estudios abiertos
SEAS
GRUPO SANVALERO

1 UNIDAD DIDACTICA

Estructura de Datos

1. Arrays

ÍNDICE

OBJETIVOS

- Comprender el concepto de los arrays.
- Dominar el tratamiento de los mismos.
- Trabajar con arrays unidimensionales.
- Trabajar con arrays multidimensionales.
- Manejar cadenas en un lenguaje de programación estructurado.
- Familiarizarse con las funciones de la librería < string.h >

INTRODUCCIÓN

Un array es una estructura de datos fundamental, ya que permite la manipulación de conjuntos de elementos del mismo tipo, accediendo a ellos a través de un índice.

Los arrays son la base para confeccionar otras estructuras de datos más complejas.

En un lenguaje como C son la esencia de las cadenas.

En esta unidad el alumno aprenderá la forma de trabajo con arrays tanto de una única dimensión como con arrays multidimensionales.

Se ahondará también en los aspectos más técnicos del manejo de cadenas, así como de los estándares más utilizados por los lenguajes para el manejo de las mismas.

Finalmente el alumno implementará las operaciones básicas de cadenas como ejemplo más claro del uso de arrays y se familiarizará con el uso de las funciones que facilita la biblioteca < string.h >.

1.1. Qué son los array

El array en un concepto basado en la concatenación de elementos simples, en un lenguaje de programación cuando hablamos de arrays, nos estamos refiriendo a un conjunto enumerado de datos simples, los cuales están ordenados de forma lineal, y su selección responde al valor de un índice que permite la manipulación de un elemento específico que forma dicho array.

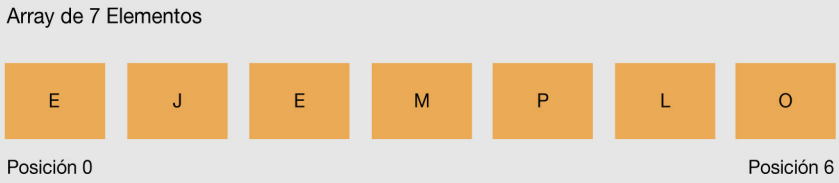
DEFINICIÓN

Un array es una estructura de datos compuesta por un conjunto de elementos del mismo tipo.

Así pues, los elementos de un array están enumerados mediante un índice, cuyo límite inferior se sitúa siempre en la posición 0 del array y su límite superior (último elemento) se ubicará en la posición longitud del array n – 1.

De tal forma que cuando hablamos de una array de *n* elementos, por tanto dicho array se podrá recorrer desde la posición *0* hasta la posición *n -1.*

RECUERDA



De tal forma que, la segunda letra de la palabra EJEMPLO, (la J) estará situada en la segunda posición del array pero su índice de acceso es el 3.

TRUCO

Regla de Acceso:
Elemento x = array [índice – 1]

1.2. Trabajo con arrays

1.2.1. Declaración de arrays

Al igual que sucede con los datos simples, el lenguaje C exige que los arrays antes de ser utilizados sean declarados.

Una declaración de un array implica definir en tiempo de compilación el número máximo de elementos que podrá contener así como el tipo de dato que albergará en su interior.

EJEMPLO

Veamos la declaración en C de un array que podrá contener hasta 5 números enteros:

```
int array_enteros [5];
```

La declaración de un array implica la reserva explícita de la memoria necesaria para poder albergar el número máximo de elementos que contendrá el array.

Es decir, en el ejemplo anterior estamos reservando en nuestra máquina memoria para almacenar 5 valores enteros.

1.2.2. Inicialización de arrays

Al igual que los tipos simples de datos, es posible asignar valores al array antes de ser utilizados existen 3 formas se inicialización de array, en función de cómo demos valor a los elementos del array.

NOTA

Inicializar un array consiste en dar un valor válido a sus elementos.

Inicialización individual de elementos

Los elementos del array se inicializan de forma individual, de tal forma que para cada elemento del array al que deseemos asignar un valor inicial deberemos realizar una operación de inicialización:

EJEMPLO

```
array_enteros[0] = 105;
array_enteros[1] = 310;
array_enteros[4] = 640;
```

Inicialización global indicando número de elementos

La totalidad de elementos son inicializados, se debe indicar el número de elemento que contiene el array:

EJEMPLO

```
float pagos_base[3] = {8.5,12.5,17.99};
```

Inicialización global sin indicar número de elementos

Al igual que en la forma anterior se inicializan la totalidad de elementos pero no hace falta indicar el número de elementos que contendrá el array:

EJEMPLO

```
int dado [ ] = {1,2,3,4,5,6};
char nombre [ ] = {'J','U','A','N'};
```

ATENCIÓN

Las formas globales de inicialización implican la declaración misma del array, esto es, no es posible definir un array previamente y después utilizar una inicialización global para dicho array.

1.2.3. Uso de un array

La utilización de un array comprende 2 operaciones básicas, éstas no son otras que la asignación y la obtención de un elemento del array:

Asignación de valores a elementos de un array

La operación de asignación es una operación homóloga a la inicialización de un array.

Una inicialización es básicamente una asignación de valores antes del uso del array.

Para asignar un valor a un elemento de un array lo haremos apoyándonos en el índice, la fórmula es la siguiente:


```
array_enteros [3] = 99;
```

Obtención de un elemento de un array

La recuperación de elementos de un array también se lleva a cabo mediante el índice.

Si queremos almacenar el valor que recuperamos del array, dicho almacenamiento de debe llevar a cabo en una variable del mismo tipo de dato que estamos recuperando.


```
int entero;
entero = array_enteros[0];
```

Veamos ahora un ejemplo de código en el que se inicializa un array con números consecutivos, posteriormente se carga con números aleatorios y luego se muestran dichos números.

EJEMPLO

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

#define N_ELEMENTOS 6
#define VALOR_MAXIMO 10

int obtenerNumeroAleatorio(const int numeroMaximo);
void visualizaArray();

void main()
{

    int i;
    //Declaración e inicialización del array original
    int array[N_ELEMENTOS]={1,2,3,4,5,6};

    system("cls");
    srand(time(NULL));

    printf("Datos del array original\n");
    visualizaArray(array);

    //Bucle para cargar los valores aleatorios a cada posición del array
    for(i=0;i<N_ELEMENTOS;i++)
    {
         array[i]=obtenerNumeroAleatorio(VALOR_MAXIMO);
    }

    printf("Datos del array cargado con números aleatorios\n");
    visualizaArray(array);

    getch();
}

int obtenerNumeroAleatorio(const int numeroMaximo)
{

  return(rand()%numeroMaximo);
}

void visualizaArray(int array[])
{
    int i;

    for(i=0;i<N_ELEMENTOS;i++)
    {
         printf("array [%d] = ",i);
         printf("%d\n",array[i]);
    }
}
```

1.3. Cadenas

Veremos a continuación conceptos teóricos de cadenas y la implementación de las mismas en programación C.

1.3.1. Concepto de cadena

DEFINICIÓN

Una cadena es un conjunto de 0 o n caracteres.

Por carácter se entiende un valor válido de la tabla ASCII de nuestra máquina.

Normalmente se trabaja con el ASCII Ext. Compuesto por elementos de 8 bits de tamaño.

ASCII Ext. (ASCII Extendido) presenta 256 valores distintos.

1.3.2. Tipos de cadenas

Las estructuras de tipo cadena se pueden presentar en varios tipos o formatos si las clasificamos según su longitud podemos encontrar:

Cadenas de longitud fija

La longitud de la cadena debe ser especificada en tiempo de compilación, posteriormente no se puede varia su longitud, de ahí su nombre.

Por tanto el número máximo de elementos que son capaces de contener es siempre conocido.

Cuando trabajamos en C con cadenas de tipo char [ ] (arrays de char), estamos trabajando con este tipo de cadenas.

Cadenas de longitud variable

Son una variante de las anteriores, ya que en tiempo de compilación se reserva memoria para la longitud máxima de la cadena y en tiempo de ejecución se especifican longitudes inferiores.

Cadenas de longitud indefinida

La definición real de la longitud de la cadena no es en tiempo de compilación sino en tiempo de ejecución.

Se irá solicitando memoria al sistema conforme se vaya necesitando.

Cuando esa memoria deje de ser útil debemos encargarnos de liberarla.

Este tipo de cadenas se realiza mediante una estructura que veremos en temas posteriores, denominada lista enlazada.

1.3.3. Manejo de cadenas

El estándar para el manejo de cadenas definido para el lenguaje C se basa en un array de caracteres donde cada carácter se almacena en una posición del array comenzando desde la posición 0 del mismo. La delimitación del final de la cadena se basa len a marca'\0', conocida como *marca de fin de cadena.* Cuando se detecta esa marca los restantes caracteres de la cadena no son tenidos en cuenta.

Este estándar es utilizado en la mayoría de lenguajes modernos de programación.

1.3.4. Operaciones con cadenas

Cuando se trabaja con cadenas se presentan un conjunto de funciones que hacen referencia a la operatoria propia del manejo de cadenas, veremos las principales que pueden presentarse.

Se indicará también si es pertinente alguna función de la librería < string.h > que realice dicha función.

Asignación

Consiste en darle valor a una cadena mediante un literal.

EJEMPLO

```
char cliente [ ] = "Germán Romeo"
```

Calcular longitud

Consiste en obtener el tamaño de un array devolviendo el número de posiciones del mismo.

En C disponemos de la función strlen para esta utilidad.

EJEMPLO

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

void main()
{
    //Declaración e inicialización del array
    char cadena1[]="abcdefg";
    int tam;

    //Guardamos en la variable tam la longitud de la cadena1
    tam=strlen(cadena1);

    printf("La longitud de la Cadena1 = '%s' es %d\n",cadena1,tam);

    getch();
}
```

Comparación

Se compara si una cadena es mayor, igual o menor que otra dada, de acuerdo con la secuencia de caracteres ASCII.

En C se pueden utilizar funciones como:

strcmp, _fstrcmp, _mbscmp, wcscmp

AVISO

No se debe confundir la comparación de cadenas con la comparación basada en longitud.

La comparación de cadenas que nos ocupa es una comparación carácter a carácter conforme a la secuencia ASCII, de tal forma que cuanto más cerca del inicio de la tabla ASCII se encuentre un carácter será menor.

Por ejemplo, el carácter '1' es menor que el carácter 'Z'.

EJEMPLO

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

void main()
{
    //Declaración e inicialización de los arrays
    char cadenaOrigen[256]="aaaaa";
    char cadenaDestino[256]="bbbbb";
    char *mensaje;
    int resultado;

    system("cls");

    //Guardamos en esta variable lo que devuelve la función de comparación
    resultado = strcmp(cadenaDestino,cadenaOrigen);

    switch(resultado)
    {
        case 0:
        {
            mensaje="Ambas son IGUALES\n";
            break;
        }

        default:
        {
            if(resultado > 0)
                mensaje="cadenaDestino es MAYOR\n";
            else
                mensaje="cadenaDestino es MENOR\n";
            break;
        }
    }

    printf ("%s",mensaje);
    getch();
}
```

Concatenación

Se agrega el contenido de una segunda cadena al contenido de la primera.

En C se pueden utilizar funciones como:

strcat, _fstrcat, _mbscat, wcscat

EJEMPLO

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

void main()
{
    //Declaración e inicialización de los arrays
    char cadenaOrigen[256]="Desarrollo";
    char cadenaDestino[256]="Investigacion";

    system("cls");

    //Concatenamos las dos cadenas en la cadena cadenaDestino
    strcat(cadenaDestino,cadenaOrigen);

    printf ("%s",cadenaDestino);
    getch();
}
```

AVISO

Si manipulamos manualmente las cadenas habrá que tener en cuanta que la capacidad de la cadena destino sea suficiente como para además de contenerse asimisma, poder contener a la otra cadena.

Copia

Obtiene los *n* caracteres de una cadena partiendo de una posición inicial.

AVISO

Hay que tener en cuenta que esta operación es una operación destructiva, lo que implica que todos los caracteres de la cadena origen se copian en la cadena destino, desapareciendo el valor de la cadena destino, aunque la cadena destino sea más larga que la cadena de origen.

C presenta las siguientes funciones para copiar cadenas:

strcpy, _mbscpy, wcscpy

EJEMPLO

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

void main()
{
    //Declaración e inicialización de los arrays
    char cadena1[]="aaaaa";
    char cadena2[]="bbbbb";

    //Copia en cadena2 el contenido de cadena1
    strcpy(cadena2,cadena1);

    printf("Cadena1 = %s\n",cadena1);
    printf("Cadena2 = %s\n",cadena2);

    getch();
}
```

Búsqueda de un carácter

El algoritmo de búsqueda se basa en verificar si cualquier carácter de una cadena dada se encuentra en otra cadena también dada. Esta función devuelve la posición donde se da la primera de las coincidencias.

AVISO

En caso de que ningún carácter de la cadena a buscar sea coincidente con la cadena original se devuelve la posición del carácter '\0' fin de cadena.

En C disponemos de la función strcspn para esta utilidad.

EJEMPLO

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

void main()
{
  //Declaración e inicialización de los arrays
  char cadena1[13] = "buenos días";
  char cadena2[5] = "aeiou";
  int posicion;

  //Guardamos en la variable posicion la primera posición
  //donde aparece un caracter de la cadena2 en la cadena1
  posicion=strcspn( cadena1, cadena2 );

  printf("Cadena1 = %s\n",cadena1);
  printf("Cadena2 = %s\n",cadena2);
  printf("Posición = %d\n", posicion);

  getch();
}
```

Cambiar

Sustituye n caracteres de una cadena por los n caracteres de otra cadena dada.

En C disponemos de la función strncpy para esta utilidad.

EJEMPLO

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

void main()
{
  //Declaración e inicialización del array
  char cadena1[15] = "buenos días";
  char cadena2[15] = "gggg";

  strncpy( cadena1, cadena2, 2 );

  printf("Cadena1 = %s\n",cadena1);
  printf("Cadena2 = %s\n",cadena2);

  getch();
}
```

1.4. Cadenas vs. arrays de char

En lenguaje C existe una pequeña aunque sustancial diferencia entre un array de char y una cadena.

Se puede afirmar que una cadena es una particularidad de un array de char, cuyo último carácter es siempre la marca de fin de cadena '\0'.

RECUERDA

NO ES LO MISMO
char animal [4] = { 'G' , 'A' , 'T' , 'O' };
char mascota [ ] = "GATO"

El array animal posee 4 caracteres, mientras que el array mascota posee 5 caracteres ya que el lenguaje añade por defecto como último carácter la marca '\0', cuando encuentra el token "", que es el indicativo de cadena.

Por tanto a las variables que responden a la primera firma se les denomina array de char y a las que se corresponden con la segundan de denominan cadenas, aunque en esencia, ambas son sucesiones de caracteres y su diferencia radica en la forma declarativa de las mismas.

1.5. Arrays y memoria

Cuando trabajamos con arrays debemos tener en cuenta una serie de puntos que afectan directamente a cómo se gestiona la memoria de nuestra máquina:

- Cuando se solicita memoria para un array ésta es siempre demandada al sistema en tiempo de compilación y es asignada de forma contigua. Será por tanto imposible aumentar el tamaño de dicha array en tiempo de compilación.

Algunos lenguajes de programación de alto nivel, si permiten variar la capacidad de los arrays en tiempo de ejecución. Tenga en cuenta que el lenguaje C NO permite una redimensión directa, si desea hacerlo será labor del programador.

- Los índices de un array no son controlados en absoluto por el sistema, es decir, el hecho de acceder a posiciones válidas de un array es labor exclusiva del programador. Un error de este tipo supondrá invadir una zona de memoria que corresponderá con otras secciones de código y según las características de estas secciones las consecuencias pueden ser dramáticas.

- Como los elementos de un array están ubicados de forma contigua en memoria, es posible desplazarnos por la memoria, para acceder a dichos elementos. Este tema será expuesto más adelante en la unidad que habla de punteros.

1.6. Array multidimensionales

En el transcurso de la unidad hemos trabajado con arrays de una única dimensión, este tipo de array se denominan unidimensionales, no obstante el lenguaje C no impone restricciones en cuanto al número de dimensiones que puede tener un array, el límite estará por tanto en la capacidad de abstracción del programador para poder manejar estructuras multidimensionales complejas.

En cuanto a arrays multidimensionales, en esta unidad trataremos únicamente a los arrays de 2 dimensiones, más conocidos como tablas o matrices.

1.6.1. Arrays bidimensionales : tablas

Cuando tratamos con arrays de 2 dimensiones ya no sólo debemos controlar un único índice, sino que ahora se duplica el trabajo con respecto a este punto, ya que habrá que tener en cuenta tanto el desplazamiento por filas como el desplazamiento por columnas.

1.6.1.1. Declaración de una tabla

La declaración de tablas es básicamente la misma que en los arrays unidimensionales, con la excepción de que la longitud de cada dimensión debe ser expresada en la definición.

EJEMPLO

Algunas declaraciones posibles son:

```
char tablero [28] [28];
double secuencias [3] [33];
```

1.6.1.2. Inicialización de tablas

Los arrays de más de una dimensión se inicializan de la misma forma que los arrays unidimensionales, esto es, dando valor a cada uno de los elementos que componen el array.

Los elementos irán encerrados entre llaves y separados por comas.

No obstante y sólo atendiendo a un formato de claridad podemos encontrar 2 tipos de inicialización:

Inicialización por secuencia

Se basa en una secuencia lineal de cada uno de lo elementos, tal y como si de un array de dimensión 1 se tratase:

EJEMPLO

```
int tabla [2] [2] = { 1,2,3, 4 };
```

Inicialización por bloques

Esta forma de inicialización es más clara o visual y se basa en separar por bloques las filas:

EJEMPLO

```
float matriz [3] [3] = {
                          {1.1, 2.2, 3.3} ,
                          {4.4, 5.5, 6.6} ,
                          {7.7, 8.8, 9.9}
                        };
char letras [2] [6] = { “Mario”,”María”};
```

1.6.1.3. Acceso y recuperación de los elementos de una tabla

El acceso y obtención de los elementos de una matriz bidimensional, es básicamente el mismo que para un matriz de dimensión única, con la excepción de que deberemos especificar 2 índices.

El primero de los índices hará referencia a la fila y el segundo a la columna.

Esta operatoria es extensible a *n* dimensiones, esto es, para obtener un elemento dado de un array de *n* dimensiones habrá que especificar la posición que ocupa dicho elemento en cada una de las *n* dimensiones que forman dicho array.

Veamos un ejemplo en código de C, que expresa como se introducen y se extraen elementos de una tabla:

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

#define DIMENSION 3
#define VALOR_MAXIMO 10

void visualizaMatriz(int[DIMENSION][DIMENSION]);
void cargarMatriz(int[DIMENSION][DIMENSION]);
int obtenerNumeroAleatorio(const int numeroMaximo);

void main()
{

    //Declaración de la matriz
    int matriz[DIMENSION][DIMENSION];

    //Limpiamos la pantalla
    system("cls");
    //Inicializamos la semilla de la obtención de números aleatorios
    srand(time(NULL));

    //Procedemos con la carga de datos
    cargarMatriz(matriz);

    //Procedemos con la visualización de datos de la matriz
    visualizaMatriz(matriz);

    getch();
}

int obtenerNumeroAleatorio(const int numeroMaximo)
{

  return(rand()%numeroMaximo);
}

void cargarMatriz(int matriz[DIMENSION][DIMENSION])
{
    int i,j;

    //Bucles para cargar los valores aleatorios a cada posición de la matriz
    for(i=0;i<DIMENSION;i++)
    {
         for(j=0;j<DIMENSION;j++)
         {
              matriz[i][j]=obtenerNumeroAleatorio(VALOR_MAXIMO);
         }
    }
}
```

```
void visualizaMatriz(int matriz[DIMENSION][DIMENSION])
{
    int i,j;

    //Bucles para visualizar los valores de cada posición de la matriz
    for(i=0;i<DIMENSION;i++)
    {
        printf("\n");

        for(j=0;j<DIMENSION;j++)
        {
            printf("[%d]    ",matriz[i][j]);
        }
    }
}
```

1.7. Arrays y funciones

Cuando pasamos un array como parámetro a una función se debe tener en cuenta que no se pasa el array por valor, como ocurre con los demás tipos básicos en C, sino que lo que realmente se pasa es la dirección donde se ubica el primer elemento del array.

Esto implica que siempre que cambiemos elementos de un array en una función los estamos cambiando también en el programa principal, ya que no se trata de una copia del array, (no es un paso por valor sino por referencia) sino estamos trabajando directamente con el array original.

Fíjate en el ejemplo anterior para ver cómo se pasan los parámetros a un función.

RESUMEN

- Los arrays son una secuencia de elementos de un mismo tipo accesibles mediante un índice, el cual siempre comienza en la posición 0 y termina en la longitud del array – 1.
- Los arrays de declaran conforme la siguiente regla:

```
tipo_dato nombre_array [ cantidad_elementos ]
```

- La inicialización de los elementos de un array es entre llaves y separados unos de otros por el carácter ‘,’ (coma).
- La asignación y obtención de valores de un array siempre se debe realizar indicando el nombre y el índice del elemento dentro del array al que se desea acceder.
- Las cadenas son un tipo específico de array de char que tienen como peculiaridad que terminan con el carácter ‘\0’.
- Existe un conjunto de operaciones que permiten la manipulación de las cadenas dentro de los sistemas informáticos. En C gran parte de estas funcionalidades las tenemos en la librería < string .h >.
- Los arrays se ubican en memoria siempre de forma contigua, es decir, la dirección del elemento n, será la siguiente de la del elemento n -1.
- En C se pueden crear arrays de n dimensiones, no existe una limitación específica para la dimensión del array. Cuando la dimensión es *n*, para acceder a los elementos del array necesitaremos también *n* elementos.
- Cuando pasamos un array como parámetro de una función, este paso de parámetro es siempre por referencia o dirección y nunca por valor.