PRÁCTICA 1 : *Introducción al Lenguaje de Programación Ada*

OBJETIVOS:

1. Familiarizarse con la sintaxis y estructuras de programación básicas de Ada.
2. Manejar el entorno de programación GnatStudio para construir proyectos Ada.
3. Implementar aplicaciones sencillas usando las características básicas de Ada.
4. Generar secuencias aleatorias utilizando paquetes predefinidos de Ada.
5. Primera toma de contacto con tareas concurrentes en Ada.

PERIODO PLANIFICADO PARA SU REALIZACIÓN: 2 semanas

EJERCICIO 1. Mi primer programa en Ada

1.a) Utilizando el entorno GnatStudio, crea un **proyecto** denominado ***practica1*** utilizando el tipo de proyecto "*Basic → Simple Ada Project*". Fíjate que en la carpeta que indiques dónde se creará el proyecto (opción "*Location → Deploy project in*") se creará un fichero con extensión **.gpr** y las carpetas **/src** y **/obj**. El nombre del fichero que contiene el programa principal se denominará "ejercicio1.adb" y su contenido se detalla a continuación. Edita, compila, crea el ejecutable (*build*) y ejecútalo:

```
procedure ejercicio1 is
   s : String = "Comenzamos las prácticas de STR";
begin
   Ada.Text_Io.Put("Hola Mundo!!! ");
   Ada.Text_Io.Put_Line(s);
end ejercicio1;
```

1.b) Corrige los errores de **compilación**.

1.c) Utiliza la **cláusula *use*** con el paquete Ada.Text_Io.

1.d) Piensa en una ventaja y un inconveniente del uso de la cláusula *use*, así como alguna recomendación de uso que se te ocurra.

EJERCICIO 2. Mi primer paquete (*package)* en Ada

2.a) Dentro del fichero ***ejercicio1.adb***, añade el siguiente procedimiento denominado ***otroMensaje*** a continuación del procedimiento ejercicio1:

```
procedure otroMensaje is
begin
   Put_Line("Vamos a iniciarnos en el lenguaje Ada");
end otroMensaje;
```

2.b) ¿Por qué no compila ahora el fichero ejercicio1.adb?

2.c) Mueve el procedimiento anterior a un **paquete** denominado ***pkg_ejercicio2*** que tendrás que crear dentro de tu proyecto (desde la opción de menú *File → New File*). Los dos ficheros asociados al paquete (con extensiones **.ads** y **.adb**) deben guardarse en el directorio /src de vuestro proyecto. Puede ser necesario utilizar la opción File → *Project → Reload Project* (o directamente desde el botón correspondiente de la barra de herramientas) para que se actualice la vista de tu proyecto y se visualicen los nuevos ficheros añadidos.

2.d) Desde el **programa principal** *ejercicio1* invoca al procedimiento *otroMensaje*. Ejecútalo y comprueba su correcto funcionamiento.

EJERCICIO 3. Ámbito de declaraciones de tipos y variables. Paquetes predefinidos de I/O

3.a) Añade en el paquete *pkg_ejercicio2* las siguientes **declaraciones de tipos y variables** :

```
type TdiasSemana is (Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo);
numAlumnos : Integer := 19;

private
   notaMedia : Float := 6.12;
```

3.b) Dentro del mismo proyecto practica1, añade un nuevo fichero denominado ***ejercicio3.adb*** que incluya un procedimiento denominado *ejercicio3* inicialmente con un *body* vacío, es decir, un *body* con una única sentencia "null;"

3.c) Añade desde el menú contextual del proyecto, en la opción *Project->Properties->Main*, este nuevo fichero como otro programa principal de nuestro proyecto. Observa que al tener en un mismo proyecto **varios programas principales**, desde los botones correspondientes nos aparecerá un desplegable (pulsar botón derecho del ratón) para hacer el *build* o el *run* del fichero que queremos que actúe como programa principal.

3.d) Añade en el *body* del procedimiento *ejercicio3*, las sentencias que permitan imprimir por pantalla el contenido de las dos variables declaradas en el paquete. Para ello utiliza los paquetes predefinidos **Ada.Integer_Text_IO** y **Ada.Float_Text_IO** (Anexo A del Manual de Referencia de Ada (ARM): ***Contents*** → A.10.8, A.10.9, que se puede acceder desde la opción *Help->GNAT->Ada 2012 Reference Manual*). Recuerda que también dispones del paquete Ada.Text_IO si necesitas imprimir mensajes de texto por pantalla. ¿Qué has hecho para acceder desde el procedimiento ejercicio3 a la variable privada declarada en el paquete que tenemos en nuestro proyecto, manteniendo su ámbito privado? Imprímela además con un sólo decimal.

NOTA SOBRE LA CONSULTA DEL ARM:

Si al consultar el ARM desde el IDE no se abre Firefox, puedes abrirlo desde el navegador en file:///usr/gnat/share/doc/gnat/html/arm12.html

EJERCICIO 4. Sentencias de selección y bucles. Paquetes genéricos predefinidos

4.a) Añade en tu programa principal *ejercicio1* la lectura por pantalla de un número de tipo Natural que representará un mes (debes usar Ada.Integer_Text_IO ya que Natural es un subtipo de Integer). Después, y utilizando la **sentencia *case***, se debe imprimir por pantalla el nombre de la estación del año correspondiente, teniendo en cuenta que los meses 1, 2 y 12 son invierno; 3, 4 y 5 primavera; 6, 7 y 8 verano; 9, 10 y 11 otoño. Si el número de mes no es correcto, se imprimirá "Mes incorrecto"

4.b) Añade en tu programa principal *ejercicio3* el siguiente **bucle *for*** que recorre los valores del tipo enumerado TdiasSemana y los imprime por pantalla. Si sólo incluyes este código, el programa no compila ¿Por qué?. Para corregir el error, ten en cuenta que para imprimir valores de un tipo enumerado necesitas una instancia del **paquete genérico *Ada.Text_IO.Enumeration_IO*** (Anexo A.10.10 del ARM). ¿Por qué no hay que declarar la variable *dia*?

```
for dia in TdiasSemana loop
   Put(dia);
   Ada.Text_IO.New_Line;
end loop;
```

4.c) A continuación del bucle for anterior, escribe las sentencias que permitan imprimir un mensaje en pantalla solicitando que se introduzca por teclado un día cualquiera, se guarde en una variable y se imprima por pantalla un mensaje indicando si hay o no clases de la asignatura en ese día. Por ejemplo, si se introduce Martes, se imprimirá el mensaje "El Martes no hay clases de STR"

4.d) Comprueba qué ocurre si no se introduce un valor que pertenece a este tipo enumerado.

EJERCICIO 5. Bloques y manejo de excepciones

5.a) Ejecuta el programa *ejercicio1* e introduce un número de mes negativo. ¿Qué ocurre? ¿Por qué? Modifica el código para que en ese caso se muestre por pantalla el mensaje "El número de mes debe ser > 0"

5.b) Añade en el programa *ejercicio1* la sentencia put_line("FIN DEL PROGRAMA"); de forma que sea el último mensaje que aparezca por pantalla independientemente de que el número de mes introducido sea negativo o positivo. Dicha sentencia solo debe aparecer una vez en tu código.

EJERCICIO 6. Generación de números aleatorios

Para generar números aleatorios de tipo discreto, Ada dispone del p**a q u e t e genérico** ***Ada.Numerics.Discrete_Random*** (Anexo A.5.2 del ARM).
En el siguiente programa tienes un ejemplo de instanciación y uso de dicho paquete:

```
with Ada.Numerics.Discrete_Random;  -- paquete genérico predefinido
with Ada.Integer_Text_IO; USE Ada.Integer_Text_IO;
with Ada.Text_IO; use Ada.Text_IO;

procedure ejercicio6 is
   subtype T_Digito is Integer range 0..9;
   -- crear instancia del paquete genérico predefinido
   package Pkg_DigitoAleatorio is new Ada.Numerics.Discrete_Random (T_Digito);
   generador_digito :  pkg_DigitoAleatorio.Generator;  -- declarar generador de valores aleatorios tipo T_Digito
   digito                : T_Digito;
begin
   pkg_DigitoAleatorio.Reset (Generador_Digito); -- Inicializa generador números aleatorios
   loop
      digito := pkg_DigitoAleatorio.Random(generador_digito); -- generar número aleatorio
      Put(digito);
      skip_line;
   end loop;
end ejercicio6;
```

6.a) ¿Qué hace este programa? Inclúyelo en tu proyecto como otro programa principal denominado ***ejercicio6*** y verifica tu respuesta ejecutándolo. Puedes interrumpir su ejecución desde la opción *Run main*, que aparece en la barra de herramientas cuando un programa se está ejecutando

6.b) En la sentencia Put(digito), ¿qué paquete se está utilizando? Justifica tu respuesta.

6.c) Implementa un nuevo programa principal en tu proyecto denominado ***numerosAleatorios*** que genere números reales entre 0.0 y 1.0, utilizando el paquete (**no genérico**) predefinido ***Ada.Numerics.Float_Random*** (Anexo A.5.2 del ARM). Los números de deben mostrar en pantalla con 4 decimales.

Ten en cuenta que tendrás que usar el tipo *Generator*, el procedimiento *Reset* y la función *Random* de forma similar al programa *ejercicio6*, aunque en este caso estaremos usando un paquete no genérico de la librería de Ada, por lo que no tenemos que crear una instancia de un paquete genérico.

EJERCICIO 7. Introducción a tareas concurrentes creadas de forma ESTÁTICA

Para empezar a familiarizarnos con la programación concurrente en Ada, añade a tu proyecto el siguiente programa principal ***main_tareas_estaticas*** y el paquete ***pkg_tareas_estaticas:***

```
with pkg_tareas_estaticas; use pkg_tareas_estaticas;
procedure main_tareas_estaticas is
   tarea1 : T_Tarea(0,9);
   tarea2 : T_Tarea(10,99);
   tarea3 : T_Tarea(100,999);
begin
   null;
end main_tareas_estaticas;
```

```
package pkg_tareas_estaticas is
 task type T_Tarea (a, b : integer);
end pkg_tareas_estaticas;
```

```
with Ada.Numerics.Discrete_Random;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
with Ada.Text_IO; use Ada.Text_IO;

package body pkg_tareas_estaticas is
   task body T_Tarea is
     subtype T_Num is Integer range a..b;
     package pkg_NumAleatorio is new Ada.Numerics.Discrete_Random (T_Num);
     use pkg_NumAleatorio;
     generador_num : Generator;
     num           : T_Num;
   begin
      Reset (generador_num);
      for i in 1..20 loop
        num:= Random(generador_num);
        Put(num);
        New_Line;
        delay(0.01);  -- La tarea queda suspendida 0.01 segundos
      end loop;
   end T_Tarea;

begin
    put_line("Sentencias de inicialización del paquete");
end pkg_ tareas_estaticas;
```

7.a) Observa que el programa principal sólo contiene la sentencia vacía "null;" e intenta comprender qué hace este programa antes de ver el resultado de su ejecución.

7.b) Ejecuta de nuevo el programa pero comentando la sentencia delay, y reflexiona sobre la diferencia en el resultado si dicha sentencia está o no comentada.

7.c) En el body de cualquier paquete existe una parte opcional al final del mismo que comienza con la palabra reservada begin, y en la que se pueden incluir sentencias. ¿Qué tipo de sentencias crees que pueden ser convenientes utilizar? ¿Cuándo se ejecutan estas sentencias? (Apartado 7.2 del ARM)

EJERCICIO 8. Introducción a tareas concurrentes creadas de forma DINÁMICA

Como ejemplo de creación de tareas de forma dinámica, copia el siguiente paquete en el proyecto de esta práctica, cuya especificación es la siguiente:

```
package pkg_tareas_dinamicas is

   MIN_VELOCIDAD_AVION  : CONSTANT := -10;
   MAX_VELOCIDAD_AVION : CONSTANT := 10;
   VELOCIDAD_VUELO     : CONSTANT := 5;
   NUM_INICIAL_AVIONES_AEROVIA : CONSTANT := 3;
   NUM_AEREOVIAS : CONSTANT := 6;

   type T_RangoVelocidad is new integer range MIN_VELOCIDAD_AVION..MAX_VELOCIDAD_AVION;

   -- identificador numérico de cada avión
   type T_IdAvion is mod NUM_INICIAL_AVIONES_AEROVIA;

   -- rango del número de aereovias de distintas altitudes
   type T_Rango_AereoVia is new integer range 1..NUM_AEREOVIAS;

   -- velocidad de los aviones en los ejes X e Y
   type T_Velocidad is
   record
      X : T_RangoVelocidad;
      Y : T_RangoVelocidad;
   end record;

   -- tipo registro para almacenar los datos de un avión
   type T_RecordAvion is
   record
      id       : T_IdAvion;  -- identificador del avion
      velocidad : T_Velocidad;
      aereovia  : T_Rango_AereoVia;
      tren_aterrizaje : Boolean;
   end record;

   type Ptr_T_RecordAvion is access T_RecordAvion;


   TASK TareaGeneraAviones;

   -- Tipo tarea encargada del comportamiento de un avion
   TASK TYPE T_TareaAvion(ptr_avion : Ptr_T_RecordAvion);
   type Ptr_TareaAvion is access T_TareaAvion;

end pkg_tareas_dinamicas;
```

Como puedes comprobar, en este paquete se han declarado constantes, tipos de datos, una tarea estática y un tipo tarea que usaremos para crear instancias dinámicas de dicho tipo.

El body del paquete es el siguiente:

```
with Ada.Text_IO; use Ada.Text_IO;

package body pkg_tareas_dinamicas is

   task body TareaGeneraAviones is
      tarea_avion : Ptr_TareaAvion;
      ptr_avion   : Ptr_T_RecordAvion;

   begin

      for id in T_IdAvion loop
         for aereovia in T_Rango_AereoVia'First..T_Rango_AereoVia'Last - 2 loop

            -- inicializar datos de un nuevo avion
            ptr_avion := new T_RecordAvion;

            ptr_avion.id := id;
            ptr_avion.velocidad.x := VELOCIDAD_VUELO;
            ptr_avion.velocidad.y := 0;
            ptr_avion.aereovia := aereovia;
            ptr_avion.tren_aterrizaje := False;

            -- Crear una tarea para el comportamiento del avion
            tarea_avion := NEW T_TareaAvion(ptr_avion);
         end loop;
      end loop;

   end TareaGeneraAviones;


   TASK BODY T_TareaAvion IS
   begin
      Put_line("TASK Avion: " & T_IdAvion'Image(ptr_avion.id) & " -"
                              & T_Rango_AereoVia'Image(ptr_avion.aereovia));
   end T_TareaAvion;


end pkg_tareas_dinamicas;
```

8.a) Crea un programa principal llamado main_tareas_dinamicas que permita ejecutar las tareas declaradas en este paquete.

8.b) Entiende el código proporcionado en este paquete, ya que te servirá de ayuda para implementar la siguiente práctica.