Memoria Práctica 1

Sistemas en Tiempo Real

Almudena García Jurado-Centurión

Índice

Apartado 1

Enunciado:

Escribir un programa de nombre p1.adb en el que se gestione un array de registros de temperaturas creado dinámicamente. El número de registros del array es un dato que debe introducir el usuario en tiempo de ejecución. Cada registro se compondrá de los campos siguientes:

1. Fecha registro: define la fecha en la que se realizó la medida de temperatura.
2. Temperatura: número en coma fija definido en el rango [-25, 75] con una resolución de 0,01.

El registro Fecha está formado por los campos:

1. Día: número positivo definido en el rango 1..31
2. Mes: valor enumerado que puede tomar los valores (Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre).
3. Año: Valor positivo definido en el rango 1900..2100;

En este programa, después de realizar la entrada de datos que consiste en leer fechas y las temperaturas correspondientes, se deben calcular las temperaturas mínima, máxima y media. La presentación de los resultados puede ser como muestra el ejemplo:

Temperatura mínima: -7 grados, alcanzada el 7 de febrero de 1999
Temperatura máxima: 45.5 grados, alcanzada el 14 de agosto de 1999
La temperatura media del período 3 de enero de 1999 a 15 de septiembre de 1999 ha sido de 16.54 grados.

En éste apartado se nos pide escribir un programa de nombre p1.adb en el que se gestione un array de registros de temperaturas creado dinámicamente.
El número de registros del array es un dato que debe introducir el usuario en tiempo de ejecución.

Para escribir el programa, primero debemos definir una función principal. Dado que no necesitamos devolver ningún dato, usaremos un bloque *procedure*, el cual nos permite definir funciones que no devuelven ningún dato de salida.

El bloque *procedure* se compone de dos secciones principales, delimitadas por la sentencia **begin.**

- Antes del **begin** se definen los tipos y variables a usar en el programa
- Después del **begin** se indica la rutina a ejecutar por el programa

Una vez declarado el *procedure,* con nombre p1, lo primero que tenemos que hacer es definir los tipos de datos correspondientes. Estos se declaran antes del bloque **begin.**

Los tipos necesarios son:

- Un tipo basado en coma fija definido en el rango [-25, 75] con una resolución de 0,01.

  Para ello, nos basaremos en el tipo *delta*, al que estableceremos restricciones mediante la orden *range.* Al definir un tipo basado en *delta*, necesitamos indicar el grado de precisión que queremos. En éste caso, la resolución es de 0.01, así que lo indicamos en la declaración.

  La declaración del tipo, con nombre *fixed_point,* quedará así :

```
type fixed_point is delta 0.01 range -25.00 .. 75.00;
```

  Al ser un tipo basado en coma fija, debemos definir el rango con la misma resolución que hemos indicado anteriormente.

- Un registro de fechas, compuesto de:

  - **Día**: número positivo definido en el rango 1..31
  - **Mes**: valor enumerado que puede tomar los valores (Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre).
  - **Año**: Valor positivo definido en el rango 1900..2100

  Para definir el registro, usaremos el tipo *tagged*, que permite definir estructuras con diferentes variables, similares a los *struct* de C.

  El día y el año quedarán definidos como tipos *Integer* con rango. En el caso del día, el rango será [1-31]; y en el año, [1900-2100].

  Para el mes, usaremos una enumeración, que contendrá los nombres de los meses. Ésta la declararemos previamente como tipo, con nombre *month_enum,* de ésta forma:

```
--Month Enum
type month_enum is (January, February, March, April, May, June, July, August, September, October, November, December);
```

Una vez declarado el tipo enumeración para los meses, declararemos el tipo registro de fechas, con el nombre *date,* de la siguiente forma:

```
--Date record
type date is tagged
   record
      Day: Integer range 1 .. 31;
      Month: month_enum;
      Year: Integer range 1900 .. 2100;
   end record;
```

- Un segundo registro compuesto por la fecha y temperatura.

  Para esto volveremos a usar el tipo *tagged.*

  - La temperatura será una variable de coma fija, la cual usará el tipo *fixed_point* definido previamente.

  - La fecha será una variable registro, basada en el tipo *date* definido previamente.

El registro, con nombre *register,* quedará así:

```
--Register record (minimal type in the vector)
type register is tagged
   record
      temperature: fixed_point;
      day: date;
   end record;
```

Éste tipo será el tipo base del vector.

Una vez definidos los tipos elementales, crearemos un tipo *vector* basado en un array de registros tipo *register,* definidos anteriormente.

Éste tipo no tendrá un tamaño prefijado, siendo éste indicado por el usuario durante la declaración de la estructura.

El tipo quedará de la siguiente forma:

```
--vector of register
type vector is array(integer range <>) of register;
```

Con esto ya tenemos nuestro tipo *vector,* que almacenará las temperaturas junto a la fecha de su medición, y podemos comenzar a escribir el programa.

En primer lugar, debemos pedir al usuario que nos indique el tamaño del vector. Para ello, declararemos una variable de tipo *Integer,* con nombre **size.**

Mostramos un mensaje por pantalla pidiendo el tamaño, usando la función Put_Line( ), y recogeremos un valor desde teclado en la variable **size.**

Para recoger el valor usaremos la función get( ) de *Text.IO*. Como ADA es un lenguaje fuertemente tipado, debemos usar el paquete de *Text_IO* correspondiente a nuestro tipo.

En este caso, nuestro tipo es Integer, así que el paquete será *Integer_IO*.
Declaramos una instancia del paquete llamada *IIO*, y la activamos para su uso.

Esta declaración irá antes del bloque **begin,** junto al resto de tipos definidos anteriormente, y quedará así:

```
package IIO is new Integer_IO(Integer);
use IIO;
```

Hecho esto, pedimos el valor al usuario:

```
--Ask vector size
size := 0;
Put_Line ("Set vector size: ");
IIO.get(size);
```

Éste quedará almacenado en la variable **size**, que usaremos más adelante para asignar el tamaño al vector.

Para poder crear el vector con el tamaño indicado por el usuario, debemos usar una estructura que nos permita definir variables en tiempo de ejecución. Ésta estructura es llamada *declare,* y funciona de forma análoga al propio *procedure.*

Las variables indicadas dentro de este bloque no serán visibles desde fuera del mismo, así que el resto de la rutina del programa tendrá que ser ejecutada dentro de este bloque.

Creamos la estructura *declare,* en la que declararemos las variables y escribiremos el resto de la rutina de nuestro programa.
Antes del bloque **begin** indicamos las variables a usar: en éste caso y por el momento, el vector. El vector tendrá por nombre *reg_list.*

```
declare
   reg_list: vector(1 .. size);
```

Después de esto, continuamos la rutina de nuestro programa. En este paso, pediremos al usuario los datos necesarios para rellenar el vector.
Para ello, debemos declarar dos nuevos paquetes derivados de *Text_IO*: *Enum_IO*, para el tipo *month_enum;* y *Fixed_IO*, para el tipo *fixed_point.*

Las declararemos antes del bloque **begin**, en el *procedure.*

```
package Enum_IO is new Enumeration_IO(month_enum);

package FIO is new Fixed_IO(fixed_point);
```

Hecho esto, continuamos la rutina de nuestro programa.
Creamos un bucle for, que irá pidiendo los datos al usuario y guardándolos en sus posiciones y campos correspondientes dentro del vector.

Para acceder a los campos del registro se usará la sintaxis **registro.campo.** Esta operación se puede repetir multiples veces, en caso de que uno de los campos sea, a su vez, otro registro.

Para ahorrar comparaciones, usamos el atributo ‘range del vector, que nos permite obtener el rango de posiciones del vector.

```
begin

   --Ask values to user
   for i in reg_list'range loop
      Put_Line("Write temperature: ");
      FIO.Get(reg_list(i).temperature);

      Put_Line("Write day: ");
      IIO.Get(reg_list(i).day.day);

      Put_Line("Write month: ");
      Enum_IO.Get(reg_list(i).day.month);

      Put_Line("Write year: ");
      IIO.Get(reg_list(i).day.year);
   end loop;
```

Una vez rellenado el vector, queremos calcular sus temperaturas máximas y mínimas, y la fecha en que se tomaron cada una.

Para ello, creamos dos variables: *max_temperature* y *min_temperature*, de tipo *register*, que guardarán los registros de la máxima y mínima temperatura.

Los añadimos en el *declare,* antes del **begin.**

```
max_temperature: register;
min_temperature: register;
```

Ya de paso, queremos calcular la temperatura media, para lo cual crearemos dos variables más: *sum,* en la que iremos sumando todas las temperaturas almacenadas en el vector; y *average,* que guardará la temperatura media.

```
average: fixed_point;
sum: fixed_point;
```

Para calcular ésto, usaremos un bucle for, que irá recorriendo todas las posiciones del vector y comparando. Para ahorrar tiempo, realizaremos los 3 procesos (maximo, minimo y suma), en el mismo vector

El bucle será así:

```
--Search in vector for get the maximal and minimal temperature
for j in reg_list'range loop

   --Get maximal temperature
   if max_temperature.temperature < reg_list(j).temperature then
      max_temperature := reg_list(j);
   end if;

   --Get minimal temperature
   if min_temperature.temperature > reg_list(j).temperature then
      min_temperature := reg_list(j);
   end if;

   --Sum the values
   sum := sum + reg_list(j).temperature;

end loop;
```

Hecho esto, calculamos la temperatura media dividiendo el valor de *sum* entre el tamaño del vector

```
--Calculate temperatures average
average := sum/size;
```

Ahora tenemos que mostrar esos valores por pantalla, indicando la temperatura máxima y mínima junto a su fecha, y la temperatura media del intervalo.

Para mostrar sus valores, usaremos Put_Line( ), concatenando cada mensaje con el valor correspondiente de su variable, mediante el operador &. Dado que Put_Line( ) espera recibir una cadena de caracteres, obtendremos la cadena equivalente a cada valor usando el atributo ‘img.

El bloque quedará así:

```
   --Print minimal and maximal temperatures with the dates in which were reached
   Put_Line("Minimal Temperature: " & min_temperature.temperature'img & " celsius degree, " & "reached the " &
min_temperature.day.day'img & " of " & min_temperature.day.month'img & " of " & min_temperature.day.year'img);
   Put_Line("Maximal Temperature: " & max_temperature.temperature'img & " celsius degree, " & "reached the " &
max_temperature.day.day'img & " of " & max_temperature.day.month'img & " of " & max_temperature.day.year'img);
   Put_Line("The average temperature of the period is " & average'img);

end;
```

Una posible salida de este programa es:

```
Set vector size:
3
Write temperature:
4.2
Write day:
1
Write month:
January
Write year:
2000
Write temperature:
20
Write day:
14
Write month:
July
Write year:
2015
Write temperature:
-2.4
Write day:
20
Write month:
November
Write year:
1941
Minimal Temperature: -2.40 celsius degree, reached the  20 of NOVEMBER of  1941
Maximal Temperature:  20.00 celsius degree, reached the  14 of JULY of  2015
The average temperature of the period is  7.27
```

Apartado 2

Enunciado: Diseñar un programa que construya una tabla de punteros a tablas de reales cuyo tamaño (filas y columnas) ha de ser introducida desde teclado. Rellenar esta tabla con la suma del índice fila y el índice columna.

En este apartado tenemos que crear un vector de punteros a vectores de reales, y rellenar la tabla con la suma de los índices fila y columna.

Para ello, procedemos de forma similar al apartado anterior.

Creamos un nuevo *procedure,* y comenzamos a definir los tipos.

Empezamos definiendo el vector de reales. En este caso, al igual que en el anterior, el tamaño será definido por el usuario desde teclado.

Para indicar los números reales, usaremos el tipo *float.* Creamos tipo array de tamaño indeterminado, basado en float, con nombre *float_vector.*

```
--define float vector
type float_vector is array(integer range<>) of float;
```

Definido éste, pasamos a crear el tipo puntero. Para crear un puntero usaremos el tipo *access,* indicando el tipo de variable a la que queremos apuntar.
En este caso, creamos un tipo puntero a *float_vector,* con nombre *PVector.*

```
--define pointer to float vector
type PVector is access float_vector;
```

Una vez con el tipo puntero, definimos el tipo array de punteros a vectores de reales, indicándolo como un array de *PVector,* también de tamaño indeterminado, con nombre *vector.*

```
--define vector of pointers to float vector
type vector is array(integer range <>) of PVector;
```

De ésta forma, cada posición de un array tipo *vector* apuntará a un array de tipo *float_vector*, correspondiente a un array de números reales.

Ya con los tipos necesarios creados, comenzamos la rutina de nuestro programa.

La primera parte es similar al apartado anterior, pidiendo el tamaño desde teclado usando el package *Integer_IO*.

```
size := 0;

--Ask vector size to user
Put_Line ("Set vector size: ");
IIO.get(size);
```

Creamos un bloque *declare,* y declaramos las variables correspondientes al puntero a vector de reales, y al vector de punteros, con el tamaño indicado por *size.*

```
declare
   v_pointer: PVector;
   p_table: vector(1 .. size);
```

El puntero a vector de reales se llamará **v_pointer** y el vector de punteros **p_table**

Ahora, con las variables ya creadas, pasamos a rellenar el vector **p_table.** Para ello, usaremos un bucle que, por cada iteración, irá creando un vector *float_vector* asignándolo a *v_pointer,* y ejecutará un nuevo bucle para rellenarlo.

Este segundo bucle asignará a cada posición de *v_pointer* el producto de los actuales valores del índice del bucle principal y del suyo propio.

Una vez inicializados los valores de *v_pointer*, se asignará su contenido a la posición correspondiente de *p_table.*

```
for i in p_table'range loop

   --Creates float vector
   v_pointer := new float_vector(1 .. size);

   --Insert values in vector
   for j in v_pointer'range loop
      v_pointer(j) := float(i*j);
   end loop;

   --Assign vector to pointers vector
   p_table(i) := v_pointer;

end loop;
```

De esta forma, cada posición de *p_table* apuntará a un vector de *float,* creado previamente por *v_pointer,* y con tantas posiciones como indicara el usuario.

Terminadas todas las asignaciones, comprobamos sus valores con otro bucle, que irá recorriendo cada vector y mostrando sus valores con el número de vector y de posición correspondiente al vector.

```
--Shows vector contents
for i in p_table'range loop

   --Get float vector from pointers vector
   v_pointer := p_table(i);

   --Shows values of vector
   for j in v_pointer'range loop
     put_line("vector " & i'img & ", position " & j'img & ":" & v_pointer(j)'img);
   end loop;

end loop;
```

La salida del programa será similar a esta:

```
Set vector size:
4
vector  1, position  1: 1.00000E+00
vector  1, position  2: 2.00000E+00
vector  1, position  3: 3.00000E+00
vector  1, position  4: 4.00000E+00
vector  2, position  1: 2.00000E+00
vector  2, position  2: 4.00000E+00
vector  2, position  3: 6.00000E+00
vector  2, position  4: 8.00000E+00
vector  3, position  1: 3.00000E+00
vector  3, position  2: 6.00000E+00
vector  3, position  3: 9.00000E+00
vector  3, position  4: 1.20000E+01
vector  4, position  1: 4.00000E+00
vector  4, position  2: 8.00000E+00
vector  4, position  3: 1.20000E+01
vector  4, position  4: 1.60000E+01
```

Apartado 3

Enunciado: Completar la tabla siguiente con valores numéricos en función del compilador de Ada con el que se trabaja:

Número entero más pequeño	
Número entero más grande	
Número natural más pequeño	
Número positivo más pequeño	
Número de dígitos significativos de los números reales.	

Para conocer estos datos, usaremos los atributos *'first* y *'last* de cada tipo, y los mostraremos por pantalla con Put_Line( ).
En el caso de los reales, usaremos el atributo *'digits* para saber su resolución.

El código quedaría así:

```
with Text_IO; use Text_IO;

procedure p1 is
begin
   Put_Line("Littlest Integer number: " & Integer'first'img);
   Put_Line("Biggest Integer number: " & Integer'last'img);
   Put_Line("Littlest Natural number: " & Natural'first'img);
   Put_Line("Littlest Positive number: " & Positive'first'img);
   Put_Line("Float significative numbers: " & Float'digits'img);

end p1;
```

Después de ejecutar este programa, el resultado es:

```
Littlest Integer number: -2147483648
Biggest Integer number:  2147483647
Littlest Natural number:  0
Littlest Positive number:  1
Float significative numbers:  6
```

Sobre la tabla anterior:

Número entero más pequeño	-2147483648
Número entero más grande	2147483647
Número natural más pequeño	0
Número positivo más pequeño	1
Número de dígitos significativos de los números reales.	6

Anexo: Códigos fuente

- **Apartado 1:** https://github.com/AlmuHS/Practicas_STR/blob/master/Practica1/p1.adb

```
with Text_IO; use Text_IO;

procedure p1 is

   size: Integer;

   package IIO is new Integer_IO(Integer);
   use IIO;

   type fixed_point is delta 0.01 range -25.00 .. 75.00;
   package FIO is new Fixed_IO(fixed_point);

   --Month Enum
   type month_enum is (January, February, March, April, May, June, July, August, September, October, November, December);
   package Enum_IO is new Enumeration_IO(month_enum);

   --Date record
   type date is tagged
      record
         Day: Integer range 1 .. 31;
         Month: month_enum;
         Year: Integer range 1900 .. 2100;
      end record;

   --Register record (minimal type in the vector)
   type register is tagged
      record
         temperature: fixed_point;
         day: date;
      end record;

   --vector of register
   type vector is array(integer range <>) of register;

begin

   --Ask vector size
   size := 0;
   Put_Line ("Set vector size: ");
   IIO.get(size);

   declare
      reg_list: vector(1 .. size);
      max_temperature: register;
      min_temperature: register;
      average: fixed_point;
      sum: fixed_point;
```

```
   begin

      --Ask values to user
      for i in reg_list'range loop
         Put_Line("Write temperature: ");
         FIO.Get(reg_list(i).temperature);

         Put_Line("Write day: ");
         IIO.Get(reg_list(i).day.day);

         Put_Line("Write month: ");
         Enum_IO.Get(reg_list(i).day.month);

         Put_Line("Write year: ");
         IIO.Get(reg_list(i).day.year);
      end loop;

      --Initialize maximal and minimal temperatures
      max_temperature := reg_list(1);
      min_temperature := reg_list(1);
      sum := 0.0;

      --Search in vector for get the maximal and minimal temperature
      for j in reg_list'range loop

         --Get maximal temperature
         if max_temperature.temperature < reg_list(j).temperature then
            max_temperature := reg_list(j);
         end if;

         --Get minimal temperature
         if min_temperature.temperature > reg_list(j).temperature then
            min_temperature := reg_list(j);
         end if;

         --Sum the values
         sum := sum + reg_list(j).temperature;

      end loop;

      --Calculate temperatures average
      average := sum/size;


      --Print minimal and maximal temperatures with the dates in which were reached
      Put_Line("Minimal Temperature: " & min_temperature.temperature'img & " celsius degree, " & "reached the " &
min_temperature.day.day'img & " of " & min_temperature.day.month'img & " of " & min_temperature.day.year'img);
      Put_Line("Maximal Temperature: " & max_temperature.temperature'img & " celsius degree, " & "reached the " &
max_temperature.day.day'img & " of " & max_temperature.day.month'img & " of " & max_temperature.day.year'img);
      Put_Line("The average temperature of the period is " & average'img);


   end;


end p1;
```

- **Apartado 2:** https://github.com/AlmuHS/Practicas_STR/blob/master/Practica1/p1-b.adb

```
with Text_IO; use Text_IO;

procedure p1 is
   size: Integer;

   package IIO is new Integer_IO(Integer);
   use IIO;

   --define float vector
   type float_vector is array(integer range<>) of float;

   --define pointer to float vector
   type PVector is access float_vector;

   --define vector of pointers to float vector
   type vector is array(integer range <>) of PVector;
begin

   size := 0;

   --Ask vector size to user
   Put_Line ("Set vector size: ");
   IIO.get(size);

   declare
      v_pointer: PVector;
      p_table: vector(1 .. size);
   begin

      for i in p_table'range loop

         --Creates float vector
         v_pointer := new float_vector(1 .. size);

         --Insert values in vector
         for j in v_pointer'range loop
            v_pointer(j) := float(i*j);
         end loop;

         --Assign vector to pointers vector
         p_table(i) := v_pointer;

      end loop;
```

```
      --Shows vector contents
      for i in p_table'range loop

         --Get float vector from pointers vector
         v_pointer := p_table(i);

         --Shows values of vector
         for j in v_pointer'range loop
           put_line("vector " & i'img & ", position " & j'img & ":" & v_pointer(j)'img);
         end loop;

      end loop;


   end;
end p1;
```

- **Apartado 3: https://github.com/AlmuHS/Practicas_STR/blob/master/Practica1/p1-c.adb**

```
with Text_IO; use Text_IO;

procedure p1 is
begin
   Put_Line("Littlest Integer number: " & Integer'first'img);
   Put_Line("Biggest Integer number: " & Integer'last'img);
   Put_Line("Littlest Natural number: " & Natural'first'img);
   Put_Line("Littlest Positive number: " & Positive'first'img);
   Put_Line("Float significative numbers: " & Float'digits'img);

end p1;
```