) {
    int singleNum = 0;
    for (int i = 0; i < n; i++) {
        singleNum ^= arr[i];
    }
    return singleNum;
}
```

# 语句

2022年3月17日 14:30

语句 { 表达式语句 (表达式后面添加;) }  
{ 选择语句 (if, switch) }  
{ 循环语句 (while, do, for) }  
{ 跳转语句 (continue, break, goto, return) }  
{ 空语句 (;) }  
{ 复合语句 ({ statements} 将多条语句用花括号括起来组成一条语句.) }



## 选择语句

2022年3月17日 14:35

### # 1. if 语句

格式1: `if (expr) statement`.

```
if (line_num == MAX_LINES)
    line_num = 0;
```

如何让if语句控制多条语句 → 使用复合语句

```
if (line_num == MAX_LINES) {
    line_num = 0;
    page_num++;
}
```

格式2: `if (expr) statement1`  
`else statement2`.

练习: 输入一个整数, 判断它  $> 0$ ,  $= 0$  还是  $< 0$ ?

```
int n;
scanf("%d", &n);

if (n > 0)
    printf("n is positive.\n");
else
    if (n == 0)
        printf("n is zero.\n");
    else
        printf("n is negative\n");
```

if/else 嵌套



```
int n;
scanf("%d", &n);

if (n > 0)
    printf("n is positive.\n");
else if (n == 0)
    printf("n is zero.\n");
else
    printf("n is negative\n");
```

→ 级联式if语句

注意事项: 级联式if语句本质上就是嵌套的if...else...语句

### # 2. switch 语句

最常见的格式:

```
switch (expr) {
    case const-expr: statements
    case const-expr: statements
    ...
    case const-expr: statements
    default: statements
}
```

- 注意事项:
- ① expr的值必须是整数类型 (char).
  - ② case后面必须是整数类型的常量表达式 (char)
  - ③ 不能够有重复标签.
  - ④ 多个分支标签可以共用一条语句

```
int grade;
scanf("%d", &grade);

switch (grade) {
    case 4:
        printf("Excellent!");
        break;
    case 3:
        printf("Good.");
        break;
    case 2:
        printf("Average.");
        break;
    case 1:
        printf("Poor.");
        break;
    case 0:
        printf("Failing.");
        break;
    default:
        printf("Illegal grade.");
        break;
}
```

```

int grade;
scanf("%d", &grade);

switch (grade) {
    case 4: case 3: case 2: case 1:
        printf("Passing.");
        break;
    case 0:
        printf("Failing.");
        break;
    default:
        printf("Illegal grade.");
        break;
}

```

⑤ 如果有省略 break 语句, 则可能发现 case 穿透现象。

```

int grade;
scanf("%d", &grade);

switch (grade) {
    case 4:
        printf("Excellent!\n");
    case 3:
        printf("Good.\n");
    case 2:
        printf("Average.\n");
    case 1:
        printf("Poor.\n");
    case 0:
        printf("Failing.\n");
    default:
        printf("Illegal grade.\n");
}

```

```

2
Average.
Poor.
Failing.
Illegal grade.

```

级联式 if...else 和 switch 语句的比较

- ① 级联式 if...else 比 switch 语句更加通用。
- ② switch 语句比级联式 if...else 可读性更高
- ③ switch 语句的执行效率优于级联式 if...else

※ 3. 条件运算符 (三目运算符)

?: 从右向左结合。

格式:  $\text{expr1} ? \text{expr2} : \text{expr3}$

计算步骤: 首先计算  $\text{expr1}$  的值, 若  $\text{expr1}$  非零, 则计算  $\text{expr2}$  的值, 并把  $\text{expr2}$  的值当作整个表达式的值; 若  $\text{expr1}$  的值为零, 则计算  $\text{expr3}$  的值, 并把  $\text{expr3}$  的值当作整个表达式的值。

练习: 请问最后, 运算的值是多少?

```
int i, j, k;
```

```
i = 1;
```

```
j = 2;
```

```
k = i > j ? i : j;
```

```
k = i > j ? i++ : j++;
```

```
k = (i >= 0 ? i : 0) + j;
```

$i=1, j=3, k=4$

$\text{expr1} ? \text{expr2} : (\text{expr3} ? \text{expr4} : \text{expr5})$  从右向左结合.

$i + j + k \Rightarrow (i + j) + k$  从左向右结合 +

$i = j = k \Rightarrow i = (j = k)$  从右向左结合

## 循环语句

2022年3月17日 15:09

### 1. while 语句

↑ 控制表达式  
格式: `while (expr) statement`  
└─ 循环体

```
while (i < n) {  
    i <= 1;  
}
```

### 2. do 语句

格式: `do statement while (expr);`

do 语句和 while 语句唯一的区别: 当初始条件为假, do 语句的循环体会执行一次, 而 while 语句一次都不会执行。

```
int i = 10;  
  
while (i > 0) {  
    printf("Counting down: %d\n", i--);  
}
```

```
int i = 10;  
  
do {  
    printf("Counting down: %d\n", i--);  
} while (i > 0);
```

```
int i = 0;  
  
while (i > 0) {  
    printf("Counting down: %d\n", i--);  
}
```

```
int i = 0;  
  
do {  
    printf("Counting down: %d\n", i--);  
} while (i > 0);
```

Microsoft Visual Studio 调试控制台

选择 Microsoft Visual Studio 调试控制台  
Counting down: 0

### 3. for 语句

格式: `for (expr1; expr2; expr3) statement`  
└─ 循环体

expr1: 初始化表达式, 只会执行一次。

expr2: 条件控制语句, 若 expr2 非零, 则执行循环体, 若 expr2 为 0, 则退出循环。

expr3: 执行完循环体后要执行的步骤。

```
for (int i = 10; i > 0; i--) {  
    printf("Counting down: %d\n", i);  
}
```

注意事项: expr1, expr2 和 expr3 都可以省略。若省略 expr2, 其默认值为 true。

惯用法: `for (i; ; ) { ... }` → 无限循环。

`while (1) { ... }`

## 跳转语句

2022年3月17日 15:55

### #1. break 语句

作用: 跳出 switch, while, do, for 语句.

最常见于跳出 while (1) 这样的无限循环

```
int n;
while (1) {
    printf("Enter a number (enter 0 to stop): ");
    scanf("%d", &n);

    if (n == 0) {
        break;
    }
    printf("%d squared is %d\n", n, n * n);
}
```

Microsoft Visual Studio 调试控制台

```
Enter a number (enter 0 to stop): 1
1 squared is 1
Enter a number (enter 0 to stop): 2
2 squared is 4
Enter a number (enter 0 to stop): 3
3 squared is 9
Enter a number (enter 0 to stop): 4
4 squared is 16
Enter a number (enter 0 to stop): 0
```

注意事项: 当 switch, while, do, for 语句嵌套时, break 只能跳出包含 break 语句的最内层嵌套.

```
while (...) {
    switch (...) {
        ---
        break;
    }
}
```

Q: 如何在 switch 语句中跳出 while 循环, goto!

### #2. continue 语句

continue 语句和 break 语句之间的区别:

- ① break 语句可以用于循环语句和 switch 语句, 而 continue 语句只能用于循环语句
- ② break 语句是跳出整个循环语句, continue 语句是跳回到循环体的末尾.

例子 对10个非零整数求和

```
int sum = 0, count = 0, n;

while (count < 10) {
    scanf("%d", &n);
    if (n == 0) {
        continue;
    }
    sum += n;
    count++;
}

printf("sum = %d\n", sum);
```

break

```
int sum, count, n;

for (sum = 0, count = 0; count < 10; count++) {
    scanf("%d", &n);
    if (n == 0) {
        continue;
    }
    sum += n;
}

printf("sum = %d\n", sum);
```

Wrong!

### #3. goto 语句

break 只能跳转至 switch 语句和循环语句后一条语句.

continue 只能跳转至循环体的末尾.

goto 语句就没有这些限制, 唯一的限制就是只能在同一个函数内进行跳转。

Q: 很显然可以利用 goto 实现 break 和 continue 的功能, 那为什么有了 goto 还需要 break 和 continue?

① 代码可读性

② 很容易出现 bug.

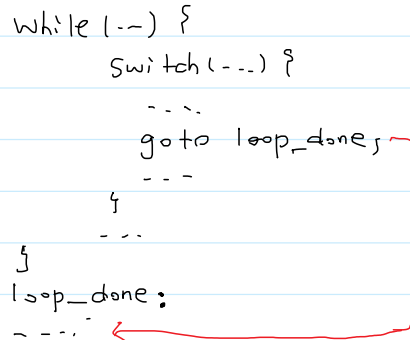
《go to statement considered harmful》

最佳实践: 尽量少用 goto 语句, 只有当其他方式实现不了时, 才考虑使用 goto 语句。

格式: goto 标签。

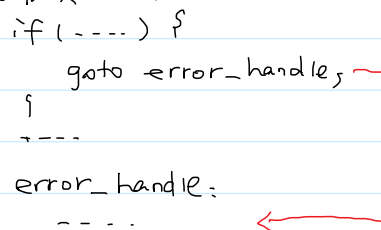
使用场景: ① 跳出外层嵌套。

```
while (--) {  
    switch (...) {  
        ...  
        goto loop_done;  
        ...  
    }  
    ...  
}  
loop_done:  
...
```



② 错误处理

```
if (....) {  
    goto error_handle;  
    ...  
}  
error_handle:  
...
```



Q1. 请用自己的语言描述一下脑海中的数组模型。

连续的一片内存空间, 并且这片连续的内存空间被划分为大小相等的小空间。

Q2. 数组是如何划分大小相等的小空间? 为什么要这么做?

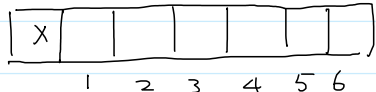
数组只能存放同一种类型的数据 可以随机访问数组元素 (在 O(1) 时间复杂度内访问数组的任一元素)

Q3. 为什么在大多数语言中, 数组的索引都是从 0 开始的?

寻址:  $i\_addr = base\_addr + i * sizeof(element\_type)$  ← 从 0 开始。

若索引从 1 开始:

$i\_addr = base\_addr + (i-1) * sizeof(element\_type)$  → 每一次寻址都会多一次减法运算。

 → 浪费一个元素内存空间。

Q4. 为什么数组的效率一般会优于链表?

1) 数组的内存空间是连续的, 而链表的内存空间不连续。数组可以更好地利用 CPU 的 cache (预读, 局部性原理)

2) 数组只需要存储数据, 链表不仅仅要存储数据, 还要存储指针域。数组的内存使用率更高

H1. 数组的声明

$element\_type \ arr\_name[size];$  →  $int \ arr[10];$

注意事项:  $size$  必须是整型的常量表达式, 在编译期间能计算出数组的大小。

H2. 数组的初始化

$int \ arr[10] = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\};$  初始化式 (initializer)

$int \ arr[10] = \{1, 2, 3\};$  其余元素会初始化为 0。

$int \ arr[10] = \{0\};$  将数组所有元素初始化为 0

$int \ arr[] = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\};$  数据的长度由编译器自行推断, 这里长度 10。

$int \ arr[9] = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\};$  X 初始化式的长度不能比数组的长度大。

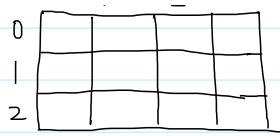
H3. 对数组使用  $sizeof$  运算符

`#define SIZE(a) (sizeof(a) / sizeof(a[0]))`

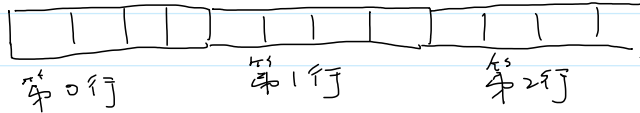
多维数组 (二维数组)

二维数组类似于数学上的矩阵, 比如  $int \ matrix[3][4]$

	0	1	2	3
0				
1				



虽然经常把二维数组看成矩阵，但是二维数组是连续存储的。（行优先）



① 二维数组的初始化。（二维数组元素是一维数组的数组）

`int matrix[3][4] = { {1, 2, 3, 4}, {2, 2, 3, 4}, {3, 2, 3, 4} }`

`int matrix[3][4] = { {1, 2, 3, 4}, {2, 2, 3, 4} }` → 其余元素初始化为零

`int matrix[3][4] = { {1, 2, 3}, {2, 2, 3} }` → 其余元素初始化为零

`int matrix[3][4] = { 1, 2, 3, 4, 2, 2, 3, 4, 3, 2, 3, 4 }` → 不建议省略内层大括号

`int matrix[3][4] = {0};` 所有元素初始化为零

`int matrix[][4] = { {1, 2, 3, 4}, {2, 2, 3, 4}, {3, 2, 3, 4} }` → 由编译器自行推断行的大小

注意事项：不能省略列的大小

② 常量数组

`const int arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};`

→ 表明数组的元素不会发生改变

作用：存放一些静态数据

练习：随机发牌小程序，用户指定发几张牌，程序打印手牌。（52张牌）

1. 如何表示一张手牌？两个属性：花色，大小。

`const char suits[4] = {'s', 'h', 'c', 'd'};`

`const char ranks[13] = {'2', '3', '4', '5', '6', '7', '8', '9', 't', 'j', 'q', 'k', 'a'};`

黑桃 → spade

红桃 → heart

梅花 → club

方块 → diamond

可以用两个数组下标表示一张牌 (suit, rank)

2. 如何生成随机数？

### rand

Defined in header `<stdlib.h>`  
`int rand();`

Returns a pseudo-random integer value between 0 and RAND\_MAX (0 and RAND\_MAX included).

### srand

Defined in header `<stdlib.h>`  
`void srand( unsigned seed );`

Seeds the pseudo-random number generator used by rand() with the value seed.

### time

Defined in header `<time.h>`  
`time_t time( time_t *arg );`

`time_t;`  
`typedef long long time_t`



```
srand((unsigned)time(NULL));  
  
for (int i = 0; i < 10; i++) {  
    printf("%d\n", rand());  
}
```

3. 如何避免生成重复的数?

bool in\_hand[4][13] = {false};