Fundamentos de la programación y la informática
Examen práctico. 21 de Diciembre de 2018
Grado en ingeniería aeroespacial en aeronavegación
Grado en ingeniería aeroespacial en vehículos aeroespaciales
Universidad Rey Juan Carlos

Preparativos

- Ejecuta el script `prepara_fpi`

  Esto creará en tu cuenta el directorio `~/fpi.diciembre.18`

  y los ficheros

  `~/fpi.diciembre.18/coordenadas_aeropuertos.txt` y

  `~/fpi.diciembre.18/cuadrado.pas`

Ejercicio 1 (3 puntos)

Escribe un programa en Pascal en el fichero `~/fpi.diciembre.18/cuadrado.pas` que escriba en pantalla un cuadrado, de tamaño *lado*, construido con los caracteres almohadilla y espacio, según la siguiente especificación:

1. El programa pedirá al usuario un número entero entre 1 y 40, ambos inclusive. Si la entrada no es un número entero o está fuera de este rango, mostrará un error y repetirá la petición, todas las veces necesarias hasta que la entrega sea correcta.

   En el fichero tienes procedimiento que puedes aprovechar, aunque tendrás que modificarlo.

2. El programa tendrá un procedimiento llamado *escribe_cuadrado*, con un parámetro entero llamado *lado*.

3. Este procedimiento comprobará (de nuevo) que *lado* sea un número mayor o igual que 1, menor o igual que 40. Si el valor está fuera de este rango, mostrará un mensaje de error y finalizará la ejecución del programa.

4. Si *lado* vale 1, el programa escribirá

   ```
   #
   ```

5. Si *lado* vale 2, el programa escribirá

   ```
   # #
   # #
   ```

6. Si *lado* vale 3, el programa escribirá

   ```
   # # #
   #   #
   # # #
   ```

7. Para el resto de valores de *lado*, el programa se comportará de forma semejante.

Sugerencia: haz que tu programa considere los valores 1 y 2 como casos particulares, en ese caso escribirá el cuadrado usando solamente sentencias *writeln*. Para los valores entre 3 y 40, que considere el caso general, donde escriba el lado superior, los lados laterales y el lado inferior.

Solución

```
{£mode objfpc}{£H-}{£R+}{£T+}{£Q+}{£V+}{£D+}{£X-}{£warnings on}
program cuadrado;

const
    LadoMinimo = 1;
    LadoMaximo = 40;

procedure lee_lado(var lado:integer);
var
    sal : boolean = FALSE;
    s : string;
    codigo : integer;
begin
    repeat
        write('Introduce un número entero entre ', LadoMinimo);
        writeln(' y ', LadoMaximo);
        readln(s) ;
        val(s, lado, codigo);
        if (codigo = 0 ) and (lado >= 1) and (lado <= 40) then begin
            sal := True;
        end else
            writeln('Error, ',s,' no es un entero entre 1 y 40');
    until sal;
end;

procedure barra_horizontal(lado: integer; tinta:string);
var
    i : integer;
begin
    for i := 1 to lado do begin
        write(tinta);
    end;
    writeln;
end;

procedure barras_verticales(lado:integer; tinta, papel:string);
    // Las barras verticales están compuestas de lado-2 líneas
    // Cada línea es un punto de tinta, lado-2 espacios y otro punto
var
    i, j : integer;
begin
    for i := 1 to lado-2 do begin
        write(tinta);
        for j := 1 to lado-2 do begin
            write(papel);
        end;
        write(tinta);
        writeln;
    end;
end;
```

```
procedure escribe_cuadrado(lado:integer; tinta,papel:string);
begin
    if (lado < LadoMinimo) or (lado > LadoMaximo) then begin
        write('Error. Lado vale ', lado);
        writeln(' . Debería estar entre ', LadoMinimo , ' y' , LadoMaximo);
        halt;
    end;

    if (lado = 1) then
        writeln('# ')
    else if (lado = 2) then begin
        writeln('# # ');
        writeln('# # ');
        writeln;
    end
    else begin
        barra_horizontal(lado, tinta);
        barras_verticales(lado, tinta, papel);
        barra_horizontal(lado, tinta);
    end
end;

var
    lado : integer = 0;
    tinta : string = '# ';
    papel : string = '  ';

begin
    lee_lado(lado);
    escribe_cuadrado(lado, tinta, papel);
end.
```

Observaciones:

- El caso de lado 2 podría generalizarse también: una línea horizontal, cero líneas verticales y otra horizontal. Donde cada línea horizontal es un punto de tinta, cero espacios y otro punto. Pero la solución escrita aquí es más fácil de escribir y de entender.

Ejercicio 2 (3.5 puntos)

En la práctica 8.4, fichero `~/fpi/practica08/ordena.pas`, ordenaste un array sencillo donde cada elemento era un nombre y una puntuación. En este ejercicio, crearás un nuevo array solo con los *mejores* jugadores, que serán aquelos cuya puntuación sea mayor o igual a una puntuación mínima.

Copia tu programa `ordena.pas` en `~/fpi.diciembre.18/mejores.pas` y modifícalo para que siga la siguiente especificación:

1. Tendrá una función *solo_mejores* que
    - Recibirá como parámetros el array de registros y un entero *puntuacion_minima*.
    - Devolverá un array que contendrá aquellos registros de jugadores con puntuación mayor o igual a *puntuacion_minima*.
    - Si *puntuación_minima* está fuera del rango permitido (de 0 a 15), la función mostrará un error y el programa finalizará.

- El array devuelto tendrá la misma estructura que el original, esto es, será del mismo tipo y marcará la última posición con una entrada especial de nombre ZZZ.

2. El cuerpo principal del programa tendrá una variable local que usarás para dar valor a *puntuacion_minima*. No leas ningún valor desde el teclado.

3. El cuerpo principal del programa mostrará el array original sin ordenar, el array original ordenado y el array con los mejores jugadores (desordenado).

4. En tu programa, la función *solo_mejores* usará el array original sin ordenar. Pero deberá poder funcionar con cualquier array de este tipo, por ejemplo el array ordenado.

Solución

No publicamos una solución completa porque sería tanto como publicar una implementación de la práctica 8.4. Pero la función a escribir debería parecerse a la siguiente:

```
function solo_mejores( jugadores:tipo_jugadores; puntuacion_minima:integer
        ):tipo_jugadores;

var
    i : integer;  // Indice del array 'viejo'
    j : integer;  // Indice del array 'nuevo'
    nuevo_array : tipo_jugadores;
    ultimo : integer;
begin
    ultimo := posicion_ultimo(jugadores);
    j := 1;
    for i := 1 to ultimo do begin
        if (jugadores[i].puntuacion >= puntuacion_minima) then begin
            nuevo_array[j] := jugadores[i];
            j := j+1;
        end
    end;
    nuevo_array[j].nombre := 'ZZZ';
    result := nuevo_array;
end;
```

Observaciones:

- Esta función recibe el array *viejo* (*jugadores*) con todos los jugadores y la puntuación mínima para considerar a un jugador entre los mejores. Devuelve el array *nuevo*, solo con los mejores jugadores.

- La variable *i* recorre el array *viejo*. Hay que tratar el array completo, todos los elementos desde la posición 1 hasta la posición de ZZZ (exclusive), por tanto usamos un bucle *for*

- Usamos *j* como índice para el array *nuevo*. No sabemos a priori cuántos elementos tendrá. Lo iniciamos a 1, y cada vez que un elemento (un jugador) tenga una puntuación mayor o igual a la requerida, lo copiamos del array nuevo al viejo e incrementamos *j*.

- Finalmente, añadimos ZZZ tras el último elemento del array nuevo, para marcar el fin.

Alternativas: como cualquier programa, se podría hacer de muchas formas distintas, similares, peores o mejores. Comentamos aquí alguna de las alternativas principales.

- En este ejemplo usamos una función *posicion_ultimo* solamente para saber la posición de último elemento, previo a ZZZ. Para un programador principiante posiblemente esto es lo más claro, permite usar un bucle sencillo. Un programador con más experiencia probablemente preferirá recorrer la variable *i* en un bucle *while*, que se vaya incrementando mientras no se alcance la posición ZZZ del array *viejo*.

- Si no fuera porque el enunciado pide una función, también podríamos haber usado un procedimiento que reciba los dos arrays, pasando el nuevo por referencia.

Ejercicio 3 (3.5 puntos)

En la práctica 9.2, fichero `~/fpi/practica09/aeropuerto_cercano.pas`, escribiste un programa que indicaba el aeropuerto más cercano a otro aeropuerto dado. Ahora lo modificarás para devolver todos los aeropuertos que estén a una distancia menor o igual que cierto valor.

Copia tu programa `~/fpi/practica09/aeropuerto_cercano.pas` en `~/fpi.diciembre.18/cercanos.pas` y modifícalo para que siga la siguiente especificación:

1. Al igual que en la versión anterior del programa, se le pedirá al usuario un código de aeropuerto, se le notificará que puede acabar pulsando *f*, se mostrará el código y el nombre del aeropuerto. En caso de error en el código, se mostrará un error.

2. A diferencia de la versión anterior, cuando el programa disponga de un código correcto, ya no se le mostrará al usuario el aeropuerto más cercano, sino que se le pedirá que introduza una distancia, en kilómetros. Si el valor introducido no es un número real o no es un número positivo, el programa lo notificará y volverá a pedir un valor numérico real, todas las veces que sea necesario hasta recibir un número válido. Consideramos *positivo* todo número estrictamente mayor que 0.

3. El programa escribirá en pantalla todos los aeropuertos (código y nombre) que estén a una distancia menor o igual que la indicada por el usuario.

4. En todo lo demás, este programa será igual que `aeropuerto_cercano.pas`. Esto es, leerá el mismo fichero, con la misma estructura, lo meterá en un array, trabajará sobre el array, etc. Cualquier cosa que no especifique este enunciado, se entiende que es igual que la práctica 9.2.

No es necesario que elimines de esta version del programa los subprogramas necesarios para la versión anterior.

Solución

No publicamos una solución completa porque sería tanto como publicar una implementación de la práctica 9.2. Pero la función a escribir debería ser similar a esta:

```
procedure lee_distancia(var distancia: real);
var
    linea: string;
    num_real: real;
    codigo: integer;
begin
    repeat
        write('Introduce la distancia máxima, en Kms: ');
        readln(linea);
        val(linea, num_real, codigo);
        if (codigo <> 0) then
            writeln('Solo se admiten numeros reales. ')
```

```
        else if (num_real <= 0.0) then
            writeln('Solo se admiten numeros reales positivos. ')
        else
            distancia := num_real;
    until (codigo = 0) and (num_real > 0);
end;
```

```
procedure escribe_cercanos( var aeropuertos: TipoAeropuertos;
                            num_aeropuertos: integer;
                            dist_maxima: real;
                            indice_aerop_elegido: integer);
var
    i: integer;
    dist: real;
begin
    for i:=1 to num_aeropuertos do begin
        if i <> indice_aerop_elegido then begin
            dist := haversine_distance( aeropuertos[indice_aerop_elegido].latitud,
                                        aeropuertos[indice_aerop_elegido].longitud,
                                        aeropuertos[i].latitud,
                                        aeropuertos[i].longitud);
            if  (dist <= dist_maxima) then begin
                write(aeropuertos[i].codigo_IATA, ' ', aeropuertos[i].nombre, ' ');
                writeln(dist:0:3, ' Kms');
            end;
        end;
    end;
end;
```

Alternativas:

- El procedimiento *lee_distancia* podría usar una variable booleana (llamada por ejemplo *sal*) de forma que cuando valga *TRUE*, el bucle finalice. La versión escrita aquí es más compacta.
- El procedimiento *escribe_cercanos* podría recibir el código de aeropuerto inicial y sus coordenadas. Pero la versión escrita aquí es mejor, basta con pasar el índice de ese aeropuerto en el array para que el propio procedimiento obtenga estos datos.
- El procedimiento *escribe_cercanos* recibe como argumento el número de aeropuertos. También podría averiguarse desde el mismo procedimiento, bien llamando a una función que lo busque o bien incluyendo la búsqueda en el bucle principal (como indicamos en el ejercicio anterior). Pero para un programador principiante, es preferible separar estas dos cosas: por un lado la búsqueda del último elemento, y por otro, la búsqueda de los elementos que cumplan la condición.
- La sentencia *write* y la sentencia *writeln* podrían mezclarse en una única sentencia *writeln*. Pero para el programador principiante es preferible separarlo, de esta forma los posibles errores se localizan más fácilmente.