LAB09

Objetivos:

- Realización de operaciones sobre matrices
- Realización de operaciones sobre registros
- Realización de operaciones sobre listas estáticas

Lab 9. Programas Ada con Matrices, registros y Listas estáticas

9.1. Lista de ejercicios

Varios de los ejercicios que aquí se proponen se han propuesto previamente. En este laboratorio cambian el tipo de datos en el que se almacenan los datos y/o los resultados. Así, la resolución del ejercicio no radicará en los cálculos a realizar sino en la manera de acceder a los datos y/o resultados.

9.1.1. Día siguiente

Implementa el procedimiento **dia_siguiente** que, dada una fecha, la modifique y devuelva la fecha correspondiente al día siguiente. El programa debe tener en cuenta el número de días que tiene cada mes incluyendo los años bisiestos. Un año bisiesto es aquel que es divisible entre 4, salvo que sea secular (el último de cada siglo, que termina en 00), aunque los seculares que son divisibles entre 400 sí son bisiestos.

9.1.2. Polar a trigonométrica

Implementa la función **trigon** que, dado un número complejo en formato polar, nos devuelve el mismo número complejo en forma trigonométrica según las fórmulas que aparecen más abajo. En Ada existe la biblioteca Ada.Numerics.Elementary_Functions que ofrece el seno (sin), el coseno (cos).

$$b = modulo\ sen(argumento)$$

$$a = modulo \cos(argumento)$$

9.1.3. Trigonométrica a polar

Implementa la función **polar** que, dado un número complejo en formato trigonométrico, nos devuelve el mismo número complejo en forma polar. Para definir exactamente el ángulo, hace falta estudiar el signo de a y b y determinar el cuadrante en el que se encuentra (como si fuera una coordenada en el plano).

La biblioteca *Ada.Numerics.Elementary_Functions* ofrece la función *arcotangente* (arctan).

$$modulo = \left|\sqrt[2]{a^2 + b^2}\right|$$

$$argumento = \begin{cases} arctan\left(\frac{b}{a}\right) & si\ a > 0\ y\ b > 0 \\ 2\pi + arctan\left(\frac{b}{a}\right) & si\ a > 0\ y\ b < 0 \\ \pi + arctan\left(\frac{b}{a}\right) & si\ a < 0 \\ +\frac{\pi}{2} & si\ a = 0\ y\ b > 0 \\ +\frac{3\pi}{2} & si\ a = 0\ y\ b < 0 \\ 0 & si\ a = 0\ y\ b = 0 \end{cases}$$

Al analizar las fórmulas, ten cuidado con aquellos valores que hacen que el cálculo no sea posible (¿qué pasa si a contiene el valor 0?) pero que se puede obtener un resultado válido como solución. Se sugiere revisar las fórmulas por si hubiera algún caso más con las mismas características (que lo hay).

9.1.4. ¿Está en lista estática?

Implementa la función **esta** que, dado un valor entero N y una lista estática con valores que pueden no estar ordenados, diga si el entero está o no en el vector.

9.1.5. Posición en lista estática

Implementa la función **posición** que, dado un valor entero X y una lista estática cuyos valores pueden no estar ordenados, diga en qué posición se encuentra X en el vector. Si dicho entero estuviera repetido, bastará con devolver la posición de la primera de sus apariciones; y en caso de no encontrarse en el vector, se devolverá el valor *Integer'Last*.

9.1.6. ¿Son iguales? (lista estática)

Implementa la función **son_iguales** que, dadas dos listas estáticas, devuelva true si el contenido de las dos listas es el mismo (mismos elementos en el mismo orden) y false si no.

9.1.7. Eliminar elemento de lista estática

Crear el procedimiento **eliminar_elemento** que, dado un entero num y una lista estática de enteros L, la modifique eliminando todas las apariciones del elemento num (si existe).

9.1.8. Insertar elemento en lista estática

Implementa el procedimiento **insertar_elemento_en_pos** que, dados dos enteros *num* y *pos,* y una lista estática con espacio para un entero más, inserte en la lista el número *num* en la posición *pos* dentro de la lista, desplazando los elementos que corresponda una posición a la derecha si la posición es un número entre 1 y el numero de elementos de la lista más uno. Si la posición no está en ese rango, se ignora y se deja la lista tal y como está.

9.1.9. Posición del máximo en lista estática

Implementa la función **pos_maximo** que, dada una lista estática de enteros, devuelva el valor del índice en el que se encuentra el máximo de esa lista.

9.1.10. Ordenar por método de la burbuja en lista estática

Implementa el procedimiento **ordenar_burbuja** que, dado una lista estática de enteros, ordene sus elementos de mayor a menor.

9.1.11. Posición de un número en una matriz

Implementa el procedimiento **posición** que, dada una matriz de N x M enteros (N>0 y M>0) y un valor entero Num, devuelva la posición en la que se encuentra Num (es decir, sus coordenadas Fila y Columna). Si el entero está repetido, se devolverá la posición de la fila más alta y, si aún así hay varios, entonces la de más a la izquierda. Si Num no está en la matriz, se devolverá como posición la fila 0 y la columna 0.

9.1.12. Máximo elemento de una matriz

Implementa el procedimiento **maximo** que, dada una matriz de enteros de dimensiones N x M (N,M>0), calcule el valor máximo y la posición en la que se encuentra. Si el valor máximo está repetido, devolver la posición de la primera de sus apariciones en la matriz contando por filas de arriba abajo y en cada fila de izquierda a derecha (o sea, como en el ejercicio anterior).

9.1.13. Número Complejo

Implementa el procedimiento que realiza la conversión de coordenadas trigonométricas a coordenadas polares o viceversa. Usa el tipo registro variante que

se proporciona. (uno que tiene los datos de un número complejo de un tipo u otro).

9.1.14. Datos Alumno de primero

Hacer un registro con los datos de un alumno de primero. Los datos cambian las asignaturas en las que está matriculado. Además, el alumno, en cada asignatura puede estar en evaluación continua (3 notas una en cada semana de horario agrupado y el día del examen final vs solo la nota del examen final). Hacer que el programa recoja las notas de 10 asignaturas en las que está matriculado y actualice los datos del alumno.

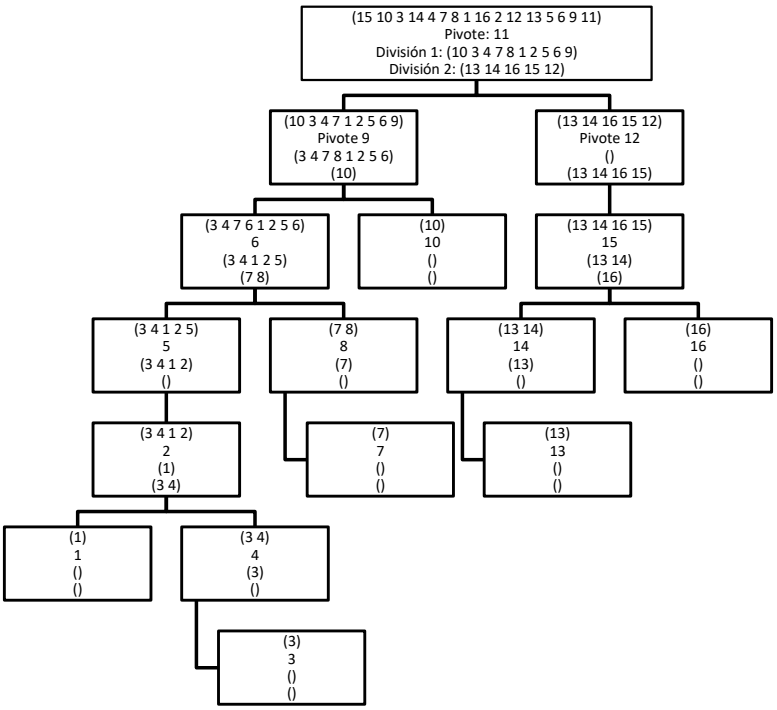
9.1.15. Ordenar un vector usando QuickSort

El método de ordenación *QuickSort* es un ejemplo de la técnica de programación llamada *divide y vencerás.* El método de ordenación consiste en tomar un elemento de la lista a ordenar al que se le llama *pivote* y dividir la lista original en dos: los elementos menores que el pivote y los elementos mayores que el pivote y aplicar de nuevo el algoritmo a las dos sublistas creadas. En cada vuelta, la lista a ordenar es más pequeña. Este algoritmo es más eficiente que cualquiera de los métodos de ordenación que se han visto en clase.

Una manera de hacer la división es tomar el elemento final de un vector como pivote y considerar que el índice (provisional) del pivote es la primera posición del vector. Luego, recorrer el vector y, cada vez que se localiza un elemento menor que el pivote, la posición del pivote es una más a la derecha (porque nos hemos topado un elemento que va a la izquierda del pivote) y se intercambian los valores del elemento localizado

con la del índice provisional del pivote. Así sucesivamente hasta recorrer todos los elementos del vector.

Una vez repartidos los elementos del vector en menores y mayores que el pivote, entonces se vuelve a realizar el algoritmo solo para los elementos menores y a los mayores. Por ejemplo, dado el vector de enteros (15 10 3 14 4 7 8 1 16 2 12 13 5 6 9 **11**), se toma el 11 como pivote (podría ser cualquier elemento del vector, pero hemos decidido que sea el último elemento del vector). Y se procede a dividir el vector en dos listas dejando el pivote en medio: (10 3 4 7 8 1 2 5 6 9 **11** 13 14 16 15 12). Ahora hay que ordenar (10 3 4 7 8 1 2 5 6 9) y (13 14 16 15 12). En la primera división el pivote es 9 y las subdivisiones son (3 4 7 8 1 2 5 6) y (10). A la segunda división, el pivote es 12 y las subdivisiones correspondientes son () y (13 14 16 15). Y así se sigue haciendo hasta que la lista a ordenar ya no tenga elementos. En la figura siguiente se muestran las distintas subdivisiones a las que daría lugar partiendo de la lista inicial.



9.1.16. Patrón de valores

Implementa una función que, dado un número positivo, cree una matriz de dos dimensiones con los valores que siguen un patrón como los siguientes (se muestran las matrices para 1, 3, 4 y 6).

1	1
3	3 3 3 3 3 3 2 2 2 3 3 2 1 2 3 3 2 2 2 3 3 3 3 3 3

4	4 4 4 4 4 4 4 4 3 3 3 3 3 4 4 3 2 2 2 3 4 4 3 2 1 2 3 4 4 3 2 2 2 3 4 4 3 3 3 3 3 4 4 4 4 4 4 4 4
6	6 6 6 6 6 6 6 6 6 6 6 6 5 5 5 5 5 5 5 5 5 6 6 5 4 4 4 4 4 4 4 5 6 6 5 4 3 3 3 3 3 4 5 6 6 5 4 3 2 2 2 3 4 5 6 6 5 4 3 2 1 2 3 4 5 6 6 5 4 3 2 2 2 3 4 5 6 6 5 4 3 3 3 3 3 4 5 6 6 5 4 4 4 4 4 4 4 5 6 6 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6

9.1.17. Patrón de valores en espiral

Implementa un procedimiento llamado *espiral* que dada una matriz cuadrada de orden impar, inicialice sus valores a enteros utilizando una espiral. Primero, se coloca el elemento central y después se introducen los valores en sentido antihorario. Los valores tienen un valor inicial y se van incrementando sucesivamente.

A continuación, se presentan los resultados de la inicialización cuando el valor inicial es 1 y se aportan, para inicializarse, matrices de 1x1, 3x3, 5x5 y 7x7.

N	Matriz resultante
1	1
3	5 4 3 6 1 2 7 8 9

5		17	16	15	14	13	
		18	5	4	3	12	
		19	6	1	2	11	
		20	7	8	9	10	
		21	22	23	24	25	
7	37	36	35	34	33	32	31
	38	17	16	15	14	13	30
	39	18	5	4	3	12	29
	40	19	6	1	2	11	28
	41	20	7	8	9	10	27
	42	21	22	23	24	25	26
	43	44	45	46	47	48	49

9.2. Registros con tamaño no definido en tiempo de compilación

Los registros que hemos visto hasta el momento tienen un tamaño que se calcula en tiempo de compilación. Pero también podemos hacer que se creen con un tamaño diferente definido en tiempo de ejecución. Así, podríamos definir una lista dinámica con un tamaño máximo definido en el momento de crear el objeto usando la siguiente definición.

Una vez definido, el tamaño del tipo queda fijado y todos los objetos del tipo *T_Lista_Estatica* serán del mismo tamaño.

```
Max : constant Natural := Calcular_Max;
--                          ^ Función,
--          no conocido en tiempo de compilación
type type T_Vector_Enteros is
    array (Positive range <>) of Integer;
type T_Lista_Estática_Adaptable is record
   Elems : T_Vector_Enteros (1 .. Max);
   cont  : Natural;
end record;
-- T_Lista_Estatica  es un tipo definido
```

```
-- pero max no se conoce en tiempo de
compilación
```

9.3. Valores por defecto en registros

Si se quiere asignar valores por defecto a las variables a la hora de crearlas, se hace incluyendo la asignación después de la descripción de cada campo. En los agregados, si se quiere asignar el valor por defecto, en el lugar correspondiente se coloca <>.

```
type T_Lista_Estática_Adaptable is record
  Elems : T_Vector_Enteros (1 .. Max)
    := (others => -1);
  cont  : Natural := 0;
end record;
L: T_Lista_Estatica_Adaptable := (<>,<>);
```

9.4. Registros con discriminantes

Si se quiere tener objetos de distinto tamaño se pueden añadir discriminantes. Los discriminantes, en su forma simple son campos que se añaden al registro a los que se accede con la notación de punto, pero que se utilizan para definir el tamaño del registro. Eso sí, una vez definido un objeto de un tamaño, ya no se puede modificar (el tamaño, no los valores asociados). El tipo es indefinido si no se declara el discriminante con una inicialización.

```
type type T_Vector_Enteros is
    array (Positive range <>) of Integer;
type T_Lista_Estatica (Max : Natural) is record
--        Discriminante^.
--   No se puede modificar tras su
inicialización.
   elem : T_Vector_Enteros (1 .. Max);
   cont : Natural := 0;
end record;
-- T_Lista estatica es un tipo indefinido
-- (como un array)
```

En el ejemplo siguiente, se define el tipo indefinido T_Coordenada: un registro sin campos y con dos discriminantes. De esta manera, nos aseguramos de que cualquier objeto de este tipo tenga ambos valores definidos ya que los registros, podrían tener alguno de sus campos sin inicializar. Eso tiene también consecuencias. Por ejemplo, no se puede definir un array de elementos de tipo T_Coordenada porque su tamaño no está definido.

```
type T_Coordenada (X, Y : Natural) is record
   null;
end record;
P : T_Coordenada;
-- ERROR: T_Coordenada es indefinido,
-- hace falta definir los discriminantes
-- o definir un valor por defecto
P2 : T_Coordenada (1, 2);
P3 : T_Coordenada:= (1, 2);
--  Las dos declaraciones son equivalentes.
```

Pero se podría añadir un valor por defecto a los objetos de ese tipo (P) permitiendo evitar tener que definir los valores de los discriminantes (que haría que se tomen los valores por defecto). En cualquier caso, las definiciones de P1 y P2 siguen siendo válidas habiendo valores por defecto. Esta opción es muy recomendable porque habilita la creación de variables sin tener que definir el discriminante y permite la variación de los campos en tiempo de ejecución.

```
type T_Coordenada2 (X, Y : Natural := 0) is
record
   null;
end record;
P : T_Coordenada;
-- ahora es correcto.
-- No hace falta definir los discriminantes,
-- se toman sus valores por defecto (
P2 : T_Coordenada (1, 2);
P3 : T_Coordenada:= (1, 2);
```

```
-- Las dos declaraciones son equivalentes
-- siguen siendo posibles.
```

A continuación, se presenta un procedimiento Put que recorre los contenidos de la lista estática y los muestra por una pantalla. Al final aparece una declaración de una variable de tipo *T_Lista_Estatica* y una posible llamada a dicho procedimiento.

```
procedure Put (L : in T_Lista_Estatica) is
  I : Integer := 1;
begin
  Put ("<Lista, elems: [");
  loop
    exit when I not in 1..L.cont;
    Put (" " & Integer'Image (L.Elems(I)));
    I:=I+1;
  end loop;
  Put_Line ("]>");
end Put;

L1 : T_Lista_Estatica :=
  (Max => 128,
   Elems => (1, 2, 3, 4, others => <>),
   Cont => 4);
begin
   Print_Lista_Estatica(L1);
end Main;
```

9.5. Registros variantes

Los discriminantes se pueden utilizar para definir distintos conjuntos de campos dependiendo de sus valores. En el ejemplo que aparece a continuación, se guardan los datos físicos de una persona. Se guarda el color de los ojos (campo *ojos*), su altura (campo *altura*), y el color del pelo (campo *color_pelo)* solo cuando la persona no es calva (campo *es_calvo*= false), y, si lo es, entonces se guarda información de si lleva tupé o no dicha persona.

```
type T_Persona (es_calvo: Boolean) is record
  ojos         : T_Color;
  altura : T_centimetros;
  case es_calvo is
    when True =>
      lleva_tupe : Boolean;
    when False =>
      Color_pelo  : T_Color;
  end case;
end record;

Jodie_Foster: T_persona (false) :=
  (ojos=>azul, altura=>160,
   color_pelo=>Amarillo);
Bruce_Willis: T_Persona (true):=
  (ojos=>verde, altura=>182,
   lleva_tupe=>false);
John_Travolta: T_Persona :=
  (ojos=>azul, altura=>188, es_calvo=>true,
   lleva_tupe=>true);
```

El tipo T_Nomina representa la nómina pagada a empleados. Para cada nómina se guarda su identificación, su nombre, y la fecha de pago. Utiliza un discriminante basado en el tipo de sueldo del empleado. Según el tipo de sueldo se guarda diferente información, cuando el sueldo es *Profesional*, se cobra una cantidad fija al mes; cuando es sueldo *por ventas,* el sueldo es una cantidad semanal, que depende de la ventas realizadas según un porcentaje de las ventas que también está relacionado con el empleado; el sueldo de un empleado *temporal* depende de las horas trabajadas y el salario por hora que cobre dicho empleado; finalmente, cuando el tipo de sueldo sea *desconocido* no se guarda ningún otro tipo de información al respecto.

```
subtype T_Dia is Ada.Calendar.Day_Number;
type T_Mes is (Ene, Feb, Mar, Abr, May, Jun,
Jul, Ago, Sep, Oct, Nov, Dic);
subtype T_Anno is Ada.Calendar.Year_Number;
type T_Fecha is record
  dia : T_Dia  := T_Dia'First;
  mes : T_mes  := T_Mes'First;
  anno: T_Anno := T_Anno'First;
end record;

subtype T_Centimos is Natural range 0..99;
type T_Cantidad is record
  Positivo: Boolean   := True;
  Euros : Natural   := 0;
  Centimos : CentsType := 0;
end record;

subtype T_Rango_Nombre is Positive range 1..20;
subtype T_Nombre is String(T_Rango_Nombre);
subtype T_Id is Positive range 1111..9999;
subtype T_Horas_Trabajo is
  Float range 0.0..168.0;
subtype T_Porcentaje_Comision is
  Float range 0.00..0.50;
```

```
type T_Tipo_Sueldo is
  (Desconocido, Profesional, Ventas, Temporal);
type T_Nomina (tipo_sueldo:T_Categoria_sueldo
     := Desconocido) is record
  ID          : T_Id := T_Id'Last;
  Long_nombre: T_Rango_Nombre;
  Nombre      : T_Nombre := (others=>' ');
  Fecha_psgo : T_Fecha;
  case tipo_sueldo is
    when Profesional =>
      Salario_mensual : T_Cantidad;
    when Ventas =>
      Salario_Semanal  : T_Cantidad;
      comision     : Float range 0.00..0.50;
      Ventas : T_Cantidad;
    when Temporal =>
      Salario_hora  : T_Cantidad;
      Horas_trabajadas : T_Porcentaje_Comision;
    when Desconocido =>
      null;
  end case;
```