Ejercicios Bloque 1

Sistemas de Tiempo Real

Grupo 17

Pablo Gómez Rivas

Jesús Fornieles Muñoz

Universidad de Almería

Almería, domingo 24 de marzo de 2024

Índice de Contenidos

Página

Ejercicio 1: Tipos de Datos en ADA

Ejercicio 1.1: Tipos de Datos Enteros

```
1  with Ada.Text_IO; use Ada.Text_IO;
2
3  procedure main1_1 is
4
5  subtype tipoA is Integer range -120 .. 120;
6  subtype tipoB is Integer range 0 .. 50;
7  subtype tipoC is Integer range 0 .. 255;
8
9     A: tipoA;
10    B: tipoB;
11    C: tipoC;
12    sol1: Integer;
13    sol2: Integer;
14    sol3: Integer;
15
16 begin
17   A := 8;
18   B := 16;
19   C := 32;
20
21   sol1 := A+B;
22   sol2 := A+C;
23   sol3 := C+B;
24
25    Put_Line("A + B =" & Integer'Image(sol1));
26    Put_Line("A + C =" & Integer'Image(sol2));
27    Put_Line("B + C =" & Integer'Image(sol3));
28 end main1_1;
```

- **Importar las bibliotecas necesarias:** En la primera línea del código, se importa el paquete Ada.Text_IO, que proporciona procedimientos y funciones para entrada/salida de texto. La cláusula use Ada.Text_IO permite usar los procedimientos y funciones de este paquete sin calificarlos con el prefijo Ada.Text_IO.
- **Definición de subtipos:** Se definen tres subtipos de enteros, tipoA, tipoB y tipoC, con rangos específicos. Estos subtipos limitan los valores que pueden tomar las variables declaradas con estos tipos.
- **Declaración de variables:** Se declaran las variables A, B y C, cada una con su respectivo subtipo (tipoA, tipoB y tipoC, respectivamente), y se declaran tres variables adicionales sol1, sol2 y sol3 de tipo Integer para almacenar los resultados de las operaciones.
- **Asignación de valores a las variables:** Se asignan valores específicos a las variables A, B y C. En este caso, A se asigna como 8, B como 16 y C como 32.
- **Realización de operaciones aritméticas:** Se realizan tres operaciones aritméticas simples: la suma de A y B, la suma de A y C, y la suma de B y C. Los resultados de estas operaciones se almacenan en las variables sol1, sol2 y sol3, respectivamente.
- **Impresión de resultados:** Se utiliza el procedimiento Put_Line para imprimir los resultados de las operaciones en la salida estándar. Cada línea imprime una cadena que representa la operación realizada y el resultado correspondiente, utilizando la función Integer'Image para convertir el resultado entero en una cadena legible para la salida.

Ejercicio 1.2: Tipos de Datos Discretos

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

procedure main1_2 is
   type Semaforo is (Rojo, Amarillo, Verde);
   Sem : Semaforo;
begin
   -- Asignando un valor a Sem
   Sem := Semaforo'Val(0);
   Put_Line("Valor de Sem después de asignar 0: " & Semaforo'Image(Sem));

   Sem := Semaforo'Val(1);
   Put_Line("Valor de Sem después de asignar 1: " & Semaforo'Image(Sem));

   Sem := Semaforo'Val(2);
   Put_Line("Valor de Sem después de asignar 2: " & Semaforo'Image(Sem));
end main1_2;
```

- **Declaración del tipo enumerado:** Se define el tipo enumerado Semaforo que tiene tres posibles valores: Rojo, Amarillo, y Verde.
- **Declaración de la variable:** Se declara una variable Sem del tipo Semaforo.
- **Asignación de valores:** Se asignan valores a la variable Sem utilizando el atributo 'Val' del tipo enumerado Semaforo. Se asignan los valores 0, 1 y 2 respectivamente, que corresponden a los valores Rojo, Amarillo, y Verde según la declaración del tipo enumerado.
- **Impresión de valores:** Se utilizan las funciones Put_Line y 'Image' para imprimir en la consola los valores asignados a Sem. 'Image' convierte el valor de Sem en su representación de cadena correspondiente.

Ejercicio 1.3: Tipos de Datos Reales

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

procedure main1_3 is

   type TipoA is delta 0.01 range -50.00 .. 50.00;
   type TipoB is digits 3 range -200.0 .. 200.0;

   A: TipoA;
   B: TipoB;
   Sol1: Float;

begin
   A := 1.00;
   B := 2.00;

   -- Convertimos A y B a Float antes de realizar la suma
   Sol1 := Float(A) + Float(B);

   Put_Line("El resultado de la suma es: " & Float'Image(Sol1));
end main1_3;
```

- **Define dos tipos de datos:** TipoA y TipoB. TipoA es un tipo decimal con precisión de 2 decimales y rango de -50.00 a 50.00. TipoB es un tipo decimal con precisión de 3 dígitos y rango de -200.0 a 200.0.
- **Declara tres variables:** A de tipo TipoA, B de tipo TipoB y Sol1 de tipo Float.

- **Asigna valores iniciales a las variables A y B:** A se inicializa con 1.00 y B con 2.00.
- **Realiza la suma de A y B** después de convertirlos a tipo Float para evitar pérdida de precisión. Esto se hace mediante la conversión explícita de A y B a Float y luego se realiza la suma.
- **Imprime el resultado de la suma** en la consola utilizando la función 'Put_Line' para mostrar el resultado como una cadena de caracteres, obteniendo la representación en texto de Sol1 mediante la función 'Float'Image'.

Ejercicio 1.4: Arrays

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

procedure Main1_4 is
   type Vector1 is array (1 .. 150) of Float;
   type Matriz1 is array (0 .. 199, 0 .. 199, 0 .. 199) of Float;
   type Vector2 is array (Integer range <>) of Float;

   V1 : Vector1;
   M1 : Matriz1;
   V2 : Vector2(0..10);
begin
   V1 := (others => 0.0);
   M1 := (others => (others => (others => 0.0)));
   V2 := (others => 0.0);

   -- Imprimir elementos de V1
   Put_Line("Elementos de V1:");
   for I in V1'Range loop
      Put(Float'Image(V1(I)) & " ");
   end loop;
   New_Line;

   -- Imprimir elementos de M1
   Put_Line("Elementos de M1:");
   for I in M1'Range(1) loop
      for J in M1'Range(2) loop
         for K in M1'Range(3) loop
            Put(Float'Image(M1(I, J, K)) & " ");
         end loop;
      end loop;
   end loop;
   New_Line;

     -- Imprimir elementos de V2
   Put_Line("Elementos de V2:");
   for I in V2'Range loop
      Put(Float'Image(V2(I)) & " ");
   end loop;
   New_Line;

end Main1_4;
```

- **Define varios tipos de datos:** Vector1 como un arreglo de 150 elementos de tipo Float, Matriz1 como un arreglo tridimensional de tipo Float y Vector2 como un arreglo de enteros de longitud variable de tipo Float.
- **Declara variables de los tipos definidos anteriormente:** V1 como un Vector1, M1 como una Matriz1 y V2 como un Vector2 con un rango de 0 a 10.
- **Inicializa las variables V1, M1 y V2** con valores predeterminados de 0.0.
- **Utiliza bucles para imprimir** los elementos de V1, M1 y V2 en la consola.
- **Imprime los elementos de V1** mediante un bucle for que recorre todos los índices del arreglo V1.

- **Imprime los elementos de M1** mediante bucles anidados for que recorren todos los índices de la matriz M1.
- **Imprime los elementos de V2** mediante un bucle for que recorre todos los índices del arreglo V2.

Ejercicio 1.5: Cadenas

```
1  with Ada.Text_IO; use Ada.Text_IO;
2
3  procedure main1_5 is
4
5     c : String := "TIEMPO REAL";
6
7  begin
8        Put_Line(c);
9  end main1_5;
```

- **Declaración del paquete:** Se incluye el paquete Ada.Text_IO, que proporciona procedimientos y funciones para entrada/salida de texto en Ada.
- **Uso del paquete:** Se usa la cláusula use para utilizar directamente los procedimientos y funciones del paquete Ada.Text_IO sin tener que calificarlos con el prefijo del paquete.
- **Declaración de la variable:** Se declara una variable c de tipo String que contiene el valor "TIEMPO REAL".
- **Impresión en la consola:** Dentro del bloque begin y end, se utiliza el procedimiento Put_Line del paquete Ada.Text_IO para imprimir la cadena de texto contenida en la variable c en una nueva línea en la consola.

Ejercicio 1.6: Registros

```
1  with Ada.Text_IO; use Ada.Text_IO;
2
3  procedure main1_6 is
4
5     subtype tipoDia is Integer range 1..31;
6     subtype tipoMes is Integer range 1..12;
7     subtype tipoAno is Integer range 1900..2024;
8
9     type Fecha is record
10       dia: tipoDia;
11       Mes: tipoMes;
12       Ano: tipoAno;
13    end record;
14
15    type Datos_Personales is record
16       Nombre : String(1..4);
17       Apellidos : String(1..11);
18       date: Fecha;
19    end record;
20
21    -- Declaración de una variable de tipo Persona
22    P1 : Datos_Personales;
23
24 begin
25    -- Inicialización de los campos del registro
26 P1 := (Nombre => "Pepe", Apellidos => "Lopez Lopez", date => (Dia => 1, Mes => 1, Ano => 2001));
27
28
29    -- Acceso a los campos del registro
30    Put_Line("Nombre: " & P1.Nombre);
31    Put_Line("Edad: " & P1.Apellidos);
32   Put_Line("Fecha de nacimiento: " & Integer'Image(P1.Date.Dia) & "/" & Integer'Image(P1.Date.Mes) & "/" & Integer'Image(P1.Date.Ano));
33
34 end main1_6;
```

- **Declaración de subtipos y tipos de registro:** Se definen subtipos para representar días, meses y años, y se define un tipo de registro llamado Fecha que contiene los campos dia, Mes y Ano. Además, se define otro tipo de registro llamado Datos_Personales que contiene campos para el nombre, apellidos y una fecha de tipo Fecha.

- **Declaración de una variable de tipo Datos_Personales:** Se declara una variable llamada P1 de tipo Datos_Personales.
- **Inicialización de la variable P1:** Se inicializa la variable P1 con los valores proporcionados para el nombre, apellidos y fecha de nacimiento.
- **Acceso a los campos del registro:** Se accede a los campos del registro P1 y se imprime el nombre, apellidos y fecha de nacimiento en la consola.

Ejercicio 2: Estructuras de Control en ADA

Ejercicio 2.1: Instrucciones y Estructuras de Control

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Strings.Unbounded; use Ada.Strings.Unbounded;
with Ada.Text_IO.Unbounded_IO; use Ada.Text_IO.Unbounded_IO;

procedure main2_1 is
   Input_String : Unbounded_String;

begin
   Put("Ingrese una cadena: ");
   Get_Line(Input_String);

   for I in 1 .. Length(Input_String) loop
      case Element(Input_String, I) is
         when 'A' | 'B' =>
            Put_Line("Opción 1");
         when 'C' | 'D' | 'E' =>
            Put_Line("Opción 2");
         when 'F' =>
            Put_Line("Opción 3");
         when others =>
            Put_Line("Otra opción");
      end case;
   end loop;
end main2_1;
```

- **Se importan las bibliotecas necesarias:** Ada.Text_IO para entrada/salida de texto, Ada.Strings.Unbounded para trabajar con cadenas de longitud variable y Ada.Text_IO.Unbounded_IO para entrada/salida de cadenas no limitadas.
- Se declara una variable Input_String de tipo Unbounded_String para almacenar la cadena ingresada por el usuario.
- **Se muestra un mensaje** pidiendo al usuario que ingrese una cadena utilizando Put.
- **Se lee la cadena ingresada** por el usuario utilizando Get_Line y se almacena en Input_String.
- **Se inicia un bucle for** que recorre cada carácter de la cadena ingresada utilizando Length(Input_String) como límite.
- Dentro del bucle, se utiliza una declaración case para evaluar cada carácter de la cadena:
    - Si el carácter es 'A' o 'B', se imprime "Opción 1".
    - Si el carácter es 'C', 'D' o 'E', se imprime "Opción 2".
    - Si el carácter es 'F', se imprime "Opción 3".
    - Si el carácter no coincide con ninguno de los anteriores, se imprime "Otra opción".

Ejercicio 3: Uso de Ficheros en ADA

Ejercicio 3.1: Fichero Transpuesto

```
with Ada.Text_IO, Ada.Integer_Text_IO;
use Ada.Text_IO, Ada.Integer_Text_IO;

procedure main3_1 is
   Input_File : File_Type;
   Output_File : File_Type;
   type matriz is array(1..5, 1..10) of Integer;
   type matriz_transpuesta is array(1..10, 1..5) of Integer;
   m : matriz;
   mt : matriz_transpuesta;

begin
-- Carga el archivo de entrada
Open(Input_File, In_File, "input.txt");

-- Lee la matriz entrada
for i in 1..5 loop
    for j in 1..10 loop
        Get(Input_File, m(i,j));
    end loop;
end loop;
Close(Input_File);

-- Calcula la matriz transpuesta
for i in 1..10 loop
    for j in 1..5 loop
        mt(i,j) := m(j,i);
    end loop;
end loop;

-- Abrir el archivo de salida
Open(Output_File, Out_File,"output.txt");

-- Escribir la matríz transpuesta en el archivo salida
for i in 1..10 loop
    for j in 1..5 loop
        Put(Output_File,mt(i,j),Width => 1);
        Put(Output_File," ");
    end loop;
    New_Line(Output_File);
end Loop;

Close(Output_File);
end main3_1;
```

- Abre un archivo de entrada llamado "input.txt".
- Lee los valores del archivo de entrada y los almacena en una matriz bidimensional de tamaño 5x10 llamada "m".

- Cierra el archivo de entrada.
- Calcula la matriz transpuesta de "m" y la almacena en una matriz bidimensional de tamaño 10x5 llamada "mt".
- Abre un archivo de salida llamado "output.txt".
- Escribe la matriz transpuesta "mt" en el archivo de salida.
- Cierra el archivo de salida.

Ejercicio 4: Ocultación de Información en ADA

Ejercicio 4.1: Estructuras Básicas de Datos

```
1  package cola is
2
3      subtype Elemento is Integer range -1000 .. 1000;
4
5      function Vacia return Boolean;
6      procedure Poner(E: Elemento);
7      function Quitar(E: in out Elemento) return Elemento;
8
9  end cola;
```

cola.ads

Se definen el tipo Elemento como entero en el rango de -1000 a 1000, y las funciones Vacia, Poner y Quitar.

```
package body cola is

   type Nodo;
   type Puntero_A_Nodo is access Nodo;

   type Nodo is record
      Dato: Elemento;
      Siguiente: Puntero_A_Nodo;
   end record;

   First, Last : Puntero_A_Nodo := Null;

   function Vacia return boolean is
   begin
      return First = Null;
   end Vacia;

   procedure Poner(E: Elemento) is
      Nuevo_Nodo: Puntero_A_Nodo := new Nodo'(Dato => E, Siguiente => null);
   begin
      if Vacia then
         First := Nuevo_Nodo;
      else
         Last.Siguiente := Nuevo_Nodo;
      end if;
      Last := Nuevo_Nodo;
   end Poner;

   function Quitar(E: in out Elemento) return Elemento is
      Nodo_A_Quitar: Puntero_A_Nodo;
   begin
      if Vacia then
         return 0; -- La cola está vacía
      end if;

      Nodo_A_Quitar := First;
      E := Nodo_A_Quitar.Dato;
      First := Nodo_A_Quitar.Siguiente;

      if First = Null then
         Last := Null;
      end if;

      return E;
   end Quitar;

end cola;
```

cola.adb

- **Se define un tipo de registro llamado "Nodo"** que representa un nodo en la estructura de datos de la cola. Cada nodo contiene un campo "Dato" de tipo "Elemento" y un campo "Siguiente" que es un puntero al siguiente nodo en la cola.
- **Se define un tipo de acceso "Puntero_A_Nodo"** que es un puntero a un nodo.
- **Se declaran dos punteros "First" y "Last"** que apuntan al primer y último nodo de la cola respectivamente. Inicialmente, ambos punteros se establecen en "Null" ya que la cola está vacía.
- **Se define una función "Vacia"** que devuelve "True" si la cola está vacía (es decir, si "First" es "Null").

- **Se define un procedimiento "Poner"** que agrega un nuevo elemento a la cola. Este procedimiento crea un nuevo nodo con el elemento dado y lo agrega al final de la cola actualizando los punteros "Last" y "First" según corresponda.
- **Se define una función "Quitar"** que elimina y devuelve el primer elemento de la cola. Esta función primero verifica si la cola está vacía. Si no lo está, guarda el elemento del primer nodo en una variable temporal, actualiza el puntero "First" para apuntar al siguiente nodo y libera el nodo eliminado de la memoria. Si después de la eliminación la cola queda vacía, también actualiza el puntero "Last".

```
1  with Ada.Text_IO; use Ada.Text_IO;
2  with cola; use cola;
3
4  procedure main4_1 is
5     -- Variables auxiliares
6     Dato: Elemento;
7  begin
8     -- Crear una cola
9     if Vacia then
10       Put_Line("La cola está vacía.");
11    end if;
12
13    -- Insertar datos en la cola
14    Poner(1);
15    Poner(2);
16    Poner(3);
17    Poner(4);
18    Poner(5);
19    Poner(6);
20    Poner(7);
21
22    -- Eliminar y mostrar los datos de la cola uno a uno
23    Put_Line("Datos en la cola:");
24    loop
25       exit when Quitar(Dato) = 0; -- Salir si la cola está vacía
26       Put(Integer'Image(Dato));
27    end loop;
28 end main4_1;
```

main4_1.adb

- Se declara una variable auxiliar llamada "Dato" de tipo "Elemento". Este tipo de dato está definido en el módulo "cola" en el rango de -1000 a 1000.
- Se verifica si la cola está vacía utilizando la función "Vacia" del módulo "cola". Si está vacía, se imprime un mensaje indicando que la cola está vacía.
- Se insertan datos en la cola utilizando la función "Poner" del módulo "cola".
- Se elimina y muestra cada dato de la cola uno por uno en un bucle. Se utiliza la función "Quitar" del módulo "cola" para quitar un dato de la cola. El bucle continúa hasta que la función "Quitar" devuelva 0, lo que indica que la cola está vacía. Se utiliza la función "Put_Line" para imprimir el dato obtenido de la cola. La función "Integer'Image" se utiliza para convertir el dato a su representación en cadena antes de imprimirlo.

Ejercicio 5: Sobrecarga de Operadores en ADA

Ejercicio 5.1: Paquete de Números Complejos

```
-- Definición del paquete para números complejos

package Numeros_Complejos is

   -- Definición del tipo de dato para números complejos
   type Complejo is record
      Real, Imaginario: Float;
   end record;

   -- Funciones para realizar operaciones con números complejos

   function Suma(Num1, Num2: Complejo) return Complejo;
   -- Realiza la suma de dos números complejos

   function Resta(Num1, Num2: Complejo) return Complejo;
   -- Realiza la resta de dos números complejos

   function Producto(Num1, Num2: Complejo) return Complejo;
   -- Realiza el producto de dos números complejos

   function Division(Num1, Num2: Complejo) return Complejo;
   -- Realiza la división de dos números complejos

   function Conjugado(Num: Complejo) return Complejo;
   -- Calcula el conjugado de un número complejo

end Numeros_Complejos;
```

numeros_complejos.ads

Se define el tipo Complejo que contiene un dato Real y otro Imaginario, ambos float. Se declaran las funciones utilizadas.

```
-- Implementación del paquete para números complejos

package body Numeros_Complejos is

   function Suma(Num1, Num2: Complejo) return Complejo is
   begin
      return (Real => Num1.Real + Num2.Real, Imaginario => Num1.Imaginario + Num2.Imaginario);
   end Suma;

   function Resta(Num1, Num2: Complejo) return Complejo is
   begin
      return (Real => Num1.Real - Num2.Real, Imaginario => Num1.Imaginario - Num2.Imaginario);
   end Resta;

   function Producto(Num1, Num2: Complejo) return Complejo is
   begin
      return (Real => Num1.Real * Num2.Real - Num1.Imaginario * Num2.Imaginario,
              Imaginario => Num1.Real * Num2.Imaginario + Num1.Imaginario * Num2.Real);
   end Producto;

   function Division(Num1, Num2: Complejo) return Complejo is
   begin
      return (Real => (Num1.Real * Num2.Real + Num1.Imaginario * Num2.Imaginario) / (Num2.Real ** 2 + Num2.Imaginario ** 2),
              Imaginario => (Num1.Imaginario * Num2.Real - Num1.Real * Num2.Imaginario) / (Num2.Real ** 2 + Num2.Imaginario ** 2));
   end Division;

   function Conjugado(Num: Complejo) return Complejo is
   begin
      return (Real => Num.Real, Imaginario => -Num.Imaginario);
   end Conjugado;

end Numeros_Complejos;
```

numeros_complejos.adb

- **Declaración del paquete:** Comienza declarando el paquete Numeros_Complejos.
- **Implementación de las funciones:**
  - **Suma:** Define una función para sumar dos números complejos.
  - **Resta:** Implementa una función para restar dos números complejos.
  - **Producto:** Define una función para multiplicar dos números complejos.
  - **División:** Implementa una función para dividir dos números complejos.
  - **Conjugado:** Define una función para obtener el conjugado de un número complejo.
- **Implementación de cada función:** Cada función toma dos números complejos como parámetros y realiza la operación correspondiente.
- **Retorno de resultados:** Cada función retorna un nuevo número complejo con el resultado de la operación aplicada.

```
-- Programa principal para probar el paquete Numeros_Complejos

with Ada.Text_IO; use Ada.Text_IO;
with Numeros_Complejos; use Numeros_Complejos;

procedure main5_1 is
   A, B, Resultado: Complejo;
begin
   -- Definir dos números complejos
   A := (Real => 3.0, Imaginario => 2.0);
   B := (Real => 1.0, Imaginario => -1.0);

   -- Realizar operaciones con los números complejos
   Resultado := Suma(A, B);
   Put_Line("Suma: " & Float'Image(Resultado.Real) & " + " & Float'Image(Resultado.Imaginario) & "i");

   Resultado := Resta(A, B);
   Put_Line("Resta: " & Float'Image(Resultado.Real) & " + " & Float'Image(Resultado.Imaginario) & "i");

   Resultado := Producto(A, B);
   Put_Line("Producto: " & Float'Image(Resultado.Real) & " + " & Float'Image(Resultado.Imaginario) & "i");

   Resultado := Division(A, B);
   Put_Line("División: " & Float'Image(Resultado.Real) & " + " & Float'Image(Resultado.Imaginario) & "i");

   Resultado := Conjugado(A);
   Put_Line("Conjugado de A: " & Float'Image(Resultado.Real) & " + " & Float'Image(Resultado.Imaginario) & "i");
end main5_1;
```

main5_1.adb

- **Importación de paquetes:** Se importan los paquetes necesarios, como Ada.Text_IO para operaciones de entrada/salida y el paquete Numeros_Complejos que contiene las operaciones con números complejos.
- **Declaración de variables:** Se declaran las variables A, B y Resultado de tipo Complejo, que representan números complejos.
- **Definición de números complejos:** Se definen dos números complejos, A y B, con valores específicos para la parte real e imaginaria.
- **Operaciones con números complejos:** Se realizan varias operaciones con los números complejos definidos:

  a. Suma de A y B.

  b. Resta de A y B.

  c. Producto de A y B.

  d. División de A y B.

  e. Cálculo del conjugado de A.

- **Impresión de resultados:** Se imprimen los resultados de cada operación realizada, mostrando la parte real e imaginaria de los números complejos obtenidos.