Estructuras de datos estáticas

Una estructura de datos es una colección de datos organizados de un modo particular. Es decir este tipo de datos se pueden ver como una agrupación para una serie de tipos de datos primitivos.
Las estructuras de datos se crearon para solucionar muchos problemas que no se podían solucionar o que no se solucionan fácilmente con los tipos de datos primitivos.

Supongamos que queremos almacenar 25 notas de un curso determinado para luego procesarlas y obtener el promedio de dicho curso o el promedio de un alumno determinado. Con la información que contamos hasta el momento el fragmento de código podría ser como el siguiente:

```
#include <stdio.h>
void main()
{
    //Declaramos 25 variables correspondiente a las notas de los alumnos
    int n0, n1,  n2,  n3,  n4,  n5,  n6, n7, n8;
    int n9,  n10, n11, n12, n13,n14,n15,n16, n17;
    int n18, n19, n20, n21, n22, n23, n24, n25;
    float promedio;

    //Se introduce el valor de la nota de cada alumno
    printf( "Introduzca las notas de 25 alumnos");
    scanf( "%d",&n0);
    scanf( "%d",&n1);
    scanf( "%d",&n2);
    scanf( "%d",&n3);
    ...
    ...
    scanf( "%d",&n25);
    //Se calcula el promedio y se muestra por pantalla
    promedio = (n0+n1+n2+n3+n4+ ... +n25)/26;
    printf( "\nEl promedio de notas es %f\n",promedio);
    return 0;
}
```

Este ejemplo, no constituye código válido en C, los puntos suspensivos reemplazan las lecturas de todas las notas declaradas en la zona de declaración. El fragmento desarrollado aquí solo intenta mostrar que tan largo se hace un trabajo que en definitiva es repetir un determinado código muchas veces. Es aquí donde se hacen útiles las estructuras de datos estáticas.

Analicemos el mismo fragmento de código desde el punto de vista del uso de estructura de datos estáticas:

```c
#include <stdio.h>
#include<stdlib.h>
main()
{
    int notas[25]; //Con esto declaramos las 25 variables o notas
    float promedio;
    int i;//Variable entera que maneja el índice del vector notas
    system("cls");
    // Aquí ingresamos las 25 notas correspondientes
    for(i=0; i<25; i++ ) {
        printf( "Ingrese la nota nro %d ",i);
        scanf( "%d",&notas[i] );
        promedio += notas[i];//Acumulamos las notas ingresadas
    }
    promedio = promedio /i;//Obtenemos el promedio
    printf( "\nLa nota promedio es %f\n", promedio );
}
```

Arrays

Un array es una colección de variables del mismo tipo que se denominan por un nombre común (Identificador) y se caracterizan por: 1) almacenar los elementos en posiciones de memoria contiguas; 2) tener un único nombre de variable que representa a todos los elementos y estos se diferencian por un índice o subíndice; 3) acceso directo o aleatorio a los elementos individuales del array.

Arrays unidimensionales o vectores

Un array de una dimensión (vector) es un tipo de dato estructurado compuesto de un número de elementos finitos, homogéneos y de tamaño fijo. Finitos indica que hay un último elemento, tamaño fijo quiere decir que su tamaño se debe conocer en tiempo de compilación y homogéneo significa que todos los elementos son del mismo tipo.

Cada elemento se almacena en posiciones contiguas de memoria, pudiendo acceder a dichas posiciones directamente a través de un índice.

Declaración de un vector

La forma o sintaxis general de declarar un array unidimensional es:

```
<Especificador_de_tipo> <Identificador_de_variable>[tamaño];
```

Aquí, `especificador_de_tipo` hace referencia al tipo de dato o elementos del array, el `identificador_de_variable`, es el nombre que recibirá el vector, por último el tamaño indica la cantidad de elementos que contendrá dicho vector.

Ejemplo:

```
//Vector denominado "valores" de 5 elementos enteros
  int valores[5];
```

Todos los vectores en C tienen el cero como índice de su primer elemento. Por lo tanto en este ejemplo se está declarando un vector de enteros de 10 elementos, enumerados del 0 al 9, dicho de otro modo tenemos un vector que va desde valores [0] hasta valores [4].

Gráficamente el vector "valores" se puede ver de la siguiente manera:

```
valores[0];
valores[1];
valores[2];
valores[3];
valores[4];
```

Cuando establecimos la definición de array dijimos que cada uno de sus elementos se distinguen mediante un índice. Como muestra la gráfica anterior, para indicar a qué elemento nos referimos usamos un número entre corchetes, este número es lo que se llama índice del array que es un valor entero mayor o igual que cero y que debe estar declarado como tal en la zona de declaración correspondiente.

También como ya se mencionó en C, y como muestra la gráfica, el primer elemento de una arreglo tiene el índice cero, el segundo tiene el índice 1, y así sucesivamente. De manera que si queremos asignar por ejemplo un valor a la posición 5 del vector declarado más arriba debemos hacer referencia al índice 4 del siguiente modo:

```
valores[4] = 20;
```

Operaciones con vectores

Entrada y salida

En lenguaje C no permite realizar operaciones que involucran arrays completos, excepto si son vectores de caracteres que veremos más adelante.
Por lo tanto, si tenemos dos arrays, con el mismo tipo de datos, la misma dimensión y el mismo tamaño, las operaciones de comparación, de asignación etc, deben realizarse elemento por elemento. Esto se hace generalmente dentro de un bucle donde cada paso del mismo se utiliza para procesar un elemento del array.

Para realizar una operación de entrada se procede del mismo modo que con tipos primitivos, utilizando la función scanf con la diferencia que en un array para referenciar la posición en el que quiere introducirse el valor hay que emplear el índice correspondiente del elemento dentro del vector.

Del mismo modo para realizar una operación de salida utilizamos la función printf que mostrará el contenido del elemento del vector especificado en el índice. También para mostrar por pantalla todos los elementos de un vector es conveniente emplear un bucle para recorrer los elementos del mismo.

Hay que tener en cuenta que tanto la salida como la entrada usarán el especificador de formato del tipo utilizado para declarar el vector. El siguiente código, muestra cómo ingresar y mostrar por pantalla un vector de 10 elementos.

```c
#include<stdio.h>
#define cant 10//Se define la cantidad de elementos del vector
int main(){
    int vector[cant]; //Declaramos un vector de  las 10 elementos enteros
    int i;//Variable entera que maneja el índice del vector
    //Ciclo for para cargar los elementos en el vector
    for(i=0; i<cant; i++ ) {
    //i indica la posición que se desea leer
        printf( "Ingrese el elemento nro %d ",i);
        scanf( "%d",&vector[i]);
    }
    //Ciclo for para mostrar los elementos del vector
    for(i=0; i<cant; i++ ) {
        printf("El elemento de la posicion %d es %d \n",i,vector[i]);
    }
    return 0;
}
```

Cabe destacar que en el ejemplo anterior utilizamos #define para definir una constante con el valor de la dimension del vector (cant), este programa es muy sencillo pero en programas más complejos o extensos es probable que lo usemos más de una vez y si en algún momento surge cambiar dicha dimensión solo cambiando la constante alcanzara el resto del programa.

Inicialización de un vector.

Una vez creado un vector, sus elementos presentan valores arbitrarios. Dependiendo del uso que vayamos a darle al vector es la forma en que vamos a inicializarlo ya sea con valores al azar o por ejemplo inicializarlo en o si es que se va a comportar como un acumulador.

El lenguaje C permite la inicialización de un vector en el momento de declararlos. La forma o sintaxis general de inicialización de un vector es la siguiente:

```
<Especificador_de_tipo><Identificador_de_variable>[tamaño]={Lista};
```

La lista son los elementos o valores que contendrá el vector cuyo tipo es compatible con especificador_de_tipo, cada valor se va colocando en la primera posición del vector, en la segunda y así sucesivamente teniendo en cuenta que cada valor va separado por coma.

Veamos los ejemplos a continuación donde se muestran distintas formas de inicializar vectores:

```
//Declara e inicializa con números de 1 a 10
int notas[10]={1,2,3,4,5,6,7,8,9,10};
//Declara e inicializa, de esta forma define el tamaño a 4
int valores[]={2,5,4,1};
int suma[5]={0};//Inicializa en 0 todas la posiciones del vector
int suma[5]={0,0,0,0,0};//Es equivalente al ejemplo anterior
int vector[cant]; //Vector de enteros, cant puede ser una constante
//Ciclo for para inicializar en 0 un vector
for(i=0; i<5; i++ ) {
        vector[i]=0;
    }
```

En la línea cuatro de este ejemplo (valores[]), el compilador deduce que el tamaño del vector es 4, se infiere a partir de la cantidad de valores que aparecen en la lista a la derecha del igual. Si estás comenzando a programar no es aconsejable que uses esta forma de declarar vectores, es preferible optar por una que declare de manera explícita el tamaño del vector.

Arrays bidimensionales, matrices.

El lenguaje C soporta arrays multidimensionales. La forma más simple de un array multidimensional es el array bidimensional. Podríamos decir que este tipo de array es en esencia un array de vectores.

Una estructura de este tipo es un array con dos índices (tabla o matriz), es decir que equivale a una tabla con múltiples filas y múltiples columnas, por lo tanto, para localizar o almacenar un valor se deben especificar dos posiciones.

Declaración de una matriz

La forma o sintaxis general de declarar un array bidimensional es:

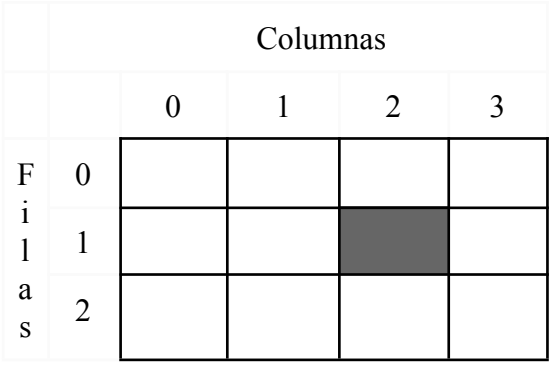
```
<Especificador_de_tipo><Id_de_variable>[tamaño 1][tamaño 2];
```

Dónde tamaño 1 y tamaño 2 si establecemos una analogía con una matriz o tabla diremos que se corresponden con las filas y las columnas de la misma.

Ejemplo:

```
int matriz[3][4];
```

Gráficamente esta matriz la vemos de la siguiente forma:



Como se mencionó anteriormente en el tema de vectores, las posiciones en C arrancan desde 0 es por eso que como muestra la gráfica, en una matriz de 3 x 4 las filas van hasta el número 2 y las columnas hasta el número 3.

Cabe destacar que siempre el primer índice, indica la fila y el segundo índice indica la columna, de esta forma si queremos hacer referencia a la posición resaltada en el gráfico usaremos la sintaxis, matriz[1][2].

También, es importante destacar que si las filas van de 0 a m y la columnas de 0 a n, el número de elementos que tendrá el array será el producto de (m+1) x (n+1).

Operaciones con matrices

Entrada y salida

Las operaciones sobre una matriz cumplen las mismas características que mencionamos con los vectores, para operarlos hay que acceder a sus posiciones una a una, a diferencia de estos cuando trabajamos con matrices corresponde entonces usar dos ciclos ya que es necesarios recorrer filas y columnas.

En el ejemplo que vemos a continuación se carga y se muestra una matriz de 3 filas por 4 columnas con números consecutivos del 1 al 12.

```
#include <stdio.h>
#include<stdlib.h>
main()
{
    int mat[3][4];//Declaración de una matriz de 3 x 4
    int i,j;      //Variables enteras que manejan los índices de la matriz
    system("cls");
    // Ciclos for para cargar la matriz
    for(i=0; i<3; i++ ) {
        for (j = 0; j < 4; j++)
        {
        printf( "Ingrese el dato de la posicion %d, %d ",i,j);
        scanf( "%d",&mat[i][j]);
        }
     }
     system("cls");
     // Ciclos for para mostrar la matriz
     for(i=0; i<3; i++ ) {
        for (j = 0; j < 4; j++)
        {
        printf( "El dato de la posicion %d, %d es %d \n",i,j,mat[i][j]);
        }
     }
     return 0;
}
```

Inicialización de una matriz.

Los arrays bidimensionales se pueden inicializar del mismo modo que un array unidimensional, en la zona de declaración. Consta de una lista de valores separados por comas y encerradas entre llaves, algunos ejemplos que se muestran a continuación:

```
//Declaración e inicialización de una matriz de 2 x 3
int tabla[2] [3]= { 1, 2, 3, 4, 5, 6};
//Estas dos formas representan lo mismo que la anterior pero visualmente son
//más gráficas
int tabla[2][3]={{1,2,3},{4,5,6}};
int tabla[2][3]={{1,2,3},
                 {4,5,6}
                };
//Declaración de una matriz de datos de tipo float
float tabla[2][2]={2.3,5.8,2.1,4.2};
//Declaración de una matriz, la cantidad de filas es indeterminado el
//segundo índice siempre es obligatorio
int tabla[][2]={1,2,3,4,5,6};

//Inicializar en cero una matriz
int tabla[2][3]={0};
```

Tanto los arrays unidimensionales como los multidimensionales se pueden inicializar con valores específicos tal y como se muestra en los ejemplos citados. Se pueden inicializar con uno o más elementos, si no se inicializan con la cantidad exacta de elementos el lenguaje C rellena el resto con ceros como en el siguiente ejemplo:

```
//En estos casos el resto de las posiciones se cargan con cero
int vector [3]= {1,2};
int matriz [2][3]={1,2,3,4};
```

Es importante tener en cuenta que el lenguaje C no comprueba límites en los arrays. Se puede sobrepasar cualquier extremo y escribir en alguna otra variable de datos o incluso en el código del programa. Es tarea del programador comprobar que no se excedan esos límites.
Otra cuestión importante es que muchas veces no podemos predecir con precisión cuál será el tamaño del arreglo y en general lo definimos pensando en la mayor cantidad de datos que pueda contener, lo que muchas veces nos lleva a desperdiciar la memoria. Así, por ejemplo, si defino un vector de 50 elementos, y sólo uso 10 posiciones, las 40 posiciones restantes están desperdiciadas.