Práctica 4

Fecha de entrega: 18/19 de Mayo

Objetivo: Practicar con listas, ficheros de texto, strings y búsquedas.

Esta cuarta práctica consiste en la creación de una pequeña aplicación que simule un intérprete para programas escritos en un lenguaje máquina, capaces de crear dibujos en una pantalla *virtual*.

1. El Dispositivo de Pantalla

El dispositivo de pantalla estará representado por una matriz bidimensional de caracteres de tamaño `NUM_FILAS`x`NUM_COLUMNAS` (dos constantes que pueden inicializarse a 10x20 pero que deben poder ser modificadas durante las pruebas).

Como veremos en la sección 3, desde el programa se podrá leer y escribir sobre cada una de las posiciones del dispositivo de pantalla.

2. La memoria

Para poder ejecutar las instrucciones de los programas, es necesario representar la memoria que será utilizada durante la ejecución de los programas para almacenar las variables y contadores. La memoria almacenará en cada una de sus posiciones un valor entero. El número de *celdas* de memoria disponible es muy grande (más de 30.000). Sin embargo, un programa en lenguaje máquina utilizará como mucho `MAX_POSICIONES` celdas (que puede inicializarse a 25).

Para almacenar los valores de todas las posiciones, por tanto, se utilizará una *lista* de posiciones individuales de memoria. Cada una de esas posiciones estará representada por un número entero que indica la dirección de memoria de dicha posición y un valor entero para representar el valor que está almacenado en dicha posición de memoria. Si se intenta *leer* de una posición de memoria que no aparece en la lista, se devolverá como contenido un 0.

Como veremos, existen dos posiciones de memoria con un significado especial, debido a que son utilizadas por las instrucciones máquina que operan con la pantalla:

Posición 0 : Esta posición almacena la fila del dispositivo de pantalla sobre la que se desea operar.

Posición 1 : Esta posición almacena la columna del dispositivo de pantalla sobre la que se desea operar.

El estado de la memoria puede guardarse y cargarse a un fichero. Un ejemplo de fichero de memoria es el siguiente:

```
1:2500
12:300
03:6900
1915:27
...
```

Cada línea del archivo de texto contiene una posición de memoria, utilizando el carácter ':'como separador entre la posición de memoria y el valor de la memoria. Cada línea del fichero de memoria se leerá y se irá procesando, es decir, creará una posición de memoria nueva que será insertada en la lista con posiciones de memoria. No se conoce inicialmente el número de posiciones de memoria que están almacenadas en el fichero, por lo que se irán leyendo las líneas del fichero hasta que llenemos todas las posiciones de memoria o hasta que no tengamos más líneas que leer.

La lista de posiciones de memoria debe mantenerse ordenada en memoria, utilizando su posición como campo de ordenación. Las posiciones de memoria almacenadas en el fichero pueden no estar en orden, por tanto, es necesario realizar una *inserción ordenada* de las posiciones de memoria que se van leyendo del fichero de texto.

3. El repertorio de instrucciones

Dentro del conjunto de instrucciones que pueden aparecer en un posible programa, podemos encontrar instrucciones generales para:

- Realizar operaciones aritméticas básicas:

  - `SUM [DIR1] [DIR2]`: se almacena en la posición de memoria `[DIR1]` el resultado de sumar el valor actual que ocupan las posiciones `[DIR1]` y `[DIR2]`.
  - `SUB [DIR1] [DIR2]`: igual que la anterior, pero *resta* a `[DIR1]` el contenido de `[DIR2]`.

- Cargar de valores a una posición de memoria: `STORE [DIR] VALOR`: almacena el valor `VALOR` del segundo parámetro en la posición de memoria indicado por el primero.
- Copiar valores de memoria: `COPY [DIR1] [DIR2]` copia el valor de la posición de memoria `[DIR2]` a la posición de memoria `[DIR1]`.
- Realizar saltos a otras instrucciones del programa (lo que permite implementar programas con bucles):
  - `JUMPZ [DIR1] [DIR2]`: salta a la instrucción cuya posición está almacenada en `[DIR2]` cuando el contenido de `[DIR1]` es cero. La posición de una instrucción es el lugar que ocupa dentro del programa donde está contenida (y que es descrito en la sección 4).
  - `JUMPP [DIR] [DIR2]`: igual que el salto anterior, pero salta cuando `[DIR1]` es un número *mayor o igual* que cero.
- Operaciones de lectura y escritura en pantalla:
  - `READ [DIR]`: almacena en la posición `[DIR]` de memoria el valor ordinal del carácter que está en la fila `[0]` y columna `[1]` de la pantalla (donde `[0] denota el contenido de la posición 0 de memoria, y [1] el contenido de la posición 1)`
  - `WRITE [DIR]`: almacena en la fila `[0]` columna `[1]` de la pantalla el caracter cuyo ordinal está almacenado en `[DIR]`.

La tabla 5.1 muestra un esquema de las instrucciones anteriores.

4. Los programas

Un programa no es más que una lista de instrucciones, esta lista de instrucciones tendrá un tamaño máximo `MAX_INSTRUCCIONES`, una constante que puede definirse con un valor de 50. Un ejemplo de programa puede ser:

```
SUM [2] [12]
STORE [3] 0
COPY [02] [01]
STORE [5] 10
STORE [6] 1000
```

Instrucción	Significado
SUM [DIR1] [DIR2]	[DIR1] := [DIR1]+[DIR2]
SUB [DIR1] [DIR2]	[DIR1] := [DIR1]-[DIR2]
STORE [DIR] VALOR	[DIR]:= VALOR
COPY [DIR1] [DIR2]	[DIR1]:=[DIR2]
READ [DIR]	[DIR]:=Pantalla([0],[1])
WRITE [DIR]	Pantalla([0],[1]) := chr([DIR])
JUMPZ [DIR1] [DIR2]	si [DIR1]=0 entonces salta a [DIR2]
JUMPP [DIR1] [DIR2]	si [DIR1]>=0 entonces salta a [DIR2]

Tabla 5.1: Esquema de instrucciones del lenguaje máquina

```
SUB [05] [06]
...
```

Al igual que con el fichero de memoria, habrá que leer una a una las líneas del fichero y procesar la línea para crear una instrucción que se insertará al final de la lista de instrucciones que forman el programa. Tampoco se conocen el número de instrucciones que se encuentran en el archivo por lo que es necesario leer línea a línea el fichero hasta que se llene la lista de instrucciones o hasta que se llegue al final del archivo.

5. Opciones del intérprete

Al comenzar la ejecución de la práctica, la pantalla *virtual* estará vacía (todos sus caracteres serán *espacios*), y no habrá ningún programa en lenguaje máquina cargado.

La aplicación presentará un menú con las siguientes opciones:

Cargar Programa : Se pedirá al usuario el nombre de un fichero que contiene el programa que se desea interpretar. Si hubiera algún otro programa cargado se sobreescribirá.

Ejecutar Programa : Al seleccionar esta opción se ejectuará el programa actualmente cargado, interpretando en secuencia cada una de las instrucciones de las que está compuesto el programa, modificando tanto las posiciones de memoria como el estado actual de la pantalla. Tras finalizar la ejecución del programa se presentará por pantalla el estado actual del dispositivo de pantalla.

Cargar Memoria : Se pedirá al usuario el nombre de un fichero que contiene el estado actual de la memoria, que será leídas y sobreescritas

por las instrucciones de los programas que se ejecutan. Si la memoría estaba inicializada, la memoria se sobreescribirá.

Guardar Memoria : Se pedirá al usuario un nombre de fichero que se utilizará para almacenar las posiciones de memorias que hay actualmente en uso.

Guardar Pantalla virtual : Se pedirá al usuario un nombre de fichero que se utilizará para guardar el estado actual de la pantalla.

Terminar : Se sale de la aplicación.

6. Implementación

Para la implementación de la práctica se debe, al menos:

- Crear un tipo enumerado para representar cada una de las instrucciones permitidas.
- Crear un tipo registro para representar las intrucciones de un programa; contendrá un campo del tipo enumerado anterior y y dos campos para representar los dos operados de las instrucciones. Para instrucciones con un sólo operando se rellenará el primero de los campos con el valor de su único operando.
- Crear un tipo registro para representar un programa como una lista de instrucciones.
- Crear un registro para representar un posición de memoria con su valor. La posición de memoria estará representada como un entero y el valor de la posición de memoria también estará representado como otro entero.
- Crear un tipo registro para representar la memoria como una lista de posiciones de memoria.
- Crear un tipo array bidimensional de caracteres para representar el dispositivo de pantallas.

El tipo de datos lista que se utilizará en la práctica es el tipo de datos explicado en la clase de teoría. No se admitirán soluciones con que utilicen una estructura distinta, por ejemplo, utilizando punteros.

Se deben declarar las constantes indicadas en el enunciado (`NUM_FILAS`, `NUM_COLUMNAS`, etc.). Todas ellas deben poder cambiar de valor de tal forma que la aplicación siga funcionando.

Se deben seguir las normas de estilo indicadas en la sección 2 de las "Normas de Entrega". Se debe además poner especial atención en la correcta

estructura del código, utilizando tipos de datos correctos, uso de parámetros adecuado y anidamiento de subprogramas. La ejecución del programa deberá ser clara y se validarán en la medida de lo posible las entradas del usuario.

Igual que en la práctica anterior, recuerda incluir las directivas del compilador al principio del programa:

```
{$X-}{$Q+}{$R+}{$S+}
```

7. Pruebas

Además de los casos de prueba que se deberán entregar y que son descritos en la sección siguiente, unos días antes de la finalización del plazo para la entrega de la práctica, serán publicados varios ejemplos de programas escritos en el lenguaje máquina con los que la aplicación deberá funcionar.

Cada uno de los casos ellos consistirá en un programa a ejecutar y un estado inicial de la memoria y se indicará para cada uno de ellos el resultado final esperado de la pantalla y estado de la memoria.

8. Normas de entrega

La práctica debe entregarse utilizando el mecanismo de entregas del campus virtual, no más tarde de la fecha indicada en el encabezado del enunciado.

Se recuerda que es necesario adjuntar en la memoria de la práctica el conjunto de casos de prueba con los que se ha probado la práctica. En este caso, los casos de prueba consistirán en:

Entrada : Programa a ejecutar, y estado de la memoria.

Salida : Estado de la memoria y estado de la pantalla.

Sólo uno de los dos miembros del grupo debe hacerlo, subiendo al campus un fichero llamado `grupoNN.zip`, donde `NN` representa el número de grupo con dos dígitos. El fichero contendrá una carpeta con nombre `grupoNN` en la que se incluirá código completo de la práctica, en particular el fichero `Pr4.pas`. Adicionalmente deberá incluir un fichero `alumnos.txt` donde se indique el nombre de los componentes del grupo.

Es obligatorio que todos los ficheros de código fuente puedan ser compilados con Turbo Pascal con las directivas de compilación indicadas en la sección 6.