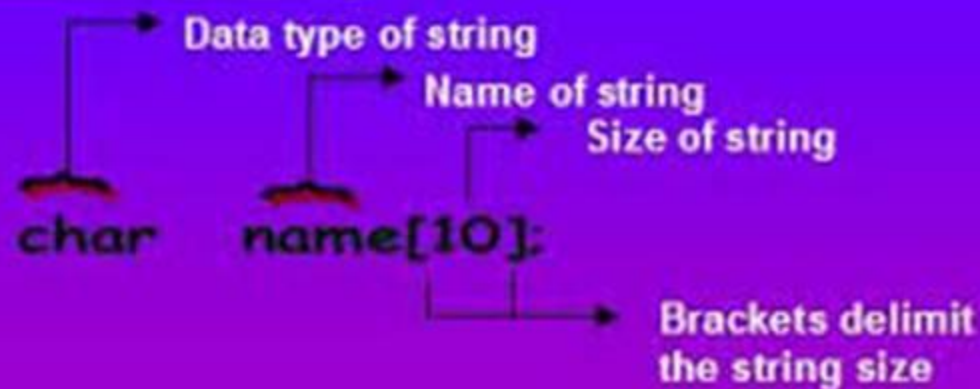




Syntax of String Definition



s1

0	1	2	3
o	n	e	\0

s2

0	1	2	3
t	w	o	\0

s3

0	1	2	3
b	a	t	\0

s4

0	1	2	3	4	5
h	e	l	l	o	\0

s5

0	1	2	3
b	y	e	\0

CADENAS EN C

Víctor Melchor Espinoza
Introducción a la Programación

Temas a Tratar

- 1. Arrays vs Cadenas de caracteres.**
- 2. Manejo de Cadenas de Caracteres**
- 3. Funciones de Caracteres**

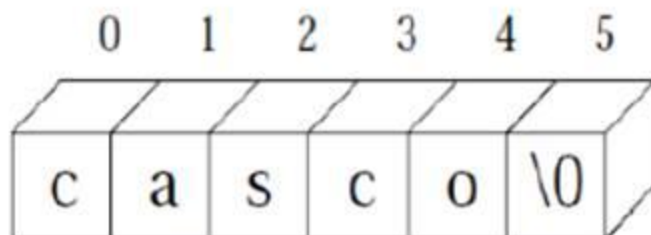
Array de caracteres

- ❖ Una cadena de texto es un conjunto de caracteres, tales como «ABCDEFGH».
- ❖ C soporta **cadenas de texto** utilizando un **array de caracteres**.
- ❖ Ejemplo
 - `Char cadena[] = "ABCDEFGH";`

Cadenas de caracteres

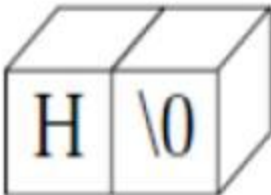
- ❖ Para que un arreglo de caracteres pueda ser considerado como una cadena de caracteres, **el ultimo de los elementos del arreglo debe ser el caracter nulo ('\\0').**
- ❖ Por ejemplo
 - `char cadena[6];`

Reserva suficiente espacio en memoria para almacenar una cadena de 5 caracteres, como la palabra "casco":



Cadenas de caracteres

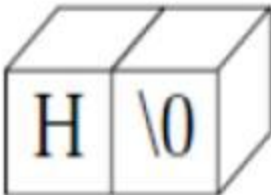
- ❖ En C pueden definirse constantes correspondientes a cadenas de caracteres. Se usan comillas dobles para delimitar el principio y el final de la cadena, a diferencia de las comillas simples empleadas con las constantes de tipo caracter.
- ❖ Por ejemplo, la cadena constante "H" tiene muy poco que ver con el caracter constante 'H'

"H" → 

'H' → 

Cadenas de caracteres

- ❖ En C pueden definirse constantes correspondientes a cadenas de caracteres. Se usan comillas dobles para delimitar el principio y el final de la cadena, a diferencia de las comillas simples empleadas con las constantes de tipo caracter.
- ❖ Por ejemplo, la cadena constante "H" tiene muy poco que ver con el caracter constante 'H'

"H" → 

'H' → 

Cadenas de caracteres - asignación

❖ Ejemplos

- `Cadena[3] = 'D';`
- `Cadena[4] = 'E';`
- `Cadena[5] = 'F';`
- `Cadena[6] = '\0';`

Sin embargo, **no se puede asignar** una cadena a un array del siguiente modo:

```
Cadena = "ABCDEFGH" ;
```


Cadenas de caracteres - asignación

- ❖ Puede asignarse cada caracter de la cadena **individualmente**. No deberá olvidarse en ningún caso que el ultimo caracter valido de la misma debe ser el caracter nulo ('\\0').

```
char cadena[10];
```

- ❖ **Ejemplo**

```
.    .    .  
cadena[0] = 'c';  
cadena[1] = 'a';  
cadena[2] = 's';  
cadena[3] = 'c';  
cadena[4] = 'o';  
cadena[5] = '\\0';
```

- ❖ El contenido del arreglo en las posiciones posteriores al caracter nulo es ignorado.

Cadenas de caracteres - asignación

- ❖ Existe un método de inicialización propio de las cadena de caracteres, cuyo formato general es:

```
char nombre [tamaño] = "cadena";
```

- ❖ Usando este tipo de inicialización, el caracter nulo es añadido automáticamente al final de la cadena.
- ❖ Ejemplo:
 - Char nombre[10]="JUAN";
 - Char nombre[]="JUAN";

Puede hacerse también de forma equivalente como:

- char nombre[10] = { 'J', 'U', 'A', 'N', '\0' };

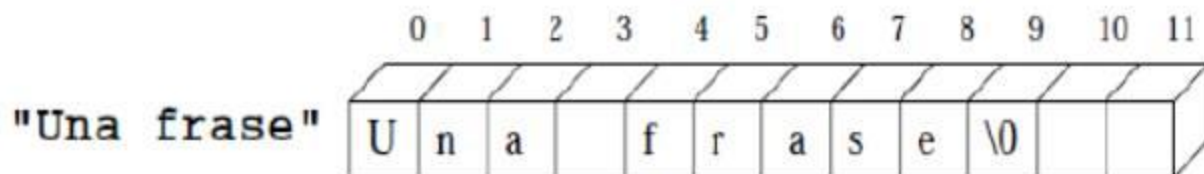
La cadena vacía

- ❖ **Consta únicamente del caracter nulo.** Puesto que los caracteres posteriores al caracter nulo son ignorados, convertir una cadena con cualquier valor almacenado a la cadena vacía es tan simple como **asignar el caracter nulo a la posición 0 de dicha cadena.**

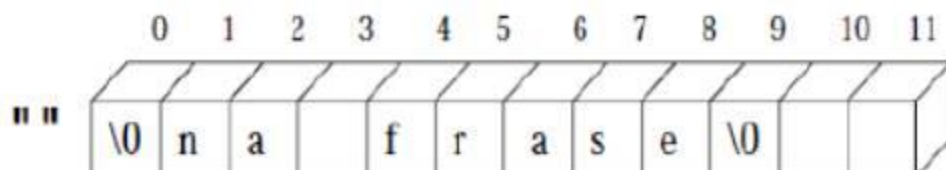
- ❖ **Ejemplo**

```
char cadena [12] = "Una frase";
```

```
cadena[0] = '\0'; /* Ahora es una cadena vacía */
```



```
cadena[0] = '\0';
```



Manejo de cadenas de caracteres

- ❖ C no incorpora en su definición operadores para el manejo de cadenas de caracteres, pero todo compilador de C proporciona una librería estándar (string.h) con funciones para facilitar su utilización.
 - **strlen**: contar el numero de caracteres de una cadena
 - **strcpy**: para copiar una cadena en otra
 - **strcat**: para concatenar dos cadenas
 - **strcmp**: para comparar dos cadenas

Función strlen

- ❖ Su nombre proviene de ***string length***, y su misión es **contar el número de caracteres de una cadena**, sin incluir el '\0' final.
- ❖ El argumento que se le pasa es el nombre de la cadena.
- ❖ Devuelve un entero sin signo que es el número de caracteres de la cadena.
- ❖ Sintaxis
 - strlen(cadena);

Función strcat

- ❖ Su nombre proviene de **string concatenation**, y se emplea para **unir dos cadenas de caracteres** poniendo s2 a continuación de s1.
- ❖ Sintaxis
 - `strcat(cadena1, cadena2);`

❖ Ejemplo

```
char nombre[] = "juan ";  
char apellido[] = "perez";  
printf("%s", strcat(nombre, apellido));
```

Función strcmp

- ❖ Sus nombres provienen de **string comparison**.
- ❖ **strcmp()** Sirve para comparar dos cadenas de caracteres. Como argumentos utiliza las cadenas que se van a comparar.
- ❖ La función **devuelve cero si las cadenas son iguales**, un valor **menor que cero** si s1 es menor en orden alfabético que s2, y un **valor mayor que cero** si s1 es mayor que s2.

Función strcpy

- ❖ Su nombre proviene de **string copy** y se utiliza para **copiar cadenas**.
- ❖ **Es muy importante tener en cuenta que en C no se pueden copiar cadenas de caracteres directamente, por medio de una sentencia de asignación.** Por ejemplo, sí se puede asignar un texto a una cadena en el momento de la declaración:

```
char s[] = "Esto es una cadena";           // correcto
```

Sin embargo, sería ilícito hacer lo siguiente:

```
char s1[20] = "Esto es una cadena";  
char s2[20];
```

```
...  
// Si se desea que s2 contenga una copia de s1  
s2 = s1;           // incorrecto: se hace una copia de punteros  
strcpy(s2, s1);    // correcto: se copia toda la cadena
```

Entrada y Salida

- ❖ En cuanto a la entrada y salida de cadenas de caracteres, existe un formato especial **%s** que puede utilizarse en las funciones scanf y printf.
- ❖ Por ejemplo, la siguiente sentencia leerá una cadena de caracteres en la variable cad. Sólo se asignarán caracteres mientras no sean caracteres blancos, tabuladores o saltos de línea. **Por lo tanto, el empleo de %s sólo tendrá sentido para la lectura de palabras.**

```
char cad[20];  
.  
.  
.  
scanf("%s", cad);
```


Entrada y Salida

- ❖ Además del formato %s existe el formato %[^\n] que permiten leer respectivamente una cadena de caracteres hasta encontrar una nueva línea.
 - Ejemplo: `scanf("%[^\n]", cadena);`
- ❖ La librería estándar de entrada y salida (stdio.h) proporciona además las funciones gets y puts, que permiten leer de teclado y mostrar por pantalla una cadena de caracteres completa, respectivamente.

Ejercicio

Realizar un programa que lea dos cadenas de caracteres, las concatena y finalmente escribe la cadena resultante.

Solución

```
#include <stdio.h>
#include <string.h>
main()
{
    char cad1[80], cad2[80], cad3[160];
    printf( "Introduzca la primera cadena:\n");
    gets(cad1);
    printf( "Introduzca la segunda cadena:\n");
    gets( cad2 );
    /* cad3 = cad1 + cad2 */
    strcpy( cad3, cad1 );
    strcat( cad3, cad2 );
    printf( "La cadena resultante es: %s \n", cad3 );
    getch();
}
```

Ejercicio

Realizar programa que cuente el numero de veces que se repite una palabra en una frase.

El programa emplea la función de comparación de cadenas strcmp.

Solución

```
#include <stdio.h>
#include <string.h>
#define MAXLIN 100
main()
{
    char pal[MAXLIN]; /* La que buscamos. */
    char palfrase[MAXLIN]; /* Una palabra de la frase. */
    char c;
    int total = 0;
    printf( "\nPALABRA:" );
    scanf( "%s", pal );
    printf( "\n\nFRASE:" );
    c = ' ';
    while (c != '\n')
    {
        scanf( "%s%c", palfrase, &c );
        if (strcmp(pal, palfrase) == 0)
            total++;
    }
    printf( "\nLa palabra %s aparece %d veces.", pal, total );
    getch();
}
```

Funciones de caracteres

- ❖ El archivo de cabecera **<ctype.h>** define un **grupo de funciones de manipulación de caracteres**. Todas las funciones devuelven un resultado de valor verdadero (distinto de cero) o falso (cero).
- ❖ Para utilizar cualquiera de las funciones no se puede olvidar incluir el archivo de cabecera **<ctype.h>** en la parte superior de cualquier programa que haga uso de esas funciones.

Funciones de caracteres

❖ Comprobación alfabética y de dígitos

- ❖ Existen varias funciones que sirven para comprobar condiciones alfabéticas:
 - **isalpha(c)** Devuelve verdadero (distinto de cero) si c es una letra mayúscula o minúscula. Se devuelve un valor falso si se pasa un carácter distinto de letra a esta función.
 - **islower(c)** Devuelve verdadero (distinto de cero) si c es una letra minúscula. Se devuelve un valor falso (0), si se pasa un carácter distinto de una minúscula.
 - **isupper (c)** Devuelve verdadero (distinto de cero) si c es una letra mayúscula, falso con cualquier otro carácter.

Funciones de caracteres

❖ Comprobación alfabética y de dígitos

- **isdigit(c)** Comprueba si *c* es un dígito de 0 a 9, devolviendo verdadero (distinto de cero) en ese caso, y also en caso contrario.
- **isalnum(c)** Devuelve un valor verdadero, si *c* es un dígito de 0 a 9 o un carácter alfabético (bien mayúscula o minúscula) y falso en cualquier otro caso.

Ejemplo

**Leer un carácter del teclado y
comprobar si es una letra.**

Solución

```
/* Solicita iniciales y comprueba que es alfabética
*/
#include <stdio.h>
#include <ctype.h>
main()
{
char inicial;
printf("¿Cual es su primer caracter inicial?: " );
scanf ( "%c" , &inicial) ;
fflush(stdin);
while (isalpha(inicial)==0)
{
    puts ("Caracter no alfabetico ") ;
    printf ("¿Cual es su siguiente inicial?: " ) ;
    scanf ("%c",&inicial) ;
    fflush(stdin);
}
puts ("Terminado!");
getch();
}
```


Funciones de conversión de caracteres

- ❖ Existen funciones que sirven para cambiar caracteres mayúsculas a minúsculas o viceversa.
 - **tolower(c)** Convierte el carácter c a minúscula
 - **toupper(c)** Convierte el carácter c a mayúscula

Ejemplo

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

int main()
{
    char resp;          /* respuesta del usuario */
    char c;

    printf("¿Es un varón o una hembra (V/H)? : ");
    scanf("%c",&resp);
    resp=toupper(resp);
    switch (resp)
    {
        case 'V':
            puts ("Es un enfermero");
            break;
        case 'H':
            puts ("Es una maestra");
            break;
        default:
            puts ("No es ni enfermero ni maestra");
            break;
    }
    return 0;
}
```