杭州电子科技大学

C 语言要点总结

草稿

邬惠峰

2011-5-20

目录

第 1 章　C 语言程序的基本形式

1.1　C 语言的结构

C 语言程序都是由一个或多个函数(Function)构成。一个 C 程序至少必须存在一个函数“main()”。它是程序运行开始时调用的一个函数。它表明该程序完成动作轮廓。C 语言程序的基本形式如下：

```
void main()
{
   变量说明语句；
   执行语句；
}
```

main()为主函数，**只能有一个**。执行语句中，可有其他函数，但不能用 main 为函数名。许多常用的函数做成标准函数与 C 编译器一起提供给用户，这就是标准库函数。

例 1.1：C 语言程序

```
/ *    This program is sum of two integer and the results * /

void main()
{
   int a，b，sum；     /*说明 a，b 和 sum 为整型变量*/
   a=123；
   b=456；
   sum=a+b；
   printf(“The sum of %d and %d is %d\n”，a，b，sum)；
 }
```

此例中，只有一个函数——主函数 main()。第一个语句是说明 a，b 和 sum 这些变量的语句，说明它们都是整数(int)型的变量。所有语句都放在左右花括号之内，各语句之间以分号“；”结束。

1.2 标识符

C 语言中所使用的每个函数和变量都应有**唯一**的名称，这样才能被识别和使用。通常，这种函数和变量名称用一串字符表达，称为标识符。C 语言使用的标识符有严格限制：

(1) 必须以**字母或下划线开头**，；

(2) 必须由**字母，数字或下划线**组成；

(3) 大小写字母是有区别的；

(4) 不允许用一些保留字(或叫关键字)。

例: 正确的函数或变量名： _abc, veb7, lev_5
错误的函数或变量名称的实例：3H, sUM$,char

1.3 保留字(即关键字)

在 C 语言中保留字或关键字并不太多，原先规定有 28 个，新标准规定改为 32 个，如下所示:

auto default extern long static void break do for
register struct volatile case double float return
switch while char enum goto sizeof typedef continue
else if signed union int short unsigned

1.4 基本数据类型

C 语言可以使用多种数据类型

(1)基本类型

- 整数类型 int
- 实数类型(浮点类型)：
  单精度浮点型 float
  双精度浮点型 double
- 字符类型 char
- 枚举类型 enum

(2)构造类型

- 数组类型 []
- 结构类型 struct
- 联合类型 union

(3)指针类型 &, *

(4)空类型 void

1. 整数型(即 integer——int 型)

整数型常量不仅可以用十进制表示，也允许用八进制或十六进制表示，例如：

62 十进制数
053 八进制数 **以 0 打头**，由 0～7 构成。
0x32 十六进制数 **以 0x 打头**，由 0～9 和 A～F 构成。

如 0x2C2 即(2×16＋12)×16＋5＝709

整数类型又可细分成不同长短的类型，应加上类型修饰符构成，即：

short int 可简化为 short

long int 可简化为 long

unsigned int　　可简化为 unsigned

2．浮点类型(即 float 型)

实数类型也叫浮点类型。可包含有整数部分和小数部分，例如：

0.012 等同于　.012

5.0 等同于 5.

科学计数法中，则用“尾数＋e＋指数”表示浮点数值，e 即 exponent(指数)例如：

6.3e5 等同于 630000.0

其中，6.3 就是尾数，含有整数部分(integer part)为 6 和小数部分(fractional part)为 3，而 5 就是指数 (exponent) 部分。尾数和指数也都有可能为负值，例如：

-1.23e4

12.34567e-8

-78e-12

3.字符类型(char 型)

字符类型的数据代表一个字符，由一对单引号将字符括起来，表示的是该字符在 ASCII 码表中的代码值，例如：

‘a’　　即 97

‘A’　　即 65

它们占 1 byte。

ASCII 码表中的某些控制字符不可显示，则通过加反斜线“\”的转义字符表示，例如：

‘\0’表示 NULL(空)　　即 0

‘\t’　　表示 tab(制表)　　即 9

‘\n’　　表示 new line(新行或换行)　　即 10

‘\r’　　表示 return(回车)　　即 13

‘\\’　　表示反斜线\　　即 92

例 1.2：ASCII 码值。

```
#include<stdio.h>
void main ( )
 {
    char c1, c2;     /*定义字符类型*/
    c1=97; c2=98;    /*赋整型值*/
    printf("%c # %c\n", c1,c2);/*以字符类型输出*/
 }
```

例 1.3: 小写转大写

```
void main ( )
{
  char c1, c2;              /*定义字符类型*/
  c1='a'; c2='b';
  c1=c1-32; c2=c2-32;   /*整型数值运算*/
  printf("%c    %c", c1, c2);
}
```

1.5 算术表达式

1. C 语言中算术运算符

算术表示达式由变量、常量及算术运算符构成。在 C 语言中算术运算符有：　　＋、－　、　*、　／、　％　、－－　、＋＋

＋，－，*，和／为四则运算符，和日常概念没有区别，其中*和／优先于＋和－。

%为取模(Modulus)运算符，是针对**整数**运算，即取整数除法之后，所得到的余数，例如：

10%3＝1　　即 10 对 3 取模，结果为 1。

13%8＝5　　即 13 对 8 取模，结果为 5。

－－为自减 1，＋＋为自增 1。

n＋＋或＋＋n 都变量 n 自增 1，最终结果与 n＝n＋1 等效。但处理过程却有所区别。＋＋n，表示 n 先自增 1，然后进到具体的式子中运算；n＋＋，则 n 本身先进入式中运算，最后 n 再增 1。

例如：已知 n＝6，则

m＝＋＋n；结果为：m＝7，n＝7

m＝n＋＋；结果为：m＝6，n＝7

n－－与－－n 同样类似。

2.数据类型与运算结果的关系

(1) 同类型数据运算结果仍保持原数据类型

整型数的除法得到的结果仍是整型数，小数部分将被去掉，例如：

5／2＝2

而不是 2.5。浮点数的除法得到的仍是浮点数，例如：

5.0／2.0＝2.5

(2) 不同数据类型混合运算，精度低的类型往精度高的类型转换后，再做运算。例如：

5.0／2＝2.5

1.6　C 语言的基本语句

1. C 语言的特点

(1) 所有 C 语句都以“；”分号结尾。一条语句可以不止一行，不必加续行符，只根据“；”来确定语句结束。两条或多条语句也可写在同一行，用“；”分开。

(2) 语句可从任一列位置开始，每行开头，有多少空格都可以，但为了可读性好，通常习惯还是按一定规律缩进。

2. 变量说明语句

变量说明语句的主要作用就是定义变量类型，其格式是：

类型说明符　变量 1［，变量 2，…］;　例如：

int number；

char a，b，c；

float t1；

3. 赋值语句

赋值语句是将常量或算术表达式的运算结果赋给变量，其格式是：

变量名＝常量或算术表达式；

例如：

int number；

number＝10；

例 1.4: 赋值操作

```
void main()
{
   int n1，n2，n3；
   int total；
   n1＝1；
   n2＝2；
   n3＝3；
   total＝n1＋n2＋n3；    /* 赋值操作*/
 }
```

4．基本输入输出语句

printf()和 scanf()是 C 语言的基本输入输出函数，都放在标准函数库中，为了使用它们，应在程序开头加上：

#include　<stdio.h>

基本输入输出语句就是直接调用这两个基本输入输出函数。

(1)输出语句

一般格式是：

printf (“控制串”［，表达式 1，…，表达式 n］;

控制串(或格式串)是用双引号括起来的输出格式控制说明。控制串中每一个变量都应当与后面相应的某个表达式对应。

控制串分两部分，即：要显示的字符和格式串。格式串以“%”开头，后跟格式码。格式串与参数一一对应。含有不同格式码的格式串表示显示不同的内容，如下所示：

%c　显示字符

%s　显示字符串

%d　以十进制格式显示整数

%o　以八进制格式显示整数

%x　以十六进制格式显示整数

%u　显示无符号整数

%f　显示浮点数

%e　以科学计数法显示数字

格式码前可加修改量以便更好地控制显示格式，主要有：

• 字符宽度控制，例如：

%4d　显示十进制整数，至少给 4 个数字位置

%10s　显示字符串，至少给 10 个字符位置

• 精度控制，例如：

%10.4f　显示浮点数，共占 10 位，小数点后取 4 位

%5.7s　显示字符串，最少占 5 位，最多占 7 位

• L 或 h

%Ld　显示十进制长整数

%hd　显示十进制短整数

%Lf　显示双精度浮点数

• 显示位置默认为右对齐，若加负号(即“－”)，则为左对齐，例如：

%d　右对齐显示十进制整数

%—d 左对齐显示十进制整数

例如：

prinft(“Welcome!”)；

屏幕上显示结果：

Welcome!

例如:

prinft(“Welcome\n”)；

与前例不同是后面加了一个换行符。

例如:

int number＝10；

printf(“The number is %d\n”，number)；

应显示：

The number is 10

例如:

float valuel，value2，value3；

value1＝2.3；

value2＝4.5；

value3＝6.7；

printf(“The average of %of and %f and %f is %f\n”，value1，value2，value3，

(value1＋value2＋value3)／3.0)；

应显示：

The average of 2.3 and 4.5 and 6.7 is 4.5

%d 和%f 表示在相应的位置显示的数据类型，且一一对应。%d 表示要显示整型数，%f 表示要显示浮点型数。

(2)输入语句

一般格式是：

scanf(控制串，地址表达式 1［，地址表达式 2，……，地址表达式 n］;

控制串(或叫格式串)与前述 printf()中的控制串类似，也包含有以“%”开头加格式码组成的格式串。控制串是用双引号括起来的输入格式控制说明。地址表达式所列出的应当是变量的地址，而不是变量名。每个地址表达式的值，对应于前面控制串中某一格式变量的地址。如:

```
int number;
scanf(“%d”，&number);
```

其中，%d 表示应以整型格式输入，&number 表示指向 number 的地址。

注意:

控制串中的非空白符，例如：

```
scanf(“%d，%d”，&i，&j)
```

上式中“%d”之间有逗号，输入时也应加逗号。还可用修改控制域宽，例如：

```
%20s 就只取前 20 个字符
%s 取全串
```

例如 ：

```
float average;
scanf(“%f”，&average);
```

其中，%f 表示应以浮点型格式输入，&average 表示指向 average 的地址。

第 2 章　流程控制

2.1 控制表达式

程序的流程走向是由条件表达式的值控制决定的。表达式总是有值的。

1．逻辑表达式

逻辑表达式由变量、常量和逻辑关系运算符构成，用以表示变量的逻辑关系。

(1) 逻辑与 (AND)　　&&

(2) 逻辑或 (OR)　　||

(3) 逻辑非 (NOT)　　!

逻辑表达式只有两种可能的取值，真假必取其一。假者取值为 0；真者取值为 1。

2．关系运算符

变量与变量或常量之间比较数值上大小的关系用关系运算符来表示。关系运算符共有六种，即：

＝＝　　等于

!＝　　不等于

<　　小于

<＝　　小于等于

>　　大于

>＝　　大于等于

关系表达式也只有两种可能的取值,例如:

'a'＝＝'b'

此式取值 0，值为假(false)。 又如，已知变量 x＝'c'，则表表达式

x>'b'

值为真(true)，取值 1。

3. 运算操作符的结合性和优先级

运算操作符的结合性和优先级规定了表达式运算操作的处理顺序。优先级高的操作应优先执行。

例如，处理表达式

a＋b*c 时，*优先于＋，相当于 a＋(b*c)。

有部分运算操作符的结合性是从右向左。最常见的赋值表达式中，赋值运算操作符的结合性就是自右向左。例如，

a＝b＋c 是将 b＋c 结果赋给 a

又如:

a*＝b＋c 相当于 a＝a*(b＋c)。

单操作数的运算操作符＋＋(自增 1)，－－(自减 1)，－(负号)的结合性也都从右向左。例如，

－i＋＋相当于－(i＋＋)，而不是(－i)＋＋

关于运算符的优先级低于算术运算符，例如：

a<b＋c 等同于 a<(b＋c)

2.2 for 循环语句

1. for 循环语句的一般格式为：

for(初始化表达式；循环条件；循环表达式)

循环体语句块

例 2.1：在屏幕上，显示 1～100

```
#include   ＜stdio.h＞
void main()
{
    int n；
    for (n＝1；n＜＝100；n＋＋)
        printf(“%d\n”，n)；                    /*循环体语句*/
 }
```

for 语句中“；”隔开的各部分：

(1)初始化表达式，用于循环开始前，为循环变量设置初始值。

(2)循环条件是一个逻辑表达式，若该式取值为真即条件满足，则继续执行循环；否则，执行循环体后的语句。

(3)循环表达式定义了每次循环时，循环变量的变化情况。

(4)循环体可以是一条语句或一组语句，若是一组语句，需要用“{”和“}”括起来。

2. for 语句执行过程

(1)计算初始表达式。

(2)判断循环条件，若满足，则执行；否则，退出循环。

(3)执行循环。

(4)返回第(2)步。

例 2.2：显示十的阶乘 10！

```
＃include   ＜stdio.h＞
void main()
{
   int n，result；
   result＝1；
   for(n＝1；n＜＝10；n＋＋)
        {
          result*＝n
          printf(“%d!＝%d\n”，n，result)；
        }                          /*    循环体语句    */
 }
```

例 2.3：打印可印刷字符的 ASCII（从 32 到 126）代码字符对照表。

```
#include <stdio.h>
main ()
{
   int i;
   for (i=32;i<127; i++)       /*  从 ASCII 码为 32 的开始  */
   {
     printf("%4d    %c",i,i);   /*  打印 ASCII 码及其对应的字符  */
     if ((i+4)%5 == 0)      printf ("\n");   /*  每行打印 5 个字符后换行  */
   }
}
```

3．嵌套的 for 语句

在 for 的循环体中还包含 for 语句，这就是嵌套的 for 语句。

例 2.4：计算五个给定数的阶乘

```
＃include   ＜stdio.h＞
void main()
{
   int i,j,number，result；
   for(i＝1;i＜＝5;i＋＋)
   {
       scanf(“%d”，&number)；
       result＝1；
            for(j＝1；j＜＝number；j＋＋)
                  result *＝j；                    /*内循环体语句*/
       printf(“%d!＝%d\n”，number，result)；
   }                                /*外循环体语句*/
 }
```

2.3 while 语句

按某一条件循环，可以不知道循环的次数，直到不满足该条件时，才停止循环。

1.while 语句的一般格式为：

while(循环条件)

循环体语句块

例 2.5：显示 1～100

```
#include  <stdio.h>
void main()
{
   int n;
   n=1;
   while(n<=100){
      printf("n=%d", n);
      n++;
    }
 }
```

2.while 语句的执行过程

(1) 判断循环条件是否满足，不满足就退出循环体。

(2) 执行循环体。

(3) 返回第(1)步。

如果循环条件根本不能成立，则永不执行循环体；反之，若循环条件总是成立，则成为永久循环(死循环)。

例 2.6：倒排输出数字

```
#include <stdio.h>
 void main()
 {
     int number,rdigit;
     scanf("%d",&number);
         while(number!=0){
            rdigit=number%10;                    /*取出低位数*/
            printf("%d", rdigit);
           number / =10;
         }
     printf("\n");                  /*循环跳出后换行*/
  }
```

输入的数若为 12345 则输出 54321。

例如：利用 while 语句和空语句跳过所有的空白字符

```
while(c=getchar())==');
```

例如：永久循环

```
while(1){
   ...
      }
```

2.4 Do- while 语句

Do- while 语句为当型循环结构,执行时先执行循环体语句再判断循环条件。

1.do- while 语句的一般格式为：

```
do
    循环体语句块
while(循环条件)；
```

例 2.7：倒排数字输出

```
#include＜stdio.h＞
void main()
 {
  int number，rdigit；
  printf(“Please input a number：”)；
  scanf(“%d”，&number)；
     do{                              /*先执行*/
        rdigit＝number%10；
          printf(“%d”，rdigit)；
          numbe/＝10；          /*   等效于 number= number／10
      }while(number!＝0)；        /*后判断*/
   }
```

2. 执行过程

(1) 执行一次循环体。

(2) 判断是否满足循环条件，若满足，则循环，转到(1)继续执行；否则，执行随后的语句。

2.5 if 语 句

1.简单的 if 语句

简单的 if 语句格式为：

```
if(条件表达式)
    语句块
```

只要条件表达式满足逻辑关系，即表达式的值为真，就执行语句块。

例 2.8：比较两个数

```
#include＜stdio.h＞
void main()
{
    int m，n；
    printf(“Please input two numbers\n”)；
    scanf(“%d %d”，&m，&n)；
    if(m>n)
       printf(“The first number is bigger\n”)；
}
```

2.复合句的 if 语句

复合的 if 语句格式为：

```
if(逻辑表达式)
    语句块 1
else
    语句块 2
```

若满足逻辑关系，即逻辑表达式的值为真，则执行语句块 1；否则，执行语句块 2。

例 2.9：比较两个数的改进

```
#include   <stdio.h>
main()
   {
      int m，n；
      printf(“Please input two numbers\n”)；
      scanf(“%d %d”，&m，&n)；
      if(m>n)
          printf(“The first number is bigger\n”)；
      else                          /*  相当于 m<n */
          printf(“The first number is bigger\n”)；
    }
```

例 2.10：检查闰年

```
/ * This program determines if a year is a leap yaer * /
#include   <stdio.h>
main()
{
    int x；
    printf(“Please input two numbers\n”)；
```

```
    scanf(“%d”，&x)；
    if(x%4＝＝0&&x%100!＝0||x%400==0)
        printf(“%d is a leap year\n”，x)；
    else                    /* 不满足上述条件 */
        printf(“%d is not a leap year\n”，x)；
}
```

可见，这种复合的 if 语句是等同于两个简单的 if 语句，即：

```
if(逻辑表达式)
        语句块 1
if(  逻辑表达式)
        语句块 2
```

3.嵌套的 if 语句

在 if 语句的语句块中还可包含 if 语句，这就形成嵌套的 if 语句。其格式为：

```
if(逻辑表达式 1)
   if(逻辑表达式 2)
        语句块 1
   else
        语句块 2
else
     语句块 3
```

例 2.11：　有一个函数　y=　-1　(x<0)；0　(x=0)；1　(x>0)

```
void main ( )
{
   int x,y;
   scanf ("%d",&x);
   if (x<0) y=-1;
      else if (x==0) y=0;          /* else 包括(x>0)和(x=0) */
           else  y=1;                  /* else 只包括(x>0) */
   printf ("x=%d,y=%d\n",x,y);
}
```

此例 if 与 else 是成对的，比较容易理解。 实际上也可能在嵌套中只有 if，而没 有 else，这样就容易造成错误。因为 else 总是与前面最相近的一个不包含 else 的 if 语句对应匹配，为避免发生这种错误，应将嵌套中的 if 语句用“{”和“}”括起来，即：

```
if(逻辑表达式 1){
    if(逻辑表达式 2)
```

```
            语句块 1
                }
        else
            语句块 2
```

4. else-if 语句

实际上，这也是一种嵌套，在 else 下再嵌套 if_else，形成如下格式：

```
if(逻辑表达式 1)
    语句块 1
else if(逻辑表达式 2)
        语句块 2
    else if(逻辑表达式 3)
            语句块 3
        …
            else
                语句块 n
```

例 2.12：用 else-if 语句，写一个程序，输入一个数 x，打印出符号函数 sign(x)的值。

$$\text{符号函数为 } sign(x)=\begin{cases}-1 & x<0\\ 0 & x=0\\ 1 & x>0\end{cases}$$

```
#include   <stdio.h>
void main()
{
   int number，sign；
   printf(“Please type in a number x＝”)；
   scanf(“%d”，&number)；
   if(number<0)
      sign＝－1；
   else if(number＝＝0)        /*else 包括=0 和>0 两者*/
         sign＝0；
      else
            sign＝1；
         printf(“sign(x)＝%d\n”，sign)；
 }
```

2.6 条件表达式运算符

条件表达式运算符的一般格式为：

条件 ？表达式 1：表达式 2

这是一个三元运算符，它控制三个操作数，第一个操作数是条件，条件通常是一个逻辑关系表达式。该条件若为真，取表达式 1 的值，就是整个条件表达式的运算结果；否则，取表达式 2 的值作为整个条件表达式的运算结果。

例 2.13：求最大值.

```
#include <stdio.h>
 main()
{
 int a，b，c；
 scanf(“%d %d”，&a，&b)；
 c＝(a>b)?a:b；              /*  取 a 和 b 中大者赋给 c    */
 printf(“The maxum is %d”，c)；
 }
```

2.7 break 语句

break 语句用在 for，while，do 以及 switch 语句中。当执行一个循环体时，在某一特定状态出现 break 语句时，立即退出循环体，且只退出本级循环。

例 2.14：　九九乘法表

```
#include   <stdio.h>
 main()
 {
     int i，j；
       for(i＝1；i<＝9；i＋＋){
           for(j＝1；j<＝9；j＋＋){
                if(j>i)break；       /*  被乘数大于乘数时跳出内循环   */
                  printf(“%d “，i*j)；
                                          }
           printf(“\n”)；
                                       }
  }
```

其结果将打出三角形的乘法表，即

```
1
2   4
3   6   9
```

4　8　12　16

…

9 18 27 36 45 54 63 72 81

2.8 continue 语句

continue 语句执行后,在本次循环中其他语句均不执行，转而继续执行循环体的下一次循环。在 while、do 循环中意味着立即转入条件测试，在 for 循环中意味着立即转到执行 i＋＋，而进入下一次重复。

例 2.15：只做正数的累加而跳过负数

```
#include <stdio.h>
 main()
 {
  int a，i，n，sum；
  scanf(“%d”，&n)；
  sum＝0；
    for(i＝0；i<n；i＋＋){
        scanf(“%d”，&a)；
        if(a<0)
           continue；
           sum＋＝a；
                                  }
   printf(“The sum＝%d\n”，sum)；
 }
```

2.9 switch 语句

switch 语句的一般格式为：

```
switch(表达式)
{
 case 常量 1：
     语句块 1
     break；
 case 常量 2：
     语句块 2
     break；
  …
 case 常量 n：
     语句块 n
     break；
```

```
    default：
        语句块 n＋1
        break；
    }
```

它是根据表达式为不同的值而执行不同的语句块，可以看成是特殊的 else_if 语句。即：

```
if(表达式＝＝常量 1)
    语句块 1
else if(表达式＝＝常量 2)
    语句块 2
    …
    if(表达式＝＝常量 n)
        语句块 n
    else
        语句块 n＋1
```

但用 switch 语句则更方便，实际上也用得更多一些。

第 3 章　数组与字符串

3.1 一维数组

1.数组及其特点

实际处理的数据，往往不仅是一个，而是量大但性质相同的数据。例如，一个班的学生成绩等等。它们是相同性质的数据。如果仍一个个加以说明，麻烦易错。改用数组，这就方便多了。例如：

int　grade［n］;

数组是相关变量的有序集合，其中所有的变量具有相同的数据类型。数组的特点：

(1) 各元素是同一种类型的变量。

(2) 各元素按顺序排列，其位置由下标来确定。

(3) 各元素可独立地作为一个基本变量被赋值和使用。

2．一维数组说明和下标

(1) 一维数组说明格式为：

类型说明符　数组名［size］

其中，size 为数组的大小，即组成该数组元素的个数。

(2) 数组的下标

C 语言中，数组的下标从 0 开始，到 size－1 为止。例如：

int x［5］

就有下标从 0 到 4 共 5 个元素，即：x［0］，x［1］，…，x［4］。

例 3.1：产生斐波那契(Fibonacci)数列前 16 项

```
#include <stdio.h>
 void main()
 {
     int Fibonacci［16］，i;
     Fibonacci［0］＝0;
     Fibonacci［1］＝1;
     for(i＝2；i＜16；i＋＋)
          Fibonacci［i］＝Fibonacci［i-2］＋Fibonacci［i-1］;
     for(i＝0；i＜16；i＋＋)
          printf(“%d，”，Fibonacci［i］);
     printf(“…\n”);
  }
```

执行结果：

0，1，1，2，3，5，8，13，21，34，55，89，144，233，377，510，…

3.2 多维数组

多维数组说明格式为：

类型说明符　数组名［size1］［xize2］…［sizen］;

其中，size1，size2，…，sizen 为数组各维的大小。组成该组元素的总个数应为 size1×size2×…×sizen。

例如：

int d［10］［20］;

说明这是一个 10×20 二维数组，即：

d［0］［0］ d［0］［1］…d［0］［19］

d［1］［0］ d［1］［1］…d［1］［19］

…

d［9］［0］ d［9］［1］…d［9］［19］

例 3.2 将一个二维数组行和列元素互换,存入另一个二维数组里

```
#include <stdio.h>
 void main ()
 {
       static int a[2][3]={{1,2,3},{4,5,6}};
       static int b[3][2],i,j;
       printf("array a : \n");
       for (i=0;i<=1;i++)              /*循环变量作为行坐标*/
       {
            for (j=0;j<=2;j++)        /*循环变量作为列坐标*/
            {
                 printf("%5d",a[i][j]);
                 b[j][i]=a[i][j];        /*行列互换*/
            }
            printf("\n");
        }
        printf("array b : \n");
        for (i=0;i<=2;i++)         /*循环变量作为行坐标*/
        {
             for (j=0;j<=1;j++)         /*循环变量作为列坐标*/
                  printf("%5d",b[i][j]);
             printf("\n");
        }
 }
```

3.3 数组元素初始化

1．数组的初始化

C 语言允许在说明时对全局数组和静态数组初始化，即给各个元素赋以初始值，例如：

```
void main()
{
   static int counters［5］＝{0，1，2，3，4}；
   static char letters［5］＝{‘a’，’b’，’’c，’d’，’e’}；
   static int   M［4］［5］＝{{10，5，－3，－17，82}，{9，0，0，8，－7}{32，
                              10，20，1，14}，{0，0，8，7，6}}；
   static int x［5］＝{0，1，2}；        / *后面两个默认为 0* /
    …
 }
```

当给出初始值的个数少于数组全部元素个数时，未列出的后若干个初始值默认为 0。

2.变长数组的初始化

有时，不直接给出数组大小，作为变长数组出现，也照样可以初始化。例如：

```
static int a［］＝{1，2，3，4，5}；
ststic float f［］＝{1.2，2.5，3.8，4.0}；
static char word［］＝{‘H’，’e’，’1’，’1’，’o’，’　’，’\0’}；
static char error［］＝{“read error\n”}；
```

3.4 字符串

字符串要用一维字符数组处理，实质上就是以空字符结尾的字符数组。

1.以空字符结尾的字符数组

空字符就是 NULL，即’\0’。一个字符串可看成结尾为’\0’的不定长的一维字符数组。例如：

“a”等同于’a’，’\0’

“”等同于’\0’

“ “等同于’’，’\0’

“Hello!\n”等同于’H’，’e’，’1’，’1’，’o’，’!’，’\n’，’\0’

2.字符串常量

字符串常量是由一对双引号包括的除双引号之外的任何字符集合，也可看成一维的字符数组常量。若要包括双引号必须加反斜线“\”。

3.字符串的初始化

可采用类似于变长数组初始化处理。

(1) 变长数组初始化，例如：

```
static int　A［］＝{0，1，2，3，4}；
```

(2) 字符串初始化，例如：

```
static char x［］＝{‘H’，’e’，’l’，’l’，’0’，’!’，’\0’}；
```

由于是字符数组，可以直接用字符数组常量即字符串常量给出初始值，编译器将自动加入空字符’\0’。显然，下面这样将更简便些；

```
static char x［］＝{“Hello　”}；
```

4.字符串的显示

可用%s 的格式显示一个以’\0’结尾的字符串。系统显示字符串遇到空格符’\0’时，就结束该字符串的显示。例如：

```
static char x［］＝{“Hello!”}；
printf(“%s\n”，x)；
```

此语句的执行结果，将显示：

```
Hello
```

5.字符串的输入

可用%s 的格式输入一个字符串。例如：

```
char s1［20］;
scanf(“%s”，s1);
```

注意：• 在 s1 前，不加“&”。

• 输入字符串时，遇到空格或回车则结束。

6.单字符的输入

用 scanf()输入字符或字符串时，遇到空格或回车结束。这不便于文字处理，一般用getchar()函数输入单字符。例如：

```
#include <stdio.h>
void main
{
  char ch；
  ch＝getchar()；
 }
```

7.转义字符

一些特殊字符，须用反斜线来转义，例如：

\0	NUL	\b	退格
\n	回车换行	\r	回车不换行
\t	横向制表	\v	纵向制表
\’	单引号	\”	双引号
\f	换页	\\	反斜线

例 3.3 输出一个钻石图形。

```
main ()
{static char diamond[ ][5]=          /*第二维大小不能省略*/
 {{' ',' ','*'},{' ','*',' ','*'},{'*',' ',' ',' ','*'},{' ','*',' '
        ',' ','*'},{' ',' ','*'}};
  int i,j;
  for (i=0;i<5;i++)                   /*  逐行  */
    {
      for (j=0;j<5;j++)                /*逐列*/
      printf("%c",diamond[i][j]);
      printf("\n");
     }
     }
```

运行结果：

```
    *
   *   *
  *     *
```

```
*    *
  *
```

例 3.4　统计输入的一段文字有多少个单词。

```
/*“L6-8 */
 #include "stdio.h"
 main ()
 {
 char string[81];
 int i,num=0,word=0;
 char c;
 gets(string);     /*  输入一段文字到一维数组 string */
  for (i=0;(c=string[i])!='\0';i++)
       if (c==' ')     /* c 为空格，没出现单词*/
            word=0;
       else if (word==0)    /* c 不为空格，但上次 c 为空格,出现单词  */
              {
                word=1;
                num++;              /*单词数累加*/
                 }
  printf("There are %d words in the line\n", num);
   }
```

3.5 字符串函数

一般 C 语言编译器都提供了字符串函数，放在文件 string.h 中，要使用字符串函数必须加有包含文件 string.h，即写成：

#include <string.h>

3.6 字符函数和字符转换及运算

1. 字符函数

一般 C 语言编译器的字符函数放在文件 ctype.h 中，要使用时必须加上包含文件 ctype.h，即写成：

```
#include     <string.h>
```

这类函数主要是用于字符类别判断及转换，例如：

isalnum(ch)　　若 ch 为字母或数字，则返回非零值；否则，返回零。

islower(ch)　　若是小写字母，则返回非零值；否则，返回零。

tolower(ch)　　将大写字母转换成小写字母

例 3.5 利用二维数组，找出三个字符串的最大者（ASCII 值）。

```
/*L6-9*/
main ()
{
   char string[20];
   char str[3][20];          /*每行一个字符串*/
   int i;
    for (i=0;i<3;i++)
        gets (str[i]);            /*  分别读入三个字符串*/
        if    (strcmp(str[0],str[1])>0)    /*  字符串比较  */
                strcpy(string,str[0]);    /*  字符串 str[0]复制到 string */
       else    strcpy(string,str[1]);
        if    (strcmp(str[2],string)>0)
                strcpy(string,str[2]);
    printf("\nthe largest string is:\n%s\n",string);
 }
```

第 4 章　函数与变量

函数是 C 语言的基本构件。利用这些基本构件，可以组成结构良好的大型程序。一个 C 源文件中可包含一个或多个函数，每个函数完成一个特定的功能。模块化编程在 C 语言程序设计中就是函数。

4.1 C 程序设计的一般形式

1. C 语言程序的一般格式

```
全局变量说明
main()
{
   局部变量说明                    主程序
   执行语句
  }
f1(形参定义)
{
   局部变量说明
   执行语句
}
  …                                函数
fn(形参定义)
{
    局部变量说明
    执行语句
}
```

例 4.1：输出 fa(n)函数值

```
#include <stdio.h>
int fa(int n)       /*  函数定义 fa，函数返回值的类型为 int*/
{
   if (n>0)    return (1);   /*  函数体，返回不同的值*/
      else return (0);
 }
main()
{
   int a;
   printf("请输入数据:\n");
   scanf(“%d”,&a);
   printf(“\n%d\n”,  fa(a));      /*  函数调用，用实参 a 替代形参 n */
```

```
}
```

例 4.2：将输入的大写字母转换为小写

```
/ *    converte a string in lowercaes    * /
#include   <stdio.h>
char uptolow();          /*  说明 uptolow()函数已定义  */
void main()
{
   char ch；
   do{
         printf(“Please type in a letter\n”)；
         scanf(“%c”，&ch)；
         printf(“%c”，uptolow(ch))；    /*  函数调用  */
       }while(ch!＝’’)；            / *   遇到空格字符则结束   * /
   printf(“\n”)；
 }
char uptolow(char k)          /*  函数定义  */
{
    if(k>＝’A’&&k<＝’Ｚ’)
      k＋＝’a’－’A’；
      return(k)；
}
```

4.2 函数

1.函数是 C 语言程序的基本构件

函数在 C 语言程序中，是完成某个算法的一个程序段，可以看成是程序的基本构件。函数的作用主要有:

(1)分解大任务

(2)利用现有成果

(3)增加可读性

例 4.3：求任意两个整数的阶乘之和

```
void main()
{
   int number1，number2，result；
   int factorial();
   printf(“Please type number1 and number2”)；
```

```
    scanf("%d %d"，&number1，&number2)；
    result＝factorial(number1)＋factorial(number2)；
    printf("number1!+ number2! ＝%d"，result)；
  }
  int factorial(int n)              /* 函数定义 */
  {
      int result1＝1；
      while(n!＝0)
      {
            result1 * ＝n；
            n--；
      }
      return result1；                  /*结果值 result1 返回 */
  }
```

2. 定义函数的一般格式

```
类型说明符 函数名(形参定义)
{
   局部变量说明
     函数体语句
}
```

注意：

- 函数也可能不需要传递参数，这时只保留括号，也无需对形参说明。
- 形参表内可含有多个变量，用逗号分隔开。

例如：

```
float sum(float a，float b)
{
  return(a＋b)；
}
```

例 4.4 函数 power()实现 xn,其中 n 是整数

```
#include <stdio.h>
double power(double x, int n)      /* 函数定义 */
{
  double p;
   if (n>0)
      for (p=1.0; n>0; n--)
      p=p*x;
```

```
  else p=1.0;
 return (p);          /* 返回函数值  */
}
void main ()
{
    double m;
    m = power(12.0,2);    /* 函数调用，参数类型，个数必须匹配 */
    printf(“\n 计算结果为: %e\n”,m);
    printf(“\n 计算结果为:%e\n”,power(12.0,3));
 }
```

4.3 函数返回值

1.函数返回值的类型说明

(1)函数返回值类型说明的一般格式为：

类型说明符　函数名()

(2) 函数默认类型为 int，可省去类型说明符。

(3) void f(b)　表示函数 f(b)没有返回值。

2.返回语句 return

有返回值的函数需要用返回语句来传送返回值。

(1)返回语句的一般格式为：

```
return；
```

或

```
return(返回变量名或表达式值)；
```

(2)返回语句的两种用途

(3)表示程序结束，从函数返回调用点,不返回函数的值，可以不用 return 语句。

例如：

```
printmessage()
{
   printf(“programming is fun \n”)；
   return；
}
```

2) 从函数返回调用点,并返回函数的值。

例如：计算阶乘

```
factorial3(int n)
{
    int result＝1；
    if(n＝＝0||n＝＝1)
        return(result)；
```

```
        while(n！＝0){
            result*＝n--;
        }
        return(result);
    }
```

在 return 语句括号内的 retult 是被返回的变量，带回函数值。

4.4 函数的调用

1.函数的参数

函数定义格式为：

类型说明符 函数名(形参表)

形参说明

函数体

函数的调用格式为：

函数名(实参表)

注意：

(1)函数形参表或实参表中，可有多个参数，也可能没有参数，但函数后仍须有()；

(2)实参类型应与形参类型一致，项数相同，且类型与项数一一对应；

(3) 函数返回值只能有一个，要想得到多个参数 a1，a2，…，值，则需定义全局变量 a1，a2，…，或用地址指针操作。

2.函数调用顺序及类型说明

(1)调用前，已对该函数加以定义说明。

例 4.5：调用函数求两数之和。

```
float sum(float a，float b)          /* 先定义函数 */
{
   return(a＋b)；
 }
void main()
{
   float first，second；
   first＝1.0；
   second＝2.0；
   printf(“%f\n”，sum(first，second)；      /* 后调用函数 */
 }
```

注意：简单的函数适于放在前头。因主函数中未对函数定义，放在后头易出错。

(2)函数调用前，已对函数返回的值加以说明。

在主函数中说明函数类型，对该函数的定义放在后面。

例 4.6：调用函数求两数之和。

```
void main()
{
  float first，second，sum()；    /* 说明函数 sum() */
  first＝1.0；
  second＝2.0；
  printf(“%f\n”，sum(first，second)；
 }
 float sum(float a，float b)          /* 后定义函数 sum(a，b) */
 {
   return(a＋b)；
 }
```

4.5 递归函数与递归调用

1. 递归函数

有些函数可用递归定义，称为递归函数。

例：n 的阶乘

$$n!=\begin{cases}1 & \text{当 } n=0 \text{ 时}\\ n*(n-1)! & \text{当 } n>0 \text{ 时}\end{cases}$$

2.递归调用

递归函数的计算程序，可采用递归调用算法实现。

例如：计算 n 的阶乘

```
factorial4(int n)
{
    int result；
    if(n＝＝0)
        result＝1；
    else
        result＝n*factorial4(n－1)；    /* 递归调用 */
    return result；
 }
```

4.6 外部函数和内部函数(了解)

1.基本概念

根据函数能否被其它源文件调用，将函数区分为内部函数和外部函数。主要解决不同源文件中函数之间的调用问题。

如果一个函数只能被本文件中其它函数所调用，称为内部函数。在定义函数时，指明其存储类是 extern,则该函数就是外部函数。当函数类型省略时，该函数为外部函数。

2. 内部函数和外部函数的定义

static（extern） 类型标识符 函数名（形参表）

如 static int fun(int a,int b) /*定义 fun 为内部函数，函数类型为 int*/

extern int f1(int a,int b) /*定义 f1 为外部函数*/

int f2(int a,int b) /*定义 f2 为外部函数，缺省时为外部类型*/

3.外部函数应用

便于程序设计结构化，将常用的函数放在一个文件中，其它的程序均可调用它。

例 4.7 外部函数

file1.c(源文件 1)

```
#include <stdio.h>
extern prime();  // 说明本文件中要用到其它文件中的函数
void main()
 {
    int   number;
    printf(“请输入一个正整数:\n”);
    scanf(“%d”,&number);
    if (prime(number))
        printf(“\n %d 是素数.”,number);
    else
    printf(“\n %d 不是素数.”,number);
 }
```

file2.c(源文件 2)

```
int prime(int number)     /*定义外部函数，此函数用于判别素数*/
{
    int flag=1,n;
    for (n=2;n<number/2 && flag==1;n++)
       if (number%n==0)      flag = 0;
    return flag;
 }
```

4.7 变量的存储类型

1. 局部变量

定义在某一函数或某一部分程序内部的变量，称为局部变量。局部变量仅在其所定义的局部范围内起作用，离开该范围，它们将自动消失，因此，又称为局部自动变量。

例 4.8：

```
#include   <stdio.h>
main()
```

```
{
  int x;                          /*局部变量*/
  x＝1;
  func1();
  func2();
  printf(“x＝%d\n”，x);    /*  值为 1 */
 }
 func1()
 {
   int x;                        /*局部变量*/
   x＝10;
   printf(“x1＝%d\n”，x);
   return;
  }
 func2();
  {;
    float x;                     /*局部变量*/
    x＝1.2;
    printf(“x2＝%f\n”，x);
    return;
  }
```

其中，局部变量 x 在三个函数中彼此无关。

执行结果：

x1＝10

x2＝1.2

x＝1

例 4.9：　局部变量 x 的变化

```
#include <stdio.h>
 main()
 {
   int x＝1;
     {
       int x＝2;
        {
          int x＝3;
          printf(“x1＝%d\n”，x);    /*   输出值为 3 */
         }                          /*   释放 x 值 3 */
        printf(“x2＝%d\n”，x);     /*   输出值为 2 */
      }                             /*   释放 x 值 2 */
```

```
        printf("x3＝%d\n"，x)；    /*  输出值为 1 */
    }
```

执行结果：

x1＝3

x2＝2

x3＝1

2. 静态局部变量

静态局部变量局限于定义它的函数之内，但并不随函数的退出而消失。

例 4.10：静态局部变量

```
/ *  Program to illustrate local and static variables *  /
#include    <stdio.h>
main()
{
   int i；
   for(i＝0；i<5；i＋＋)
       as()；
 }
as()
{
   int 1v＝0；
   static int sv＝0；    / *静态局部变量每次调用的保留上次调用的值  * /
    printf("local＝%d，static＝%d\n"，1v，sv)；
    1v＋＋；
    sv＋＋；
    return；
}
```

执行结果：

1ocal＝0，static＝0

1ocal＝0，static＝1

1ocal＝0，static＝2

1ocal＝0，static＝3

1ocal＝0，static＝4

静态局部变量的特点：

(1)只在第一次调用时，才作初始化赋值， 即编译时赋一次值。

(2) 下次调用的初值，是前一次退出时留下的值。

3．全局变量（全程变量）

定义在所有函数之外，可供所有函数访问的变量，称为全程变量或全局变量。

例 4.11：　求输入数的阶乘。

```
#include <stdio.h>
int number;                    /* 定义 number 全程变量  */
main()
{
   printf("Please type in the number \n");
   scanf("%d", &number);
   printf("%d！＝%d\", number, factorial2());
}
factorial2()
{
   int result＝1;              /* 定义 result 局部变量  */
   while(number！＝0)     /* 直接使用全局变量 number */
     {
       result*＝number;
       number--;
      }
   return(result);
 }
```

函数 factorial2()没有形参，可直接对全程变量 number 进行处理，函数调用时也不需要实参。

例 4.12 程序分析

（1）

```
#include <stdio.h>
main ()
{
 int i,j=1;     /* main 定义变量  */
 for (i=0;i<3;i++) /* 使用 main 中定义的变量 i */
   {
     int i=0;
     i=i+j++;            /* 使用循环体内定义的变量 i */
    printf("%d,",i); /* 使用循环体内定义的变量 i */
    }
printf("%d, %d\n",i,j);/* 使用 main 内定义的变量 i */
}
```

程序的输出结果:

1，2，3，3，4

(2)

```
#include <stdio.h>
int func();
int i=5;   /* 外部变量 */
```

```
void main ()
{
    int i,x;    /* 自动变量  */
    x=10;
    for (i=0;i<3;i++) /* 使用 main 中定义的变量 i */
     {
        int x=1;
        printf("%d,",++x);    /* x 先取值后自加 */
        x=func();
      }
    printf("%d, \n",x);
}
int func()
{
    printf("%d, ",i);
    i++;
    return(i);
}
```

程序执行结果:

2，5，2，6，2，7，10

（3）

```
#include <stdio.h>
 void inner();
 void st_pr();
 char c='A';
 void main ()
 {
     int i;
     for (i=0; i<3; i++)
     {
          inner();
          st_pr();
          printf("%c;",c);
          c++;
       }
 }
```

```
void inner ()
{
      char c='a';
      printf("%c",c);
      c++;
}
void   st_pr()
{
      static char c='Z';
      printf("%c",c);
        c--;
}
```

程序执行后的输出结果:

aZA; aYB; aXC;

4．寄存器变量

(1)寄存器变量的定义和说明

变量的存储类型除了真正使用内存单元的全程变量、局部自动变量和静态变量之外，还有一种称为寄存器变量。指定它使用寄存器存贮。其说明格式如下：

Register　类型说明符　变量名；

(2)寄存器变量的使用

1)使用限制

- 寄存器变量仅适用于整数和字符类型的变量名，不能用结构或数组等复杂数据类型；
- 一个程序模块只允许使用少量寄存器变量；
- 寄存器变量仅适用于局部变量，不能用于全程变量；
- 当指定的寄存器变量个数超过系统所能提供的寄存器数量时，多出的寄存器变量视同自动变量。

2)寄存器变量可提高程序的效率，应将使用频率最高的变量说明成为寄存器变量。一般常用于循环变量，例如：

```
…
{
  register int i;
  for(i=0；i<LIMIT；i++){
  …
  }
  …
}
```

例 4.13 求阶乘。

```
#include <stdio.h>
int fac(int n)
{
   register int i,f=1;
   for (i=1;i<=n;i++)
   f=f*i;
   return(f);
}
```

```
main()
{
   int i;
   for(i=1;i<=5;i++)
   printf("%d!=%d\n",i,fac(i));
}
```

第 5 章　指针

5.1 指针和地址

指针是 C 语言中一个极其重要的概念，也是 C 语言程序设计的难点。

1.指针的定义

指针变量是用于存放其他变量的地址，简称为指针。指针变量所包含的是内存地址，而该地址便是另一个变量在内存中存储的位置。指针它本身也是一种变量，和其他变量一样，要占有一定数量的存储空间，用来存放指针值(即地址)。

2.指针的作用

C 语言中引入指针类型变量，不仅简单地实现了类似于机器间址操作的指令，更重要的是使用指针可以实现程序设计中利用其他数据类型很难实现，甚至无法实现的工作。指针的作用主要体现如下：

(1)使代码更为紧凑和有效，提高了程序的效率。

(2)便于函数修改其调用参数，为函数间各类数据传递提供简捷而便利的方法。

(3)实现动态分配存储空间。

正确地使用指针会带来许多好处，但指针操作也有难以掌握的一面，若使用不当，可能会产生程序失控甚至造成系统崩溃。

5.2 指针变量和指针运算符

1．指针变量及其说明

指针变量是存放地址的变量。和其他变量一样，必须在使用之前，加以定义说明。其定义说明的一般形式：

类型说明符　*指针变量名；

例如：

int　*P；　说明指针变量 P 是指向 int 类型变量的指针.

char　*s；　说明指针变量 s 是指向 char 类型变量的指针。

float　*f；　说明指针变量 f 是指向 float 类型变量的指针。

Double *d；　说明指针变量 d 是指向 double 类型变量的指针。

static int pa；　说明指针变量 pa 是指向 int 类型变量的指针，而 pa 本身的存储类型是静态变量。

int　*fpi()；　说明 fpi()是一个函数，该函数返回指向 int 类型变量的指针。

Int　(*pfi)()；　说明 pfi 是指向一个函数的指针，该函数返回 int 类型变量。

指针变量值表达的是某个数据对象的地址，只允许取正的整数值。然而，不能因此将它与整数类型变量相混淆。所有合法指针变量应当是非 0 值。如果某指针变量取 0 值，即为 NULL，则表示该指针变量所指向的对象不存在。这是 NULL 在 C 语言中又一特殊用处。

2.指针运算符&和*

指针变量虽是正的整数值，但不是整数类型变量。指针变量最基本的运算符是&和*。

(1)&——取地址

它的作用是返回后随变量(操作数)的内存地址。请注意，它只能用于一个具体的变量或数组元素，不可用于表达式。

例：

```
int *p;
int m;
m=200;
p=&m;
```

这是将整数类型变量 m 的地址赋给整数类型的指针变量 p。

(2) *——取值

它的作用是返回其后随地址(指针变量所表达的地址值)中的变量值。

例：

```
int *p;
int m;
int n;
m=200;
p=&m;
n=*p;
```

这是将指针变量 p 所指向的整数类型变量(即 m)的值赋给整数类型变量 n。其结果与赋值语句 m=n；一样，但操作过程用了指针变量 p。

&与*运算符互为逆运算。对指针变量 px 指向的目标变量*px 实行取址运算：

```
&(*px)
```

其结果就是 px。对变量 x 的地址实行取值运算：

```
*(&x)
```

其结果就是 x。

注意，x 与*px 同级别可写成 x=*px 或*px=x。px 与&x 同级别可写成 px=&x，但决不可写成 &x=px，也不能写成 px=x。

3.指针的初始化和赋值运算

指针的初始化往往与指针的定义说明同时完成，初始化一般格式：

```
类型说明符      指针变量名＝初始地址值;
```

例如：

```
char   c;
char   *charp=&c;
```

请注意：

1)　任何一个指针在使用之前:

① 必须加以定义说明。

② 必须经过初始化。

③ 未经过初始化的指针变量禁止使用。

2) 在说明语句中初始化，也是把初始地址值赋给指针变量，只有在说明语句中，才允许这样写。而在赋值语句中，变量的地址只能赋给指针变量本身。

3) 指针变量初始化时，变量应当在前面说明过，才可使用&变量名作为指针变量的初始值。否则，编译时将出错。

4) 必须以同类型数据的地址来进行指针初始化。

指针变量可以进行赋值运算，这种赋值运算操作也仅限制在同类之间才可实现。

例 5.1：指针变量的引用。

```
void main()
{
  int    x=10
  int    *p1=&x，*p2；       /* p1 指向 x */
  p2=p1；                    /* p2 指向 x */
  printf(“%d\n”，*p2)；
 }
```

执行结果应显示：10

例 5.2：指针变量的运算。

```
void main()
{
    char c=‘A’；              /*变量 c 的初始值为’A’*/
    char   *charp=&c；        /*变量 c 的地址作为指针变量 charp 的初始值*/
    printf(“%c%c\n”，c，*charp)；
    c=‘B’；                   /*变量 c 赋值为’B’*/
    printf(“%c%c\n”，c，*charp)；
    *charp=‘a’；              /*将指针变量 charp 所指向地址的内容改为’a’*/
    printf(“%c%c\n”，c，*charp)；
 }
```

执行结果应显示：

```
AA
BB
aa
```

4.sizeof 运算符

sizeof 运算符可以准确地得到在当前所使用的系统中某一数据类型所占字节数。sizeof 运算的格式：

```
sizeof(类型说明符)
```

其值为该类型变量所占字节数。用输出语句可方便地打印出有关信息，例如：

```
printf(“%d\n”，sizeof(int))；
```

```
printf("%d\n"，sizeof(float))；
printf("%d\n"，sizeof(double))；
```

5.指针的算术运算

指针的算术运算是按地址计算规则进行。由于地址计算与相应数据类型所占字节数有关，所以，指针的算术运算应考虑到指针所指向的数据类型。

指针的算术运算只有如下四种：

＋，－，＋＋，－－

(1)指针自增 1 运算＋＋和指针自减 1 运算－－

指针自增运算意味着指针向后移动一个数据的位置，指向的地址为原来地址＋sizeof(类型说明符)。例如：

```
int i，*p；
p=&i；
p＋＋；
```

结果是 p 指向了变量 i 的地址＋sizeof(int)。

(2)指针变量的＋和－运算

指针变量 px 加(＋)正整数 n，表示指针向后移过 n 个数据类型，使该指针所指向的地址变为原指向的地址＋sizeof(类型说明符)*n。指针变量减(－)正整数 n，表示指针往前移回 n 个数据，使该指针所指向的地址变为原指向的地址－sizeof(类型说明符)*n。

6.指针的关系运算

指向同一种数据类型的指针，才有可能进行各种关系运算。指针的关系运算符包括有：

＝＝，!＝，<，<＝，>，>＝。

它们实际所进行的是两个指针所指向的地址的比较。

> 不同类型指针之间的比较是没有意义的。

> 指针与整数常量或变量的比较也没有意义。

> 唯独常量 0 是例外的，一个指针变量为 0(NULL)表示该指针指向空间不存在，称为空指针。

如果有一个指针 p，可以用 p==0 或 p!=0 来判断 p 是否为空指针。

7.使用指针编程中的常见错误

(1)使用未初始化的指针变量，例如：

```
main()
{
  int  x，*p；
  x=0；
 *p=x；          /*指针变量未初始化*/
  …
}
```

*p 未被初始化。

(2)指针变量所指向的数据类型与其定义的类型不符，例如：

```
void main()
```

```
{
    float x，y；
    short int *p；
    p=&x；    /*数据类型不符*/
    y=*p；
    …
}
```

x，y 与*p 数据类型不符。

(3)指针的错误赋值，例如：

```
void main()
{
    int x，*p；
    x=10
    p=x
    …
}
```

错在以普通变量 x 赋给指针变量 p。

(4)用局部变量的地址去初始化静态的指针变量，例如：

```
int   glo；
func()
{
   int   1oc1，1oc2；
   static int   *gp=&glo；
   static int *1p=&1oc1；
   int   *rp＝&1oc2；
    …
}
```

用局部变量 1oc1 去初始化静态的指针变量 1p 是错误的。

5.3 指针与函数参数

在函数调用中，使用指针作为参数，可以改变调用环境中的变量之值。这是使用一般变量作为参数所难以实现的。让我们来观察一个函数的调用：

例 5.3：x 和 y 的值互换。

```
#include<stdio.h>
swap1(int x，int y)
{
   int temp；
   temp=x；
   x=y；
   y=temp；
}
```

```
void main()
{
   int a，b；
   a=10；
   b=20；
   swap1(a，b)；
   printf(“a=d%db=%d\n”，a，b)；
}
```

执行结果：

a=10　b=20

虽然，swap1()函数中，x 和 y 的值互相交换了，但 x，y 是局部变量，主函数对该函数的调用 swap1(a，b)是“传值”调用，x 与 y 的交换结果，不可能影响调用者中的变量 a，b，a 与 b 之值并未交换。

改用指针作为参数，也就是用变量的地址作为参数，即所谓“传址”(或称为“传名”)的调用，就可改变相应地址中的变量。

例 5.4：x 和 y 值互换。

```
#include<stdio.h>
swap2(int *px，int *py)
{
      inttemp；
      temp=*px；
      *px=*py；
      *py=temp；
}
```

```
void main()
{
      inta，b；
      a=10；
      b=20；
      swap2(&a，&b)；    /*传地址*/
      printf(“a=%db=%d\n”，a，b)；
}
```

执行结果：

a=20　b=10

a 与 b 的地址，即&a 与&b 并未交换，但其值已被交换。

5.4 指针、数组和字符串指针

1．指针和数组

用指针和用数组名访问内存的方式几乎完全一样，却又有微妙而重要的差别。指针是地址变量，数组名则是固定的某个地址，是常量。

若定义一个数组：

int a[10]；

则在内存中占有从一个基地址开始的一块足够大的连续的空间，以包容 a[0]，a[1]…，a[9]。基地址(或称为首地址)是 a[0]存放的地址。其他依下标顺序排列。

若定义一个指针：

int *p；

则意味着：p=&a[0]；即

p 指向 a[0]

p＋1 指向 a[1]

p＋2 指向 a[2]

…

p＋i=&a[i]；

p＋i 指向 a[i] (i=0，…，9)

(1)在 C 语言中，数组名本身是指向零号元素的地址常量。因此，p=&a[0]可写成 p=a，a 当作常量指针，它指向 a[0]。p=a+i 与 p=&a[i]等价，故数组名可当指针用，它指向 0 号元素。 (2)若定义 p 为指针，a 为数组名，且 p=a，则 p+i 指向 a[i]，a+i 也指向 a[i]。

这样，a[i]与*(p+i)或*(a+i)可看成是等价的，可互相替代使用。

注意：

由于 a 只是数组名，是地址常量，而不是指针变量，所以可以写 p=a，但不能写 a=p，可以用 p++，但不能用 a++。

2.字符串指针

在 C 语言中，可用 char 型数组来处理字符串，而数组又可用相应数据类型的指针来处理。所以可以用 char 型指针来处理字符串。通常把 char 型指针称为字符串指针或字符指针。

例 5.5：计算串长度的函数 strlen()是使用字符串指针的一个实例。

```
#include<stdio.h>
strlen( char *s)
 {
  intn；
  for(n=0；*s!='\0'；s＋＋)
  n++
  return(n)；
}
```

```
void main()
 {
      static char str[]={"Hello!   "}；
      printf("%c\n"，*str)；
      printf("%d\n"，strlen(str))；
      printf("%c\n"，*(str+1))；
}
```

执行结果：H

6

e

定义字符指针可以直接用字符串作为初始值，来初始化字符指针。例如：

①char *message="Hello!"；

或

②char *message；

message="Hello!"；

这样的赋值并不是将字符串复制到指针中，只是使字符指针指向字符串的首地址。但对于数组操作，例如：

char text[80]；

则不可写成：

```
text="Hello!";
```

因为，此处 text 是数组名，而不是指针，只能按字符数组初始化操作。即：

```
static  char  text[80]={"Hello!"};
```

当字符串常量作为参数(实参)传递给函数时，实际传递的是指向该字符串的指针，并未进行字符串拷贝。例如：

例 5.6：向字符指针赋字符串

```
void main()
{
  char s="Hello!";    /*相当于 s 数组*/
  char *p;
    while(*s!='\0')
      printf("%c", *s++);
  printf("\n");
  p="Good-bye!";    /*指针指向字符串*/
    while(*p!='\o')
      printf("%c", *p++);
  printf("\n")
 }
```

执行结果：

```
Hello!
Good-bye!
```

此例中字符串是逐个字符输出，也可以字符指针为参量，作字符输出，例如：

```
printf("%s", s);
```

或

```
printf("%s", p);
```

输入函数 scanf()也可以字符指针为参量，如：

```
scanf("%s", p);
```

例 5.7：以字符指针为参数来调用串比较函数 strcomp()。

```
strcomp(char *s, char *t)
{
  for(; *s==*t; s++, t++)
    if(*s=='\o')
      return(0);
  return(*s-*t);    /* 字符串相同返回零值 */
}
void main()
{
  static char s1[]={"Hello!"};
```

```
        char *s2=“Hello!”；
        printf(“%d\n”，strcomp(s1，s2))；
        printf(“%d\n”，strcomp(“Hello”，s2))；
      }
```

s 和 t 初值已由实参传递过来了，所以 for 语句的第一个表达式为空语句。

例如：用指针处理字符串的拷贝，比用数组处理更精练

```
strcpy(char *s，*t)
{
  while(*s++=*t++)；
 }
```

如果用数组处理则要复杂些，例如：

```
strcpy(char  s[]，char t[])
{
   int i；
   i=0；
   while((s[i]=t[i])!=‘\0’)
     i++；
 }
```

直接演化出指针处理，即：

```
strcpy(char* s，char * t)
{
   while((*s=*t)!=‘\o’)
   {
     s++；
     t++；
    }
}
```

优化后便是如前所示的指针处理方法的程序。其中，也使用了空语句(；)，而且不必进行与’\0’的比较。

5.5 指针数组

1.指针数组的定义和说明

同类指针变量的集合，就形成了指针数组。或者说，以指针变量为元素的数组，就称为指针数组。这些指针变量应具有相同的存储类型，并且，指向的目标数据类型也应相同。

指针数组的一般表示格式：

类型说明符 *指针数组名［元素个数］；

例如：

```
int   *p[2]；
```

p[2]是含有 p[0]和 p[1]两个指针的指针数组，指向 int 型数据。

2.指针数组的初始化

指针数组的初始化可以在说明的同时进行。与一般数组一样，只有全局的或静态的指针数组才可进行初始化。而且，不能用局部变量的地址去初始化静态指针。

例 5.8：指针数组。

```
#include   <stdio.h>
void main()
{
 static int   b[2][3]={{1，2，3}，{4，5，6}}；
 static int   *pb[]={b[0]，b[1]}；    /*指针数组指向行首 */
 int i，j；
  for(i=0；i<2；i++){
       for(j=0；j<3；j++)
       printf(“b[%d][%d]=%d”，i，j，*(pb[i]+j))；
       printf(“\n”)
     }
 }
```

执行结果：

```
b[0][0]=1   b[0][1]=2   b[0][2]=3
b[1][0]=4   b[1][1]=5   b[1][2]=6
```

此例中，将一个二维数组 b[2][3]分解成两个一维数组。它们的首地址，分别为 b[0]和 b[1]，并被赋给指针 pb[0]和 pb[1]。

3．字符指针数组

字符指针可以用来处理一个字符串。字符指针数组可以用来处理多个字符串。这正是字符指针数组的主要作用。

例 5.9 ：

```
#include<stdio.h>
void sort(char *name[ ],int n);
void print(char *name[ ], int n);
void main()
{
   static char *name[ ]={"Follow
   me","BASIC","Great Wall",
    "FORTRAN","Computer design"};
   int n=5;
   sort(name,n);
   print(name,n);
 }
void print(char *name[ ], int n)
{
    int i;
    for (i=0;i<n;i++)
    printf("%s\n",name[i]);
 }
```

```
void sort(char *name[ ],int n)
{
   char *temp;
   int i,j,k;
    for (i=0;i<n-1;i++)/*指针数组排序*/
    {
        k=i;
        for (j=i+1;j<n;j++)
           if (strcmp(name[k],name[j])>0)
              k=j;
        if (k!=i)
        {
           temp=name[i];
           name[i]=name[k];
           name[k]=temp;}
        }
     }
}
```

运行结果为:

BASIC

Computer design

FORTRAN

Follow me

Great Wall

5.6 多级指针

指针的指针，就形成了多级指针，也就出现了多级间址访问的现象。多级指针也称为指针链。一般很少使用超过二级的指针。过多的间址难于调试跟踪，也容易出错。常用的多级指针也只是二级指针，其定义说明的一般格式如下：

类型说明符 **指针变量名

例如：

int **p；

这里 p 为二级指针，它所指向的是一个指向整数型数据的指针。又如：

static char **charp；

说明 charp 本身是静态的二级指针，指向的是个指向最终目标为字符型变量的指针。

例 5.10：二级指针

```
main()
 {
  int  x，*p，**q；
  x=10；
```

```
p=&x;
q=&p;
printf("%d\n", * *q);
}
```

用指针来处理数组是很自然的，在实际使用中，二级指针常用来处理字符指针数组。

例 5.11：

```
#include <stdio.h>
main()
{
char  **pp;
static char  *di[]={"up","down","left","right",""};   /* 字符指针
                          数组指向字符串*/
pp=di;            /*二级指针 pp 指向字符指针数组 di*/
while(**pp!=NULL)
printf("%s\n", *pp++);
}
```

执行结果：

```
up
down
left
right
```

例 5.12: 指针数组指向整型数组

```
main()
  {static a[5]={1,3,5,7,9};   /*整型数组*/
   static int *num[5]={&a[0],&a[1],&a[2],&a[3],&a[4]}; /* 指针数组刺 */
   int **p ,i;
   p=num;                  /*指向指针数组*/
   for(i=0;i<5;i++)
      {printf("%d\t",**p);p++;}
   }
```

运行结果:

```
1   3   5   7   9
```

5.7 返回指针的函数

一个函数被调用后，可以返回各种类型的数据，也可以返回指针数据，也就地址。

例 5.13: 有若干学生,每个学生 4 门课,要求输入学生序号后,能输出该学生的全部成绩。

```
main()
```

```
{static float score[][4]={{60,70,80,90,},{56,89,67,88},{34,78,90,66}};
  float   *search();   /*说明函数返回指针*/
  float   *p;
  int i,m;
  printf("enter the number of stuednt:");
  scanf ("%d",&m);
  printf("The scores of No. %d are:\n",m);
  p=search(score,m);      /*  得到该位同学的行首地址，指向列  */
  for(i=0;i<4;i++)
        printf("%5.2f\t",*(p+i));
  }
  float *search(float (*pointer)[4],int n)
  {
     float * pt;
     pt=*(pointer+n);     /*  指向列  */
     return(pt);
   }
```

运行结果：

```
enter the number of student:1↙
The score of No.1 are:
 56.00     89.00     67.00     88.00
```

5.8 函数指针

1．函数指针的定义和说明

函数也具有与数组类似的特性，可以用函数名表示函数的存储首地址，即执行该函数的入口地址。指向函数入口地址的指针，就称为函数指针。函数指针定义说明的一般格式如下：

数据类型说明　　(*函数指针名)()

例如：

```
int (*func)();
```

2．函数指针的作用

对于指向某函数的函数指针 func，实现访问目标的运算*，即(*func)()，其结果是使程序控制转移到指针所指向的地址去执行该函数。所以，函数指针所指向的是程序代码区，而不是数据区。这与一般数据指针变量有原则的区别，一般数据指针指向数据区。

第 6 章 结构体

6.1 结构的定义

1.结构的定义及其一般格式

数组是将同类型元素组成单个逻辑整体的一种数据类型。而结构则是将不同类型元素组成单个逻辑整体的一种数据类型。结构是 C 语言中一种强有力的数据类型, 是很重要的概念之一。

例如，日期由年、月、日构成：

```
int year；
int month；
int day；
```

上述表示方法不能看出它们是彼有关系的三个量。可以采用下面的定义表示：

```
struct date
{
 int month；
 int day；
 int year；
 }；
```

这样，就可把年、月、日当作一个整体处理。年、月、日是该结构的成员。结构的成员也可以是不同数据类型的变量，例如：

```
struct wage
{
 char name［30］；
 float salary；
 }；
```

上面定义了两个结构数据类型，date 和 wage。可用这些已定义的结构类型来说明一组具体结构类型变量。例如：

```
struct date today，tomorraw；    /*定义了两个结构类型变量 today，tomorraw*/
struct wage worker1，worker2，worker3；/*定义了三个结构类型变量*/
```

结构定义的一般格式为：

```
struct［结构类型名］
{
    类型说明符    成员变量名；
    …
    类型说明符    成员变量名；
 }［结构变量列表］；
```

结构变量列表是指以逗号分隔开的若干结构类型变量，例如：

```
struct date
{
  int month;
  int day;
  int year;
 }today;
```

或

```
struct
{
 int month;
 int day;
 int year;
 }today;
```

或

```
struct date          /*  定义了一种结构类型 */
{
  int month;
  int day;
  int year;
 };
struct date today;  /*  说明了一个具体的结构变量  */
```

这三种说明结构类型变量的方式都可以使用，第三种写法的风格较好。

2．结构的存取

结构变量成员可作为单独变量来操作，也就是说，可以直接访问结构中的一个成员变量，其格式为：

结构变量名.成员变量名

例 6.1：　显示输入的年、月、日

```
#include   <stdio.h>
void main()
{
    struct   date          /*  定义了结构类型 */
    {
  int month;
  int day;
  int year;
 };
struct date today;     /*  结构体变量 today */
```

```
printf(“Enter today’s date(年，月，日)\n”);
scanf(“%d，%d，%d”，&today.year，&today.month，&today.day);
printf(“Today’s date is %d/%d%/d\n”，today.year, today.month，
        today.day )
}
```

执行后可以显示出所输入的今天的年、月、日。

例 6.2：对外部存储类型的结构体变量的初始化。

```
/*L10-1*/
struct student                  /*  定义结构类型 */
{ long int num;
  char name[20];
  char sex;
  char addr[20];
 }a={89031,”Li Lin”,’M’,”123 Beijing Road”};  /*结构体变量赋初值*/
main ()
   {printf(“No.:%ld\nname:%s\nsex:%c\naddress:%s\n”,a.num,a.name,
          a.sex,a.addr);
    }
```

例 6.3： 对静态存储类型的结构体变量的初始化。

```
Main ()
{
  static struct student       /*  静态结构类型 */
  {long int num;
    char name[20];
    char sex;
    char addr[20];
    }a={89031,”Li Lin”,’M’,”123 Beijing Road”};      /*赋初值*/
  printf(“No.:%ld\nname:%s\nsex:%c\naddress:%s\n”,a.num,a.name,
         a.sex,a.addr);
}
```

6.2 结构数组

不仅结构变量成员可以是数组，而且，同一类结构变量的集合还可构成结构数组，例如：

```
struct wage
{
  char name［30］;
  float salary;
 };
```

```
static wage persons［100］;
```

这就说明了 100 个结构变量所组成的数组。

例 6.4：输入后选人名，对后选人投票计数，统计输出每个人的得票结果。

```
Struct person                /* 全局结构类型 */
{char name[20];
 int count;
 }leader[3]={“Li”,0,”Zhang”,0,”Fun”,0};    /*初始化*/
main ()
 {int I,j;
   char leader_name[20];
    for (I=1;I<=10;I++)
      {scanf(“%s”,leader_name);      /*输入人名*/
    for (j=0;j<3;j++)
      if (strcmp(leader_name,leader[j].name)==0)
          leader[j].count++;      /*累加票数*/
       }
   printf(“\n”);
   for (I=0;I<3;I++)
   printf(“%5s:%d\n”,leader[I].name,leader[I].count);/* 每人得票数*/
   }
```

6.3 结构与函数

1.向函数传递结构变量成员

结构变量成员可作为参数传递给函数。例如有一结构为：

```
struct fred
{
 char x;
 int y;
 float z;
 char s［10］;
}mike;
```

其中各个成员均可作为函数的参数，例如：

```
func0(mike.x);          /*字符型参数*/
func1(mike.y);          /*整数型参数*/
func2(mike.z);          /*浮点型参数*/
func3(mike.s);          /*字符型数组参数*/
```

```
func4(mike.s[2]);        /*字符型参数*/
```

2．向函数传递完整的结构。

不仅结构元素可作为参数传递给函数，整个结构也可作为参数传递的函数。完整结构向函数传递的操作中，应注意：

(1)整个结构按传值方式传递。和普通变量一样，与数组不同，在函数内所引起结构参数中某个值的变化，只影响函数调用时所产生的结构拷贝，不影响原来的结构。

(2)结构分全局定义和局部定义，定义在所有函数外部的结构称为全局结构，定义在函数内部的结构称为局部结构。它们被引用的范围不同。

例 6.6：全局定义的结构

```
struct st                /*全局定义*/
{
 int a，b;
 char ch;
 };
main()
{
 struct st arg;
      arg.a=1000;
      f1(arg);
}
f1(struct st    parm)          /*定义函数据库*/
{
    printf("%d\n"，parm.a);
    return;
}
```

其中，结构类型 st 是全局定义的，各函数都可引用。

6.4 结构的初始化

1.简单结构的初始化

结构说明时，行尾加上分号，随后在花括号中按结构定义的各成员顺序给以各自的初始值，并用逗号分隔之。例如：

```
struct date
 {
  int month;
  int day;
  int year;
  };          /*行尾加上分号*/
```

```
static struct date today={11，19，1991};
```

注意：局部结构变量不能初始化。

第 7 章 预处理程序

7.1 什么是预处理程序

预处理程序是一些行首以#开始的特殊语句，例如：#include，#define 等就是预处理语句。在编译程序的编译过程中，进行其它编译处理(词法分析、语法分析、代码生成、优化和连接等)之前，先进行这些语句的分析处理。

预处理语句使用的目的在于帮助程序员编写出易读、易改、易移植并便于调试的程序。预处理语句主要有四种：

宏定义和宏替换、文件包含、条件编译和行控制。

预处理语句的作用范围是从被定义语句开始直至被解除定义或是到包含它的文件结术为止均有效。

7.2 宏定义和宏替换

“宏”是借用汇编语言中的概念。为的是在 C 语言程序中方便的作一些定义和扩展。这些语句以#define 开头，分为两种：符号常量的宏定义和带参数的宏定义。

1.符号常量的宏定义和宏替换

符号常量的宏定义语句是一般格式：

```
#define  标识符  字符串
```

其中标识符就叫作宏名称。

注意：标识符与字符串之间不要用‘＝’，结尾不要加‘;’。

2.带有参数的宏定义及其替换

复杂的宏定义带有参数列表，参数列表中可有不止一个参数，其一般格式：

```
#define  标识符(参数列表)  字符串
```

对带有参数的宏定义进行宏替换时，不仅对宏标识符作字符串替换，还必须作参数的替换。

例如：

```
#define  SQ(x) ((x)*(x))
```

那么 SQ(a+b)将被宏替换成(a+b)*(a+b)。

宏定义也可嵌套使用，即一个宏定义可用另一个宏定义来定义。例如：

```
#define  SQ(x)  ((x)*(x))
#define  CUBE(x) (SQ(x)*(x))
```

3.宏定义类函数

宏定义常用于把直接插入的代码来代替函数，以提高执行效率。这一类的宏，就称做宏定义类函数，例如：

```
#define  MIN(x，y)  (((x)<(y))?(x):(y))
```

有了这样的宏之后，就可以直接引用，例如：

m=MIN(a，b)；

这语句将被预处理成：

m=(((a)<(b))?(a):(b))；

7.3 文件包含

文件包含是指一个程序文件将另一个指定义文件的内容包含进来，用#include 语句来说明。

一般有两种格式：

(1) #include <文件名>

(2) #include “文件名”

第一种，用尖括号表示在标准库目录下找该文件；第二种，用双引号表示先在当前目录(源文件所在目录)中找包含文件，若找不到，再到标准库目录中找。系统的标准库文件都是.h 文件。例如：

```
#include   <stdio.h>    /* 标准输入输出的基本常量和宏或函数文件 */
#include   <string.h>   /* 串函数文件 */
#include   <malloc.h>   /* 内存分配函数文件 */
#include   <ctype.h>    /* 字符函数文件 */
#include   <math.h>     /* 数学函数库文件 */
```

用文件包含，可以减少重复工作，提高程序正确性，还便于维护修改。程序员可以把自己常用的一些符号常量、类型定义和带参数的宏定义，以及一些常用自编函数都放在.h 文件中，通过#include 语句包含引用之。

7.4 条件编译

提供条件编译措施使同一源程序可以根据不同编译条件(参数)产生不同的目录代码，其作用在于便于调试和移植。

条件编译控制语句有不同形式，下面分别讨论。

1.#ifdef 语句及其使用

一般格式：

```
#ifdef 标识符
    语句块 1
#else
    语句块 2
#endif
```

7.4 格式化输入/输出

格式化的控制台 I/O 函数有两种，它们都与标准 I/O 库有关。源程序开头应包含标准输入输出头文件：

```
#include   <stdio.h>
```

1.printf()

printf()函数功能为按指定格式输出显示各种基本类型数据，其一般格式：

printf(“控制串”，参数列表)

控制串分两部分，即：要显示的字符和格式串。格式串以“%”开头，后跟格式码。格式串与参数一一对应。

2.scanf()

scanf()的功能是读入各种类型数据，并自动将其转换为恰当的格式，其一般格式为：scanf(“控制串”，参数列表)

控制串与前述 printf()中的控制串类似，也包含有以“%”开头加格式码组成的格式串。参数列表所列出的应当是变量的地址，而不是变量名。

第 8 章 枚举、位操作

8.1 枚举

1.枚举的定义和说明

枚举是一个有名字的某些整数型常量的集合。这些整数常量是该类型变量可取的所有合法值。枚举定义应当列出该类型变量的可取值。

一个完整的枚举定义说明语句的一般格式：

enum 枚举名{枚举列表}变量列表；

枚举的定义和说明也可写成两句，即：

enum 枚举名{枚举列表}；

enum 枚举名 变量列表；

例如：

enum day{Sun，Mon，Tue，Wed，Thu，Fri，Sat}d1，d2；

enum day d1，d2；

用上述枚举定义说明语句，表明枚举类型 day 的变量 d1 和 d2 只可能取 Sun，Mon 等七个值。

2.枚举类型的使用

使用枚举类型有两个好处，其一，使程序更清晰，既提高可读性，也减少书写的错误；其二，迫使编译程序对所定义的类型加以严格检查，防止某些错误发生。

例如：计算第二天为星期几

```
enum day{Sun，Mon，Tue，Wed，Thu，Fri，Sat}；
enum day day_after(d)
enum day d；
{
 return((enum day)(((int)d+1)%7))；
 }
```

8.2 位操作运算符

利用位操作运算符可对一个数按二进制格式进行位操作。

1.按位“与”运算&

b3=b1&b2

例如： char b1=25，b2=77，化成二进制(或十六进制)表示为：

b1=00011001(或 0x1B)

b2=01001101(或 0x4D)

b3=00001001(或 0x09) 即 b3=9

&可用作“掩码”运算，即屏蔽掉某些位。例如，掩码为127，化成二进制表示为01111111，可屏蔽掉第7位。

2.按位“或”运算 |

b3=b1|b2

b1=00011001(或 0x1B

b2=01001101(或 0x4D)

b3=01011101(或 0x5D)　即 b3=93

3.按位“异或”运算^

b3=b1^b3

b1=00011001(0x1B)

b2=01001101(0x4D)

b3=01010100(0x54)　即 b3=84

很容易验证：b3=b1^b2^b2=b1。这个特性可用于某些数据处理。例如，一个文本的每个字经过与一个关键字做异或处理，就被简单地加密了。要解密时，只需用同一个关键字再做一次异或处理，使其恢复原样。又如，用下列异或处理程序：

```
swap1(int a，int b)
{
        a=a^b;
        b=a^b;
        a=a^b;
}
```

代替下列程序：

```
swap(int a，int b)
{
   int temp;
   temp=a;
   a=b;
   b=temp;
}
```

同样实现 a 和 b 交换，异或处理的速度更快。

4.按位取反运算符～

w2=~w1，w2 为 w1 二进制按位取反。

例如：unsigned int w1=0122457，w2；

二进制表示　　(8 进制表示)

w1 1010010100101111　　(0122457)

w2 0101101011010000 (0055320)

5.左移运算符<<

将二进表示的数值各位顺序左移，最高位丢失，最低位补 0。

例如：w1=014712，左移 1 位，即 w2=w1<<1。

	二进制表示	(8 进制表示)
w1	0001100111001010	(014712)
w2	0011001110010100	(031624)

左移 n 位相当于乘 2 的 n 次方，但左移速度比乘法快。

6.右移运算符>>

将二进制表示的数值各位顺序右移，最低位丢失，对无符号整数最高位补 0。

例如：w=040273，右移 2 位，即 w2=w1>>2。

	二进制表示	(8 进制表示)
w1	0100000010111011	(040273)
w2	0001000000101110	(010056)

右移 n 位相当于除以 2 的 n 次方，但右移速度比除法快。

例如：用移位方法测试出机器的字长(即 int 含二进制位数)

```
woldlength()
{
    int i;
    usinged v=～0;                /*变量 v 为全 1*/
    for(i=1; (v=v>>1)>0; i++);
    return(i);
 }
```

第 9 章文件

9.1 ASCII 码文件的存取

1．fopen()函数

对文件的各种操作都要涉及到文件指针。进行文件操作，必须加上：

```
#include   <stdio.h>
```

并对文件指针进行说明：

```
FILE    *fp;
```

对文件操作时，应先打开文件取得文件指针。 fopen()函数的功能是打开指定的文件并返回该文件指针。其一般调用格式是：

```
FILE    *fp;
Fp=fopen(文件名，模式);
```

其中，模式有

“r”—— 读

“w”—— 写

“a ”—— 插入

如果该文件不存在，就不能为读或插入而打开文件，将返回空指针（NULL）。如果为写打开文件，已存的文件将被重写（覆盖原有的内容），要是文件不存在，将建立新文件以便写入。若磁盘空间已满，则无法写入，将返回空指针（NULL）。

例；

```
FILE   *fp;
If ((fp=fopen(“test”,”r”))==NULL{
puts(“Cannot open the file”);
Exit(0);
}
```

2. fclose()函数

一般来说，计算机系统能允许同时打开的文件数量是有限的， 所以应当关闭当前暂时不用的文件, fclose()函数的功能就是关闭指定的文件,调用格式为:

```
fclose(fp)
```

其中, fp 为指向要关闭的文件的指针。

3.getc()和 putc()

getc()和 putc()分别是对指定的文件进行单个字符的读写操作，需要以文件指针作为参数。一般调用格式为：

```
FILE   *infp ;
char ch;
ch=getc(infp);        /*从指定的文件中读取一个字符*/
```

或

```
FILE   *outfp ;
char ch;
putc(ch,outfp);    /*向指定的文件中写入一个字符*/
```

9.2 二进制文件的存取

二进制文件比 ASCII 文件少占磁盘空间，处理速度快， 但文件不能直接显示出来。

1．open()函数

与 ASCII 文件相似，对文件操作时应先打开文件取得文件指针。 只是模式不同。其一般调用格式是：

```
FILE    *fp;
Fp=fopen(文件名，模式);
```

其中，模式有

“rb”—— 读二进制代码

“wb”—— 写二进制代码

“ab ”—— 插入二进制代码

要是该文件不存在，就不能为读或插入而打开文件，返回空指针（NULL）。如果为写打开文件，已存的文件将被重写（覆盖原有的内容），若文件不存在，将建立新文件以便写入。如果磁盘空间已满，无法写入返回空指针（NULL）。

2. fclose()函数

与 ASCII 文件一样，应当关闭当前暂时不用的文件, fclose()函数的功能就是关闭指定的文件,调用格式为:

```
fclose(fp)
```

其中, fp 为指向要关闭的文件的指针。

3.fread()和 fwrite()

这两个文件操作函数实现内存缓冲区与文件之间指定数量的二进制数据的传递，因此需指定缓冲区和文件的指针，及读或写的数据量。一般调用格式：

```
fread(buffer,bn,dn,fp);
fwrite(buffer,bn,dn,fp);
```

其中：

buffer 为指向缓冲区指针（应事先定义类型）；

bn 为每一个数据项所包含的字节数；

dn 为每一次读写操作的数据项数；

fp 为打开文件的指针。