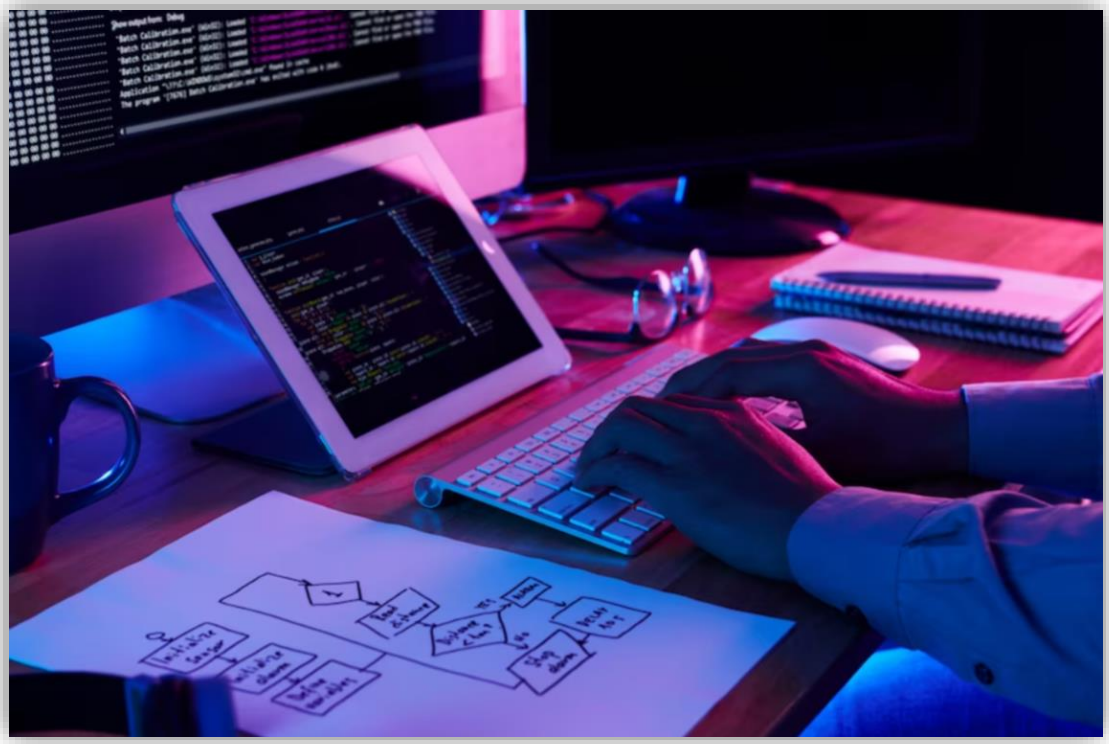
UNIVERSIDAD DE ALMERÍA

PRÁCTICA 1

Relaciones 1-5.



SISTEMAS DE TIEMPO REAL

3º Ingeniería Informática

Lucas Barrientos Muñoz

Contenido

Relación 1.

Ejercicio 1.

ADS

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

package Ejercicio1 is

   subtype Tipo_A is Integer range -120 .. 120;
   subtype Tipo_B is Integer range 0 .. 50;
   subtype Tipo_C is Integer range 0 .. 255;

   A : Tipo_A := 100;
   B : Tipo_B := 25;
   C : Tipo_C := 200;

   procedure Operaciones_Ejercicio1;

end Ejercicio1;
```

ADB

```
package body Ejercicio1 is

   procedure Operaciones_Ejercicio1 is
      Resultado_AB : Integer;
      Resultado_AC : Integer;
      Resultado_BC : Integer;
   begin
      Resultado_AB := A + B;
      Resultado_AC := A + C;
      Resultado_BC := B + C;

      Put_Line("Ejercicio 1:");
      Put_Line("Resultado de A + B: " & Integer'Image(Resultado_AB));
      Put_Line("Resultado de A + C: " & Integer'Image(Resultado_AC));
      Put_Line("Resultado de B + C: " & Integer'Image(Resultado_BC));

      New_Line;

   end Operaciones_Ejercicio1;

end Ejercicio1;
```

Ejercicio 2.

ADS

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

package Ejercicio2 is

   type Semaforo is (Rojo, Amarillo, Verde);

   procedure Mostrar_Semaforo_Ejercicio2(Color : Semaforo);

end Ejercicio2;
```

ADB

```
package body Ejercicio2 is

   procedure Mostrar_Semaforo_Ejercicio2(Color : Semaforo) is
   begin

      Put_Line("Ejercicio 2: ");

      case Color is
         when Rojo =>
            Put_Line("El semaforo esta en rojo.");
         when Amarillo =>
            Put_Line("El semaforo esta en amarillo.");
         when Verde =>
            Put_Line("El semaforo esta en verde.");
      end case;

      New_Line;

   end Mostrar_Semaforo_Ejercicio2;

end Ejercicio2;
```

Ejercicio 3.

ADS

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

package Ejercicio3 is
```

```
   type A is delta 0.01 range -50.0 .. 50.0;
   type B is delta 0.001 range -200.0 .. 200.0;

   A_Variable : A := 25.0;
   B_Variable : B := 150.123;

   procedure Operaciones_Ejercicio3;

end Ejercicio3;
```

ADB

```
package body Ejercicio3 is

   procedure Operaciones_Ejercicio3 is
      Resultado : Float;
   begin

      Put_Line("Ejercicio 3: ");

      Resultado := Float(A_Variable) + Float(B_Variable);
      Put_Line("Resultado de A + B: " & Float'Image(Resultado));

      New_Line;

   end Operaciones_Ejercicio3;

end Ejercicio3;
```

Ejercicio 4.

ADS

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

package Ejercicio4 is

   type A_Array is array (1 .. 150) of Float;
   type B_Array is array (1 .. 200, 1 .. 200, 1 .. 200) of Integer;
   type C_Array is array (Positive range <>) of Float;

   A_Variable : A_Array := (others => 0.0);
   B_Variable : B_Array := (others => (others => (others => 0)));
   C_Variable : C_Array := (0.0, 0.0, 0.0, 0.0, 0.0); -- Ejemplo de
inicialización con tamaño 5
```

```
   procedure Operaciones_Ejercicio4;

end Ejercicio4;
```

ADB

```
package body Ejercicio4 is

   procedure Operaciones_Ejercicio4 is
   begin

      Put_Line("Ejercicio 4: ");

      Put_Line("Variable A:");
      for I in A_Variable'Range loop
         Put(A_Variable(I)'Image & " ");
      end loop;
      New_Line;

      Put_Line("Variable B:");
      for I in B_Variable'Range(1) loop
         for J in B_Variable'Range(2) loop
            for K in B_Variable'Range(3) loop
               Put(B_Variable(I, J, K)'Image & " ");
            end loop;
            New_Line;
         end loop;
         New_Line;
      end loop;

      Put_Line("Variable C:");
      for I in C_Variable'Range loop
         Put(C_Variable(I)'Image & " ");
      end loop;

      New_Line;

   end Operaciones_Ejercicio4;

end Ejercicio4;
```

Ejercicio 5.

ADS

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

package Ejercicio5 is

   Constante_Cadena : constant String := "TIEMPO REAL";

   procedure Operaciones_Ejercicio5;

end Ejercicio5;
```

ADB

```
package body Ejercicio5 is

   procedure Operaciones_Ejercicio5 is
   begin

      Put_Line("Ejercicio 5:");

      Put_Line("La constante de cadena es: " & Constante_Cadena);

      New_Line;

   end Operaciones_Ejercicio5;

end Ejercicio5;
```

Ejercicio 6.

ADS

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

package Ejercicio6 is

   type Fecha_Nacimiento is record
      Dia    : Integer;
      Mes    : Integer;
      Anio   : Integer;
   end record;

   type Datos_Personales is record
      Nombre          : Unbounded_String;
      Apellidos       : Unbounded_String;
```

```
      Fecha_Nac       : Fecha_Nacimiento;
   end record;

   Lucas : constant Datos_Personales := (
      Nombre      => To_Unbounded_String("Lucas"),
      Apellidos   => To_Unbounded_String("Barrientos Muñoz"),
      Fecha_Nac   => (Dia => 27, Mes => 7, Anio => 2003)
                                        );

   Adrian : constant Datos_Personales := (
      Nombre      => To_Unbounded_String("Adrian"),
      Apellidos   => To_Unbounded_String("Antequera Ramirez"),
      Fecha_Nac   => (Dia => 12, Mes => 2, Anio => 2003)
                                        );

   procedure Imprimir_Ejercicio6;

end Ejercicio6;
```

ADB

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

package body Ejercicio6 is

   procedure Imprimir_Ejercicio6 is
   begin

      Put_Line("Ejercicio 6: ");

      Put_Line("Nombre: " & To_String(Lucas.Nombre));
      Put_Line("Apellidos: " & To_String(Lucas.Apellidos));
      Put_Line("Fecha de Nacimiento: " & Lucas.Fecha_Nac.Dia'Image & "/"
&
                                         Lucas.Fecha_Nac.Mes'Image &
"/" &
                                         Lucas.Fecha_Nac.Anio'Image);

      New_Line;

      Put_Line("Nombre: " & To_String(Adrian.Nombre));
      Put_Line("Apellidos: " & To_String(Adrian.Apellidos));
      Put_Line("Fecha de Nacimiento: " & Adrian.Fecha_Nac.Dia'Image & "/"
&
                                         Adrian.Fecha_Nac.Mes'Image &
"/" &
```

```
                                                Adrian.Fecha_Nac.Anio'Image);

   end Imprimir_Ejercicio6;

end Ejercicio6;
```

Main para ejecutar todos los procedimientos.

```
with Ejercicio1; use Ejercicio1;
with Ejercicio2; use Ejercicio2;
with Ejercicio3; use Ejercicio3;
with Ejercicio4; use Ejercicio4;
with Ejercicio5; use Ejercicio5;
with Ejercicio6; use Ejercicio6;

procedure Main is

begin

   Operaciones_Ejercicio1;

   Mostrar_Semaforo_Ejercicio2(Verde); -- Introducir por par�metro el
color del sem�foro

   Operaciones_Ejercicio3;

   Operaciones_Ejercicio4;

   Operaciones_Ejercicio5;

   Imprimir_Ejercicio6;

end Main;
```

Relación 2.

Ejercicio 1.

ADS

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

package Ejercicio1 is
```

```
   procedure Procesar_Cadena;

end Ejercicio1;
```

ADB

```
with Ada.Text_IO; use Ada.Text_IO;
with Ejercicio1;

package body Ejercicio1 is

   procedure Procesar_Cadena is
      Cadena : constant String := "ABCDEFG";
   begin
      for I in Cadena'Range loop
         case Cadena(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;

      New_Line;

   end Procesar_Cadena;

end Ejercicio1;
```

Parte Opcional.

ADS

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

package Ejercicio1_Opcional is

   procedure Procesar_Cadena_Entrada;

end Ejercicio1_Opcional;
```

ADB

```
with Ada.Text_IO; use Ada.Text_IO;
with Ejercicio1_Opcional;

package body Ejercicio1_Opcional is

procedure Procesar_Cadena_Entrada is
   Cadena : String(1 .. 100); -- Permitimos cadenas de hasta 100
caracteres
   Last   : Natural;
   begin

      Put_Line("Parte Opcional: ");
      New_Line;

   Put_Line("Ingrese una cadena:");
   Get_Line(Cadena, Last);

      for I in Cadena'Range loop
         case Cadena(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 Procesar_Cadena_Entrada;

end Ejercicio1_Opcional;
```

Main para ejecutar todos los procedimientos.

```
with Ejercicio1; use Ejercicio1;
with Ejercicio1_Opcional; use Ejercicio1_Opcional;

procedure Main is

begin

   Procesar_Cadena;
```

```
   Procesar_Cadena_Entrada;

end Main;
```

Relación 3.

Leer y escribir matriz.

ADS

```
package Ejercicio1 is
   type Matrix is array(Integer range <>, Integer range <>) of Integer;

   procedure Leer_Matriz(FileName : String; M2 : out Matrix);
   procedure Escribir_Matriz(FileName : String; M2 : Matrix);
end Ejercicio1;
```

ADB

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

package body Ejercicio1 is



   procedure Leer_Matriz(FileName : String; M2 : out Matrix) is
      F : File_Type;
      Rows, Cols : Integer;
   begin
      Open(F, In_File, FileName);
      Get(F, Rows);
      Get(F, Cols);
      New_Line(F);

      for I in 1 .. Rows loop
         for J in 1 .. Cols loop
            Get(F, M2(I, J));
         end loop;
         New_Line(F);
      end loop;

      Close(F);
```

```
   end Leer_Matriz;

   procedure Escribir_Matriz(FileName : String; M2 : Matrix) is
      F : File_Type;
      Rows, Cols : Integer;
   begin
      Open(F, Out_File, FileName);
      Rows := M2'Length(1);
      Cols := M2'Length(2);

      Put(F, Rows);
      Put(F, ' ');
      Put(F, Cols);
      New_Line(F);

      for I in 1 .. Rows loop
         for J in 1 .. Cols loop
            Put(F, M2(I, J), Width => 3);
            Put(F, ' ');
         end loop;
         New_Line(F);
      end loop;

      Close(F);
   end Escribir_Matriz;

end Ejercicio1;
```

Transponer matriz.

ADS

```
package Matriz_Transpuesta is
   type Matrix is array(Integer range <>, Integer range <>) of Integer;

   procedure Transponer(M : in out Matrix);
end Matriz_Transpuesta;
```

ADB

```
package body Matriz_Transpuesta is

   procedure Transponer(M : in out Matrix) is
      Temp : Integer;
   begin
```

```
      for I in M'Range(1) loop
         for J in M'Range(2) loop
            if J > I then
               Temp := M(I, J);
               M(I, J) := M(J, I);
               M(J, I) := Temp;
            end if;
         end loop;
      end loop;
   end Transponer;

end Matriz_Transpuesta;
```

Main para ejecutar todos los procedimientos.

```
with Ada.Text_IO; use Ada.Text_IO;
with Ejercicio1;
with Matriz_Transpuesta;

procedure Main is

   M2 : Ejercicio1.Matrix(1 .. 5, 1 .. 10);
   M : Matriz_Transpuesta.Matrix(1 .. 5, 1 .. 10);

   output : File_Type;

begin

   Create(output, Out_File, "output.txt");

   Ejercicio1.Leer_Matriz("input.txt", M2);
   Matriz_Transpuesta.Transponer(M);
   Ejercicio1.Escribir_Matriz("output.txt", M2);

   Close(output);

end Main;
```

Relación 4.

Ejercicio 1.

ADS

```
package Cola is

   type Elemento is range -100 .. 100;
   function Vacia return Boolean;
   procedure Poner(E: Elemento);
   procedure Quitar(E: out Elemento);

end Cola;
```

ADB

```
package body Cola is
   type Nodo;
   type Enlace is access Nodo;
   type Nodo is
      record
         Contenido : Elemento;
         Siguiente : Enlace;
      end record;
   Primero, Ultimo : Enlace := null;

   function Vacia return Boolean is
   begin
      return Primero = null;
   end Vacia;

   procedure Poner(E: Elemento) is
      Nuevo : Enlace;
   begin
      Nuevo := new Nodo;
      Nuevo.Contenido := E;
      Nuevo.Siguiente := null;
      if Vacia then
         Primero := Nuevo;
      else
         Ultimo.Siguiente := Nuevo;
      end if;
      Ultimo := Nuevo;
   end Poner;

   procedure Quitar(E: out Elemento) is
      Viejo : Enlace;
   begin
      Viejo := Primero;
      E := Viejo.Contenido;
      Primero := Viejo.Siguiente;
      if Primero = null then
```

```
         Ultimo := null;
      end if;
   end Quitar;

end Cola;
```

Main para ejecutar todos los procedimientos.

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

procedure Main is

   E : Elemento;
   N : Integer;

begin

   for i in 1 .. 10 loop
      E := Elemento(i);
      Poner(E);
   end loop;

   Put_Line("Datos de la cola: ");

   while Vacia = false loop
      Quitar(Elemento(N));
      Put(Integer'Image(N));
   end loop;


end Main;
```

Relación 5.

Ejercicio 1.

ADS

```
package Numeros_Complejos is

   type Complejo is private;

   function "+"(X, Y : Complejo) return Complejo;
   function "-"(X, Y : Complejo) return Complejo;
   function "*"(X, Y : Complejo) return Complejo;
```

```
   function "/"(X, Y : Complejo) return Complejo;
   function Conj(X : Complejo) return Complejo;
   function P_Real(X : Complejo) return Complejo;
   function P_Imag(X : Complejo) return Complejo;
   function Comp(R, I : Float) return Complejo;
   function numero_float(X : Complejo) return String;

private type Complejo is
      record
         P_Real : Float;
         P_Imag : Float;
      end record;

end Numeros_Complejos;
```

ADB

```
package body Numeros_Complejos is

   function "+" (X, Y : Complejo) return Complejo is
   begin
      return Complejo'(X.P_Real + Y.P_Real, X.P_Imag + Y.P_Imag);
   end "+";

   function "-" (X, Y : Complejo) return Complejo is
   begin
      return Complejo'(X.P_Real - Y.P_Real, X.P_Imag - Y.P_Imag);

   end "-";

   function "*" (X, Y : Complejo) return Complejo is
      P_Real : Float := X.P_Real * Y.P_Real - X.P_Imag * Y.P_Imag;
      P_Imag : Float := X.P_Real * Y.P_Real + X.P_Imag * Y.P_Imag;
   begin
       return Complejo'(P_Real => P_Real, P_Imag => P_Imag);
   end "*";

   function "/" (X, Y : Complejo) return Complejo is
      P_Real : Float := (X.P_Real * X.P_Real + X.P_Imag * Y.P_Imag) /
(Y.P_Real ** 2 + Y.P_Imag ** 2);
      P_Imag : Float := (X.P_Imag * Y.P_Real + X.P_Real * Y.P_Imag) /
(Y.P_Real ** 2 + Y.P_Imag ** 2);
   begin
      return Complejo'(P_Real => P_Real, P_Imag => P_Imag);
   end "/";

   function Conj (X : Complejo) return Complejo is
```

```
   begin
      return Complejo'(P_Real => X.P_Real, P_Imag => -X.P_Imag);
   end Conj;

   function P_Real (X : Complejo) return Complejo is
   begin
      return Comp(X.P_Real, 0.0);
   end P_Real;

   function P_Imag (X : Complejo) return Complejo is
   begin
      return Comp(X.P_Imag, 0.0);
   end P_Imag;
   function Comp (R, I : Float) return Complejo is
      C : Complejo;
   begin
      C.P_Real := R;
      C.P_Imag := I;
      return C;
   end Comp;

   function numero_float (X : Complejo) return String is
   begin
      return Float'Image(X.P_Real) & " + " & Float'Image(X.P_Imag) & "i";
   end numero_float;

end Numeros_Complejos;
```

Main para ejecutar todos los procedimientos.

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

procedure Main is

   x, y, Suma, Resta, Multiplicacion, Division, Conjugado : Complejo;

begin

   x := Comp(5.0, 3.0);
   y := Comp(4.0, 9.0);
   Suma := x + y;
   Resta := x - y;
   Multiplicacion := x * y;
   Division := x / y;
   Conjugado := Conj(x);

   Put_Line("Suma = " & numero_float(Suma));
```

```
   Put_Line("Resta = " & numero_float(Resta));
   Put_Line("MultiplicaciÓn = " & numero_float(Multiplicacion));
   Put_Line("DivisiÓn = " & numero_float(Division));
   Put_Line("Conjugado = " & numero_float(Conjugado));


end Main;
```