

# C 语言

## 第七讲、分支



张晓平

武汉大学数学与统计学院



2017 年 4 月 5 日

1. if 语句
2. if else 语句
3. 获取逻辑性
4. 一个统计字数的程序
5. 条件运算符
6. continue 和 break 语句

7. switch 语句

8. goto 语句

## 1. if 语句



## if 语句 I

```
1 // colddays.c:
2 #include <stdio.h>
3 int main(void)
4 {
5     const int FREEZING = 0;
6     float temperature;
7     int cold_days = 0;
8     int all_days = 0;
9     printf("Enter the list of daily low
10    temperature.\n");
11    printf("Use Celsius, and enter q to quit.\n");
12    while (scanf("%f", &temperature)==1) {
13        all_days++;
14        if (temperature < FREEZING)
15            cold_days++;
```

## if 语句 II

```
15     }
16     if (all_days != 0)
17         printf("%d days total: %.1f%% below freezing\n",
18               all_days, 100.0*(float)cold_days/
19               all_days);
20     if (all_days == 0)
21         printf("No data entered.\n");
22     return 0;
23 }
```

## if 语句

```
Enter the list of daily low temperature.
```

```
Use Celsius, and enter q to quit.
```

```
-10 -5 0 12 5 6 -4 8 -2 15
```

```
q
```

```
10 days total: 40.0% were below freezing.
```

## if 语句

if 语句被称为分支语句，其一般形式为

```
if (condition)
    statement
```

```
if (condition) {
    statements
}
```

- ▶ 若 condition 的值为真，则执行 statements；否则跳过该语句。
- ▶ if 结构和 while 结构相似，主要区别在于，在 if 结构中，判断和执行仅有一次，而在 while 结构中，判断和执行可以重复多次。



## if 语句

- ▶ condition 是一个关系表达式，通常是比较两个量的大小。更一般地，condition 可以是任何表达式，其值为 0 就被视为假。
- ▶ 语句部分可以是一条简单语句，也可以是一个由花括号括起的复合语句：

```
if (score >= 60)
    printf("Pass!\n");
```

```
if (a > b) {
    a++;
    printf("You lose. b.\n");
}
```



## if else 语句

```
if (condition)
    statement1
else
    statement2
```

## if else 语句

```
if (condition)
    statement1
else
    statement2
```

- ▶ 若满足条件（condition 为真），则执行 statement1；  
若不满足条件（condition 为假），则执行 statement2。
- ▶ 语句可以是简单语句或复合语句。
- ▶ 注意缩进。

## if else 语句

若 if 和 else 之间有多条语句，**必须**使用花括号。

```
// wrong structure
if (x > 0)
    printf("Incrementing x:\n");
    x++;
else
    printf("x <= 0\n");
```

## if else 语句

若 if 和 else 之间有多条语句，**必须**使用花括号。

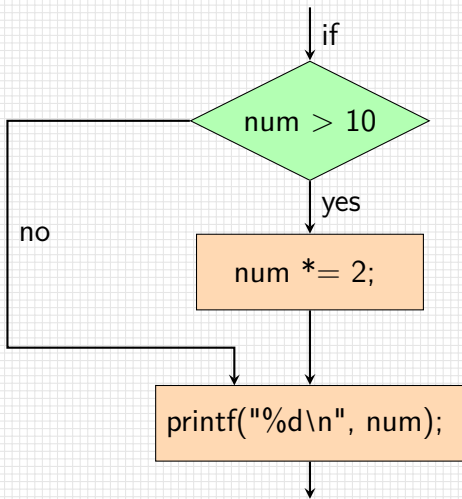
```
// wrong structure
if (x > 0)
    printf("Incrementing x:\n");
    x++;
else
    printf("x <= 0\n");
```

- ▶ 编译器会把 printf 语句看做 if 的一部分，而把 x++; 看做是一条单独的语句，而不是 if 的一部分。
- ▶ 然后认为 else 没有对应的 if，于是报错。

## if else 语句

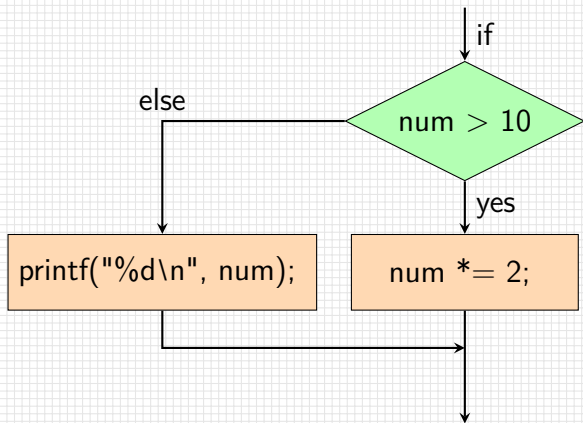
```
// right structure
if (x > 0) {
    printf("Incrementing x:\n");
    x++;
}
else
    printf("x <= 0\n");
```

## if else 语句





## if else 语句



## getchar 与 putchar 函数

- ▶ 函数 `getchar` 没有参数，返回来自输入设备的下一个字符。

```
ch = getchar();           ⇔           scanf("%c", &ch);
```

- ▶ 函数 `putchar` 打印它的参数。

```
putchar(ch);              ⇔           printf("%c", ch);
```

## getchar 与 putchar 函数

- ▶ 只处理字符，比函数 `scanf` 和 `printf` 更快更简洁。
- ▶ 不需要格式说明符。
- ▶ 在 `stdio.h` 中定义。事实上，它们只是宏定义，不是真正的函数。

## getchar 与 putchar 函数

```
1 // cypher1.c
2 #include <stdio.h>
3 #define SPACE ' '
4 int main(void)
5 {
6     char ch;
7     ch = getchar();
8     while (ch != '\n') {
9         if (ch == SPACE)
10             putchar(ch);
11         else
12             putchar(ch+1);
13         ch = getchar();
14     }
15     putchar(ch);
16     return 0;
17 }
```

## getchar 与 putchar 函数

```
1 // cypher1.c
2 #include <stdio.h>
3 #define SPACE ' '
4 int main(void)
5 {
6     char ch;
7     ch = getchar();
8     while (ch != '\n') {
9         if (ch == SPACE)
10             putchar(ch);
11         else
12             putchar(ch+1);
13         ch = getchar();
14     }
15     putchar(ch);
16     return 0;
17 }
```

Hello World  
Ifmmp Xpsme

## getchar 与 putchar 函数

```
ch = getchar();  
while (ch != '\n') {  
    ...  
    ch = getchar();  
}
```

可改写为

```
while ((ch = getchar()) != '\n') {  
    ...  
}
```

## getchar 与 putchar 函数

```
ch = getchar();  
while (ch != '\n') {  
    ...  
    ch = getchar();  
}
```

可改写为

```
while ((ch = getchar()) != '\n') {  
    ...  
}
```

这体现了典型的 C 编程风格：将两个动作合并为一个表达式。

# getchar 与 putchar 函数

更建议写成

```
while (
    (ch = getchar())
    != '\n') {
    ...
}
```



## getchar 与 putchar 函数

- ▶ 两个动作：将某个值赋给 `ch`，并将这个值与换行符作比较。
- ▶ 圆括号使 `ch = getchar()` 称为 `!=` 的左操作数。
- ▶ 先调用函数 `getchar`，将其返回值赋给 `ch`。而赋值表达式的值等于左操作数的值，故 `ch = getchar()` 的值等于 `ch` 的值。
- ▶ 最后将 `ch` 与换行符做比较。

## getchar 与 putchar 函数

圆括号是必须的。若写成

```
while ( ch = getchar() != '\n' ) {  
    ...  
}
```

首先会计算表达式 `getchar() != '\n'`，其值为 0 或 1，然后这个值被赋给 `ch`。于是 `ch` 将会被赋为 0 或 1，而不是 `getchar` 的返回值。

## ctype.h: 字符函数

```
1 // cypher2.c
2 #include <stdio.h>
3 #include <ctype.h>
4 int main(void)
5 {
6     char ch;
7     while ((ch = getchar()) != '\n') {
8         if (isalpha(ch))
9             putchar(ch+1);
10        else
11            putchar(ch);
12    }
13    putchar(ch);
14    return 0;
15 }
```

## ctype.h: 字符函数

```
Look! It's a programmer!  
Mpp1! Ju't b qsphsbnnfs!
```

## ctype.h: 字符函数

表: 字符判断函数

函数名	为如下参数时, 返回值为真
isalnum	字母或数字
isalpha	字母
isblank	标准空白字符 (空格、水平制表符或换行符)
iscntrl	控制符, 如 Ctrl+B
isdigit	阿拉伯数字
isgraph	除空格字符之外的所有可打印字符

## ctype.h: 字符函数

表: 字符判断函数

函数名	为如下参数时，返回值为真
islower	小写字母
isprint	可打印字符
ispunct	标点符号
isspace	空白字符：空格、换行、水平（垂直）制表符、回车
isupper	大写字母
isxdigit	十六进制数字字符

## ctype.h: 字符函数

表: 字符映射函数

函数名	动作
tolower	若参数为大写字母, 则返回相应的小写字母; 否则返回原始参数
toupper	若参数为小写字母, 则返回相应的大写字母; 否则返回原始参数

## ctype.h: 字符函数

字符映射函数不改变原始参数，只返回改变后的值。也就是说，以下语句不改变 `ch` 的值

```
tolower(ch);
```

若想改变 `ch`，可使用

```
ch = tolower(ch);
```



## 多重选择 else if

例 某电力公司的费率如下：

第一个 360kwh	\$0.12589/kwh
下一个 320kwh	\$0.17901/kwh
超过 680kwh	\$0.20971/kwh

编制程序，计算你的用电费用。

## 多重选择 else if I

```
1 // electric.c
2 #include <stdio.h>
3 #define RATE1 0.12589
4 #define RATE2 0.17901
5 #define RATE3 0.20971
6 #define BREAK1 360.0
7 #define BREAK2 680.0
8 #define BASE1 (RATE1 * BREAK1)
9 #define BASE2 (BASE1 + RATE2 * (BREAK2 - BREAK1)
10 )
11 int main(void)
12 {
13     double kwh, bill;
14     printf("Please enter the kwh used.\n");
15     scanf("%lf", &kwh);
```

## 多重选择 else if II

```
15  if (kwh <= BREAK1)
16      bill = RATE1 * kwh;
17  else if (kwh <= BREAK2)
18      bill = BASE1 + RATE2 * (kwh - BREAK1);
19  else
20      bill = BASE2 + RATE3 * (kwh - BREAK2);
21  printf("The charge for %.1f kwh is $%.2f.\n",
22         kwh, bill);
23  return 0;
24 }
```

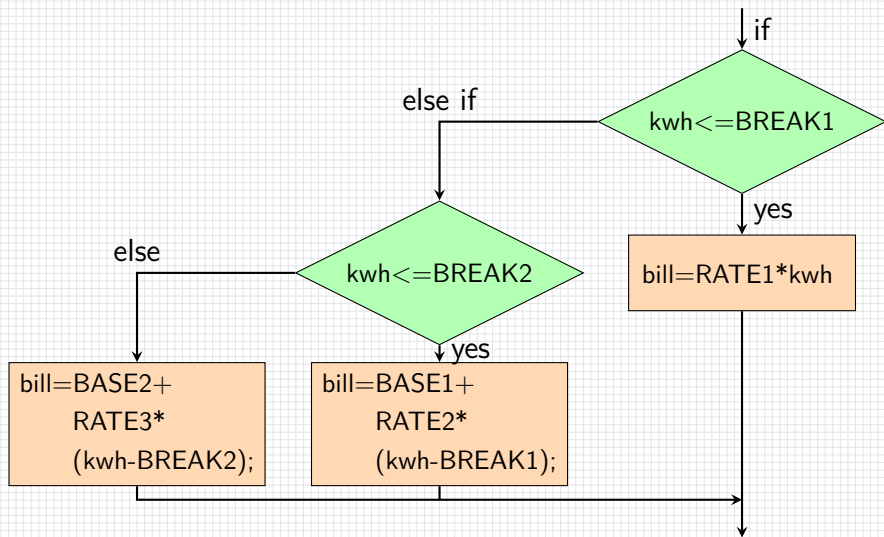
## 多重选择 else if

```
Please enter the kwh used.
```

```
580
```

```
The charge for 580.0 kwh is $84.70.
```

## 多重选择 else if



## 多重选择 else if

```
if (kwh <= BREAK1)
    bill = RATE1 * kwh;
else if (kwh <= BREAK2)
    bill = BASE1 + RATE2 * (kwh - BREAK1);
else
    bill = BASE2 + RATE3 * (kwh - BREAK2);
```

等价于

```
if (kwh <= BREAK1)
    bill = RATE1 * kwh;
else
    if (kwh <= BREAK2)
        bill = BASE1 + RATE2 * (kwh - BREAK1);
    else
        bill = BASE2 + RATE3 * (kwh - BREAK2);
```

## 多重选择 else if

- ▶ 第二种形式是 if else 语句的嵌套。因整个 if else 结构是一条语句，故第一个 else 后面不需要用花括号。
- ▶ 虽然两种形式完全等价，但建议采用第一种形式，它可以更清晰地展示出有三种选择。

## 多重选择 else if

可以把多个所需的 else if 语句连成一串使用。

```
if (score < 1000)
    bonus = 0;
else if (score < 1500)
    bonus = 1;
else if (score < 2000)
    bonus = 2;
else if (score < 2500)
    bonus = 3;
else
    bonus = 4;
```

编译器对嵌套层数有限制，C99 标准要求编译器最少支持 127 层嵌套。



## else 与 if 的配对

```
1 // elseif.c:
2 #include <stdio.h>
3 int main(void)
4 {
5     int number;
6     printf("Enter an integer: ");
7     scanf("%d", &number);
8     if (number > 6)
9         if (number < 12)
10             printf("You're close!\n");
11     else
12         printf("Sorry, you loose a turn!\n");
13     return 0;
14 }
```

## else 与 if 的配对

```
Enter an integer: 5
```

## else 与 if 的配对

```
Enter an integer: 5
```

```
Enter an integer: 10  
You're close!
```

## else 与 if 的配对

```
Enter an integer: 5
```

```
Enter an integer: 10  
You're close!
```

```
Enter an integer: 15  
Sorry, you loose a turn!
```

## else 与 if 的配对

### 规则

如果没有花括号，else 与和它最近的一个 if 相匹配。

## else 与 if 的配对

上例最好改写为

```
if (number > 6)
    if (number < 12)
        printf("You're close!\n");
    else
        printf("Sorry, you loose a turn!\n");
```

## else 与 if 的配对

若真的希望 else 和第一个 if 匹配，请写成

```
if (number > 6) {  
    if (number < 12)  
        printf("You're close!\n");  
}  
else  
    printf("Sorry, you loose a turn!\n");
```

## 多层嵌套的分支结构

**例** 编写程序，由用户输入一个整数，然后判断其是否为质数。如果不是质数，请求出其公约数。



# 多层嵌套的分支结构 I

```
1 #include <stdio.h>
2 #include <stdbool.h>
3 int main(void)
4 {
5     unsigned long num;
6     unsigned long div;
7     bool isPrime;
8
9     printf("Enter an integer ");
10    printf("(Enter q to quit).\n");
11    while (scanf("%lu", &num) == 1) {
12        for (div = 2, isPrime = true;
13             div * div <= num;
14             div++) {
15            if (num % div == 0) {
```

## 多层嵌套的分支结构 II

```
16         if (div * div != num)
17             printf("%lu is divisible by %lu and %
                lu.\n",
18                 num, div, num / div);
19         else
20             printf("%lu is divisible by %lu.\n",
21                 num, div);
22         isPrime = false;
23     }
24 }
25 if (isPrime)
26     printf("%lu is prime.\n", num);
27 printf("Enter another integer ");
28 printf("(Enter q to quit).\n");
29 }
30 printf("Bye.\n");
```

## 多层嵌套的分支结构 III

```
31 |  
32 |   return 0;  
33 | }
```

## 多层嵌套的分支结构 I

```
Enter an integer (Enter q to quit).  
36  
36 is divisible by 2 and 18.  
36 is divisible by 3 and 12.  
36 is divisible by 4 and 9.  
36 is divisible by 6.  
Enter another integer (Enter q to quit).  
149  
149 is prime.  
Enter another integer (Enter q to quit).  
30777  
30777 is divisible by 3 and 10259.  
Enter another integer (Enter q to quit).  
q  
Bye.
```

### 3. 获取逻辑性



## 获取逻辑性

**例** 编写程序，首先输入一个句子，然后计算除单引号和双引号意外的字符出现的次数。

# 获取逻辑性

```
1 // chcount.c:
2 #include <stdio.h>
3 #define PERIOD '.'
4 int main(void)
5 {
6     int ch;
7     int charcount = 0;
8     while ((ch = getchar()) != PERIOD) {
9         if (ch != '"' && ch != '\\')
10             charcount++;
11     }
12     printf("There are %d non-quote characters.\n",
13           charcount);
13     return 0;
14 }
```

## 获取逻辑性

```
"I'm fine".
```

```
There are 7 non-quote characters.
```



## 获取逻辑性

- ▶ 首先，程序读入一个字符并检查它是不是一个句号。
- ▶ 接下来的语句中，使用了逻辑与运算符 `&&`。  
此时，`if` 语句的含义为“若字符不是双引号也不是单引号，则 `charcount` 增加 1”。
- ▶ 要使整个表达式为真，则两个条件都必须为真。逻辑运算符的优先级低于关系运算符，故不必使用圆括号。

# 获取逻辑性

表: 逻辑运算符

运算符	含义
& &	与
	或
!	非

## 获取逻辑性

设 `exp1` 和 `exp2` 为两个简单的关系表达式，则

- ▶ 仅当 `exp1` 和 `exp2` 都为真时，`exp1 && exp2` 才为真。
- ▶ 若 `exp1` 或 `exp2` 为真或二者都为真，`exp1 || exp2` 为真。
- ▶ 若 `exp1` 为假，则 `!exp1` 为真；若 `exp1` 为真，则 `!exp1` 为假。

# 获取逻辑性

请判断以下表达式的值。

- ▶ `5 > 2 && 4 > 7`
- ▶ `5 > 2 || 4 > 7。`
- ▶ `! (4 > 7)`

## 头文件 `iso646.h`

- ▶ C99 标准为逻辑运算符增加了可供选择的拼写法，它们在头文件 `iso646.h` 中定义。
- ▶ 若包含了该头文件，可用 `and` 代替 `&&`，用 `or` 代替 `||`，用 `not` 代替 `!`。

## 头文件 iso646.h

若包含了头文件 iso646.h, 则

```
if (ch != '"' && ch != '\\')  
    charcount++;
```

可重写为

```
if (ch != '"' and ch != '\\')  
    charcount++;
```

表: 逻辑运算符的可选表示法

传统用法	iso646.h
&&	and
	or
!	not

# 优先级

- ▶ 逻辑非运算符！为单目运算符，优先级同增量运算符相同，仅次于圆括号。
- ▶ && 的优先级高于 ||，两者的优先级都低于关系运算符，高于赋值运算符。

如

`a > b && b > c || b > d`

会被视为

`((a > b) && (b > c)) || (b > d)`



# 求值顺序

- 除了那些两个运算符共享一个操作数的情况外，C 通常不保证复杂表达式的哪个部分首先被求值。

如以下语句中

```
b = (5 + 3) * (9 + 6)
```

可能先计算  $5 + 3$  的值，也可能先计算  $9 + 6$  的值。

- C 允许这种不确定性，以便编译器设计者可以针对特定系统做出最有效率的选择。

# 求值顺序

- ▶ 但对逻辑运算符的处理是个例外，C 保证逻辑表达式是从左到右求值的。
- ▶ `&&` 和 `||` 是顺序点，故在程序从一个操作数前进到下一个操作数之前，所有副作用都会生效。
- ▶ C 保证一旦发现某个元素使表达式总体无效，求值会立即停止。

# 求值顺序

```
while ((c = getchar()) != ' ' && c != '\n')
```

---

## 求值顺序

```
while ((c = getchar()) != ' ' && c != '\n')
```

---

- ▶ 该结构用于循环读入字符，直到出现第一个空格符或换行符。
- ▶ 第一个子表达式给 `c` 赋值，然后该值用于第二个子表达式中。
- ▶ 若没有顺序保障，计算机可能试图在 `c` 被赋值之前判断第二个表达式。

# 求值顺序

```
while (x++ < 10 && x + y < 20)
```

---

# 求值顺序

```
while (x++ < 10 && x + y < 20)
```

---

&& 是顺序点，故保证了在对右边表达式求值之前，先把 x 的值增加 1。

## 求值顺序

```
if (number != 0 && 12/number == 2)
    printf("The number is 5 or 6.\n");
```

---

## 求值顺序

```
if (number != 0 && 12/number == 2)
    printf("The number is 5 or 6.\n");
```

---

若 number 值为 0，则第一个表达式为假，就不再对关系表达式求值。这就避免了计算机试图把 0 作为除数。



## 范围

可把 `&&` 用于测试范围。如要检查 90 到 100 范围内的得分，可以这样做

```
if (score >= 90 && score <= 100)
    printf("Excellent!\n");
```

## 范围

可把 `&&` 用于测试范围。如要检查 90 到 100 范围内的得分，可以这样做

```
if (score >= 90 && score <= 100)
    printf("Excellent!\n");
```

## 范围

请避免以下做法：

```
if (90 <= score <= 100)
    printf("Excellent!\n");
```

---

## 范围

请避免以下做法：

```
if (90 <= score <= 100)
    printf("Excellent!\n");
```

---

这段代码没有语法错误，但有语义错误。因对  $\leq$  运算符的求值顺序是从左到右的，故测试表达式会被解释为

$$(90 \leq \text{score}) \leq 100$$

而子表达式  $90 \leq \text{score}$  的值为 1 或 0，总小于 100。故不管  $\text{range}$  取何值，整个表达式总为真。

[illegible]

# 一个统计字数的程序

编制程序，读取一段文字，并报告其中的单词个数，同时统计字符个数和行数。

## 一个统计字数的程序

- ▶ 该程序应该逐个读取字符，并想办法判断何时停止。
- ▶ 应该能够识别并统计字符、行和单词。

# 一个统计字数的程序

```
1 // pseudo code
2 read a character
3 while there is more input
4     increment character count
5     if a line has been read, increment line
      count
6     if a word has been read, increment word
      count
7     read next character
```



# 一个统计字数的程序

```
// 循环输入结构
while ((ch = getchar()) != STOP)
{
    ...
}
```

---

# 一个统计字数的程序

```
// 循环输入结构
while ((ch = getchar()) != STOP)
{
    ...
}
```

---

在通用的单词统计程序中，换行符和句号都不适合标记一段文字的结束。我们将采用一个不常见的字符 |。

## 一个统计字数的程序

- ▶ 程序使用 `getchar` 来循环输入字符，可在每次循环通过递增一个**字符计数器**的值来统计字符。
- ▶ 为统计行数，程序可检查换行符。若字符为换行符，程序就递增**行数计数器**的值。若 `STOP` 字符出现在一行的中间，则将该行作为一个**不完整行**来统计，即该行有字符但没有换行符。

# 一个统计字数的程序

如何识别单词？

# 一个统计字数的程序

## 如何识别单词？

- ▶ 可将一个单词定义为不包含空白字符的一系列字符。
- ▶ 一个单词以首次遇到非空白字符开始，在下一个空白字符出现时结束。

## 一个统计字数的程序

- ▶ 检测非空白字符的判断表达式为

```
c != ' ' && c != '\n' && c != '\t'
```

或

```
!isspace(c) // #include <ctype.h>
```

- ▶ 检测空白字符的判断表达式为

```
c == ' ' || c == '\n' || c == '\t'
```

或

```
isspace(c) // #include <ctype.h>
```

## 一个统计字数的程序

- ▶ 为了判断一个字符是否在某个单词中，可在读入一个单词的首字符时把一个标志 (命名为 inword) 设置为 1，同时在此处递增单词个数。
- ▶ 只要 inword 为 1，后续的非空白字符就不标记为一个单词的开始。到出现下一个空白字符时，就把 inword 设置为 0。

```
1 // pseudo code
2   if c is not a whitespace and inword is false
3       set inword to true  and count the word
4   if c is a white space and inword is true
5       set inword to false
```

# 一个统计字数的程序 I

```
1 // wordcnt.c:
2 #include <stdio.h>
3 #include <ctype.h>
4 #include <stdbool.h>
5 #define STOP '|'
6 int main(void)
7 {
8     char c;
9     char prev;
10    long n_chars = 0L;
11    int n_lines = 0;
12    int n_words = 0;
13    int p_lines = 0;
14    bool inword = false;
15    printf("Enter text (| to quit):\n");
```



## 一个统计字数的程序 II

```
16 prev = '\n';
17 while ((c = getchar()) != STOP) {
18     n_chars++;
19     if (c == '\n')
20         n_lines++;
21     if (!isspace(c) && !inword) {
22         inword = true;
23         n_words++;
24     }
25     if (isspace(c) && inword)
26         inword = false;
27     prev = c;
28 }
29 if (prev != '\n')
30     p_lines = 1;
```

## 一个统计字数的程序 III

```
31 | printf("characters = %ld, words = %d, lines =  
   | %d, ", n_chars, n_words, n_lines);  
32 | printf("partial lines = %d\n", p_lines);  
33 | return 0;  
34 | }
```

# 一个统计字数的程序

```
Enter text (| to quit):
```

```
Reason is a
```

```
powerful servant but
```

```
an inadequate master.
```

```
|
```

```
characters = 56, words = 9, lines = 3, partial  
lines = 0
```

[illegible]

# 条件运算符

C 提供一种简写方式来表示 `if else` 语句，被称为条件表达式，并使用条件运算符 (`?:`)。它是 C 语言中唯一的三目操作符。

# 条件运算符

求绝对值

```
x = (y < 0) ? -y : y;
```

---

# 条件运算符

求绝对值

```
x = (y < 0) ? -y : y;
```

---

- ▶ 含义：若  $y$  小于 0，则  $x = -y$ ；否则， $x = y$ 。
- ▶ 用 if else 描述为

```
if (y < 0)
    x = -y;
else
    x = y;
```

# 条件运算符

## 条件表达式的语法

`expresion1 ? expression2 : expression3`

---



# 条件运算符

## 条件表达式的语法

`expresion1 ? expression2 : expression3`

---

若 `expresion1` 为真，则条件表达式的值等于 `expression2` 的值；  
若 `expresion1` 为假，则条件表达式的值等于 `expression3` 的值。

# 条件运算符

若希望将两个可能的值中的一个赋给变量时，可使用条件表达式。典型的例子是将两个值中的最大值赋给变量：

```
max = (a > b) ? a : b;
```

# 条件运算符

if else 语句能完成与条件运算符同样的功能。但是，条件运算符语句更简洁；并且可以产生更精简的程序代码。

# 条件运算符

**例** 设每罐油漆可喷 200 平方英尺，编写程序计算向给定的面积喷油漆，全部喷完需要多少罐油漆。

# 条件运算符

```
1 // paint.c:
2 #include <stdio.h>
3 #define COVERAGE 200
4 int main(void)
5 {
6     int sq_feet, cans;
7     printf("Enter number of square feet to be
8     painted:\n");
9     while (scanf("%d", &sq_feet)) {
10         cans = sq_feet / COVERAGE;
11         cans += (sq_feet % COVERAGE == 0) ? 0 : 1;
12         printf("You need %d %s of paint.\n",
13             cans, cans == 1 ? "can" : "cans");
14         printf("Enter next value (q to quit):\n");
15     }
16     return 0;
17 }
```

# 条件运算符

```
Enter number of square feet to be painted:  
200  
You need 1 can of paint.  
Enter next value (q to quit):  
225  
You need 2 cans of paint.  
Enter next value (q to quit):  
q
```

[illegible]

## continue 和 break 语句

continue 和 break 语句用于循环结构，根据判断条件来忽略部分循环甚至终止循环。



## continue 和 break 语句：continue 语句

- ▶ 当程序运行到 continue 语句时，其后的内容将被忽略，开始进入下一次循环。
- ▶ 当 continue 语句用于嵌套结构时，仅影响包含它的那一层循环。

## continue 和 break 语句：continue 语句

**例** 输入 1-100 之间的多个分数，求其平均分、最低分和最高分。当输入分数不在 1-100 之间时，程序应该不做处理。

## continue 和 break 语句: continue 语句 I

```
1 #include <stdio.h>
2 int main(void)
3 {
4     const float MIN = 0.0f;
5     const float MAX = 100.0f;
6
7     float score;
8     float total = 0.0f;
9     int n = 0;
10    float min = MIN;
11    float max = MAX;
12
13    printf("Enter the first score (q to quit): ");
14    while (scanf("%f", &score) == 1)
15    {
```

## continue 和 break 语句: continue 语句 II

```
16     if (score < MIN || score > MAX)
17     {
18         printf("%.1f is invalid.Try again: ",
19             score);
19         continue;
20     }
21     printf("Accepting %.1f:\n", score);
22     min = (score < min) ? score : min;
23     max = (score > min) ? score : max;
24     total += score;
25     n++;
26     printf("Enter next score (q to quit): ");
27 }
28 if (n > 0)
29 {
```

## continue 和 break 语句: continue 语句 III

```
30     printf("Average of %d scores is %.1f.\n", n,  
    total/n);  
31     printf("Low = %.1f, High = %.1f.\n", min,  
    max);  
32 }  
33 else  
34     printf("No valid scores were entered.\n");  
35  
36     return 0;  
37 }
```

## continue 和 break 语句: continue 语句

```
Enter the first score (q to quit): 20
Accepting 20.0:
Enter next score (q to quit): -1
-1.0 is invalid. Try again: 30
Accepting 30.0:
Enter next score (q to quit): 110
110.0 is invalid. Try again: q
Average of 2 scores is 25.0.
Low = 0.0, High = 30.0.
```

## continue 和 break 语句：continue 语句

- ▶ 对于 while 和 do while 循环，continue 语句之后发生的动作是求循环表达式的值。
- ▶ 而对于 for 循环，下一个动作是先求更新表达式的值，然后再求判断表达式的值。

## continue 和 break 语句: continue 语句

```
count = 0;
while (count < 10)
{
    ch = getchar();
    if (ch == '\n')
        continue;
    putchar(ch);
    count++;
}
```

读取除换行符外的 10 个字符，并回显它们。注意：换行符不会被计数。



## continue 和 break 语句: continue 语句

```
for (count = 0; count < 10; count++)  
{  
    ch = getchar();  
    if (ch == '\n')  
        continue;  
    putchar(ch);  
}
```

读取包含换行符在内的 10 个字符，换行符不被回显，但会被计数。

## continue 和 break 语句：break 语句

- ▶ 当程序运行到 break 语句时，将会终止包含它的循环，跳出该循环体。
- ▶ 当 break 语句用于嵌套结构时，仅影响包含它的那一层循环。

## continue 和 break 语句：break 语句

**例** 输入矩形的长和宽，用一个循环来计算其面积。若输入一个非数字作为矩形的长或宽，终止循环。

## continue 和 break 语句: break 语句 I

```
1 #include <stdio.h>
2 int main(void)
3 {
4     float length, width;
5     printf("Enter the length of the rectangle: ");
6
7     while (scanf("%f", &length) == 1) {
8         printf("Length = %.2f.\n", length);
9         printf("Enter its width: ");
10        if (scanf("%f", &width) != 1)
11            break;
12        printf("Width = %.2f;\n", width);
13        printf("Area = %.2f; \n", length * width);
14        printf("Enter the length of the rectangle: "
15              );
```

## continue 和 break 语句: break 语句 II

```
15     }  
16     printf("Done.\n");  
17     return 0;  
18 }
```

## continue 和 break 语句: continue 语句

```
Enter the length of the rectangle: 10
Length = 10.00.
Enter its width: 20
Width = 20.00;
Area = 200.00;
Enter the length of the rectangle: 10
Length = 10.00.
Enter its width: q
Done.
```

## continue 和 break 语句：break 语句

- ▶ break 语句使程序直接跳转到该循环后的第一条语句；在 for 循环中，更新表达式也将被跳过。
- ▶ 嵌套循环中，break 语句只能使程序跳出当前循环，要跳出外层循环还需另外一个 break 语句。

## continue 和 break 语句: break 语句

```
int p, q;
scanf("%d", &p);
while (p > 0) {
    printf("%d\n", p);
    scanf("%d", &q);
    while (q > 0) {
        printf("%d\n", p*q);
        if (q > 100)
            break;
        scanf("%d", &q);
    }
    if (q > 100)
        break;
    scanf("%d", &p);
}
```





## switch 语句

多重选择时，可以使用

```
if (condition1)
    ...
else if (condition2)
    ...
else if (condition3)
    ...
else
```

但多数情况下，使用 switch 语句会更加方便。

## switch 语句 I

```
1 // animals.c
2 #include <stdio.h>
3 #include <ctype.h>
4 int main(void)
5 {
6     char ch;
7     printf("Give a letter, and I will give you an
8 ");
9     printf("animal name beginning with that letter
10 .\n");
11     printf("Please type in a letter: # to quit.\n"
12 );
13     while ((ch = getchar()) != '#')
14     {
15         if ('\n' == ch)
```

## switch 语句 II

```
13         continue;
14     if (islower(ch))
15     {
16         switch (ch)
17         {
18             case 'a':
19                 printf("alligator\n");
20                 break;
21             case 'b':
22                 printf("buffalo\n");
23                 break;
24             case 'c':
25                 printf("camel\n");
26                 break;
27             case 'd':
28                 printf("dove\n");
```

## switch 语句 III

```
29         break;
30     case 'e':
31         printf("eagle\n");
32         break;
33     default:
34         break;
35     }
36 }
37 else
38     printf("I only recognize only lowercase
39     letters.\n");
40 while (getchar() != '\n')
41     continue;
42 printf("Please typer another letter or a #.\n
43 n");
44 }
```

## switch 语句 IV

```
43 | printf("Bye!\n");  
44 | return 0;  
45 | }
```

## switch 语句

```
Give a letter, and I will give you an animal  
name beginning with that letter.
```

```
Please type in a letter: # to quit.
```

```
dog
```

```
dove
```

```
Please typer another letter or a #.
```

```
a
```

```
alligator
```

```
Please typer another letter or a #.
```

```
eff
```

```
eagle
```

```
Please typer another letter or a #.
```

```
#
```

```
Bye!
```

# switch 语句

```
// switch语法
switch (integer expression)
{
    case constant1:
        statements
    case constant2:
        statements
    ...
    default:
        statements
}
```



## switch 语句

- 1、判断表达式应该具有整数值，包括 int, char 和 enum 类型。

## switch 语句

```
1 // switch1.c: float is not allowed in switch
2 #include <stdio.h>
3 int main(void)
4 {
5     float x = 1.1;
6     switch (x) {
7         case 1.1: printf("Choice is 1");
8             break;
9         default: printf("Choice other than 1, 2 and 3"
10             );
11             break;
12     }
13     return 0;
14 // Compiler Error: switch quantity not an
15 integer
```

## switch 语句

2、break 使程序跳出 switch 结构，执行 switch 之后的下一条语句。若没有 break 语句，从相匹配的标签到 switch 末尾的每一条语句都会被执行。

## switch 语句

```
1 // switch2.c: There is no break in all cases
2 #include <stdio.h>
3 int main()
4 {
5     int x = 2;
6     switch (x) {
7         case 1: printf("Choice is 1\n");
8         case 2: printf("Choice is 2\n");
9         case 3: printf("Choice is 3\n");
10        default: printf("Choice other than 1, 2 and
11                  3\n");
12    }
13    return 0;
14 }
```

## switch 语句

```
Choice is 2  
Choice is 3  
Choice other than 1, 2 and 3
```

## switch 语句

```
1 // switch3.c: There is no break in some cases
2 #include <stdio.h>
3 int main()
4 {
5     int x = 2;
6     switch (x) {
7         case 1: printf("Choice is 1\n");
8         case 2: printf("Choice is 2\n");
9         case 3: printf("Choice is 3\n");
10        case 4: printf("Choice is 4\n"); break;
11        default: printf("Choice other than 1, 2, 3 and
12                    4\n"); break;
13    }
14    printf("After Switch");
15    return 0;
16 }
```

# switch 语句

```
Choice is 2  
Choice is 3  
Choice is 4  
After Switch
```

## switch 语句

3、case 标签必须是整型常量或整型常量表达式，不能用变量作为 case 标签。



## switch 语句

```
1 // switch4.c: A program with variable
  expressions in labels
2 #include <stdio.h>
3 int main()
4 {
5     int x = 2;
6     int arr[] = {1, 2, 3};
7     switch (x) {
8         case arr[0]: printf("Choice 1\n");
9         case arr[1]: printf("Choice 2\n");
10        case arr[2]: printf("Choice 3\n");
11    }
12    return 0;
13 }
14 // Compiler Error: case label does not reduce to
    an integer constant
```

## switch 语句

4、default 语句块可放在 switch 结构中的任意位置，若判断表达式与标签均不匹配，它会被执行。

## switch 语句

```
1 // switch5.c: The default block is placed above
  other cases.
2 #include <stdio.h>
3 int main()
4 {
5     int x = 4;
6     switch (x) {
7         default: printf("Choice other than 1 and 2");
8                 break;
9         case 1: printf("Choice is 1");    break;
10        case 2: printf("Choice is 2");    break;
11    }
12    return 0;
13 }
```

## switch 语句

```
1 // switch5.c: The default block is placed above
  other cases.
2 #include <stdio.h>
3 int main()
4 {
5     int x = 4;
6     switch (x) {
7         default: printf("Choice other than 1 and 2");
8                 break;
9         case 1: printf("Choice is 1");    break;
10        case 2: printf("Choice is 2");    break;
11    }
12    return 0;
13 }
```

Choice other than 1 and 2

## switch 语句 I

5、case 之前的语句不会被执行。一旦进入 switch 结构，将直接转入标签匹配。

## switch 语句 I

```
1 // Statements before all cases are never
  executed
2 #include <stdio.h>
3 int main()
4 {
5     int x = 1;
6     switch (x) {
7         x = x + 1; // This statement is not
          executed
8     case 1: printf("Choice is 1");
9         break;
10    case 2: printf("Choice is 2");
11        break;
12    default: printf("Choice other than 1 and 2");
13        break;
```

## switch 语句 II

```
14     }  
15     return 0;  
16 }
```

Choice is 1

## switch 语句 I

6、两个 case 标签不能有相同值。



## switch 语句 I

```
1 // switch7.c: Program where two case labels have
  same value
2 #include <stdio.h>
3 int main()
4 {
5     int x = 1;
6     switch (x) {
7         case 2: printf("Choice is 1");
8             break;
9         case 1+1: printf("Choice is 2");
10            break;
11     }
12     return 0;
13 }
14 // Compiler Error: duplicate case value
```

## switch 语句

**例** 编制程序，输入一段文字，按 # 停止输入，然后统计该段文字中字母 a, e, i, o, u 出现的次数（不计大小写）。

## switch 语句 I

```
1 // vowels.c:
2 #include <stdio.h>
3 int main(void)
4 {
5     char ch;
6     int na, ne, ni, no, nu;
7     na = ne = ni = no = nu = 0;
8     printf("Enter some text: enter # to quit.\n");
9     while ((ch = getchar()) != '#') {
10         switch (ch) {
11             case 'a':
12             case 'A': na++;
13                 break;
14             case 'e':
15             case 'E': ne++;
```

## switch 语句 II

```
16         break;
17     case 'i':
18     case 'I': ni++;
19         break;
20     case 'o':
21     case 'O': no++;
22         break;
23     case 'u':
24     case 'U': nu++;
25         break;
26     default:
27         break;
28     }
29 }
30 printf("Number of text: %4c %4c %4c %4c\n", 'A',
        'E', 'I', 'U');
```

## switch 语句 III

```
31 | printf("          %4d %4d %4d %4d\n", na  
   | , ne, ni, nu);  
32 | return 0;  
33 | }
```

## switch 语句

```
Enter some text: enter # to quit.
```

```
See you tommorrow!#
```

```
Number of text:      A      E      I      U  
                   0      2      0      1
```

## switch 语句

- ▶ 若输入字母为 i，则 switch 语句定位到标签为 case 'i': 的位置。因没有 break 同该标签相关联，故程序将前进到下一条语句，即 ni++。
- ▶ 若输入字母为 l，程序将直接定位到这条语句。
- ▶ 本质上，两个标签都指向相同的语句。

## switch 语句

在该例中，可通过 `ctype.h` 中的 `toupper` 函数在进行判断之前将所有的字母转换为大写字母以避免多重标签。



## switch 语句

```
ch = toupper(ch);  
switch (ch) {  
    case 'A':  na++;  
        break;  
    case 'E':  ne++;  
        break;  
    case 'I':  ni++;  
        break;  
    case 'O':  no++;  
        break;  
    case 'U':  nu++;  
        break;  
    default:  
        break;  
}
```

## switch 语句

若希望保留 ch 的值不变，可以这么做

```
switch (toupper(ch)) {  
    case 'A': na++;  
        break;  
    case 'E': ne++;  
        break;  
    case 'I': ni++;  
        break;  
    case 'O': no++;  
        break;  
    case 'U': nu++;  
        break;  
    default:  
        break;  
}
```

## switch 语句: switch 与 if else

- ▶ 若选择是基于求一个浮点型变量或表达式的值, 就不能使用 switch。
- ▶ 若变量必须落入某个范围, 使用 if 语句会更方便。如  

```
if (integer < 1000 && integer > 2)
```
- ▶ 若可以使用 switch, 程序会运行得稍快些, 并且代码会更紧凑。

[illegible]

## goto 语句

```
// goto 语法  
goto label;  
...  
label: printf("Refined analysis.\n");
```

- ▶ goto 语句包括两个部分：goto 和一个标签。标签的命名方式与变量命名相同。
- ▶ 必须包含由标签定位的其它语句：标签名后紧跟一个冒号，然后是一条语句。

## goto 语句

goto 语句非常容易被滥用，建议谨慎使用，或者根本不用。

## goto 语句

```
if (size > 12)
    goto a;
goto b;
a: cost *= 1.05;
    flag = 2;
b: bill = cost * flag;
```

等效于

```
if (size > 12)
{
    cost *= 1.05;
    flag = 2;
}
bill = cost * flag;
```

## goto 语句

```
if (n > 14)
    goto a;
m = 2;
goto b;
a: m = 3;
b: k = 2 * m;
```

等效于

```
if (size > 12)
    m = 2;
else
    n = 3;
k = 2 * m;
```



## goto 语句

```
readin: scanf("%d", &score);  
if (score < 0)  
    goto stage2;  
lots of statements;  
goto readin;  
stage2: morestuff;
```

### 等效于

```
scanf("%d", &score);  
while (score >= 0)  
{  
    lots of statements;  
    scanf("%d", &score);  
}  
more stuff;
```

## goto 语句

- ▶ 调到循环末尾并开始下一轮循环，用 `continue` 代替。
- ▶ 跳出循环，用 `break` 代替。
- ▶ 事实上，`break` 和 `continue` 是 `goto` 的特殊形式。使用它们的好处是其名称表明会完成什么动作；并且不需要标签，故不存在错放标签位置的潜在危险。