CAPÍTULO 4

FUNCIONES INTRÍNSECAS DE C

Gran parte de las funciones intrínsecas de C que se estudiarán en este capítulo son aplicables a cadenas de caracteres. Las cadenas se han analizado con profundidad en las secciones anteriores pero se ofrece a continuación un resumen que permite asentar los conceptos más importantes.

4.1 Cadenas de caracteres

El lenguaje C no tiene datos predefinidos del tipo cadena de caracteres (*string*). Una cadena de caracteres es un tipo de dato compuesto por un vector de elementos de tipo carácter (`char`) y terminado por un carácter nulo (`'\0'`).

Ejemplo de cadena: `La cadena de test\0`

Declaración de variables cadena: se declaran como un vector de tipo `char`.

Ejemplo:
```
char texto[81];
char orden[40];
```

Inicialización: se realiza en la sentencia de declaración utilizando doble apóstrofo a los lados. El programador no debe añadir el carácter '\0' ya que lo hace el propio programa.

Ejemplo:
```
char texto[81]="Esto es una cadena";
char orden[40]="Esto es una cadena muy larga";
```

Así, el siguiente programa muestra que el carácter nulo se añade a la cadena:

```
#include<stdio.h>
void main ( )
{
  int i;
  char s[ ]="ABCD";
  for(i=0;i<5;i++)
      printf("s[%d]=%c,codigo %d  \n",i,s[i],s[i]);
}
```

La salida a pantalla es:

```
s[0]=A,codigo 65
s[1]=B,codigo 66
s[2]=C,codigo 67
s[3]=D,codigo 68
s[4]=  ,codigo 0
```

4.2 Funciones para la manipulación de cadenas de caracteres. Las librerías string.h y stdlib.h

Algunas de estas funciones ya han sido empleadas en capítulos anteriores, aunque en esta sección se aporta una definición más estricta y se explica el valor que devuelven, cuando ese dato se considere importante.

La nomenclatura que se utiliza para la definición de las funciones intrínsecas, en este apartado y en los posteriores, es la indicada a continuación:

```
tipo nombre(tipo_1 arg_1, tipo_2 arg_2, ...)
```

donde:

tipo: es el tipo de datos del valor devuelto por la función
nombre: es el nombre de la función
tipo_1: es el tipo de datos del primer argumento de la función
arg_1: es el nombre del primer argumento de la función
tipo_2: es el tipo de datos del segundo argumento de la función
arg_2: es el nombre del segundo argumento de la función

Los argumentos o parámetros de una función se considerarán, a este nivel, como sus datos de entrada. Se profundizará en esta cuestión en el capítulo 7.

La llamada a la función intrínseca se realiza en cualquier parte del código del programa, de la forma:

```
nombre( arg_1,  arg_2, ...)
```

insertándose en ese lugar el valor devuelto por la función.

4.2.1 Escritura de cadenas

Para escribir una cadena se utiliza la función puts ( ) cuya especificación es:

```
int  puts(char  *cad)
```

puts( ) recibe como parámetro una cadena (en realidad una cadena es una constante puntero de tipo char, de ahí la definición char * cad, aunque es un concepto que no se entenderá hasta que se analice el capítulo 6) y devuelve un entero. Añade al final de la cadena un carácter de salto de línea. Devuelve un número positivo o cero si todo va bien y un –1 si hay algún error.

Ejemplo: el programa

```
#include<stdio.h>
 void main( )
 {
  char texto[81]="Esto es una cadena";
  puts(texto);
 }
```

produce la siguiente salida por pantalla: Esto es una cadena.

De la misma forma se puede utilizar la función printf( ),

int printf(char *cad)

Ambas, son funciones de la librería stdio.h

4.2.2 Lectura de cadenas

Método 1: usando la función **scanf( )**, que sólo lee datos delimitados por blancos.

int scanf(char *cadena de control, char *cadena)

Esta función recibe, al menos, dos parámetros de cadena (punteros) y devuelve un entero.

Ejemplo:

```
#include <stdio.h>
 void main ( )
 {
   char nombre[30];
   scanf( "%s" , nombre) ;
   puts(nombre) ;
 }
```

En este caso, si se teclea como dato Pedro imprime como salida Pedro, pero si se teclea Pedro López la salida es también Pedro.

Método 2: para evitar la lectura fraccionada se utiliza la función **gets( ).**

char *gets (char *cad)

Esta función recibe como parámetro un puntero y devuelve un puntero.

Ejemplo:

```
#include<stdio.h>
void main( )
{
  char nombre[30];
  gets(nombre) ;
  puts(nombre) ;
}
```

Si se teclea como dato Pedro Sanz el programa ofrece como salida Pedro Sanz.

Método 3: el siguiente método consiste en leer carácter a carácter. Para ello se utiliza la función getchar( ). Su especificación es:

int getchar(void)

Esta función devuelve el carácter leído y no tiene argumento de entrada.

Para escribir un carácter se utiliza la función putchar( ) cuya especificación es:

int putchar(int ch)

Esta función recibe como argumento el carácter a escribir y devuelve un entero.

Ejemplo 1: car= getchar( ); equivale a scanf("%c",&car);

Ejemplo 2: putchar(car); equivale a printf("%c",car);

Ejemplo 3:

```
#include<stdio.h>
 void main( )
 {
  char nombre[30];
  int i=0 ;
  while((nombre[i]=getchar( )) != '\n')  i=i+1;
  nombre[i]=0;
  puts(nombre);
 }
```

Si se teclea cantares da como resultado cantares. En la séptima sentencia se podría haber escrito: nombre[i]='\0';

Todas las funciones anteriores están incorporadas a la librería stdio.h

4.2.3 La libreria string.h

La biblioteca estándar de C contiene la biblioteca de funciones para tratamiento de cadenas llamada string.h que incorpora las funciones de manipulación de cadenas utilizadas más frecuentemente.

Algunas de la funciones más usuales son:

strlen( ): devuelve el número de caracteres de una cadena (longitud real de la cadena hasta la posición anterior al carácter nulo).

Su especificación es:

int strlen(char *cad)

Recibe como argumento un puntero (cadena) y devuelve un entero.

Ejemplo:

```
#include<stdio.h>
#include<string.h>
#define MAX 80
void main( )
{
 char cad[MAX];
 int numcar;
 puts ("Dame una cadena");
 gets(cad);
 numcar=strlen(cad);
 printf("La cadena tiene %d caracteres \n",numcar);
 }
```

Por ejemplo si se teclea COSACOS CANSADOS el programa imprime como salida 16.

strcat( ): su finalidad es concatenar dos cadenas de caracteres.

char * strcat (char*cad1,char*cad2)

Esta función añade la cadena cad2 al final de la cadena cad1 (a continuación del carácter previo al nulo de cad1).

Ejemplo:

```
#include<stdio.h>
#include<string.h>
 main( )
 {
 char cad1[25]="El lenguaje C";
 char cad2[12]="es potente";
 printf("%s \n",strcat(cad1,cad2));
 }
```

Este programa imprime la siguiente salida: El lenguaje Ces potente

strcpy( ): esta función copia en cad1 el contenido de cad2 y devuelve cad1.

char * strcpy (char*cad1,char*cad2)

Ejemplo:

```
#include <stdio.h>
#include<string.h>
 void main( )
{
 char cad1[13]="Lenguaje C";
 char cad2[13]="Lenguaje ADA"
 printf("%s  \n",strcpy(cad1,cad2));
 }
```

Este programa imprime la siguiente salida por pantalla: Lenguaje ADA

Si strlen(cad1) > strlen(cad2), al copiar cad2 en cad1 se incluye el carácter '\0' y por tanto los restantes elementos de cad1 no aparecen.

Ejemplo:

```
#include <stdio.h>
#include<string.h>
main( )
{
 char cad1[13]="Lenguaje C";
 char cad2[13]="OK"
 printf("%s  \n",strcpy(cad1,cad2));
}
```

Este programa imprime la siguiente salida por pantalla: OK

4.2.4 Funciones para la conversión de caracteres

Dentro de la librería stdlib.h se encuentran las siguientes funciones de conversión:

tolower( ): Convierte su argumento en una letra minúscula si procede.

int tolower(int c)

toupper( ): Convierte su argumento en una letra mayúscula si procede.

int toupper(int c)

4.3 Funciones de tipo matemático. Las librerias math.h y stdlib.h

Las siguientes funciones se encuentran en la librería indicada por la sentencia

#include <math.h>

sin( ): **double sin (double x)**

Da como resultado el seno de x (x en radianes).

cos( ): **double cos (double x)**

Da como resultado el coseno de x (x en radianes).

tan( ): **double tan (double x)**

Da como resultado la tangente de x (x en radianes).

asin( ): **double asin (double x)**

Da como resultado el arco, en el rango $-\pi/2$ a $\pi/2$, cuyo seno es x.

acos( ): **double acos (double x)**

Da como resultado el arco, en el rango 0 a π, cuyo coseno es x.

atan( ): **double atan (double x)**

Da como resultado el arco, en el rango $-\pi/2$ a $\pi/2$ cuya tangente es x.

exp( ): **double exp (double x)**

Da como resultado el valor de e^x.

log( ): **double log (double x)**

Da como resultado el valor del logaritmo neperiano de x.

log10( ): **double log10 (double x)**

Da como resultado el valor del logaritmo en base 10 de x.

fabs( ): **double fabs (double x)**

Da como resultado el valor absoluto de x.

pow(x,y): **double pow (double x, double y)**

Da como resultado el valor de x^y.

sqrt( ): **double sqrt (double x)**

Da como resultado la raíz cuadrada de x. Si x es negativo, ocurre un error.

floor( ): **double floor (double x)**

Da como resultado el mayor entero que es menor o igual que x.

Ejemplo: **floor**(3.141593)=3.

ceil( ) : **double ceil (double x)**

Da como resultado el menor entero que es mayor o igual que x.

Ejemplo: **ceil**(3.141593)=4.

Las siguientes funciones se encuentran en la librería indicada por la sentencia

#include <stdlib.h>

rand( ): **int rand (void x)**

Da como resultado un número pseudoaleatorio entre 0 y 32767.
(Los números pseudoaleatorios son generados por medio de una función determinista (no aleatoria) y aparentan ser aleatorios. Se generan a partir de un valor inicial (semilla) aplicando iterativamente la función).

srand( ): **void srand (int arg))**

Fija el punto de comienzo para la generación de números pseudoaleatorios, en otras palabras, inicializa el generador de números pseudoaleatorios en función del valor de su argumento.

Ejemplo:

```
#include<stdio.h>
#include<stdlib.h>
main( )
{
int i;
printf("Introduce la semilla \n");
scanf("%d",&i);
srand(i);
printf("%d %d %d \n", rand( ),rand( ), rand( ));
}
```

Si en la primera ejecución del programa, se introduce como semilla 15, el resultado sería:

27056 5105 87

Si en la segunda ejecución del programa, se introduce como semilla 100, el resultado sería:

5415 1216 365

Si en la tercera ejecución del programa, se introduce como semilla 1000, el resultado sería:

26849 8221 3304

4.4 Enunciados de las prácticas

Ejercicio 1.

Utilizando las funciones srand y rand, diseñar un programa C que obtenga como salida una columna de cinco números aleatorios entre 0 y 1, que puedan ser distintos cada vez que se ejecute el programa (mediante la modificación de la semilla).

Ejercicio 2.

Escribir la salida a pantalla del siguiente programa C:

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

void main(  )
{
    double t,v;
    printf("t \t \t v \n" ) ;
    for(t=0.0; t<=3.1; t=t+0.5)
    {
       v=100*(1-exp(-2*t)) ;
       printf("%f \t %f  \n",t,v) ;
    }
}
```

Ejercicio 3.

El siguiente programa C cuenta las veces que aparece la letra *t* en el flujo de entrada. Ejecutarlo con diferentes flujos de entrada.

```
#include<stdio.h>

void main( )
{
   int car;
   int cuenta=0 ;
   while((car=getchar( )) != '\n')
   if(car== 't')    ++cuenta;
   printf("\n %d letras  t  \n", cuenta);
}
```

NOTA: Al término del flujo de entrada, teclear INTRO .

Ejercicio 4.

El siguiente programa hace "eco" del flujo de entrada y convierte cada palabra en una palabra idéntica que comienza con una letra mayúscula. Analizar el significado de cada instrucción y ejecutarlo con diferentes flujos de entrada.

```
#include<stdio.h>
#include<stdlib.h>
void main( )
{
    char car, pre='\n';
    while ((car=getchar( )) != '\n')
    {
        if(pre == ' '  ||   pre == '\n')
        {
            putchar(toupper(car));
            pre=car;
         }
        else
        {
           putchar(car);
           pre=car;
         }
    }
}
```

Ejercicio 5.

Escribir un programa C que lea una cadena de caracteres (de 80 caracteres como máximo) y que convierta en mayúsculas las letras minúsculas que aparecen en dicha cadena.
En este caso se pide realizar el ejercicio sin utilizar la función toupper( ).

4.5 Soluciones a las prácticas del capítulo 4

Ejercicio 1.

Una solución válida es la siguiente:

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

void main( )
{
   int  i,seed;
   double x;
   printf("Introduce el valor de la semilla:");
   scanf("%d", &seed);
   srand(seed);
   for(i=1;i<=5;i++)
   {
      x=rand( )/32767. ;
      printf("%d \t %f \n",i,x) ;
   }
}
```

Ejemplos de salida a pantalla:

Para seed = 439

1 0.044923
2 0.234291

Para seed = 5000

1 0.499466
2 0.312235

3	0.978332	3	0.504379
4	0.455794	4	0.844661
5	0.456832	5	0.149388

Ejercicio 2.

La salida a pantalla es la siguiente:

t	v
0.000000	0.000000
0.500000	63.212056
1.000000	86.466472
1.500000	95.021293
2.000000	98.168436
2.500000	99.326205
3.000000	99.752125

Ejercicio 3.

Si se teclea, por ejemplo:

tetraedro y triangulo INTRO

aparece en pantalla:

3 letras t

Ejercicio 4.

Si se teclea:

la casa del sol naciente INTRO

aparece:

La Casa Del Sol Naciente.

Si se teclea:

El amanecer en marte INTRO

aparece:

El Amanecer En Marte

Ejercicio 5.

Un programa válido es el siguiente:

```
#include<stdio.h>
#define LONG_MAX 81
void main( )
{
    char cadena[LONG_MAX];
    int i=0, desp='a'-'A';
    printf("Introducir una cadena: \n");
    gets(cadena);
    for(i=0;cadena[i] != '\0';i++)  /*también se puede utilizar como condición
                                      de este bucle: i<strlen(cadena), incluyendo
                                      la librería string.h*/
   {
       if(cadena[i]>='a' && cadena[i]<= 'z')
                cadena[i]=cadena[i]-desp;
   }
    printf("%s \n", cadena);
}
```

NOTA: Como el valor ASCII de 'a' es 97 y el de 'A' es 65, se tiene que:

desp=’a’-‘A’=32=’b’-‘B’=’c’-‘C’ .....= ‘z’-‘Z’ es la diferencia en código entre la letra minúscula y su correspondiente mayúscula.
Entonces, si por ejemplo, se teclea:

La casa de todos

aparece en pantalla :

LA CASA DE TODOS