Maynard Kong

Lenguaje de Programación

C

PONTIFICIA UNIVERSIDAD CATOLICA DEL PERU
FONDO EDITORIAL 1988

Maynard Kong. En 1964 ingresó a la Facultad de Ciencias Físicas y Matemáticas de la Universidad Nacional de Ingeniería. Egresó en 1968 y desde 1969 se ha desempeñado como profesor del Departamento de Ciencias de la Universidad Católica en cursos de Matemáticas de niveles y especialidades variados. Obtuvo el grado de doctor (PhD) en la Universidad de Chicago (Estados Unidos de América) en 1976. Fue profesor visitante en la Universidad de Stuttgart (República Federal de Alemania) en 1979, y al mismo tiempo becario de la Fundación von Humboldt en un programa de posdoctorado, y posteriormente, también en Venezuela, durante 4 años.

Ha publicado importantes trabajos de investigación y varios textos de consulta universitaria, entre los que se pueden mencionar: Teoría de Conjuntos *(coautor con César Carranza),* Cálculo Diferencial, Cálculo Integral, BASIC y Lenguaje de Programación Pascal.

Ha participado en numerosos eventos de Matemáticas, promoción de las Ciencias Básicas e Informática tanto en el país como en el extranjero.

LENGUAJE DE PROGRAMACION "C"

Maynard Kong

Lenguaje de Programación

C

PONTIFICIA UNIVERSIDAD CATOLICA DEL PERU
FONDO EDITORIAL 1988

Primera edición: marzo de 1988

Cubierta: Carlos González R.

LENGUAJE DE PROGRAMACION "C"

ISBN 84-89292-86-8

Impreso en el Perú - Printed in Peru

CONTENIDO

CAPITULO 9 APLICACIONES

PROLOGO

En la presente obra se desarrollan los conceptos fundamentales del lenguaje de programación C, que ha demostrado ser sumamente útil no sólo en el desarrollo de programas utilitarios (sistemas operativos, administradores de base de datos, etc.) sino también en las aplicaciones de propósito general.

La exposición incluye numerosos ejemplos de programas completos y secciones de ejercicios a fin de que el lector pueda comprobar sus conocimientos.

A excepción del capítulo 9, todos los programas expuestos pueden correr o ejecutarse con la mayoría de los compiladores o intérpretes de C.

Diciembre de 1987

Maynard Kong

CAP. 1

INTRODUCCION

1 CARACTERISTICAS

El lenguaje de programación C presenta las siguientes características:

a) Es de propósito general;

b) es un lenguaje de alto nivel como Pascal, BASIC, etc.;

c) es un lenguaje estructurado, pues contiene las tres estructuras básicas de formación de instrucciones: composición, selección y lazo o ciclo con entrada única;

d) se le considera lenguaje de nivel medio, esto es, entre lenguaje ensamblador y de alto nivel, pues permite el manejo de datos a nivel de bits, bytes y direcciones;

e) programación modular: se pueden escribir programas independientes (módulos) constituyendo librerías de programas para ser usados posteriormente;

f) transportabilidad: los programas escritos en este lenguaje pueden ser ejecutados en distintos computadores.

El lenguaje C se emplea para escribir sistemas de operación, compiladores, programas ensambladores, programas de base de datos, etc.

2 ESTRUCTURA DE UN PROGRAMA EN C

Un programa en C se compone de uno o más bloques de instrucciones o sentencias denominados **funciones**:

```
función1(...)
función2(...)
...

funciónk(...)
```

Cada función se designa con un nombre o identificador seguido por un par de paréntesis, entre los cuales pueden ir otros datos como se verá más adelante.

3 IDENTIFICADORES O NOMBRES EN C

Un identificador o nombre en C se compone de uno o varios caracteres y se forma según las siguientes reglas:

1) el primer carácter debe ser una letra o el signo del subrayado "_";

2) los caracteres restantes pueden ser letras, el signo del subrayado o dígitos;

3) debe ser distinto de las palabras reservadas por el lenguaje, por ejemplo, es incorrecto emplear los nombres: int, float, if, else, etc., para designar identificadores.

Los identificadores se utilizan para designar funciones, variables, constantes, tipos de datos, etc.

EJEMPLOS DE IDENTIFICADORES

1) límite
2) valor1
3) Suma_total
4) _mensaje
5) A
6) X3
7) #define máximo 100

la directiva **#define** permite definir al identificador máximo como una constante numérica.

4 LA FUNCION PRINCIPAL **main()**

Todo programa en lenguaje C debe contener al menos la función main(). Esta función gobierna la ejecución del programa, es decir el programa simplemente ejecuta todas las instrucciones que forman el bloque de la función main() según el orden en que aparecen, algunas de las cuales pueden ser llamadas de ejecución de otras funciones.

5 PROGRAMA 1

```
/* Primer programa en C.
   Se muestra la función printf y algunas operaciones*/

main()
{
   printf("C es un lenguaje estructurado\n");
   printf("La suma y el cociente entero es 23 y 5: ");
   printf("%d",23+5);
   printf("%6d",23/5);
}
```

EJECUCION

```
C es un lenguaje estructurado
La suma y el cociente entero de 23 y 5 : 28     4
```

EXPLICACION

1) Los símbolos /* y */ sirven para encerrar comentarios dentro de un programa. Todo el texto comprendido entre ellos es ignorado por el programa traductor, esto es, no se convierte en instrucciones ejecutables.

2) Este programa consiste sólo de la función main() cuyo bloque está delimitado por los símbolos { y }

3) Se utiliza la función de salida printf para imprimir datos en el dispositivo de salida estándar (por defecto, la pantalla).

4) Entre los paréntesis que siguen a cada printf se incluyen los datos a imprimir. El primer dato siempre es una cadena de caracteres encerrada entre comillas. Los caracteres de esta cadena se imprimen tal como aparecen, exceptuando aquellos que están precedidos por el signo de porcentaje % que se utiliza para especificar el tipo de impresión de los datos restantes. Así, %d significa que el dato siguiente se ha de imprimir como número entero, y %6d, que el dato siguiente se imprimirá como entero en una zona o campo de 6 columnas.

5) El símbolo \n se interpreta como un sólo carácter en C y representa cambio de línea o línea nueva.

6) El símbolo ; se utiliza como finalizador de proposición.

7) En lugar de las dos últimas instrucciones printf se puede utilizar:

```
printf("%d%6d",23+5,23/5);
```

6 INDICACIONES GENERALES SOBRE PROGRAMAS EN C

Hemos indicado anteriormente que un programa en C es una colección de una o varias funciones, una de las cuales necesariamente debe ser la función main().

Ahora precisamos algunas indicaciones básicas que deben tenerse en cuenta en los programas en C:

1) En cualquier lugar del programa se pueden incluir comentarios empleando los símbolos /* y */:

```
/*  comentarios ...   */
```

2) el símbolo de punto y coma (;) se utiliza para finalizar una proposición

3) cualquier grupo de instrucciones puede formar un bloque y ser tratado como una instrucción simple, para lo cual basta encerrarlo entre los símbolos de llaves { }:

```
{   grupo de instrucciones }
```

4) toda función forma un bloque de instrucciones

```
{   el cuerpo o bloque de la función }
```

5) se pueden anidar o encajar bloques de instrucciones:

```
{      /*  comienzo de bloque  */
  ...
    {   /*  comienzo de sub-bloque */
        ...
        ...
    }   /*  fin de sub-bloque  */
  ...

}       /*  fin de bloque     */
```

6) el procesamiento o ejecución de un programa es normalmente secuencial (esto es, las instrucciones se ejecutan según el orden en que aparecen) salvo que alguna instrucción de transferencia cambie este curso.

7 TIPOS DE DATOS SIMPLES Y CADENAS

Mencionamos algunos tipos de datos simples: char, int, float, double.

Los datos de un tipo dado pueden ser **constantes** o **variables**.

a) Tipo char

Los datos constantes de tipo char o carácter comprenden a los valores enteros entre 0 y 255, inclusive. Estos números representan los códigos en algún sistema de codificación de caracteres, por ejemplo en el sistema ASCII.

También se pueden representar los caracteres normales (letras, dígitos, y algunos símbolos visibles) encerrándolos con apóstrofos o comillas simples. Por ejemplo:

espacio en blanco: ' ' (valor ASCII 32)

caracteres de dígitos: '0','1',...,'9' (valores ASCII: 48...57)

letras mayúsculas: 'A','B',...,'Z' (valores ASCII: 65...91)

letras minúsculas: 'a','b',...,'z' (valores ASCII: 97...123

Estos caracteres se ordenan por sus valores. Así el carácter de blanco es menor que cualquier carácter de dígito; los caracteres de dígitos son menores que las letras mayúsculas, y éstas son menores que las minúsculas.

Debe tenerse presente que un dato de tipo char siempre consiste de un sólo carácter. Para representar caracteres especiales se emplean los símbolos:

carácter de línea nueva:	'\n'	
carácter nulo:	'\0'	(valor cero)
carácter de tabulador:	'\t'	
carácter de apóstrofo:	'\''	
carácter de diagonal invertida:	'\\'	

b) **Tipo int**

Los datos constantes de tipo int o entero comprenden a los números enteros positivos, negativos o cero, entre dos valores límites, por ejemplo entre -32768 y 32767 , inclusive.

Ejemplos: 12, -450, 0, 25678.

c) **Tipo float**

Los datos constantes de tipo float o números de punto flotante son los números decimales positivos, negativos o cero, entre ciertos valores límites.

Ejemplos: 12.0, -450.0, 0.0, 18.789
45 e2 (igual a 45 por 10 al exponente 2 =4500)
-2.5 e-1 (igual a -2.5 por 10 al exponente -1 = -0.25)

d) **Tipo double**

Los datos constantes de tipo double o de precisión doble comprenden a los números decimales positivos, negativos o cero, que son representados con un número mayor de dígitos que los números de punto flotante de manera que los cálculos que se realizan con ellos tienen un mayor grado de precisión.

NOTA

Los tipos mencionados son compatibles en forma ascendente, esto significa que cada tipo de datos se considera incluido en el siguiente tipo. Así, char es parte de int, int es parte de float y float es parte de double.

e) **Cadenas de caracteres**

Una cadena de caracteres constante es una secuencia de caracteres encerrada entre comillas.

Ejemplos:
```
"EL RESULTADO ES "
"Ingrese valor "
"NOMBRE  SALDO\n"
```

(la última cadena contiene el carácter de cambio de línea '\n')

f) **Constantes dadas por nombres**

Se puede usar la directiva **#define** al comienzo de un programa para nombrar valores constantes:

```
#define  Límite  100
#define  pi  3.1415
#define  mensaje  "TABLA DE VALORES\n"
#define  formato1  "%d , %8.3f"
```

Entonces los nombres Límite, pi, mensaje y formato1 serán reemplazados, cada vez que aparezcan en el resto del programa, por sus correspondientes expresiones: 100, 3.1415, etc.

8 OPERACIONES ARITMETICAS

Con los datos constantes o variables de los tipos mencionados, denominados en general datos numéricos, se pueden realizar las siguientes operaciones, en orden de jerarquía descendente:

OPERACION	SIMBOLO	EXPRESION	RESULTADO
multiplicación	*	X*Y	producto de X por Y
división	/	X/Y	cociente de X entre Y
módulo	%	X%Y	resto de la división entre los enteros X e Y
adición	+	X+Y	suma de X e Y
sustracción	-	X-Y	diferencia de X menos Y

Con excepción de la operación módulo (%) que se aplica sólo a datos enteros, todas las otras operaciones dan resultados

a) del mismo tipo que los operandos si ambos son del mismo tipo,

o b) del tipo de mayor rango si los operadores son de tipos distintos.

Por ejemplo, si una expresión contiene operaciones con datos de tipos char, float y double, entonces el resultado o valor de la expresión será de tipo double.

Se pueden emplear paréntesis para agrupar datos y especificar una manera de cálculo. Al evaluarse una expresión, se asume el siguiente orden de precedencia, de mayor a menor:

1) paréntesis 2) *, /, % y 3) +, -

Las operaciones que tienen igual precedencia se evalúan de izquierda a derecha.

EJEMPLOS

1)	15/2 + 3.0*2	es igual a	7 + 6.0 = 13.0	(punto flotante)
2)	15/2.0 - 3.0*2	es igual a	7.5 + 6.0 = 13.5	
3)	(3 - 4.7)*5	es igual a	-1.7*5 = -8.5	
4)	1 + 45%12	es igual a	1 + 3 = 4	(entero)

9 VARIABLES

Una variable es un identificador o nombre que sirve para almacenar valores de un tipo de datos y se declara en la forma:

```
T   nv;
```

en donde
- nv es el nombre de la variable
- T es el tipo de datos que la variable ha de asumir.

Se pueden declarar varias variables del mismo tipo separándolas por comas:

```
T   nv1,  nv2, ..., nvk ;
```

JEMPLOS

```
a)  char  letra;          /*  letra es una variable de tipo char */

b)  int   n, i, j;        /*  n,i, j son tres variables enteras  */

c)  float  Raiz1, Raiz2;  /* Raiz1 y Raiz2 son variables de tipo
                                                    float   */

d)  char  mensaje[ 25] ;      /* mensaje es una variable de cadena
                                 (o arreglo) de 25 caracteres   */
```

Las variables pueden ser locales o globales, según se declaren dentro de una función o fuera de las funciones.

Las variables locales se declaran dentro de una función y al comenzar cualquier bloque, esto es, a continuación de { , y tienen validez sólo en el interior de dicho bloque.

ASIGNACION DE VALORES A VARIABLES

a) **Asignación directa en el programa con el signo de igualdad**

variable = valor asignado

en donde valor asignado puede ser una constante, una variable o una expresión.

En este caso, si es necesario, el valor asignado se convierte al tipo de dato de la variable.

EJEMPLOS

1) `letra=A;  n=2*3;  i=n+1;  i=i-1;`

2)
```
Raiz1= 4 + cuadrado(3);
        /* se asume que cuadrado(X) es una función que
           devuelve el cuadrado del argumento X      */
```

3) Si m es una variable de tipo int, la asignación:

`m=4*2.2;`

entrega a m el valor 8 (la parte entera de 8.8)

b) **Durante corrida mediante una función de entrada de datos**, por ejemplo, con la función scanf en la forma:

scanf("especificación de tipo", dirección de variable)

La función de entrada de datos scanf permite leer datos y asignarlos a las variables desde el dispositivo de entrada de datos preestablecido (por defecto, el teclado).

Las especificaciones de tipo son:

%c	para datos de tipo char
%d	para datos de tipo int
%f	para datos de tipo float
%s	para datos de tipo cadena de caracteres

Y la dirección de una variable de tipo simple (char, int, float, double) se obtiene anteponiendo el operador de dirección & al nombre de la variable. En el caso de una cadena variable su nombre representa una dirección de modo que no se requiere el operador &.

EJEMPLOS

```
a) scanf("%c",&letra);          /* lee un carácter desde el dispositivo de entrada y lo asigna a la variable
                                   letra */
b) scanf("%d",&n);              /* lee entero y lo asigna a la variable n */
c) scanf("%d %f",&i, &Raiz1);   /* lee dos datos, entero y flotante,
                                   y los asigna a la variable i,
                                   Raiz1, respectivamente */
d) scanf("%s", mensaje);        /* lee una palabra, comprendida entre caracteres de blancos, en la cadena
                                   variable mensaje */
```

10 PROGRAMA: AREA DE UN CIRCULO

El siguiente programa lee durante ejecución el radio de un círculo en la variable Radio e imprime el valor del área del mismo dado por la variable Area.

```
#define  pi 3.14159
main()
{ float  Radio, Area;
       printf("Ingrese el valor del radio ");
       scanf("%f",&Radio);
       Area= pi*Radio*Radio;
       printf("área = %8.3f",Area);
}
```

EJECUCION

```
Ingrese valor del radio 5
área = 78.540
```

11 PROGRAMA: MAXIMO DE DOS NUMEROS

El siguiente programa calcula el máximo de dos números de punto flotante, Los números son leidos en las variables a y b durante corrida.

```
main()
{ float a,b,max;
      printf("valores de a y b : ");
      scanf("%f %f",&a,&b);
      if (a>b)  max=a;
      else      max=b;
      printf("máximo = %f", max);
}
```

EJECUCION

```
valores de a y b : 18.70  45
máximo = 45
```

EXPLICACION

Empleamos una variable max de tipo float para almacenar el valor máximo.

Hemos usado la instrucción if ... else :

```
if (a>b)    max=a;
else        max=b;
```

por la cual, si la condición (a>b) es verdadera se ejecuta la proposición: max=a;

y en caso de ser falsa, se ejecuta la proposición: max=b;

Así, max resulta con el valor máximo de los valores contenidos en las variables a y b.

12 PROGRAMA: SALDO DE UNA CUENTA

El programa que se muestra a continuación calcula el saldo de una cuenta. Se inicia solicitando el saldo anterior y se ejecuta mientras las cantidades ingresadas (ingresos o egresos) sean distintos de cero. Al final se imprime el saldo actual.

```
main()
{ float   Saldo,  Cantidad;
  printf("Saldo anterior : ");
  scanf("%f",&Saldo);
  printf("Ingrese cantidades y termine con 0\n");
  Cantidad=1;
  while (Cantidad )
   {    scanf("%f",&Cantidad);
        Saldo=Saldo+Cantidad;
    }
  printf("Saldo final = %8.3f\n",Saldo);
}
```

EJECUCION

```
Saldo anterior : 1200.50
Ingrese cantidades y termine con 0
400  -220  270.50  0
Saldo final = 1651.000
```

EXPLICACION

Hemos utilizado la instrucción while:

```
while (Cantidad)
   {  scanf("%f",&Cantidad);
      Saldo=Saldo+Cantidad;
   }
```

para repetir la proposición que le sigue, en este caso un bloque.

La instrucción while comprueba si el valor de Cantidad es distinto de cero o no. Si es cero termina la instrucción while y se prosigue con el resto del programa. Si es distinto de cero se procesa la proposición que le sigue: se lee en la variable Cantidad y se actualiza el saldo, y luego se vuelve a comprobar, y así sucesivamente.

Notemos que Cantidad ha sido previamente igualada a 1 a fin de que la proposición de while se procese la primera vez. Este valor se pierde con la primera lectura.

13 PROGRAMA: TABLA DE CUADRADOS

El programa siguiente imprime los valores de los enteros 1,2, ..., 24 y sus respectivos cuadrados.

```
main()
{ int n;
     for (n=1;n<=24;n=n+1)   printf("%6d  %6d \n",n,n*n);
}
```

EJECUCION

```
     1       1
     2       4
     3       9
     .       .
     .       .
     .       .
    24     476
```

EXPLICACION

Utilizamos la instrucción for:

```
for (n=1; n<=24;n=n+1)   printf("%6d  %6d  \n", n, n*n);
```

cuyo efecto es equivalente al conjunto de instrucciones:

```
n=1;
while (n<=25)
     {   printf("%6d  %6d \n",n,n*n);
         n=n+1;
     }
```

es decir:

a) Primero se procesa n=1.

b) Se comprueba si n es menor o igual que 25 mediante la expresión n <= 25 (esta expresión toma el valor 1 si la condición es verdadera, o el valor 0 si la condición es falsa) :

- Si la condición es falsa termina la instrucción for y se prosigue con el resto del programa;
- si la condición es verdadera se procesa la proposición afectada por for (en este caso printf), luego se incrementa el valor de n: n=n+1, esto es, se ejecuta la expresión dada por el tercer argumento de for, y a continuación se repite b).

14 OPERACIONES LOGICAS

En el lenguaje C un valor distinto de 0 puede ser interpretado como verdadero, y el valor cero se considera falso.

Sean X e Y dos expresiones que toman valores numéricos.

Tenemos las siguientes operaciones lógicas:

OPERACION	SIMBOLO	EXPRESION	DESCRIPCION
negación	!	!X	devuelve 1 si X es cero, y el valor 0 de otra manera
y	&&	X && Y	devuelve el valor 1 si X e Y son distintos de cero, y el valor 0 en caso contrario
o	||	X || Y	devuelve el valor 1 si X ó Y es distinto de cero, y el valor 0 de otra manera.

15 RELACIONES DE COMPARACION

La comparación entre datos numéricos es una expresión que toma uno de los valores 1 o 0, según estos datos hagan verdadera o falsa la comparación, respectivamente.

Sean X, Y dos expresiones que toman valores numéricos.
Las relaciones de comparación son:

RELACION	SIMBOLO	EXPRESION	DESCRIPCION
igualdad	==	X == Y	toma el valor 1 si X e Y son iguales, y el valor 0 de otra manera
desigualdad	!=	X != Y	toma el valor 1 si X e Y son distintos, y el valor 0 de otra manera
menor	<	X < Y	toma el valor 1 si X es menor que Y, y el valor 0 en caso contrario
mayor	>	X > Y	toma el valor 1 si X es mayor que Y, y el valor 0 de otra manera
menor o igual	<=	X <= Y	toma el valor 1 si X es menor o igual que Y, y el valor 0 de otra manera
mayor o igual	>=	X >= Y	toma el valor 1 si X es mayor o igual que Y, y el valor 0 en caso contrario

EJEMPLOS

1) Si X vale 3 entonces el valor de (X >2) && (2*X==6) es 1 o verdadero.

2) La relación (Y=='A') || (Y=='E') será verdadera (valor 1) si Y es 'A' o 'B'.

16 EJERCICIOS

1. Escriba un programa que imprima su nombre y la fecha actual en dos líneas consecutivas.

2. Escriba un programa que lea tres números enteros durante corrida e imprima el mayor y el menor.

3. Escriba un programa que lea un ángulo en grados sexagesimales y lo convierta a radianes.
   INDICACION: Use el factor de conversión 3.1415/180.

4. Escriba un programa para calcular el monto total de una lista de N productos: Se lee el número N de productos, y para cada producto el número de unidades y el precio por unidad.

5. Escriba un programa que lea un número de segundos D y lo convierta a H horas, M minutos y S segundos.

6. Escriba un programa que imprima una tabla con los números 1.0, 1.5, 2.0, ..., 9.5, 10.0 y sus respectivos cubos.

7. Escriba un programa que lea un entero n y calcule la suma

   $$1 + 2 + \ldots + n.$$

8. Escriba un programa que lea un entero n y determine si es primo.

9. Escriba un programa que halle el máximo común divisor de dos enteros a y b.

10. Escriba un programa que lea un número x de tipo float y calcule su valor redondeado al entero n más cercano.

CAP. 2

INSTRUCCIONES DE CONTROL

17 INSTRUCCIONES DE PROCESAMIENTO CONTROLADO

El lenguaje C proporciona construcciones que permiten la ejecución de instrucciones en las siguientes formas:

a) selectivas : if, switch, operador condicional ?:

b) repetitivas : while, do

c) transferencia: goto, break, continue

17.1 La instrucción **if ... else** se expresa así:

```
if (expresión) proposición1;
          else proposición2;
```

Efecto: Si la expresión toma un valor distinto de cero (verdadero) se ejecuta la proposición1; y si la expresión tiene el valor 0 se ejecuta la proposición2. A continuación se prosigue con el resto del programa.

Las proposiciones involucradas pueden ser simples o compuestas (bloques).

Para un ejemplo véase el programa 11.

También se dispone de la forma if:

```
if (expresión) proposición;
```

por la cual se ejecuta la proposición afectada sólo si la expresión toma un valor no nulo, y después se continúa con las otras instrucciones del programa.

17.2 La instrucción **while** se usa en la forma:

while (expresión) proposición;

y su efecto es el siguiente:

1) se comprueba si el valor de la expresión es cero
2) si es cero se prosigue con el resto del programa
3) si es distinto de cero se ejecuta la proposición y luego se repite el proceso a partir del paso 1).

EJEMPLO: Véase el programa 12.

17.3 La instrucción **do ... while** tiene la siguiente forma:

do proposición;
while (expresión);

y su efecto es el siguiente:

1) se ejecuta la proposición
2) se comprueba si la expresión vale cero
3) si es cero se prosigue con el resto del programa
4) si no es cero se repite el proceso a partir del paso 1).

NOTA: Con la instrucción while la proposición se ejecuta cero o más veces y el control o comprobación siempre se realiza antes de ejecutar la proposición. En cambio con la instrucción do la proposición se ejecuta una o más veces y el control se lleva a cabo después de ejecutar la proposición.

EJEMPLO: El siguiente programa utiliza una instrucción do para calcular el valor del factorial $f=1 \times 2 \times \ldots \times n$ de un número entero n leído durante corrida.

```
main()
{ int n;  float f;
  printf("ingrese entero positivo n " ) ;
  scanf("%d",&n) ;
  f=1;
  do { f=f*n; n=n-1; }  while(n >0) ;
  printf("factorial = %8.0f\n",f) ;
}
```

17.4 La instrucción **for** se utiliza así:

```
for (expr1; expr2; expr3) proposición;
```

y tiene el siguiente efecto:

0) se ejecuta expr1; (sólo una vez)
1) se comprueba si expr2 es cero
2) si es cero se prosigue con el resto del programa
3) si es distinto de cero:
   - se ejecuta la proposición afectada,
   - se ejecuta expr3;
   - y se repite el proceso a partir del paso 1).

Nótese que expr3 se ejecuta después de la proposición.

Usualmente expr1 se refiere a una inicialización de valores, expr2 se usa como una relación de comparación y expr3 para reinicializar valores.

Se pueden omitir algunas de las expresiones pero no los puntos y comas. Si se omite expr2 se asume que tiene un valor no nulo (verdadero).

EJEMPLO: Véase el programa 13.

17.5 La instrucción **switch** se utiliza en la forma:

```
switch (expresión)
  { case c1 : prop1;
    case c2 : prop2;
    ...
    case ck : propk;

    default : prop;
  }
```

en donde expresión debe tener un valor entero (int) y c1, c2, ... , ck son constantes enteras distintas precedidas por la palabra case.

EFECTO

Si el valor de la expresión coincide con una de las constantes se transfiere la ejecución del programa al lugar de esa constante, esto es, se continúa el procesamiento a partir de la correspondiente proposición - Las constantes sirven como etiquetas de salto o transferencia -.

La palabra default puede ser usada en forma opcional para tratar situaciones en las que la expresión no coincide con ninguna de las constantes, en cuyo caso se hace el salto al lugar etiquetado con default.

Si no se emplea default y la expresión no coincide con ninguna constante, simplemente se prosigue con el resto del programa.

EJEMPLO

El siguiente programa lee un carácter y lo imprime sólo si es una vocal mayúscula.

```
main()
{  char letra;
   printf("ingrese letra ");
   scanf("%c",&letra);
   switch  (letra)
    { case 'A':
      case 'E':
      case 'I':
      case 'O':
      case 'U': printf "%c\n",letra);
    }  /*  fin de proposición afectada por switch */
}
```

Nótese que si ingresa una vocal mayúscula para la variable letra, por ejemplo E, el programa se continuará ejecutando desde el lugar case'E': ; y por lo tanto se imprimirá el valor de la variable.

17.6 La instrucción

break;

finaliza la ejecución del while, do, for o switch más interno que la contenga y se prosigue con el resto del programa.

EJEMPLO El siguiente programa lee un número entero entre 1 y 7 e imprime el nombre del día de semana correspondiente domingo, ..., sábado.

```
main()
{  int  nd;
   printf("ingrese número de día (1-7) ");
   scanf("%d",&nd);
   switch(nd)
   {  case 1: printf("domingo\n");   break;
      case 2: printf("lunes\n");     break;
      case 3: printf("martes\n");    break;
      case 4: printf("miércoles\n"); break;
      case 5: printf("jueves\n");    break;
      case 6: printf("viernes\n");   break;
      case 7: printf("sábado\n");    break;
      default: printf("número incorrecto\n");
   }
   printf("fin de programa\n");

}
```

Obsérvese que a causa de break sólo se procesa a lo sumo uno de los casos considerados.

17.7 La instrucción

continue;

permite transferir la ejecución a la parte final de la proposición afectada por el ciclo while, do o for más interno que la contenga.

17.8 La instrucción

goto etiqueta;

en donde etiqueta es un identificador o nombre, permite transferir incondicionalmente la ejecución del programa al lugar en donde se haya declarado el nombre de la etiqueta seguida por dos puntos

etiqueta: /* destino del salto */

Los saltos sólo pueden hacerse dentro de una función y la etiqueta debe ser única en un bloque y es reconocida como tal en los subbloques, a menos que se redeclare en alguno de éstos en los cuales sólo será válida la etiqueta más interna.

EJEMPLO

El programa siguiente lee un número decimal o de punto flotante e imprime su valor redondeado a número entero. El programa se ejecuta hasta que se ingrese un número negativo.

```
main()
{ float num; int r;
  otro: printf("número ");       /* se declara etiqueta otro */
        scanf("%f",&num); r=num +0.5;
        printf("valor redondeado %d\n",r);
        if (r>=0) goto otro;  /* salto con goto */
  printf("fin de programa\n");
}
```

Obsérvese cómo se calcula el valor redondeado: Se suma 0.5 al valor contenido en la variable num y luego se asigna a la variable entera r, que sólo recibe la parte entera del número.

17.9 El operador condicional **? :** se aplica así:

expr ? expr1 : expr2

con tres expresiones expr , expr1 y expr2 .

EFECTO

La expresión compuesta es igual a expr1 o a expr2, según expr sea distinto de cero (verdadero) o nó.

EJEMPLO

```
main()    /* calcula máximo de dos números usando ? :  */

{ float a,b:
  printf("ingrese dos valores "); scanf("%f %f,&a,&b);
  printf("el mayor es %f\n",(a>b)? a:b);    }
```

18 OPERADORES DE INCREMENTO Y DECREMENTO: ++ Y --

Los operadores ++ y -- suman o restan 1 el valor de la variable afectada.

Se aplican en las formas de prefijo o de sufijo así:

++ var o var++

para incrementar. En el primer caso se incrementa el valor de la variable antes de volver a utilizarla; en cambio en el segundo caso el incremento se realiza después de haber utilizado su valor actual.

De igual forma se tienen los operadores de decremento:

-- var o var--

EJEMPLO

```
main()
{  int  p, s;
   printf("prefijo : "); for (p=0;++p <3;) printf("%6d",p);
   printf "\nsufijo :)" ;for (s=0;s++ < 3;) printf("%6d",s);
                                          /* ^ uso de valor */
}
```

EJECUCION

```
prefijo :  1    2
sufijo  :  1    2    3
```

19 ALGUNAS FUNCIONES MATEMATICAS

Algunos compiladores de C contienen funciones matemáticas para ser empleadas directamente por los usuarios. Tales funciones se hallan en librerías especiales y frecuentemente pueden ser utilizadas escribiendo previamente una directiva de inclusión como:

```
#include <math.h>
o
#include <math.lib>
```

Ejemplos de tales funciones son :

abs(X)	da el valor absoluto del número X
sqrt(X)	da la raíz cuadrada de X
sin(X)	igual a seno de X radianes
cos(X)	igual a coseno de X radianes
tan(X)	igual a tangente de X radianes

El lector debe consultar el manual de referencia del compilador de C con el que trabaja para enterarse sobre las funciones disponibles en su sistema (y su forma de uso).

20 EJERCICIOS

1. Escriba un programa para hallar las raíces de la ecuación de segundo grado:

$$ax^2 + bx + c = 0$$

que son dadas por raiz1=(-b + d)/2a y raiz2=(-b-d)/2a, siendo d=sqrt(b^2- 4ac)

2. Escriba un programa que imprima una tabla de valores de la función seno para los ángulos 0, 5, 10 ..., 90 grados.

3. Escriba un programa que obtenga los dígitos de un número entero.

4. Escriba un programa que lea una fecha, por ejemplo en la forma MA 25 12 1984 (martes 25 de diciembre de 1984) , e imprima el calendario del mes correspondiente.

5. Extienda el programa anterior de modo que imprima los calendarios de los 12 meses a partir de la fecha dada.

6. Escriba un programa que calcule el interés I que produce un capital C durante n años a una tasa de R% anual de interés compuesto. Se emplea la fórmula:

$$I = C\,(1+r)^n - C\,, \qquad \text{siendo } r = R/100.$$

7. Escriba un programa que lea n datos y calcule su media aritmética, la suma total, el máximo y el mínimo de los datos.

8. Escriba un programa que cuente palabras ingresadas por el dispositivo estándar de entrada.

   Utilice la función getchar() cuyo valor (de tipo int) es el siguiente carácter ingresado o leído, o el valor -1 si se está en el final (por ejemplo Control Z para el teclado).

   Asuma que las palabras se separan con caracteres de blancos, de tabulación o de cambios de líneas: ' ','\t','\n', respectivamente.

CAP. 3

ARREGLOS

21 ARREGLOS

21.1 Un arreglo unidimensional A, o simplemente un **arreglo**, de N variables de tipo T se declara en la forma:

```
T A[N];
```

y consiste de la colección de variables A[0],..., A[N-1] de tipo T, que son referidas por el nombre del arreglo y el correspondiente subíndice entre corchetes.

El nombre del arreglo es un identificador definido por el usuario y el número N, llamado el orden o dimensión del arreglo, es un entero positivo constante.

Las variables componentes del arreglo se tratan como cualquier variable del tipo de base T.

EJEMPLO 1

La declaración:

```
int lista[100];
```

define el arreglo lista con 100 variables componentes de tipo int:

```
lista[0], ..., lista[99].
```

Una expresión que tome valores dentro del intervalo de enteros 0,...,N-1, puede ser utilizada como subíndice para especificar una variable componente del arreglo.

Si I es una variable entera con valor actual 10 y lista es el arreglo del ejemplo anterior, la instrucción:

```
lista[I*I-25]=560 ;
```

asigna el valor 560 a la variable lista[75].

21.2 INDICACIONES SOBRE ARREGLOS

Consideremos un arreglo A definido por: T A[N];

Entonces:

1) Los datos de las variables componentes A[0],..., A[N-1], se almacenan unos a continuación de otros.

2) El nombre del arreglo es igual a la dirección de la primera variable del arreglo A[0]. Así A tiene el valor &A[0].

3) La dirección de cada componente A[i], dada por &A[i], también puede ser expresada por A+i.

21.3 PROGRAMA: ORDENACION DE DATOS DE UN ARREGLO

El programa que se muestra a continuación lee K números en un arreglo dado de tipo float, los ordena en forma ascendente y luego los imprime.

Se asume que K puede variar entre 1 y 50.

```
#define  límite 50

main()
{  float dato[límite],temp;
   int  K,i,j;
   printf("ingrese número de datos ");
   scanf("%d",&K):

   /* lectura */
      printf("ingrese %d valores \n",K);
      for (i=0;i<K;i++) scanf("%f",&dato[i]);

   /* ordenación  */
      for (i=0; i<K;i++)
       for (j=i+1; j<K;j++)
         if (dato[j]<dato[i])  /* si dato[j] es menor que
                                  dato[i], intercambiar */
            { temp=dato[i]; dato[i]=dato[j]; dato[j]=temp; }

   /* impresión de datos */
      printf("datos ordenados ...\n");
      for (i=0;i<K;i++) printf("%10.3f",dato[i]);

}
```

21.4 ARREGLOS DE VARIAS DIMENSIONES

Un arreglo de dimensión dos de datos de tipo de base T se declara en la forma:

```
T  A[M][N] ;
```

en donde A es el nombre del arreglo y M, N son dos constantes enteras positivas.

EJEMPLO

Si se declara el arreglo dimensional NOTAS de tipo int

```
int NOTAS[ 60][ 5] ;
```

entonces se dispone de 60x5=300 variables simples de tipo int

NOTAS[0][0],..., NOTAS[0][4], NOTAS[1][0],..., NOTAS[1][4], ..., NOTAS[59][0],..., NOTAS[59][4].

Los datos de estas variables se almacenan según el orden en que se presentan.

El arreglo NOTAS puede ser considerado como un arreglo simple NOTAS[0], ..., NOTAS[59]

en donde cada componente NOTAS[i] es a su vez un arreglo de 5 variables de tipo int.

Obsérvese que las tres direcciones siguientes son iguales:

NOTAS, NOTAS[0] y &NOTAS[0][0].

También pueden declararse arreglos de dimensión 3. Por ejemplo:

```
float r[ 5][ 4][ 10];
```

define un arreglo de nombre r con variables de tipo float que pueden ser referidas a través de tres subíndices.

21.5 CADENAS O ARREGLOS DE CARACTERES

Una **cadena** (variable) C es simplemente un arreglo de caracteres:

```
char  C[M];
```

en donde M es el máximo número de caracteres que C puede alcenar.

En cada variable componente C[i] se puede almacenar un carácter.

EJEMPLO

En este programa se lee un texto ingresado por el teclado en una variable de cadena llamada línea y luego se imprime el número de palabras que la componen.

Se supone que las palabras se separan con uno o más caracteres de blancos y que la lectura se hace con la función scanf hasta que se ingrese un cambio de línea (caracter '\n').

Cuando termina la lectura de caracteres se agrega el carácter nulo ('\0' o simplemente 0) a la cadena como símbolo de fin o delimitador de la cadena.

```
main()
{  char  línea[80], car, siguiente, blanco;
   int     i, npal;
   printf("ingrese texto : ");
   i=0;
   /* lectura de cadena  */
   while (1)
     { scanf("%c",&car);
       if (car=='\n')  {  línea[i]='\0';  break; }
       línea[i++]=car;
     }
```

```
/*  conteo de palabras  */
npal=i=0;   blanco=' ';
car=línea[i];                          */  comienzo de la cadena  */

while (car!='\0')
   { if (car!=blanco)
         { siguiente=línea[i+1];
           if  ( (siguiente==blanco)  || (siguiente=='\0') ) +npal;
         }
      car=línea[++i];
   }

printf("texto : %s\n tiene %d palabras\n",línea,npal);
```

21.6 LECTURA DE CARACTERES

La lectura de caracteres se puede hace igualmente empleando una de las siguientes funciones: getchar(), getch() o getc(), que devuelven el siguiente carácter leído o el valor -1 si la lectura ha concluido (fin del archivo de entrada: en algunas versiones de C equivale a ingresar Control Z).

Por ejemplo, el bloque afectado por while (1) del programa puede reemplazarse por:

```
while ((car=getchar())!='\n')
    línea[i++]=car;
línea[i]='\0';                    /* fin de cadena */
```

21.7 LECTURA DE CADENAS

En el ejemplo anterior hemos leido y asignado en forma individual cada uno de los caracteres que forman una cadena. También se dispone de otras funciones que permiten leer y asignar valores a una cadena sin hacer referencia a los componentes de la misma.

Si t es una cadena variable, por ejemplo definida por char t[30]; , entonces se pueden utilizar las funciones **scanf()** o **gets()** así:

1) scanf("%s",t);

con la cual se lee una palabra en la cadena t. La función scanf() distingue las palabras utilizando los caracteres de blancos.

o 2) gets(t);

que lee una línea completa de caracteres en la cadena t.

En ambos casos el sistema añade el carácter nulo ('\0') al final de los caracteres asignados en la cadena.

```
main()
{ char  t[80];
  printf("ingrese texto : ");
  gets(t);
  printf("texto leido : %s",t);
}
```

EJECUCION

```
ingrese texto : Uno para todos
texto leido : Uno para todos
```

21.8 INICIALIZACION DE ARREGLOS

En general un arreglo definido dentro de una función no puede ser inicializado al momento de su declaración. En particular esto se aplica a los arreglos de caracteres o cadenas variables.

Hay dos maneras de hacer inicializaciones con los arreglos:

1) Si son globales, esto es, cuando se declaran fuera de las funciones

2) Si son locales estáticos, esto es, se declaran dentro de una función y se preceden por la palabra **static**.

Los arreglos globales o estáticos pueden ser inicializados empleando el signo de igualdad seguido por una lista de valores iniciales, separados por comas, encerrados entre llaves:

```
tipo    A[N] = { lista de valores iniciales } ;
```

EJEMPLO 1 ARREGLO GLOBAL INICIALIZADO

```
      /*  valor de un arreglo global  */
float   valor[5]={ 23.45, 30.70, 100, 406, 370.75 };

main()
{ int i;
  for (i=0;i<5;i++) printf("%8.3f",valor[i]);
}
```

EJEMPLO 2 ARREGLO LOCAL ESTATICO INICIALIZADO

```
main()
{
   static float  valor[5]={ 23.45, 30.70, 100, 406, 370.75 };
   int i;

   for (i=0;i<5;i++) printf("%8.3f",valor[i]);
}
```

EJEMPLO 3 INICIALIZACION DE CADENAS

Si mensaje es un arreglo de caracteres global o estático puede ser inicializado en la forma anterior:

```
char  mensaje[]={'H','O','L','A','\0'} ;
```

pero para esta clase de arreglos se admiten inicializaciones con cadenas constantes, esto es, con el texto encerrado entre comillas:

```
char  mensaje[]="HOLA";
```

siendo esta expresión completamente equivalente a la anterior, es decir, el compilador le agrega el carácter nulo.

En este ejemplo se ha omitido la dimensión del arreglo, en cuyo caso el programa traductor la calcula contando los valores inicializadores y por lo tanto el arreglo mensaje será de dimensión 5.

22 EJERCICIOS

1. Escriba un programa que lea n números de tipo int en un arreglo dato[0], ..., dato[99] y determine cuántos son iguales al valor máximo.

2. Escriba un programa que lea dos matrices A y B de 3x3 datos de tipo float y calcule su suma y su producto.

3. Escriba un programa que lea n líneas de texto en un arreglo bidimensional de caracteres línea[60][80], de modo que las líneas leidas se almacenen sucesivamente en las cadenas

   línea[0], ..., línea[n];

4. Escriba un programa que lea dos cadenas y las compare según el orden del diccionario.

5 Escriba un programa que lea los coeficientes de un polinomio en un arreglo P[0], ..., P[n] de datos de tipo float, y que evalúe el polinomio en un valor X leído durante corrida.

6. Escriba un programa que lea un texto introducido por el teclado y almacene las palabras en un arreglo de cadenas

   palabra[0], ..., palabra[n]

   en donde palabra se declara en la forma:

   char palabra[60][80];

7. Escriba un programa que ponga en un arreglo de enteros los 1000 primeros números primos.

CAP. 4

FUNCIONES

23 FUNCIONES

Ya se ha indicado que un programa en C se compone de una colección de una o más funciones, una de las cuales es necesariamente la función main().

Una función es simplemente un grupo de instrucciones cuya ejecución se lleva a cabo cada vez que se invoca el nombre de la función posiblemente con algunos valores, llamados argumentos. Cuando termina la ejecución de la función se retorna al punto de llamada para continuar con las instrucciones siguientes.

Una función f se declara en la forma:

```
T  f(lista de parámetros)
declaración de los tipos de parámetros;
{
   cuerpo de la función
}
```

en donde:

1) f es el nombre de la función
2) T es el tipo de los valores que la función ha de devolver.

3) la lista de parámetros (que puede ser vacía) contiene una serie de nombres, separados por comas si hay mas de uno, a través de los cuales se espera pasar valores a la función

4) la declaración de los tipos de los parámetros sirve para especificar los tipos de parámetros;

5) el cuerpo de la función se compone de instrucciones como en la función main, es decir puede contener sus propias variables (locales a la función), bloques, instrucciones de entrada y salida, de control, llamadas a otras funciones, etc.

23.1 LA INSTRUCCION RETURN

La función se ejecuta hasta que se encuentre el final del cuerpo de la función o hasta que se encuentre una función de retorno:

```
        return;
o       return(expresión);
```

En ambos casos finaliza la ejecución de la función, pero en el segundo además devuelve el valor de la expresión.

23.2 LLAMADA A UNA FUNCION

Consideremos la función f definida así:

```
T   f(X,Y,Z)

T1  X;
T2  Y;
T3  Z;

{     cuerpo de la función  }
```

con parámetros (formales) X,Y,Z de tipos T1,T2,T3, respectivamente.

En cualquier lugar del programa se puede llamar la función f -incluso desde el mismo cuerpo de f- para ser ejecutada, simplemente utilizando su nombre con valores en los lugares de los parámetros:

```
f(E1,E2,E3)
```

en donde E1, E2 y E3 son expresiones que toman valores de los tipos de los parámetros, en el orden indicado. Tales valores son recibidos por los parámetros X, Y, Z, los cuales pueden ser tratados y utilizados como variables locales de la función, previamente inicializadas con los valores pasados.

Debe tenerse presente que estas variables X, Y, Z (y sus valores finales) desaparecen cuando concluye la ejecución de la función.

Los valores E1, E2 y E3 siguen siendo los mismos antes y después de la ejecución de la función, aunque éstos sean variables. Así las llamadas a funciones siempre se hacen pasando el valor de los argumentos (**llamadas por valor**).

La llamada a una función puede utilizar o no su valor: En el primer caso el valor de la función f(E1,E2,E3) se emplea como dato de una instrucción, por ejemplo:

```
        v=f(E1,E2,E3);
o       printf("%f",f(E1,E2,E3));
```

suponiendo que f es de tipo float,

y en el segundo caso basta escribirla como una proposición, por ejemplo: `f(E1,E2,E3);`

Un ejemplo típico es la función scanf que devuelve el número de datos que ha logrado asignar o almacenar.

NOTA

Si se emplea la función f de tipo T en otra función debe ser anunciada conjuntamente con su tipo en la declaración de variables. Esto debe hacerse indicando el tipo seguido por el nombre de la función y un par de paréntesis, sin los parámetros.

Si la función es de tipo int (que incluye a char) puede omitirse la declaración de éste en la definición de la función y también se puede omitir la indicación precedente.

EJEMPLO 1

El siguiente programa utiliza una función área(X,Y) de tipo float que devuelve el área de un rectángulo.

```
main()
{ float base, altura, r, área() ;    /* se anuncia función
                                        área de tipo float */
  printf("ingrese base y altura ");
  scanf("%f %f",&base,&altura);
  r=área(base,altura);
  printf("el área es %10.3f\n",r); }

/* función área  */

float área(x,y)
float x,y;
{ return(x*y); }        /* devuelve el producto de x por y  */
```

EJEMPLO 2

En este programa se presenta una función Factorial(N) de tipo float que devuelve el valor del factorial del entero positivo N.

```
main()

{ int n; float Factorial();
  printf("ingrese entero positivo ");
  scanf("%d",&n);
  printf("%10.0f\n",Factorial(n));   /* llamada a función
                                        Factorial  */
}

  float Factorial(n)
  int n;
{    if (n<1) return(1);
     else return(n*Factorial(n-1));
}
```

Obsérvese que la función Factorial se llama a sí misma, (llamada recursiva) en su propio cuerpo.

También debe indicarse que el parámetro n de la función Factorial es distinto de la variable n de main.

EJEMPLO 3. FUNCIONES DE MANEJO DE CADENAS

En este programa se incluyen dos funciones lee_cadena y comparar para leer cadenas de caracteres y comparar dos cadenas según el orden alfabético o de diccionario.

Estas funciones tomarán valores enteros y por lo tanto no se especifican sus tipos ni se anuncian en la función main.

Recuérdese que el nombre de un arreglo es la dirección del comienzo del arreglo.

```
main()
{ char   s[80],t[80];   /* dos cadenas s y t con 80
                                                caracteres  */
  int res;
  printf("ingrese primera cadena : "); lee_cadena(s);
  printf("ingrese segunda cadena : "); lee_cadena(t);
  res=comparar(s,t);
  if (res>0)
      printf("la primera es mayor\n");
   else if (res==0) printf("son iguales\n");
        else printf("la segunda es mayor\n");
}



lee_cadena(x)
char x[];
{ int n;   char c;
  n=0;
  while   ((c=getchar())!='\n') && (n<79))
          x[n++]=c;
  x[n]='\0';          /* agrega delimitador nulo  */


/*  incluir:    return(n);
    si se desea que la función devuelva la longitud de
    la cadena, esto es, el número de caracteres que la
    componen  */
}



/*  función comparar devuelve :
                    número positivo si x es menor que y,
                    cero si x e y  son  iguales,
                    número positivo si x es mayor que y  */



comparar(x,y)
char x[],y[];
{ int n;
  n=0;
  while ((x[n]!='\0') && (y[n]!='\0') && (x[n]==y[n]))  n++;
  return(x[n]-y[n]);
}
```

COMENTARIO

1) Ambas funciones tienen como parámetros cadenas de caracteres. No se especifican las dimensiones de estos arreglos pues quedan determinadas por las dimensiones de los arreglos que se pasan en las llamadas.

2) Cuando se llama una función pasando el nombre de un arreglo como parámetro, ésta recibe la dirección del arreglo y por lo tanto es posible acceder a cada variable componente del mismo.

3) Obsérvese también que lee_cadena lee a lo sumo 79 caracteres reservando un lugar para el carácter delimitador '\0' (cero).

4) Las funciones lee_cadena() y comparar() son equivalentes a las funciones gets() y strcmp() respectivamente, la última se incluye usualmente en una librería de manejo de cadenas, que puede ser accedida, por ejemplo, mediante:
#include <string.h>

23.3 FUNCIONES DE ENTRADA Y SALIDA PRINTF Y SCANF

Función printf

La función printf permite imprimir una lista de datos de acuerdo a una forma especificada.

Se emplea en la forma:

```
printf("cadena de formato",d1,d2,...,dk) ;
```

en donde

d1,d2,...,dk son los datos a imprimir

y la cadena de formato se compone de caracteres de dos clases:

a) caracteres ordinarios, que se imprimen directamente,

y b) grupos de conversión, que son precedidos por el símbolo % :

% m.n carácter de conversión

el carácter de conversión puede ser:

- c para caracteres
- d para números enteros
- f para números decimales o de punto flotante
- s para cadenas
- m es el ancho mínimo del campo de impresión, esto es, el menor número de columnas que ha de ocupar el dato impreso
- n se aplica a los números decimales y a las cadenas: en el primer caso especifica el número de dígitos decimales, y en el segundo el máximo número de caracteres que deben imprimirse de la cadena.

Por defecto los datos se imprimen justificados por la derecha en su respectivo campo, es decir ajustados al lado derecho; si se desea que resulten ajustados al comienzo del campo se puede utilizar el signo menos después del signo %:

%-m.n carácter de conversión

Se puede omitir los valores de m,. o n, en cuyo caso se imprime de acuerdo a las especificaciones preestablecidas por el traductor.

Cada grupo de conversión afecta la impresión del dato correspondiente, de modo que estos grupos de conversión deben guardar el mismo orden que los datos que se van a imprimir.

EJEMPLOS

1) printf("El resultado es \n") ;

imprime la cadena de formato pues no tiene grupos de conversión.

2) printf("\n%40s","El resultado es");

En este caso la cadena de formato es "\n%40s" y el dato a imprimir es la cadena constante "El resultado es ". La impresión resultante es:

```
                                        El resultado es
                                                      ↑— columna 40
```

3) Si n es una cadena que contiene "RAMOS JOSE LUIS", v es una variable que contiene el valor 4560.5, entonces

printf("%30.10s tiene %-8.3f",n,v);

efectúa la impresión:

```
          primer campo                          segundo campo
---------- 30----------              --- 8 --
                          RAMOS JOSE  tiene 4560.500
                          --- 10 ---             -3-
ajuste por la derecha     ↑                      ↑ ajuste por la izquierda
```

Función scanf

La función scanf lee una cadena de caracteres desde el dispositivo estándar de entrada, extrae las subcadenas componentes, las convierte en datos de tipos especificados y asigna éstos a las variables especificadas.

Las subcadenas son delimitadas por espacios en blanco o por caracteres de coincidencia.

La función scanf se aplica en la forma:

```
scanf("cadena de forma",dir1,dir2, ..., dirk);
```

en donde dir1, dir2, ..., dirk, representan las direcciones de las variables.

Si las variables son de los tipos simples sus direcciones se calculan anteponiéndoles el operador de dirección &.

La cadena de formato se compone de dos clases de caracteres:

a) caracteres ordinarios, que son utilizados para efectuar comparaciones de coincidencia con los datos ingresados, es decir se espera que los datos ingresados, contengan tales caracteres.

   Si concuerdan se prosigue para ensayar la siguiente asignación; si no concuerdan se interrumpe el proceso de asignación.

b) grupos de conversión, precedidos por el símbolo %:

   % m carácter de conversión

   en donde el carácter de conversión puede ser c, d, f, etc, como en la función printf.

   El número m indica que a lo sumo se han de tomar m caracteres de la subcadena para los efectos de la conversión.

EJEMPLO 1

Si se tienen las variables:

```
float cod;   char ap[20], n[20];
```

y en respuesta a scanf("%f %s %2s",&cod,ap,n);

se ingresa la cadena:

```
12 CASTRO     ALBERTO
```

entonces:

1) "12" se convierte a número de punto flotante y se asigna a cod

2) la subcadena "CASTRO" se asigna a ap

y 3) la subcadena "AL" se asigna a n

EJEMPLO 2

Se declaran las variables:

```
int n;
float x;
```

y se ejecuta la instrucción: scanf("%4d12%f",&n,&f);

En este caso 12 son caracteres de coincidencia.

Si se ingresa:

```
23451245.6
```

entonces: "2345" se convierte a entero y se asigna a n,

"12" coinciden con los caracteres de coincidencia especificados

"45.6" se convierte a número de punto flotante y se asigna a f.

Si en cambio se ingresa:

23457245.6

la función scanf termina al comprobar que no hay coincidencia entre el carácter 1 de "12" y el carácter 7 de "72". Así sólo se realiza la asignación de valor para n, pero no para la variable f.

24 CLASIFICACION DE LAS VARIABLES

Las variables pueden ser **locales o globales.**

24.1 Las variables **locales** son aquéllas que se declaran dentro de alguna función. A su vez pueden ser **automáticas** o **estáticas.**

Las variables locales automáticas se declaran en la forma simple:

```
{   T var;
```

y si son de tipo simple (char, int, float, double) pueden ser inicializadas en el momento de la declaración usando el signo de igualdad:

```
{   T var=valor inicial;
```

Estas variables se crean, y son inicializadas, si así han sido declaradas, cada vez que se ejecuta el bloque en donde se han definido. Luego desaparecen cuando se abandona su bloque.

Las variables locales estáticas se declaran utilizando la palabra reservada **static**:

```
{ static T var;
```

y pueden ser inicializadas con el signo de igualdad seguido del valor inicial, o por un grupo de valores encerrados entre llaves si la variable es de tipo compuesto, por ejemplo: los arreglos, y las estructuras y uniones que serán tratadas más adelante.

Estas variables reciben sus valores iniciales durante la traducción del programa (si no se les asigna, por defecto se les da el valor 0), y no desaparecen cuando termina la ejecución del bloque donde han sido declaradas sino por el contrario retienen sus valores para la siguiente ejecución del bloque.

EJEMPLO

```
main()
{ int n;
  for (n=1;n<4;n++)
    { printf("llamada %d a función muestra : ")
      muestra();
    }
}

muestra()

{ int a=0;           /* variable local automática */
  static int b=0;    /* variable local estática */
    a=a+1;
    b=b+1;
  printf("a=%d    , b=%d\n",a,b);

}
```

EJECUCION

```
llamada 1 a función muestra: a=1    , b=1
llamada 2 a función muestra: a=1    , b=2
llamada 3 a función muestra: a=1    , b=3
```

Nótese como la variable automática a siempre repite su valor 1; en cambio la variable estática b incrementa su valor con cada llamada.

24.2 Las variables **globales** son aquéllas que se definen fuera de las funciones. Estas variables pueden ser utilizadas por todas las funciones que se declaren después de ellas (en el mismo programa fuente) y mantienen sus valores entre las distintas llamadas.

Pueden ser inicializadas en la misma forma que las variables locales estáticas: reciben el valor inicial sólo una vez, durante el proceso de traducción.

Si se requiere usar una variable global que se declara después en el mismo archivo fuente o está declarada en otro archivo, se puede usar la palabra reservada **extern** a fin de indicar que dicha variable global se halla declarada en otro lugar:

```
extern T var;
```

con lo cual se puede acceder a dicha variable a partir de este lugar.

EJEMPLO

```
main()
{ printf("valor de f(3) : %d\n",f(3));
  printf("valor de g(2) : %d\n",g(2));
  printf("\n");
  printf("valor de f(3) : %d\n",f(3));
  printf("valor de g(2) : %d\n",g(2));

}

extern float x;     /* se usa extern para poder utilizar x
                       en la función  f(y)  */

f(y)
int y;
{  return(x+y); }

float x=5.3;        /* declaración de variable global x */

g(t)
int t;
{ x=x+1; return(x*t); }
```

EJECUCION

```
valor de f(3) = 8
valor de g(2) = 12

valor de f(3) = 9
valor de g(2) = 14
```

EXPLICACION

La variable global x se declara antes de la función g(t) y recibe el valor inicial 5.3 (por una sola vez al iniciarse el programa). La declaración extern especifica que se trata de una variable global definida después, pero que puede ser usada por la función siguiente: f(y).

Cuando se calcula por primera vez f(3) se emplea el valor inicial de x, que es 5.3, por lo tanto f(3) devuelve la parte entera de 5.3 + 3 = 8.3, que es 8.

Luego se calcula f(2). Aquí, x incrementa su valor en 1 y por lo tanto su valor actual es 6.3. g(2) es la parte entera de 6.3*2=12.6, o sea 12.

La siguiente llamada a f, para calcular f(3) otra vez, emplea x con su actual valor 6.3, y por lo tanto f(3) es 9.

25 EJERCICIOS

1. Escriba una función pot(a,n) que devuelva la potencia n-ésima del número a.

2. Escriba una función vol(r) que devuelva el volúmen de una esfera de radio r, dado por

   $4/3*pi*r^3$ en donde pi=3.1415.

3. Escriba una función f(x,y) cuyo valor es dado por:

$$\begin{cases} x+y & \text{si } x>y, \\ x=y & \text{de otra manera.} \end{cases}$$

4. Escriba una función promedio(d,n) que devuelva el promedio de una lista de n números almacenados en el arreglo d[0], ..., d[n] de tipo float.

5. Escriba una función mcd(a,b) que devuelva el máximo común divisor de dos enteros a y b.

6. Escriba una función distancia(x1,y1,x2,y2) que devuelva la distancia entre los puntos (x1,y1) y (x2,y2) dada por

$$[(x1-x2)^2 + (y1-y2)^2]^{\frac{1}{2}}$$

7. Escriba una función recursiva leecar() que lea una línea de caracteres y los imprima en orden inverso.

CAP. 5

APUNTADORES

26 APUNTADORES

Un **apuntador** o **puntero** a datos de tipo T es una variable p cuyo valor es la dirección de una variable *p de tipo T (se suele decir que p apunta a la variable *p).

Se declara en la forma:

```
T   *p;
```

Aquí p es la variable apuntador y *p designa la variable de tipo T a la que p apunta.

El apuntador p puede ser considerado como un par de variables:

(1) p que almacena una dirección (valor entero)
y (2) *p que es la variable de tipo T cuya dirección es dada por p.

EJEMPLO 1

Se declaran dos variables:

```
T  x,  *p;
```

en donde x es una variable de tipo T y p es un apuntador de tipo T.

Entonces para la variable x se tiene:

(1) &x, la dirección (constante) de la variable x
(2) x, variable de tipo T que representa su contenido

y para la variable apuntador p se dispone de:

(1) p, que es una variable de dirección (puede ser cambiada)
(2) *p, variable de tipo T cuya dirección esdada por p.

En este caso se puede asignar a p la dirección de x:

```
p= &x;
```

y por lo tanto p apunta a x, esto es, *p viene a ser la variable x. Por eso, si ahora se modifica el valor de *p, x resultará con el nuevo valor.

USO DE APUNTADORES

Los apuntadores se utilizan, por ejemplo, para modificar el valor de una variable que se pase como parámetro de una función. En efecto, se pasa la dirección de la variable de modo que la función puede acceder a la variable referida por dicha dirección.

Tal es el caso de los arreglos utilizados como parámetros de funciones, pues como se sabe el nombre de un arreglo es la dirección del mismo.

EJEMPLO 2

Este programa usa la función cambiar(u,v) para intercambiar los valores de dos variables de tipo float. Los parámetros de cambiar son direcciones de tipo float, esto es apuntadores de este tipo.

```
main()
{  float a,b;
   printf("ingrese valores de a y b "); scanf("%f %f",&a,&b);
   cambiar( &a,&b) ;      /*  se pasan direcciones de a y b  */
   printf("valores intercambiados:  %f  %f\n",a,b) ;
}

cambiar(u,v)
float *u, *v;
{  float t;
   t=*u;    *u=*v;   *v=t;
}
```

EXPLICACION

Cuando se llama la función cambiar(&a,&b) se pasan las direcciones de a y b, de modo que los parámetros u y v reciben los valores de &a y &b, respectivamente. Como ambos son apuntadores, *u y *v vienen a ser las variables a y b, y sus valores son intercambiados en el cuerpo de la función.

Al terminar la ejecución de la función cambiar, los apuntadores u y v desaparecen (y sus contenidos que son direcciones), sin embargo los valores de a y b (no sus direcciones) ya han sido modificados.

27 MANEJO DE APUNTADORES

En general se debe usar un apuntador de modo que apunte a un objeto válido. En otras palabras, se debe cuidar que p tome solamente direcciones de variables u objetos previamente definidos, pues no se asigna memoria a *p en la declaración. de p.

Sea p un apuntador a datos de tipo T:

```
T *p;
```

1) Si p es otro apuntador a datos de tipo T, se puede hacer:

```
p=q
```

de manera que p apunte a la misma variable a la que q apunta. Así resultan iguales *p y *q.

2) Se puede asignar la dirección nula (cero) a cualquier apuntador:

```
p=0
```

por ejemplo para realizar comprobaciones posteriores.

3) Se puede sumar o restar un dato entero n a un apuntador:

```
p+n  o  p-n
```

que representan las direcciones del n-ésimo objeto de tipo T, ubicado después o antes del objeto referido actualmente por p.

4) Se pueden emplear los operadores de incremento o de decremento:

```
++p,   p++,   --p,   p--
```

para acceder a la dirección del siguiente o anterior objeto.

5) Si q es otro apuntador a datos de tipo T, y p y q apuntan a los componentes de un mismo arreglo de tipo T, se pueden restar los apuntadores:

```
p-q
```

y la diferencia es el número de objetos de tipo T comprendidos entre el menor (excluido) y el mayor.

28 APUNTADORES Y ARREGLOS

Supongamos que se declaran un arreglo a y un apuntador p a datos de tipo T:

```
T  a[N], *p;
```

entonces:

1) El nombre del arreglo a es una dirección constante, igual a &a[0], en cambio p es una variable de dirección.

2) Se puede asignar, por ejemplo, p=a (o p=&a[0]), para que p también apunte al comienzo del arreglo. Y se puede acceder a la i-ésima variable componente del arreglo por medio de p mediante: *(p+i) (igual a a[i]).

3) Las tres direcciones p+i, a+i y &a[i], son iguales.

4) El número de datos componentes del arreglo es fijado por N. El apuntador p puede apuntar a un número arbitrario de datos de tipo T.

5) Puesto que los arreglos pasan sus direcciones cuando se utilizan como parámetros de una función, se pueden declarar apuntadores del mismo tipo en lugar de los arreglos parámetros de la función. Así, en lugar de

```
T   f(x)
T1  x[];          /* x es un parámetro arreglo */
{     ... }
```

se puede emplear

```
T   f(x)
T1   *x;          /* x es un parámetro apuntador */
{   ...  }
```

29 ARREGLO DE APUNTADORES

Se puede declarar un arreglo de N apuntadores a datos de tipo T:

```
T  *p[N];
```

de modo que se dispone de N variables apuntadores a T:

```
p[0], ..., p[N-1],
```

que apuntan a las variables *p[0], ..., *p[N-1] de tipo T, respectivamente.

EJEMPLO

El siguiente programa lee las líneas de un texto en un arreglo de apuntadores línea[0], ..., línea[N-1] de caracteres.

Se asume un máximo de 60 líneas de longitud arbitraria. Los datos serán almacenados en una región reservada por el arreglo área[5000] de tipo char.

El programa lee las líneas y luego las ordena en forma ascendente simplemente modificando las direcciones de los apuntadores. La lectura se hace hasta que se ingrese un cambio de línea al comienzo de una línea.

```
main()
{ char *línea[60], *temp, área[5000];
  int nl,nc,i,j;
  nl=0;
  línea[0]=área;  /* primer apuntador apunta al comienzo de
                     la región reservada  */
  printf("Ingrese líneas de texto\n");
  printf("Termine presionando enter al comienzo de una línea \n");
while (1)
{   nc=lee_cad[línea[nl]);
    if (nc==1) break;
     /*  cada línea termina con el carácter nulo; si no se
         ingresan caracteres la longitud de la línea es 1  */
  línea[nl+1]=línea[nl]+nc;
  nl++;       /*  siguiente línea  */
}
```

```
   for (i=0; i<nl;i++)
      for (j=i+1;j<nl;j++)
       if (comp(línea[i],línea[j])>0)
          { temp=línea[i]; línea[i]=línea[j] ; línea[j]=temp; }
            /*  se intercambio de direcciones si líneas
                no están ordenadas  */

   printf("líneas ordenadas ... \n");
   for (i=0;i<nl;i++)  printf("%s\n",línea[i]);
}


lee_cad(x)
char *x;
{  char *x0, c;
   x0=x;
   while ((c=getchar())!='\n')  *x++=c;
   *x++='\0';
   return(x-x0);    /* devuelve el número de caracteres leídos */
}

comp(x,y)
char *x,*y;
   while((*x!='\0') && (*y!='\0') && (*x==*y)) { x++; y++; }
   return(*x-*y);
}
```

30 ARGUMENTOS DE LA FUNCION MAIN

La función main puede recibir datos, en la forma de cadenas, al momento de iniciarse la ejecución del programa:

np c1 c2 c3 ... ck

en donde np es el nombre del programa ejecutable y c1, c2, ..., ck son cadenas (separadas por espacios en blancos)

que pueden ser utilizadas por la función main si se declara con argumentos o parámetros:

```
main(n,v)
int n;
char *v[];

{ cuerpo de main()    }
```

en donde n es una variable entera que recibe el número de palabras ingresadas: k+1,

y en cada componente v[0], v[1], ..., v[k], del arreglo de apuntadores se reciben las (direcciones de las) cadenas np, c1, c2, ..., ck, respectivamente.

La variable v[0] siempre contiene el nombre del programa u otro nombre predefinido.

EJEMPLO 1

Si se ejecuta el siguiente programa, al que llamaremos SALUDAR, con la orden:

SALUDAR PEDRO

entonces el programa imprime el texto:

Hola PEDRO!

```
                /*  PROGRAMA SALUDAR  */

main(narg,p)
int narg;
char *p[];

{ if (narg!=2) { printf "por favor ingrese un nombre \n");
                 exit(0);
               }
  printf("Hola %s!\n",v[1]);
 }
```

Al ejecutarse el programa narg recibe el valor 2, y por lo tanto están disponibles los valores v[0] y v[1], que contienen las cadenas "SALUDAR" y "PEDRO".

El programa comprueba si se han leido dos palabras o argumentos con:

```
if (narg!=2)
```

de no ser cierto se imprime el mensaje indicado y se utiliza la función exit(0) para dar término al programa.

EJEMPLO 2

El siguiente programa denominado SUMAR, calcula la suma de datos que se ingresan al iniciarse la ejecución del programa.

Por ejemplo, al dar la orden de ejecución o corrida:

```
SUMAR  12  13  14 -42
```

el programa imprimirá:

```
RESULTADO = 28.0
```

```
/*  programa SUMAR  */

main(ndatos,cadnum)
int ndatos;
char *cadnum[ ];    /* ndatos = número de datos
                       cadnum = cadena de números  */

{ int i;
  float temp,res;
  res=0;
  for (i=1; i ndatos; i++)
      { sscanf(cadnum[i],"%f",&temp) ; /*función sscanf*/
        res=res+temp;

  printf("RESULTADO = %8.2f\n",res);
}
```

Funciones sscanf() y sprintf()

En el programa precedente se utiliza la función sscanf que es similar a scanf pero en la cual la cadena de entrada se incluye como primer argumento, a diferencia de scanf que lee la cadena por el dispositivo de entrada estándar (por ejemplo, el teclado):

```
sscanf(cadena_entrada,"cadena de formato", dir1, ..., dirk);
```

En forma análoga, correspondiente a printf se dispone de la función sprintf:

```
sprintf(s,"cadena de formato",dato1, ..., datok);
```

que en lugar de imprimir la lista de datos en el dispositivo estándar de salida, los pone en la cadena s.

En este caso s debe ser un arreglo de caracteres suficientemente grande para recibir a todos los datos.

NOTA

La función sprintf puede ser utilizada para asignar valores a cualquier cadena variable s en forma directa:

```
sprintf(s,cadena)
```

con la cual se copia la cadena en la variable s.

EJEMPLO 3

Supongamos que se declara

```
char  c[100], t[100];
float x;
```

Entonces

```
sprintf(c,"Valor=%8.3f",459.3);
```

asigna a c la cadena "Valor= 459.3" (seguida con '\0').

Si después se aplica

```
sscanf(c,"%s %f,t,&x);
```

entonces t recibe la cadena "Valor=" y x, el valor numérico 459.3 resultante de convertir la cadena "459.3" a número.

31 APUNTADORES A FUNCIONES

En C una función f puede ser pasada como dato argumento de otra función g:

```
g(...,f,...)
```

de manera que g recibe la dirección de la función f.

En este caso el parámetro P de g correspondiente a f debe ser declarado como un apuntador a funciones de la forma:

```
T (*P)();
```

lo cual significa que P es un apuntador - contiene la dirección- de la función (*P)() que toma valores de tipo T. Nótese que en el último par de paréntesis no se incluyen parámetros.

EJEMPLO

El siguiente programa presenta una función fmax que permite calcular el valor máximo de dos funciones cualesquiera cuando se evalúan en un número x. Esto es, fmax(f1, f2, x) devuelve el máximo de los valores f1(x) y f2(x).

En el programa las funciones f y g son pasadas en la lista de argumentos de fmax.

```
main()

{  int x,res, f(), g(), fmax();
   printf("ingrese valor de x : "); scanf("%d",&x);

   res=fmax(f,g,x);        /*  f y g son argumentos  */

   printf("valor máximo de f(x) y g(x): %d\n", res);

}


f(x)            /*  declaración de función f  */
int x;
{ return(x*x);   }

g(x)            /*  declaración de función g  */
int x;
{ if (x<=0) return(0);
  else return(x); }

fmax(p,q,x)                 /*  declaración de fmax  */
int (*p)(), (*q),x;         /*  p y q contienen direcciones de
                                funciones (*p) y (*q)    */

{ int a,b;
    a=(*p)(x);    /*  valor de función  *p en x  */
    b=(*q)(x);    /*  valor de función  *q en x  */
    return( a>b  ? a: b);

} /*  fin de programa  */
```

Cuando se hace la llamada a la función fmax con la instrucción:

```
r=fmax(f,g,x);
```

p y q reciben las direcciones de f y g, respectivamente, de modo que en el cuerpo de la función fmax, las funciones asociadas (*p) y (*g) son precisamente f y g.

NOTA

En la sección de declaración de variables no se debe omitir la indicación de las funciones que toman valores de tipo int y que son empleadas como argumentos de otras funciones. En efecto, una llamada como

```
g(...,f,...)
```

utiliza el nombre f sin los paréntesis y por lo tanto se requiere que antes se especique que se trata de una función:

```
{  int  f();
```

pues de no hacerlo, f aparecerá como un símbolo no definido.

32 EJERCICIOS

1. Escriba una función ordenar(a,b,c) con parámetros:

   ```
   float *a,*b,*c;
   ```

   que cuando se llame por ordenar(&x,&y,&z)
   resulten las variables x,y,z con sus valores ordenados de menor a mayor.

2. Escriba una función f(x,y) con parámetros:

```
int *x,  *y;
```

tal que f(&m,&n) cambie los valores de m y n por los valores de m+n y m*n, respectivamente.

3. Escriba una función lee_datos(lista) con parámetro:

```
float *lista;
```

que sirva para leer datos en un arreglo de tipo float.

El valor de la función debe ser igual al número de datos leidos.

4. Escriba una función polares(x,y,r,a) en la forma:

```
polares (x,y,pr,pa)
float x,y,*pr,*pa;
{  ... }
```

que convierta de coordenadas cartesianas (x,y) a coordenadas polares (r,a) mediante polares(x,y, r,a) en donde

$$r = \text{sqrt}(x^2+y^2);$$
$$a = \text{atan}(y/x)$$

siendo sqrt(X) y atan(X) las funciones raíz cuadrada de X y arco tangente de X (radianes), respectivamente.

5. Un arreglo de caracteres texto[1000] contiene una colección de líneas de textos separadas por '\0' (carácter cero). El arreglo contiene el carácter * para indicar el final del mismo.

Escriba una función buscar(p), con valores apuntador a caracteres, declarada en la forma:

```
char *buscar(p)
char *p;
{    ... }
```

de modo que devuelva la posición en el arreglo en donde se encuentra por primera vez el texto dado por p, o el apuntador nulo en caso contrario.

6. Escriba una función suma_vect(a,b,res) que devuelva en el arreglo res la suma de los arreglos a y b.

   Asuma que los componentes de los arreglos son de tipo float.

7. Escriba un programa llamado sumar que se ejecute en la forma:

   sumar a b

   en donde a y b son dos enteros.,

   que imprima la suma de los enteros desde a hasta b.

8. Escriba un programa convertir que se ejecute en la forma:

   convertir x y

   en donde x e y son dos números de tipo float.

   e imprima las coordenadas polares r y a correspondientes al punto (x,y).

9. Escriba un programa calendar que se ejecute, por ejemplo, en la forma:

   calendar ma 12 5 84

   e imprima el calendario del mes correspondiente a la fecha ma 12 5 84 (martes 12 de mayo de 1984).

CAP.6

ESTRUCTURAS Y UNIONES

33 ESTRUCTURAS

Una (variable de) **estructura** es un identificador o nombre que consiste de una colección de variables de tipos diversos.

Empleando la palabra reservada **struct** se declaran las estructuras en la forma:

```
struct   ns  {   T1   C1;
                 T2   C2;
                 ...
                 Tk   Ck; } lista de estructuras;
```

en donde

- ns es un identificador que sirve para dar nombre al tipo de estructura creado. Este nombre es: struct ns por medio del cual el tipo puede ser referido posteriormente al igual que los otros tipos int, char, float, etc.

- C1, C2, ..., Ck designan los miembros o componentes del tipo de estructura, y son de tipos T1, T2, ...,Tk, respectivamente

- la lista de estructuras contiene la relación de las variables de estructuras (separadas por comas si hay más de una): x, y, ..., z

En este caso, si x es una (variable de) estructura presente en la declaración, entonces consiste de la colección de variables

```
x.C1  de tipo  T1
x.C2  de tipo  T2
...
x.Ck  de tipo Tk
```

en donde se emplea el operador . seguido por el nombre del miembro Ci para acceder a la variable componente correspondiente x.Ci

Una estructura como x ofrece la ventaja de referir un conjunto de variables posiblemente de diferentes tipos utilizando un sólo nombre. Cada variable componente se trata como una variable cualquiera del tipo miembro. El nombre de la estructura, a diferencia de los arreglos, no representa una dirección, pero se le puede aplicar el operador & para obtener la dirección de la estructura. Así, &x representa la dirección de la estructura x.

Los datos componentes de una variable de estructura se almacenan uno a continuación de otro.

EJEMPLO

Se puede emplear una estructura, a la que denominaremos dato, que contenga el nombre y el apellido de una persona con su respectivo salario:

```
main()
 {  struct  {  char    nombre[30];
               char    apellido[30];
               float    salario;      dato;
    printf("ingrese nombre  apellido salario ");
    scanf( "%s %s %f",dato.nombre, dato.apellido,&dato.salario);
    printf("Salario de %s es %8.3f n",dato.apellido,dato.salario);
 }
```

En este caso la estructura dato se compone de tres variables:

dato.nombre, dato.apellido y dato.salario

en donde las dos primeras son arreglos de caracteres de dimensión 30 y la última es una variable de tipo float.

En la lectura con scanf se ha aplicado el operador & para calcular la dirección de la variable dato.salario:

&dato.salario es equivalente a &(dato.salario)

pues el operador . tiene precedencia sobre &.

OBSERVACIONES

1) Los tipos T1, T2, ..., Tk miembros de la estructura son nombres o identificadores de tipos válidos, es decir predefinidos, como int, char, float, etc, o definidos previamente por el usuario, como arreglos, otras estructuras, etc.

2) Si varios miembros Ci son del mismo tipo Ti se pueden declarar separándolos por comas.

3) Si no se desea definir el tipo de la estructura se puede omitir el nombre, como en el ejemplo anterior.

4) También se puede definir primero el tipo de estructura y las variables de estructuras después.

Por ejemplo:

```
                    /*  define tipo estructura: struct sdato  */
struct  sdato  { char nombre[ 30], apellido[ 30] ;
                 float salario; } ;

main
{ struct sdato dato;        /* define a la variable de estructura
                               dato  */
...
}
```

34. UNIONES

Una (variable de) **unión** es un nombre o identificador que puede asumir valores de tipos distintos.

Se declaran en la misma forma que las estructuras pero con la palabra reservada **union** en lugar de struct:

```
union  nu  { T1  C1;
             T2  C2;
             ...
             Tk  Ck; }    x,y ... z;
```

en donde nu es un nombre que sirve para identificar el tipo de unión creado: union nu.

Para acceder a la variable de union x como un dato de tipo T1 se emplea la notación x.C1, que representa en efecto una variable de tipo T1. De igual modo se tratan los casos en que se requiera considerar a x como variable de los otros tipos.

EJEMPLO 1

La variable de unión x puede asumir valores de caracteres o valores de tipo float.

```
main()
{ union  { char car;
           float num; } x;
  x.car='A';
  printf("x contiene caracter %c = \n",x.car);
  x.num=-345.642;
  printf("x contiene número %8.2f\n",x.num);
}
```

EJEMPLO 2

Se declara una variable de unión llamada estado para representar una cadena (comentario) o un número (saldo)

```
union  ES { char comentario[30]; float saldo; } ;
       /* crea tipo:  union ES  */

main()

{ int dec;
  union  ES estado;
       /* la variable de unión llamada estado puede ser una
          cadena: estado.comentario
          o una variable numérica: estado.saldo  */

printf("ingrese 1=comentario o 2=saldo : ");
scanf( %d",%dec);

if (dec==1) { printf("escriba comentario : ");
              scanf("%s",estado.comentario);
              printf("comentario leido :%s\n",estado.comentario);
            }
if (dec==2) { printf("ingrese saldo : ");
              scanf("%f",&estado.saldo);
              printf("saldo leido : %10.2f\n",estado.saldo);
            }
}
```

35 APUNTADORES A ESTRUCTURAS Y UNIONES

La declaración

```
struct  nu  { T1  C1;
            ...   ...
                  Tk  Ck ;}    *p;
```

especifica que p es una variable apuntador a la variable de estructura *p.

En este caso p contiene la dirección de la estructura *p, cuyos miembros son las variables (*p).C1 de tipo T1, ...

Si en lugar de la palabra struct, en la declaración anterior se usa la palabra union, entonces se puede emplear la variable (*p).C1 cuando se requiera acceder a *p como un dato de tipo T1.

NOTACION

En ambos casos se suele escribir **p->C1** en lugar de (*p).C1, y de igual modo con los otros miembros de *p.

Es conveniente reiterar que p es la dirección del agregado de datos cuyos miembros son p->C1, ..., p->Ck.

NOTA

Los apuntadores a estructuras o a uniones son frecuentemente utilizados como parámetros de funciones para pasar las direcciones de las variables de estructuras o de uniones.

EJEMPLO 1

En el siguiente programa se crea el tipo de estructura struct sreg.

Se desea leer en las variables componentes de la estructura dato utilizando una función leer_reg que reciba la dirección de la estructura como argumento. Por eso leer_reg tiene como parámetro un apuntador x al tipo struct sreg. Las variables a la que apunta x son: x->nombre (cadena de dimensión 20) y x->saldo (de tipo float). Puesto que scanf requiere una dirección en cada argumento, para la variable x->saldo se debe poner

&(x->saldo) que es equivalente a &x->saldo

ya que el operador -> se evalúa antes que &.

El programa es:

```
struct  sreg  { char nombre[20];  float  saldo;  };

main()
{ struct sreg  dato;
  printf("ingrese apellido y saldo ");
  leer_reg(&dato);
  printf("datos recibidos: %s %8.2f\n",dato.nombre,dato.saldo);
}

leer_reg(x)
struct sreg * x;
{ scanf("%20s %f",x->nombre,&x->saldo);
}
/*  fin de programa  */
```

EJEMPLO 2

Se puede inicializar una variable de estructura si es global o estática.

Por ejemplo, si struct sreg es el tipo de estructura anterior, entonces:

```
struct sreg  dato= {"TORRES JUAN",3450.00};
```

define la estructura dato y asigna "TORRES JUAN" a dato.nombre y 3450.00 a dato.saldo, respectivamente.

EJEMPLO 3

Si x es el apuntador utilizado en el ejemplo 1:

1) ++x->saldo y x->saldo++ equivalen a ++(x->saldo) y (x->saldo)++, respectivamente, esto es a incrementar la variable miembro x->saldo.

2) La expresión (++x)->saldo incrementa el valor (dirección) de x, y luego usa la variable x->saldo (de la nueva dirección).

3) La expresión (x++)->saldo usa la variable x->saldo en la dirección actual de x, y luego incrementa esta última.

36. EJERCICIOS

1. Escriba un programa que lea datos en la variable de estructura prod dada por:

```
struct { char nombre[20];
         char código[3];
         int  unidades; } prod;
```

2. Escriba un programa similar al ejercicio 1 pero en donde prod sea un apuntador al tipo de estructura indicado.

3. Se define la variable de unión cantidad mediante:

```
union { float peso;
        int   unidades; }  cantidad;
```

Escriba un programa que para un producto lea el peso o el número de unidades en la variable cantidad.

4. Escriba un programa que utilice un arreglo datos[0],..., datos[99], para almacenar el nombre y el salario de n personas según la siguiente especificación de tipo:

```
struct sregistro { char nombre[30];
                   float salario; };

struct sregistro datos [100];
```

5. Escriba una función total(p):

```
float total(p)
struct sregistro *p;
```

que calcule la suma de todos los salarios del arreglo datos del ejercicio anterior cuando se llama por total(datos).

6. Defina el tipo estructura sfecha por:

```
struct sfecha { int  dia;
                char mes[10];
                int  año;} ;
```

y escriba una función lee_fecha(x), en donde x es un apuntador a struct sfecha, para leer una fecha en x.

7. Se declaran los tipos compuestos:

```
struct sfecha { int dia, mes, año;};

struct slista {   char nombre[30] ;
                  float cantidad, precio,total;
                  struct  sfecha  *pfecha; };
```

Escriba un programa que lea datos en un arreglo lista utilizando una función lee_lista(lista) en donde:

```
struct slista  lista [100];

lee_lista(p)
struct slista *p;
{  ...     }
```

La función devuelve el número de datos leidos.

CAP. 7

INCLUSION DE ARCHIVOS Y MACROS

37 INCLUSION DE ARCHIVOS

La directiva **#include** permite incluir un archivo de programa durante el proceso de traducción, como si las instrucciones que componen dicho archivo hubiesen sido escritas en el programa actual.

En general se usa en la forma:

```
#include  "nombre de archivo"
```

y en particular, si se sabe que el archivo a incluir se halla en uno de los directorios preestablecidos por el sistema, en la forma:

```
#include  <nombre de archivo>
```

38 MACROS

Una **macro**, es un texto o colección de caracteres provisto de un nombre, que se declara utilizando la palabra **#define** al comienzo de una línea y en cualquier lugar del programa:

```
#define  nm   texto de la macro nm
```

en donde nm es el nombre dado a la macro.

El texto está formado por todos los caracteres (incluyendo espacios en blanco) que siguen al nombre de la macro en la misma línea. Para continuar el texto de la macro en la síguiente línea se emplea el símbolo \ después del grupo de caracteres de la presente línea.

El resto del programa fuente puede entonces utilizar una o varias veces el nombre de la macro nm en lugar de escribir su texto.

Antes de iniciarse el proceso de traducción de las instrucciones del programa fuente, se sustituye el nombre de cada macro, que aparezca después de su declaración, por su correspondiente texto. Esta tarea es realizada por un programa llamado preprocesador de C.

No se hacen sustituciones dentro de cadenas encerradas entre comillas.

Se continúa reemplazando hasta finalizar el programa fuente actual o hasta encontrar la directiva:

#undef nm

que instruye al preprocesador para cancelar la definición de la macro nm.

39 USO DE LAS MACROS

1) Para nombrar constantes

EJEMPLOS

```
#define  pi  3.14156
#define  mensaje  "Hola"
#define  formato  "El resultado es %8.3f\n"
```

2) Para dar otros nombres a identificadores y símbolos.

EJEMPLO

```
#define puntero   int  *
#define comienzo {
#define fin ;}
#define mientras  while
#define programa  main
#define imprimir  printf

programa()
comienzo
   puntero n;
   *n=1;
   mientras ((*n)++<5)      imprimir("%6d",*n);
fin
```

3) Para reemplazar grupos de instrucciones.

Se puede definir una macro con parámetros de la forma:

#define nm(P1,P2,...,Pk) texto de la macro

en donde el nombre nm de la macro es seguido por una lista de parámetros P1, P2, ..., Pk, encerrada entre paréntesis y el texto contiene los símbolos P1, P2, ..., Pk.

En este caso la macro se utiliza en la forma:

nm(d1,d2, ..., dk)

con tantos símbolos d1, d2, ..., dk, cuantos parámetros contenga la macro.

Cuando se reemplaza el texto de nm se sustituye a su vez cada parámetro Pi por el correspondiente símbolo di.

EJEMPLO 1

En este ejemplo se define la macro max(x,y) con dos parámetros x, y.

```
#define  max(x,y)    (x)>(y)? (x):(y)

main()
{ float a,b;
  printf("ingrese dos números : "); scanf(%f %f",&a,&b);
  printf("el mayor es %8.2f\n", max (a,b));

}
```

Para el programa traductor, la instrucción printf que contiene a la macro max es precisamente:

```
printf("el mayor es %8.2f\n",  (a)>(b)? (a): (b) );
```

NOTA

1) En algunos casos se deben encerrar entre paréntesis los parámetros que aparecen en el texto de la macro a fin de obtener las expresiones requeridas cuando se haga la sustitución.

   Por ejemplo, para calcular el cuadrado de un dato numérico es preferible escribir la macro:

   ```
   #define cuadrado(x) (x)*(x)
   ```

   y no:

   ```
   #define cuadrado(x)  x*x
   ```

pues esta última no podrá ser usada correctamente con expresiones compuestas, por ejemplo, la sustitución de cuadrado (3+5) es la cadena 3+5*3+5, que será evaluada a 23, que es distinto de 64.

2) Se puede incluir macros, previamente definidas, en otra macro.

EJEMPLO 2

En este programa la macro saludar utiliza la macro mensaje.

```
#define mensaje   "Hola!\n"
#define saludar   printf(mensaje)

main()
{   saludar;   }
```

3) Una macro puede utilizar funciones del programa.

EJEMPLO 3

El siguiente programa contiene la macro lee_entero con dos parámetros t, x, en donde t se usa para imprimir una cadena de mensaje y x para leer un valor entero.

```
#define lee_entero(t,x)      printf("%s",t); \
                             scanf("%d",&x)

main()
{ int a;

      lee_entero("Ingrese número : ",a);
      printf("valor leido %d \n",a);
}
```

40 COMPILACION CONDICIONAL

En el programa fuente se puede indicar al compilador para que compile un grupo de instrucciones según se cumpla una condición o nó.

Las directivas son:

1) #if ... #else ... #endif
2) #ifdef ... #else ... #endif

y 3) #ifndef ... #else ... #endif

La forma **#if** se usa así:

```
#if c
instrucciones 1
#else
instrucciones 2
#endif
```

en donde c es una expresión constante (formada por constantes numéricas posiblemente relacionadas por operaciones).

Su efecto es el siguiente: Se compila el grupo de instrucciones1 si el valor de c es distinto de cero, o se compila el segundo grupo de otra manera.

Se puede omitir #else y el grupo de instrucciones2, en cuyo caso se compilará el grupo afectado por if sólo si c es distinto de cero.

Las otras formas se utilizan de manera similar a la tratada, pero con:

```
#ifdef  n
```

o

```
#ifndef n
```

en lugar de #if c.

En ambos casos n es un nombre o identificador.

En el primer caso se compila el grupo de instrucciones1 si n está definido en el programa, de otra forma se compila el segundo grupo.

En el segundo caso se compila el grupo de instrucciones1 si n no está definido en el programa, o se compila el segundo grupo en caso contrario.

También puede omitirse la parte #else.

Usualmente se utilizan estas construcciones de compilación condicional con las constantes y nombres afectados por #if, #ifdef, etc, posiblemente definidos en un archivo que se incluye en el programa actual. De esta manera se pueden obtener diferentes versiones ejecutables del programa según las especificaciones de las constantes y nombres relacionados con el proceso de compilación condicional.

EJEMPLO

```
main()
{
#ifdef SALUDAR
     printf( SALUDAR) ;
#else
     printf( "Fin de programa\n") ;
#endif
}
```

En este programa sólo se compilará la instrucción afectada por #else puesto que el nombre SALUDAR no está definido.

En cambio si se define SALUDAR por:

```
#define SALUDAR "Hola!\n"
```

antes de #ifdef, en el mismo programa o en un archivo previamente incluido (por ejemplo de nombre "TABLA"):

```
#include  "TABLA"
```

entonces sólo se compilará la instrucción asociada a #ifdef.

41. EJERCICIOS

1. Escriba un programa que lea cinco números de tipo float y calcule su suma.

   Utilice una función lee_float() para leer los números y salve su texto en un archivo num.c.

   El programa debe utilizar la función a través de una inclusión de archivos:
   `#include "num.c"`

2. Escriba una macro cubo(X) para calcular el cubo de un número.

3. Escriba una macro redondeo(Y) para calcular el valor de Y redondeado a dos dígitos decimales.

4. Escriba una macro leedatos(cad,m,n) que sirva para leer una cadena cad y dos números m y n.

5. Defina una macro es_digito(c) que se utilice para determinar si un carácter c es un dígito o nó, devolviendo el valor 1 ó 0, respectivamente.

CAP. 8

MANEJO DE ARCHIVOS

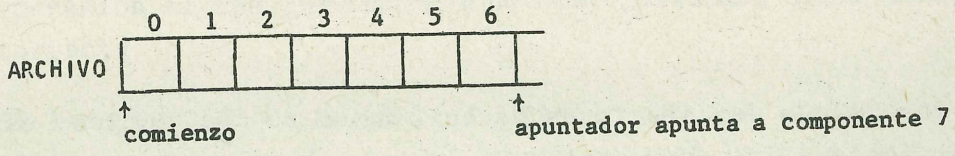
42 INTRODUCCION

Un **archivo** es una colección de datos que con un título o nombre se almacenan o graban en dispositivos tales como discos, disquetes, cintas magnéticas, etc. Los datos de un archivo pueden ser leídos y utilizados por otros medios.

Las operaciones básicas con un archivo son:

1) Apertura del archivo empleando un nombre para identificarlo.
2) Escritura de datos en el archivo
3) Lectura de datos del archivo
4) Cierre del archivo.

Los datos de un archivo son simplemente bytes o caracteres almacenados uno a continuación de otro que son enumerados con 0, 1, ...



Es necesario abrir un archivo antes de proceder a hacer operaciones de escritura o de lectura en él. Cuando se abre un archivo usualmente queda listo para acceder al componente 0, lo que suele decirse que el archivo apunta al comienzo.

Después de cada operación de escritura o de lectura, el archivo apunta al componente que sigue al grupo de datos escritos o leídos: Desplazamiento secuencial.

Debe tenerse presente que para cada archivo abierto el sistema siempre mantiene un apuntador al componente actual o de acceso siguiente. También es posible desplazar o ubicar el apuntador a un componente particular dentro del archivo, empleando funciones de movimiento del apuntador: Movimiento directo.

Se debe cerrar un archivo cuando ya no se desee trabajar con él.

Se dispone de dos sistemas de funciones para operar un archivo:

1) Funciones de entrada y salida de alto nivel.

y 2) funciones de entrada y salida de bajo nivel (al modo del sistema operativo UNIX).

En el primer caso el sistema provee un área de almacenamiento intermedio (buffer) que sirve para almacenar temporalmente los datos escritos o leidos en el archivo. Este sistema permite un control mayor sobre el flujo de los datos y se facilita la conversión de los mismos.

En cambio con las funciones de bajo nivel, cuyo empleo es más simple, las operaciones de lectura y escritura se realizan directamente y el control de los datos queda a cargo del usuario.

43 FUNCIONES DE ALTO NIVEL

Este grupo de funciones se encuentran en la librería estándar de entrada y salida de C y pueden ser accedidas incluyendo el archivo de cabecera stdio.h:

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

Para usar estas funciones se debe declarar una variable apuntador del tipo predefinido **FILE** en la forma:

```
FILE  *arch;
```

en donde arch es un nombre o identificador.

Se abre un archivo usando la función **fopen**(), que toma valores a apuntadores de tipo FILE, en la forma:

```
arch=fopen("nombre de archivo", modo)
```

en donde modo puede ser:

"w" para escribir,
"r" para leer
y "a" para añadir

NOTA Algunos sistemas de C ofrecen modos adicionales.

El modo de escritura crea un nuevo archivo con el nombre dado (si existe uno con el mismo nombre se destruye antes).

La función fopen() devuelve la dirección 0 (cero o NULL) si la operación de apertura falla, o un valor (dirección) distinto si es correcta.

Si la operación es correcta, a partir de este instante el archivo queda referido por el apuntador arch hasta el momento en que se cierra con **fclose** mediante:

```
fclose(arch)
```

que actualiza el archivo (y su directorio) y finaliza el vínculo con la variable arch.

ARCHIVOS DE ENTRADA Y SALIDA ESTANDAR

Los archivos de entrada y salida estándar pueden ser referidos por los identificadores constantes **stdin** y **stdout** que se declaran en el archivo stdio.h

ESCRITURA EN ARCHIVOS

Para escribir en un archivo (abierto en los modos "w" o "a") se pueden utilizar las funciones:

1) **fprintf(arch,"cadena de control", lista de datos)**

   Efecto: igual que printf escribe datos con formato, pero la salida se hace en arch

2) **fputc(c,arch)**

   Efecto: escribe un carácter c en arch

3) **fputs(s,arch)**

   Efecto: escribe una cadena (o apuntador a caracteres) en arch hasta encontrar su final ('\0')

4) **fwrite(s,m,n,arch)**

   Efecto: escribe en arch n datos de tamaño o longitud m (igual a m*n bytes) ubicados a partir de la dirección s.

LECTURA DE ARCHIVOS

Las funciones de lectura son:

1) **fscanf (arch,"cadena de formato",lista de direcciones)**

   Efecto: igual que scanf pero la lectura se hace en arch

2) **fgetc(arch)**

Efecto: devuelve el siguiente carácter leído en arch.

3) **fgets(s,n,arch)**

Efecto: lee una línea de caracteres, hasta un máximo de n-1 carateres o hasta encontrar un cambio de línea ('\n'), del archivo arch y los almacena en la dirección s. Añade al final el carácter nulo ('\0').

4) **fread(s,m,n,arch)**

Efecto: lee en arch n datos de tamaño o longitud m (igual a m*n bytes) y los almacena a partir de la dirección s.

VALORES DE LAS FUNCIONES

Las constantes **EOF** y **NULL** (se declaran en el archivo stdio.h), y son iguales a -1 y 0 (cero), respectivamente, se utilizan para comprobar si estas funciones se ejecutan correctamente.

Los valores de estas funciones son los siguientes:

a) fprintf() y fscanf() devuelven un valor de tipo int igual al número de datos convertidos y almacenados o el valor EOF en caso de error.

b) fputc() y fgetc() devuelven un valor de tipo int igual al carácter escrito o leído o el valor EOF en caso de error.

c) fputs() devuelve un valor de tipo int igual al último carácter escrito o el valor EOF en caso de error.

fgets() devuelve un apuntador a caracteres: el apuntador argumento o el apuntador NULL en caso de error.

d) fwrite() y fread() devuelven un valor de tipo igual al número de datos escritos o leídos.

Así, fwrite() devuelve el número de n datos especificados o un valor menor en caso de error. Y fread() devuelve el valor n o un número menor, por ejemplo 0 si se está en el final del archivo.

43.1 EJEMPLO **Archivo de números**

El siguiente programa escribe y lee los cuadrados de los números enteros 1, 2, ..., 10, en el archivo TABLA.

```
#include <stdio.h>
main()
{ FILE *p;          /* p es un apuntador de tipo FILE  */
  int  n;
  /*  abre (crea) archivo TABLA para escribir;
                        p queda asociado a TABLA  */
  p=fopen("TABLA","w");
  /* escritura en p con fprintf()  */
  for  (n=1;n<=10;n++)      fprint(p,"%6d",n*n);
  /*  cierra archivo TABLA  */
  fclose(p);
  printf("DATOS DE ARCHIVO TABLA : \n");
  /*  abre archivo TABLA para leer  */
  p=fopen("TABLA","r");
  /*  lectura con fscanf()  */
  while  (fscanf(p,"%d",&n)>0 ) printf("%6d\n",n);
  fclose(p);
}
```

NOTA

Las funciones printf() y scanf() son equivalentes a las funciones fprintf(stdout,...) y fscanf(stdin, ...), respectivamente.

43.2 EJEMPLO: COPIA DE TEXTO DE STDIN (TECLADO)

Los dos programas siguientes leen un texto ingresado por el teclado archivo stdin y lo almacena en el archivo TEXTO.

En ambos programas se controla la operación de apertura comparando el valor de fopen() con la constante cero o NULL. En caso de error se da por terminado el programa con la función **exit(0)**.

```
   /* programa usa las funciones fgetc y fputc  */

#include <stdio.h>
main()
{ FILE *t; int c;
  printf("ingrese texto (Control Z para terminar)...\n");
  if ( (t=fopen("TEXTO","w")) ==NULL )
      { printf(" error de creación ... \n"); exit(0); }
  /* archivo creado listo para escribir  */
  while ( (c=fgetc(stdin)) !=EOF )   fputc(c,t);
  fclose(t);

}     /*  fin de programa  */
```

```
/*  programa usa funciones fgets() y fputs() para añadir datos
    en archivo  TEXTO  */

#include <stdio.h>

/*  número máximo de caracteres a leer   */
#define max 80

main()
{ FILE *txt;    char linea[max];

  if ( (txt=fopen("TEXTO","a")) ==NULL)
    {  printf("error ...\n");  exit(0); }

  printf("ingrese texto a ser añadido ...\n");
  while ( fgets(linea,max,stdin) !=NULL ) fputs(linea,txt);
  fclose(txt);
}
```

NOTA

Son equivalentes las funciones:

putc(c)	y putchar(c)	con fputc(c,stdout)
getc()	y getchar()	con fgetc(stdin)

tienen efectos parecidos:

gets(s)	y	fgets(s,BUFSIZ,stdin)
puts(s)	y	fputs(s,stdout)

(BUFSIZ es una constante definida por el sistema, por ejemplo se suele tomar igual a 512)

43.3 EJEMPLO COPIA ARCHIVOS

En este programa se copia un archivo a otro usando las funciones fread() y fwrite().

Los nombres de los archivos son leídos en las cadenas nfuente y ndestino, respectivamente.

```
#include <stdio.h >
 /* número máximo de bytes a ser transferidos  */
#define max 512

main()
{ FILE *fuente, *destino;
  char  área[max], nfuente[12], ndestino[12] ;
  int n;

  printf("nombre de archivo fuente : ");
  scanf( "%s",nfuente);
  printf("nombre de archivo destino : ");
  scanf ("%s",ndestino);

  if ( (fuente=fopen(nfuente,"r")) ==NULL )
     { printf("no existe archivo fuente!\n");
       exit(0); }

  if ( (destino=fopen(ndestino,"w")) ==NULL )
     { printf("no  se puede crear archivo destino!\n");
       exit(0); }

  /* copiar  n  bytes leídos mientras n>0   */
  while ( (n=fread(área,1,max,fuente))>0)
            fwrite(área,1,n,destino);

  fclose(destino);
  fclose(fuente);

}
```

44. FUNCION **sizeof**

La función sizeof(x)

en donde x es una variable o un tipo de datos,

devuelve el número de bytes que ocupa la variable o requiere el tipo.

Esta función se aplica, por ejemplo, a una variable que sirve como área de almacenamiento intermedio (o buffer) para la transmisión de datos con un archivo, a fin de escribir o leer exactamente el número de bytes de la variable.

Mostramos algunos valores de la función sizeof() frecuentemente empleados por algunas versiones de C :

1) Para los tipo char, int y float, y variables de estos tipos, la función sizeof() devuelve los valores 1, 2 y 4, respectivamente.

2) Si x es un arreglo de tipo float, declarado por:

   float x[100] ;

   el valor de sizeof(x) es 400=100*4 (100 variables de tipo float).

3) Si s es una variable de estructura:

   struct { char nombre[30]; float saldo; } s;

   entonces sizeof(s) = sizeof(s.nombre) +sizeof(s.saldo)
   = 30*1 + 4 = 34

NOTA

El uso de la función sizeof() evita el que se tenga que calcular el número de bytes que requiere el almacenamiento datos en una variable (este número usualmente depende de realización del compilador).

45 FUNCIONES DE BAJO NIVEL

Las funciones de entrada y salida de datos de bajo nivel son las siguientes:

1) creat() : crea archivo nuevo y lo abre (para escribir)
2) open() : abre archivo existente
3) write() : escribe en archivo
4) read() : lee archivo
5) close() : cierra archivo abierto
6) lseek() : ubica componente en archivo (para leer o escribir)

Para trabajar con un archivo empleando este grupo de funciones, el sistema le asigna un número entero (distinto de -1) llamado **descriptor** que usualmente se almacena en una variable de tipo int, y sirve para referir al archivo.

El valor del descriptor es obtenido con las funciones creat() y open().

También se dispone de la función **unlink()** para eliminar o borrar archivos. Por ejemplo, se borra el archivo de nombre DATOS mediante:

```
unlink("DATOS");
```

Para crear un archivo se emplea la función creat() en la forma:

arch=creat("nombre de archivo",pmodo)

en donde:

arch es una variable de tipo int que recibe el valor del descriptor asignado si se ha creado el archivo, o el valor -1, en caso contrario;

y pmodo es un número entero que sirve para especificar el modo de protección o permiso que tendrá el archivo. (En algunas versiones de C este argumento no se utiliza, o puede omitirse, con lo cual se asume un archivo de lectura y escritura: permiso regular).

ADVERTENCIA

En los ejemplos que presentamos emplearemos el argumento pmodo con el valor 0 para leer y escribir. El lector debe consultar el manual de referencia del compilador de C con el que trabaje para precisar el uso y los valores apropiados de pmodo.

Una vez que se crea un archivo, con descriptor (en la variable) arch, queda abierto para escribir datos con la función write():

write(arch,dir,n)

que escribe o copia en el archivo n bytes ubicados en la dirección dir.

Esta función devuelve el valor n si la operación es correcta o un valor distinto en caso de error.

La función open() se utiliza para abrir un archivo existente a fin de efectuar operaciones de escritura o lectura:

arch=open("nombre de archivo",modo)

en donde:

arch es una variable de tipo int que recibe el descriptor o el valor -1 en caso de error,

y modo puede ser:

0, sólo lectura
1, sólo escritura
2, lectura y escritura

Para realizar operaciones de escritura se emplea la función write() como se ha indicado antes.

Las operaciones de lectura se realizan con read():

read(arch,dir,n)

que lee n bytes del archivo y los almacena en la dirección dir.

Esta función devuelve el número de bytes leídos (entre 0 y n) o el valor -1 en caso de error. El valor 0 se obtiene si se está en el final del archivo.

Un archivo de descriptor arch abierto con creat() o con open se cierra con la función close() :

close(arch)

EJEMPLO 1: CREACION Y ESCRITURA DE UN ARCHIVO

En este programa se crea un archivo de nombre DATOS para almacenar una colección de números de tipo float.

```
#define pregular 0

main()
{   int  desc, s, num;
    float  cantidad;

    /* crea archivos DATOS: descriptor resulta en desc  */
    desc = creat("DATOS",pregular);
    printf("número de datos : "); scanf("%d",&num);
    printf("ingrese %d números ...\n",num);
    s=sizeof(cantidad);
    while(num>0) {  scanf("%f",&cantidad);

        /* escribe contenido de cantidad  */
        write(desc,&cantidad,s);

        num--;
                }
    close(desc);      /* cierra archivo  */
}
```

OBSERVACIONES SOBRE EL PROGRAMA

1. La instrucción

```
desc=creat("DATOS",pregular);
```

crea el archivo de nombre DATOS (si existe uno con dicho nombre lo destruye) y pone el descriptor en la variable desc.

NOTA

Hemos usado el valor del permiso regular (lectura y escritura) igual a 0 para crear un archivo. Refiérase al manual de referencia de su compilador para utilizar el valor adecuado de pmodo.

El archivo queda entonces abierto para realizar operaciones de escritura en él.

2. Nótese cómo se escribe o graba el contenido de la variable cantidad con la función write() :

```
write(desc,&cantidad,s);
```

que copia en el archivo dado por el descriptor desc el valor de la variable cantidad ya que los dos últimos argumentos son precisamente la dirección de la variable y el número de bytes que requiere su contenido.

3. La instrucción

```
close(desc);
```

cierra el archivo asociado a desc, en este caso DATOS, y actualiza el directorio de éste. De no usarse, no obstante que las operaciones ya han sido realizadas, la información relativa al directorio del archivo no será registrada. Por ejemplo, en el caso presente, lo más probable es que ni siquiera aparezca en el directorio de la unidad de almacenamiento.

EJEMPLO 2 LECTURA DE DATOS DE UN ARCHIVO

El siguiente programa lee el archivo DATOS del ejemplo anterior y calcula la suma de todos sus datos.

```
#define nbytes sizeof(float)
#define lectura 0
main()
{   int arch;
    float  dato,suma;

    arch=open("DATOS",lectura); /*  abre en modo de lectura */
    printf("archivo DATOS contiene: \n");
    suma=0;
    while (read(arch, datos,nbytes)>0 )
          {printf("%15.3f\n",dato);
           suma=suma+dato; }

    close(arch);
    printf("\nSuma total = %15.3f\n",suma);

}
```

ARCHIVOS DE ACCESO DIRECTO

Si un archivo se abre con open() en el modo de lectura y escritura (modo 2) es posible ubicar el apuntador en la posición de un byte componente particular con:

lseek(arch,desp,origen)

en donde:

1) arch es el descriptor del archivo
2) origen es un entero que sirve para indicar el punto de referencia en el archivo respecto del cual se ha de efectuar el movimiento. Usualmente sus valores son:

   0, para el comienzo
   1, para la posición actual
   2, para el final del archivo

y 3) desp es un dato de tipo entero grande (long int o simplemente **long**) que representa el número de bytes que debe desplazarse el apuntador del archivo respecto del origen especificado

El desplazamiento puede ser dado por datos de tipo int convertidos a tipo long en la forma:

```
(long) dato entero
```

He aquí algunos ejemplos con lseek :

a) Para moverse al comienzo del archivo

```
lseek(f,(long)0,0);
```

b) Para moverse al fin del archivo

```
lseek(f,(long)0,2);
```

c) Si el archivo ha sido organizado de modo que, a partir del comienzo, cada bloque de 34 bytes representa un registro, los que numeramos con 1, 2, ..., entonces para acceder al bloque o registro 10, que principia en la posición (10-1)*34 = 306, podemos usar:

```
lseek(f,(long)306,0);
```

NOTA

En general si cada bloque requiere t bytes, el bloque de número n, empieza en la posición (n-1)*t.

d) Para desplazarse en el archivo a 15 bytes anteriores al componente actual:

```
lseek(f,(long)-15,1);
```

EJEMPLO

El siguiente programa procesa un archivo de nombre BASE en modo de acceso directo o aleatorio usando la función lseek().

El archivo se compone de registros con 25 bytes para almacenar una colección de nombres.

El programa se ejecuta mientras se ingrese un número válido de registro (entre 1 y 100), muestra el contenido del registro correspondiente y permite modificarlo o nó (si se lee sólo un cambio de línea).

```
#define nbytes    25
#define max       100

/*  crea archivo con permiso regular y abrir en modo directo
    NOTA.  Estos valores dependen del compilador, por ejemplo
           para TurboC: pregular= 0x180 y directo= 0x 8004   */

#define pregular 0
#define directo  2

main()

{   int desc, num, ncar;
    long pos;
    char nombre[nbytes];

/*   abrir archivo en modo de lectura/ escritura  */

if ((desc=open("BASE",directo))==-1)

    {
      desc=creat("BASE",pregular); /* si no existe, crearlo */
      close(desc);
      desc=open("BASE",directo);   }
```

```
    while(1)
      { printf("ingrese numero de registro ");
        scanf("%d",&num);
        if ( (1<=num) && (num <=max) )
         { pos=(num-1)*nbytes;
           lseek(desc,pos,0);    /* ubicar y leer */
           if (read(desc,nombre,nbytes)>0)
                              printf("%s\n",nombre);
           printf("ingrese nombre)" ;
           ncar=leer(nombre); /* leer nombre en una línea */
           if (ncar==1 ) continue;
           lseek(desc,pos,0);     /* ubicar y escribir */
           write(desc,nombre,ncar);
         }
        else break;
      }
    close(desc);
}

leer(t)
char *t;
{   char *s;
    s=t;
    while (1)
      { /* lee un carácter de la entrada estándar (teclado)
                cuyo descriptor es igual a 0 */
        read(0,s,1);
        if (*s=='\n') break;
        s++;
      }
    *s='\0';
    return(s-t+1);
}
```

46 EJERCICIOS

En los ejercicios 1-4 utilice las funciones de manejo de archivos de alto nivel, es decir usando el tipo FILE.

1. Escriba un programa que escriba en un archivo llamado DATOS una lista de números de tipo float.

2. Escriba un programa que lea los datos del archivo del anterior, los imprima y calcule la suma total y el promedio.

3. Escriba un programa para añadir datos al arvhivo del problema 1.

4. Escriba un programa ORDENAR que se ejecute en la forma:

   ORDENAR Fuente Destino

   (por ejemplo, ORDENAR DATOS DATOS_ORD)

   que obtenga un nuevo archivo Destino con los mismos datos que el archivo Fuente pero ordenados en forma ascendente.

   Se asume que Fuente consiste de datos de tipo float.

5. Resuelva los ejercicios 1-4 usando funciones de manejo de archivos de bajo nivel.

6. Escriba un programa que permita escribir y leer en un archivo registro de datos que contengan el nombre, salario y la fecha de ingreso de cada cliente.

   Utilice los tipos siguientes:

```
struct  sfecha  { int dia, mes, año;};

struct  sreg    { char nombre[25];
                  float salario;
                  struct sfecha;};
```

   El programa debe leer durante corrida el nombre del archivo y abrirlo en modo directo si existe (crearlo antes si es necesario). Emplee el número de registro (a partir de 1) para acceder a los datos del archivo.

CAP. 9

APLICACIONES

A continuación se desarrollan algunos ejemplos de programas en C que se ejecutan con el procesador 8086 bajo el sistema operativo DOS de Microsoft.

Estos programas pueden ser compilados con los compiladores Lattice C, Microsoft C o Turbo C para el procesador 8086.

Utilizamos la función int86() disponible en tales versiones para acceder a las rutinas de interrupción y también mostramos ejemplos de enlace de programas en C con subrutinas escritas en lenguaje ensamblador (ASM o MASM de Microsoft).

Para una información más detallada se refiere al lector interesado tanto a los manuales de referencia de los compiladores mencionados como a algún texto sobre lenguaje ensamblador del 8086 (por ejemplo, "Lenguaje Ensamblador Macro Assembler", por M.Kong, editado por la Facultad de Ciencias e Ingeniería y el Magíster en Infórmatica de la PUCP, 1987).

47 PROGRAMA: BORRA PANTALLA Y UBICA CURSOR

El siguiente programa muestra dos funciones borrar() y cursor() para borrar la pantalla y ubicar el cursor en la fila x y columna y.

El texto del programa es el siguiente:

```
main()
{   borrar();
    cursor(10,10);
    printf("Hola\n");
}

struct es_reg { char  al,  ah, bl, bh, cl, ch, dl, dh;
                int   si,  di, cf, f; } ;


borrar()
{   struct es_reg  reg;

    reg.al=0;   reg.ah=6;   reg.cl=0; reg.ch=0; reg.bh=7;
    reg.dh=24;  reg.dl=79;
    int86(16,&reg,&reg);  /* ejecuta rutina de video int 16 */
    cursor(0,0);

}


cursor(x,y)
char x,y;
{   struct es_reg  reg;

    reg.ah=2;  reg.dh=x;   reg.dl=y;  reg.bh=0;
    int86(16,&reg,&reg);

}
```

COMENTARIO

Ambas funciones utilizan la interrupción de video 16 ó 10 hexadecimal.

2. La función para borrar la pantalla emplea los siguiente valores en los registros AH, BH, CL, CH, DL y DH del procesador 8086.

AH=6 : desplazar las líneas de la pantalla hacia arriba;

AL=0 : número de líneas a desplazar (0 significa todas)

CH, CL y DH, DL definen los extremos (superior izquierdo e inferior derecho) de la página de video.

BH=7 : atributo de video normal

Después que se borra la pantalla se pone el cursor en la posición 0,0.

3. La función para ubicar el cursor en la fila x y columna y usa los siguientes valores:

AL=2 : ubicar cursor

DH, DL: contiene la nueva posición

BH=0 : para indicar la página actual.

48 PROGRAMA: Enlace con subrutinas escritas en lenguaje ensamblador

El siguiente programa en Lenguaje C Microsoft C, Lattice C o Turbo C llama a las subrutinas cls y cursor escritas en lenguaje ensamblador.

```
            /*  Programa  PRG.C   */

main()
{
cls();                  /*  borra pantalla  */
cursor(10,20);          /*  ubica cursor en 10,20  */
printf(" FIN DE PROGRAMA\n");
}
```

En estas versiones de C las llamadas a las subrutinas son cercanas o NEAR .

La primera subrutina cls no tiene parámetros. La segunda pasa dos parámetros por valor, esto es, en la pila se encontrarán los valores pasados 10 y 20. En C se apilan los parámetros empezando por el último. Así, en este caso primero se apila 20 y luego 10.

También se pueden pasar parámetros por referencia, es decir los valores pasados son direcciones en donde se localizan los datos.

Presentamos el programa en lenguaje ensamblador, al que llamaremos VIDEO.ASM, para ser enlazado con programas en Turbo C:

```
_TEXT  segment byte public 'CODE'
assume  cs:_TEXT
public   _cls,  _cursor      ; símbolos públicos
_cls    proc near            ; subrutina cercana _cls
        push bp
        mov ah,6
        mov al,0
        mov cx,0
        mov dx,184Fh
        mov bh,7
        int 10h
        pop bp
        ret
_cls    endp

_cursor proc near            ; subrutina cercana _cursor
        push bp              ; salva bp para acceder a la pila
        mov  bp,sp
        mov  dh,[bp+4]       ; número de fila = parám. 1
        mov  dl,[bp+6]       ; número de columna = parám. 2
        mov  bh,0
        mov  ah,2
        int  10h
        pop  bp              ; desapila valor inicial de bp
        ret
cursor  endp

_TEXT ends
      end
```

PROCEDIMIENTO PARA OBTENER LA VERSION EJECUTABLE PRG.EXE USANDO TURBO C

1) Compile PRG.C y obtenga PRG.OBJ

2) Compile VIDEO.ASM con MASM y obtenga VIDEO.OBJ

3) Obtenga PRG.EXE respondiendo con:

   CØS PRG VIDEO

   a la pregunta de módulos de objetos Object Modules ,

   y con: PRG

   para el programa ejecutable.

NOTA

Si se trabaja con los compiladores Microsoft C o Lattice C se deben hacer las siguientes modificaciones:

1) En el texto del programa VIDEO.ASM cambiar:

   _cls por cls
   _cursor por cursor

2) En el procedimiento para obtener la versión ejecutable cambiar: CØS por CS

49 PROGRAMA FECHA

El siguiente programa utiliza la función para obtener la fecha dada por el computador en la forma:

fecha()

cuyos valores son apuntadores a una estructura de datos compuesta por

```
char *nd ;   /*  nombre de día de semana  */
char d   ;   /*  número de día            */
char *nm ;   /*  nombre de mes            */
int  a   ;   /*  número de año            */
```

El texto del programa es:

```
struct sfecha  { char  *nd, d,  *nm;
                 int   a; };
main()

{ struct sfecha *hoy, *fecha();

 hoy=fecha();
 printf("La fecha de hoy es :");
 printf("%s, %d de %s de %d\n",hoy->nd,hoy->d,hoy->nm,hoy->a);
}

struct sfecha *fecha()
 {  struct { char al, ah, bl, bh;
             int  cx;
             char dl, dh;
             int si,  di, cf,  f;}    reg;

   static struct sfecha t;  /* variable para valor de fecha */

   static char *nombre_dia[]= {"Domingo", "Lunes",
     "Martes", "Miércoles","Jueves","Viernes",Sábado"};

   static char *nombre_mes[]= {"enero","febrero",
     "marzo","abril","mayo","junio","julio","agosto",
     "setiembre","octubre","noviembre","diciembre"};


    reg.ah=0x2a;
    int86(0x21,&reg,&reg);

    t.nd=nombre_dia[reg.al];
    t.d=reg.dl;
    t.nm=nombre_mes[reg.dh];
    t.a=reg.cx;

    return &t;

 }
```

COMENTARIO

1. La función fecha() toma valores apuntadores de tipo struct sfecha.

2. La variable hoy recibe el valor de la función fecha().

3. El valor de fecha() es igual a la dirección de su variable local t, que se declara estática a fin de preservar su contenido.

4. Los nombres de los días de semana y también de los meses, se almacenan en los arreglos de apuntadores a caracteres nombre_dia y nombre_mes, que se declaran estáticos para inicializarlos durante el proceso de compilación y mantenerlos durante corrida.

5. Se ha empleado la interrupción 21h (0x21) con el valor AH=0x2A, para leer la fecha. Los valores devueltos en los registros del procesador 8086 son:

   AL = día de semana (0=domingo, ..., 6=sábado)
   CX = año (1980-2099)
   DH = mes (1-12)
   DL = dia (1-31)

50 PROGRAMA IMPRIME DIRECTORIO

Este programa muestra una función

```
dir("nombre de directorio")
```

que sirve para hacer un listado en pantalla de los archivos con atributo normal (lectura y escritura) del directorio cuyo nombre se especifique.

La función acepta como argumento un apuntador a caracteres que apunta el nombre del directorio, el cual puede contener los caracteres * y ? como en el sistema operativo DOS.

El valor de la función es un apuntador a caracteres que apunta al nombre del siguiente archivo o el valor 0, cuando la búsqueda concluye.

```
main()
{ char ndir[13], *p, *dir();
  printf("Ingrese nombre de directorio ");
  scanf("%13s",ndir);
  while ((p=dir(ndir))>0)    printf("%s\n",p);
}

char *dir(pn)
char *pn;
{

    static char proseguir=0;

    static struct {  char al , ah;
                     char *bx;
                     int  cx;
                     char *dx;
                     int  si, di, cf, f; } reg;

    static char tdisco[43];

    if (proseguir==0)
      {

       /*  establecer dirección de transferencia de
                disco=tdisco  */

        reg.ah=0x1a;
        reg.dx=tdisco;
        int86(0x21,&reg,&reg);

        reg.dx=pn;     /*  buscar primer archivo  */
        reg.ah=0x4e;
        reg.cx=0;      /*  atributo normal  */
       int 86(0x21,&reg,&reg);
        if (reg.cf==0)
            { proseguir=1; return(tdisco+30); }
        else  return 0;
      }


    else              /*  buscar siguiente  */

      { reg.ah=0x4f;
        int86(0x21,&reg,&reg);
        if (reg.cf==0)   return(tdisco +30);
        else { proseguir=0; return 0;}
       }

}
```

COMENTARIO

1. Para encontrar el primer archivo cuyo nombre coincida con el nombre especificado se utiliza primero la interrupción 0x21 con:

   AH=0x4e (función "buscar el primer archivo"),

   DS:DX= la dirección del nombre del directorio seleccionado,

   CX=0 (atributo de archivo normal)

   Resultado:

   Si la operación es correcta no hay acarreo. En este caso, el nombre del archivo junto con otros datos se localiza en la región de transferencia de disco, que se compone de 43 bytes y que puede ser establecida por el usuario (Como se verá a continuación).

   El nombre del archivo se halla en esta región a partir de la posición 30.

2. Los nombres de los siguientes archivos se encuentran empleando la interrupción 0x21 con:

   AH=0x4f (función "buscar siguiente archivo")

   cuyo resultado es idéntico al de la función anterior.

3. La función dir() contiene una variable estática, llamada proseguir, cuyo valor es 0 si es la primera vez que la función es llamada, ó 1, si se trata de llamadas posteriores, a fin de considerar los dos casos mencionados antes.

4. En el programa se establece la dirección de la región de transferencia de disco igual a la dirección del arreglo estático tdisco, compuesto por 43 caracteres. Para ello se utiliza la interrupción 0x21 con:

AH=0x1a (función "establecer dirección de transferencia de disco")

DS:DX = tdisco (valor de nueva dirección)

5. Cada vez que se obtiene un nuevo nombre de archivo, éste se encuentra en la dirección tdisco+30, que es el valor que devuelve la función dir().

INDICE ALFABETICO

Lenguaje de programación "C" de Maynard Kong, se terminó de imprimir el mes de marzo de 1988 en los talleres de Editorial e Imprenta Desa (Reg. Ind. 16521) General Varela 1577, Lima 5, Perú. La edición estuvo al cuidado de *Miguel Angel Rodríguez Rea.* Se hicieron mil quinientos ejemplares

BIBLIOTECA UNIVERSITARIA DEL FONDO EDITORIAL
DE LA PONTIFICIA UNIVERSIDAD CATOLICA DEL PERU

- *El sistema jurídico (Introducción al Derecho)* de Marcial Rubio Correa. (Colección de Textos Jurídicos). 4a. ed.

- *Física básica; teoría y problemas* de Hugo Medina Guzmán. (Agotado).

- *Derecho Constitucional General* de Carlos Blancas Bustamante y Marcial Rubio Correa. (Colección de Textos Jurídicos) 2a. ed.

- *Lenguaje de programación Pascal* de Maynard Kong. 3a. ed.

- *Lenguaje de programación* "C" de Maynard Kong.