Sesión 5: Vectores (2h)

En esta sesión de laboratorio seguiremos trabajando el tema 3 de la asignatura (tipos de datos estructurados) y, en concreto, el tipo de dato *vector*. Se introducirá la manipulación de variables de tipo vector y se realizarán operaciones básicas sobre ellas (recorrido, búsquedas, inserción, eliminación, ordenación, etc.). El objetivo de esta sesión es que el alumno se familiarice con los vectores y sea capaz de declarar variables de tipo vector y manipularlas correctamente en programas escritos en C.

Ejercicio 1:

Indique cómo se declaran e inicializan en C las siguientes variables, teniendo en cuenta la información que van a almacenar:

(a) La variable `frase` debe almacenar una frase de como máximo 200 caracteres y se quiere inicializar con la frase "Esto es una frase."
(b) La variable `v1` debe almacenar una secuencia de como máximo 20 números reales y se quiere inicializar todas sus posiciones a 1.0
(c) La variable `vfechas` debe almacenar una secuencia de como máximo 5 fechas, donde una fecha consta de un día, un mes y un año. Se quiere inicializar el vector con las siguientes fechas: 1 de Enero de 1945, 7 de Julio de 1999, 12 de Abril de 2000, 16 de Octubre de 1989 y 8 de Marzo de 2012.

Compruebe si las declaraciones son correctas escribiendo un programa (en el fichero *sesion5_ej1.c*) que contenga las declaraciones e inicializaciones de las variables y el código para mostrar por pantalla el valor que almacenan. Tenga en cuenta que para la variable `frase` solo se debe mostrar la frase que almacena hasta el carácter punto (`'.'`), incluido. El resto del vector `frase` no se debe mostrar por pantalla.
Ejemplo de ejecución:

```
Valor de frase:
Esto es una frase.

Valor de v1:
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0

Valor de vfechas:
1/1/1945, 7/7/1999, 12/4/2000, 16/10/1989, 8/3/2012
```

Ejercicio 2:

Escriba un programa (en el fichero *sesion5_ej2.c*) que primero pida al usuario que introduzca por teclado una frase acabada en '.' y la almacene en un vector de caracteres. A continuación el programa debe codificar la frase y mostrarla por pantalla. La frase codificada debe quedar almacenada en otro vector de caracteres diferente al que almacena la frase original.
Para la codificación usaremos el cifrado César, también conocido como cifrado por desplazamiento. Consiste en sustituir cada letra del texto original por otra letra que se encuentra a un número fijo de posiciones más adelante en el alfabeto. En nuestro programa usaremos un desplazamiento de 4. Por lo tanto, la A será sustituida por E, la B

por F, …, la W por A, la X por B, la Y por C y la Z por D. Los caracteres numéricos se sustituyen de la misma manera: 0 por 4, 1 por 5, …., 6 por 0, 7 por 1, 8 por 2 y 9 por 3. Los signos de puntuación y los espacios no se sustituyen. Suponga que la frase (incluyendo el punto) tiene como máximo 100 caracteres. La capitalización de las letras se debe mantener en la frase codificada.
Ejemplo de ejecución (en negrita los datos introducidos por el usuario):

```
Introduzca una frase acabada en punto (<=100 caracteres):
ERROR XZ34: persona no identificada.
Frase codificada:
IVVSV BD78: tivwsre rs mhirxmjmgehe.
```

Ejercicio 3:

a) Complete el programa del fichero *sesion5_ej3a.c* para que pida al usuario que introduzca un valor entero e indique por pantalla si el vector `vdesord` contiene dicho valor. En caso de encontrarlo se debe indicar por pantalla la posición donde se encuentra la primera aparición. Tenga en cuenta que los elementos del vector `vdesord` no siguen ningún orden.
Ejemplos de ejecución (en negrita los datos introducidos por el usuario):

```
Contenido del vector vdesord:
vdesord[0]=3          vdesord[1]=56
vdesord[2]=23         vdesord[3]=109
vdesord[4]=238        vdesord[5]=32
vdesord[6]=56         vdesord[7]=67
vdesord[8]=10         vdesord[9]=88

Introduzca un valor entero: 56
La primera aparicion del valor 56 se encuentra en la posicion 1
del vector
```

```
Contenido del vector vdesord:
vdesord[0]=3          vdesord[1]=56
vdesord[2]=23         vdesord[3]=109
vdesord[4]=238        vdesord[5]=32
vdesord[6]=56         vdesord[7]=67
vdesord[8]=10         vdesord[9]=88

Introduzca un valor entero: 88
La primera aparicion del valor 88 se encuentra en la posicion 9
del vector
```

```
Contenido del vector vdesord:
vdesord[0]=3          vdesord[1]=56
vdesord[2]=23         vdesord[3]=109
vdesord[4]=238        vdesord[5]=32
vdesord[6]=56         vdesord[7]=67
vdesord[8]=10         vdesord[9]=88

Introduzca un valor entero: 676
No se ha encontrado el valor 676 en el vector
```

b) Complete el programa del fichero *sesion5_ej3b.c* para que pida al usuario que introduzca un valor entero e indique por pantalla si el vector `vorden` contiene dicho valor. En caso de encontrarlo se debe indicar por pantalla la posición donde se encuentra la primera aparición. Intente realizar una implementación eficiente, teniendo en cuenta que los elementos del vector `vorden` están ordenados de mayor a menor.
Ejemplos de ejecución (en negrita los datos introducidos por el usuario):

```
Contenido del vector vorden:
vorden[0]=238   vorden[1]=109
vorden[2]=88    vorden[3]=67
vorden[4]=56    vorden[5]=56
vorden[6]=32    vorden[7]=23
vorden[8]=10    vorden[9]=3

Introduzca un valor entero: 56
La primera aparicion del valor 56 se encuentra en la posicion
4 del vector
```

```
Contenido del vector vorden:
vorden[0]=238   vorden[1]=109
vorden[2]=88    vorden[3]=67
vorden[4]=56    vorden[5]=56
vorden[6]=32    vorden[7]=23
vorden[8]=10    vorden[9]=3

Introduzca un valor entero: 3
La primera aparicion del valor 3 se encuentra en la posicion
9 del vector
```

```
Contenido del vector vorden:
vorden[0]=238   vorden[1]=109
vorden[2]=88    vorden[3]=67
vorden[4]=56    vorden[5]=56
vorden[6]=32    vorden[7]=23
vorden[8]=10    vorden[9]=3

Introduzca un valor entero: 666
No se ha encontrado el valor 666 en el vector
```

Ejercicio 4:

a) Dado el tipo de dato `tvector_int` definido en el fichero *sesion5_ej4a.c*, complete el programa de dicho fichero para que muestre por pantalla los elementos del vector y, a continuación, pida al usuario que introduzca un entero y lo busque en la variable `vect`. Si no encuentra el elemento en el vector, el programa mostrará un mensaje por pantalla indicándolo. Si lo encuentra, debe eliminar ese elemento del vector. Suponga que no hay elementos repetidos en el vector. Tenga en cuenta en la implementación que el vector está ordenado de menor a mayor.
Ejemplos de ejecución (en negrita los datos introducidos por el usuario):

```
Vector inicial: 3, 23, 56, 109, 238

Introduzca un valor entero: 45
No se ha encontrado el valor 45 en el vector
```

```
Vector inicial: 3, 23, 56, 109, 238

Introduzca un valor entero: 23
Se ha encontrado el valor 23 en la posicion 1 del vector
Procedemos a eliminarlo

Vector: 3, 56, 109, 238
```

```
Vector inicial: 3, 23, 56, 109, 238

Introduzca un valor entero: 238
Se ha encontrado el valor 238 en la posicion 4 del vector
Procedemos a eliminarlo

Vector: 3, 23, 56, 109
```

b) Dado el tipo de dato `tvector_int` definido en el fichero *sesion5_ej4b.c*, complete el programa de dicho fichero para que pida al usuario que introduzca un entero y lo busque en la variable `vect`. Si no lo encuentra y el vector no está lleno, debe insertar el elemento en el vector, manteniendo el orden. En caso contrario, el programa debe mostrar por pantalla los mensajes correspondientes.
Ejemplos de ejecución (en negrita los datos introducidos por el usuario):

```
Vector inicial: 3, 10, 23, 56, 109, 238

Introduzca un valor entero: 10
Se ha encontrado el valor 10 en la posicion 1 del vector
```

```
Vector inicial: 3, 10, 23, 56, 109, 238

Introduzca un valor entero: 13
No se ha encontrado el valor 13 en el vector
Procedemos a insertarlo en la posicion 2

Vector: 3, 10, 13, 23, 56, 109, 238
```

```
Vector inicial: 3, 10, 23, 56, 109, 238

Introduzca un valor entero: 2
No se ha encontrado el valor 2 en el vector
Procedemos a insertarlo en la posicion 0

Vector: 2, 3, 10, 23, 56, 109, 238
```

```
Vector inicial: 1, 2, 3, 10, 15, 23, 56, 109, 238, 400

Introduzca un valor entero: 34
No se ha encontrado el valor 34 en el vector
El vector ya tiene 10 elementos y esta lleno.
No se puede insertar el nuevo valor.
```

Ejercicio 5:

Dado el tipo de dato `tbaraja` definido en el fichero *sesion5_ej5.c*, complete el programa de dicho fichero para que ordene un conjunto de cartas que el usuario introducirá por teclado. Cada carta está identificada por un palo (b-bastos, c-copas, e-espadas y o-oro) y un valor (entre 1 y 12). El programa debe mostrar las cartas ordenadas de la siguiente manera: primero muestras los bastos, luego las copas, a continuación las espadas y finalmente los oros. Dentro de cada palo, las cartas van ordenadas de menor a mayor según el valor. El formato de entrada y salida de las cartas se muestran en los ejemplos de ejecución. Utilice el algoritmo de ordenación que desee y suponga que el usuario introducirá entre 1 y 100 cartas. Tenga en cuenta que puede haber cartas repetidas.

Ejemplo de ejecución (en negrita los datos introducidos por el usuario):

```
Cuantas cartas tiene su baraja?  10
Introduzca las cartas separadas por guiones (o4-e10-b1...):
o3-b12-b4-e11-c3-b12-o2-e8-c9-o12
Cartas ordenadas: b4-b12-b12-c3-c9-e8-e11-o2-o3-o12
```

```
Cuantas cartas tiene su baraja?  4
Introduzca las cartas separadas por guiones (o4-e10-b1...):
o5-e8-c10-b12
Cartas ordenadas: b12-c10-e8-o5
```

Ejercicio 6:

a) Sea el tipo de dato `tlista_matriculas` definido en el fichero *sesion5_ej6a.c*. Complete el programa de dicho fichero para que dadas dos listas de matriculas (`lista1` y `lista2`), genere una nueva lista `lunion` con la unión de las dos listas. La lista `lunio` debe contener todas las matrículas de `lista1` y `lista2`, pero no puede contener repeticiones. Suponga que `lista1` y `lista2` no están vacías ni contienen elementos repetidos dentro de la propia lista, pero entre ellas pueden haber elementos comunes. Suponga que no existe ningún criterio de ordenación en las listas.

Ejemplos de ejecución:

```
Lista1: 1111-ABC, 1111-ACC, 1122-ABC, 1134-BBC
Lista2: 2222-BDF, 1134-BBC, 2223-BDF

Lista union: 1111-ABC, 1111-ACC, 1122-ABC, 1134-BBC, 2222-BDF,
2223-BDF
```

```
Lista1: 1111-ABC, 1111-ACC, 1122-ABC, 1134-BBC
Lista2: 1111-ABC, 1134-BBC, 1111-ACC

Lista union: 1111-ABC, 1111-ACC, 1122-ABC, 1134-BBC
```

b) Implemente de manera eficiente el programa anterior (en el fichero *sesion5_ej6b.c*), considerando ahora que `lista1` y `lista2` están ordenados de menor a mayor, primero por letra y luego por número de matrícula. La lista `lunion` debe quedar ordenada de la misma manera.

Ejemplos de ejecución:

```
Lista1: 1111-ABC, 1122-ABC, 1111-ACC, 0234-ADA, 1134-BBC
Lista2: 1112-ABC, 1111-ACC, 1134-BBC, 0333-FDA

Lista union: 1111-ABC, 1112-ABC, 1122-ABC, 1111-ACC, 0234-ADA,
1134-BBC, 0333-FDA
```

```
Lista1: 1111-ABC, 1122-ABC, 1111-ACC, 0234-ADA, 1134-BBC
Lista2: 1111-ABC, 1111-ACC, 1134-BBC

Lista union: 1111-ABC, 1122-ABC, 1111-ACC, 0234-ADA, 1134-BBC
```

c) Finalmente, complete el programa del fichero *sesion5_ej6c.c* para que dadas dos listas de matrículas (`lista1` y `lista2`), genere una nueva lista `linter` con la intersección de las dos listas. La lista `linter` debe contener todas las matrículas comunes a `lista1` y `lista2`. Suponga que `lista1` y `lista2` no están vacías ni contienen elementos repetidos dentro de la propia lista. Suponga que todas las listas están ordenadas de menor a mayor, primero por letra y después por número.
Ejemplos de ejecución:

```
Lista1: 1111-ABC, 1122-ABC, 1111-ACC, 0234-ADA, 1134-BBC
Lista2: 1112-ABC, 1111-ACC, 1134-BBC, 0333-FDA

Lista interseccion: 1111-ACC, 1134-BBC
```

```
Lista1: 1111-ABC, 1122-ABC, 1111-ACC, 0234-ADA, 1134-BBC
Lista2: 1112-ABC, 1113-ACC, 1734-BBC, 0333-FDA

Lista interseccion: LISTA VACIA
```

Ejercicio 7 – OPCIONAL (Problema de examen final):

En la asignatura de Informática para aprendices de super-héroes, el Profesor Oliver Queen (alias *Arrow*) pide a sus alumnos que diseñen un programa *arrow.c* para descubrir cuál es la flecha que tiene más poder dentro de un conjunto de flechas.
La flecha más poderosa es la de mayor longitud y existen dos tipos de flechas:

(1) flecha a la izquierda: un carácter < seguido inmediatamente de cero o más caracteres consecutivos =
Ejemplos: <, <=, <==
(2) flecha a la derecha: un carácter > precedido inmediatamente de cero o más caracteres consecutivos =
Ejemplos: >, =>, ==>

La longitud de una flecha es igual al número total de caracteres que la forman: 1, 2 y 3 en los ejemplos anteriores. El poder de una flecha se expresa en GigaJulios y se calcula sumando la energía liberada al detonar todos sus caracteres. Suponer la siguiente cantidad de tones (toneladas de TNT ó dinamita) para los caracteres que forman una flecha: 2 tones para cualquier tipo de cabeza <, > y 1 ton para un carácter =
(1 ton = 4.184 GigaJulios)

Escriba en lenguaje C el programa *arrow.c* para que lea desde el teclado una cadena de caracteres, calcule la longitud de la flecha con más poder que hay dentro de la cadena y muestre por pantalla la cadena junto con su poder. Si la cadena no contiene ninguna flecha, asigna -1 a esa longitud.

a) Defina la constante `TON` con el valor de 4.184 para representar los GigaJulios que vale 1 ton y `MAX_CAR` con el valor de 80. Declare la variable `cadena` para guardar una cadena de caracteres de como máximo `MAX_CAR` caracteres.

b) Pida y lea desde teclado una cadena con un máximo de `MAX_CAR` caracteres y almacénela en `cadena`. La cadena acaba con el carácter fin de línea. Este carácter se almacena en la cadena, pero no cuenta en su longitud. Deje la longitud en la variable `lon_cad`. Suponga que nunca se introduce una cadena que supere la longitud `MAX_CAR`. Declare las variables necesarias.

c) Calcule la longitud de la flecha más larga de todas las que contiene la cadena y almacénela en la variable `max_lon`. La cadena puede tener flechas de los dos tipos (hacia la derecha y/o hacia la izquierda) o no tener flechas (en este caso `max_lon` vale -1). Declare las variables necesarias.

d) Muestre por pantalla un mensaje con la cadena introducida, la longitud de su flecha más larga y su poder. Si la cadena no tiene ninguna flecha muestre un mensaje indicándolo. Declare las variables necesarias. Utilice formatos de salida como los mostrados en los ejemplos de ejecución siguientes (en negrita se indica lo que introduce el usuario):

```
Introduce la cadena: <<<<<<<<<<
La cadena: <<<<<<<<<< tiene la flecha mas larga de longitud: 1 y
poder: 8.37 GigaJulios
```

```
Introduce la cadena: =========
La cadena: ========= no tiene ninguna flecha.
```

```
Introduce la cadena: <========>
La cadena: <========> tiene la flecha mas larga de longitud: 9 y
poder: 41.84 GigaJulios
```

```
Introduce la cadena: <===>===>
La cadena: <===>===> tiene la flecha mas larga de longitud: 4 y
poder: 20.92 GigaJulios
```