Introducción al Lenguaje de Programación Ada

Mario Aldea Rivas

Departamento de Electrónica y Computadores
Universidad de Cantabria

1. El lenguaje Ada

UNIVERSIDAD DE CANTABRIA

Ada (en honor de Lady Augusta Ada Byron)

- **Lenguaje inspirado en Pascal**
- **Creado en 1983 (Ada 83) por el DoD de los EEUU**
  - **Lenguaje de propósito general**
  - **pero orientado a sistemas empotrados de tiempo real**
  - **detecta muchos errores en tiempo de compilación: facilita la creación de programas fiables**

Revisión en 1995 (Ada 95):

- **versión mejorada, incluye programación orientada al objeto**

Nueva versión a punto de salir (Ada 06)

- **mejoras en todos los aspectos (también en tiempo real)**

Comparación con Pascal: Estructura de un programa

UNIVERSIDAD DE CANTABRIA

Ada:

```
procedure Mi_Programa is

 Max_Empleados : constant := 50;

 type T_Empleados is
    range 1 .. Max_Empleados;

 Num_Empleados :
    T_Empleados := 10;

 procedure Un_Proc is
 begin
    ...; -- instrucciones
 end Un_Proc;

begin
   ...; -- instrucciones
end Mi_Programa;
```

Pascal:

```
program Mi_Programa;

const
  Max_Empleados = 50;

type
  T_Empleados=1..Max_Empleados;

var
  Num_Empleados : T_Empleados;

  procedure Un_Proc;
  begin
    ...; (* instrucciones *)
  end;

begin
  ...; { instrucciones }
end.
```


Comparación con Pascal: Instrucciones de control

UNIVERSIDAD DE CANTABRIA

Ada:

```
if I < 0 then
   Negativo := True;
   I := Abs (I);
else
   Negativo := False;
end if;


for I in 1..100 loop
   Suma := Suma + I;
end loop;


loop
   Anadir_Gota;
   Medir_Nivel (En_Nivel_Mx);
   exit when En_Nivel_Mx;
end loop;
```

Pascal:

```
if I < 0 then
begin
  Negativo := True;
  I := Abs (I);
end
else
  Negativo := False;

for I:=1 to 100 do
  Suma := Suma + I;


repeat
   Anadir_Gota;
   Medir_Nivel (En_Nivel_Mx);
until En_Nivel_Mx;
```

Comparación con Pascal: Procedimientos y funciones

UNIVERSIDAD DE CANTABRIA

Ada:

```
procedure Suma
  (A, B : in  Integer;
   S    : out Integer) is
begin
   S := A + B;
end Suma;
```

```
function Area (R : Float)
               return Float is
  PI : constant := 3.141592;
begin
  return PI * R * R;
end Area;
```

Pascal:

```
procedure Suma
  (A, B : Integer;
   S    : var Integer);
begin
   S := A + B;
end;
```

```
function Area (R : Real): Real;
const
  PI : 3.141592;
begin
  Area := PI * R * R;
end;
```


Comparación con Pascal: Entrada/Salida

UNIVERSIDAD DE CANTABRIA

Ada:

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

procedure Mi_Programa is
   N : Integer;
begin
   Put ("Introduce número...");
   Get (N); Skip_Line;
   Put ("Has escrito el ");
   Put (N);
   New_Line;
   --  o también:
   Put_Line ("Has escrito el " &
             Integer'Image(N));
end Mi_Programa;
```

Pascal:

```
program Mi_Programa;
var
  N : Integer;
begin
  Write ('Introduce número...');
  Readln (N);
  Writeln ('Has escrito el ',
           N);
end.
```

Tipos numéricos

UNIVERSIDAD DE CANTABRIA

Definición de tipos:

```
type Nota_Examen is range 0 .. 10;
type Nota_Con_2_Decimales is digits 2 range 0.0 .. 10.0;
```

Declaración de variables y constantes

```
I : Integer;
Precio : Float;
Nota : Nota_Examen;
Nota_Inicial : Nota_Examen := 5;
Nota_De_Corte : constant Nota_Con_2_Decimales := 5.0;
```

Ejemplos de uso:

```
I := I + 1;
Precio := 10.0 * Float (I);
```


Arrays

UNIVERSIDAD DE CANTABRIA

Definición de tipos:

```
type Notas_Alumnos is array (1 .. N) of Nota_Examen;
```

Declaración de variables y constantes:

```
Diez_Numeros : array (1 .. 10) of Integer;
Notas : Notas_Alumnos := (others => 5); -- valor inicial
Vector : constant array (1 .. 10) of Float :=
     (3 => 4.0, others => 3.0);
```

Ejemplos de uso:

```
for I in Notas'Range loop
  Notas (I) := Notas (I) + 1;
end loop;

Diez_Numeros := (others => 1);
```

Strings

UNIVERSIDAD DE CANTABRIA

Definición de tipos:

```
type Nombre_Cliente is new String (1 .. 15);
```

Declaración de variables y constantes:

hay que completar hasta 15 caracteres

```
Nombre : Nombre_Cliente := "Pepe García    ";
Direccion : String (1 .. 30);
Saludo : constant String := "Hola!";
```

Ejemplos de uso:

```
Nombre (4) := 'a'; -- Ahora Nombre es Pepa García

Nombre (1 .. 4) := "Jose";  -- Ahora Nombre es
                            -- Jose García
N := Nombre'Length; -- N vale 15

Direccion := "General Dávila"; -- Error: no tiene
                               -- 30 caracteres!!
```


Registros

UNIVERSIDAD DE CANTABRIA

Definición de tipos:

```
type Ficha_Cliente is record
    Nombre : Nombre_Cliente;
    Edad   : Integer;
end record;
```

Declaración de variables y constantes:

```
Cliente1 : Ficha_Cliente;
Cliente_Fijo : constant Ficha_Cliente :=
  (Nombre => "José García    ", Edad => 20);
Cliente_Vacio : Ficha_Cliente :=
  (Nombre => (others => ' '), Edad => 0);
```

Ejemplos de uso:

```
Cliente_Vacio.Nombre (1 .. 7) := "Antonio";
Cliente1 := ("Pepa García    ", 20);
```

Instrucción condicional lógica (if)

UNIVERSIDAD DE CANTABRIA

Forma simple:

```
if I < 0 and not Neg then
  I := 0;
end if;
```

Con else:

```
if Valor in 50 .. 70 then
  Valor correcto := True;
  Put_Line ("OK");
else
  Valor_Correcto := False;
end if;
```

Forma múltiple:

```
if Nivel > 50.0 then
  Activa_Alarma (Grave);
elsif Nivel > 40.0 then
  Activa_Alarma (Media);
elsif Nivel > 30.0 then
  Activa_Alarma (Leve);
else
  Funcionamiento_Normal;
end if;
```


Instrucción condicional discreta (case)

UNIVERSIDAD DE CANTABRIA

Cubre todos los casos:

```
case Hoy is
  when Lunes | Martes |
       Jueves =>
    Trabajo;
  when Miercoles =>
    Trabajo;
    Futbol;
  when Viernes =>
    Trabajo;
    Salir;
  when Sabado =>
    Salir;
  when Domingo =>
    null;
end case;
```

No cubre todos los casos:

```
Nota : Integer;

case Nota is
  when 0 .. 4 =>
    Suspenso;
  when 5 .. 6 =>
    Aprobado;
  when 7 | 8 =>
    Notable;
  when 9 .. 10 =>
    Sobresaliente;
  when others =>
    Nota_Incorrecta;
end case;
```

como no se cubren todos los valores es obligatorio poner "others"

Lazos

UNIVERSIDAD DE CANTABRIA

Lazo simple:

```
for I in 1..100 loop
  Suma := Suma + I;
end loop;
```

Lazos y arrays:

```
for I in Vector'range loop
  Suma := Suma + Vector (I);
end loop;
```

Orden inverso:

```
for J in reverse 1..10 loop
  ...;
end loop;
```

Con condición de salida:

```
loop
  Suma := Suma + N;
  exit when Suma > 100;
  N := N + 1;
end loop;
```

Lazo while:

```
while Suma <= 100 loop
   J := J + 2;
   Suma := Suma + J;
end loop;
```


Procedimientos y funciones

UNIVERSIDAD DE CANTABRIA

Parámetros "in" y "out":

si no se pone nada es "in"

```
procedure Resta (Minuendo    : Float;
                 Sustrayendo : in Float;
                 Resultado   : out Float) is
begin
  Resultado := Minuendo - Sustrayendo;
end Resta;
```

Llamada a un procedimiento:

mejor así, para evitar confusiones

```
Resta (14, 12, R);
Resta (Minuendo => 14, Sustrayendo => 12,
       Resultado => R);
Resta (Sustrayendo => 12, Minuendo => 14,
       Resultado => R);
```

si se cambia el orden no pasa nada

Procedimientos y funciones (cont.)

UNIVERSIDAD DE CANTABRIA

Parámetros "out":

```
procedure Incrementa (Valor      : in out Integer;
                      Incremento : in     Integer) is
begin
  Valor := Valor + Incremento;
end Incrementa;
```

Funciones:

las funciones sólo pueden tener parámetros "in"

```
function Suma (A, B : in Integer) return Integer is
begin
  return A + B;
end Suma;
```

Llamada a una función:

```
S := Suma (4, C);
```


Práctica de programación en Ada: Juego del ahorcado

UNIVERSIDAD DE CANTABRIA

Completar el programa que se muestra a continuación:

```
with MaRTE_OS;
with Text_IO; use Text_IO;

procedure Ahorcado is

  subtype Palabra is String (1 .. 20);

  Objetivo : Palabra; -- palabra a acertar

  Acertadas : Palabra := (others => '_'); -- palabra
         -- con las letras acertadas en su sitio y con
         -- '_' en las letras que faltan por acertar

  Long_Palabra : Integer; -- Longitud de la palabra
                          -- objetivo
```

Práctica de programación en Ada: Juego del ahorcado (cont.)

UNIVERSIDAD DE CANTABRIA

```
procedure Busca_Aciertos (Objetivo : Palabra;
                          Long_Palabra : Integer;
                          Letra : Character;
                          Acertadas : in out Palabra;
                          Acierto : out Boolean) is
 begin
    --  Comprueba si 'Letra' está en 'Objetivo', en caso
    --  afirmativo retorna True en 'Acierto'. Además
    --  marca en 'Acertadas' las letras que se han
    --  acertado
    ...;
 end Busca_Aciertos;
```


Práctica de programación en Ada: Juego del ahorcado (cont.)

UNIVERSIDAD DE CANTABRIA

```
function Todas_Acertadas (Acertadas : Palabra;
                          Long_Palabra : Integer)
                          return Boolean is
begin
  --  Retorna true cuando todas las letras han sido
  --  acertadas. Esto puede saberse porque no hay
  --  ningún '_' en los 'Long_Palabra' primeros
  --  caracteres de 'Acertadas'
  ...;
end Todas_Acertadas;
```

Práctica de programación en Ada: Juego del ahorcado (cont.)

UNIVERSIDAD DE CANTABRIA

```
begin
  --  Pide palabra objetivo (con su longitud)
  Get_Line (Objetivo, Long_Palabra);

  --  Bucle de intentos
  loop
    -- Muestra palabra 'Acertadas'
    -- Pide letra
    Busca_Aciertos (Objetivo, Long_Palabra, Letra,
                    Acertadas, Acierto);
    --  Indica si ha acertado o fallado (incrementa el
    --  contador de fallos cuando haya fallado)
    exit when Todas_Acertadas (Acertadas, Long_Palabra)
              or alcanzado el límite de fallos;
  end loop;
  --  Muestra palabra objetivo e indica si ganó o perdió
end Ahorcado;
```