

## Cadenas de caracteres

Todos los lenguajes de programación disponen de algún método para tratar las cadenas de caracteres (en inglés string), ya que frecuentemente un programa tiene que trabajar con nombres, descripciones, códigos alfanuméricos, etc.

Mediante las cadenas de caracteres podemos representar palabras, y realizar operaciones sobre ellas.

En C no hay un tipo de dato primario para las cadenas. C dispone del tipo de dato primario char, que permite almacenar 1 caracter. Por ejemplo:

```
int main()
{
    char letra, numero, otra;
    letra='a';
    numero='1';
    cin>>otra;
    cout<< letra;
    cout<<numero;
    if(letra!=numero) cout<<"\nLetra y numero son distintos";
    cout<<otra;
    return 0;
}
```

A la variable letra se le asignó el caracter 'a', y a numero el caracter '1'. Nótese que para representar un caracter, éste debe estar entre comillas simples. De esta forma se diferencia al 1 como número, que se puede almacenar en una variable int, y al '1' como caracter. Además permite que el compilador distinga a una letra, que puede representar a una variable, de la misma letra utilizada como un caracter.

El tipo de dato char ocupa 1 byte de memoria. Internamente C almacena el char con el valor numérico equivalente en el código ASCII. Por ejemplo:

```
.....
char letra;
letra='a';
cout<<letra;
cout<<(int)letra;
.....
```

si se imprime el contenido de la variable letra con formato caracter, se imprimirá una a; en cambio si se imprime en formato entero se verá un 97, que es el valor ASCII equivalente para la a minúscula.

Como tipo de dato primario, con las variables de tipo char se pueden hacer el conjunto de operaciones que con las demás variables numéricas (obviamente las operaciones matemáticas no son aplicables). Básicamente lo anterior significa que podemos asignar directamente un caracter a una variable char, y aplicarle los operadores relacionales igual que a las variables numéricas. También se pueden utilizar las funciones de entrada y salida estándar.

Del mismo modo que con las variables numéricas, es posible declarar vectores de char.

Para trabajar con cadenas de caracteres, C usa vectores de char. Por ejemplo, para almacenar una palabra de 10 caracteres se tendría que declarar un vector de la siguiente manera:

```
char palabra[11];
```

Ahora ¿por qué un vector de 11 elementos, si se quiere guardar 10 letras?. La razón es que para indicar el fin de una cadena, C exige un caracter “terminador” especial, el cual será luego reconocido por las funciones que tratan las cadenas. Este caracter es el caracter nulo ‘\0’.

Al igual que con los vectores numéricos, C no controla si el programa se excede en la cantidad de elementos del vector: es responsabilidad del programador. Por ejemplo, si al vector palabra se le ingresan 15 caracteres, el compilador no dará ningún mensaje de error. Sin embargo, puede ser que cuando se ejecute el programa se produzca un error, ya que se ocuparon posiciones de memoria no reservadas para el programa. Que el error ocurra o no, depende si las posiciones adicionales de memoria (desde la posición 11 a la 14 del vector) están asignadas o no al programa.

En cualquier caso, siempre hay que definir el vector de char del tamaño máximo necesario, y luego sumar 1 para el caracter terminador. Por ejemplo, si se quiere almacenar en una cadena el nombre de un mes del año, y no se sabe de antemano cuál de los meses será, habrá que analizar primero cual es el mes que más letras tiene, y luego sumarle 1. Para este caso, el vector será char mes[10]. Si se almacena un mes con menor cantidad de letras que 9, C pondrá luego del último caracter ingresado el caracter nulo, de la siguiente manera:

‘N’	‘O’	‘V’	‘I’	‘E’	‘M’	‘B’	‘R’	‘E’	‘\0’
-----	-----	-----	-----	-----	-----	-----	-----	-----	------

‘n’	‘o’	‘v’	‘i’	‘e’	‘m’	‘b’	‘r’	‘e’	‘\0’
-----	-----	-----	-----	-----	-----	-----	-----	-----	------

‘A’	‘b’	‘r’	‘i’	‘l’	‘\0’				
-----	-----	-----	-----	-----	------	--	--	--	--

Trabajar con cadenas, entonces, es trabajar con vectores de char, por lo tanto no es válida la siguiente asignación:

```
mes="abril";
```

Tampoco es válida la siguiente proposición lógica:

```
if(mes= "Abril")
```

Para este tipo de acciones, será necesario hacer funciones propias, o utilizar las funciones que C dispone para trabajar con cadenas. Todas estas funciones reciben la dirección de inicio de la cadena (el nombre del vector), y marcan su final con el carácter nulo.

Lo que si es posible es asignar cadenas literales al declarar las variables, de las siguientes maneras:

```
char cadena[5]="hola";
```

```
char cadena2[ ]="hola";
char* cadena3="hola";
```

En todos los casos el compilador creará un vector de char de 5 caracteres que contendrá la cadena "hola", y el carácter terminador '\0'. En el segundo caso se omitió dentro de los corchetes la dimensión, ya que cualquiera que sea el número que allí se coloque el vector se ajustará al tamaño estrictamente necesario para almacenar los caracteres que se asignen.

La última inicialización, en la que se usa un puntero, puede ser tratada por algunos compiladores de manera tal que no permita su modificación. No debe utilizarse entonces si en alguna parte del código es necesario modificar el contenido de la cadena.

También puede asignarse directamente a un puntero char un vector de char. En este caso, lo que se asigna es la dirección que contiene el nombre del vector al otro puntero.

```
.....
char cad[10], *pcad;
cin>>cad;
pcad=cad;
cout<<pcad<<endl; // se mostrará lo que contiene la cadena cad
.....
```

## Funciones de cadenas

Como sabemos toda la actividad de un programa en C se desarrolla por medio de funciones. También sabemos que esas funciones pueden ser de la librería de C, o creadas por el programador. En esta sección analizaremos algunas de las funciones de entrada y salida para cadenas, y otras de la librería string.h de C. Todas estas funciones pueden utilizarse en el Code::Block sin necesidad de agregar la librería a la que pertenecen.

### Funciones de entrada y salida

Para almacenar en memoria una cadena se puede utilizarse la función scanf() con el especificador "%s". Por ejemplo:

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

capturará los caracteres que se ingresen por teclado hasta que se presione la tecla Enter. Al encontrar el valor correspondiente a la tecla Enter, se dará por concluida la entrada y se colocará automáticamente el carácter nulo al final de la cadena.

scanf() no acepta espacios en blanco. Si se ingresa un espacio en blanco, por ejemplo para separar un nombre de un apellido, scanf() pondrá el terminador en el lugar del espacio en blanco.

Nótese que no hace falta poner el & antes de la variable donde se quiere almacenar la cadena. La razón es que mes es el nombre de un vector, y como tal es la dirección de inicio de la cadena.

Otra función para la entrada es gets(). La sintaxis es:

```
gets(mes);
```

gets() requiere como parámetro la dirección de inicio de la cadena (esto es el nombre del vector de char declarado). Acepta espacios en blanco como parte de la cadena. Está incluida en stdio.h. Tampoco controla la cantidad de caracteres ingresados.

Para imprimir por pantalla se puede usar printf() con el especificador “%s”, o la función puts(), de la siguiente manera:

```
printf(“%s”, mes);  
puts(mes);
```

La sintaxis de printf() puede parecer extraña, ya que se le pasa como parámetro una dirección (mes). Sin embargo, al encontrar el especificador “%s” C interpreta que se trata de una cadena, y muestra todos los caracteres que hay desde la dirección mes, hasta que encuentra el caracter nulo.

puts() requiere como parámetro la dirección de inicio de la cadena. También puede utilizarse puts para imprimir una cadena definida directamente dentro de los paréntesis. Por ejemplo:

```
puts(“Este es un cartel que se imprimirá en la pantalla”);
```

Para definir una cadena, el texto debe ponerse entre comillas dobles. puts() está incluido en stdio.h

### **Funciones de la librería string.h**

Algunas de las funciones de uso más frecuente son:

```
strcpy(cadena1, cadena2);
```

strcpy() copia el contenido de la cadena2 en la cadena1, y es la función que debe utilizarse para asignar una cadena a otra cadena. Por ejemplo:

```
#include <iostream>
```

```
using namespace std;
```

```
int main()  
{  
    char cadena[5];  
    strcpy(cadena, “Hola”);  
    puts(cadena);  
    return 0;  
}
```

el programa imprimirá “Hola”.

O

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
{
    char cadena1[5], cadena2[5];
    gets(cadena1);
    strcpy(cadena2, cadena1);
    puts(cadena1);
    return 0;
}
```

### **int strcmp(cadena1, cadena2)**

strcmp() se utiliza para comparar si una cadena es “mayor”, “menor” o “igual” que otra. Que una cadena sea mayor o menor que otra, se evalúa de acuerdo al orden lexicográfico de las cadenas, es decir, su posición relativa en el diccionario.

La función devuelve los siguientes valores:

- 0, si ambas cadenas son iguales.
- <0, si cadena1 es menor que cadena2.
- >0, si cadena1 es mayor que cadena2.

Por ejemplo:

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
{
    char cadena1[5], cadena2[5];
    int valor_devuelto;
    strcpy(cadena1, "Hola");
    strcpy(cadena2, "Chau");
    valor_devuelto=strcmp(cadena1,cadena2);
    if(valor_devuelto>0)
        puts("La cadena 1 está después en el diccionario que la cadena 2");
    else
        if(valor_devuelto<0)
            puts("La cadena 1 está antes en el diccionario que la
cadena 2");
        else
            puts("La dos cadenas son iguales");
    return 0;
}
```

El programa informará “La cadena 1 está después en el diccionario que la cadena 2”.

Se puede colocar directamente a strcmp() dentro del if de la siguiente manera:

```
.....  
if(strcmp(cadena1,cadena)= = 0)
```

.....  
En este caso, C ejecutará primero la función strcmp(), y evaluará luego el valor devuelto con la condición. Obviamente se puede utilizar cualquier operador relacional (>, <, !=, etc.) en esta operación.

### Otras funciones:

**int strlen(char \*):** devuelve la cantidad de caracteres de la cadena, sin contar el caracter nulo.

**strcat(cadenafinal, cadena):** agrega a cadenafinal el contenido de cadena.

**strlwr(cadena):** convierte todos los caracteres de cadena a minúsculas.

**strupr(cadena):** convierte todos los caracteres de cadena a mayúsculas.

Además de las funciones, se puede utilizar para el ingreso y la salida por pantalla **cin>>** y **cout<<**. Debe tenerse en cuenta que **cin>>** no admite espacios, esto es, más de una palabra en una misma cadena. Al encontrar un espacio cin colocará el carácter nulo.

Para almacenar cadenas con espacios puede usarse cin.getline(char \*cad, int longitud). Esta función tiene como parámetros la cadena sobre la que se quiere escribir, más la longitud de la cadena. Si se ingresan más caracteres que los definidos en el parámetro longitud serán ignorados. Por ejemplo:

```
int main()  
{  
    char cadena1[5];  
    cin.getline(cadena1,4); //sólo se aceptarán 4 caracteres  
    cout<<cadena1;  
    return 0;  
}
```

Nota: cin.getline() puede no funcionar si antes de su ejecución hubo otra instrucción que requirió que se presione la tecla Enter. Para evitar este problema se puede anteponer a cin.getline() la instrucción cin.ignore() o fflush(stdin).

### “Vectores” de cadenas

Como sabemos, un vector es una estructura de datos que nos permite agrupar bajo un mismo nombre de variable un conjunto de datos de un mismo tipo, y acceder a cada elemento individual –para leer o escribir- utilizando el subíndice correspondiente. De

esta manera, la programación resulta más sencilla y eficiente que si tuviéramos que trabajar con variables simples “sueltas”.

Al igual que para los tipos de datos int, float o char, poder almacenar un conjunto de cadenas dentro de una sola estructura de datos puede resultar de mucha utilidad para el desarrollo de programas; pero a diferencia de los tipos de datos primitivos una cadena ya es un vector, por lo que necesitaríamos no sólo una dimensión que indique el largo de la cadena, sino otra dimensión más para indicar cuántas cadenas queremos guardar. Veámoslo gráficamente:

‘N’	‘O’	‘V’	‘I’	‘E’	‘M’	‘B’	‘R’	‘E’	‘\0’
‘n’	‘o’	‘v’	‘i’	‘e’	‘m’	‘b’	‘r’	‘e’	‘\0’
‘D’	‘i’	‘c’	‘i’	‘e’	‘m’	‘b’	‘r’	‘e’	‘\0’
‘E’	‘n’	‘e’	‘r’	‘o’	‘\0’				
‘A’	‘b’	‘r’	‘i’	‘l’	‘\0’				

En este caso el largo máximo de cada cadena es de 10 caracteres, y la cantidad máxima de cadenas que se pueden almacenar es de 5. Si analizamos la cantidad de filas y columnas, estaríamos hablando de una estructura de 5 filas y 10 columnas. Este tipo de estructura de datos se denomina **matriz** (o array bidimensional) y se declara de la siguiente manera:

```
char palabras[5][10];
```

Una matriz de dos dimensiones es una estructura de datos definida por un valor entero de filas (primera dimensión) y otro valor entero de columnas(segunda dimensión). Por cada intersección de fila y columna se puede almacenar un valor. Pueden declararse y utilizarse matrices para cualquier tipo de datos, pero este tipo será uniforme para todas los componentes o celdas de la matriz. Por ejemplo:

Una matriz de enteros de nombre m de 5 filas por 5 columnas se declararía de la siguiente manera:

```
int m[5][5]; // tipo de dato nombre[cant. de filas][cant. de columnas]
```

Su representación gráfica puede ser:

columna 0	columna 1	columna 2	columna 3	columna 4	
m[0][0]	m[0][1]	m[0][2]	m[0][3]	m[0][4]	fila 0
m[1][0]	m[1][1]	m[1][2]	m[1][3]	m[1][4]	fila 1
m[2][0]	m[2][1]	m[2][2]	m[2][3]	m[2][4]	fila 2
m[3][0]	m[3][1]	m[3][2]	m[3][3]	m[3][4]	fila 3
m[4][0]	m[4][1]	m[4][2]	m[4][3]	m[4][4]	fila 4

luego para cada celda de la matriz se pueden aplicar todas las operaciones conocidas para variables del tipo de datos definido para la matriz. Si queremos asignarle a m[0][0] un valor, la instrucción sería:

```
m[0][0]=10;
```

Si quisieramos comparar dos elementos de la matriz:

```
if(m[0][0] != m[0][1])
```

.....

En la materia Programación II profundizaremos el estudio de las matrices en general. Ahora que conocemos que son y como se definen las matrices, veamos como podemos utilizar una matriz de char para almacenar un conjunto de cadenas.

```
int main( )
{
    char palabras[5][15]; //matriz para almacenar 5 palabras de hasta 14 caracteres
    cargarPalabras(palabras);
    mostrarPalabras(palabras);
    return 0;
}
```

```
void cargarPalabras(char pal[5][15])
{
    int i;
    for(i=0;i<5;i++)
    {
        cout<<"Ingrese una palabra: ";
        cin>>pal[i];
    }
}
```

```
void mostrarPalabras(char pal[5][15])
{
    int i;
    for(i=0;i<5;i++)
    {
        cout<<"Palabra: "<<i+1<<"\t";
        cout<<pal[i]<<endl;
    }
}
```

En main() se declaró una matriz de 5 filas por 15 columnas. Como utilizaremos cada fila para una cadena, se podrán almacenar 5 cadenas; a su vez, como se definió que la cantidad de columnas es de 15, la cantidad máxima de caracteres de cada cadena será de 14 (recordar que es necesario reservar un espacio para el carácter nulo).

Luego se utilizó una función para cargar las cadenas y otra para mostrar las cadenas. En ambos casos para acceder a cada cadena se utilizó el nombre de la matriz (o del parámetro en el que se la recibió en la función) y el primer subíndice (que representa a las filas). Esto es:

.....

```
cin>>pal[0]; //se almacena en la fila 0 de la matriz una cadena
```



```
.....  
cout<<pal[2]; //se muestra la cadena de la fila 2
```

Veamos ahora como podemos hacer para buscar una cadena dentro de una matriz:

```
int buscarPalabra(char *cad, char pal[5][15])  
{  
    int i;  
    for(i=0;i<5;i++)  
        if(strcmp(pal[i],cad)= =0)  
            return i;  
    return -1;  
}
```

La función recibe una cadena (primer parámetro char \*cad), y una matriz (segundo parámetro char pal[5][15]), y debe devolver la posición que la cadena cad ocupa en la matriz pal; en caso que la cadena recibida no exista en la matriz devolverá -1 (valor absurdo, ya que la posición -1 no existe).

Por cada ciclo del for se analiza, usando la función strcmp(), si cad coincide con pal[i]; si la coincidencia existe se devuelve el valor de i. Si no se da la coincidencia se termina el for y se devuelve el valor -1. Este valor le servirá a la función que hizo la llamada para saber si la cadena existe o no, ya que -1 no es un valor de fila válido.

En resumen:

- Para trabajar con un conjunto de cadenas es necesario declarar una matriz de char, identificando la cantidad de filas (que equivaldrá a la cantidad de palabras que se deseen almacenar), y la cantidad de columnas (que determinará el largo máximo de cada cadena): nombreMatriz[cantidadFilas][cantidadColumnas] .
- Cada cadena se ubicará en una fila de la matriz, por lo que para acceder a las cadenas se utilizará el nombre de la matriz, un solo par de corchetes y el número de fila correspondiente: nombreMatriz[numeroFila]
- Sobre cada cadena contenida en nombreMatriz[numeroFila] se podrán aplicar todas las funciones de cadena conocidas.
- El tipo de dato que se utilizará para recibir una cadena como parámetro es distinto al tipo de dato con el que se recibe una matriz. Para el caso de una cadena (que es un vector de char) será char \*nombreParam; para el caso de la matriz sera char nombreMatriz[cantidadFilas][cantidadColumnas].