Práctica 0

Ciclo de desarrollo de un programa Ada

0.1 Objetivos

Con esta práctica dirigida se pretende que el alumno:

- conozca el entorno de desarrollo de aplicaciones con lenguaje Ada que hay instalado en el Laboratorio de Programación de Sistemas en Tiempo Real,
- practique con este entorno y aprenda el ciclo de implementación de un programa.

0.2 Desarrollo teórico

Para realizar las prácticas dispone en el laboratorio del compilador de Ada GNAT para LINUX que posee las siguientes características:

- GNAT es un compilador de Ada 95 integrado en el sistema de desarrollo GCC.
- GNAT está disponible para una amplia variedad de plataformas hardware y sistemas operativos y puede ser usado como compilador cruzado (*cross-compiler*) desde cualquier máquina origen hacia cualquier máquina destino.
- GNAT dispone de una interfaz excelente para realizar desarrollo multilenguaje: Ada 95, C, C++, Fortran, etc.
- GNAT (fuentes y ejecutables) se distribuyen gratuitamente a través de Internet.
- GCC es un sistema de desarrollo altamente transportable con múltiples frentes (*front-ends*): C, C++, Objective-C, Fortran77, Pascal, Chill y Ada 95.
- GCC produce código para muchos tipos de procesadores:
  - CISC: Intel x86, Motorola 68k.
  - RISC: PowerPC, DEC Alpha, HP-PA RISC, RS600, MIPS.
- GCC (fuentes y ejecutables) también se distribuyen gratuitamente a través de Internet.

Como primer ejemplo para familiarizarnos con el entorno podemos editar el programa siguiente:[1]

```
$ gedit numeros.adb
```

[1] El alumno no debe preocuparse si no entiente el texto del programa en su totalidad ya que algunas partes del mismo serán explicadas con posterioridad y el objetivo de esta práctica es otro.

Programa 1: Primer programa de ejemplo (`numeros.adb`)

```
1  -- Este programa pretende ilustrar la forma de manejo de las herramientas
2  -- del entorno de desarrollo en Ada.
3
4  -- Paquete estándar que vamos a manejar.
5  with Ada.Text_Io; use Ada.Text_Io;
6
7  -- Procedimiento principal.
8  procedure numeros is
9
10    -- Creación de ejemplares, para la entrada/salida de números, a partir
11    -- de los paquete genéricos "Ada.Text_Io.Integer_Io" y
12    -- "Ada.Text_Io.Float_Io".
13    package Ent_Es is new Ada.Text_Io.Integer_Io(Integer);
14    package Real_Es is new Ada.Text_Io.Float_Io(Float);
15
16
17    -- Declaración de variables locales.
18    I, J: Integer;
19    X, Y: Float;
20
21 -- Cuerpo del procedimiento "Numeros".
22 begin
23    -- Entrada/salida de números enteros.
24    -- Lectura de datos
25    New_Line;
26    Put ("Introduce un número entero: ");
27    Ent_Es.Get (I);
28    Put ("Introduce otro número entero: ");
29    Ent_Es.Get (J);
30    New_Line;
31    -- Presentación de los resultados en el sistema decimal
32    Ent_Es.Put (I); Put ('+');
33    Ent_Es.Put (J); Put ('=');
34    Ent_Es.Put (I+J);
35    New_Line;
36
37    -- Presentación de los resultados en binario
38    Ent_Es.Put (I, Base => 2);
39    Put ('+');
40    Ent_Es.Put (J, Base => 2);
41    Put ('=');
42    Ent_Es.Put (I+J, Base => 2);
43    New_Line;
44
45    -- Presentación de los resultados en hexadecimal
46    Ent_Es.Put (I, Base => 16);
47    Put ('+');
48    Ent_Es.Put (J, Base => 16);
49    Put ('=');
50    Ent_Es.Put (I+J, Base => 16);
51    New_Line;
52
```

```
53    -- Presentación de los resultados con formato
54    Ent_Es.Put (I);
55    Put ('+');
56    Ent_Es.Put (J, Width => 0);
57    Put ('=');
58    Ent_Es.Put (I+J, Width => 6);
59    New_Line; New_Line;
60
61    -- Entrada/salida de números reales.
62    -- Lectura de los datos
63    Put ("Introduce un número real: ");
64    Real_Es.Get (X);
65    Put ("Introduce otro número real: ");
66    Real_Es.Get (Y);
67    New_Line;
68    -- Presentación de los resultados
69    Real_Es.Put (X); Put ('/'); Real_Es.Put (Y); Put ('=');
70    Real_Es.Put (X/Y);
71    New_Line;
72    -- Presentación de los resultados con formato
73    Real_Es.Put (X, Fore => 2, Aft => 2, Exp => 3);
74    Put ('/');
75    Real_Es.Put (Y, Fore => 0, Aft => 0, Exp => 0);
76    Put ('=');
77    Real_Es.Put (X/Y, Fore => 10, Aft => 4, Exp => 0);
78    New_Line; New_Line;
79 end numeros;
```

- Una forma automática de compilar, encuadernar y enlazar es con la orden `gnatmake`. Esta orden es especialmente útil a la hora de compilar aplicaciones multimodulares ya que la compilación y recompilación se realiza en función de las necesidades según los ficheros que se hayan modificado y las dependencias que hay entre ellos:

  `$ gnatmake numeros.adb`

  o

  `$ gnatmake numeros` (`numeros` es el nombre del procedimiento de entrada a la aplicación, procedimiento principal)

- Para ejecutar el programa basta con escribir:

  `$ ./numeros`

En la figura 1 podemos ver las órdenes que se ejecutan de forma automática al llamar a `gnatmake`.

0.3 Tareas a realizar

En esta práctica se tienen que realizar las tareas siguientes:

1. Editar un fichero de nombre `numeros.adb` con un texto como el indicado en el desarrollo teórico anterior.
2. Compilar, encuadernar y enlazar `numeros.adb`.

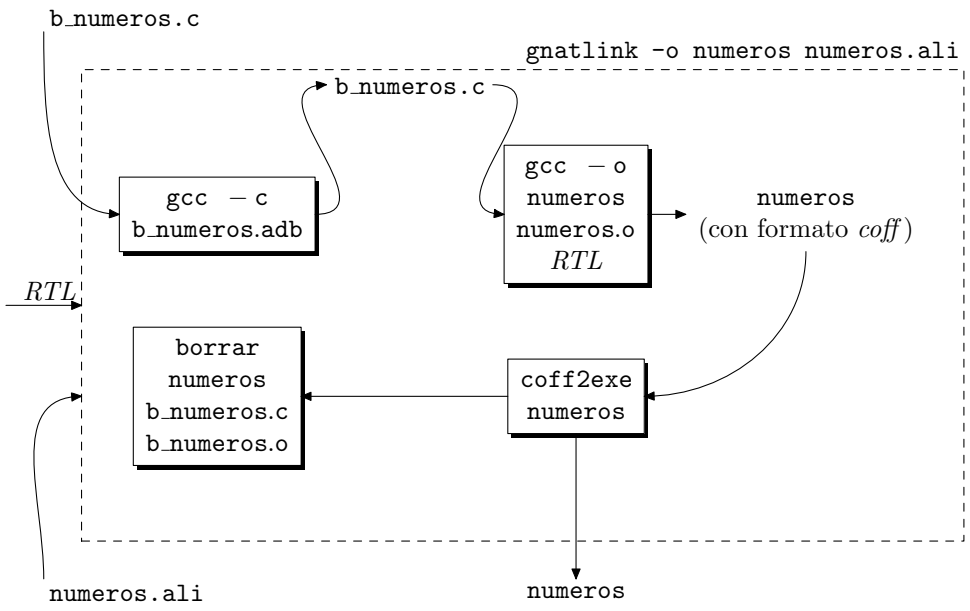

Fig. 1: Proceso de compilación, encuadernación y enlazado.

3. Ejecutar el programa `numeros`.
4. Cuando llamamos al procedimiento Put de la línea 26 ¿en qué paquete se encuentra este procedimiento?
5. Si quisiéramos poder reescribir la línea 34 para que quedara Put(I+J) ¿qué deberíamos añadir en el código y donde? Deja la línea 34 como se indica en esta cuestión y haz los cambios necesarios para que el programa funcione.
6. En general ¿cómo crees que el lenguaje sabe a qué procedimiento Put tiene que llamar si no se lo indicamos en el código?
7. En la parte declarativa del procedimiento **numeros**, declara un tipo discreto enumerado de la siguiente forma: `type Luz_de_Trafico_t is (Rojo, Ambar, Verde);`
8. En la parte declarativa del procedimiento **numeros**, declara una variable de tipo `type Luz_de_Trafico_t` con nombre `semaforo` e inicialízala a Rojo
9. Al igual que has visto en `numeros.adb` para las variables de tipo entero y flotante, crea un paquete llamado `Sem_ES` derivado del paquete `Ada.Text_Io.Enumeration_IO` para gestionar la entrada y salida por teclado de las variables de tipo `type Luz_de_Trafico_t`
10. Muestra por pantalla el valor de la variable `semaforo`

0.4 Obtención del compilador

El compilador del lenguaje Ada que hay instalado en el laboratorio forma parte de la distribución UBUNTU y se puede instalar en cualquier distribución con la orden:

```
$ sudo apt-get install gnat-7
```

Pueden instalarse múltiples versiones de gnat (gnat-X.X) siendo recomendable utilizar una versión superior a la 4. También puede instalarse con apt:

```
$ sudo apt install gnat-5
```

También existen compiladores para WINDOWS. En `www.macada.org` está disponible el compilador para OS X. También existen compiladores online como por ejemplo `https://onecompiler.com/ada`