C 语言程序设计

知识整理

目　录

第 1 章 程序设计基础和 C 语言

1.1 C 语言程序的结构特点：

1.一个程序由一个或多个源程序文件组成

小程序往往只包括一个源程序文件，可以只有一个函数或者有两个函数.....

一个源程序文件中可以包括三个部分：

- 预处理指令　　#include <stdio.h>等
- 全局声明　　在函数之外进行的数据声明
- 函数定义　　每个函数用来实现一定的功能

2.函数是 C 程序的主要组成部分

一个 C 程序是由一个或多个函数组成的

必须包含一个 main 函数（只能有一个）

每个函数都用来实现一个或几个特定功能

被调用的函数可以是库函数，也可以是自己编制设计的函数

3.一个函数包括两个部分：

函数首部

int　　max　(　int　　x，　　int　　y)

若函数无参，在括弧中写 void 或空括弧。　　如：int main(void)　　或　　int main(　)

函数体

声明部分

定义在本函数中所用到的变量

对本函数所调用函数进行声明

执行部分：由若干个语句组成，指定在函数中所进行的操作

函数体---可以是空函数，如：

```
void dump ( )
{           }
```

4. 程序总是从 main 函数开始执行

5. C 程序对计算机的操作由 C 语句完成

C 程序书写格式是比较自由的;

一行内可以写几个语句;

一个语句可以分写在多行上;

为清晰起见，习惯上每行只写一个语句.

4. 程序总是从 main 函数开始执行

5. C 程序对计算机的操作由 C 语句完成

6. 数据声明和语句最后必须有分号

7. C 语言本身不提供输入输出语句，使用标准库函数（#include <stdio.h>）

8. 程序应当包含注释，增加可读性

1.2 运行 C 程序的步骤与方法

1.上机输入和编辑源程序（.c 文件）
2.对源程序进行编译（.obj 文件）
3.进行连接处理（.exe 文件）
4.运行可执行程序，得到运行结果

第 2 章 算法---程序的灵魂

著名计算机科学家沃思(Nikiklaus Wirth)提出一个公式：

算法 + 数据结构 = 程序

一个程序主要包括以下两方面的信息：

(1) 对数据的描述。在程序中要指定用到哪些数据以及这些数据的类型和数据的组织形式-----这就是数据结构(data structure)

(2) 对操作的描述。即要求计算机进行操作的步骤-----也就是算法(algorithm)

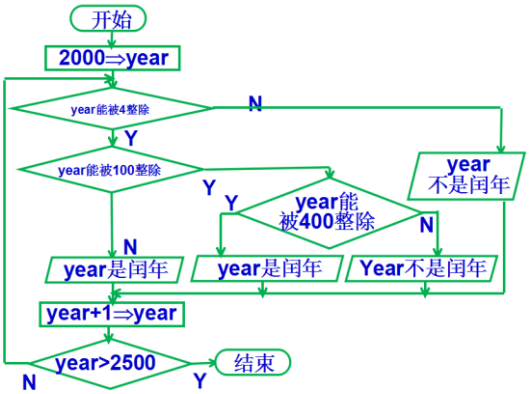
例 2.3 判定 2000—2500 年中的每一年是否闰年，并将结果输出。

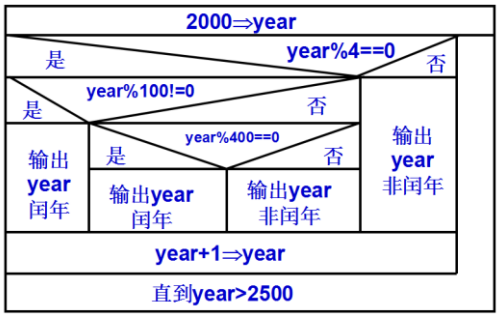
闰年的条件：

(1)能被 4 整除，但不能被 100 整除的年份都是闰年，如 2008、2012、2048 年

(2)能被 400 整除的年份是闰年，如 2000 年

不符合这两个条件的年份不是闰年,例如 2009、2100 年





例 2.4 求 $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{99}-\frac{1}{100}$

规律：

①第 1 项的分子分母都是 1

② 第 2 项的分母是 2，以后每一项的分母子都是前一项的分母加 1

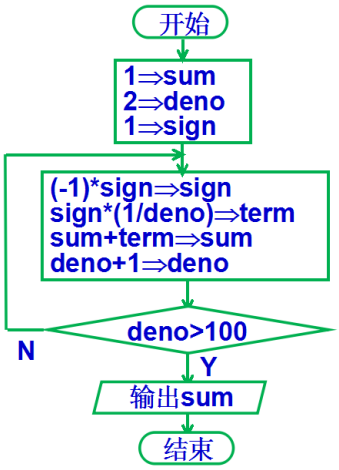
③ 第 2 项前的运算符为“-”，后一项前面的运算符都与前一项前的运算符相反

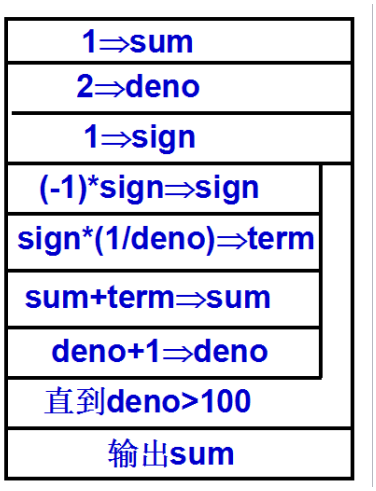
算法分析：

S1：sign=1
S2：sum=1
S3：deno=2
S4：sign=(-1)*sign
S5：term=sign*(1/deno)
S6：sum=sum+term
S7：deno=deno+1

sign—当前项符号
term—当前项的值
sum—当前各项的和
deno—当前项分母

S8：若 deno≤100 返回 S4；否则算法结束





例 2.5 给出一个大于或等于 3 的正整数，判断它是不是一个素数。

所谓素数(prime)，是指除了 1 和该数本身之外，不能被其他任何整数整除的数

例如，13 是素数，因为它不能被 2，3，4，…，12 整除。

算法分析：

判断一个数 n(n≥3)是否素数：将 n 作为被除数，将 2 到(n-1)各个整数先后作为除数，如果都不能被整除，则 n 为素数

S1：输入 n 的值

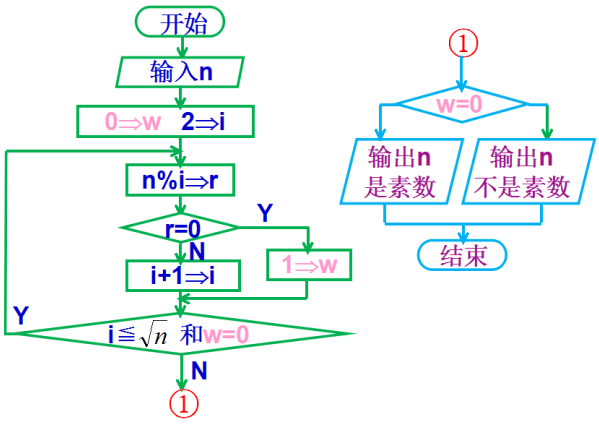
S2：i=2　　（i 作为除数）

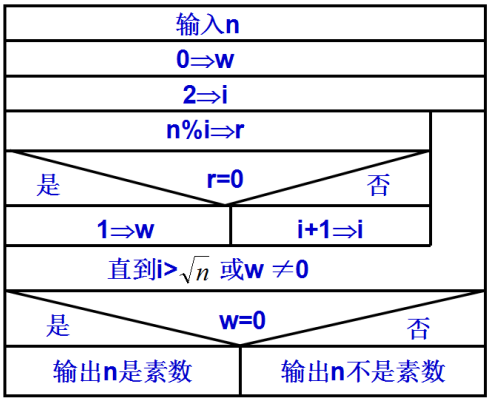
S3：n 被 i 除，得余数 r

S4：如果 r=0，表示 n 能被 i 整除，则输出 n“不是素数”，算法结束；否则执行 S5

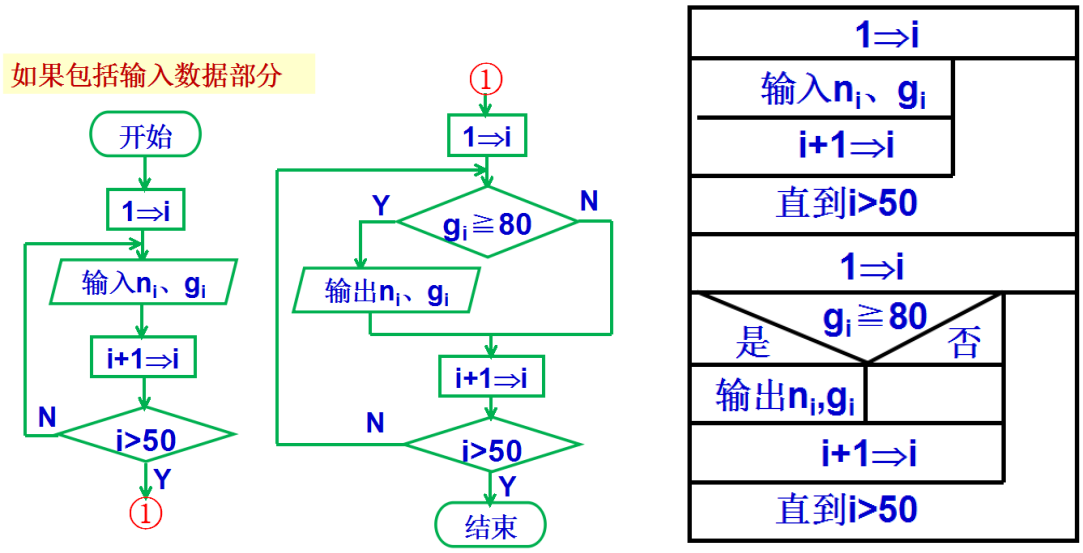
S5：i+1

S6：如果 i≤$\sqrt{n}$，返回 S3；否则输出 n “是素数”，然后结束。





例 2.7 例 2.2 的算法用流程图表示。有 50 个学生，要求将成绩在 80 分以上的学生的学号和成绩输出。



第 3 章 最简单的 C 程序设计

3.1 顺序程序设计举例

例 3.1 有人用温度计测量出用华氏法表示的温度(如 F，今要求把它转换为以摄氏法表示的温度(如 C) 。$c=\frac{5}{9}(f-32)$

```
#include <stdio.h>
int main ( )
{
    float f,c;
    f=64.0;
    c=(5.0/9)*(f-32);
    printf("f=%f\nc=%f\n",f,c);
    return 0;
 }
```

3.2 数据的表现形式及其运算

一、 常量和变量

1.常量：在程序运行过程中，其值不能被改变的量

整型常量：如 1000，12345，0，-345

实型常量

十进制小数形式：如 0.34　　-56.79　　0.0

指数形式：如 12.34e3 (代表 12.34　　103)

字符常量：如’?’

转义字符：如’\n’

字符串常量：如”boy”

符号常量：#define　PI　3.1416

2. 变量：在程序运行期间，变量的值是可以改变的

变量必须先定义，后使用

定义变量时指定该变量的名字和类型

变量名和变量值是两个不同的概念

变量名实际上是以一个名字代表的一个存储地址

从变量中取值，实际上是通过变量名找到相应的内存地址，从该存储单元中读取数据

3.常变量：const int a=3;

4.标识符：一个对象的名字（变量名或函数名）

C 语言规定标识符只能由字母、数字和下划线 3 种字符组成，且第一个字符必须为字母或下划线。　　（注意：大小写字母是不同的字符）

合法的标识符：如 sum，average, _total, Class, day, BASIC, li_ling

不合法的标识符：M.D.John，￥123，＃33，3D64，a＞b

二、数据类型

- 所谓类型，就是对数据分配存储单元的安排，包括存储单元的长度(占多少字节)以及数据的存储形式。
- 不同的类型分配不同的长度和存储形式。

三、　整型数据

1. 整型数据的分类

基本整型(int 型)：占 2 个或 4 个字节

短整型(short int)：VC++6.0 中占 2 个字节

长整型(long int)：VC++6.0 中占 4 个字节

有符号基本整型　[signed] int;

无符号基本整型　unsigned int;

有符号短整型　[signed] short [int];

无符号短整型　unsigned short [int];

有符号长整型　[signed] long [int];

无符号长整型　unsigned long [int]

四、字符型数据

字符是按其代码(整数)形式存储的；C99 把字符型数据作为整数类型的一种；

1.字符与字符代码

大多数系统采用 ASCII 字符集

字母：A ~Z，a ~z

数字：0～9

专门符号：29 个：！ ” # & ‘ ( ) *等

空格符：空格、水平制表符、换行等

不能显示的字符：空(null)字符(以‘\0’表示)、警告(以‘\a’表示)、退格(以‘\b’表示)、回车(以‘\r’表示)等

字符’1’和整数 1 是不同的概念：

字符’1’只是代表一个形状为’1’的符号，在需要时按原样输出，在内存中以 ASCII 码形式存储，占 1 个字节

整数 1 是以整数存储方式(二进制补码方式)存储的，占 2 个或 4 个字节

2.字符变量：用类型符 char 定义字符变量

```
char c = ’?’;                    // 系统把“?”的 ASCII 代码 63 赋给变量 c
printf(”%d  %c\n”,c,c);      //输出结果是： 63  ?
```

五、浮点型数据

浮点型数据是用来表示具有小数点的实数。

float 型(单精度浮点型)

double 型(双精度浮点型)

编译系统为 float 型变量分配 4 个字节，数值以规范化的二进制数指数形式存放

编译系统为 double 型变量分配 8 个字节，15 位有效数字

long double(长双精度)型

六、怎样确定常量的类型

字符常量：由单撇号括起来的单个字符或转义字符

整型常量：不带小数点的数值。系统根据数值的大小确定 int 型还是 long 型等

浮点型常量：凡以小数形式或指数形式出现的实数。C 编译系统把浮点型常量都按双精度处理，分配 8 个字节

七、运算符和表达式

1.基本的算术运算符：

+ ：正号运算符(单目运算符)

- ：负号运算符(单目运算符)

* ：乘法运算符

/ ：除法运算符

% ：求余运算符

+ ：加法运算符

- ：减法运算符

2. 自增、自减运算符：作用是使变量的值 1 或减 1

++i，--i：在使用 i 之前，先使 i 的值加（减）1

i++，i--：在使用 i 之后，使 i 的值加（减）1

3. 算术表达式和运算符的优先级与结合性：

用算术运算符和括号将运算对象（也称操作数）连接起来的、符合 C 语法规则的式子，称为 C 算术表达式

运算对象包括常量、变量、函数等

C 语言规定了运算符的优先级和结合性

4.不同类型数据间的混合运算：（系统自动进行的类型转换）

(1)+、-、*、/ 运算的两个数中有一个数为 float 或 double 型，结果是 double 型。系统将 float 型数据都先转换为 double 型，然后进行运算

(2) 如果 int 型与 float 或 double 型数据进行运算，先把 int 型和 float 型数据转换为 double 型，然后进行运算，结果是 double 型

(3)字符型数据与整型数据进行运算，就是把字符的 ASCII 代码与整型数据进行运算

例 3.3 给定一个大写字母，要求用小写字母输出。

解题思路：关键是找到大、小写字母间的内在联系，同一个字母，用小写表示的字符的 ASCII 代码比用大写表示的字符的 ASCII 代码大 32。

```
#include <stdio.h>
int main ( )
{
    char c1,c2;
    c1=’A’;
    c2=c1+32;
    printf("%c\n",c2);
    printf(” %d\n”,c2);
    return 0;
}
```

5. 强制类型转换运算符

强制类型转换运算符的一般形式为

（类型名）（表达式）

(double)a （将 a 转换成 double 类型）

(int) (x+y) （将 x+y 的值转换成 int 型）

(float)(5%3)（将 5%3 的值转换成 float 型）

有两种类型转换：**系统自动进行的类型转换、强制类型转换**

6. C 运算符

(1) 算术运算符 （+ - * / % ++ --）

(2) 关系运算符 （＞ ＜ ＝＝ ＞＝ ＜＝ ！＝）

(3) 逻辑运算符 （！ ＆＆ ||）

(4) 位运算符 （<< >> ~ | ∧ ＆）

(5) 赋值运算符 （＝及其扩展赋值运算符）

(6) 条件运算符 （？：）

(7) 逗号运算符 （，）

(8) 指针运算符 （*和＆）

(9) 求字节数运算符 （sizeof）

(10) 强制类型转换运算符 （ (类型) ）

(11) 成员运算符 （.->）

(12) 下标运算符 （[ ]）

(13) 其他 （如函数调用运算符（））

3.3 C 语句

C 语句的作用和分类

在 C 程序中，最常用的语句是：赋值语句、输入输出语句。

归纳总结：

1.赋值运算符

“=”是赋值运算符，作用是将一个数据赋给一个变量，也可以将一个表达式的值赋给一个变量。

2.复合的赋值运算符

在赋值符“＝”之前加上其他运算符，可以构成复合的运算符。如 a += 3 ，等价于 a = a +3.

3.赋值表达式

一般形式为：　　变量　赋值运算符　表达式

对赋值表达式求解的过程：求赋值运算符右侧的“表达式”的值,赋给赋值运算符左侧的变量。

赋值表达式“a=3*5”的值为 15，对表达式求解后，变量 a 的值和表达式的值都是 15。

“a=(b=5)”和“a=b=5”等价

“a=b”和“b=a”含义不同

4.赋值过程中的类型转换

5.赋值表达式和赋值语句

6.变量赋初值

int a=3,b=3,c;

int a=3;　相当于　int a;　　a=3;

3.4 数据的输入输出

一、有关数据输入输出的概念

(1) 所谓输入输出是以计算机主机为主体而言的

从计算机向输出设备(如显示器、打印机等)输出数据称为输出；

从输入设备（如键盘、磁盘、光盘、扫描仪等）向计算机输入数据称为输入。

(2) C 语言本身不提供输入输出语句

输入和输出操作是由 C 标准函数库中的函数来实现的。

printf 和 scanf 不是 C 语言的关键字，而只是库函数的名字

putchar、getchar、puts、gets

(3)在使用输入输出函数时，要在程序文件的开头用预编译指令

#include <stdio.h> 或　　#include 〞stdio.h〞

二、 用 printf 函数输出数据

在 C 程序中用来实现输出和输入的，主要是 printf 函数和 scanf 函数

这两个函数是格式输入输出函数，用这两个函数时，必须指定格式

1.printf 函数的一般格式

printf（格式控制，输出表列）

例如：printf(”i=%d,c=%c\n”,i,c);

2. 常用格式字符

d 格式符。用来输出一个有符号的十进制整数

可以在格式声明中指定输出数据的域宽

printf(”%5d%5d\n”,12,-345);

%d 输出 int 型数据；

%ld 输出 long 型数据；

c 格式符。用来输出一个字符

char ch=’a’;　　printf(”%c”,ch);　或　printf(”%5c”,ch);

s 格式符。用来输出一个字符串

printf（”%s”,”CHINA”）;

f 格式符。用来输出实数，以小数形式输出

①不指定数据宽度和小数位数，用%f

例 3.6 用%f 输出实数，只能得到 6 位小数。

```
double a=1.0;
printf(”%f\n”,a/3);
```

② 指定数据宽度和小数位数。用%m.nf

```
printf("%20.15f\n",1/3);      printf("%.0f\n”,10000/3.0);
float a;   a=10000/3.0;   printf("%f\n",a);
```

③ 输出的数据向左对齐，用%-m.nf

- float 型数据只能保证 6 位有效数字
- double 型数据能保证 15 位有效数字
- 计算机输出的数字不都是绝对精确有效的

0.333333

e 格式符。指定以指数形式输出实数

3333.333333

%e，VC++给出小数位数为 6 位

指数部分占 5 列，小数点前必须有而且只有 1 位非零数字

```
printf(”%e”,123.456);          //输出：1.234560 e+002
```

%m.ne

```
printf(”%13.2e”,123.456);       //输出：1.23e+002      (前面有 4 个空格)
```

三、用 scanf 函数输入数据

1. scanf 函数的一般形式

scanf（格式控制，地址表列）

2. scanf 函数中的格式声明

与 printf 函数中的格式声明相似，以%开始，以一个格式字符结束，中间可以插入附加的字符。如：scanf("a=%f,b=%f,c=%f",&a,&b,&c);

3.使用 scanf 函数时应注意的问题

```
scanf(”%f%f%f”,a,b,c);            错
scanf(”%f%f%f”,&a,&b,&c);     对
```

对于

```
scanf("a=%f,b=%f,c=%f",&a,&b,&c);
        1 3 2↙                    错
        a=1,b=3,c=2↙              对
        a=1 b=3 c=2↙
```

对于 scanf(”%c%c%c”,&c1,&c2,&c3);

```
        abc↙            对
        a b c↙          错
```

四、 字符数据的输入输出

1.用 putchar 函数输出一个字符。从计算机向显示器输出一个字符

putchar 函数的一般形式为：　　putchar(c)

例 3.8 接收三个字符，然后输出三个字符。

```
#include <stdio.h>
int   main ( )
{ char a,b,c;
   a=getchar();
   b=getchar();
```

```
    c=getchar();
    putchar(a); putchar(b); putchar(c);
    putchar('\n');
    return 0;
}
```

第 4 章 选择结构程序设计

4.1 选择结构和条件判断

C 语言有两种选择语句：

(1)if 语句，实现两个分支的选择结构

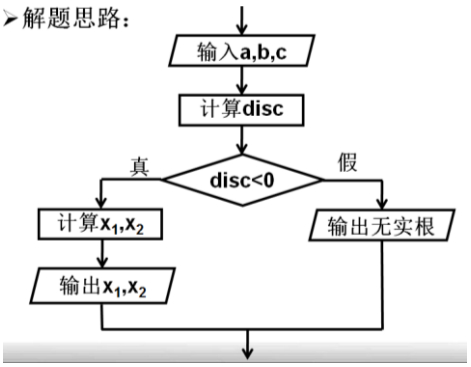
(2)switch 语句，实现多分支的选择结构

例 4.1 在例 3.5 的基础上对程序进行改进。题目要求是求方程的根。

由键盘输入 a,b,c。假设 a,b,c 的值任意，并不保证 b^2-4ac>=0。需要在程序中进行判别，如果 b^2-4ac>=0 ，就计算并输出方程的两个实根，否则就输出“方程无实根”的信息。

```
#include <stdio.h>
#include   <math.h>
int main ( )
 {
     double a,b,c,disc,x1,x2,p,q;
     scanf("%lf%lf%lf",&a,&b,&c);
     disc=b*b-4*a*c;
   if (disc<0)
       printf(“has not real roots\n”);
  else
  {   p=-b/(2.0*a);
       q=sqrt(disc)/(2.0*a);
       x1=p+q;
       x2=p-q;
       printf(“real roots:\n x1=%7.2f\t ,x2=%7.2f\n”,x1,x2);
   }
   return 0;
}
```

➢解题思路：



4.2 用 if 语句实现选择结构

4.2.1 用 if 语句处理选择结构举例

例 4.2 输入两个实数，按代数值由小到大的顺序输出这两个数。

```
#include <stdio.h>
int main()
{ float a,b,t;
    scanf("%f,%f",&a,&b);
    if(a>b)
      {   t=a;
          a=b;
```

```
            b=t;
        }
    printf("%5.2f,%5.2f\n",a,b);
    return 0;
}
```

例 4.3 输入 3 个数 a，b，c，要求按由小到大的顺序输出。

➢ 解题思路：可以先用伪代码写出算法：

◆ if a>b，a 和 b 对换 （a 是 a、b 中的小者）

◆ if a>c，a 和 c 对换 （a 是三者中最小者）

◆ if b>c，b 和 c 对换 （b 是三者中次小者）

◆ 顺序输出 a，b，c

```
#include <stdio.h>
int main()
{ float a,b,c,t;
    scanf("%f,%f,%f",&a,&b,&c);
    if(a>b)
    {    t=a;   a=b;   b=t;    }
    if(a>c)
    {    t=a;   a=c;   c=t;    }
    if(b>c)
    {    t=b;   b=c;   c=t;    }
    printf("%5.2f,%5.2f,%5.2f\n",a,b,c);
    return 0;
}
```

4.2.2 if 语句的一般形式

最常用的 3 种 if 语句形式：

1. if (表达式) 语句 1 **(没有 else 子句)**
2. if (表达式) 语句 1
   else 语句 2 **(有 else 子句)**
3. if（表达式 1 ） 语句 1
   else if（表达式 2 ） 语句 2
   else if（表达式 3 ） 语句 3
   ┆
   else if（表达式m ） 语句m
   else 语句 m+1

(在 else 部分又嵌套了多层的 if 语句)

例如：

```
if(number > 500)    cost = 0.15;
else if (number > 300)    cost = 0.10;
else if (number > 100)    cost = 0.075;
else if (number > 50)     cost = 0.05;
else       cost=0;
```

等价于：

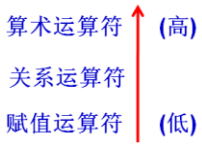
```
if (number > 500)    cost = 0.15;
```

```
else
    if (number > 300)      cost = 0.10;
    else
        if (number > 100)    cost = 0.075;
        else
            if (number > 50)    cost = 0.05;
            else      cost = 0;
```

4.3 关系运算符和关系表达式

4.3.1 关系运算符及其优先次序

➢关系、算术、赋值运算符的优先级



4.3.2 关系表达式

用关系运算符将两个数值或数值表达式连接起来的式子。关系表达式的值是一个逻辑值，即“真”或“假”，在 C 的逻辑运算中，以“ 1 ”代表“真”，以“ 0 ”代表“假”。

4.4 逻辑运算符和逻辑表达式

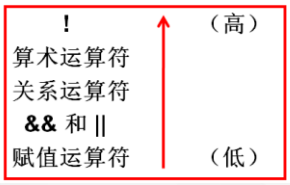
4.4.1 逻辑运算符及其优先次序

- 3 种逻辑运算符：&&（逻辑与） ||（逻辑或） !（逻辑非）
- &&和||是双目(元)运算符
- !是一目(元)运算符
- 逻辑表达式：用逻辑运算符将关系表达式或其他逻辑量连接起来的式子

➢逻辑运算符的优先次序

! → && → ||　　（!为三者中最高）

➢与其他运算符的优先次序



4.4.2 逻辑表达式

逻辑表达式的值应该是逻辑量“真”或“假”。

编译系统在表示逻辑运算结果时，以数值 1 代表“真”，以 0 代表“假”；但在判断一个量是否为“真”时，以 0 代表“假”，以非 0 代表“真”。

注意：将一个非零的数值认作为“真”

4.5 条件运算符和条件表达式

有一种 if 语句，当被判别的表达式的值为“真”或“假” 时，都执行一个赋值语句且向同一个变量赋值。

如：if (a>b)

```
等价于：
max = (a > b) ? a : b;
```

```
        max=a;
    else
        max=b;
```

条件表达式的一般形式为：　　表达式１？表达式２：表达式３

- 条件运算符的执行顺序：
  - 求解表达式 1
  - 若为非 0（真）则求解表达式 2，此时表达式 2 的值就作为整个条件表达式的值
  - 若表达式 1 的值为 0（假），则求解表达式 3，表达式 3 的值就是整个条件表达式的值
- 条件运算符优先于赋值运算符
- 条件运算符的结合方向为“自右至左”
- 以下为合法的使用方法：

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

  a>b ? printf(“%d”,a): printf(“%d”,b);

例 4.4 输入一个字符，判别它是否大写字母，如果是，将它转换成小写字母；如果不是，不转换。然后输出最后得到的字符。

```
#include <stdio.h>
int main()
{
   char ch;
   scanf("%c",&ch);
   ch=(ch>='A' && ch<='Z')?(ch+32):ch;
   printf("%c\n",ch);
   return 0;
}
```

4.6 选择结构的嵌套

- 在**if**语句中又包含一个或多个**if**语句称为**if**语句的嵌套
- 一般形式：

```
if( )
    if( ) 语句1
    else  语句2
else
    if( ) 语句3
    else  语句4
```

else总是与它上面最近的未配对的**if**配对

内嵌**if**

- 在**if**语句中又包含一个或多个**if**语句称为**if**语句的嵌套

```
if ( )
{
    if ( ) 语句1
}
else语句2
```

内嵌**if**

{ }限定了内嵌**if**范围

4.7 用 switch 语句实现多分支选择结构

switch 语句的作用是根据表达式的值，使流程跳转到不同的语句。switch 语句的一般形式：

```
switch（表达式）
{   case 常量 1 ： 语句 1
    case 常量 2 ： 语句 2
    ┆      ┆          ┆
    case 常量 n ： 语句 n
}
```

整数类型(包括字符型)

值不能相同

```
        default       :  语句 n+1
    }
```

例 4.6 要求按照考试成绩的等级输出百分制分数段，A 等为 85 分以上，B 等为 70～84 分，C 等为 60～69 分 ，D 等为 60 分以下 。成绩的等级由键盘输入。

```
#include <stdio.h>
int main()
{   char grade;
    scanf("%c",&grade);
    printf("Your score:");
    switch(grade)
   { case 'A': printf("85～100\n");break;
      case 'B': printf("70～84\n");break;
      case 'C': printf("60～69\n");break;
      case 'D': printf("<60\n");break;
      default:   printf("enter data error!\n");
   }
    return 0;
}
```

4.8 选择结构程序综合举例

例 4.8 写一程序，判断某一年是否闰年。

解题思路：

- 在前面已介绍过判别闰年的方法。本例用不同的方法编写程序。
- 用变量 leap 代表是否闰年的信息。若闰年，令 leap=1；非闰年，leap=0。最后判断 leap 是否为 1 （真），若是，则输出“闰年”信息。

```
#include <stdio.h>
int main()
{ int year,leap;
  printf("enter year:"); scanf("%d",&year);
  if (year%4==0)
      if(year%100==0)
          if(year%400==0)      leap=1;
          else       leap=0;
      else       leap=1;
  else       leap=0;
  if (leap)         printf("%d is ",year);
  else    printf("%d is not ",year);
  printf("a leap year.\n");
  return 0;
}
```

等价于：

```
if((year%4==0 && year%100!=0)
                    || (year%400==0))
      leap=1;
else
      leap=0;
```

第 5 章 循环结构程序设计

5.1 为什么需要循环控制

- 大多数的应用程序都会包含循环结构
- 循环结构和顺序结构、选择结构是结构化程序设计的三种基本结构，它们是各种复杂程序的基本构造单元

5.2 用 while 语句实现循环

while 语句的一般形式如下：

```
while (表达式)
  {语句或复合语句}
```

循环条件表达式
- “真”时执行循环体语句
- “假”时不执行

注意：非复合语句可以不加{ }

while 循环的特点是：先判断条件表达式，后执行循环体语句

例 5.1 求 1+2+3+…+100

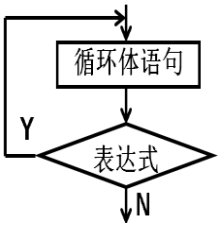
```
#include <stdio.h>
int main()
{
    int i=1,sum=0;
    while (i<=100)
    {   sum=sum+i;
        i++;
    }
    printf("sum=%d\n",sum);
    return 0;
}
```

5.3 用 do---while 语句实现循环

do---while 语句的特点：先无条件地执行循环体，然后判断循环条件是否成立

do---while 语句的一般形式为：

```
do
    {语句或复合语句}
while (表达式);
```



例 5.2 求 1+2+3+…+100

```
#include <stdio.h>
int main()
{   int i=1,sum=0;
    do
     {
        sum=sum+i;
        i++;
     }while(i<=100);
     printf("sum=%d\n",sum);
     return 0;
}
```

5.4 用 for 语句实现循环

- for 语句不仅可以用于循环次数已经确定的情况，还可以用于循环次数不确定而只给出循环结束条件的情况
- for 语句完全可以代替 while 语句
- **for 语句的一般形式为**

for(表达式 1；表达式 2；表达式 3)
{语句或复合语句}

作为循环的调整器，例如使循环变量增值，它是在执行完循环体后才进行的

循环条件表达式，用来判定是否继续循环。在每次执行循环体前先执行此表达式，决定是否继续执行循环

设置初始条件，只执行一次。可以为零个、一个或多个变量设置初值执行

例如：for(i=0,j=100 ; i<=j; i++,j--)
k=i+j;

例如：for(sum=0,i=1 ; i<=100; i++)
sum=sum+i;

例如：for(; (c=getchar())!=’\n’;)
printf(″%c″, c);

例如：for(i=0; (c=getchar())!=’\n′; i+=c)
;

5.5 循环的嵌套

- 一个循环体内又包含另一个完整的循环结构，称为循环的嵌套
- 内嵌的循环中还可以嵌套循环，这就是多层循环
- 3 种循环(while 循环、do…while 循环和 for 循环)可以互相嵌套

例 5.2 输出以下九九乘法表。

```
#include <stdio.h>
main()
{
    int i,j;
    for(i=1;i<=9;i++)
    {
        for(j=1;j<=i;j++)
        {
        printf("%d*%d=%2d\t",j,i,i*j);
        }
    printf("\n");
    }
}
```

```
1*1= 1
1*2= 2  2*2= 4
1*3= 3  2*3= 6  3*3= 9
1*4= 4  2*4= 8  3*4=12  4*4=16
1*5= 5  2*5=10  3*5=15  4*5=20  5*5=25
1*6= 6  2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7= 7  2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49
1*8= 8  2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9= 9  2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81
```

例 5.3 输出以下 4*5 的矩阵。

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20

```
#include <stdio.h>
int main()
{   int i,j,n=0;
    for (i=1;i<=4;i++)                          //控制输出 4 行
        for (j=1;j<=5;j++,n++)                  //变量 n 用来累计输出数据的个数
                                                 //控制每行中输出 5 个数据
        {   if (n%5==0) printf (“\n”);          //控制一行内输出 5 个数据
            printf ("%d\t",i*j);
        }
     printf("\n");
     return 0;
}
```

5.6 几种循环的比较

(1)一般情况下，3 种循环可以互相代替

(2) 在 while 和 do---while 循环中，循环体应包含使循环趋于结束的语句。

(3) 用 while 和 do---while 循环时，循环变量初始化的操作应在 while 和 do---while 语句之前完成。而 for 语句可以在表达式 1 中实现循环变量的初始化。

5.7 改变循环执行的状态

5.7.1 用 break 语句提前终止循环

break 语句可以用来从“循环体”内或 switch 结构跳出，即提前结束循环或下面 case 后在语句执行，接着执行“循环”或“switch 结构”下面的语句

5.7.2 用 continue 语句提前结束本次循环

有时并不希望终止整个循环的操作，而只希望提前结束本次循环，而接着执行下次循环。这时可以用 continue 语句

例 5.5 要求输出 100～200 之间的不能被 3 整除的数。

编程思路：

对 100 到 200 之间的每一个整数进行检查

如果不能被 3 整除，输出，否则不输出

无论是否输出此数，都要接着检查下一个数(直到 200 为止)。

关键代码：

```
for(n=100;n<=200;n++)
  {   if (n%3==0)
            continue;
      printf("%d    ",n);
  }
```

5.7.3 break 语句和 continue 语句的区别

- continue 语句只结束本次循环，而不是终止整个循环的执行

- break 语句结束整个循环过程，不再判断执行循环的条件是否成立

5.8 循环程序举例

例 5.7 用 $\frac{\pi}{4} \approx 1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$ 公式求 π 的近似值，直到发现某一项的绝对值小于 10-6 为止。(该项不累计加)。

```
#include <stdio.h>
#include <math.h>
int main()
{   int sign=1; double pi=0,n=1,term=1;
    while(fabs(term)>=1e-6)
    {   pi=pi+term;
        n=n+2;
        sign=-sign;
        term=sign/n;
     }
    pi=pi*4;
    printf("pi=%10.8f\n",pi);
    return 0;
}
```

例 5.8　著名的 Fibonacci(费波那契数列问题)

方法一：每次输出一个数

```
#include <stdio.h>
int main()
{
    int f1=1,f2=1,f3,i;
    printf("%12d\n%12d\n",f1,f2);
    for(i=1;i<=38;i++)     //注意，因为有 f1,f2，要求有 40 个，要么写<=38,要么写<39。
    {
        f3=f1+f2;
        printf("%12d\n",f3);
        f1=f2;     //为下次的 f3 计算做准备，依次修改 f1、f2 的值.
        f2=f3;     //为下次的 f3 计算做准备，依次修改 f1、f2 的值.
    }
    return 0;
}
```

方法二：每次输出 2 个数

```
#include <stdio.h>
int main()
 {   int f1=1,f2=1;    int i;
     for(i=1; i<=20; i++)
    { printf("%12d %12d ",f1,f2);
        if(i%2==0) printf("\n");
```

```
            f1=f1+f2;            //为下次的输出做准备，依次修改 f1、f2 的值.
            f2=f2+f1;            //为下次的输出做准备，依次修改 f1、f2 的值.

        }
        return 0;
    }
```

例 5.9　判断输入的数是否是素数。

```
#include <stdio.h>
#include <math.h>
int main()
 {   int n,i,k;
     printf(“n=?");
     scanf("%d",&n);
     k=sqrt(n);
     for (i=2; i<=k; i++)
          if(n%i==0) break;
     if(i<=k) printf("%d is not\n",n);
     else printf("%d is\n",n);
     return 0;
}
```

例 5.10　求 100 至 200 间的素数.—附关键部分代码：

```
for(n=101;n<=200;n=n+2)
{ k=sqrt(n);
    for (i=2;i<=k;i++)
         if (n%i==0) break;
    if (i>=k+1)
     {   printf("%d ",n);
         m=m+1;
     }
     if(m%10==0) printf(“\n”);
}
```

第 6 章 利用数组处理批量数据

6.1 怎样定义和引用一维数组

6.1.1 怎样定义一维数组

定义一维数组的一般形式为：

类型符　数组名[常量表达式];

数组名的命名规则和变量名相同。如：　int a[10];

定义一维数组时赋初值

对全部或部分数组元素赋初值

例如：　int x[8]={1，2，3，4，5，6，7，8};

　　int x[8]={1，2，3，4，5};

对全部数组元素赋初值时，可以不指定数组的长度，系统将根据初值数据个数确定数组长

度。例如：int x[ ]={1，2，3，4，5};

对全部数组元素初始化为 0 时，可以写成：

int x[5]={0，0，0，0，0};

或更简单地 ：int x[5]={0};

6.1.2 怎样引用一维数组元素

引用数组元素的表示形式为：

数组名［下标］

如 a[0]=a[5]+a[7]-a[2*3]　　合法

6.1.3 一维数组的初始化

在定义数组的同时，给各数组元素赋值

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

int a[10]={0,1,2,3,4};相当于

int a[10]={0,1,2,3,4,0,0,0,0,0};

int a[10]={0,0,0,0,0,0,0,0,0,0};相当于

int a[10]={0};

int a[5]={1,2,3,4,5};可写为

int a[ ]={1,2,3,4,5};

例 6.2 用数组处理求 Fibonacci 数列问题

```
#include <stdio.h>
int main()
 {   int i;   int f[20]={1,1};
       for(i=2;i<20;i++)
            f[i]=f[i-2]+f[i-1];
       for(i=0;i<20;i++)
     {     if(i%5==0) printf(“\n”);
            printf(“%12d”,f[i]);
     }
     printf("\n");
     return 0;
}
```

例 6.3 有 10 个数，要求对它们按由小到大的顺序排列。（附关键代码）

```
int a[10];
int i,j,t;
printf("input 10 numbers :\n");
for (i=0;i<10;i++)    scanf("%d",&a[i]);
printf("\n");
for(j=0;j<9;j++)
      for(i=0;i<9-j;i++)
         if (a[i]>a[i+1])
            {t=a[i];a[i]=a[i+1];a[i+1]=t;}
printf("the sorted numbers :\n");
for(i=0;i<10;i++)    printf("%d ",a[i]);
printf("\n");
```

6.2 怎样定义和引用二维数组

6.2.1 怎样定义二维数组

- 二维数组定义的一般形式为

  类型符 数组名[常量表达式][常量表达式];

  如：float a[3][4],b[5][10];

- 二维数组可被看作是一种特殊的一维数组：

  它的元素又是一个一维数组

- 例如，把 a 看作是一个一维数组，它有 3 个元素：

  a[0]、a[1]、a[2]

- 每个元素又是一个包含 4 个元素的一维数组

1、对二维数组的全部元素赋初值。

例如：int x[2][4]= {{1，2，3，4}，{6，7，8，9}};

在初始化格式的一对花括号内，初值表中每行数据另用一对花括号括住，此方式一目了然，通过赋值，在二维数组 x 中，各元素的初始化值为：

x[0][0]=1，x[0][1]=2，x[0][2]=3，x[0][3]=4，

x[1][0]=6，x[1][1]=7，x[1][2]=8，x[1][3]=9

又如：int u[2][4]= {1，2，3，4，5，6，7，8};

u [0][0]=1，u [0][1]=2，u [0][2]=3，u [0][3]=4，

u [1][0]=5，u [1][1]=6，u [1][2]=7，u [1][3]=8

2、对二维数组的部分元素赋初值

例如：int x[3][5]= {{1}，{6，7}，{ }};

int u[3][5]= {1，6，7};

同样为 3 行 5 列有 15 个数组元素的二维数组，在数组 x 中元素的赋初值结果：x[0][0]=1，x[1][0]=6，x[1][1]=7，其余元素均为 0；而在数组 u 中的结果为：u [0][0]=1，u [0][1]=6，u [0][2]=7，其余元素均赋初值 0；作为行标志的花括号在此所起的作用是明显的。

给二维数组的全部元素赋初值，也可以不指定第一维的长度，**但第二维的长度不能省略**。

例如：int x[ ][5]= {{1，2，3，4，5}，{6，7，8，9，10}，{11，12，13，14，15}};

int x[ ][5]= {1，2，3，4，5，6，7，8，9，10， 11，12，13，14，15};

他们都表示定义了一个 3 行 5 列的二维数组 x，且每一数组元素的取值在上例两种方式中是得到同样的结果的。

6.2.2 怎样引用二维数组的元素

二维数组元素的表示形式为：

数组名［下标］［下标］

int a[3][4]={{1,2,3,4},{5,6,7,8}, {9,10,11,12}};

int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};

int a[3][4]={{1},{5},{9}};等价于 int a[3][4]={{1,0,0,0},{5,0,0,0}, {9,0,0,0}};

int a[3][4]={{1},{5,6}};相当于 int a[3][4]={{1},{5,6},{0}};

6.2.3 二维数组的初始化

int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};

等价于：int a[ ][4]={1,2,3,4,5,6,7,8,9,10,11,12};

int a[][4]={{0,0,3},{ },{0,10}};合法

例 6.4 将一个二维数组行和列的元素互换，存到另一个二维数组中。

$$a=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} \Longrightarrow b=\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}$$

```
#include <stdio.h>
int main()
 {  int a[2][3]={{1,2,3},{4,5,6}};
    int b[3][2],i,j;
    printf("array a:\n");
    for (i=0;i<=1;i++)            //处理a的一行中各元素
    { for (j=0;j<=2;j++)            //处理a中某一列元素
      {  printf("%5d",a[i][j]);              //输出a的各元素
         b[j][i]=a[i][j];           //a元素值赋给b相应元素
       }
      printf("\n");
     }
   printf("array b:\n");
   for (i=0;i<=2;i++)
   {  for(j=0;j<=1;j++)
         printf("%5d",b[i][j]);    //输出b的各元素
      printf("\n");
   }
   return 0;
}
```

例 6.5 有一个 3×4 的矩阵，要求编程序求出其中值最大的那个元素的值，以及其所在的行号和列号。

```
……
   int i,j,row=0,colum=0,max;
   int a[3][4]={{1,2,3,4},{9,8,7,6}, {-10,10,-5,2}};
   max=a[0][0];
   for (i=0;i<=2;i++)
      for (j=0;j<=3;j++)
         if (a[i][j]>max)
         {   max=a[i][j];   row=i;   colum=j; }
   printf("max=%d\nrow=%d\n colum=%d\n",max,row,colum);
……
```

6.3 字符数组

6.3.1 怎样定义字符数组

(1)一维字符数组的定义

char　数组名[数组长度];

有如下程序段：

```
char ch[12];
ch[0]= 'H'; ch[1]= 'o'; ch[2]= 'w'; ch[3]= '  '; ch[4]= 'a'; ch[5]= 'r';
```

ch[6]= 'e';ch[7]= '　';ch[8]= 'y'; ch[9]= 'o'; ch[10]= 'u'; ch[11]= '\0';

(2)二维字符数组的定义与引用

char　数组名[长度 1][长度 2]；

字符数组元素也可通过数组名和下标引用。字符数组也可以在定义时初始化，方法和其他类型的数组一样。若没有对字符数组全部元素赋值，编译系统会对剩余的元素自动赋值为空字符。**空字符用'\0'来表示，是 ASCII 码值为 0 的字符，表示什么都不做，也不显示。**在定义字符数组之后，只能逐个给数组元素赋值。

6.3.2 字符数组的初始化

对字符数组初始化时有下面两种情况：

(1)可以对数组元素逐个初始化，例如：

char a[10]={ 'I', '　', 'a', 'm', '　', 'h', 'a', 'p', 'p', 'y'};

初值个数可以少于数组长度，多余元素自动为 '\0' ('\0'是二进制 0)。指定初值时，若未指定数组长度，则长度等于初值个数。例如：

char a[ ]={ 'I', '　', 'a', 'm', '　', 'h', 'a', 'p', 'p', 'y };

等价于：char a[10]={ 'I', '　', 'a', 'm', '　', 'h', 'a', 'p', 'p', 'y };

(2)用字符串常量对数组初始化，例如：

char c[]={"I am happy"};

也可以这样初始化(不要大括号)：char c[]="I am happy";

字符串在存储时，系统自动在其后加上结束标志 '\0' (占一字节，其值为二进制 0)

6.3.3 字符串常量赋给字符数组

若在定义字符数组的同时赋初值，则可将字符串常量赋给它。

例如：　char c[20]={ "The great wall"};

或写为：char c[20]= "The great wall";

若定义字符数组时完成赋初值，则可以在定义中省略数组的长度。系统会根据所赋字符串常量的实际长度来确定字符数组的长度。例如：char c[]="The great wall";

字符串常量末尾处有系统自动加的结束标志'\0'，所以要求数组长度比字符串长度至少大 1。例如：char c[14]= "The great wall";　//此定义，数组 c 不能满足字符串长度。

6.3.4 字符串的输入与输出

字符数组的输入输出可以有两种方法：

- 逐个字符输入输出（%c）
- 整个字符串一次输入输出（%s）

输出的字符中不包括结束符’\0’。用%s 输出字符串时，printf 函数中的输出项是字符数组名，不是数组元素名。

(1)使用 printf 函数输出字符串变量的方式是使用转换字符序列“%s”。如下面程序段：

char str[]="Hello";

printf("%s",str);　　//运行结果：Hello

(2)使用 scanf 函数输入字符串给字符串变量也使用转换字符序列“%s”。如下面程序段：

char str1[10],str2[10],str3[10],str4[10];

scanf("%s%s%s%s",str1,str2,str3,str4);　//scanf("%s")方式输入字符串以空格为结束标记

若输入字符串：　I love　my mother,则将四个字符串依次存到 4 个字符型数组中。

例 6.3.2．输入某月份的整数值 1～12，输出该月份的英文名称。

#include "stdio.h"

```
main( )
{     char month[ ][15]={"Illegal month.","January","February","March","April","May","June",
      "July","August","September","October","Novenber","Decenmber"};
      int    m;
      printf("\n 请输入月份的数字:");
      scanf("%d",&m);
      puts((m>0&&m<13)?month[m]: "输入不合法！ ")              /*使用条件运算符输出*/
      getcher();
}
```

例 6.3.3 当运行下面程序时，从键盘上输入 AabD↙,则写出下面程序的运行结果 AzyD 。

```
main ( )
{
      char s[80];
      int i=0;
      gets(s);   /*从键盘上输入字符串*/
      while (s[i]!= '\0')
      {
          if (s[i]<= 'z'&&s[i]>= 'a')
          s[i]= 'z'+'a'-s[i] ;
          i++;
      }
      puts(s);
      getcher();
}
```

如果字符数组元素为小写字母，则进行 a→z,b→y,c→x……z→a 的转换。

6.3.5 善于使用字符串处理函数

1.puts 函数----输出字符串的函数，作用是将一个字符串输出到终端

其一般形式为：　puts (字符数组)

```
char str[20]=”China”;
puts(str);          //输出 China
```

2. gets 函数----输入字符串的函数,作用是输入一个字符串到字符数组

其一般形式为：　gets(字符数组)

```
char str[20];
gets(str);
  Computer↙
```

使用下面这些字符串函数时,在程序开头用#include <string.h>

3. strcat 函数----字符串连接函数,其作用是把两个字符串连接起来，把字符串 2 接到字符串 1 的后面，结果放在字符数组 1 中

其一般形式为：　strcat(字符数组 1，字符数组 2)

```
char str1[30]=”People”;
char str2[]=”China”;
printf(”%s”, strcat(str1,str2));
  输出：PeopleChina
```

4. strcpy 和 strncpy 函数-字符串复制，作用是将字符串 2 复制到字符数组 1 中去

(1)strcpy 一般形式为：　strcpy(字符数组 1,字符串 2)

char str1[10],str2[]=”China”;
strcpy(str1,str2);　　//相当于 strcpy(str1,”China”);

char str1[10],str2[]=”China”;
str1=str2;　　错误

(2)strncpy 函数可以将字符串 2 中前面 n 个字符复制到字符数组 1 中去，复制的字符个数 n 不应多于 str1 中原有的字符

strncpy(str1，str2，2);
//作用是将 str2 中最前面 2 个字符复制到 str1 中，取代 str1 中原有的最前面 2 个字符

5. strcmp 函数----字符串比较函数，作用是比较字符串 1 和字符串 2

其一般形式为:　　strcmp(字符串 1，字符串 2)

strcmp(str1,str2);

字符串比较的规则是：

- 将两个字符串自左至右逐个字符相比，直到出现不同的字符或遇到’\0’为止
- 如全部字符相同，认为两个字符串相等
- 若出现不相同的字符，则以第一对不相同的字符的比较结果为准

比较的结果由函数值带回

- 如果字符串 1=字符串 2，则函数值为 0
- 如果字符串 1>字符串 2，则函数值为一个正整数
- 如果字符串 1<字符串 2，则函数值为一个负整数

例如：　if(strcmp(str1,str2)>0)　　printf(”yes”);

6. strlen 函数----测字符串长度的函数，函数的值为字符串中的实际长度，不包含结束标记。

其一般形式为：　　strlen (字符数组)

7. strlwr 函数----转换为小写的函数，作用是将字符串中大写字母换成小写字母

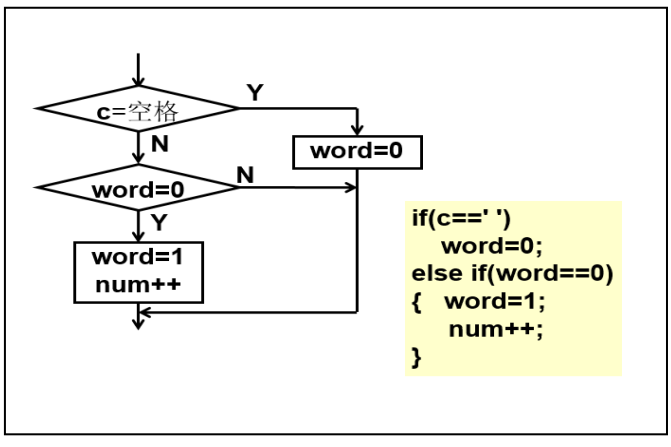
其一般形式为　　strlwr (字符串)

8. strupr 函数----转换为大写的函数，作用是将字符串中小写字母换成大写字母

其一般形式为：　　strupr (字符串)

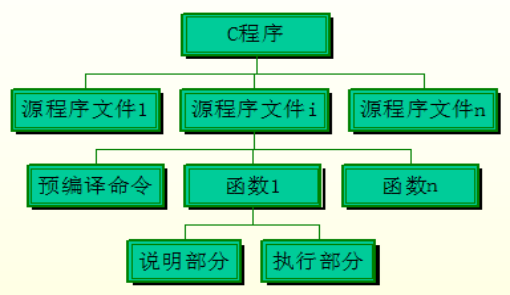
例 6.8 输入一行字符，统计其中有多少个单词，单词之间用空格分隔开。

```
……
char string[81],c;    int i,num=0,word=0;
gets(string);
for (i=0;(c=string[i])!=‘\0’;i++)
      if(c==‘ ’) word=0;
      else if(word==0)
      { word=1;
        num++;
      }
 printf(“%d words\n”,num);
 ……
```



第 7 章 用函数实现模块化程序设计

7.1 函数的定义、调用及声明



说明：

(1) 一个 C 程序由一个或多个程序模块组成，每一个程序模块作为一个源程序文件。对较大的程序，一般不希望把所有内容全放在一个文件中，而是将它们分别放在若干个源文件中，由若干个源程序文件组成一个 C 程序。

(2) 一个源程序文件由一个或多个函数以及其他有关内容（如预处理指令、数据声明与定义等）组成。一个源程序文件是一个编译单位，在程序编译时是以源程序文件为单位进行编译的，而不是以函数为单位进行编译的。

(3) C 程序的执行是从 main 函数开始的，如果在 main 函数中调用其他函数，在调用后流程返回到 main 函数，在 main 函数中结束整个程序的运行。

(4) 所有函数都是平行的，即在定义函数时是分别进行的，是互相独立的。一个函数并不从属于另一个函数，即函数不能嵌套定义。函数间可以互相调用，但不能调用 main 函数。main 函数是被操作系统调用的。

(5) 从用户使用的角度看，函数有两种。

- 库函数，它是由系统提供的，用户不必自己定义而直接使用它们。
- 用户自己定义的函数。它是用以解决用户专门需要的函数。

(6) 从函数的形式看，函数分两类。

- 无参函数。无参函数一般用来执行指定的一组操作。无参函数可以带回或不带回函数值，但一般以不带回函数值的居多。
- 有参函数。在调用函数时，主调函数在调用被调用函数时，通过参数向被调用函数传递数据，一般情况下，执行被调用函数时会得到一个函数值，供主调函数使用。

7.1.1 函数的定义

现代风格

```
函数类型  函数名（形参类型说明表）
{
  说明部分
  语句部分
}
```

函数体

函数返回值类型
缺省 int 型
无返回值 void

【例 7.1】编写函数,求三个整型参数的最大值。

```
int max(int x1,int x2,int x3)     /*定义函数的返回值类型，函数名，形参*/
{     int max;
      if (x1>x2)   max=x1;
      else   max=x2;
      if (max<x3) max=x3;          /*通过比较得到三个数的最大值放到 max 中*/
```

```
    return(max);                    /*返回运算结果*/
}
```

传统风格：

```
函数类型    函数名（形参表）
形参类型说明
{
        说明部分
        语句部分
}
```

例如：　有参函数（传统风格）（-------不推荐使用，但要能读懂）

```
int max(x,y)
int x,y;
{       int z;
        z=x>y?x:y;
        return(z);
}
```

7.1.2 函数的返回值

- 返回语句形式：

  return(表达式);

  或　return　表达式;

  或　return;
- 功能：使程序控制从被调用函数返回到调用函数中，同时把返值带给调用函数
- 说明：
  - 函数中可有多个 return 语句
  - 若无 return 语句，遇到函数结束的“}”时，自动返回调用函数
  - 若函数类型与 return 语句中表达式值的类型不一致，按前者为准，自动转换------函数调用转换
  - void 型函数：无返回值的函数

7.1.3 函数的调用

调用形式:　　函数名(实参表)

- 函数语句：函数调用可单独成为一个语句

  例：output();
- 函数表达式：调用函数后的返回值可参加表达式的计算，这时要求被调函数带返回值。

  例：sum=add(a,b);

  c=fac(n)/(fac(k)*fac(n-k)) ；
- 函数参数：带返回值的函数调用亦可作为其它函数的参数（实际参数）

  例：printf(“%d”,max(a,b));

  m=max(a,max(b,c));

调用函数的过程是：

① 为函数的所有形参分配内存单元。

②计算各个实参表达式的值，一一对应的赋值给相应形参（若是无参函数，上述过程不执行）。

③ 进入函数体，执行函数中的语句，实现函数的功能。

④执行到 return 语句时，计算 return 语句中表达式的值（若是无返回值函数，本项不做），返回

到主调函数。

⑤ 释放形参及本函数内的局部变量所占内存, 继续执行主调函数中的后继语句。

说明：

- 实参与形参个数相等，类型一致，按顺序一一对应
- 形参与实参的结合顺序，因系统而定（Turbo C 自右向左）

7.1.4 对被调用函数的声明和函数原型

函数原型的一般形式有两种：

如 float add(float x, float y);

float add(float, float);

原型说明位置：可以放在文件的开头，这时所有函数都可以使用此函数；也可以放在主调函数的声明部分。

C 语言中对被调用函数要求：

- 必须是已存在的函数
- 库函数: #include <*.h>
- 用户自定义函数：函数类型说明

对被调用函数声明的作用：告诉编译系统函数类型、参数个数及类型，以便检验;

- 函数定义与函数说明不同;
- 被调用函数定义出现在主调函数之前，可不作函数说明;

7.1.5 函数参数及其传递方式

形式参数：定义函数时函数名后面括号中的变量名。

实际参数：调用函数时函数名后面括号中的表达式。

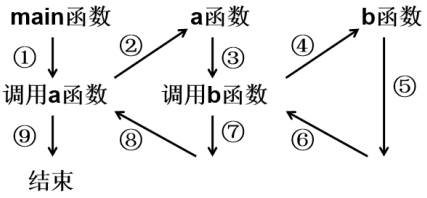
- 函数调用是值传递方式，即函数的实际参数和形式参数之间的数据传递方向是**单向**的。
- 在内存中，形式参数与实际参数占用不同的内存单元。当调用函数时，给形参分配内存单元，将实参的值赋值给形参，调用后，形参单元释放，实参仍保留调用前的值，形参值的变化不影响实参。

参数传递说明：

- 实参必须有确定的值
- 形参必须指定类型
- 形参与实参类型一致，个数相同
- 若形参与实参类型不一致，自动按形参类型转换——函数调用转换
- 形参在函数被调用前不占内存;函数调用时为形参分配内存；调用结束，内存释放

7.2 函数的嵌套调用、递归调用

7.2.1 函数的嵌套调用



7.2.2 函数的递归调用---在调用一个函数的过程中又出现直接或间接地调用该函数本身.

【例 7.7】 用递归方法求 n！

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

```
 {int fac(int n);
  int n;    int y;
  printf("input an integer number:");
  scanf("%d",&n);
  y=fac(n);
  printf("%d!=%d\n",n,y);
  return 0;
}
int fac(int n)
 {
      int f;
      if(n<0)
            printf("n<0,data error!");
      else if(n==0 | | n==1)
            f=1;
      else    f=fac(n-1)*n;
      return(f);
 }
```

7.3 数组作为函数参数

7.3.1 数组元素作为参数

【例 7.8】依序比较两个数组的元素值大小，最后输出 a 数组元素小于、大于、等于 b 数组元素的个数。

```
#include "stdio.h"
main( )
{
      int a[10]={1,2,3,4,5,6,7,8,9,10};
      int b[10]={2,3,4,4,5,6,7,7,8,9};
      int    i,na=0,nb=0,eq=0;
      int large(int,int);
      for(i=0;i<10;i++)
        {    if(large(a[i],b[i])==1)    na++;
             else if(large(a[i],b[i])==-1)    nb++;
             else eq++;
        }
      printf("a 大于 b:%d 个，a 小于 b:%d 个，a 等于 b:%d 个\n",na,nb,eq);
 }

int large(int x,int y)
{
      if(x>y) return 1;
      else if(x<y) return -1;
      else return 0;}
```

7.3.2 数组名作为参数---数组名中存放的是第一个数组元素的地址

------传递数组名，实际是传递地址，使实参和形参数组指向相同地址空间，实现双向传递。

【例 7.8】使用函数返回数组的平均值。

```
#include "stdio.h"
main( )
{
   int a[5],b[10],i;
   float avg(int array[],int y);
   float aver;
   printf("Please Enter Array-a:\n");
   for(i=0;i<5;i++)
   scanf("%d",&a[i]);
   printf("\n");
   printf("Please Enter Array-b:\n");
   for(i=0;i<10;i++)
   scanf("%d",&b[i]);
   printf("a 数组的平均值：%f,b 数组的平均值：%f\n",avg(a,5),avg(b,10));
}
float avg(int array[],int n)
{
   int i;
   float sum=0;
   for(i=0;i<n;i++)
     {
        sum+=array[i];
     }
   return(sum/n);
}
```

7.4 变量的作用域和生存期

变量的属性：

- 数据类型：变量所持有的数据的性质（操作属性）

存储属性

- 存储器类型：寄存器、静态存储区、动态存储区
- 生存期：变量在某一时刻存在-------静态变量与动态变量
- 作用域：变量在某区域内有效-------局部变量与全局变量

变量的存储类型

- auto -----自动型
- register-----寄存器型
- static ------静态型
- extern -----外部型

变量定义格式: [存储类型] 数据类型 变量表;

7.4.1 变量的作用域

从作用域角度考虑分为：内部变量、外部变量

C 语言中，按作用域的大小，可将变量的作用域分为四个级别：程序级、文件级、函数级、复合语句级。

一、内部变量（局部变量）

定义：在函数内或复合语句内定义

内部变量（局部变量）的作用域：是定义在**函数内**或**复合语句内**，在它的作用域之外，内部变量是不可见的，即一个函数内定义的内部变量是不能被其它的函数所引用的。即使不同的函数定义了同名的内部变量，对应不同的存储空间，所以也不会相互影响。

说明：

- main 中定义的变量只在 main 中有效
- 形参属于局部变量
- 局部变量可用存储类型：auto　　register　　static　（默认为 auto）

二、外部变量（全局变量）

定义：在函数外面定义的变量。

外部变量的作用域：对于只有一个源程序文件构成的程序，外部变量的作用域是**从定义它的位置开始，直至它所在源程序文件的结束。**可以 extern 扩展外部变量的作用域。

外部变量定义与外部变量说明不同：

在同一个文件中，定义在后使用在前的外部变量，在使用前需要对其进行声明。

在包含多个文件的程序中，一个文件若使用其它文件中定义的外部变量也要进行声明。

外部变量说明：

extern　　数据类型　　变量表；

	定义	说明
◆次数：	只能1次	可说明多次
◆位置：	所有函数之外	函数内或函数外
◆分配内存：	分配内存,可初始化	不分配内存,不可初始化

7.4.2　变量的存储类别------动态变量与静态变量

- 动态变量：动态存储类别的变量当进入定义它的函数或复合语句时被分配存储空间，当离开时所占内存空间被释放。
- 静态变量：静态存储类别的变量在源程序编译的时候被分配固定的存储空间，从程序开始执行到程序运行结束，一直占用该内存空间，直至程序运行结束，才被释放内存空间。

一、内部变量的存储类别

内部变量的作用域是定义它的函数或复合语句。内部变量的存储类别是指它存放的位置。内部变量可存放于内存的动态区，寄存器和内存的静态区。但无论内部变量存放在何处，它的作用域是不变的。

内部变量可以定义为：自动的（auto ）　//在函数内定义的变量默认的都是自动的。

寄存器（register ）//寄存器变量，如：register int i;

静态的（static ）　**//被分配在内存的静态存储区中。如：static　int i;**

关于 static 变量的说明：

- 分配内存后、赋初值，并且**只被赋初值一次**，未赋值的内部 static 变量，系统自动给它赋值为 0。
- static 变量在内存的静态存储区占用的固定的内存单元；即使它所在的函数被调用结束后，也不释放存储单元，它所在单元的值也会继续保留-----因此：下次再调用该函数时，static 变量仍使用原来的存储单元，仍使用原来存储单元中的值。

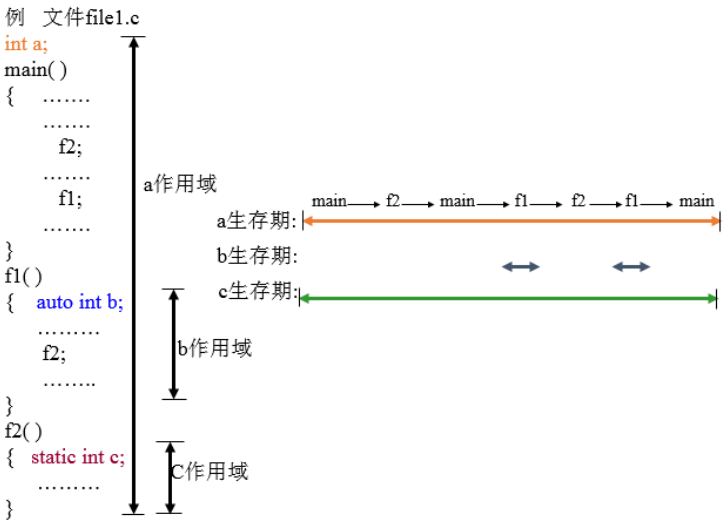
- 虽然 static 变量在整个程序运行期间都是存在的，但在它的作用域外，它是不可见的，也就是说其它函数是不能引用它的

二、外部变量的存储类别

外部变量只能存放在内存的静态存储区 。它在整个程序的运行期间一直占用内存单元。

在同一个文件中，定义在后使用在前的外部变量，在使用前需要“extern”对其进行声明。外部变量的作用域可以通过关键字“extern”来扩展。（扩展至某个文件，甚至被其它源程序文件使用）

在不同文件中（但同属一个程序），使用其它文件中定义的外部变量（但是不要企图使用 static 声明了的外部变量）。---用 static 声明的外部变量能够限制它的作用域的扩展，达到信息隐蔽的目的。



【例 7.10】 变量的寿命与可见性

```
#include <stdio.h>
int i=1;
main()
{     static int a;
      register int b=-10;
      int c=0;
      printf("-----MAIN------\n");
      printf("i:%d a:%d b:%d c:%d\n",i,a,b,c);
      c=c+8;
      other();
      printf("-----MAIN------\n");
      printf("i:%d a:%d b:%d c:%d\n",i,a,b,c);
      i=i+10;
      other();
}
other()
{     static int a=2;
       static int b;
      int c=10;
      a=a+2;     i=i+32;    c=c+5;
      printf("-----OTHER------\n");
      printf("i:%d a:%d b:%d c:%d\n",i,a,b,c);
      b=a;
}
```

输出结果：

```
-------Main------
i:1  a:0  b:-10  c:0
------Other------
i:33  a:4  b:0  c:15
-------Main-----
i:33  a:0  b:-10  c:8
-------Other-------
i:75  a:6  b:4  c:15
```

第 8 章 善于利用指针

8.1 指针是什么

指针是一个地址，指针变量的值为另一个变量的地址，即内存位置的直接地址。

就像其他变量或常量一样，您必须在使用指针变量存储其他变量地址之前，对其进行声明。

8.2 指针变量

8.2.1 使用指针变量的例子

```
#include <stdio.h>
int main()
{   int a=100,b=10;
    int *pointer_1, *pointer_2;
    pointer_1=&a;
    pointer_2=&b;
    printf("a=%d,b=%d\n",a,b);
    printf("*pointer_1=%d,*pointer_2=%d\n",*pointer_1,*pointer_2);
    return 0;
}
```

8.2.2 怎样定义指针变量

指针变量声明的一般形式为：type　*var-name;

在这里，**type** 是指针的基类型，它必须是一个有效的 C 数据类型，**var-name** 是指针变量的名称。用来声明指针的星号 * 与乘法中使用的星号是相同的。但是，在这个语句中，星号是用来指定一个变量是指针。

以下是有效的指针声明：

```
int     *ip;     /* 一个整型的指针 */
double *dp;      /* 一个 double 型的指针 */
float   *fp;     /* 一个浮点型的指针 */
char    *ch      /* 一个字符型的指针 */
```

所有指针的值的实际数据类型，不管是整型、浮点型、字符型，还是其他的数据类型，都是一样的，**都是一个代表内存地址的长的十六进制数**。不同数据类型的指针之间唯一的不同是，指针所指向的变量或常量的数据类型不同。

8.2.3 怎样引用指针变量

使用指针时会频繁进行以下几个操作：定义一个指针变量、把变量地址赋值给指针、访问指针变量中可用地址的值。这些是通过使用一元运算符 * 来返回位于操作数所指定地址的变量的值。

在引用指针变量时，可能有三种情况：

- 给指针变量赋值。如：p=&a;
- 引用指针变量指向的变量。如有　p=&a;　*p=1;
  则执行 printf(“%d”,*p);　将输出 1
- 引用指针变量的值。如：printf(“%o”,p);

要熟练掌握两个有关的运算符：

(1) &　取地址运算符。

&a 是变量 a 的地址

(2) * 指针运算符（“间接访问”运算符）

如果： p 指向变量 a，则*p 就代表 a。

k=*p; (把 a 的值赋给 k)

*p=1; (把 1 赋给 a)

【例 8.2.1】 指针变量的定义及使用

```
#include <stdio.h>
int main ()
{
    int   var=20;          //实际变量的声明
    int   *ip;             // 指针变量的声明
    ip=&var;               // 在指针变量中存储 var 的地址
    printf("Address of var variable: %x\n", &var   );    // 在指针变量中存储的地址
    printf("Address stored in ip variable: %x\n", ip );    // 使用指针访问值
    printf("Value of *ip variable: %d\n", *ip );
    return 0;
}
```

【例 8.2.2】输入 a 和 b 两个整数，按先大后小的顺序输出 a 和 b。

```
#include <stdio.h>
int main()
{ int *p1,*p2,*p,a,b;
    printf("integer numbers:");
    scanf("%d,%d",&a,&b);
    p1=&a;      p2=&b;
    if(a<b)
    {   p=p1; p1=p2; p2=p; }
    printf("a=%d,b=%d\n",a,b);
    printf("%d,%d\n",*p1,*p2);
    return 0;
}
```

8.2.4 指针变量作为函数参数

【 例 8.2.3 】对输入的两个整数按大小顺序输出。现用函数处理，而且用指针类型的数据作函数参数。

```
#include <stdio.h>
int main()
{void swap(int *p1,int *p2);
   int a,b;   int*pointer_1,*pointer_2;
   printf("please enter a and b:");
   scanf(“%d,%d”,&a,&b);
   pointer_1=&a;
   pointer_2=&b;
   if (a<b)   swap(pointer_1,pointer_2);
   printf(“max=%d,min=%d\n”,a,b);
```

```
    return 0;
  }
 void swap(int *p1,int *p2)
 { int temp;
     temp=*p1;
     *p1=*p2;
     *p2=temp;
 }
```

【例 8.2.4】 传递一个无符号的 **long** 型指针给函数，并在函数内改变这个值。

```
#include <stdio.h>
#include <time.h>
void getSeconds(unsigned long *par);
int main ()
{
    unsigned long sec;
    getSeconds( &sec );     /* 输出实际值 */
    printf("Number of seconds: %ld\n", sec );
    return 0;
}
void getSeconds(unsigned long *par)                    /* 获取当前的秒数 */
{
    *par = time( NULL );
    return;
}
```

8.3 通过指针引用一维数组

8.3.1 什么是数组元素的指针？

一个变量有地址，一个数组包含若干元素，每个数组元素都有相应的地址；指针变量可以指向数组元素（把某一元素的地址放到一个指针变量中）；所谓数组元素的指针就是数组元素的地址

可以用一个指针变量指向一个数组元素

```
int a[10]={1,3,5,7,9,11,13,15,17,19};
int  *p;
```

p=&a[0]; //或者 p=a; 注意：数组名 a 不代表整个数组，只代表数组首元素的地址。“p=a;”的作用是“把 a 数组的首元素的地址赋给指针变量 p”，而不是“把数组 a 各元素的值赋给 p”。

8.3.2 数组元素的指针运算

在指针指向数组元素时，允许以下运算：

- ✓ 加一个整数(用+或+=)，如 p+1
- ✓ 减一个整数(用-或-=)，如 p-1
- ✓ 自加运算，如 p++，++p
- ✓ 自减运算，如 p--，--p
- ✓ 两个指针相减，如 p1-p2 (只有 p1 和 p2 都指向同一数组中的元素时才有意义)
- ✓ 注意：数组名是常量，指向数组元素首地址。不能做自加、自减运算。

(1) 如果指针变量 p 已指向数组中的一个元素，则 p+1 指向同一数组中的下一个元素，p-1 指向同一数组中的上一个元素。

(2) 如果 p 的初值为&a[0]，则 p+i 和 a+i 就是数组元素 a[i]的地址，或者说，它们指向 a 数组序号为 i 的元素.

(3) *(p+i)或*(a+i)是 p+i 或 a+i 所指向的数组元素，即 a[i]。

(4) 如果指针 p1 和 p2 都指向同一数组, p2-p1 的值是元素之间的偏移量(为整数),不能 p1+p2。

8.3.3 数组元素的引用

可用下面两种方法：

(1)下标法，如 a[i] 或 p[i]

(2)指针法，如*(a+i)或*(p+i) //其中 a 是数组名，p 是指向数组元素的指针变量，其初值 p=a

【例 8.3.1】有一个整型数组 a，有 10 个元素，要求输出数组中的全部元素。

(1) 下标法或指针法

```
#include <stdio.h>
int main()
{ int a[10],i; int *p=a;                    //等价于*p=&a[0];
    printf("enter 10 integer numbers:\n");
    for(i=0;i<10;i++) scanf("%d",&a[i]);       //&a[i] 可替换为 a+i 或 p+i
    for(i=0;i<10;i++)   printf("%d ",a[i]);     //下标法。 a[i]可替换为*(a+i)或*(p+i)
}
```

(2) 用指针变量指向数组元素

```
#include <stdio.h>
int main()
{ int a[10];   int *p,i;
   printf("enter 10 integer numbers:\n");
   for(i=0;i<10;i++)   scanf("%d",&a[i]);
   for(p=a;p<(a+10);p++)
        printf("%d ",*p);        //注意：循环完毕后，指针已经移动到了最后一个数组元素下面！
}
```

8.3.4 用数组名作函数参数

用数组名作函数参数时，因为实参数组名代表该数组首元素的地址，形参应该是一个指针变量。

```
int main()
{ void fun(int arr[],int n];
    int array[10];        ⋮
      ⋮
    fun (array,10);
    return 0;
}
void fun(int arr[ ],int n)          //等价于：fun(int *arr,int n)
{  ⋮   }
```

【例 8.3.2】 将数组 a 中 n 个整数按相反顺序存放

```
#include <stdio.h>
int main()
{ void inv(int x[ ],int n);
    int i, a[10]={3,7,9,11,0,6,7,5,4,2};
```

```
    for(i=0;i<10;i++) printf("%d ",a[i]);
    printf("\n");
    inv(a,10);
    for(i=0;i<10;i++) printf("%d ",a[i]);
    printf("\n");
    return 0;
}
void inv(int x[ ],int n)
{ int temp,i,j,m=(n-1)/2;
    for(i=0;i<=m;i++)
    { j=n-1-i;
      temp=x[i];x[i]=x[j];x[j]=temp;
    }
}
```

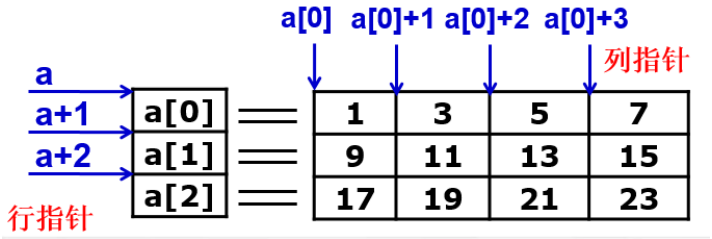
优化：

```
void inv(int x[ ],int n)
{ int temp,*i,*j;
    i=x;   j=x+n-1;
    for(    ; i<j; i++,j--)
    { temp=*i; *i=*j; *j=temp; }
}
```

8.4 指向二维数组的指针

8.4.1. 多维数组元素的地址

int a[3][4]={{1,3,5,7},{9,11,13,15},{17,19,21,23}};



行指针：a+i 代表行号为 i 的行首地址（按行变化）

*(a+i)代表什么？-----**相当于 a[i]**

列指针：a[i]+j 代表谁的地址？----**代表 a[i][j]的地址**

(a[i]+j)代表什么？(*(a+i)+j)代表什么？ ----**代表元素 a[i][j]**

8.4.2 指向多维数组元素的指针变量

(1) 指向数组元素的指针变量

【例 8.12】 有一个 3×4 的二维数组，要求用指向元素的指针变量输出二维数组各元素的值。

```
#include <stdio.h>
int main()
{ int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
    int *p;
    for(p=a[0];p<a[0]+12;p++)
    { if((p-a[0])%4==0) printf("\n");
        printf("%4d",*p);   //逐个访问各元素时常用此类指针
    }
}
```

(2) 指向由m个元素组成的一维数组的指针变量

【例 8.13】 输出二维数组任一行任一列元素的值。

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

```
int main()
{int a[3][4]={1,3,5,7,9,11,13,15, 17,19,21,23};
   int (*p)[4],i,j;                    //指针变量 p 指向包含 4 个整形元素的一维数组
   p=a;
   printf("enter row and colum:");
   scanf("%d,%d",&i,&j);
   printf("a[%d,%d]=%d\n", i,j,*(*(p+i)+j));
}
```

(3)用指向数组的指针作函数参数

- 一维数组名可以作为函数参数，多维数组名也可作函数参数。
- 用指针变量作形参，以接受实参数组名传递来的地址。
- 可以有两种方法：

①用指向变量的指针变量

②用指向一维数组的指针变量

【例 8.14】 有一个班，3 个学生，各学 4 门课，计算总平均分数以及第 n 个学生的成绩。

```
#include <stdio.h>
int main()
{   void average(float *p,int n);
    void search(float (*p)[4],int n);
    float score[3][4]={{65.55,67,70.5,60},{80.22,87,90,81},{90.33,99,100,98}};
    int x;
    average(*score,12);
    printf("The score of No.%d are:\n",x);
    scanf("%d",&x);
    search(score,x);
}
void average(float *p,int n)
{ float *p_end;
    float sum=0,aver;
    p_end=p+n-1;
    for(     ;p<=p_end; p++)
            sum=sum+(*p);
    aver=sum/n;
    printf("average=%5.2f\n",aver);
}
void search(float (*p)[4],int n)
{ int j;
  for(j=0;j<4;j++)
          printf("%5.2f ",*(*(p+n)+j));
  printf("\n");
}
```

8.5 通过指针引用字符串

8.5.1 字符串的引用方式

字符串是存放在字符数组中的。引用一个字符串，可以用以下两种方法：

(1) 用字符数组存放一个字符串，可以通过数组名和格式声明“%s”输出该字符串，也可以通过数组名和下标引用字符串中一个字符。

(2) 用字符指针变量指向一个字符串常量，通过字符指针变量引用字符串常量。

【例 8.16】 定义一个字符数组，在其中存放字符串“I love China!”，输出该字符串和第 8 个字符。

```
#include <stdio.h>
int main()
{ char string[]=“I love China!”;
    printf(“%s\n”,string);
    printf(“%c\n”,string[7]);
    return 0;
}
```

【例 8.17】 通过字符指针变量输出一个字符串。

```
#include <stdio.h>
int main()
{ char *string=“I love China!”;
    printf(“%s\n”,string);
    return 0;
}
```

【例 8.18】 将字符串 a 复制为字符串 b，然后输出字符串 b。

```
#include <stdio.h>
int main()
{ char a[ ]=“I am a student.”,b[20];
    int i;
    for(i=0;*(a+i)!='\0';i++)
        *(b+i)=*(a+i);
    *(b+i)=‘\0’;
    printf(“string a is:%s\n”,a);
    printf("string b is:");
    for(i=0;b[i]!='\0';i++)
        printf(“%c”,b[i]);
    printf("\n");
    return 0;
}
```

```
#include <stdio.h>
int main()
{char a[]="I am a boy.",b[20],*p1,*p2;
   p1=a;   p2=b;
   for(     ; *p1!=‘\0’; p1++,p2++)
        *p2=*p1;
   *p2=‘\0’;
   printf(“string a is:%s\n”,a);
   printf(“string b is:%s\n”,b);
   return 0;
}
```

8.5.2 字符指针作函数参数

如果想把一个字符串从一个函数“传递”到另一个函数，可以用地址传递的办法，即用字符数组名作参数，也可以用字符指针变量作参数。

【例 8.20】 用函数调用实现字符串的复制。

(1) 用字符数组名作为函数参数

```
#include <stdio.h>
int main()
{void copy_string(char from[],char to[]);
```

```
    char a[]="I am a teacher.";
    char b[]="you are a student.";
    printf("a=%s\nb=%s\n",a,b);
    printf("copy string a to string b:\n");
    copy_string(a,b);
    printf("a=%s\nb=%s\n",a,b);
    return 0;
}
void copy_string(char from[], char to[])
{ int i=0;
    while(from[i]!='\0')
    {    to[i]=from[i];
         i++;
     }
     to[i]='\0';
}
```

8.5.3 使用字符指针变量和字符数组的比较

用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

(1) 字符数组由若干个元素组成，每个元素中放一个字符，而字符指针变量中存放的是地址（字符串第 1 个字符的地址），决不是将字符串放到字符指针变量中。

用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

(2) 赋值方式。可以对字符指针变量赋值，但不能对数组名赋值。

char *a; a=”I love China!”； 对

char str[14];str[0]=’I’； 对

char str[14]; str=”I love China!”；错

用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

（3）初始化的含义

char *a=”I love China!”； //char *a; a=”I love China!”;等价

char str[14]= ”I love China!”； //char str[14]; str[]=”I love China!”； 不等价

(4) 存储单元的内容

编译时为字符数组分配若干存储单元，以存放各元素的值，而对字符指针变量，只分配一个存储单元。

```
char *a; scanf(“%s”,a);    错
char *a,str[10];
a=str;
scanf (“%s”,a);          对
```

(5) 指针变量的值是可以改变的，而数组名代表一个固定的值(数组首元素的地址)，不能改变。

(6) 用指针变量指向一个格式字符串，可以用它代替 printf 函数中的格式字符串。

```
char *format;
format=”a=%d,b=%f\n”;
printf(format,a,b);
```

相当于：

printf(“a=%d,b=%f\n”,a,b);

8.6 指向函数的指针

8.5.1 什么是函数指针及定义函数指针

如果在程序中定义了一个函数，在编译时，编译系统为函数代码分配一段存储空间，这段**存储空间的起始地址**，称为这个函数的指针。

可以定义一个指向函数的指针变量，用来存放某一函数的起始地址，这就意味着此指针变量指向该函数。

定义指向函数的指针变量的一般形式为：

数据类型 (*指针变量名)(函数参数表列);

例如：

int (*p)(int,int);

定义 p 是指向函数的指针变量，它可以指向类型为整型且有两个整型参数的函数。p 的类型用 int (*)(int,int)表示。

8.5.2 用函数指针变量调用函数

```
#include <stdio.h>
int main()
{ int max(int,int);
    int (*p)(int,int);   //只能指向函数返回值为整型且有两个整型参数的函数
    int a,b,c;
    p=max;                    //必须先指向，若写成 p=max(a,b); 错
    printf("please enter a and b:");
    scanf("%d,%d",&a,&b);
    c=(*p)(a,b);
    printf(“%d,%d,max=%d\n",a,b,c);
    return 0;
}
```

8.5.3 使用指向函数的指针变量

如： int (*p)(int,int);

p=max; 对

p=max(a,b); 错

p+n,p++,p--等运算无意义

8.5.4 用指向函数的指针作函数参数

指向函数的指针变量的一个重要用途是把函数的地址作为参数传递到其他函数。

指向函数的指针可以作为函数参数，把函数的入口地址传递给形参，这样就能够在被调用的函数中使用实参函数。

```
……
int main()
{   ……    fun(f1,f2)   ……    }

void   fun(int (*x1)(int),int (*x2)(int,int))
{   int a,b,i=3,j=5;
```

```
    a=(*x1)(i);          //相当于 a=f1(i);
    b=(*x2)(i,j);        //相当于 b=f2(i,j);
}
```

【例 8.24】 有两个整数 a 和 b，由用户输入 1,2 或 3。如输入 1，程序就给出 a 和 b 中大者，输入 2，就给出 a 和 b 中小者，输入 3，则求 a 与 b 之和。

```
#include <stdio.h>
int main()
{void fun(int x,int y, int (*p)(int,int));
   int max(int,int);    int min(int,int);
   int add(int,int);      int a=34,b=-21,n;
   printf("please choose 1,2 or 3:");
   scanf(“%d”,&n);
   if (n==1)       fun(a,b,max);
   else if (n==2) fun(a,b,min);
   else if (n==3) fun(a,b,add);
   return 0;
 }
int fun(int x,int y,int (*p)(int,int))
{ int resout;
    resout=(*p)(x,y);
    printf(“%d\n”,resout);
}
int max(int x,int y)
{ int z;
    if(x>y)    z=x;
    else      z=y;
    printf("max=" );
    return(z);
   }
int min(int x,int y)
{ int z;
    if(x<y) z=x;
    else z=y;
    printf("min=");
    return(z);
}
int add(int x,int y)
{ int z;
    z=x+y;
    printf("sum=");
    return(z);
}
```

8.7 返回指针值的函数

一个函数可以返回一个整型值、字符值、实型值等，也可以返回指针型的数据，即地址。其概念与以前类似，只是返回的值的类型是指针类型而已。定义返回指针值的函数的一般形式为：

类型名 *函数名(参数表列);

```
int * myFunction(      )
{
…
}
```

另外，C 不支持在函数外返回局部变量的地址，除非定义局部变量为 static 变量。

【例 8.7.1】生成 10 个随机数，并使用表示指针的数组名（即第一个数组元素的地址）来返回它们。

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

int * getRandom( )                                  /*  要生成和返回随机数的函数  */
{
    static int    r[10];
    int i;
    srand( (unsigned)time( NULL ) );                /*  设置随机种子  */
    for ( i = 0; i < 10; i++)
    {
        r[i] = rand();
        printf("%d\n", r[i] );
    }
    return r;
}
int main ()
{
    int *p;                 /*  一个指向整数的指针  */
    int i;
    p=getRandom( );
    for ( i = 0; i < 10; i++ )
    {
          printf("*(p + [%d]) : %d\n", i, *(p + i) );
    }
     return 0;
}
```

【例 8.25】有 a 个学生，每个学生有 b 门课程的成绩。要求在用户输入学生序号以后，能输出该学生的全部成绩。用指针函数实现。

```
#include <stdio.h>
int main()
{float score[ ][4]={{60,70,80,90}, {56,89,67,88},{34,78,90,66}};
   float    *search(float (*pointer)[4],int n);
```

```
    float   *p;   int i,k;
    scanf("%d",&k);
    printf("The scores of No.%d are:\n",k);
    p=search(score,k);     //返回 k 号学生课程首地址
    for(i=0;i<4;i++)
        printf("%5.2f\t",*(p+i));
    printf("\n");
    return 0;
}
float *search(float (*pointer)[4],int n)
{ float *pt;
    pt=*(pointer+n);
    return(pt);
}
```

8.8 指针数组和多重指针

8.8.1 什么是指针数组

一个数组，若其元素均为指针类型数据，称为指针数组，也就是说，指针数组中的每一个元素都存放一个地址，相当于一个指针变量。

定义一维指针数组的一般形式为:

类型名*数组名[数组长度];

例如： int *p[4];

- 指针数组比较适合用来指向若干个字符串，使字符串处理更加方便灵活
- 可以分别定义一些字符串，然后用指针数组中的元素分别指向各字符串
- 由于各字符串长度一般是不相等的，所以比用二维数组节省内存单元

【例 8.27】 将若干字符串按字母顺序（由小到大）输出。

```
#include <stdio.h>
#include <string.h>
int main()
{void sort(char *name[ ],int n);
   void print(char *name[ ],int n);
   char *name[ ]={"Follow","Great", "FORTRAN","Computer"};
   int n=4;
   sort(name,n);
   print(name,n);
   return 0;
}
void sort(char *name[ ],int n)
{char *temp;   int i,j,k;
   for (i=0;i<n-1;i++)
   { 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; }
        }
    }
    void print(char *name[ ],int n)
    {   int i;
        for(i=0;i<n;i++)
            printf("%s\n",name[i]);
    }
```

【例 8.28】　指针数组的定义、赋值及使用

```
#include <stdio.h>
const int MAX=3;
int main ()
{
    int    var[] = {10, 100, 200};
    int i, *ptr[MAX];
    for(i=0;i<MAX;i++)
    {
        ptr[i] = &var[i];        // 赋值为整数的地址
    }
    for(i=0;i<MAX;i++)
    {
        printf("Value of var[%d]=%d\n",i,*ptr[i]);
    }
    return 0;
}
```

8.8.2 指向指针数据的指针------指向指针的指针

指向指针的指针是一种多级间接寻址的形式，或者说是一个指针链。通常，一个指针包含一个变量的地址。当我们定义一个指向指针的指针时，第一个指针包含了第二个指针的地址，第二个指针指向包含实际值的位置。

一个指向指针的指针变量必须如下声明，即在变量名前放置两个星号。

例如，下面声明了一个指向 int 类型指针的指针：

```
int **var;
```

当一个目标值被一个指针间接指向到另一个指针时，访问这个值需要使用两个星号运算符，如下面实例所示：

【例 8.29】　多重指针的定义、赋值及使用

```
#include <stdio.h>
int main ()
{
    int    var;
```

```
    int    *ptr;
    int    **pptr;
    var = 3000;
    ptr = &var;                                    /* 获取 var 的地址 */
    pptr = &ptr;                                   /* 使用运算符 & 获取 ptr 的地址 */
    printf("Value of var = %d\n", var );           /* 使用 pptr 获取值 */
    printf("Value available at *ptr = %d\n", *ptr );
    printf("Value available at **pptr = %d\n", **pptr);
    return 0;
}
```

【例 8.30】 字符指针数组的定义及使用

```
#include <stdio.h>
const int MAX = 4;
int main ()
{
    char *names[]= {"Zara Ali","Hina Ali","Nuha Ali", "Sara Ali"};
    int i = 0;
    for ( i = 0; i < MAX; i++)
    {
        printf("Value of names[%d] = %s\n", i, names[i] );
    }
    return 0;
}
```

8.8.3 指针数组作 main 函数的形参

main 函数是操作系统调用的，实参只能由操作系统给出。

- 一般情况下 main 函数写成以下形式： int main() 或 int main(void)
  表示 main 函数没有参数，调用 main 函数时不必给出实参。
- 在某些情况下，main 函数可以有参数，例如： int main(int argc,char *argv[])
  其中，argc 和 argv 就是 main 函数的形参，它们是程序的“命令行参数”。
  argv 是*char 指针数组，数组中每一个元素(其值为指针)指向命令行中的一个字符串。

【例 8.31】 输出传递给 main 函数的参数列表。

```
#include <stdio.h>
int main(int argc,char *argv[])
{ while(argc>1)
    { ++argv;
      printf(“%s\n”, *argv);
      --argc;
    }
    return 0;
}
```

8.9 有关指针的小结

1.首先要准确地弄清楚指针的含义。指针就是地址，凡是出现“指针”的地方，都可以用“地址”代替，例如，变量的指针就是变量的地址，指针变量就是地址变量

要区别指针和指针变量。指针就是地址本身，而指针变量是用来存放地址的变量。

2. 什么叫“指向”？地址就意味着指向，因为通过地址能找到具有该地址的对象。对于指针变量来说，把谁的地址存放在指针变量中，就说此指针变量指向谁。但应注意：只有与指针变量的基类型相同的数据的地址才能存放在相应的指针变量中。

3. 要深入掌握在对数组的操作中怎样正确地使用指针，搞清楚指针的指向。一维数组名代表数组首元素的地址

int *p,a[10];

p=a;

- p 是指向 int 类型的指针变量，p 只能指向数组中的元素，而不是指向整个数组。在进行赋值时一定要先确定赋值号两侧的类型是否相同，是否允许赋值。
- 对“p=a;”，准确地说应该是：p 指向 a 数组的首元素

int p;	定义简单变量

int *p;	定义一个基类型为整型的指针变量
	主要用法：1.指向一个简单变量；
	2.用作形参；（注意：在主调函数中的实参应该是传递地址值）
	main()
	{
	……int *pa=&a,*pb=&b;
	Max(&a,&b); //max(pa,pb)
	}
	
	Void max(int *p1,int *p2)
	{
	……
	}
	3.指向一维数组。
	int *p; int a[10];
	p=a; //p=&a[0];
	下标法： a[i] //p[i]
	指针法： *(a+i) //*(p+i)

int a[10];	定义一维数组（数组名是首元素的地址，是一个常量）
char *p[10];	指针数组 char *name[10]={“hhh”,”aaa”,”bbb”,……} for(i=0;i<10;i++) printf(“%s\t”,name[i]);
int (*p)[10];	指向一维数组的指针变量，常用于作为二维数组的行指针 int a[5][10]; int (*p)[10]; p=a; 访问数组元素 a[i][j]的几种方法： ➔*(a[i]+j) ➔*(*(a+i)+j) ➔*(p[i]+j) ➔*(*(p+i)+j)
int p(int);	函数,如： void max(int x,int y) {

	}
int p(int *);	形参带指针变量的函数
int *p(int);	指针函数（返回值为指针的函数）

int (*p)(int);	函数指针
	main()
	{……
	int (*p)(int，int); //定义函数指针
	p=max; //函数指针赋值
	c=(*p)(a,b) //使用函数指针来访问函数，等价于:c=max(a,b)
	}
	int max(int x,int y)
	{
	}

char *p；	字符指针，通常用来指向一维字符数组；或者直接指向字符串
	char *p=”hello”;// char *p; p=”hello”; 对
	char str[]=”hello”;// char str[10]; str=”hello”;错
	printf(“%s”,p);
	printf(“%s”,str);

char **p;	int a;
	int *pa; pa=&a;
	int **pp; pp=&pa;
	三种方法访问 a：(1) a (2) *pa (3)**pp
	双重指针，通常结合字符指针数组来使用；
	char *name[10]={“hhh”,”aaa”,”bbb”,……}
	char **p;
	p=name; //p=&name[0];
	for(i=0;i<10;i++)
	printf(“%s\t”,name[i]); //等价于： printf(“%s\t”,*(p+i));

第 9 章 结构体与共同体

9.1 结构体类型的定义

结构体类型是在应用原有类型的基础上，用户构造的一种类型，其成员丰富，引用时可以整体引用。以前学过的数组在定义后所有数组元素都属同一类型，而本章所学结构体各个成员可以是不同类型。它的定义形式为：

struct 结构体标识符
{ 类型名 成员变量名 1；
类型名 成员变量名 2；
……
类型名 成员变量名 n；}

这里，struct 是定义结构体类型的关键字，结构体标识符要求是合法的 C 语言标识符。实例中，定义了一个记录个人信息的结构体类型，有 num、name、sex、score 这 4 个成员。

9.2 结构体类型的变量和指针变量的定义

结构体类型的变量定义方法有多种。可以在定义结构体的同时定义结构体类型的变量；也可以先定义结构体类型，然后再定义相应的变量；也可以通过 typedef 关键字先为定义的结构体类型命个新名字，再用新名字定义结构类型的变量。

对于结构体类型的变量，定义时在内存空间中为其分配存储空间，分配时按先后顺序连续分配，所占空间总的大小为所有成员所占空间大小的和值。

```
struct  STU
{int num;      char *name;          char sex;          float score;      }s,*p;
```

//定义结构体类型的名字 struct STU，然后用该结构体类型申明该类型的变量和指针变量，这样多次定义时比较简洁。

结构体类型的嵌套定义，注意引用时分层引用，不可越层。例如：

```
struct  stu
{int num;
char *name;
struct  birthday { int  year;   int  month;   int  day;} birth;
                    // struct  birthday 为类型名，birth 为 struct  stu 类型的成员名；
char sex;
float score; }s,*p;
```

【相关知识】

```
struct  stu
      { int num;
        char *name;
        char sex;
        float score;
      }s={101,"lihong",'F',95.5},*p;
```

以上程序段是在定义结构体类型变量的同时对变量进行了初始化。

结构体类型定义时，可以用关键字 typedef 为定义的结构体类型变量新命一个名字，然后可以用这个新名字来定义结构类型的变量。如下实例中的定义形式：

```
typedef struct
    {int num;
     char *name;
     char sex;
     float score;
    }STU;
STU girl1, girl2,*girl3;      /*定义结构体类型变量*/
```

typedef 还可以为其他类型起个别名，如 int、char 等，均可以为之起别名。

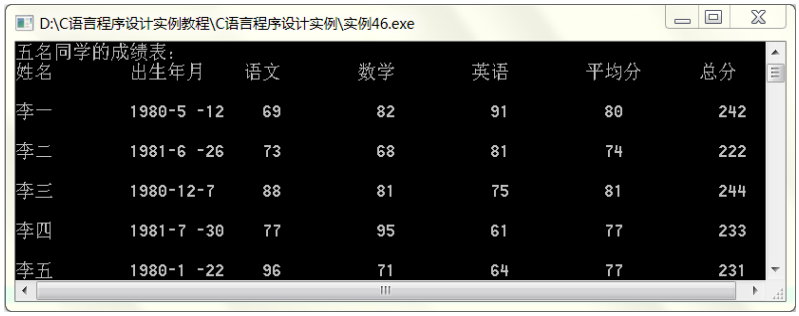
9.3 结构体成员的引用

结构体成员的引用与数组元素的引用相似，对各个成员要分别引用。结构体成员引用的运算符有“*”和“->”，引用形式如下：

- 结构体变量名.结构体成员　　如：**girl2.num**
- 结构体变量指针->结构体成员　　如：**girl3->num**

- (*结构体变量指针).结构体成员　　如：**(*girl3).sex**

【实例 9.1】嵌套定义一个结构体，定义一个结构类型的变量并赋初值，编程输出学生的信息并输出每位同学的平均成绩和总成绩。程序运行结果如图所示。




```
#include   "stdio.h"
struct   birthday
{int year;
 int month;
 int day;};
struct student
{ char name[10];
   struct birthday date;
   int chinese;
   int math;
   int english;
   int ave;
   int count;
}stu[5]={ {"李一",1980,5,12,69,82,91},
          {"李二",1981,6,26,73,68,81},
          {"李三",1980,12,7,88,81,75},
          {"李四",1981,7,30,77,95,61},
          {"李五",1980,1,22,96,71,64}};
/*嵌套定义结构体类型，定义结构体类型数组的同时进行初始化*/
main()
{ char *p[10]={"姓名","出生年月","语文","数学","英语","平均分","总分"};
   int i;
 for(i=0;i<5;i++)
 {stu[i].count=stu[i].chinese+stu[i].math+stu[i].english;
  stu[i].ave=(stu[i].count)/3;
 }
printf("五名同学的成绩表：\n");
for(i=0;i<7;i++)
printf("%-12s",p[i]);
for(i=0;i<5;i++)
printf("\n%-12s%-4d-%-2d-%-6d%-12d%-12d%-12d%-12d",stu[i].name,stu[i].date.year,stu[i].date.month,stu[i].date.day,stu[i].chinese,stu[i].math,stu[i].english,stu[i].ave,stu[i].count);
```

```
getch();
}
```

【实例 9.2】定义一个结构体类型，然后定义两个自定义结构体类型的变量，通过引用这两个变量输出个人信息。

```
#include "stdio.h"
main()
{ typedef struct
   {int num;
    char *name;
    char sex;
    float score;
   }STU;
STU girl1, girl2,*girl3;                /*定义结构体类型*/
girl1.num=102;   girl1.name="lihong";
printf("请输入性别和成绩：");
scanf("%c%f",&girl1.sex,&girl1.score);
girl2=girl1;                            /*可以为结构体类型的变量整体赋值*/
girl3=&girl1;                           /*可以让结构体类型的变量指向结构体类型的变量*/
printf("输出我的学号、姓名、性别、成绩的个人信息为：\n");
printf("%10d%10s   %c   %.2f",girl2.num,girl2.name,girl2.sex,girl2.score);
printf("\n 通过指针变量输出我的学号、姓名、性别、成绩的个人信息为：\n");
printf("%10d%10s   %c   %.2f",girl3->num,girl3->name,(*girl3).sex,(*girl3).score);
getch();
}
```

9.4 结构体类型的数组的定义与引用——成绩统计

1．结构体类型数组的定义

结构体类型一旦定义好后，就可以和 C 语言固有类型一样定义数组，只是每个数组元素都是该结构体类型。

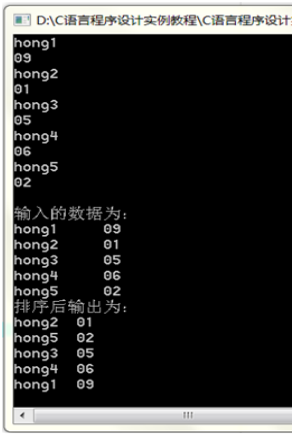
可以在定义结构体类型 student 的同时定义结构体类型的数组，并为每个数组元素赋初值。每个数组元素都是结构体类型，赋值时按各成员的顺序依次赋值。且每个数组元素的值用｛｝括起来。

2．结构体类型数组元素的引用

结构体类型数组元素引用时，要指明引用哪个数组元素的哪个成员，如表达式 stu[i].count 就是引用数组元素 stu[i]的 count 成员。

当结构体类型数组元素为结构体类型指针变量赋值时，可以直接取该数组元素的地址，让该指针变量指向该数组元素，则可以通过指针变量引用数组元素的各个成员。

【实例 9.3】输入几名学生的姓名和学号，然后按学号由小到大的顺序排序，再输出排序后的结果。运行结果如图。




```
#include "stdio.h"
typedef struct
{   char   name[20];     char   number[5];
```

```
}STU;
main()
{ STU s[5];
   int i;
   for(i=0;i<5;i++)
   getdata(&s[i]);              /*将结构体类型变量的地址传递给形参*/
   printf("\n 输入的数据为： ");
   for(i=0;i<5;i++)
      printf("\n%-10s%-5s",s[i].name,s[i].number);
tosort(s);                      /*将结构体类型的一维数组元素的首地址传递给形参*/
   outdata(s);
   getch();
}

getdata(STU *p)
{   {gets(p->name);
     gets(p->number);
    }
}

tosort(STU *p)
{ int i,j,k;
   STU temp;
   for(i=0;i<4;i++)
   {k=i;
     for(j=i+1;j<5;j++)
       if(strcmp(p[k].number,p[j].number)>0) k=j;
       temp=p[k];p[k]=p[i];p[i]=temp;
   }
}
```

【相关知识】

1．向形参传递结构体类型变量成员的值

结构体变量的成员，和前几章学过的基本数据类型的变量、数组、指针等变量是一样的，值传递时，可以直接传递成员的值；地址传递时，直接取成员的地址传递给形参就可以了。

2．向形参传递结构体类型变量的值

当向形参传递结构体类型变量的值的时候，形参需定义为该结构体类型。在自定义函数内对形参的任何重新赋值过程不会影响到实参。

3．向形参传递结构体类型变量的地址

当向函数传递结构体类型变量的地址时，对应的形参需定义为基类型为该结构体类型的指针变量。这样，指针变量直接指向实参的结构体类型的变量，系统中只需开辟一个存储单元存放指针变量即可。实例中，调用 getdata 函数时，向形参传递的就是结构体类型变量的地址。

4.向形参传递结构体类型数组名

当向函数传递结构体类型数组名时，形参需定义为基类型是该结构体类型的指针变量。因为数组名的值就是这个数组的首地址，数据传递后该指针变量就指向了这个结构体类型的数组，

引用时可以通过指针的移动来引用各个数组元素，且自定义函数时对结构体成员的重新赋值即对实参各个成员的重新赋值。实例中，调用 tosort 和 outdata 函数时，向形参传递的是结构体类型数组名。

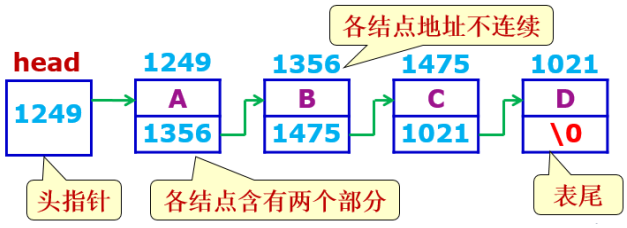
5.函数数返回结构体类型值或返回指向结构体类型的指针

结构体类型除定义变量、指针变量、数组等元素外，还可以定义函数，来返回结构体类型的值或指针值。这在用法上和其他类型的函数一样，需要在调用处定义相同基类型的变量来接收函数返回的值。

9.5 用指针处理链表

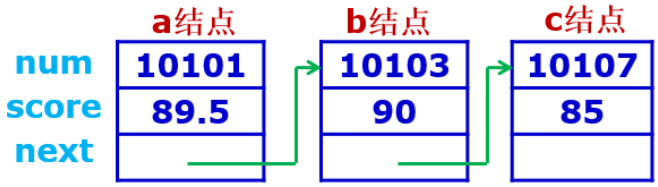
9.5.1 什么是链表

链表是一种常见的重要的数据结构，它是动态地进行存储分配的一种结构，链表必须利用指针变量才能实现。



9.5.2 建立简单的静态链表

【例 9.8】建立一个如图所示的简单链表，它由 3 个学生数据的结点组成，要求输出各结点中的数据。




```
#include "stdio.h"
#define NULL   0
struct student{
    long int    num;
    float       score;
    struct student    *next;
};
main()
{
    struct student    a,b,c,*p,*head;
    a.num=10101;a.score=90.5;
    b.num=10102;b.score=80.5;
    c.num=10103;c.score=70.5;
    head=&a;
    a.next=&b;
    b.next=&c;
    c.next=NULL;
```

```
        p=head;
        while(p!=NULL)
        {
            printf("%ld     %4.1f\n",p->num,p->score);
            p=p->next;
        }
    }
```

9.5.3 建立动态链表

所谓建立动态链表是指在程序执行过程中从无到有地建立起一个链表，即一个一个地开辟结点和输入各结点数据，并建立起前后相链的关系。

例 9.9 写一函数建立一个有 3 名学生数据的单向动态链表。

```
#include <stdio.h>
#include <stdlib.h>
#define LEN sizeof(struct student)
struct student *creat(void) ;
void print(struct student *head) ;
struct student *delete(struct student *head,long num);
struct student * insert(struct student *head,struct student *p0 );

struct student
{   long num;
    float score;
    struct student *next;
};
int n;      //n 为全局变量，存储结点个数

int main()
{   struct student *pt,*stu;
    long del_num;
    pt=creat();
    print(pt);
    printf("Please input the delete number:");
    scanf("%ld",&del_num);
    while(del_num!=0)
    {
        pt=delete(pt,del_num);
        printf("Please input the delete number:");
        scanf("%ld",&del_num);
    }
    print(pt);

    printf("\nPlease input the insert record:");
    stu=( struct student * )malloc(LEN);
    scanf("%ld %f",&stu->num,&stu->score);
```

```
    while(stu->num!=0)
     {
          pt=insert(pt,stu);
          printf("\nPlease input the insert record:");
          stu=( struct student * )malloc(LEN);
          scanf("%ld %f",&stu->num,&stu->score);
     }
     print(pt);
}

struct student *creat(void)                //定义创建链表函数，此函数返回一个指向链表头的指针
{   struct student *head,*p1,*p2;
    n=0;
    p1=p2=( struct student*) malloc(LEN);      //开辟第一个结点
    scanf("%ld %f",&p1->num,&p1->score);
    head=NULL;
    while(p1->num!=0)                          //判断新开辟的结点里 num 为 0，则停止添加结点
    {   n++;
        if(n==1) head=p1;
        else   p2->next=p1;
        p2=p1;
        p1=(struct student*)malloc(LEN);
        scanf("%ld %f",&p1->num,&p1->score);
     }
    p2->next=NULL;       return(head);
}

void print(struct student *head)
{     struct student *p;
      printf("\n  当前有%d 个记录！ \n",n);
      p=head;
      do
    {printf("%ld %5.1f\n",p->num,p->score);
      p=p->next;
    }while(p!=NULL);
}


struct student *delete(struct student *head,long num)
{
      struct student *p1,*p2;
      if(head==NULL) {printf("\n    List is null!\n");    goto end;  }
      p1=head;
      while((num!=p1->num)&&(p1->next!=NULL))
```

```
                                                    //p1 指向的不是要找的结点，并且后面还有结点
    {p2=p1; p1=p1->next;}   //p1、p2 分别后移一个结点
    if(num==p1->num)            //找到了
        {
            if(p1==head) head=p1->next ;   //若 p1 指向的是首结点，把第二个结点的地址赋予
head
            else p2->next=p1->next;          //否则把下一结点地址赋给前一结点地址
        n--;
        printf("Delete:%ld\n",num);
        }
    else
      printf("%ld not been found!\n",num);        //找不到该结点
    end:
    return (head);
}

struct student * insert(struct student *head,struct student *p0 )   //p0 指向要插入的节点
{
    struct student *p1,*p2;
    p1=head;                                          //p1 指向第一个结点
    if(head==NULL) {head=p0;p0->next=NULL;}   //原来的链表为空，则使 p0 指向的节点作为
头结点
    else
      {
        while((p0->num >p1->num)&&(p1->next!=NULL))
         {
            p2=p1;                    //p2 指向刚才 p1 指向的结点
            p1=p1->next ;             //p1 后移一个结点
         }
         if(p0->num<=p1->num)
              {if(head==p1) head=p0;       //插到原来第一个结点之前
               else p2->next=p0;           //插到 p2 指向的结点之后
               p0->next =p1;
              }
         else
            {p1->next =p0;p0->next=NULL;     }     //插到最后的结点之后
      }
    n++;                                              //完成结点插入后，结点数加 1
    return (head);
}
```

9.6 共同体

- 有时想用同一段内存单元存放不同类型的变量。

- 使几个不同的变量共享同一段内存的结构，称为 “共用体”类型的结构。
- “共用体”与“结构体”的定义形式相似，但它们的含义是不同的。
  - 结构体变量所占内存长度是各成员占的内存长度之和，每个成员分别占有其自己的内存单元。而共用体变量所占的内存长度等于最长的成员的长度。
  - 共同体类型的定义、共同体变量的声明，赋值与结构体方法一致。
- 只有先定义了共用体变量才能引用它，但应注意，不能引用共用体变量，而只能引用共用体变量中的成员。
- 例如，前面定义了 a,b,c 为共用体变量，下面的引用方式是正确的： a.i a.ch a.f

注意区别：共同体类型中各成员共用存储空间，所以各成员不能同时使用，否则出错。

【实例 9.7】共同体类型的定义、共同体变量的声明，赋值。

```
#include <stdio.h>
    union student{
        int num;
        char name[5];
        char sex;
        float score1;
        float score2;
    };
int main( )
{
    union student   stu1;
    stu1.num=300;
    stu1.name[0]='H';stu1.name[1]='e';stu1.name[2]='l';
    stu1.sex='F';
    stu1.score1=88.3;
    stu1.score2=77.1;
    printf("num:%-5dname:%-15ssex:%-3cscore1:%-8.1fscore2:%-8.1f\n",stu1.num,stu1.name,stu1.sex,stu1.score1,stu1.score2);
}
```

第 10 章 对文件的输入输出

函数名	调用形式	功能	返回值

fopen	**fp= fopen(“a1”,”r”)**	if ((fp=fopen(“file1”,’r〃))==NULL)
		{printf(“cannot open this file\n”);
		exit(0);
		}	

fclose	**fclose(fp)**	关闭文件。如果不关闭文件将会丢失数据。	
rewind	**rewind(fp)**	重新移动文件指针到文件头部	
fgetc	**fgetc(fp)**	从 fp 指向的文件读入一个字符	读成功，带回所读的字符，失败

			则返回文件结束标志 EOF(即-1)
fputc	**fputc(ch,fp)**	把字符 ch 写到文件指针变量 fp 所指向的文件中	写成功，返回值就是输出的字符；输出失败，则返回 EOF（即-1）
fgets	**fgets(str,n,fp)**	从 fp 指向的文件读入长度为(n-1)的字符串，存放到字符数组 str 中	读成功，返回地址 str，失败则返回 NULL)
fputs	**fputs(str,fp)**	str 所指向的字符串写到文件指针变量 fp 所指向的文件中	写成功，返回 0；否则返回非 0 值
fread	**fread(buffer ，size，count，fp);**	fread(p,sizeof(struct student),1,fp1);	按指定的块大小、块数量进行文件**块读写**
fwrite	**fwrite(buffer ，size，count，fp);**	fwrite(p,sizeof(struct student),1,fp1);	
fprintf	**fprintf**(fp,char *format,arg_list)	fprintf(fp,"%c %d/n",ch,num);	按指定的格式进行文件**块读写**
fscanf	**fscanf**(fp,char *format,arg_list)	fscanf(fp,"%c %d/n",&ch,&num);	
feof	**feof(fp)**	判断文件指针是否已经到文件尾，是则返回为 “真”	

10.1 C文件的有关基本知识

当文件按指定的工作方式打开以后，就可以执行对文件的读和写。下面按文件的性质分类进行操作。针对文本文件和二进制文件的不同性质，

- 对文本文件来说，可按字符读写或按字符串读写；打开方式为：r、w、a
- 对二进制文件来说，可进行成块的读写或格式化的读写。打开方式为：rb、wb、ab

10.2 读写字符

C 提供 fgetc 和 fputc 函数对文本文件进行字符的读写，其函数的原型存于 stdio.h 头文件中，格式为：

fgetc(fp) fgetc( )函数从输入流的当前位置返回一个字符，并将文件指针指示器移到下一个字符处，如果已到文件尾，函数返回 EOF，此时表示本次操作结束，若读写文件完成，则应关闭文件。

fputc(ch,fp) fputc（）函数完成将字符 ch 的值写入所指定的流文件的当前位置处，并将文件指针后移一位。fputc()函数的返回值是所写入字符的值，出错时返回 EOF。

【例 10.1】字符方式写文件、读文件

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

```
main()
{
    char ch;
    FILE *fp;   //定义文件指针
    if((fp=fopen("a1","w"))==NULL)
        {printf("Can't open file a1.");exit(0);}
        /*若文件正常打开，返回值为文件指针的位置；若文件不能打开，退出程序。*/
         while((ch=getchar())!='#')          fputc(ch,fp);
         fclose(fp);   //关闭文件
    if((fp=fopen("a1","r"))==NULL)
        {printf("Can't open file a1.");exit(0);}
    while((ch=fgetc(fp))!=EOF)    putchar(ch);
    fclose(fp);
}
```

10.3 读写字符串

C 提供读写字符串的函数原型在 stdio.h 头文件中，其函数形式为：

Char *fgets(char *str,int num,FILE *stream)

fgets() 函数从流文件 stream 中读取至多 num-1 个字符，并把它们放入 str 指向的字符数组中。读取字符直到遇见回车符或 EOF（文件结束符）为止，或读入了所限定的字符数。

int fputs(char *str,FILE *stream)

fputs( )函数将 str 指向的字符串写入流文件。操作成功时，函数返回 0 值，失败返回非零值。

【例 10.2】字符串方式写文件、读文件

```
#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
main()
{
    char str[100];
    FILE *fp;
    if((fp=fopen("text.txt","w"))==NULL)
        {printf("Can't open.");exit(0);}
    while((strlen(gets(str)))!=0)
    {
        fputs(str,fp);
        fputs("\n",fp);
    }
    fclose(fp);
    if((fp=fopen("text.txt","r"))==NULL)
        {printf("Can't open.");exit(0);}
    while((strlen(fgets(str,100,fp)))>0&&!feof(fp))
    {   printf("%s",str);      }
    fclose(fp);
```

}

10.4 格式化的读写

前面的程序设计中，我们介绍过利用 scanf( )和 printf( )函数从键盘格式化输入及在显示器上进行格式化输出。对文件的格式化读写就是在上述函数的前面加一个字母 f 成为 fscanf( )和 fprintf( )。其函数调用方式：

int **fscanf**(FILE *stream,char *format,arg_list)

int **fprintf**(FILE *stream,char *format,arg_list)

其中，stream 为流文件指针，其余两个参数与 scanf( )和 printf( )用法完全相同。

10.5 成块读写

前面介绍的几种读写文件的方法，对其复杂的数据类型无法以整体形式向文件写入或从文件读出。C 语言提供成块的读写方式来操作文件，使其数组或结构体等类型可以进行一次性读写。成块读写文件函数的调用形式为：

fread(void *buf,int size,int count,FILE *stream)

fwrite(void *buf,int size,int count,FILE *stream)

fread（）函数从 stream 指向的流文件读取 count （字段数）个字段，每个字段为 size(字段长度)个字符长，并把它们放到 buf（缓冲区）指向的字符数组中。fread（）函数返回实际已读取的字段数。若函数调用时要求读取的字段数超过文件存放的字段数，则出错或已到文件尾，实际在操作时应注意检测。

fwrite( )函数从 buf(缓冲区)指向的字符数组中，把 count(字段数)个字段写到 stream 所指向的流中，每个字段为 size 个字符长，函数操作成功时返回所写字段数。

关于成块的文件读写，在创建文件时只能以二进制文件格式创建。

【例 10.3】向磁盘写入格式化数据，再从该文件读出显示到屏幕。

```
#include "stdio.h"
#include "stdlib.h"
main( )
{
    FILE *fp1;
    int i;
    struct stu{ /*定义结构体*/
        char name[15];
        char num[6];
        float score[2];
    }student;
    if((fp1=fopen("test.txt","wb"))==NULL)     /*以二进制只写方式打开文件*/
    {
        printf("cannot open file");
        exit(0);
    }
    printf("input data:\n");
    for( i=0;i<2;i++)
    {
```

```
            scanf("%s %s %f %f",student.name,student.num,
                                        &student.score[0],&student.score[1]); /* 输入一记录*/
            fwrite(&student,sizeof(student),1,fp1); /* 成块写入文件*/
        }
        fclose(fp1);

      if((fp1=fopen("test.txt","rb"))==NULL)       /*重新以二进制只写打开文件*/
        {
            printf("cannot open file");
            exit(0);
        }
        printf("output from file:\n");
        for (i=0;i<2;i++)
        {
        fread(&student,sizeof(student),1,fp1);     /* 从文件成块读*/
        printf("%s %s %7.2f %7.2f\n",student.name,student.num,
                                        student.score[0],student.score[1]); /* 显示到屏幕*/
        }
        fclose(fp1);
}
```

第 11 章 预处理命令

预处理命令作用：对源程序编译之前做一些处理,生成扩展 C 源程序。

种类

- 宏定义　　#define
- 文件包含　　#include
- 条件编译　　#if--#else--#endif 等　　//不考

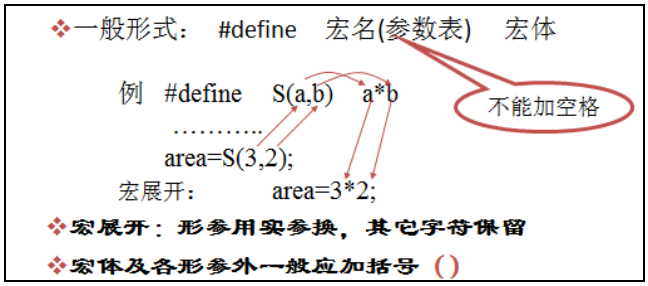
格式：

- “#”开头
- 占单独书写行
- 语句尾不加分号

11.1 不带参数宏定义

❖一般形式：　#define　宏名　[宏体]
❖功能:用指定标识符(宏名)代替字符序列(宏体)
❖定义位置:任意(一般在函数外面)
❖作用域:从定义命令到文件结束
❖#undef可终止宏名作用域
　格式：　#undef　宏名
❖宏展开：预编译时,用宏体替换宏名---不作语法检查
❖引号中的内容与宏名相同也不置换
❖宏定义可嵌套，不能递归
❖宏定义中使用必要的括号（）

11.2 带参数宏定义




```
#include "stdio.h"
#define        PI          3.1415926
#define        OUT         printf("Hello,World");
#define    S(a,b)    a*b
#define    D(a,b)    (a)*(b)
main( )
{
printf("%10.2f\n",PI);
printf("%d\n",S(3+2,2+1));
printf("%d\n",D(3+2,2+1));
OUT
}
```

11.3 文件包含

功能：一个源文件可将另一个源文件的内容全部包含进来

一般形式： #include “文件名”

或 #include <文件名>

处理过程：预编译时,用被包含文件的内容取代该预处理命令，再对“包含”后的文件作一个源文件编译。

第 12 章 位运算

12.1 位运算符和位运算

C 语言提供的位运算符有：

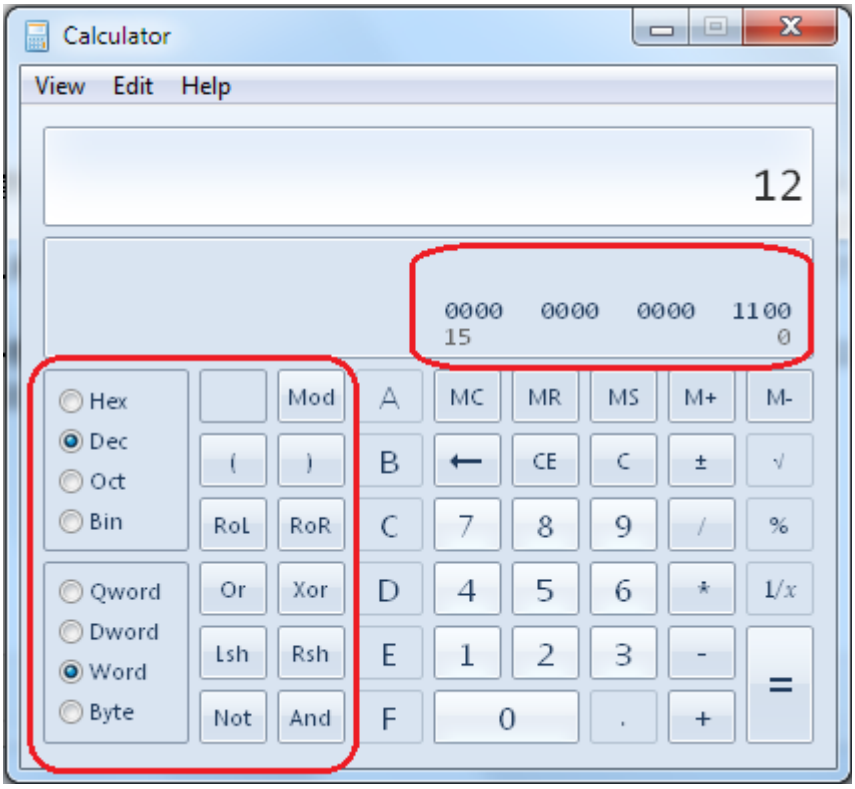
运算符	含义	运算符	含义
&	按位与	~	取反
|	按位或	<<	左移
^	按位异或	>>	右移

说明：

(1)位运算符中除~以外，均为二目（元）运算符，即要求两侧各有一个运算量。

(2)运算量只能是整型或字符型的数据，不能为实型数据。

可以使用程序员计算器来实现位运算：



这是我们程序员的模式，你可以使用不同的进制来表示数，也可以限定数据的字节长度，而且每个数都在下方给出了其二进制的值，非常贴心。所谓程序员计算器，除了这些，还包括各种位运算，下面一一介绍：

1.And, Or, Not, Xor： 最基本的与或非和异或操作，不作解释。

2.Lsh, Rsh：全称是 Left Shift 和 Right Shift，也就是左移和右移操作，你需要输入你要移动的位数（不能大于最大位数）

12.2 位运算举例

【例 12.1】 取一个整数 a 从右端开始的 4～7 位。

① 先使a右移4位：a >> 4

目的是使要取出的那几位移到最右端

(a) (b)

未右移时的情况 右移4位后的情况

② 设置一个低 4 位全为 1,其余全为 0 的数。

~ (~ 0 << 4)

③ 将上面①、②进行&运算。

(a >> 4) & ~ (~ 0 << 4)

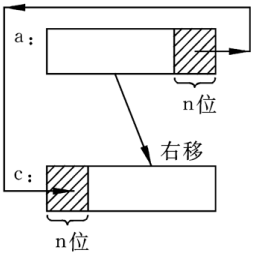
程序如下：

```
#include <stdio.h>
main()
  {  unsigned a,b,c,d;
       scanf(“%o”,&a);
       b=a>>4;
       c=～(～0<<4);
      d=b&c;
      printf(“%o,%d\n%o,%d\n”,a,a,d,d);
}
```

```
运行情况如下：３３１．（输入）
３３１，　217　（ａ的值）
１５，　13　　（ｄ的值）
输入ａ的值为八进制数331，
其二进制形式为11011001
经运算最后得到的d为00001101
即八进制数１５，十进制数13。
```

【例 12.2】 循环移位。要求将 a 进行右循环移位

将 a 右循环移 n 位，即将 a 中原来左面（１６－ｎ）位右移 n 位，原来右端 n 位移到最左面 n 位。



步骤：

① 将 a 的右端 n 位先放到 b 中的高 n 位中，实现语句：ｂ＝ａ＜＜（１６－ｎ）；

② 将 a 右移 n 位，其左面高位 n 位补 0，实现语句：ｃ＝ａ＞＞ｎ；

③ 将 c 与 b 进行按位或运算，即 ｃ＝ｃ|ｂ；

程序如下：

```
#include <stdio.h>
main()
{       unsigned a,b,c;
        int n;
        scanf("a=%o,n=%d",&a,&n);
        b=a<<(16-n);
        c=a>>n;
        c=c|b;
        printf("%o\n%o",a,c);
}
```