概览

使用C的理由

1. 设计特性(自顶而下,结构化编程)编写程序更可靠/易懂
2. 高效性(只有汇编能与之比拟)
3. 可移植性
4. 强大而灵活(可解决物理工程问题及为电影制作特效)
5. 对于编程者友好(语法较灵活,可访问硬件)

使用C的七个步骤

1. 定义程序目标(程序需要的信息、需要进行的计算和操作、程序需要报告的信息，用一般概念思考问题)
2. 设计程序（用一般概念思考问题，而非考虑具体代码，可用伪代码、程序流程图）
3. 编写代码（注意加入注释）
4. 编译（将源代码转换为可执行代码（.obj)，即计算机本机语言或机器语言表示的代码）
5. 运行程序(.obj->exe.,可以直接键入可执行文件名即可运行)
6. 测试和调试程序(**后面重点学习)**
7. 维护和修改程序(如果前期采用了清楚地文字注释和良好的设计风格可简化这一步骤)

编程机制

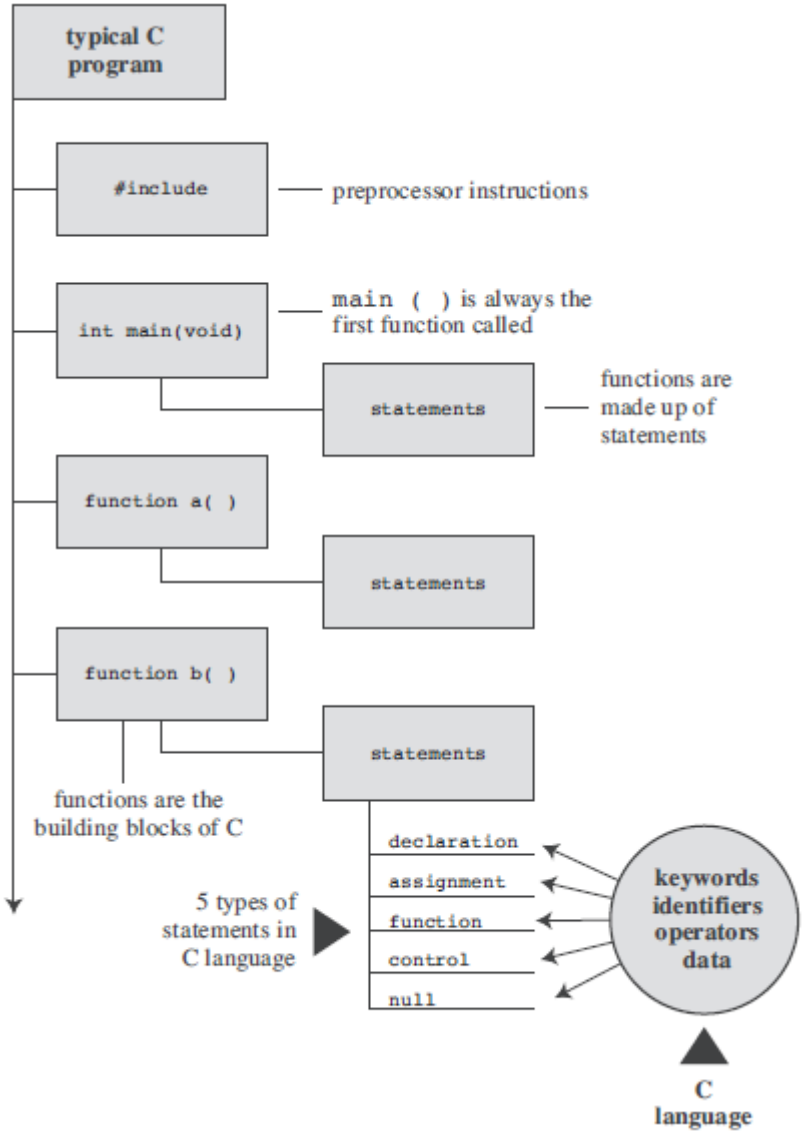
编译与链接

语言标准

短小、清楚、高效

C语言概述

典型C程序




```c
// first.c
#include <stdio.h>
int main(void)
/* a simple program */
{
  int num;                  /* define a variable called num* /
  num = 1;                  /* assign a value to num        */

  printf("I am a simple "); /* use the printf() function    */
  printf("computer.\n");
  printf("My favorite number is %d because it is first.\n",num);
  return 0;
}
```

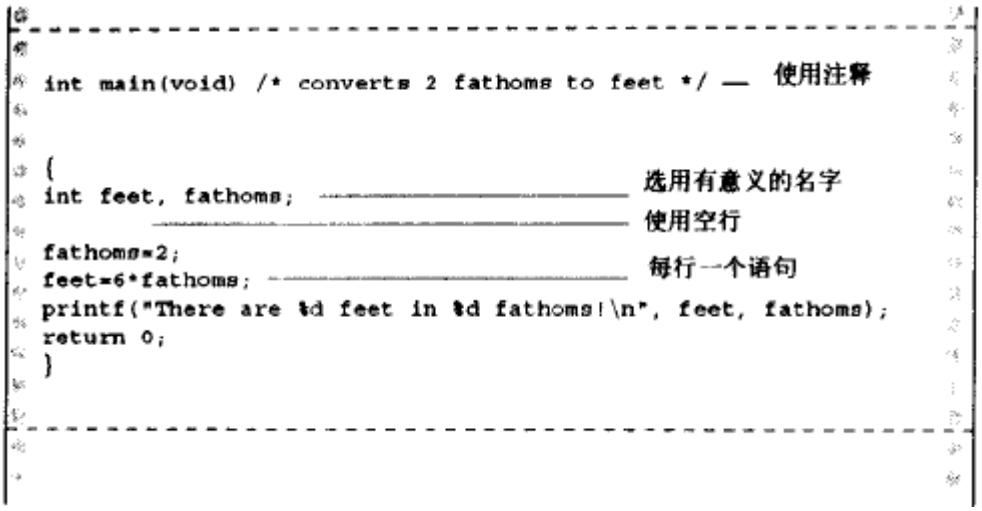
- 程序开始处用注释说明文件的名称和程序的目的
- `#include`指示和头文件(可单独放在一个文件中)
- `main()`函数(一种特殊的函数,可被其他函数调用,最先被执行)
- 注释:块注释`/*...*/`+行注释`//`(为了更容易理解程序,方便未来的维护)

- 花括号、程序体和代码快:花括号规定了函数的界限
- 声明（注意函数声明必须放在函数调用之前）
  - 数据类型：声明数据类型就意味着分配了存储空间
  - 变量名：只能含小写字母、大写字母、数字和下划线，第一个字母必须是字母或下划线，不能使用关键字，最好取有意义的名字
- 赋值：在一个内存区间段选一个特定地址，后面还可以重新选择（即重新赋值）
- `printf()`函数:格式化输出,在头文件中已包含它的声明

一个简单程序的结构

```
#include<stdio.h>
int main(void)
{
  statements
  return 0;
}
```

使程序更可读的技巧



多个函数

```
//* two_func.c -- a program using two functions in one file */
#include <stdio.h>
void butler(void);      /* ANSI/ISO C function prototyping */
int main(void)
{
    printf("I will summon the butler function.\n");
    butler();
```

```
    printf("Yes. Bring me some tea and writeable DVDs.\n");

    return 0;
}

void butler(void)       /* start of function definition */
{
    printf("You rang, sir?\n");
}
```

- 所有的函数都是从`main()`函数开始执行

调试

语法错误

```
/*  nogood.c -- a program with errors */
#include <stdio.h>
int main(void)
(
 int n, int n2, int n3;

/* this program has several errors
 n = 5;
 n2 = n * n;
 n3 = n2 * n2;
 printf("n = %d, n squared = %d, n cubed = %d\n", n, n2, n3)

 return 0;
 )
```

如何检测程序语法错误:

1. 编译前浏览程序源代码看是否有明显错误;
2. 查看由编译器发现的错误;
3. 编译器有时会出错.
    - 某位置上一个真的的语法错误可能导致编译器误认为它发现了其他错误,不用修改所有报错,只需修改前几个,再重新编译;
    - 编译器往往发现的错误位置比真正的错误要滞后一行.

语义错误

```
/* stillbad.c -- a program with its syntax errors fixed */
#include <stdio.h>
int main(void)

{
```

```
    int n, n2, n3;

    /* this program has a semantic error */
    n = 5;
    n2 = n * n;
    n3 = n2 * n2;
    printf("n = %d, n squared = %d, n cubed = %d\n", n, n2, n3);

    return 0;
}
```

检测方法:

1. 比较程序实际得到的结果和预期结果;
2. 把自己想像成计算机,跟着程序一步步执行

程序状态

跟踪程序的方法

1. 自己逐步执行程序.避免按照自己期望去执行步骤,应该按照实际代码去执行
2. 在程序中关键点处加入额外的`printf()`语句以监视所选变量的值
3. 使用调试器(**后面重点学习**)

关键字和保留标识符

关键字:

ISO C Keywords

auto	extern	short	while
break	float	signed	_Alignas
case	for	sizeof	_Alignof
char	goto	static	_Bool
const	if	struct	_Complex
continue	inline	switch	_Generic
default	int	typedef	_Imaginary
do	long	union	_Noreturn
double	register	unsigned	_Static_assert
else	restrict	void	#_Thread_local
enum	return	volatile	

保留标识符=以下划线开头的标识符+标准库函数的名称,例如`printf()`

编程练习

```
//练习7.多个函数的使用
```

```c
//调用one_three()函数.一行显示单词"one",再调用two()函数,在一行显示单词"three"
#include<stdio.h>
void one_three(void);
void two(void);
int main(void)
{
    printf("starting now:\n");
    one_three();
    return 0;
}
void one_three(void)
{
    printf("one\n");
    two();
    printf("three\n");

}
void two(void)
{
    printf("two\n");
}
```

数据和C

示例

```c
/* platinum.c  -- your weight in platinum */
#include <stdio.h>
int main(void)
{
    float weight;    /* user weight             */
    float value;     /* platinum equivalent     */

    printf("Are you worth your weight in platinum?\n");
    printf("Let's check it out.\n");
    printf("Please enter your weight in pounds: ");

    /* get input from the user                    */
    scanf("%f", &weight);
    /* assume platinum is $1700 per ounce         */
    /* 14.5833 converts pounds avd. to ounces troy */
    value = 1700.0 * weight * 14.5833;
    printf("Your weight in platinum is worth $%.2f.\n", value);
    printf("You are easily worth that! If platinum prices drop,\n");
    printf("eat more to maintain your value.\n");

    return 0;
}
```

- 使用浮点变量类型
- 注意其中的交互性

变量与常量数据

变量值可变`int i`,常量不可变`const int i`

数据:数据类型关键字

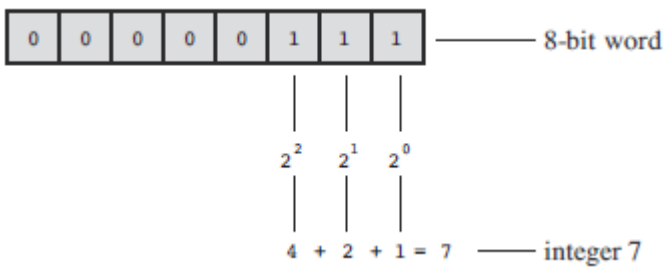
Table 3.1 C Data Keywords

Original K&R Keywords	C90 K&R Keywords	C99 Keywords
int	signed	_Bool
long	void	_Complex
short		_Imaginary
unsigned		
char		
float		
double		

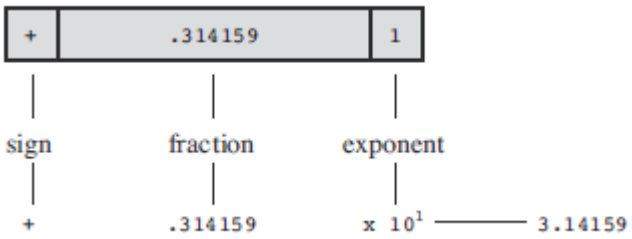
> 位、字节和字的区别:
>
> **位(bit):**最小的存储单位
>
> **字节(byte):**常用的计算机存储单位.通常,1 byte=8 bit
>
> **字(word):**自然的存储单位.64位机中1 word=64 bit

整数与浮点类型

整数(integer)



浮点数(floating point)



C数据类型

总结：基本数据类型

关键字：

基本数据类型使用 11 个关键字：int、long、short、unsigned、char、float、double、signed、_Bool、_Complex 和_Imaginary。

有符号整数：

这种类型可以取正值及负值。

- int：系统的基本整数类型。C 保证 int 类型至少有 16 位长。
- short 或 short int：最大的 short 整数不大于最大的 int 整数值。C 保证 short 类型至少有 16 位长。
- long 或 long int：这种类型的整数不小于最大的 int 整数值。C 保证 long 至少有 32 位长。
- long long 或 long long int：这种类型的整数不小于最大的 long 整数值。long long 类型至少是 64 位长。

一般地，long 类型长于 short 类型，int 类型和它们其中的一个长度相同。例如，PC 机上基于 DOS 的系统提供 16 位长的 short 和 int 类型，以及 32 位长的 long 类型；而基于 Windows 95 的系统提供 16 位长的 short 以及 32 位长的 int 类型和 long 类型。

如果您喜欢，可以使用 signed 关键字修饰任何一种有符号类型，以明确表示这一属性。

无符号整数：

无符号整数只有 0 和正值，这使得无符号数可以表达比有符号数更大的正值。使用 unsigned 关键字表示无符号数，例如：unsigned int、unsigned long 和 unsigned short。单独的 unsigned 等价于 unsigned int。

字符：

字符包括印刷字符，如 A、&和+。在定义中，char 类型使用 1 个字节的存储空间表示一个字符。出于历史原因，字符字节通常为 8 位，但出于表示基本字符集的需要，它也可以为 16 位或者更长。

- char：字符类型的关键字。一些实现使用有符号的 char，另外一些则使用无符号 char。C 允许使用 signed 和 unsigned 关键字标志 char 的符号属性。

布尔值：

布尔值表示 true 和 false；C 使用 1 代表 true，0 代表 false。

- _Bool：此类型的关键字。布尔值是一个无符号整数，其存储只需要能够表示0和1的空间。

实浮点数：

实浮点数可以有正值或负值。

- float：系统的基本浮点类型。至少能精确表示6位有效数字。
- double：范围（可能）更大的浮点类型。能表示比float类型更多的有效数字（至少10位，通常会更多）以及更大的指数。
- long double：范围（可能）再大的浮点类型。能表示比double类型更多的有效数字以及更大的指数。

复数和虚浮点数：

虚数类型是可选的类型，实部和虚部基于如下相应的实数类型：

- float _Complex
- double _Complex
- long double _Complex
- float _Imaginary
- double _Imaginary
- long double _Imaginary

总结：如何声明简单变量

1. 选择所需类型。
2. 选用合法的字符为变量起一个名字。
3. 使用下面的声明语句格式：

```
type-specifier variable-name;
```

type-specifier 由一个或多个类型关键字组成，下面是一些声明的例子：

```
int erest;
unsigned short cash;
```

4. 可以在同一类型后声明多个变量，这些变量名之间用逗号分隔，如下例所示：

```
char ch, init, ans;
```

5. 可以在声明语句中初始化变量，如下例所示：

```
float mass = 6.0E24;
```

类型大小

表 3.4 典型系统的整数类型大小（bit）

类　型	Macintosh Metrowerks CW（默认）	PC 机上的 Linux 系统	IBM PC 机上的 Windows XP 和 Windows NT 系统	ANSI C 规定的最小值
char	8	8	8	8
int	32	32	32	16
short	16	16	16	16
long	32	32	32	32
long long	64	64	64	64

表 3.5 典型系统的浮点数情况

类　型	Macintosh Metrowerks CW（默认）	PC 机上的 Linux 系统	IBM PC 机上的 Windows XP 和 Windows NT 系统	ANSI C 规定的最小值
float	6 位	6 位	6 位	6 位
	-37 到 38	-37 到 38	-37 到 38	-37 到 37
double	18 位	15 位	15 位	10 位
	-4931 到 4932	-307 到 308	-307 到 308	-37 到 37
long double	18 位	18 位	18 位	10 位
	-4931 到 4932	-4931 到 4932	-4931 到 4932	-37 到 37

对于每种类型，上面的行代表有效数字位数，下面的行代表指数的范围（以 10 为基数）。

```c
//* typesize.c -- prints out type sizes */
#include <stdio.h>
int main(void)
{
    /* c99 provides a %zd specifier for sizes */
    printf("Type int has a size of %zd bytes.\n", sizeof(int));
    printf("Type char has a size of %zd bytes.\n", sizeof(char));
    printf("Type long has a size of %zd bytes.\n", sizeof(long));
    printf("Type long long has a size of %zd bytes.\n",
           sizeof(long long));
    printf("Type double has a size of %zd bytes.\n",
           sizeof(double));
    printf("Type long double has a size of %zd bytes.\n",
           sizeof(long double));
    return 0;
}
```

- `sizeof`以字节为单位给出类型的大小,`%zd`表无符号整数类
- C将`char`类型长度定义为1个字节大小

使用数据类型

变量的名称可表数据类型,如`i_`前缀表int变量,`us_`前缀表unsigned short变量

参数和易犯错误

```c
/* badcount.c -- incorrect argument counts */
#include <stdio.h>
int main(void)
{
    int n = 4;
    int m = 5;
    float f = 7.0f;
    float g = 8.0f;

    printf("%d\n", n, m);    /* too many arguments   */
    printf("%d %d %d\n", n); /* too few arguments    */
    printf("%d %d\n", f, g); /* wrong kind of values */

    return 0;
}
```

- 使用`%d`显示float值不会把float转换为近似的int值,而是显示垃圾值;使用`%f`也不能把int值转换为float值

转义序列

表 3.2 转 义 序 列

序　列	意　义
\a	警报（ANSI C）
\b	退格
\f	走纸
\n	换行
\r	回车
\t	水平制表符
\v	垂直制表符
\\	反斜杠（\）
\'	单引号（'）
\"	双引号（"）
\?	问号（?）
\0oo	八进制值（o 表示一个八进制数字）
\xhh	十六进制值（h 表示一个十六进制数字）

```c
/* escape.c -- uses escape characters */
#include <stdio.h>
int main(void)
{
```

```
    float salary;

    printf("\aEnter your desired monthly salary:");/* 1 */
    printf(" $_______\b\b\b\b\b\b\b");             /* 2 */
    scanf("%f", &salary);
    printf("\n\t$%.2f a month is $%.2f a year.", salary,
           salary * 12.0);                         /* 3 */
    printf("\rGee!\n");                            /* 4 */

    return 0;
}
```

- 标准C规定在以下几种情况下,将缓冲区内容传给屏幕:缓冲区满时、遇到换行符时、需要输入时。将缓冲区内容传送给屏幕或文件称为刷新缓冲区（flushing the buffer）

字符串和格式化输入与输出

示例

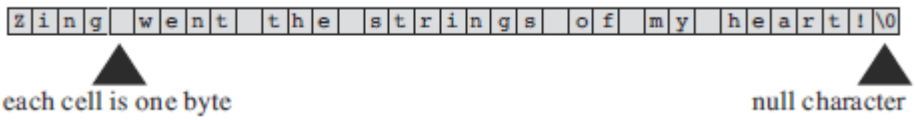
```
// talkback.c -- nosy, informative program
#include <stdio.h>
#include <string.h>      // for strlen() prototype
#define DENSITY 62.4     // human density in lbs per cu ft
int main()
{
    float weight, volume;
    int size, letters;
    char name[40];        // name is an array of 40 chars

    printf("Hi! What's your first name?\n");
    scanf("%s", name);
    printf("%s, what's your weight in pounds?\n", name);
    scanf("%f", &weight);
    size = sizeof name;
    letters = strlen(name);
    volume = weight / DENSITY;
    printf("Well, %s, your volume is %2.2f cubic feet.\n",
           name, volume);
    printf("Also, your first name has %d letters,\n",
           letters);
    printf("and we have %d bytes to store it.\n", size);

    return 0;
}
```

- 使用数组存放一个字符串
- `%s`转换说明符处理字符串的输入与输出
- 用`#define`预先定义了符号常量

- 用`strlen()`函数来获取字符串的长度

字符串简介

char数组类型和空字符



e.g.:`char name[40]`

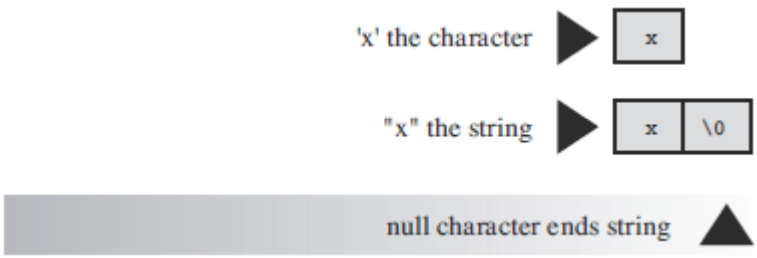
使用字符串

```
/* praise1.c -- uses an assortment of strings */
#include <stdio.h>
#define PRAISE "You are an extraordinary being."
int main(void)
{
    char name[40];

    printf("What's your name? ");
    scanf("%s", name);
    printf("Hello, %s. %s\n", name, PRAISE);

    return 0;
}
```

> What's your name? **Guo Wei**
>
> Hello, **Guo**. You are an extraordinary being.

- `#define`可直接定义一个字符串
- `scanf`只读取了`Guo Wei`中的`Guo`,在第一个空白字符(空格、制表符、换行符)处停止读取,一般地,使用`%s`的`scanf()`开始只会把一个单词而不是把整个语句作为字符串读入,可用其他函数(如`gets()`)来处理一般字符

字符串与字符



strlen()函数

```
/* praise2.c */

// try the %u or %lu specifiers if your implementation
```

```c
// does not recognize the %zd specifier
#include <stdio.h>
#include <string.h>      /* provides strlen() prototype */
#define PRAISE "You are an extraordinary being."
int main(void)
{
    char name[40];

    printf("What's your name? ");
    scanf("%s", name);
    printf("Hello, %s. %s\n", name, PRAISE);
    printf("Your name of %zd letters occupies %zd memory cells.\n",
           strlen(name), sizeof name);
    printf("The phrase of praise has %zd letters ",
           strlen(PRAISE));
    printf("and occupies %zd memory cells.\n", sizeof PRAISE);

    return 0;
}
```

- 处理很长的`printf()`语句:
  - 语句占据两行(可以在参数之间断开,但不能在字符串间断开)
  - 使用多个`printf()`语句
- `sizeof运算符`返回的数字比`strlen()`大1,因为其把字符串结尾的空字符也计算在内
- `sizeof`后的`()`对于类型是必须的,如`sizeof(char)`,但对具体量是可选,通常加上最好

常量和C预处理器

- 格式:`#define NAME value`
- 常量最好大写,可加`c_`之类的前缀,第一个字符不能是数字
- `#define`语句可以定义字符和字符串常量,前者用单引号,后者用双引号

const修饰符

格式:`const int MONTHS=12;//MONTHS为表12的符号常量,MONTHS成了一个只读值`

- `const`比`#define`更灵活

系统定义的明显常量

- C头文件`limit.h`和`float.h`分别提供有关整数类型和浮点类型大小限制的详细信息
- 可用`printf("Maximum int value on this system =%d\n",INT_MAX);`输出整数类型的最大值
- 尽量利用C预处理器

研究和利用printf()和scanf()

printf()函数

表 4.3　　转换说明符及作为结果的打印输出

转 换 说 明	输　　出
%a	浮点数、十六进制数字和 p-记数法（C99）
%A	浮点数、十六进制数字和 P-记数法（C99）
%c	一个字符
%d	有符号十进制整数
%e	浮点数、e-记数法
%E	浮点数、E-记数法
%f	浮点数、十进制记数法
%g	根据数值不同自动选择%f 或%e。%e 格式在指数小于-4 或者大于等于精度时使用
%G	根据数值不同自动选择%f 或%E。%E 格式在指数小于-4 或者大于等于精度时使用
%i	有符号十进制整数（与%d 相同）
%o	无符号八进制整数
%p	指针
%s	字符串
%u	无符号十进制整数
%x	使用十六进制数字 0f 的无符号十六进制整数
%X	使用十六进制数字 0F 的无符号十六进制整数
%%	打印一个百分号

使用printf()

- `printf( Control-string , item1 , item2 ,...);`,其中`item1,item2,...`是要打印项目,可为变量或常量或打印前进行计算的表达式,`Control-string`为描述项目如何打印的字符串

> 注意:每个项目都必须对应一个转换说明,否则可能危及系统

printf()转换说明修饰符

表 4.4 printf（）修饰符

修饰符	意　义
标志	五种标志（-、+、空格、#和 0）都将在表 4.5 中描述。可以使用零个或者多个标志 示例：" %-10d"
digit（s）	字段宽度的最小值。如果该字段不能容纳要打印的数或者字符串，系统就会使用更宽的字段 示例："%4d"
.digit（s）	精度。对于%e、%E 和%f 转换，是将要在小数点的右边打印的数字的位数。对于%g 和%G 转换，是有效数字的最大位数。对于%s 转换，是将要打印的字符的最大数目。对于整数转换，是将要打印的数字的最小位数：如果必要，要使用前导零来达到这个位数。只使用"."表示其后跟随一个零，所以%.f 与%.0f 相同 示例："%5.2f"打印一个浮点数，它的字段宽度为 5 个字符，小数点后有两个数字
h	和整数转换说明符一起使用，表示一个 short int 或 unsigned short int 类型数值 示例："%hu"、"%hx"和"%6.4hd"
hh	和整数转换说明符一起使用，表示一个 signed char 或 unsigned char 类型数值 示例："%hhu"、"%hhx"和"%6.4hhd"
j	和整数转换说明符一起使用，表示一个 intmax_t 或 uintmax_t 值 示例："%jd"和"%8jX"
l	和整数转换说明符一起使用，表示一个 long int 或 unsigned long int 类型值 示例："%ld"和"%8lu"
ll	和整数转换说明符一起使用，表示一个 long long int 或 unsigned long long int 类型值（C99） 示例："%lld"和"%8llu"
L	和浮点转换说明符一起使用，表示一个 long double 值 示例："%Lf"和"%10.4Le"
t	和整数转换说明符一起使用，表示一个 ptrdiff_t 值（与两个指针之间的差相对应的类型）（C99） 示例："%td"和"%12ti"
z	和整数转换说明符一起使用，表示一个 size_t 值（sizeof 返回的类型）（C99） 示例："%zd"和"%12zx"

表 4.5 printf（）的标志

标　志	意　义
-	项目是左对齐的：也就是说，会把项目打印在字段的左侧开始处 示例："%-20s"
+	有符号的值若为正，则显示带加号的符号；若为负，则带减号的符号 示例："%+6.2f"
（空格）	有符号的值若为正，则显示时带前导空格（但是不显示符号）；若为负，则带减号符号。+标志会覆盖空格标志 示例："% 6.2f"
#	使用转换说明的可选形式。若为%o 格式，则以 0 开始；若为%x 和%X 格式，则以 0x 或 0X 开始。对于所有的浮点形式，#保证了即使不跟任何数字，也打印一个小数点字符。对于%g 和%G 格式，它防止尾随零被删除 示例："%#o"、"%#8.0f"和"%+#10.3E"
0	对于所有的数字格式，用前导零而不是用空格填充字段宽度。如果出现-标志或者指定了精度（对于整数）则忽略该标志 示例："%010d"和"%08.3f"

```
// floats.c -- some floating-point combinations
#include <stdio.h>
```

```c
int main(void)
{
    const double RENT = 3852.99;  // const-style constant

    printf("*%f*\n", RENT);
    printf("*%e*\n", RENT);
    printf("*%4.2f*\n", RENT);
    printf("*%3.1f*\n", RENT);
    printf("*%10.3f*\n", RENT);
    printf("*%10.3E*\n", RENT);
    printf("*%+4.2f*\n", RENT);
    printf("*%010.2f*\n", RENT);

    return 0;
}
```

```c
/* flags.c -- illustrates some formatting flags */
#include <stdio.h>
int main(void)
{
    printf("%x %X %#x\n", 31, 31, 31);
    printf("**%d**% d**% d**\n", 42, 42, -42);
    printf("**%5d**%5.3d**%05d**%05.3d**\n", 6, 6, 6, 6);

    return 0;
}
```

```c
/* stringf.c -- string formatting */
#include <stdio.h>
#define BLURB "Authentic imitation!"
int main(void)
{
    printf("[%2s]\n", BLURB);
    printf("[%24s]\n", BLURB);
    printf("[%24.5s]\n", BLURB);
    printf("[%-24.5s]\n", BLURB);
    return 0;
}
```

- 格式说明符中`.5`使printf()只打印5个字符

转换说明的意义

打印较长的字符串

- 使用多个printf()语句
- 使用`/`+`回车键`,下一行必须从最左边开始

- 使用字符串连接."字符串"+空白字符+"字符串"

使用scanf()

- 使用scanf()读取一般数据变量类型,在变量名前加&,但把一个字符串读进字符数组中,不用加&

格式字符串中的常规字符

- 格式字符串中的空格意味着跳过下一个输入项之前的任何空格
- 除%c以外的说明符会自动跳过输入项之前的空格

scanf()的返回值

- scanf()返回成功读入项目的个数
- 当检测到"文件结尾",返回EOF(通常定义为-1)
- 可用scanf()返回值检测和处理不匹配的输入(比如while语句中)

printf()和scanf()的*修饰符

```c
/* varwid.c -- uses variable-width output field */
#include <stdio.h>
int main(void)
{
    unsigned width, precision;
    int number = 256;
    double weight = 242.5;

    printf("Enter a field width:\n");
    scanf("%d", &width);
    printf("The number is :%*d:\n", width, number);
    printf("Now enter a width and a precision:\n");
    scanf("%d %d", &width, &precision);
    printf("Weight = %*.*f\n", width, precision, weight);
    printf("Done!\n");

    return 0;
}
```

- 由程序自己指定字段宽度

```c
/* skiptwo.c -- skips over first two integers of input */
#include <stdio.h>
int main(void)
{
    int n;

    printf("Please enter three integers:\n");
    scanf("%*d %*d %d", &n);
    printf("The last integer was %d\n", n);

    return 0;
}
```

- 跳过两个整数,把第三个整数赋给n
- 可用于读取文件中特定的列

printf()用法提示

- 指定足够大的固定字段宽度可使输出更加整齐清晰
- 在两转换说明之前加空白字符,可确保即使一个数字溢出自己的字段,也不会影响第二个数的输出
- 在语句中加入数字,用`%.2f`类型格式比较好

运算符、表达式和语句

基本运算符

> 与赋值运算符=有关的几个术语:
>
> - 数据对象(data object):泛指数据存储区(能用于保存值)
> - 左值(lvalue):标识一个特定的数据对象的名字或表达式
> - 右值(rvalue):能赋给可修改左值的量
> - 操作数(operand):运算符操作的对象

- C中允许多重赋值,如`A=B=C=1`
- 整数除法结果的中的小数部分都会被舍弃,称为"截尾",准确说应该是"趋零截尾"
- 取模运算符`%`用于整数运算,注意用强制类型转换将数转换成整数,结果符号和第一个数的符号相同

增量/减量运算符

`a_post = a++; //后缀:使用a的值之后改变a`

`pre_b = ++b; // 前缀:使用b值之前改变b`

- 优先级:最有圆括号的优先级高于增量和减量运算符
- 使用原则:
  - 如果一变量出现在同一函数的多个参数中,不对它使用增量或减量运算符
  - 一变量多次出现在一个表达式中,不对它使用增量或减量运算符

运算符优先级和求值顺序

表 5.2　　按优先级递减顺序排列的运算符

运　算　符	结　合　性
( )	从左到右
+ - （一元运算符）	从右到左
* /	从左到右
+ - （二元运算符）	从左到右
=	从右到左

表达式和语句

总结：表达式和语句

表达式：

表达式（expression）是运算符和操作数的组合。最简单的表达式只有一个常量或一个变量而没有运算符，例如 22 或者 beebop。更复杂的例子是 55+22 和 vap=2*（vip+（vup=4））。

语句：

语句（statement）是对计算机的命令。有简单语句和复合语句。简单语句（simple statement）以一个分号结束，如下面这些例子：

声明语句：
```
int toes;
```
赋值语句：
```
toes = 12;
```
函数调用语句：
```
printf ("%d\n", toes);
```
结构化语句：
```
while (toes < 20) toes = toes + 2;
```
空语句：
```
; /* 什么也不做 */
```

复合语句（compound statement）或代码块（block）由一个或多个括在花括号里的语句（这些语句本身也可能是复合语句）构成。下面的 while 语句中含有一个例子：

```
while (years < 100)
{
    wisdom = wisdom * 1.05;
    printf ("%d %d\n", years, wisdom);
    years = years + 1;
}
```

- 副作用(side effect):对数据对象或文件的修改
- 顺序点(sequence point):程序执行中的一个点,在该点处所有的副作用在进入下一步前都被计算,语句的分号标志了一个顺序点,完整表达式(即不是一个更大表达式的子表达式)的结束也是个顺序点

类型转换

指派运算符

格式:`(type)`,如`(int)1.6`→`1`

总结:C中的运算

总结：C 中的运算

表 5.4 中列出的是到目前为止我们已经讨论过的运算符：

表 5.4　　C 的一些运算符

赋值运算符：	
=	将它右边的值赋给它左边的变量
算术运算符：	
+	将它右边的值和它左边的值相加
-	从它左边的值里减掉它右边的值
-	作为一元运算符，改变它右边值的符号
*	用它左边的值乘以它右边的值
/	用它右边的值去除它左边的值。如果两个操作数都是整数，那么结果被截尾
%	当它左边的值被它右边的值除时，得到的余数（只对整数）
++	对它右边的值加 1（前缀模式），或者对它左边的值加 1（后缀模式）
--	与++类似，只不过是减 1
其他运算符：	
sizeof	给出它右边的操作数的字节大小。操作数可以是在圆括号里的一个类型说明符，例如 sizeof (float)；或者是一个具体的变量、数组等的名字，例如 sizeof foo
(type)	作为指派运算符，它将跟在它后面的值转换成由圆括号中的关键字所指定的类型。例如，(float) 9 将整数 9 转换成浮点数 9.0

带有参数的函数

形式参量和实际参数:形式参量是变量,实际参数赋值给形式参量

```
/* pound.c -- defines a function with an argument   */
#include <stdio.h>
void pound(int n);   // ANSI function prototype declaration
int main(void)
{
    int times = 5;
    char ch = '!';   // ASCII code is 33
    float f = 6.0f;

    pound(times);    // int argument
    pound(ch);       // same as pound((int)ch);
    pound(f);        // same as pound((int)f);

    return 0;
}

void pound(int n)    // ANSI-style function header
{                    // says takes one int argument
    while (n-- > 0)
        printf("#");
    printf("\n");
}
```

编程练习

```c
//8题.将输入华氏温度转换成摄氏温度和绝对温度
#include<stdio.h>
void Temperature(double Farenheit);//函数声明
int main(void)
{
    double Farenheit;

    printf("Farenheit:\n");
    scanf("%lf",&Farenheit);
    Temperature(Farenheit);//函数调用
    while(Farenheit=!"q")//while循环判断语句
    {
        printf("Farenheit:\n");
        scanf("%lf",&Farenheit);
    }
    return 0;
}
void Temperature(double Farenheit)//函数编写
{
   double Celsius,Kelvin;

   Celsius=1.8*Farenheit+32.0;
   Kelvin=Celsius+271.3;
   printf("Farenheit=%6.2lf,Celsius=%6.2lf,Kelvin=%6.2lf",Farenheit,Celsius,Kelvin);
}
```

C控制语句:循环

while循环

```c
/* summing.c -- sums integers entered interactively */
#include <stdio.h>
int main(void)
{
    long num;
    long sum = 0L;      /* initialize sum to zero   */
    int status;

    printf("Please enter an integer to be summed ");
    printf("(q to quit): ");
    status = scanf("%ld", &num);
    while (status == 1) /* == means "is equal to"   */
    {

        sum = sum + num;
```

```
        printf("Please enter next integer (q to quit): ");
        status = scanf("%ld", &num);
    }
    printf("Those integers sum to %ld.\n", sum);

    return 0;
}
```

- `scanf()`告诉程序何时终止
- 可用伪代码简单表示程序,如:

> initialize sum to 0 prompt user read input while the input is an integer, add the input to sum, prompt user, then read next input after input completes, print sum

- while循环是入口条件循环

```
/* while2.c -- watch your semicolons */
#include <stdio.h>
int main(void)
{
    int n = 0;

    while (n++ < 3);              /* line 7 空语句*/
    printf("n is %d\n", n);  /* line 8 */
    printf("That's all this program does.\n");

    return 0;
}
```

- 可使用带有空语句的while语句,使所有的工作都在判断过程中进行

总结：while 语句

关键字：

while

总体注解：

while 语句创建了一个在判断表达式变为假（或零）之前重复执行的循环。While 语句是一个入口条件循环，也就是说，是否执行循环的决定是在进入循环之前就做出的。因此，循环有可能永远不被执行。该形式的 statement 部分可以是一个简单语句或一个复合语句。

形式：

```
while (expression)
    statement
```

在 expression 变为假（或 0）之前重复执行 statement 部分。

例如：

```
while (n++ < 100)
    printf (" %d %d\n", n, 2 * n + 1);      /* 单个语句 */

while (fargo < 1000)
{                                           /* 复合语句 */
    fargo = fargo + step;
    step = 2 * step;
}
```

比较大小:关系运算符和表达式

总结：关系运算符和表达式

关系运算符：

每个关系运算符都把它左边的值与它右边的值进行比较。

<	小于
<=	小于或等于
==	等于
>=	大于或等于
>	大于
!=	不等于

关系表达式：

一个简单的关系表达式由一个关系运算符及其两侧的操作数组成。如果关系为真，关系表达式的值为 1；如果关系为假，关系表达式的值为 0。

例如：

5 > 2 为真，则该关系表达式的值为 1。

（2 + a）== a 为假，则该关系表达式的值为 0。

- 注意区别`==`与`=`
- 注意不等于为`!=`
- 关系表达式可用于字符的比较,但不能比较字符串
- 浮点比较只能用`<`和`>`,因为舍入误差可能使逻辑相等的两个数不等
- `math.h`头文件中声明的`fabs()函数`可用于浮点数的判断,如:

```
// cmpflt.c -- floating-point comparisons
```

```c
#include <math.h>
#include <stdio.h>
int main(void)
{
    const double ANSWER = 3.14159;
    double response;

    printf("What is the value of pi?\n");
    scanf("%lf", &response);
    while (fabs(response - ANSWER) > 0.0001)
    {
        printf("Try again!\n");
        scanf("%lf", &response);
    }
    printf("Close enough!\n");

    return 0;
}
```

真值

- 所有非零值为真,只有0为假
- 表达式为真,值为1;若为假,值为0
- 可用Bool类型,true和false更直观

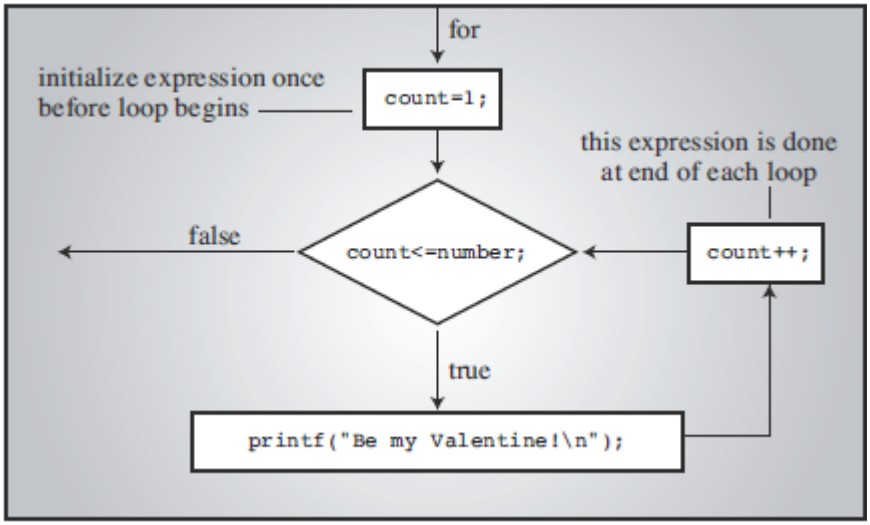
关系运算符的优先级

表 6.2 运算符优先级

运算符（优先级从高到低）	结 合 性
( )	从左到右
- + ++ — sizeof（type）（所有的一元运算符）	从右到左
* / %	从左到右
+ -	从左到右
< > <= >=	从左到右
== !=	从左到右
=	从右到左

for循环

```c
for (count = 1; count <= NUMBER; count++)
        printf("Be my Valentine!\n");
```



for的灵活性

- 可使用减量运算符减小计数器而不是增加它
- 可让计数器加2,10,...
- 可以判断迭代次数之外的条件
- 可让数量几何增加而不是算数增加
- 第三个表达式可为任何合法表达式
- 可让一个或多个表达式为空(但不可漏分号),只要确保循环能终止
- 第一个表达式不必初始化一个变量,可以是某种类型的printf()语句
- 循环中的动作可以改变循环表达式的参数

总结

总结：for 语句

关键字：

for

总体注解：

for 语句使用由分号分开的三个控制表达式来控制循环过程。initialize 表达式只在循环语句执行之前执行一次。然后对 test 表达式求值，如果该表达式为真（或非零）循环就被执行一次。然后计算 update 表达式，接着再次检查 test 表达式。for 语句是一个入口条件循环，即是否再次执行循环的决定是在循环执行之前做出的。因此，有可能循环一次也不执行。该形式的 statement 部分可以是一个简单语句或一个复合语句。

形式：

```
for (initialize; test; update)
     statement
```

在 test 为假（或零）之前重复执行循环。

例如：

```
for (n = 0; n < 10; n++)
    printf ("%d %d\n", n, 2 * n + 1);
```

其他赋值运算符:+= 、-= 、*=、/=、%=

scores += 20	等于	scores = scores + 20
dimes -= 2	等于	dimes = dimes - 2
bunnies *= 2	等于	bunnies = bunnies * 2
time /= 2.73	等于	time = time / 2.73
reduce %= 3	等于	reduce = reduce % 3

逗号运算符

```
// postage.c -- first-class postage rates
#include <stdio.h>
int main(void)
{
    const int FIRST_OZ = 46; // 2013 rate
    const int NEXT_OZ = 20;  // 2013 rate
    int ounces, cost;

    printf(" ounces  cost\n");
    for (ounces=1, cost=FIRST_OZ; ounces <= 16; ounces++,
         cost += NEXT_OZ)                                    //逗号运算符
        printf("%5d   $%4.2f\n", ounces, cost/100.0);

    return 0;
}
```

- 作为顺序点的逗号保证左边子表达式的副作用在计算右边子表达式之前生效
- 整个逗号表达式的值是右边成员的值

总结:新运算符

总结：新运算符

赋值运算符：

这些运算符使用指定的操作根据其右边的值来更新其左边的变量。

+=	把右边的值加到左边的变量上
-=	从左边的变量中减去右边的值
*=	把左边的变量乘以右边的值
/=	把左边的变量除以右边的值
%=	给出左边的变量除以右边的值之后的余数

例如：

```
rabbits *= 1.6;
```

等于

```
rabbits = rabbits * 1.6;
```

这些复合赋值运算符和普通的赋值运算符有着同样的比较的运算优先级，比算术运算符的优先级要低得多。因此，以下的两条语句最终效果相同：

```
contents *= old_rate + 1.2;
contents = contents * (old_rate + 1.2);
```

逗号运算符：

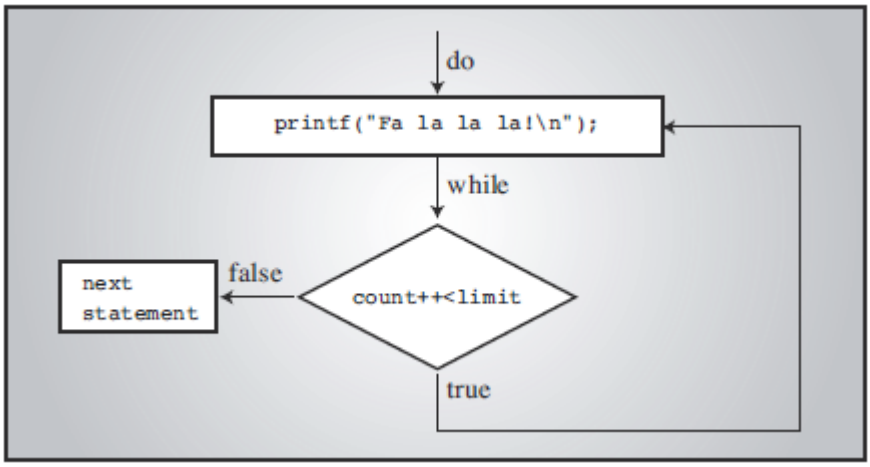
逗号运算符把两个表达式链接为一个表达式，并保证最左边的表达式最先计算。它通常被用在 for 循环的控制表达式中以包含多个信息。整个表达式的值是右边表达式的值。

例如：

```
for (step = 2, fargo = 0; fargo < 1000; step *= 2)
        fargo += step;
```

退出条件循环

```c
/* do_while.c -- exit condition loop */
#include <stdio.h>
int main(void)
{
    const int secret_code = 13;
    int code_entered;

    do
    {
        printf("To enter the triskaidekaphobia therapy club,\n");
        printf("please enter the secret code number: ");
        scanf("%d", &code_entered);
    } while (code_entered != secret_code);
    printf("Congratulations! You are cured!\n");

    return 0;
}
```



- 使用`while`循环实现,代码会更长

总结：do while 语句

关键字：

do，while

总体注解：

do while 语句创建了一个在判断表达式为假（或零）之前重复执行的循环。do while 语句是一个退出条件循环，是否再次执行循环的决定是在执行了一次循环之后做出的。因此循环必须至少被执行一次。该形式的 statement 部分可以是一个简单语句或一个复合语句。

形式：

```
do
        statement
while (expression);
```

在 expression 为假（或零）之前重复执行 statement 部分。

例如：

```
do
        scanf ("%d", &number);
while (number != 20);
```

while和for循环的比较

- 在循环涉及到初始化和更新变量时多用for循环,在其他条件下多使用while循环,如:

```
while(scanf("%ld",&num)==1)//while循环更自然
for(count=1;count<=100;count++)//for循环更自然
```

嵌套循环(nested loop)

定义:在另一个循环之内的循环

示例

```
/* rows1.c -- uses nested loops */
//外部循环创建行,内部循环打印字符
#include <stdio.h>
#define ROWS  6
#define CHARS 10
int main(void)
{
    int row;
    char ch;
                                                /*嵌套循环*/
    for (row = 0; row < ROWS; row++)            /*外部循环*/
    {
        for (ch = 'A'; ch < ('A' + CHARS); ch++)  /*内部循环*/
            printf("%c", ch);
        printf("\n");
    }

    return 0;
}
```

> ABCDEFGHIJ ABCDEFGHIJ ABCDEFGHIJ ABCDEFGHIJ ABCDEFGHIJ ABCDEFGHIJ

- 内部循环在外部循环的每次单独循环中都完全执行它的所有循环

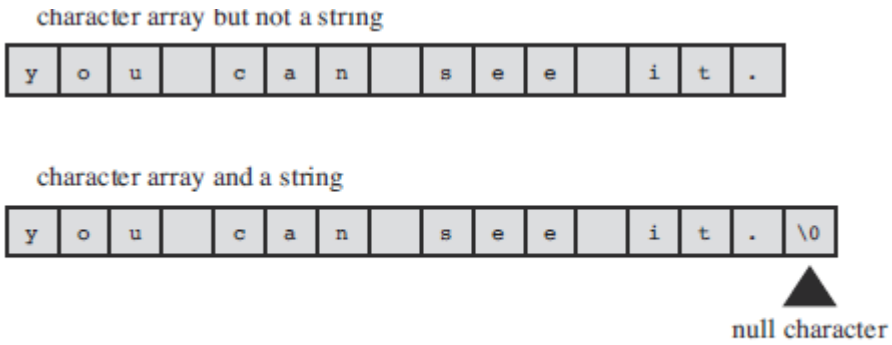
嵌套变化

```
// rows2.c -- using dependent nested loops
#include <stdio.h>
int main(void)
{
    const int ROWS = 6;
    const int CHARS = 6;
    int row;
    char ch;

    for (row = 0; row < ROWS; row++)
    {
        for (ch = ('A' + row);  ch < ('A' + CHARS); ch++)//内部循环依赖于外部循环
            printf("%c", ch);
        printf("\n");
    }

    return 0;
}
```

> ABCDEF BCDEF CDEF DEF EF F

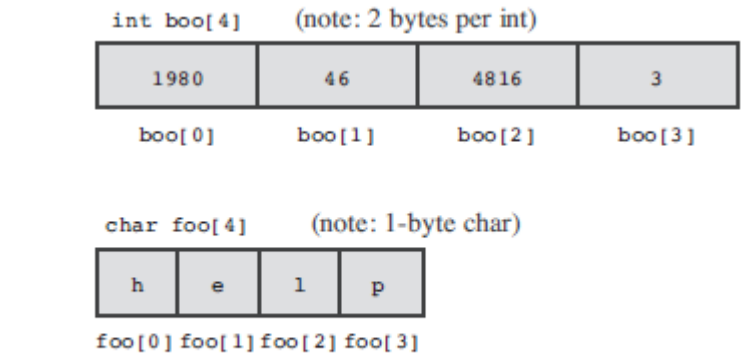
数组

- 内存中的字符数组和字符串:



Character arrays and strings.

- 内存中的char和int数组:



The char and int arrays in memory.

在for循环中使用数组

```
// scores_in.c -- uses loops for array processing
//读取十个数,求其平均值
#include <stdio.h>
#define SIZE 10
#define PAR 72
int main(void)
{
    int index, score[SIZE];
    int sum = 0;
    float average;

    printf("Enter %d golf scores:\n", SIZE);
    for (index = 0; index < SIZE; index++)
        scanf("%d", &score[index]);  // read in the ten scores
    printf("The scores read in are as follows:\n");
    for (index = 0; index < SIZE; index++)
        printf("%5d", score[index]); // verify input
    printf("\n");
    for (index = 0; index < SIZE; index++)
```

```
        sum += score[index];          // add them up
    average = (float) sum / SIZE;     // time-honored method
    printf("Sum of scores = %d, average = %.2f\n", sum, average);
    printf("That's a handicap of %.0f.\n", average - PAR);

    return 0;
}
```

- 使用`#define`指令创建一个指定数组大小的常量(SIZE),增加了可读性和代码重复利用率
- 注意数组的边界:第一个元素指标为`0`,最后个元素指标为`SIZE-1`
- 注意其中模块化(modularity)编程的原则(多个for循环的使用):程序应该被分为单独的单元,每个单元执行一个任务,使程序更可读且更容易升级和修改,可以将每个单元放入函数中增强模块性

使用函数返回值的循环例子

```
// power.c -- 计算数值的整数次幂
#include <stdio.h>
double power(double n, int p); // ANSI prototype,函数声明,返回值为double型
int main(void)
{
    double x, xpow;
    int exp;

    printf("Enter a number and the positive integer power");
    printf(" to which\nthe number will be raised. Enter q");
    printf(" to quit.\n");
    while (scanf("%lf%d", &x, &exp) == 2)
    {
        xpow = power(x,exp);   // function call(函数调用)
        printf("%.3g to the power %d is %.5g\n", x, exp, xpow);
        printf("Enter next pair of numbers or q to quit.\n");
    }
    printf("Hope you enjoyed this power trip -- bye!\n");

    return 0;
}

double power(double n, int p)  // function definition
{
    double pow = 1;
    int i;

    for (i = 1; i <= p; i++)
        pow *= n;

    return pow;                // return the value of pow
}
```

> 写一个具有返回值的函数需要完成:

> 1. 当定义循环时,说明它的返回值类型
> 2. 使用关键字return指示要返回的值

编程练习

```
//练习1.字符数组的存储和显示
//创建一个具有26个元素的数组,存储26个小写字母,并显示
#include<stdio.h>
int main(void)
{
    char ch[26];
    int i;
    for(i=0; i<26; i++)
        printf("%c",ch[i]='a'+i);//注意数组的第一位元素指标为0
    return 0;

}
```

```
//练习2.嵌套循环
/*循环产生图案:
$
$$
$$$
$$$$
$$$$$
*/
#include<stdio.h>
int main(void)
{
    int m,n;
    for(n=0; n<5; n++)//嵌套循环
    {
        for(m=0; m<=n; m++)
            printf("$");
        printf("\n");
    }
    return 0;
}
```

```
//练习3.嵌套循环
/*
循环产生图案:
F
FE
FED
FEDC
FEDCB
```

```
FEDCBA
*/
#include<stdio.h>
int main(void)
{
    char ch[26];
    int m,n;
    for(n=0; n<6; n++)
    {
        for(m=0; m<=n; m++)
            printf("%c",ch[m]='F'-m);//降序排列
        printf("\n");
    }
    return 0;
}
```

```
//练习4.嵌套循环
/*
循环产生图案:
    A
   ABA
  ABCBA
 ABCDCBA
ABCDEDCBA

*/
#include<stdio.h>
int main(void)
{
    char ch[26];
    int i,j,m,n;

    for(n=0; n<5; n++)
    {
        for(i=0; i<4-n; i++)//空格
            printf(" ");
        for(m=0; m<=n; m++)//升序
            printf("%c",ch[m]='A'+m);
        for(j=0; j<n; j++)
            printf("%c",ch[n+1]=ch[n]-j-1);//降序
        printf("\n");
    }
    return 0;
}
```

```
//练习5.for循环
//打印一个表,表每行给出整数、其平方和立方

#include<stdio.h>
```

```c
#define MAX 10
#define MIN 0
int main(void)
{
    int n;

    for(n=MIN; n<=MAX; n++)
    {
        printf("%d %d %d\n",n,n*n,n*n*n);
    }
    return 0;
}
```

```c
//练习6.for循环;strlen()函数
//把单词读入字符数组,再反向打印
#include<stdio.h>
#include<string.h>
int main(void)
{
    char st[20];
    int i;

    scanf("%s",st);
    for(i=strlen(st); i>0; i--)//用strlen计算字符长度,不包含\n
        printf("%c",st[i-1]);
    return 0;
}
```

```c
//练习7.退出条件循环;条件判断
//输入两浮点数,打印出二者的差值除以二者的乘积的结果
#include<stdio.h>
int main(void)
{
    float a,b;
    int staus;

    do
    {
        staus=scanf("%f %f",&a,&b);
        if(staus==2)//if else语句提前使用
        printf("%f\n",(a-b)/(a*b));
        else
            break;//跳出循环
    }
    while(staus);//退出循环的条件
    return 0;
}
```

```c
//练习8.退出条件循环;函数使用
//对练习7修改,使用函数来返回计算值
#include<stdio.h>
float result(float a,float b);//函数声明
int main(void)
{
    float a,b;
    int staus;

    do
    {
        staus=scanf("%f %f",&a,&b);
        printf("%f\n",result(a,b));//函数调用
    }
    while(staus);
    return 0;
}
float result(float a,float b)//函数编写
{
    float c;

    c=(a-b)/(a*b);
    return c;
}
```

```c
//练习9.入口条件循环;for循环
//输入下限、上限整数，计算从下限到上限的每一个整数的平方的加和，显示结果
#include<stdio.h>
int main(void)
{
    int Min,Max;
    int i,sum;

    printf("Enter lower and upper integer limits:");
    scanf("%d %d",&Min,&Max);
    while(Max!=Min)
    {
        sum=0;//注意初值的位置
        for(i=Min; i<=Max; i++)
        {
            sum+=i*i;
        }
        printf("The sums of the squares for %d to %d is %d\n",Min,Max,sum);
        printf("Enter next set of limit:");
        scanf("%d %d",&Min,&Max);
    }
    printf("Done");
    return 0;
}
```

```c
//练习10.数组;for循环
//把8个整数读入数组,再反序打印
#include<stdio.h>
int main(void)
{
    int i,num[8];

    for(i=0;i<8;i++)
    scanf("%d",&num[i]);
    for(i=0;i<8;i++)
    printf("%d  ",num[7-i]);//注意数组及其元素指标的差别
    return 0;
}
```

```c
//练习11.入口条件循环;for循环;强制类型转换;取余运算
/*
输入下列序列项数,求和,判断收敛性
1.0 + 1.0/2.0 + 1.0/3.0 + 1.0/4.0 + ...
1.0 - 1.0/2.0 + 1.0/3.0 - 1.0/4.0 + ...
*/
#include<stdio.h>
#include<math.h>
int main(void)
{
    float i,Max,sum_1=0.0,sum_2=0.0;//注意赋初值的位置
    int staus;
    printf("Max:");
    staus=scanf("%f",&Max);
    while(staus)
    {
            for(i=0.0; i<Max; i++)
        {
            sum_1+=1/(i+1);
            sum_2+=pow(-1,(int)i%2)/(i+1);//注意取余运算变量必须为整型
        }
     printf("sum_1=%f,sum_2=%f\n",sum_1,sum_2);
     printf("Max:");
     staus=scanf("%f",&Max);
    }
    return 0;
}
```

```c
//练习12.for循环;数组;退出条件循环
//创建8个元素的int数组,元素设为2的前8次幂,打印这些值
#include<stdio.h>
#include<math.h>//注意头文件的完整性
int main(void)

{
```

```
    int i,ch[8];

    for(i=0; i<8; i++)
        ch[i]=pow(2,i+1);
    i=0;
    do
    {
        printf("%d ",ch[i]);
    }while(++i<8);
    return 0;
}
```

```
//练习13.数组;嵌套循环
//创建8个元素的double数组,输入元素的值,把第二个数组元素设置为第一个数组元素的累积和
#include<stdio.h>
int main(void)
{
    double num_1[8],num_2[8];
    int i,j;

    printf("num_1[8]:");
    for(i=0; i<8; i++)
        scanf("%lf",&num_1[i]);
        for(i=0; i<8; i++)
        {
            num_2[i]=0.0;//注意赋初值的位置
            for(j=0; j<=i; j++)
            {
                num_2[i]+=num_1[j];//第二个数组元素为第一个数组元素的累积和
            }
        }
    for(i=0; i<8; i++)
        printf("%0.2lf     ",num_1[i]);
    printf("\n");
    for(i=0; i<8; i++)
        printf("%0.2lf     ",num_2[i]);
    return 0;
}
```

```
//练习14.数组;for循环
//读入一行输入,反向打印
#include<stdio.h>
int main(void)
{
    char ch[5];
    int i;

    for(i=0; i<5; i++)

        scanf("%c",&ch[i]);//注意按下回车时会产生\n
```

```
    for(i=0; i<5; i++)
        printf("%c ",ch[4-i]);
    return 0;
}
```

```
//练习15.for循环
//实际应用:单利息与复利息的比较.单利息为年利率为本金的10%,复利息为结余的5%
#include<stdio.h>
int main(void)
{
    int i;
    float Da=100.0,De=100.0;

    for(i=1; Da>=De; i++)
    {
        Da+=100*0.1;
        De*=1.05;
    }
    printf("%d",i-1);//注意不为i
    return 0;
}
```

```
//练习16.for循环
//实际应用:利息问题.存100万,年利率为10%,每年取10万,多久清空账户
#include<stdio.h>
int main(void)
{
    int i;
    float sum=100.0;

    for(i=1; sum>0.0; i++)
    {
        sum*=1.08;
        sum-=10.0;
    }
    printf("%d",i-1);
    return 0;
}
```

C控制语句:分支和跳转

if语句

示例

```c
// colddays.c -- finds percentage of days below freezing
#include <stdio.h>
int main(void)
{
    const int FREEZING = 0;
    float temperature;
    int cold_days = 0;
    int all_days = 0;

    printf("Enter the list of daily low temperatures.\n");
    printf("Use Celsius, and enter q to quit.\n");
    while (scanf("%f", &temperature) == 1)
    {
        all_days++;
        if (temperature < FREEZING)
            cold_days++;
    }
    if (all_days != 0)
        printf("%d days total: %.1f%% were below freezing.\n",
               all_days, 100.0 * (float) cold_days / all_days);//类型转换
    if (all_days == 0)
        printf("No data entered!\n");

    return 0;
}
```

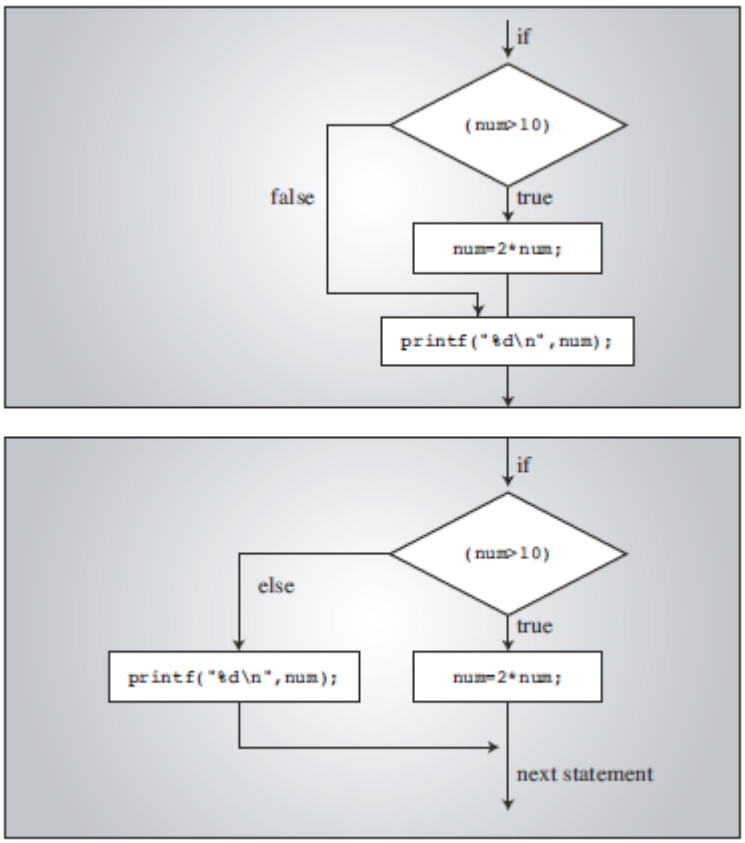
> if语句的一般形式:
>
> if (expression)
>
> statement
>
> **expression为真,执行statement;为假,跳出语句**

if else语句

```c
//原程序
if (all_days != 0)
    printf("%d days total: %.1f%% were below freezing.\n",
           all_days, 100.0 * (float) cold_days / all_days);
if (all_days == 0)
    printf("No data entered!\n");
//修改后
if (all_days!= 0)
    printf("%d days total: %.1f%% were below freezing.\n",
           all_days, 100.0 * (float) cold_days / all_days);
else
    printf("No data entered!\n");
```

> if else 语句的一般形式:

> if (expression) statement1 else statement2
>
> **expression为真,执行statement1;为假,执行statement2**



if versus if else.

例:介绍getchar()和putchar()

```
// cypher1.c -- alters input, preserving spaces
#include <stdio.h>
#define SPACE ' '             // that's quote-space-quote
int main(void)
{
    char ch;

    ch = getchar();           // 第八行.read a character
    while (ch != '\n')        // 第九行.while not end of line
    {
        if (ch == SPACE)      // leave the space
            putchar(ch);      // character unchanged
        else
            putchar(ch + 1);  // change other characters
        ch = getchar();       // get next character
    }
    putchar(ch);              // print the newline

    return 0;
}
```

- 可将八九行合并为`while ((ch = getchar()) != '\n')`

ctype.h系列字符函数

`ctype.h`系列函数接受一个字符作为参数,如果字符属于某特定的种类则返回非零值(真),否则返回零(假)

```
// cypher2.c -- alters input, preserving non-letters
#include <stdio.h>
#include <ctype.h>            // for isalpha()
int main(void)
{
    char ch;

    while ((ch = getchar()) != '\n')
    {
        if (isalpha(ch))      // if a letter,
            putchar(ch + 1);  // display next letter
        else                  // otherwise,
            putchar(ch);      // display as is
    }
    putchar(ch);              // display the newline

    return 0;
}
```

> 本地化:能够指定一个本地以修改或扩展C的基本用法的C工具

- 映射函数不改变原参数值,只返回改变后的值,若要改变原参数的值可使`ch = tolower(ch)`

表 7.1　ctype.h 的字符判断函数

函　数　名	为如下参数时，返回值为真
isalnum（）	字母数字（字母或数字）
isalpha（）	字母
isblank()	一个标准的空白字符（空格、水平制表符或者换行）或者任何其他本地化指定为空白符的字符
iscntrl（）	控制符，例如 Ctrl+B
isdigit（）	阿拉伯数字
isgraph（）	除空格符之外的所有可打印字符
islower（）	小写字母
isprint（）	可打印字符
ispunct（）	标点符号（除空格和字母数字外的可打印字符）
isspace（）	空白字符：空格、换行、走纸、回车、垂直制表符、水平制表符、或可能是其他本地化定义的字符
isupper（）	大写字母
isxdigit（）	十六进制数字字符

表 7.2 ctype.h 的字符映射函数

函 数 名	动 作
tolower（）	如果参数是大写字符，返回相应的小写字符；否则，返回原始参数
toupper（）	如果参数是小写字符，返回相应的大写字符；否则，返回原始参数

多重选择else if

```
// electric.c -- calculates electric bill
#include <stdio.h>
#define RATE1   0.13230       // rate for first 360 kwh
#define RATE2   0.15040       // rate for next 108 kwh
#define RATE3   0.30025       // rate for next 252 kwh
#define RATE4   0.34025       // rate for over 720 kwh
#define BREAK1  360.0         // first breakpoint for rates
#define BREAK2  468.0         // second breakpoint for rates
#define BREAK3  720.0         // third breakpoint for rates
#define BASE1   (RATE1 * BREAK1)
// cost for 360 kwh
#define BASE2  (BASE1 + (RATE2 * (BREAK2 - BREAK1)))
// cost for 468 kwh
#define BASE3   (BASE1 + BASE2 + (RATE3 *(BREAK3 - BREAK2)))
//cost for 720 kwh
int main(void)
{
    double kwh;               // kilowatt-hours used
    double bill;              // charges

    printf("Please enter the kwh used.\n");
    scanf("%lf", &kwh);       // %lf for type double
    if (kwh <= BREAK1)
        bill = RATE1 * kwh;
    else if (kwh <= BREAK2)   // kwh between 360 and 468
        bill = BASE1 + (RATE2 * (kwh - BREAK1));
    else if (kwh <= BREAK3)   // kwh betweent 468 and 720
        bill = BASE2 + (RATE3 * (kwh - BREAK2));
    else                      // kwh above 680
        bill = BASE3 + (RATE4 * (kwh - BREAK3));
    printf("The charge for %.1f kwh is $%1.2f.\n", kwh, bill);

    return 0;
}
```

- 预处理器不做计算,在`BASE1`出现的地方用`0.12589*360.0`代替

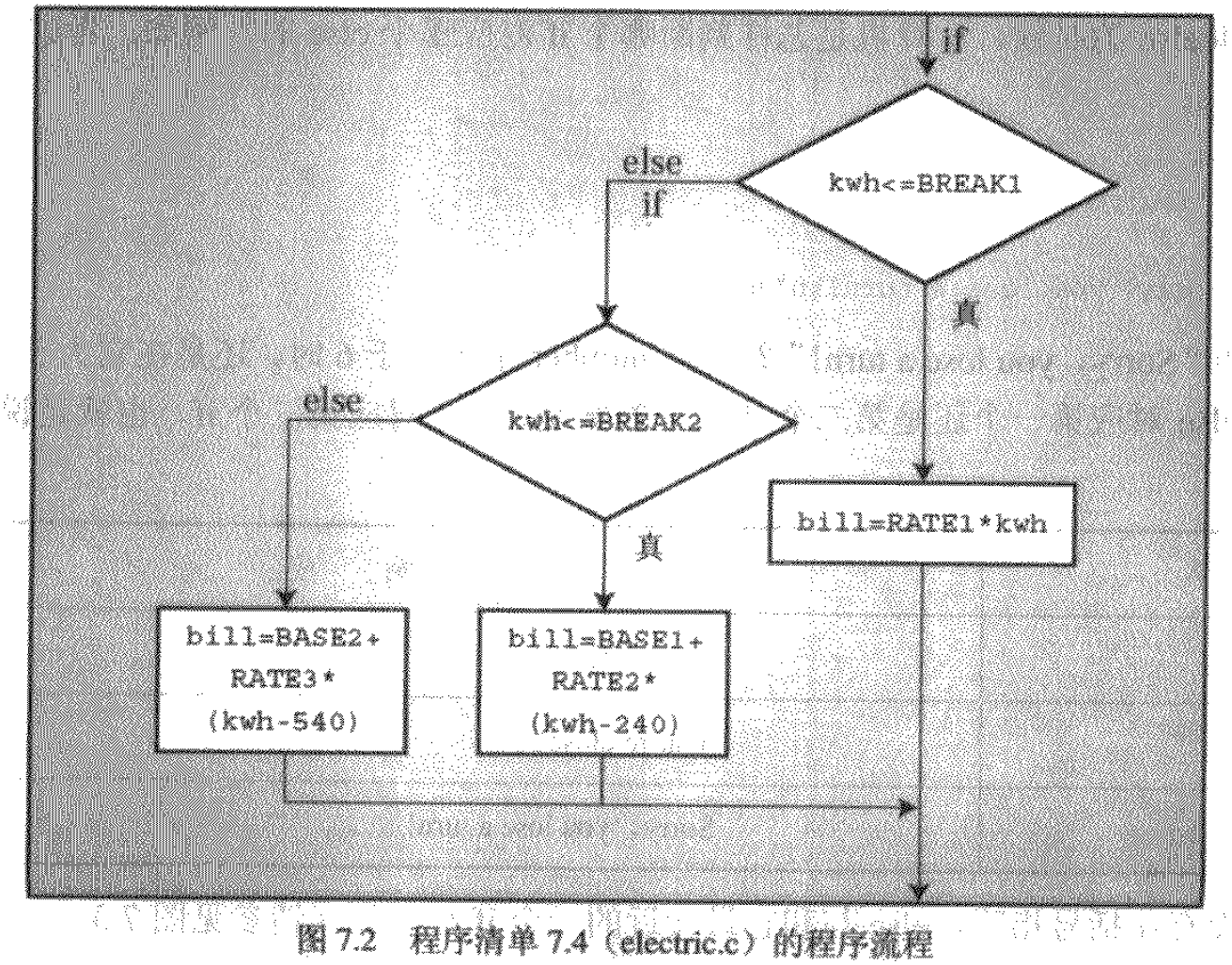


图 7.2　程序清单 7.4（electric.c）的程序流程

把else与if配对

如果没有花括号指明,else与它最接近的一个if相匹配

多层嵌套的if

```
// divisors.c --使用嵌套if显示一个数的约数
//给定一个整数,显示它的所有约数,若没有约数,则报告该数为素数
#include <stdio.h>
#include <stdbool.h>
int main(void)
{
    unsigned long num;          // 可检查的约数
    unsigned long div;          //可能的约数
    bool isPrime;               // 素数的标志

    printf("Please enter an integer for analysis; ");
    printf("Enter q to quit.\n");
    while (scanf("%lu", &num) == 1)
    {
        for (div = 2, isPrime = true; (div * div) <= num; div++)
            //为了提高计算效率,改进了循环界限;乘法比开方计算效率高;使用了逗号运算符
        {
```

```
            if (num % div == 0)
            {
                if ((div * div) != num)
                    printf("%lu is divisible by %lu and %lu.\n",
                           num, div, num / div);
                else
                    printf("%lu is divisible by %lu.\n",
                           num, div);
                isPrime= false; // 不是一个素数
            }
        }
        if (isPrime)
            printf("%lu is prime.\n", num);
        printf("Please enter another integer for analysis; ");
        printf("Enter q to quit.\n");
    }
    printf("Bye.\n");

    return 0;
}
```

- 通过改进循环界限,可以大大提高计算效率
- 整数乘法比求平方根更快
- if else语句很长时,可通过加上花括号增加程序的可读性和易用性
- 通过加上`stdbool.h`头文件,可以用`bool`代替关键字`Bool`表示这种类型,并用标识符`true`和`false`代替`1`和`0`
- 通过设置作为标志的变量,可以进行特定的操作(如这里素数的标志为1(即true))

总结:if语句

总结：使用 if 语句进行选择

关键字：

if，else

总体注解：

下列每种形式中，语句部分可以是一个简单语句或者是一个复合语句。一个真表达式意味着它具有非零值。

形式 1：

```
if (expression)
     statement
```

如果 expression 为真则执行 statement。

形式 2：

```
if (expression)
     statement1
else
     statement2
```

如果 expression 为真，则执行 statement1；否则执行 statement2。

形式 3：

```
if (expression1)
     statement1
else if (expression2)
     statement2
else
     statement3
```

如果 expression1 为真，则执行 statement1；如果 expression1 为假而 expression2 为真，则执行 statement2；否则，如果两个表达式都为假，执行 statement3。

例如：

```
if (legs == 4)
     printf ("It might be a horse.\n");
else if (legs > 4)
     printf ("It is not a horse.\n");
else    /* case of legs < 4 */
{
     legs++;
     printf ("Now it has one more leg.\n")
}
```

获得逻辑性

运算符	含义
&&	与
||	或
!	非

优先级

优先级从高到低:

`()` > `!` = `++` > `*` > `&&` / `||` > `=`

求值顺序

- 逻辑表达式从左至右求值
- `&&` 与 `||` 运算符是序列的分界点,程序从一个操作数到下一个操作数之前所有的副作用都会生效
- 一旦某个元素使表达式失效,求值停止

总结:逻辑运算符和表达式

总结：逻辑运算符和表达式

逻辑运算符：

逻辑运算符通常使用关系表达式作为操作数。!运算符带一个操作数。其他两个逻辑运算符带有两个操作数：一个在左边，一个在右边。

运算符	意义
&&	与
||	或
!	非

逻辑表达式：

当且仅当两个表达式都为真时，expression1&&expression2 为真。如果其中一个为真或两个表达式都为真，expression1||expression2 为真。如果 expression 为假，则!expression 为真，反之亦然。

求值顺序：

逻辑表达式是从左到右求值的。一旦发现有使表达式为假的因素，立即停止求值。

例如：

```
6 > 2 && 3 == 3          为真
!(6 > 2 && 3 == 3)       为假

x != 0 && (20 / x) < 5 只有当 x 不为 0 时，才计算第二个表达式的值
```

范围

可用 `&&` 运算符测试范围,如

```
if (range >= 90 && range <= 100)
printf("Good show!\n");

if (ch >= 'a' && ch <= 'z')//等价于if (islower(ch)),推荐这种,因为其移植性更好
printf("That's a lowercase character.\n");
```

统计字数的程序

> 关键思想:
>
> - 识别行:检测换行符
> - 识别不完整的行:追踪STOP前一个字符,不是换行符,就计为不完整行
> - 识别单词:单词从一个空白字符到下一个空白字符之间的字符串;为了判断一个字符是不是在一个单词里,可以读入一个单词的首字符时把一个标志(这里命为inword)设置为1

```
// wordcnt.c -- counts characters, words, lines
#include <stdio.h>
#include <ctype.h>          // for isspace()
#include <stdbool.h>        // for bool, true, false
#define STOP '|'
int main(void)
{
    char c;                 // read in character
    char prev;              // previous character read
    long n_chars = 0L;      // number of characters
    int n_lines = 0;        // number of lines
    int n_words = 0;        // number of words
    int p_lines = 0;        // number of partial lines
    bool inword = false;    // == true if c is in a word

    printf("Enter text to be analyzed (| to terminate):\n");
    prev = '\n';            // used to identify complete lines
    while ((c = getchar()) != STOP)
    {
        n_chars++;          // count characters
        if (c == '\n')
            n_lines++;      // count lines
        if (!isspace(c) && !inword)//使用isspace()函数识别空白字符
        {
            inword = true;  // starting a new word
            n_words++;      // count word
        }
        if (isspace(c) && inword)
            inword = false; // reached end of word
        prev = c;           // save character value
    }

    if (prev != '\n')

        p_lines = 1;
```

```
    printf("characters = %ld, words = %d, lines = %d, ",
           n_chars, n_words, n_lines);
    printf("partial lines = %d\n", p_lines);

    return 0;
}
```

条件运算符

总结：条件运算符

条件运算符：

?:

总体注解：

这个运算符带有三个操作数，每个操作数都是一个表达式。它们如下排列：

expression1 ? expression2: expression3

如果 expression1 为真，整个表达式的值为 expression2。否则为 expression3 的值。

例如：

（5>3）?1：2 等于 1。

（3>5）?1：2 等于 2。

（a>b）?a：b 取 a 和 b 中较大的值。

循环辅助手段:continue和break

continue语句

- 可用于三种循环形式中.当运行到该语句时,剩余的迭代部分将被忽略,开始下一次迭代.当处于嵌套结构中,仅仅影响包含它的最里层结构.
- continue两种应用场合:
    1. 当循环剩余部分放在一个else代码块中,可用continue语句代替,如

```
if (score < 0 || score > 100)
/* printf() statement */
else
{
/* statements */
}
```

或

```
if (score >= 0 && score <= 100)
{
    /* statements */
}
```

> 这种情况下,使用continue的好处是:消除主语句的一级缩排.当语句很长或有很深的嵌套时,可增加语句的可读性

2. 可作为占位符使用,如

```
while(gechar()!='\n')
    ;
```

可改为:

```
while(gechar()!='\n')
    continue;
```

> 通过加入continue在空语句中,增加了可读性

注意:如果不能简化代码,反而使其复杂化,不使用,如

```
while ((ch = getchar() ) != '\n')
{
if (ch == '\t')
continue;
putchar(ch);
}
```

通过对if判断取逆,可简化为:

```
while ((ch = getchar()) != '\n')
if (ch != '\t')
putchar(ch);
```

break语句