Tema 7: Cadenas y arreglos

1. Estructuras de datos

Tal como se mencionó en el tema 4, los datos se clasifican en **simples** y **compuestos**.

Los tipos de datos **simples o primitivos** son aquellos que no están definidos en términos de otros tipos de datos. Se denominan "primitivos" también porque son los tipos de datos originales que proporcionan la mayoría de los lenguajes de programación. Tienen como característica común que cada variable (o identificador) representa un valor de dato único. Los tipos de datos simples estándar son: *entero*, *real*, *carácter*, *lógico*.

Los tipos de datos **compuestos o estructurados** están construidos en base a los tipos de datos primitivos. Tienen como característica común que un identificador (nombre) puede representar múltiples datos individuales, pudiendo cada uno de éstos ser referenciado independientemente. Dentro de estos se encuentran los tipos de datos *cadenas*, *registros* y *arreglos*, que son tratados en esta asignatura.

Las estructuras de datos compuestas pueden clasificarse teniendo en cuenta diferentes características.

Según el tipo de los datos que las forman, se puede distinguir entre:

- **Estructuras homogéneas**: Si los datos que las componen son todos del mismo tipo.
- **Estructuras heterogéneas**: Si los datos que las componen son de distinto tipo.

Si se tiene en cuenta la cantidad de espacio de memoria utilizado por la estructura durante la ejecución del programa, se pueden clasificar en:

- **Estructuras estáticas**: cuando la cantidad de elementos que contienen es fija, es decir, el tamaño ocupado en memoria se define antes de que el programa se ejecute y no puede modificarse dicho tamaño durante la ejecución del programa.
- **Estructuras dinámicas**: cuando el número de componentes y, por lo tanto, la cantidad de memoria, pueden variar durante la ejecución de un programa (este tipo de estructura no se contempla en esta asignatura).

La siguiente tabla resume los tipos de datos simples y estructurados más frecuentes utilizados en los diferentes lenguajes de programación.

<table>
<tr><td rowspan="6">Datos Simples o Primitivos</td><td rowspan="4">estándar</td><td>entero(integer)</td></tr>
<tr><td>real (real)</td></tr>
<tr><td>carácter (char)</td></tr>
<tr><td>lógico (boolean)</td></tr>
<tr><td rowspan="2">definido por el programador (no estándar)</td><td>subrango (subrange)</td></tr>
<tr><td>Enumerativo (enumerated)</td></tr>
<tr><td rowspan="6">Datos Estructurados o Datos compuestos</td><td rowspan="3">estáticos</td><td>Arreglos (vectores/matrices)</td></tr>
<tr><td>registro (record)</td></tr>
<tr><td>cadenas (string)</td></tr>
<tr><td rowspan="3">dinámicos</td><td>listas (pilas/colas)</td></tr>
<tr><td>listas enlazadas</td></tr>
<tr><td>árboles y grafos</td></tr>
<tr><td></td><td>ficheros (archivos)</td></tr>
</table>

En este capítulo, se describen los datos compuestos, **Cadenas y Arreglos.**

1. Cadena de caracteres

1.1. Características

La información de texto se almacena en constantes y en variables de tipo “cadena”. Las cadenas de caracteres son elementos fundamentales para el manejo de texto en los lenguajes de programación.

Una cadena de caracteres se define como un conjunto ordenado de caracteres (letras, números, signos de puntuación, espacios en blanco, etc.). Por ejemplo: “9 de julio 1449” es una cadena de caracteres válida.

Internamente una cadena de caracteres se almacena en posiciones consecutivas de memoria, asignando un byte para almacenar el código ASCII de cada uno de los caracteres que la componen. El problema que tiene esta representación es que requiere que se indique a la computadora donde termina la cadena. El lenguaje C utiliza una “marca de fin de cadena“, el carácter NUL **o** ASCII 0, que se representa con **\0.**.

1.2. Declaración de una variable tipo cadena en C

El lenguaje C no tiene datos predefinidos tipo cadena (string) como otros lenguajes de programación. Una variable de cadena se declara con el tipo de dato **char** seguida de la longitud máxima de la cadena de caracteres entre corchetes.

```
char <identificador> [<longitud máxima>];
```

Ejemplos:

```
char cadena_hola[10];
```

Una cadena se caracteriza porque tiene una marca de fin de cadena, el carácter '\0'.

Una cadena se puede inicializar con una constante de caracteres entre comillas. El compilador automáticamente le agrega el carácter \0 de fin.

```
char cadena_hola[]= "Hola";

char otro_hola[]={'H','o','l','a','\0'};

char cadena_vacia[]="";
```

Una cadena se diferencia de un arreglo de caracteres por la marca de fin de cadena **/0.**

Arreglo de caracteres:

V	i	r	g	i	n	i	a	
0	1	2	3	4	5	6	7	8

Cadena de caracteres:

V	i	r	g	i	n	i	a	\0
0	1	2	3	4	5	6	7	8

Un arreglo de caracteres, como veremos más adelante, también es un tipo de dato compuesto. En el ejemplo anterior se muestra un arreglo de caracteres de 9 elementos, que contiene la palabra Virginia, y una cadena con los mismos datos. Se puede observar que la cadena ocupa un byte más para contener el carácter de fin de datos.

Ejemplo de uso de cadenas en C

Algoritmo que ingresa un nombre por teclado y lo muestra en la pantalla

```
#include<stdio.h>
/* Ingresa un nombre y lo muestra en pantalla */
main() {
      char nombre [20];
      printf ("Introduzca un nombre (max 20 letras): ");
      scanf ("%s", nombre);
      printf ("\nEl nombre ingresado es: %s\n", nombre);
      return 0;
}
```

1.3. Funciones para manejo de cadenas

La biblioteca estándar de ANSI C contiene el archivo de cabecera **string.h,** que permite utilizar las funciones de manipulación de cadenas de caracteres más usuales, como calcular la longitud de una cadena, comparar alfabéticamente dos cadenas, concatenar cadenas, etcétera.

1.3.1. Función strcpy

Copia la cadena apuntada por *cadena2* a *cadena1*, incluyendo el carácter nulo. Su sintaxis es:

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

La función toma el contenido de *cadena2* y lo almacena en *cadena1*. *Cadena1* debe ser de tamaño igual o mayor a *cadena2,* ya que almacena completamente hasta el carácter de terminación de *cadena2.*

Ejemplo:

```
      char cadena1 [15]= "Hola";
      char cadena2 [15] = "Hola de nuevo";
      printf ("Valor de la cadena 1: %s", cadena1);
      printf ("\nValor de la cadena 2: %s", cadena2);
      /* copia cadena2 en cadena1 */
      strcpy (cadena1, cadena2);
      printf ("\nValor de la cadena 1: %s", cadena1);
```

1.3.2. Función strcat

Concatena dos cadenas, copiando una a continuación de la otra, incluyendo su carácter nulo de terminación. Su sintaxis es:

```
char strcat(char cadena1,char cadena2)
```

Ejemplo:

```
      char cadena1 [15]= "Alan ";
      char cadena2 [15] = "Turing";
      printf ("Valor de la cadena 1: %s", cadena1);
      printf ("\nValor de la cadena 2: %s", cadena2);
      /* agrega cadena2 a cadena1 */
      strcat (cadena1, cadena2);
```

```
printf ("\nValor de la cadena 1: %s", cadena1);
```

1.3.3. Función strcmp

Compara elementos sucesivos de dos cadenas, hasta que encuentre elementos distintos o hasta que se acaben las cadenas. Devuelve 0 si las cadenas son iguales, un número negativo si cadena1 es menor que cadena2 (en términos lexicográficos), y un número positivo si es mayor.
Su sintaxis es:

```
int resultado = strcmp(char cadena1, char cadena2)
```

Ejemplos:

`resultado = strcmp("Waterloo","Windows")` Devuelve un valor negativo

`resultado = strcmp("Facebook","Face")` Devuelve un valor positivo

`resultado = strcmp("Pedro","Pedro")` Devuelve cero

1.3.4. Longitud de una cadena

El número total de bytes de una cadena en C, en la memoria RAM, es igual a la longitud de la cadena (cantidad de elementos) más 1 (marca de fin de cadena \0).

Ejemplo: char vocales [5] ="AEIOU";

A	E	I	O	U	\0

En tanto que la longitud de una cadena, es la cantidad de caracteres válidos, es decir, sin contar la marca de fin de cadena. En el ejemplo la longitud de la cadena vocales es 5.

a) Calcular la longitud recorriendo una cadena

Dado que una cadena es un dato de tipo compuesto, podemos recorrer y contar cada uno de sus elementos hasta identificar la marca de fin de cadena.

El código C que se muestra a continuación obtiene la longitud de una cadena:

```
/* devuelve la cantidad de caracteres en la cadena sin contar el '\0' */
#include <stdio.h>
#include <string.h>
int calculaLongitudCadena();
char cadena[] = "Algoritmos y Estructuras de Datos I";
int main() {
      printf("Valor de la cadena: %s", cadena);
      printf("\nLongitud de la cadena: %i", calculaLongitudCadena());
      return 0;
}
int calculaLongitudCadena() {
      int largo = 0;
      while (cadena[largo]!='\0') {
            largo++;
      }
      return largo;
}
```

b) Calcular la longitud utilizando la función strlen

Esta función calcula el número de caracteres del parámetro cadena, excluyendo el carácter nulo de terminación de la cadena. Su sintaxis es:

```
size_t strlen(const char *s);
```

donde size_t corresponde a un tipo de entero sin signo que se utiliza generalmente para representar el tamaño de un objeto.

Ejemplo para calcular la longitud de una cadena, utilizando la función **strlen en C.**

```
/* devuelve la cantidad de caracteres en cadena sin contar el '\0' */
#include <stdio.h>
#include <string.h>
char cadena[] = "Algoritmos y Estructuras de Datos I";
int main() {
      printf("Valor de la cadena: %s", cadena);
      printf("\nLongitud de la cadena: %i", strlen(cadena));
      return 0;
}
```

1.3.5. Funciones de conversión entre mayúsculas y minúsculas

El archivo de cabecera **<ctype.h>** provee funciones de conversión entre mayúsculas y minúsculas.

tolower	Convierte un dato carácter a minúscula.
toupper	Convierte un dato carácter a mayúsculas.

Ejemplo:

Convierte los caracteres de una cadena a mayúscula

```
int main(){
      ingresaFrase();
      convierteAminusculas();
      printf("%s\n", frase);
      return 0;
}
void ingresaFrase() {
      printf("Introduce una frase en minusculas: ");
      scanf("%[^\n]s",frase);
}
void convierteAminusculas() {
      int i;
      for(i = 0; frase[i]; i++) {
      frase[i] = toupper(frase[i]);
    }
}
```

En este ejemplo, prestar atención a la instrucción **scanf** que ingresa el dato de tipo cadena frase:

- Se agrega al especificador **%s** el modificador **[^\n].** El acento circunflejo indica hasta qué carácter leer, en este caso al tener **\n** lee hasta el salto de línea. Esto se hace para que lea la frase completa, de lo contrario, leerá solo la primera palabra.
- La variable, en este caso, *frase*, no necesita llevar el **&** previamente, dado que, el nombre de un arreglo (cadena en este caso), apunta a la dirección de comienzo del propio arreglo.

2. Arreglos (vectores y matrices)

Un arreglo es un conjunto finito y ordenado de elementos homogéneos.

- Es finito porque se debe conocer el número máximo de elementos para reservar la memoria para cada uno de ellos.
- La propiedad "**ordenado",** significa que el elemento primero, segundo, enésimo de un arreglo puede ser identificado.
- La propiedad "**homogéneo"**, quiere decir que los elementos son del mismo tipo de datos. Por ejemplo, un arreglo puede tener todos sus elementos de tipo entero, o todos sus elementos de tipo char.

Un arreglo agrupa elementos similares identificando al conjunto con un solo nombre o identificador.

Para acceder al valor de una determinada posición dentro del arreglo se utilizan **índices**.

La forma en la que se puede acceder a los elementos de un arreglo es **aleatoria** y no **secuencial**, dado que éstos son almacenados en la memoria RAM que posee la misma característica.

2.1. Arreglos unidimensionales: vectores

Los vectores son arreglos unidimensionales que requieren de un solo índice para acceder a cada elemento de los mismos. El **índice** (1, 2, …. ,i, n) indica su posición relativa en el vector.

Ejemplo 1: Se tiene un vector V de ocho elementos:

V[1]	V[2]	V[3]	V[4]	V[5]	V[6]	V[7]	V[8]
12	5	-7	14.5	20	1.5	2.5	-10

El nombre o identificador V representa el vector, es decir, el dato compuesto. Los elementos del vector se referencian por su ***índice***, es decir, por su posición relativa en el vector. Los índices de un vector pueden ser números enteros, variables o expresiones enteras.

Para i = 4

V [i+1] representa el elemento V [5] cuyo valor es 20

V [i+2] representa el elemento V [6] cuyo valor es 1.5

V [i-2] representa el elemento V [2] cuyo valor es 5

El número de elementos de un vector se denomina **rango del vector**. En este ejemplo, el rango del vector V es 8.

Los elementos de un vector se almacenan en la memoria RAM en un orden adyacente.

Ejemplo 2: Un vector de treinta números denominado NUMEROS se representa físicamente por treinta posiciones de memoria sucesivas.

	Memoria	
1		Dirección x
2		Dirección x + 1
3		Dirección x + 2
…		…
30		Dirección x + 29

Cada elemento de un vector se puede procesar como si fuese una variable simple al ocupar una posición de memoria. Así:

NUMEROS [25] = 72 almacena el valor entero o real 72 en la posición 25ª del vector NUMEROS.

La instrucción de salida **ESCRIBIR** NUMEROS [25] visualiza el valor almacenado en la posición 25ª del vector NUMEROS, en este caso 72.

Esta propiedad significa que cada elemento de un vector es accesible directamente, lo que representa una de las ventajas más importantes de usar un vector.

Por otra parte, los arreglos (tanto unidimensionales como multidimensionales) necesitan ser dimensionados previamente a su uso dentro del programa. *Dimensionar* significa reservar el espacio necesario en la memoria principal.

2.1.1. Declaración de vectores en el lenguaje C

```
TipoDeElementos nombreDelArray [numeroElementos];
```

Ejemplo: **int vectorEnteros [4];**

Declara un vector de enteros que contendrá 4 valores de tipo int.

Importante: **En el lenguaje C la numeración de elementos se inicia en 0 y no en 1.**
En el ejemplo dado, los índices de elementos en el arreglo serán 0, 1, 2, 3. Por tanto, el elemento vectorEnteros [4] no existirá.

Ejemplos de declaración de vectores:

char alfabeto [27], double decimalNum [24], int DiasSemana [7]

2.1.2. Operaciones con vectores:

a) Asignación directa

La asignación de valores a un elemento del vector se realizará con la instrucción de asignación:

A [3] = 5 asigna el valor 5 al elemento 3 del vector A

b) Asignación desde dispositivos de E/S

La asignación de valores a los elementos de un arreglo puede hacerse también mediante dispositivos de E/S como el teclado. En este caso, la instrucción de lectura se representará como:

```
int A[5];
      scanf("%d", &A[2]);
      printf("Elemento de A[2]: %d \n", A[2]);
```

En este ejemplo, se muestra el ingreso de un valor en la posición 3 del vector A.

c) Acceso secuencial (recorrido)

A la operación de efectuar una acción general sobre todos los elementos de un vector se denomina **recorrido** del vector.

Estas operaciones se realizan utilizando estructuras repetitivas, cuyas variables de control (por ejemplo "i") se utilizan como índice del vector, por ejemplo, S [i]. El incremento del contador del bucle producirá el tratamiento sucesivo de los elementos del vector.

Ejemplo 1: Asignación de valores a un vector denominado temperaturaMaximaMensual.

Se declara un vector de 12 elementos, uno para cada mes del año, y se ingresan los valores desde el teclado. Los elementos se cargan en el vector comenzando con el elemento 0 y terminando con el 11.

Ejemplo en C

```
#include <stdio.h>
/* crea un vector entero de 12 elementos e ingresa los datos por teclado */
int main(){
  float tempMaxMensual[12];
  int i;
```

```
    for (i=0;i<12;i++){
        printf("Ingrese el valor de la temperatura máxima del mes: ");
        scanf("%f", &tempMaxMensual[i]);
    }
    for (i = 0; i < 12; i++) {
        printf("\nTemperatura Maxima mes %d: %.2f", i+1, tempMaxMensual[i]);
    }
    getc(stdin); /* lee caracter desde el teclado */
    return 0;
}
```

La función getc(stdin) se utiliza para solicitar el ingreso de cualquier carácter, a fin de mostrar la salida del printf y luego continuar.

La salida o escritura de vectores se representa de un modo similar al ingreso de datos. Las estructuras iterativas permiten visualizar el vector completo (un elemento en cada línea independiente), tal como se muestra en el ejemplo anterior que utiliza un for para iterar el printf para cada elemento.

2.2. Arreglos bidimensionales: matrices

Los arreglos de tipo bidimensional se denominan *matrices*.

Una matriz es un conjunto de elementos, todos del mismo tipo, en el cual el orden de los componentes es significativo y se necesitan dos índices para identificar cada elemento del arreglo.

Ejemplo: Matriz Calificaciones de 20 filas y 4 columnas y Vector Nombre de 20 elementos.

	0	1	2	3
0	9	8	9	8
1	6	5	7	6
2	8	10	10	9
3	8	8	8	9
4	7	9	9	7
5	6	7	8	6
6	10	9	8	8
..				
20	6	8	6	8

Calificaciones [ ] [ ]

Juan
María
Patricia
Emanuel
José
Ana
Cecilia
…
Antonio

Alumnos []

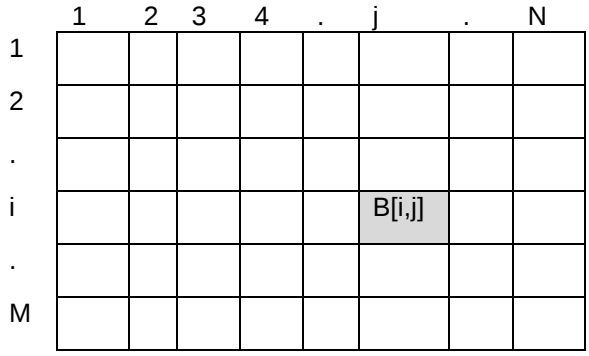
La matriz representa las notas obtenidas por 20 alumnos en 4 actividades de evaluación. La matriz podría asociarse a un vector de alumnos de la misma cantidad de filas de la matriz.

2.2.1. Ventajas de los arreglos

Los arreglos posibilitan una organización de los datos que permite realizar muchas operaciones a la vez sobre grandes conjuntos de datos de manera eficiente. Algunos ejemplos de problemas que requieren manipular grandes secuencias de números son: la predicción del clima, la construcción de edificios, y el análisis de indicadores financieros, entre muchos otros.

La característica de esta estructura es que los datos del arreglo están disponibles en la RAM en el mismo momento, con lo cual se pueden realizar varias operaciones operando sobre el conjunto de datos. Por ejemplo, con los datos de la matriz Calificaciones podríamos obtener el promedio de calificaciones de cada alumno, o el desempeño promedio de los estudiantes en cada examen, o qué evaluación tuvo la mayor cantidad de desaprobados (notas menores a 6).

En notación estándar, B [i, j] es el elemento de B que ocupa la iª (*i-ésima*) fila y la jª (*j-ésima*) columna.



El elemento B [i , j] se puede representar en notación algorítmica. El arreglo B con elementos de tipo T (numérico o alfanumérico), con índice fila que varía en el rango de 1 a M e índice columna en el rango de 1 a N es:

B (1: M, 1: N) = {B [i, j]}

Donde i = 1 .. M o 1 <= i <= M
Donde j = 1 .. N o 1 <= j <= N

2.2.2. Declaración de matrices en C

De la misma manera que los arreglos unidimensionales (vectores), los arreglos bidimensionales (matrices) se crean con declaraciones TIPO y VAR. Se deben indicar:

1) El rango permitido por cada índice.
2) Tipo del arreglo (todos los elementos del arreglo deben ser del mismo tipo).
3) Nombre del arreglo.

```
TipoDato nombreDelArray [elementosFila, elementosColumna];
```

Ejemplo: **int matrizEnteros [4,5];**

Declara un matriz de enteros de 20 elementos, dispuestos en 4 filas y 5 columnas.

2.2.3. Recorrido y manipulación de matrices

En general, el orden natural para procesar los vectores es el orden secuencial: del primero al último elemento. En el caso de los arreglos bidimensionales, existen diferentes ordenamientos para sus recorridos. Los más usuales son: *recorrido por filas* y *recorrido por columnas.*

Recorrido por filas
Los elementos de la primera fila se procesan primero, luego los de la segunda y así sucesivamente. Eso implica que primero se hace variar el índice de fila y dentro de éste al índice de columna. Al iniciar cada nueva fila se recorre desde la primera columna nuevamente.

Recorrido por columnas
Los elementos de la primera columna se procesan primero, luego los de la segunda y así sucesivamente. Eso implica que primero se hace variar el índice de columna y dentro de éste al índice de fila. Al iniciar cada nueva columna se recorre desde la primera fila nuevamente.

Ejemplo 1: Leer desde el teclado y almacenar los siguientes datos en una matriz **A** de 3 filas por 4 columnas, con elementos de tipo entero:

A

25	35	12	20
11	1	18	34
6	12	18	24

Recorrido por filas	Recorrido por columnas

for (fila=0; fila <3; fila++) {
for (columna=0; columna<4; columna++) {
scanf("%d", &A[fila][columna]);
}
}
	for (columna=0; columna<4; columna++) {
	for (fila=0; fila <3; fila++) {
	scanf("%d" ,&A[fila][columna]);
	}
	}

En el recorrido por filas, el primer **for** se ubica en la **fila 0** y el segundo **for** se ubica en la **columna 0**, luego incrementa el índice de columna para ubicarse en la siguiente columna de la misma fila y continua hasta completar el recorrido de toda la fila, en este caso, mientras que columna sea menor que 4. Finalizado el for más interno, continua con el for de la fila, incrementando el índice para pasar a la siguiente fila. En este punto, inicia nuevamente el for mas interno que recorre todas las columnas hasta completar la fila. El proceso se repite mientras el valor de la fila sea menor que 3.