CFGM SISTEMAS MICROINFORMÁTICOS Y REDES
SISTEMAS OPERATIVOS MONOPUESTO
TEMA 4 - PRÁCTICA 01

Conceptos

1. Concepto y declaración de tablas

Las tablas son conjuntos secuenciales de datos del mismo tipo agrupados bajo el nombre de una sola variable. Cada dato individual de una tabla se denomina *elemento*. La declaración de una tabla de una dimensión sigue la siguiente sintaxis:

```
tipoElemento nombreTabla [tamaño];
```

donde *tipoElemento* corresponde con el tipo de dato de los elementos que forman parte de la tabla, *nombreTabla* es el nombre de la variable que referencia a la tabla que se está declarando, y *tamaño* es el número máximo de elementos que puede contener la tabla. Por ejemplo, una tabla de enteros, llamada *v*, con capacidad para 10 elementos será declarada como sigue:

```
int v[10];
```

Una tabla multidimensional se puede ver como una tabla de tablas unidimensionales, en la que cada nueva dimensión añade un nivel más de tablas unidimensionales. Su declaración sigue la siguiente sintaxis:

```
tipoElemento nombreTabla [tamaño1][tamaño2]…[tamañoN];
```

donde *tamaño1*, …, *tamañoN* corresponden con el número de filas o columnas que podrán ser alojadas en la dimensión a la que se refiere. El número de elementos que podrá alojar la tabla será *tamaño1* x *tamaño2* x… x *tamañoN*.

Por ejemplo, una tabla bidimensional *w* que alojará 4 filas y 5 columnas (por tanto, 20 elementos) de enteros se declarará como sigue:

```
int w[4][5];
```

Una buena práctica de programación, en lo que se refiere a la declaración y uso de tablas, es la definición del tamaño de las tablas usando constantes simbólicas, definidas empleando la directiva de compilación *define*. Teniendo en cuenta esta buena práctica de programación, la definición de las tablas v y *w* quedarían modificadas respectivamente como sigue:

```
#define TAM 10
  ...
int v[TAM];
```

```
#define TAM1 4
#define TAM2 5
  ...
int w[TAM1][TAM2];
```

Las constantes simbólicas que indican el tamaño de la tabla se emplearán en las sentencias del programa que necesiten conocer a priori el tamaño de la tabla. De esta forma, si decide cambiar su programa de tal forma que la tabla tenga otra capacidad sólo será necesario hacer un pequeño cambio para variar el valor de dicha constante simbólica, ahorrándose el proceso de revisión y modificación del programa entero.

A partir del estándar C99, se incorpora a C la posibilidad de definir tablas utilizando una variable para determinar su tamaño. Así, el programador puede elegir definir la tabla cuando conozca su tamaño y asignarle consumir exactamente la memoria que necesita. Por ejemplo, este código lee el tamaño de un vector de enteros y una vez lo conoce define el vector exactamente de ese tamaño:

```
int tam;
printf("Introduzca la longitud del vector\n");
scanf("%d",&tam);
int v[tam];
```

Esta es una buena práctica que se deberá seguir siempre que sea posible (en ocasiones no podemos conocer la longitud *a priori*).

2. Inicialización de tablas

Las tablas, de manera opcional, pueden ser inicializadas durante su definición como cualquier otra variable. Para ello habrá que añadir a la estructura anterior lo siguiente:

```
tipoElemento nombreVector [tamaño] = {ele1,ele2,...};
```

donde ele_i es el valor que asociamos al elemento i-ésimo de la tabla. Por ejemplo, podríamos declarar una tabla *p* que contenga los 4 primeros múltiplos de 2 de la siguiente forma:

```
int p[] = {2,4,6,8};
```

De la misma forma, para declarar una tabla multidimensional como por ejemplo la matriz *m* con dos filas y tres columnas siguiente:

$$m = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Para ello, deberemos de hacer la siguiente definición incluyendo la inicialización:

```
int m[2][3] = {{1,2,3},{4,5,6}};
```

Observe que en la declaración de *p* no ha sido necesario indicar el número de elementos de la tabla ya que al ser inicializada el compilador determina automáticamente éste. Si se indica el tamaño y éste es menor que la lista de elementos para inicializar, los elementos sobrantes serán ignorados. En caso de que el tamaño sea mayor que la lista de elementos para inicializar se rellenará el espacio sobrante de la tabla con el número 0. Esto sólo ocurre cuando la inicialización se realiza de esta manera. En cambio, si la inicialización de la tabla es llevada a cabo con posterioridad a la declaración, aquellas posiciones no inicializadas contendrán valores no válidos, denominados habitualmente "*basura*".

Observe que en el caso de las tablas multidimensionales es obligatorio indicar el tamaño de cada dimensión. Observando la inicialización se puede ver claro que una tabla multidimensional es una tabla de tablas. Fíjese en que la inicialización del ejemplo se corresponde a la inicialización de una tabla de dos componentes, siendo cada componente las tablas *{1, 2, 3}* y *{4, 5, 6}*. En el caso de que la tabla tenga más dimensiones será necesario abrir una nueva llave por cada dimensión, e incluir dentro de ella, de izquierda a derecha, los elementos que contendrá.

En el caso de que en la inicialización no se haya indicado el valor de todos los elementos de la tabla multidimensional, los elementos omitidos serán inicializados automáticamente por el compilador a cero.

3. Acceso a los componentes de una tabla

El valor de un elemento de una tabla puede ser accedido empleando su índice en cada una de las dimensiones. Para ello se empleará el nombre de la variable que representa la tabla seguido del índice asociado al elemento en el que estamos interesados encerrado entre corchetes. El índice de un elemento es la posición que ocupa en la tabla menos uno. Por ejemplo, para hacer que el cuarto elemento, de una tabla unidimensional v de enteros, almacene el valor 100 se emplea esta sentencia:

```
v[3] = 100;
```

Por ejemplo, empleando la matriz *m* anterior, si deseamos acceder al elemento situado en la fila 2, columna 3, y asignarle el valor 30, se empleará la sentencia:

```
m[1][2] = 30;
```

Observe que el valor de índice para acceder al cuarto elemento de la tabla *v* es 3. Es muy importante recordar que los índices de las tablas en C van desde el 0 para el primer elemento, hasta el tamaño de la tabla menos uno, para el último elemento.

Usando la misma sintaxis, se puede obtener el valor almacenado en un elemento de una tabla. Por ejemplo, para imprimir por pantalla el valor del quinto elemento de la tabla v se emplea la siguiente sentencia:

```
printf("%d",v[4]);
```

En el lenguaje C no es posible operar con las tablas como un todo. Cualquier operación que deseemos realizar sobre las componentes de una tabla deberá de hacerse a cada uno de los elementos por separado. Por tanto, en estas ocasiones será necesario el empleo de bucles *for* para realizar la tarea que necesitemos. Por ejemplo, si deseamos multiplicar por dos el valor de los elementos de una tabla de enteros *v* debemos usar un programa similar al siguiente:

```
#define TAM 5
...

int v[TAM];
int i;
...

for(i=0;i<TAM;i++)
        v[i] =v[i] * 2;
...
```

Observe en el código anterior el uso de la constante simbólica *TAM*, y las ventajas que ello conlleva a la hora de alterar el tamaño del vector. Adicionalmente, note que los valores que tomará la variable *i* para acceder a las componentes variarán desde 0 hasta *TAM-1*.

En el caso de las tablas multidimensionales, como norma general, necesitaremos una variable contador y un bucle *for* por cada dimensión, estando dichos bucles anidados. Por ejemplo, si deseamos alterar los elementos de una tabla tridimensional *n* de enteros, con tamaño 3x5x4, sumándoles el valor 5 emplearíamos un código similar al siguiente:

```
#define TAM1 3
#define TAM2 5
#define TAM3 4
...

int n[TAM1][TAM2][TAM3];
int i;
int j;
int k;
...

for(i=0;i<TAM1;i++)
    for(j=0;j<TAM2;j++)
        for(k=0;k<TAM3;k++)
            n[i][j][k] = n[i][j][k] + 5;
...
```

Observe el anidamiento de los tres bucles en el código anterior. Este anidamiento nos permite recorrer todas las dimensiones de la tabla y por tanto acceder a todos los elementos de la misma.

Experimentos

E1. a) Analice y ejecute este código. Observe cómo se inicializan los elementos de la tabla, y cómo se accede a los mismos usando su índice.

```
#include<stdio.h>

void main(void){
    int v[] = {1,2,3,5,7,11,13,17};
    int i;

    for(i=0;i<8;i++)
        printf("%d\n",v[i]);

}
```

b) Modifique el anterior código de tal forma que se indique un tamaño de la tabla *v* menor que 8. No modifique el límite establecido para la variable *i* en el bucle *for*. Ejecute de nuevo el código y observe que ocurre con los valores sobrantes en la inicialización ¿Qué se imprime por pantalla? ¿Por qué?

c) Modifique el código anterior de tal forma que se indique un tamaño de la tabla *v* mayor que 8. Así mismo, modifique el límite para la variable i en el bucle *for* para que coincida con el nuevo tamaño. Ejecute el código y observe qué ocurre con los elementos no inicializados ¿Qué se imprime por pantalla? ¿Por qué?

d) Finalmente, dejando el tamaño de la tabla a un valor superior a 8, modifique el programa anterior inicializando la tabla elemento a elemento después de su declaración (v[0]=1; ... ; v[7]=17;). ¿Qué se imprime por pantalla en esta ocasión? ¿Por qué?

E2. El programa objeto de este experimento ofrece como salida por pantalla el contenido de una tabla tridimensional de tamaño 2x4x3. Dicha tabla contendrá números enteros y será inicializada de tal forma que su primera capa (primer elemento de la primera dimensión) esté formada por la lista de números impares comenzando por el 1, de tal forma que al terminar una fila se pasará a la siguiente hasta completar los 12 elementos que puede albergar la capa. De igual forma será inicializada la segunda capa, pero con números pares partiendo del número 2.

a) Ejecute el siguiente código y observe los resultados que se muestran por pantalla.

```
#include<stdio.h>
#define TAM1 2
#define TAM2 4
#define TAM3 3

void main(void){
    int m[TAM1][TAM2][TAM3] = {
                            {
                                {1,3,5},
                                {7,9,11},
                                {13,15,17},
                                {19,21,23}
                            },
                            {
                                {2,4,6},
                                {8,10,12},
                                {14,16,18},
                                {20,22,24}
                            }
                        };
    int i,j,k;

          for(i=0;i<TAM1;i++){
       printf("k=   ");
       for(k=0;k<TAM3;k++)
              printf("%2d ",k);
       printf("\n---------------- i=%d\n",i);
       for(j=0;j<TAM2;j++){
              printf("j=%d |",j);
              for(k=0;k<TAM3;k++)
              printf("%2d ",m[i][j][k]);
              printf("|\n");
       }
       printf("----------------\n");

    }

}
```

b) Analice el código anterior y estudie como se ha implementado cada parte para que ofrezca dicho resultado por pantalla.

c) Ejecute paso a paso el código anterior para observar el funcionamiento del mismo.

Ejercicios

EJ1. Escriba un programa que lea desde teclado 6 números correspondientes a la combinación ganadora de la lotería primitiva. A continuación, el programa leerá otros 6 números correspondientes a un boleto con el que se participa en dicho sorteo. Finalizada la lectura, el programa deberá de indicar el número de aciertos del boleto cuyos números hemos introducido.

EJ2. Escriba un programa que pida desde teclado los elementos enteros de una tabla de tamaño 3x4. Una vez leída la tabla se mostrará ésta por pantalla.

EJ3. Escriba un programa que lea el tiempo, en minutos, en que se produce el paso por meta de 5 ciclistas. Una vez leídos los 5 números el programa deberá de imprimir por pantalla la lista de tiempos de los ciclistas con respecto al primero. Por ejemplo, si los ciclistas entran por meta en los minutos 5, 6, 8, 11 y 15 el programa deberá de devolver la lista de diferencias de tiempos siguiente 0, 1, 3, 6, 10.

Problemas

P1. Escriba un programa que lea desde teclado dos tablas bidimensionales de tamaño fijo (seleccione usted el tamaño). El programa calculará la tabla correspondiente a la suma de las tablas introducidas, y mostrará la tabla resultante por pantalla.

P2. Escriba un programa que dada una tabla de enteros de tamaño fijo declarado con una constante simbólica, lea desde teclado dicha tabla, componente a componente, y la imprima por pantalla. Posteriormente, el programa debe calcular la tabla inversa (invirtiendo el orden de las componentes de la tabla) sobre la misma tabla, es decir, sin definir ninguna tabla auxiliar, e igualmente la imprimirá por pantalla.

P3. Escriba un programa que rellene automáticamente una tabla bidimensional (de tamaño prefijado por usted) con la serie de números primos comenzando desde 1, de tal forma que una vez agotado el espacio de una fila se pasará a la siguiente.

P4. Escriba un programa que lea desde teclado un vector de caracteres (uno a uno) de tamaño fijado por una constante simbólica. Una vez leído, el programa debe indicar si dicho vector forma un palíndromo. Un palíndromo es una palabra, frase o número que se lee igual de derecha a izquierda que de izquierda a derecha, por ejemplo: "*reconocer*". Nota: No trate la tabla de caracteres como una cadena.

P5. Escriba un programa que inicialice una tabla bidimensional de caracteres, de tamaño fijado por constantes simbólicas, con una serie aleatoria de caracteres (emplear para la generación aleatoria de caracteres el código adjunto). Una vez rellena la tabla se mostrará ésta y se permitirá al usuario pulsar un carácter. Pulsado el carácter se recorrerá la tabla y se sustituirán los elementos que coincidan con dicho carácter por un asterisco. El programa seguirá repitiéndose hasta que el usuario pulse la tecla Esc (cuyo código ASCII es 27).

```
#include<stdlib.h>
...
char letra;
srand((unsigned)time(NULL));
...
letra = rand()%('z'-'a') + 'a';
```

Recuerde que se requiere que incluya la cabecera de la librería *stdlib*.