

**Tema: Programación Concurrente en**

# Ada

## The Language For A Complex World

**1.- Antecedentes**

Durante los años 70 el Departamento de Defensa de los EEUU (DoD) tuvo que ocuparse de una gran variedad de lenguajes de programación (cerca de 450) que daban soporte a sus sistemas de computadoras embebidas. Esta gran variedad de lenguajes hizo que mucho software no se pudiera volver a emplear por falta de compatibilidad, dependencia del hardware y falta de soporte seguro en programación modular. Por este motivo se propusieron desarrollar un lenguaje de alto nivel para sistemas embebidos.

Se licito a concurso y quedaron cuatro finalistas, quienes fueron identificados por colores: Rojo (Red : *Intermetrics* liderado por Benjamin Brosgol), Verde (Green: *CII Honeywell Bull*, liderado por Jean Ichbiah), Azul (Blue: *SofTech*, liderado por John Goodenough), y Amarillo (Yellow: *SRI International*, liderado por Jay Spitzen). Después de dos fases fue elegida la compañía *CII Honeywell Bull* liderado por Jean Ichbiah. La tercera fase se inició con la elección del ganador y a continuación se le dio el nombre de Ada -en honor a Augusta Ada, Countess of Lovelace.

Tiempo después las especificaciones del lenguaje fueron publicadas por ACM (Association for Computing Machinery). En noviembre de 1979, con más de 500 sugerencias, se publicó en febrero de 1980 un diseño revisado del lenguaje. Después de algunos cambios menores a este documento la versión oficial se publicó. Durante los siguientes años han ocurrido significativos añadidos. Siendo los más nombrados Ada 95 y Ada 2005. Añadiendo aspectos como programación orientada a objetos, procesamiento concurrente y paralelo, etc.

A diferencia de muchos estándares ISO, la definición del lenguaje Ada (conocido como *Ada Reference Manual* o *ARM*, o algunas veces como *Language Reference Manual* o *LRM*) es de contenido libre. Así, se convierte en una referencia común para programadores y no sólo para los programadores que implementan los compiladores Ada.

Una notable herramienta de software libre que es usado por programadores Ada como ayuda en el desarrollo de código fuente es GPS, conocido como **GNAT Programming Studio**.

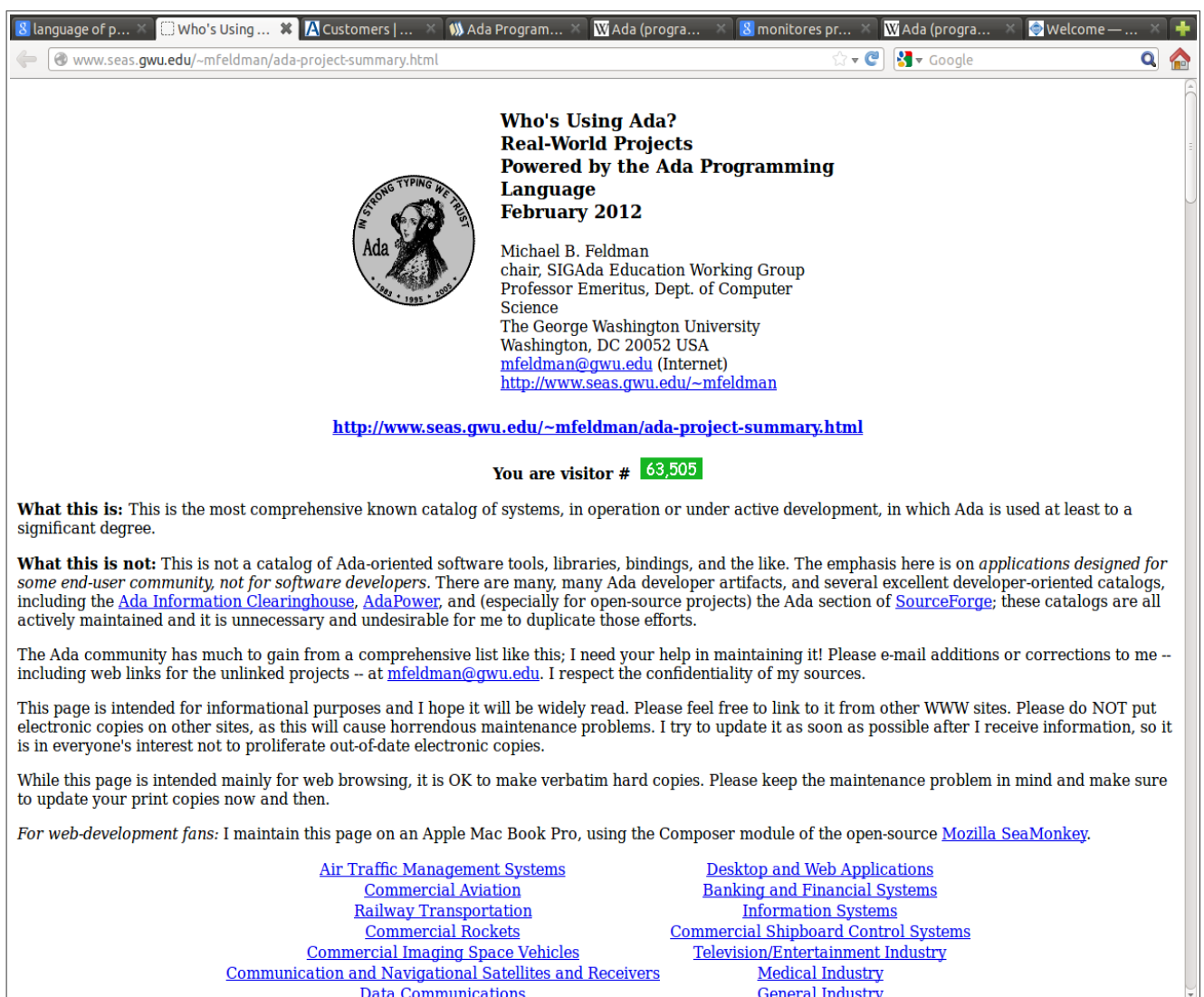
En este parte del curso los laboratorios se llevarán a cabo empleando este ambiente de programación. Como es de costumbre usted es libre de usar cualquier editor que le permita escribir, modificar, probar y ejecutar su código, ya que la compilación puede ser llevado en línea de comando. Las referencias a este procedimiento deben ser investigadas por su propia cuenta.

## 2.- ¿Es Ada un lenguaje de programación obsoleto?

Debido al soporte de aspectos de seguridad crítica de Ada, ahora no sólo se emplea en aplicaciones militares, sino también en proyectos comerciales donde un error de software puede tener serias consecuencias, por ejemplo: aviación y control de tráfico aéreo, cohetes comerciales, satélites y otros sistemas espaciales. Transporte ferroviario y de la banca.

Si desea puede visitar estas páginas:

[Who's using Ada?](#)



**Who's Using Ada?**  
**Real-World Projects**  
**Powered by the Ada Programming Language**  
**February 2012**

Michael B. Feldman  
 chair, SIGAda Education Working Group  
 Professor Emeritus, Dept. of Computer Science  
 The George Washington University  
 Washington, DC 20052 USA  
[mfeldman@gwu.edu](mailto:mfeldman@gwu.edu) (Internet)  
<http://www.seas.gwu.edu/~mfeldman>

<http://www.seas.gwu.edu/~mfeldman/ada-project-summary.html>

**You are visitor # 63,505**

**What this is:** This is the most comprehensive known catalog of systems, in operation or under active development, in which Ada is used at least to a significant degree.

**What this is not:** This is not a catalog of Ada-oriented software tools, libraries, bindings, and the like. The emphasis here is on *applications designed for some end-user community, not for software developers*. There are many, many Ada developer artifacts, and several excellent developer-oriented catalogs, including the [Ada Information Clearinghouse](#), [AdaPower](#), and (especially for open-source projects) the Ada section of [SourceForge](#); these catalogs are all actively maintained and it is unnecessary and undesirable for me to duplicate those efforts.

The Ada community has much to gain from a comprehensive list like this; I need your help in maintaining it! Please e-mail additions or corrections to me -- including web links for the unlinked projects -- at [mfeldman@gwu.edu](mailto:mfeldman@gwu.edu). I respect the confidentiality of my sources.

This page is intended for informational purposes and I hope it will be widely read. Please feel free to link to it from other WWW sites. Please do NOT put electronic copies on other sites, as this will cause horrendous maintenance problems. I try to update it as soon as possible after I receive information, so it is in everyone's interest not to proliferate out-of-date electronic copies.


While this page is intended mainly for web browsing, it is OK to make verbatim hard copies. Please keep the maintenance problem in mind and make sure to update your print copies now and then.

For web-development fans: I maintain this page on an Apple Mac Book Pro, using the Composer module of the open-source [Mozilla SeaMonkey](#).

[Air Traffic Management Systems](#)  
[Commercial Aviation](#)  
[Railway Transportation](#)  
[Commercial Rockets](#)  
[Commercial Imaging Space Vehicles](#)  
[Communication and Navigational Satellites and Receivers](#)  
[Data Communications](#)

[Desktop and Web Applications](#)  
[Banking and Financial Systems](#)  
[Information Systems](#)  
[Commercial Shipboard Control Systems](#)  
[Television/Entertainment Industry](#)  
[Medical Industry](#)  
[General Industry](#)

### Cientes de Ada Core


Company | Contact Us | GNAT Tracker Access


[Products](#)
[Developers](#)
[Training](#)
[Customers](#)
[Ada Answers](#)
[Academia](#)
[Support](#)


Home / Customers

**Customers**
Customer Projects | Customer List | Customer Quotes


## Customer Projects

The use of Ada and GNAT Pro continues to grow in high-integrity and safety-critical applications, including commercial and defense avionics, air traffic control, railroad systems, financial services and medical devices. Below, you'll find a series of case studies, articles and press releases showcasing some of our client projects. You can also [view a list of AdaCore clients worldwide](#) and [read customers' quotes](#) about our support and products.







**Metering and Grid Management**  
SmartSide  
SmartSide has adopted the Ada




**Land and Naval Surveillance and Defense Systems**  
Saab




**Military Helicopter ARINC 653**  
Eurocopter  
Eurocopter has chosen the GNAT Pro




**AMX Modernization program**  
Embraer  
Embraer Defence and Security has



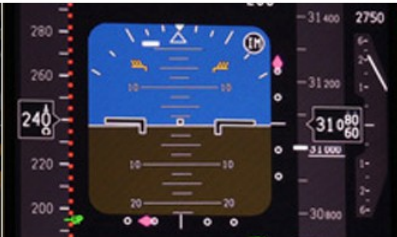
**Air Surveillance and Control System**  
Digicom Research




**Railway Control System**  
Siemens  
The Mobility Division of Siemens



**Advanced Jet Avionics Display**  
Barco  
Barco has developed an advanced



**Advanced Avionics Display System**  
Rockwell Collins



**nEUROn Unmanned Aircraft**  
EADS CASA  
EADS CASA is using the GNAT Pro High-

### 3.- Instalando GNAT y GNAT-GPS en Linux Mint 16.

Ejecute el *Gestor de Paquetes Synaptic*

Ubique y marque los paquete *gnat* y *gnat-gps* haciendo *click* derecho. A continuación acepte todas las dependencias que se muestran. Luego elija la opción *Aplicar*, del menú superior de *Synaptic*.

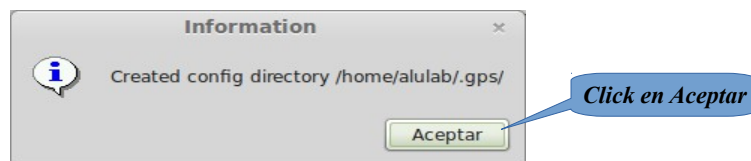
<input checked="" type="checkbox"/>	gnat	4.6ubuntu1	4.6ubuntu1	GNU Ada compiler
<input checked="" type="checkbox"/>	gnat-4.6	4.6.4-0ubuntu2	4.6.4-0ubuntu2	GNU Ada compiler
<input checked="" type="checkbox"/>	gnat-4.6-base	4.6.4-0ubuntu2	4.6.4-0ubuntu2	GNU Ada compiler (common files)
<input checked="" type="checkbox"/>	gnat-4.6-doc	4.6.4-0ubuntu2	4.6.4-0ubuntu2	GNU Ada compiler (documentation)
<input checked="" type="checkbox"/>	gnat-4.6-sjlj	4.6.4-0ubuntu2	4.6.4-0ubuntu2	GNU Ada compiler (setjump/longjump runtime library)
<input type="checkbox"/>	gnat-doc		4.6ubuntu1	Documentation for the GNU Ada compiler
<input checked="" type="checkbox"/>	gnat-gps	5.0-16	5.0-16	integrated development environment for C and Ada
<input checked="" type="checkbox"/>	gnat-gps-common	5.0-16	5.0-16	integrated development environment for C and Ada (common files)
<input type="checkbox"/>	gnat-gps-dbg		5.0-16	integrated development environment for C and Ada (debugging symbols)
<input checked="" type="checkbox"/>	gnat-gps-doc	5.0-16	5.0-16	integrated development environment for C and Ada (documentation)

### 4.- Nuestro primer Proyecto en Ada

Para iniciar el programa desde una terminal escriba<sup>1</sup>:

```
gnat-gps &
```

Si es la **primera vez** que es invocado, se mostrará la siguiente ventana



A continuación se mostrará las siguientes ventanas:

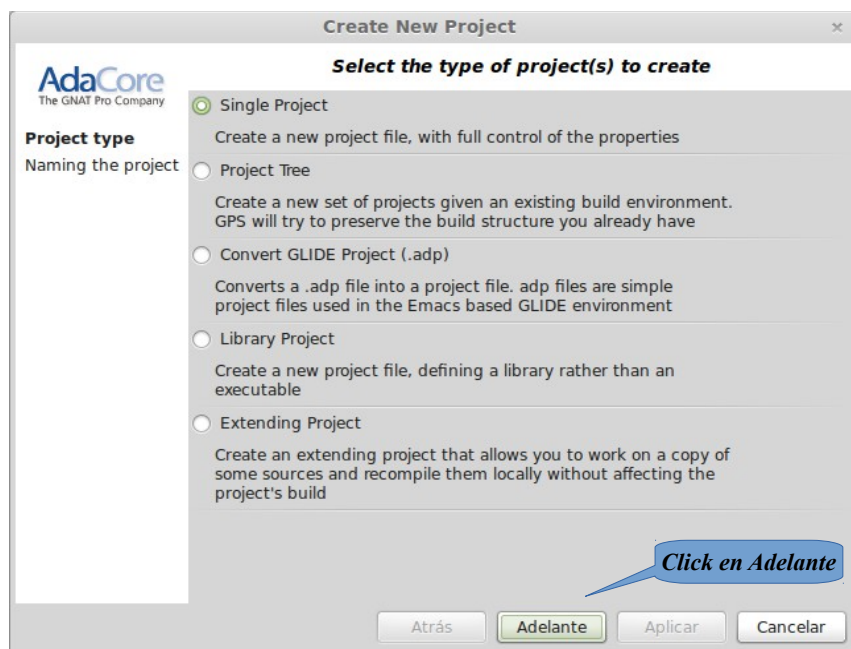


Después que se muestre la pantalla de *splash* de GPS, aparecerá una caja de diálogo para iniciar un proyecto. Para nuestro caso elija **Create new project with wizard**.

<sup>1</sup> Se puede crear una entrada en el menú principal. Se puede buscar en Internet para ver el procedimiento.



A continuación se muestra la primera pagina del asistente, elija la opción **Single Project**



En la segunda página se debe ingresar el nombre del proyecto así como el la carpeta donde este será almacenado. Para este último caso existen algunas posibilidades:

- Normalmente el asistente sugiere el directorio *home*. Pero usted puede elegir escribir la ruta absoluta de la carpeta o elegir la carpeta con la ayuda del botón *Browse*.
- Otra opción es que la carpeta no exista, pero el asistente le ofrecerá crearla después.

Para efectos prácticos de esta guía, escriba como nombre de proyecto: **Lab2SO20141** y como carpeta del proyecto **/home/alulab/Documentos/20111205/** (escriba usted sólo su código de alumno), tal como se muestra a continuación:



The screenshot shows the 'Create New Project' dialog box with the 'Name & Location' tab selected. The dialog has a sidebar on the left with the following options: Project type, Naming the project, VCS, Source dirs, Objects, Main files, Naming scheme, and Switches. The main area contains the following fields and controls:

- Name & Location**: A section header.
- Enter the name of the project to create:** A text input field containing 'Lab2SO20141'.
- Enter the directory where the project file will be created:** A text input field containing '/home/alulab/Documentos/20111205/' with a 'Browse' button to its right.
- General**: A section header.
- ☒ **Use relative paths in the projects**: A checkbox that is checked.

At the bottom right, there is a blue callout bubble with the text 'Click en Adelante'. At the bottom center, there are four buttons: 'Atrás', 'Adelante', 'Aplicar', and 'Cancelar'.

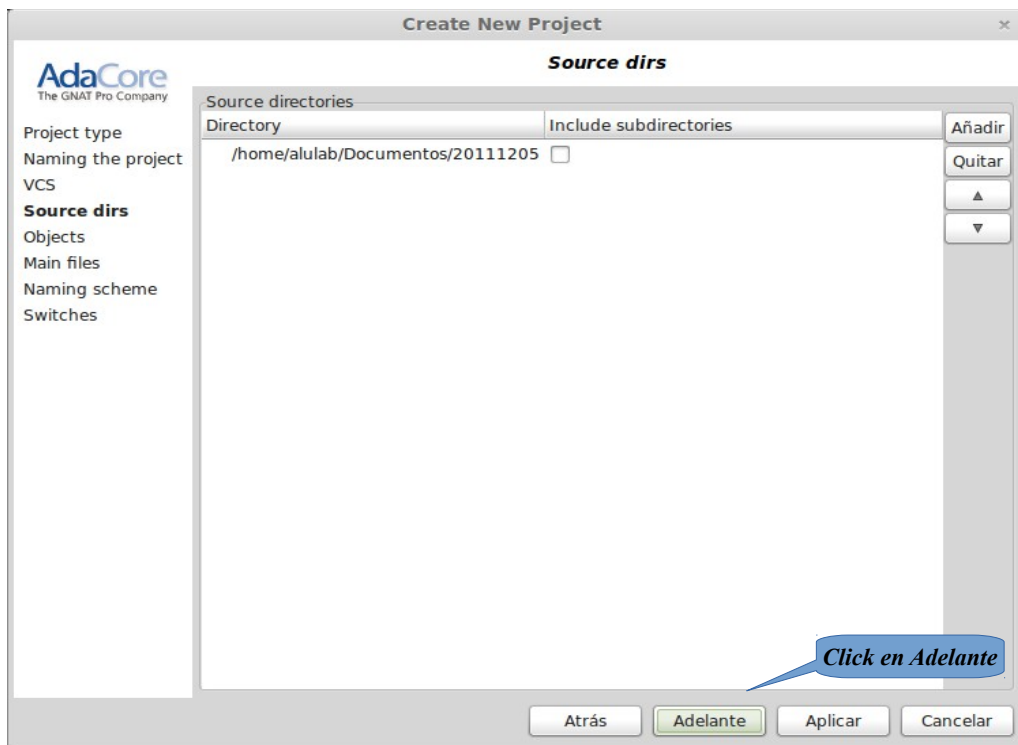
En la tercera página no modifique opción alguna.

The screenshot shows the 'Create New Project' dialog box with the 'VCS' tab selected. The sidebar on the left is the same as in the previous screenshot, but 'VCS' is now selected. The main area contains the following fields and controls:

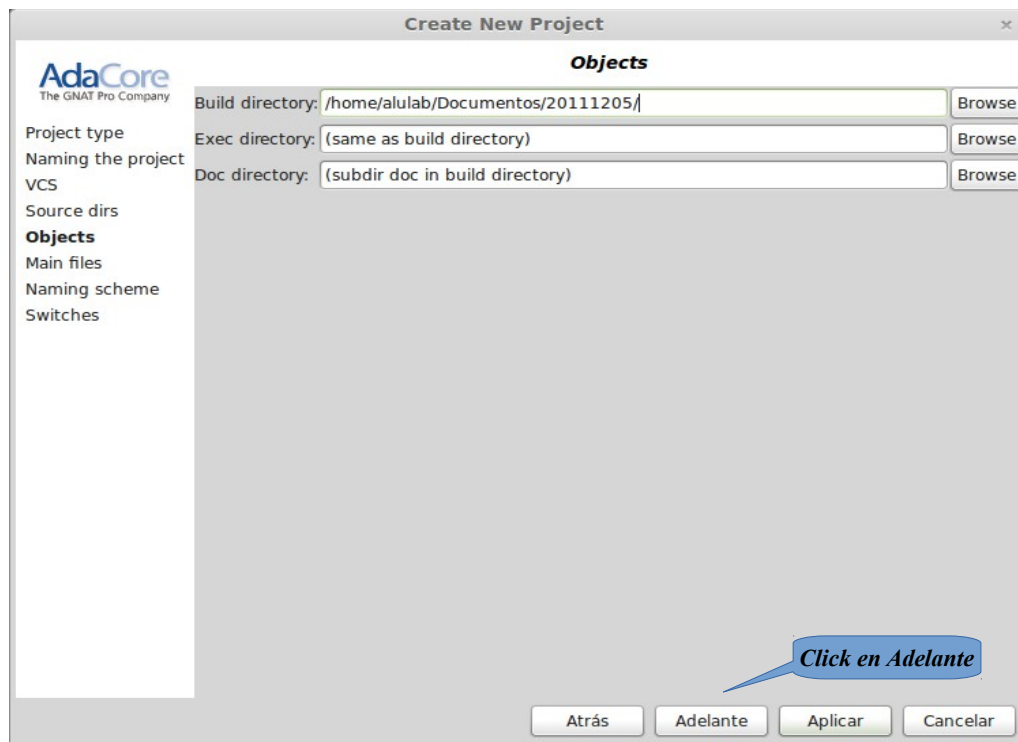
- System**: A dropdown menu with 'None' selected.
- Actions**: A section header.
- Log checker:** A text input field.
- File checker:** A text input field.
- Path**: A section header.
- Repository:** A text input field.
- Patch:** A text input field.

At the bottom right, there is a blue callout bubble with the text 'Click en Adelante'. At the bottom center, there are four buttons: 'Atrás', 'Adelante', 'Aplicar', and 'Cancelar'.

En la cuarta página no modifique opción alguna



En la quinta página se le solicitará ingresar la carpeta donde se construirá el ejecutable. Puede indicarle la misma carpeta que se indico al asistente en la segunda página: */home/alulab/Documentos/20111205/*



En la sexta página el asistente nos permite adjuntar un programa fuente (Ada) ya existente. En caso de que no tengamos alguno (como es este caso) lo añadiremos después.

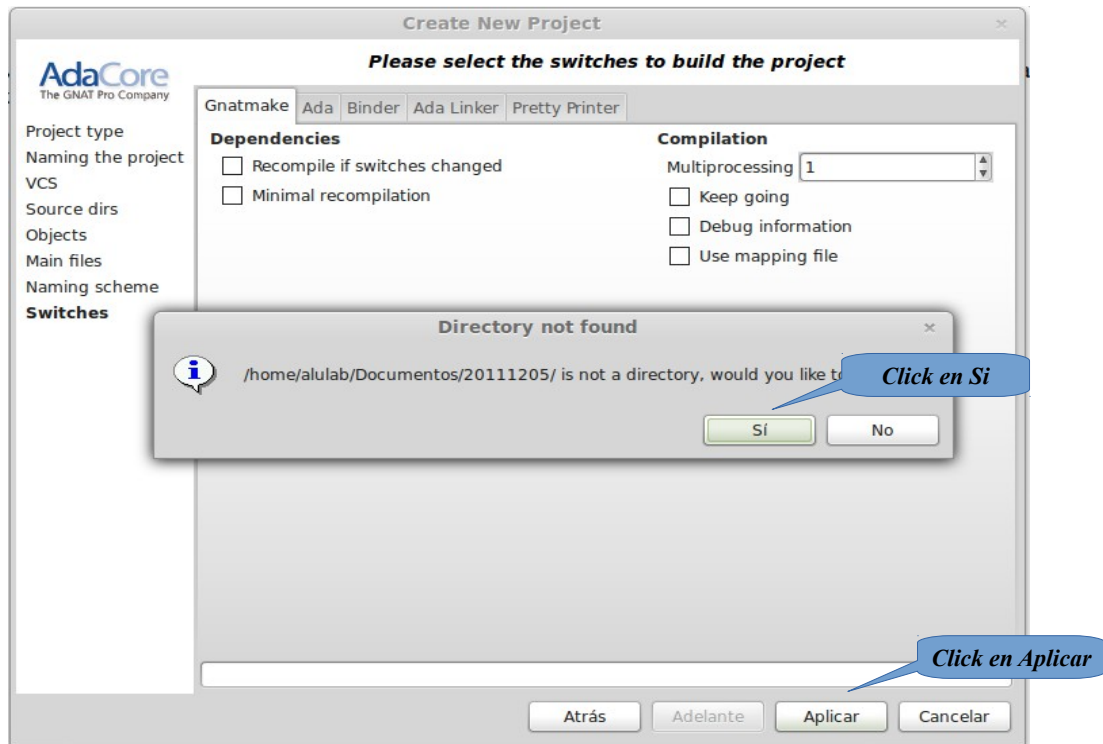
The screenshot shows the 'Create New Project' dialog box with the 'Main files' tab selected. The left sidebar lists the steps: Project type, Naming the project, VCS, Source dirs, Objects, Main files (highlighted), Naming scheme, and Switches. The main area has a 'Main files' list with 'Añadir' and 'Quitar' buttons. Below this is a 'Default suffix:' field and an 'Executable names' section with 'main (Click to edit)'. At the bottom are 'Atrás', 'Adelante', 'Aplicar', and 'Cancelar' buttons. A blue callout bubble points to the 'Adelante' button with the text 'Click en Adelante'.

En la séptima página no modifique opción alguna

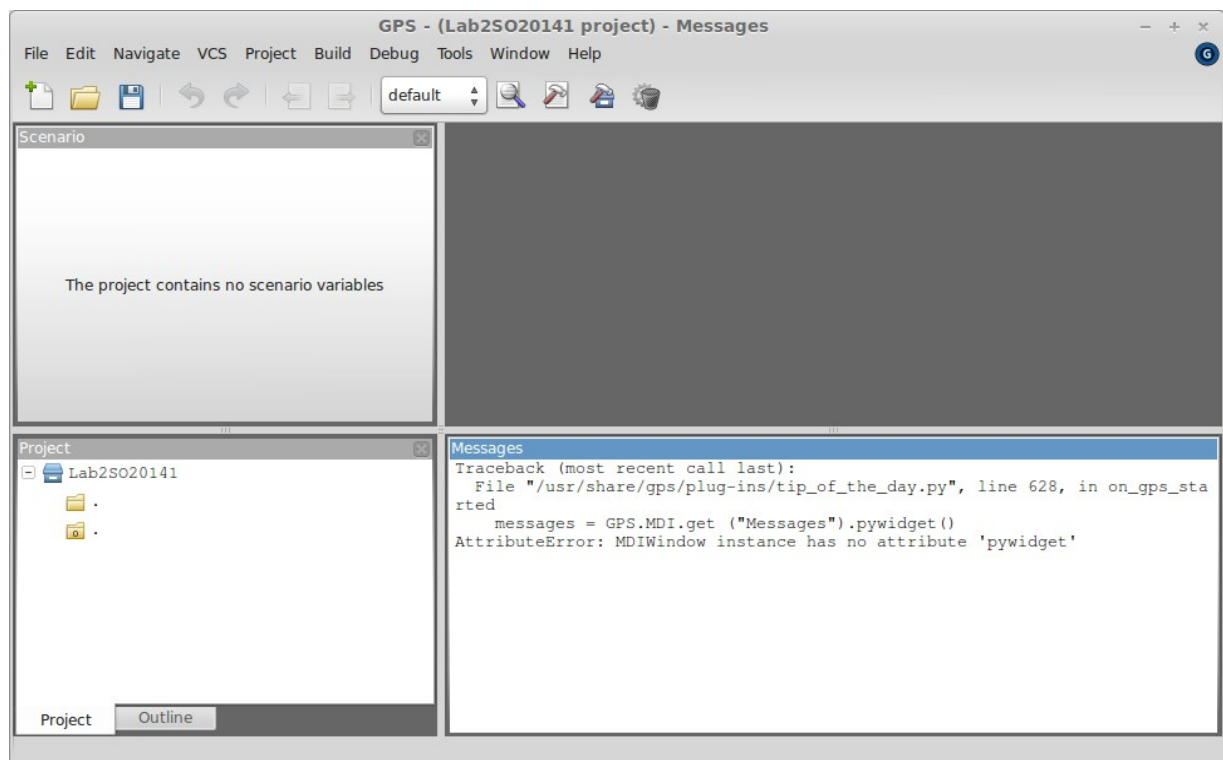
The screenshot shows the 'Create New Project' dialog box with the 'Naming scheme' tab selected. The left sidebar highlights 'Naming scheme'. The main area is titled 'Please select the naming scheme to use' and shows the 'Ada' section. It includes a 'Naming scheme:' dropdown set to 'GNAT default', a 'Details' section with 'Filename casing:' set to 'lowercase', 'Dot replacement:' set to '-', 'Spec extensions:' set to '.ads', 'Body extensions:' set to '.adb', and 'Separate extensions:' set to '.adb'. There is an 'Exceptions' section with tabs for 'Unit name', 'Spec filename', and 'Body filename'. At the bottom are 'Atrás', 'Adelante', 'Aplicar', and 'Cancelar' buttons. A blue callout bubble points to the 'Adelante' button with the text 'Click en Adelante'.



En la octava página acepte las opciones por defecto. En este momento si la carpeta que indicamos tanto en la página 2 como en la página cinco no existe, el asistente se ofrecerá a crearlo.



Al finalizar el asistente habrá completado el proceso de creación del proyecto y deberá mostrar el entorno dividido en varias ventanas. La pequeña ventana de la derecha indica que el proyecto no tiene ningún programa fuente.



Para crear un archivo nuevo puede hacerlo de dos formas:

- Elija del menú principal **File** → **New**
- De la barra de herramientas haga *click* en el ícono



Una vez creado el archivo vacío escriba el siguiente programa:

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

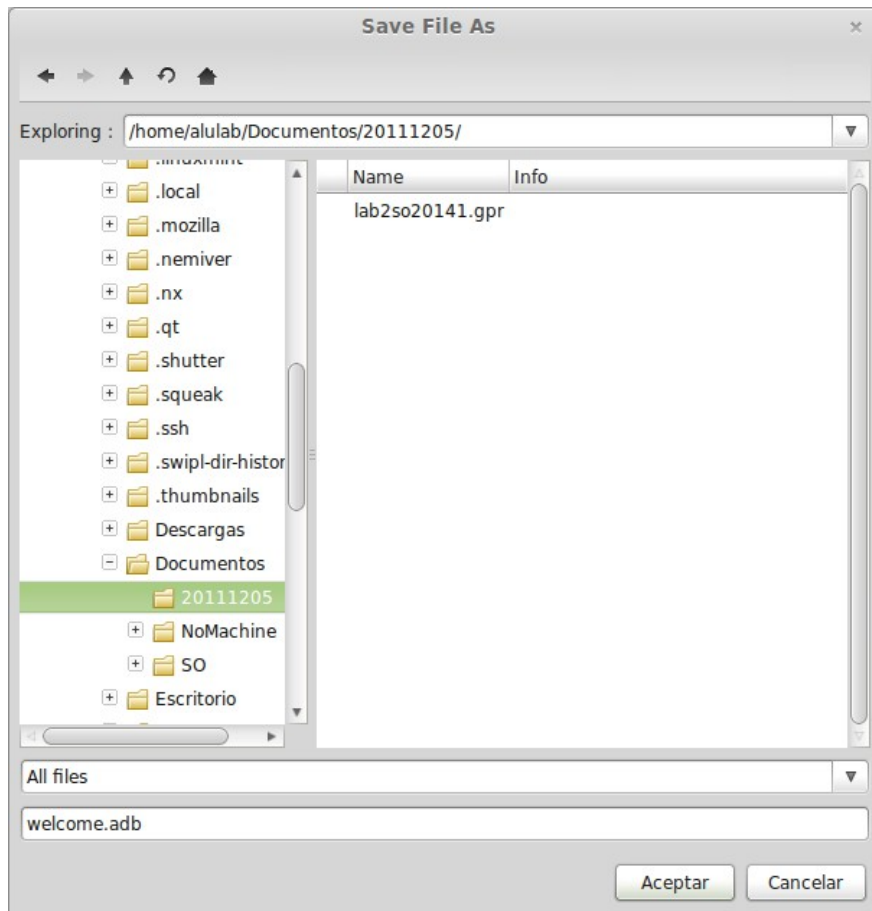
procedure Welcome is
begin
  Put_Line("Welcome to Ada Programming");
end Welcome;
```

Después de escribir el código grabe el archivo

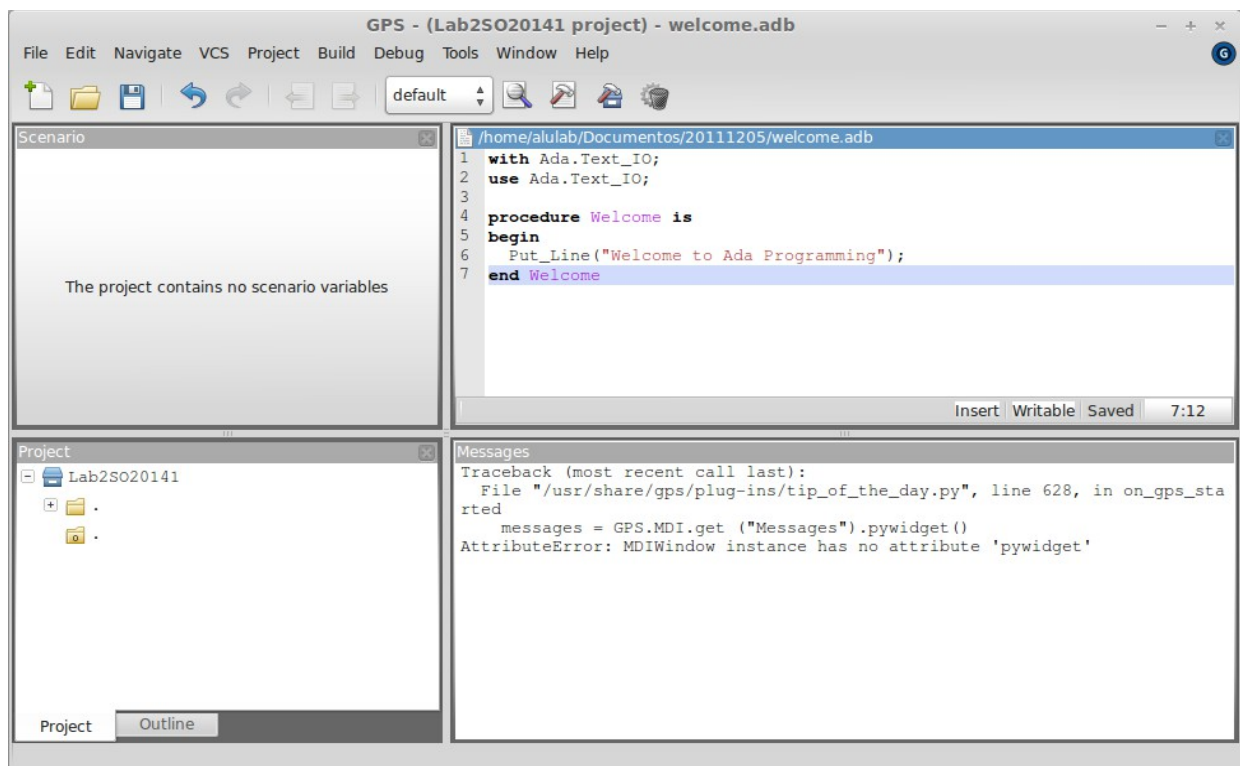
- Elija del menú principal **File** → **Save**
- Presione Ctrl + S
- Desde la barra de herramientas haga *click* en



Si anteriormente no ha sido grabado el archivo, se mostrará una caja de diálogo en la que usted deberá indicar tanto el nombre como el lugar. El nombre normalmente viene sugerido por GPS. Ubique el archivo en la carpeta donde se creó el proyecto (*/home/alulab/Documentos/20111205/*).

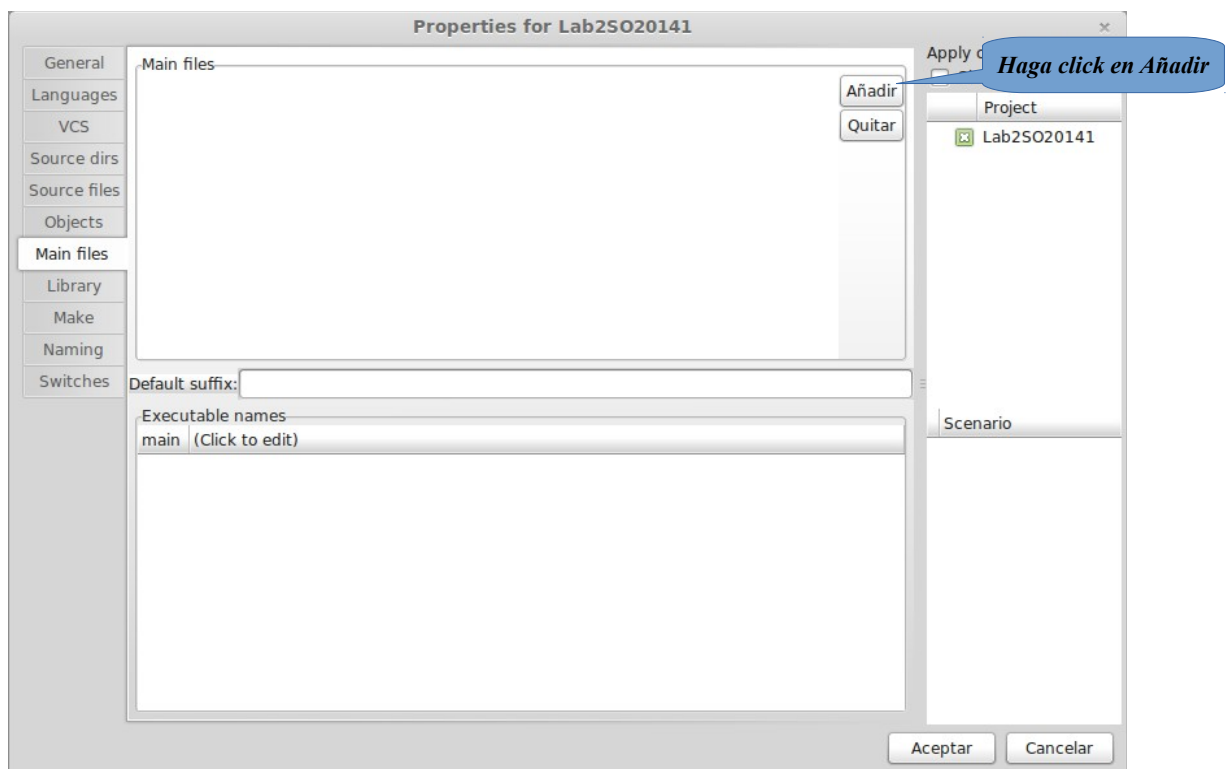
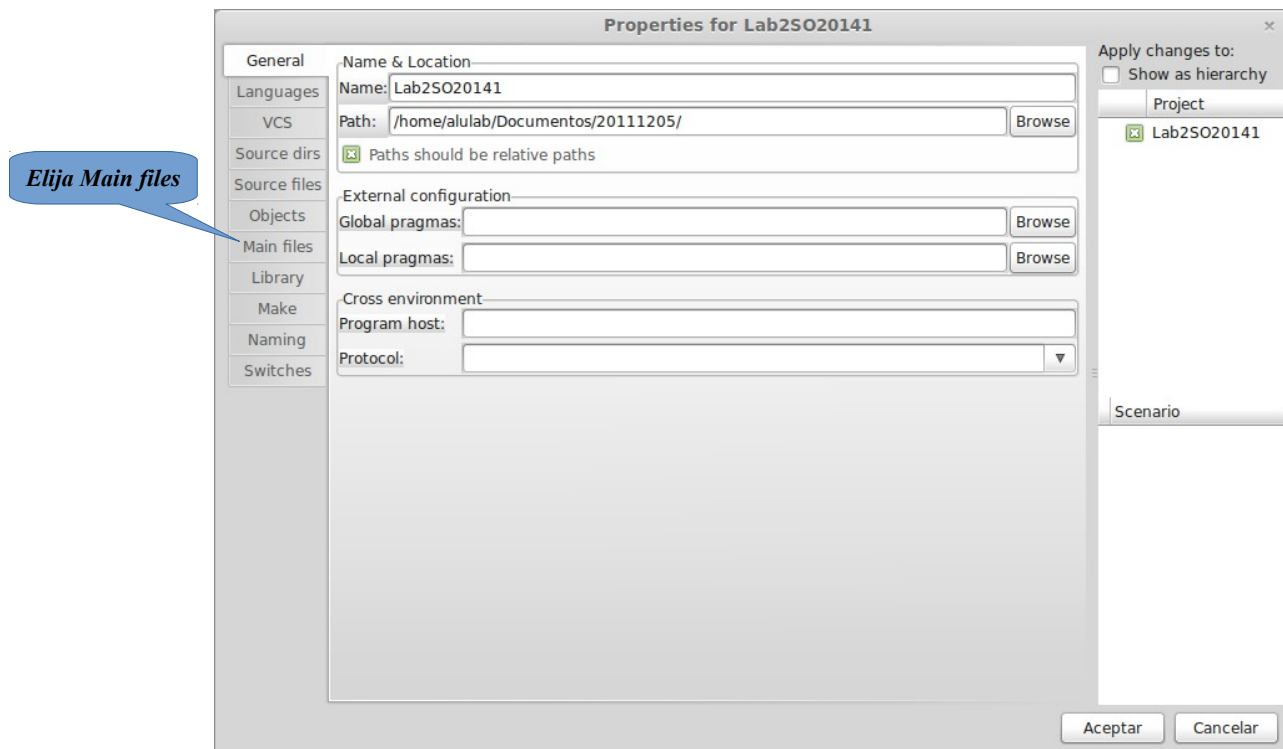


Al finalizar usted deberá obtener la siguiente ventana.



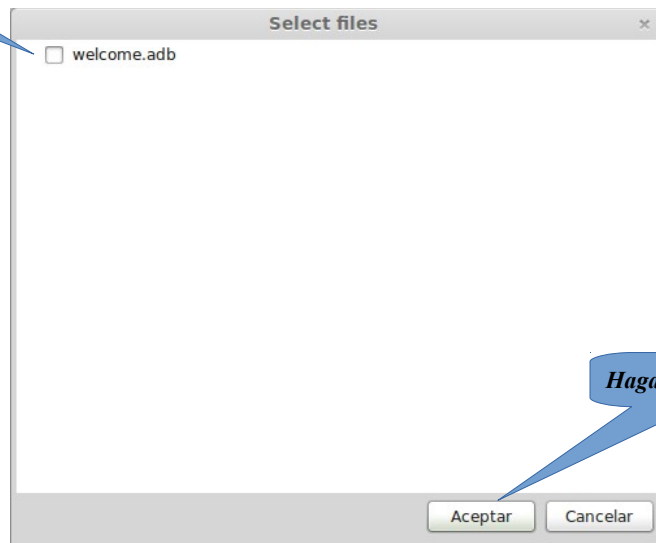
Después de haber creado y grabado el archivo, tiene que ser añadido al proyecto. Este proceso debería ser llevado a cabo eligiendo del menú principal: **Project** → **Edit Project Properties**.

Se mostrará la siguiente ventana:



Aparecerá la siguiente ventana:

Marque esta casilla

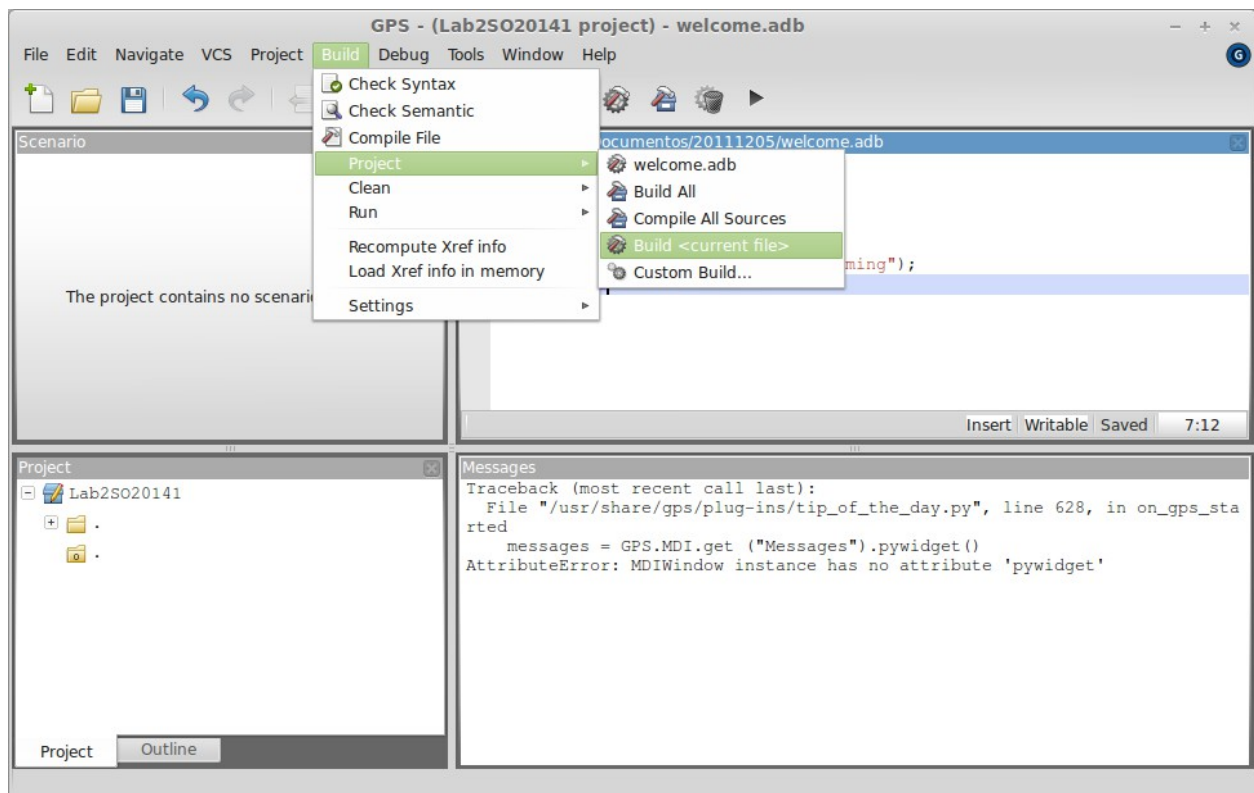


Haga click en Aceptar

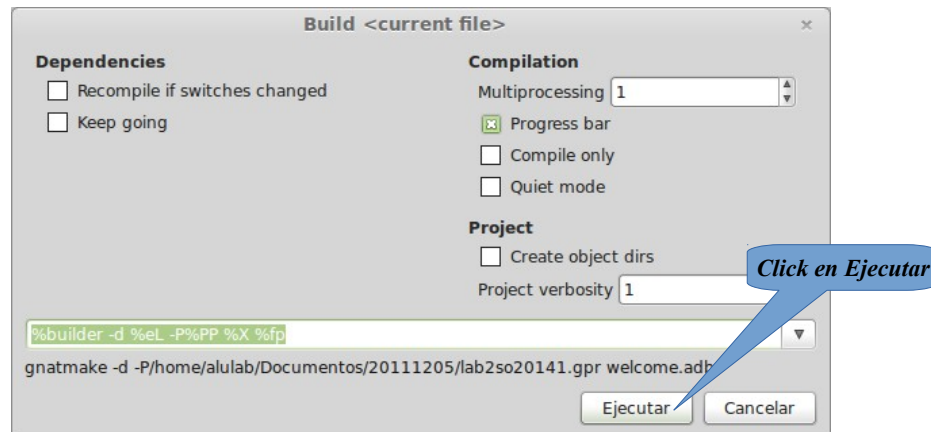
Regresará a la ventana de propiedades del proyecto. En esta ventana haga *click* en Aceptar, y eso es todo.

Ahora construiremos el proyecto.

Elija la opción **Build** → **Project** → **Build <current file>**



Se mostrará una caja de diálogo, haga *click* en **Ejecutar**

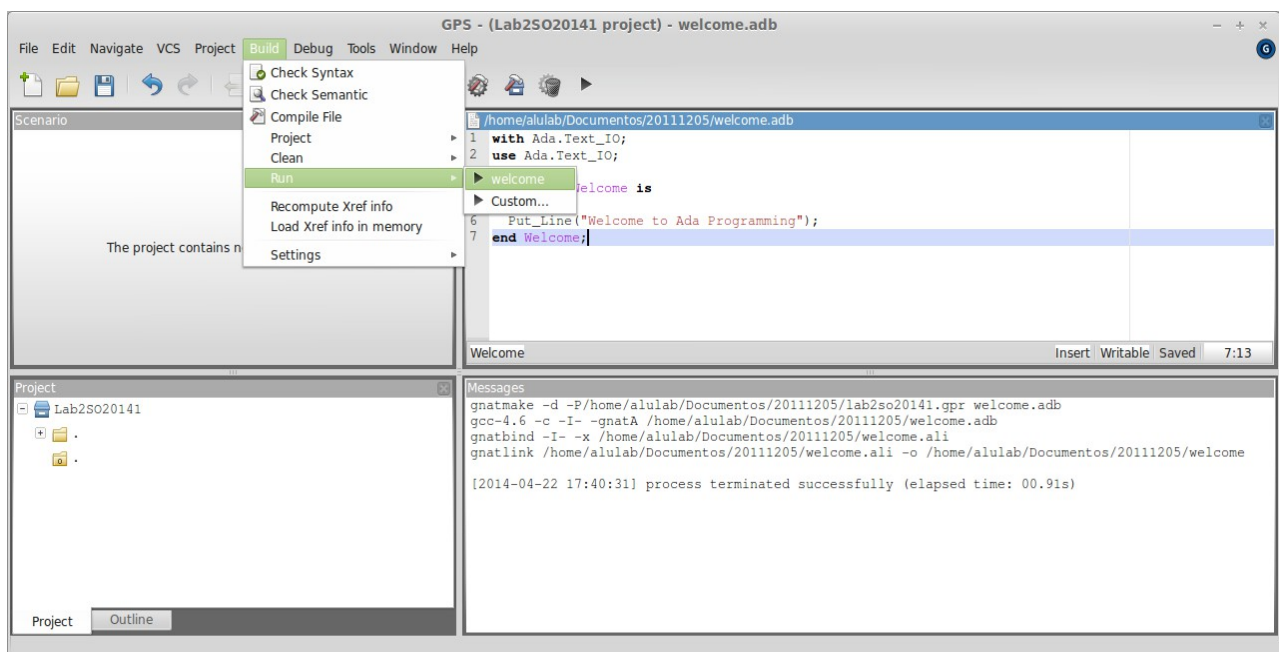


En la parte inferior derecha (sección de mensajes) se mostrará el resultado de la compilación tal como se muestra en la siguiente imagen.



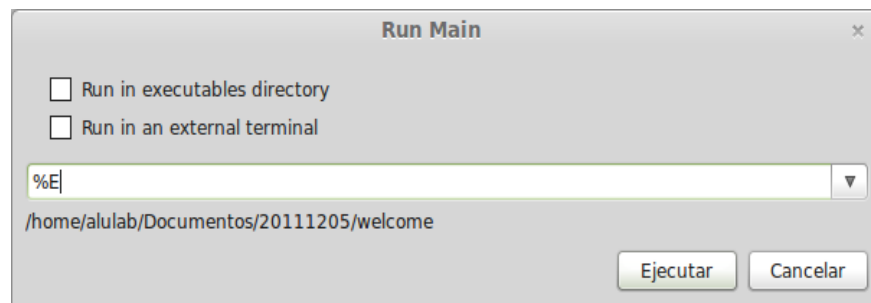
Para ejecutar el programa

Elija del menú principal la opción **Build** → **Run** → **welcome**

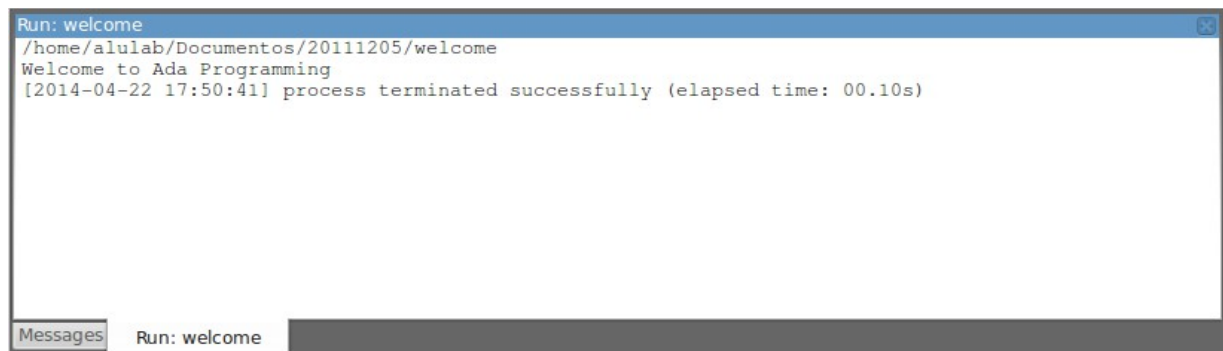




Se mostrará una caja de diálogo, si no necesita ingresar ningún argumento (como en este caso) haga *click* en *Ejecutar*.



Observe la salida en la ventana inferior derecha.



## 5.- Algunas consideraciones

- a) En un proyecto en Ada no hay un programa principal como en Pascal.
- b) Un procedimiento puede ejecutarse como un proceso.
- c) Si un procedimiento necesita ser modular puede invocar a otros procedimientos o funciones pero deben estar declarados en el interior del procedimiento principal (de forma anidada, al estilo Pascal).
- d) El conjunto de procedimientos y funciones auxiliares pueden escribirse como módulos separados pero tanto la interfaz como la implementación deben ir en archivos separados (esto sería el análogo de las Unidades en Pascal, con la diferencia de que la interfaz y la implementación se escriben de forma separada)
- e) El objetivo del laboratorio es explorar los aspectos de concurrencia que proporciona el lenguaje de programación Ada, que a conocer son tres:
  - Cargar y almacenar variables compartidas.
  - Objetos protegidos para compartir recursos asíncronos
  - *Rendezvous* para comunicación síncrona directa de tarea a tarea.

Para llevar a cabo este último objetivo dividiremos el material relacionado con el lenguaje en dos partes:

- i) Programación Básica: se mostrará los tipos básicos, los constructores fundamentales del lenguaje con algunos ejemplos sencillos de aplicación. El objetivo es familiarizarse con el lenguaje sin tener que profundizar en detalles particulares o diferentes a la programación concurrente.
- ii) Programación Concurrente: se mostrará los constructores básicos que permite emplear los aspectos de concurrencia para modelar diferentes problemas clásicos tales como: productor-consumidor, lectores-escritores, la cena de los filósofos, el barbero dormilón, etc.

## 6.- Programación Básica

Un procedimiento en Ada tiene la siguiente estructura:

- Las especificaciones del contexto: son directivas al compilador Ada. Puede incluir determinadas librerías.
- Las especificaciones del procedimiento: se refiere a la cabecera del procedimiento e incluyen el nombre y las variables que reciben los argumentos si los tuviera.
- La parte declarativa: en esta parte se declaran las variables del procedimiento o procedimientos y funciones que este va a emplear.
- El cuerpo del procedimiento.

### Tipos predefinidos y operadores

**Tipo *integer*:** Representa una secuencia de uno o más dígitos. Ada permite que cualquiera de los dígitos (excepto el primero) esté precedido por el carácter “\_” para dar legibilidad al programa.

Ejemplo:

```
57
18_502_007
10_18502007_001
```

Por defecto, se asume que son números decimales, pero existen mecanismos para expresar enteros en otra base:

Ejemplo:

8#377# 377	octal (255 en decimal)
16#FF# FF	hexadecimal (255 en decimal)
2#1111_1111# 1111_1111	binario (255 en decimal)

La base siempre se define en decimal y puede ser entre 2 y 16.

Están definidas las operaciones aritméticas tradicionales tales como +, -, \*, /, y -(negación) Los números negativos técnicamente son expresiones y no literales. Existen dos procedimientos en **ada\_io** para leer y escribir números enteros: **get(i)** y **put(i)**.

**Tipo flotante:** se distinguen por la presencia de un punto decimal. Pueden incluir un exponente.

Ejemplo:

```
10.0
0.5
3.14159_26
1.15E-12
1.0e+6
```

Las mismas operaciones que para los reales de Pascal.

**Importante:** Los tipos a pesar que son numéricos no pueden mezclarse entre sí.

**Tipo Boolean:** Los valores booleanos están denotados por unos de dos literales predefinidos: **true** y **false**. Los operadores para este tipo son:

```
x and y
x or y
x xor y
not x
```

y los operadores relacionales (=, !=, <, >, etc.)

**Tipo Character:** Los valores de este tipo representan valores ASCII. Son caracteres encerrados entre comillas simples.

Ejemplo: 'a', 'A', "

Las únicas operaciones definidas para el tipo *character* son las relacionales.

**Tipo string:** todos los tipos anteriores son tipos escalares, esto es, que tiene valores que no pueden ser divididos en partes más pequeñas. Ada tiene un tipo predefinido que es un tipo compuesto, el tipo **String**, que consiste de una secuencia de valores *character*.

Un literal *string* es una secuencia de cero, o más caracteres encerrados entre comillas dobles.

Ejemplo:

```
"estos es un string"
" "
"123"
```

**Expresiones** No se pueden mezclar variables de distintos tipos en una misma expresión.

Ejemplo:

```
i: integer;
x: float;

x:= 3.7;
i:= 1;
x:= x+i; (incorrecto)
```

La forma correcta es hacer un *casting*, por ejemplo:

```
x := x + float(i);
```

## Declaraciones

Existen dos tipos de objetos: variables y constantes.

Ejemplo:

```
x: integer;  
y: constant float:=1.9;
```

**x** es una variable e **y** es una constante de tipo flotante.

La declaración de constantes como en Pascal, debe incluir una inicialización. Las declaraciones de variables pueden incluir una inicialización que especifica el valor inicial de las variables.

Ejemplo:

```
n: integer:= 10;  
p: integer:= n+1;
```

A continuación se muestran diferentes estructuras en ADA para que usted pueda escribir programas más complejos.

### If else

```
if condition then  
    statement;  
else  
    other statement;  
end if;
```

```
if condition then  
    statement;  
elsif condition then  
    other statement;  
elsif condition then  
    other statement;  
...  
else  
    another statement;  
end if;
```

```
with Ada.Text_IO;  
use Ada.Text_IO;  
...  
type Degrees is new Float range -273.15 .. Float'Last;  
...  
Temperature : Degrees;  
...  
if Temperature >= 40.0 then  
    Put_Line ("Wow!");  
    Put_Line ("It's extremely hot");  
elsif Temperature >= 30.0 then  
    Put_Line ("It's hot");  
elsif Temperature >= 20.0 then  
    Put_Line ("It's warm");  
elsif Temperature >= 10.0 then  
    Put_Line ("It's cool");  
elsif Temperature >= 0.0 then  
    Put_Line ("It's cold");  
else  
    Put_Line ("It's freezing");  
end if;
```

*case*

```
case X is
  when 1 =>
    Walk_The_Dog;
  when 5 =>
    Launch_Nuke;
  when 8 | 10 =>
    Sell_All_Stock;
  when others =>
    Self_Destruct;
end case;
```

*Lazos infinitos*

```
Endless_Loop :
loop
  Do_Something;
end loop Endless_Loop;
```

*Lazos con condición al inicio*

```
While_Loop :
while X <= 5 loop
  X := Calculate_Something;
end loop While_Loop;
```

*Lazos con condición al final*

```
Until_Loop :
loop
  X := Calculate_Something;
  exit Until_Loop when X > 5;
end loop Until_Loop;
```

*Lazos con condición en medio*

```
Exit_Loop :
loop
  X := Calculate_Something;
  exit Exit_Loop when X > 5;
  Do_Something (X);
end loop Exit_Loop;
```

*Lazos for*

```
For_Loop :  
  for I in Integer range 1 .. 10 loop  
    Do_Something (I)  
  end loop For_Loop;
```

*Una combinación de formatos*

```
if Boolean expression then  
  statements  
elsif Boolean expression then  
  statements  
else  
  statements  
end if;
```

```
while Boolean expression loop  
  statements  
end loop;
```

```
for variable in range loop  
  statements  
end loop;
```

```
declare  
  declarations  
begin  
  statements  
exception  
  handlers  
end;
```

```
procedure P (parameters : in out type) is  
  declarations  
begin  
  statements  
exception  
  handlers  
end P;
```

```
function F (parameters : in type) return type is  
  declarations  
begin  
  statements  
exception  
  handlers  
end F;
```

```
package P is  
  declarations  
private  
  declarations  
end P;
```

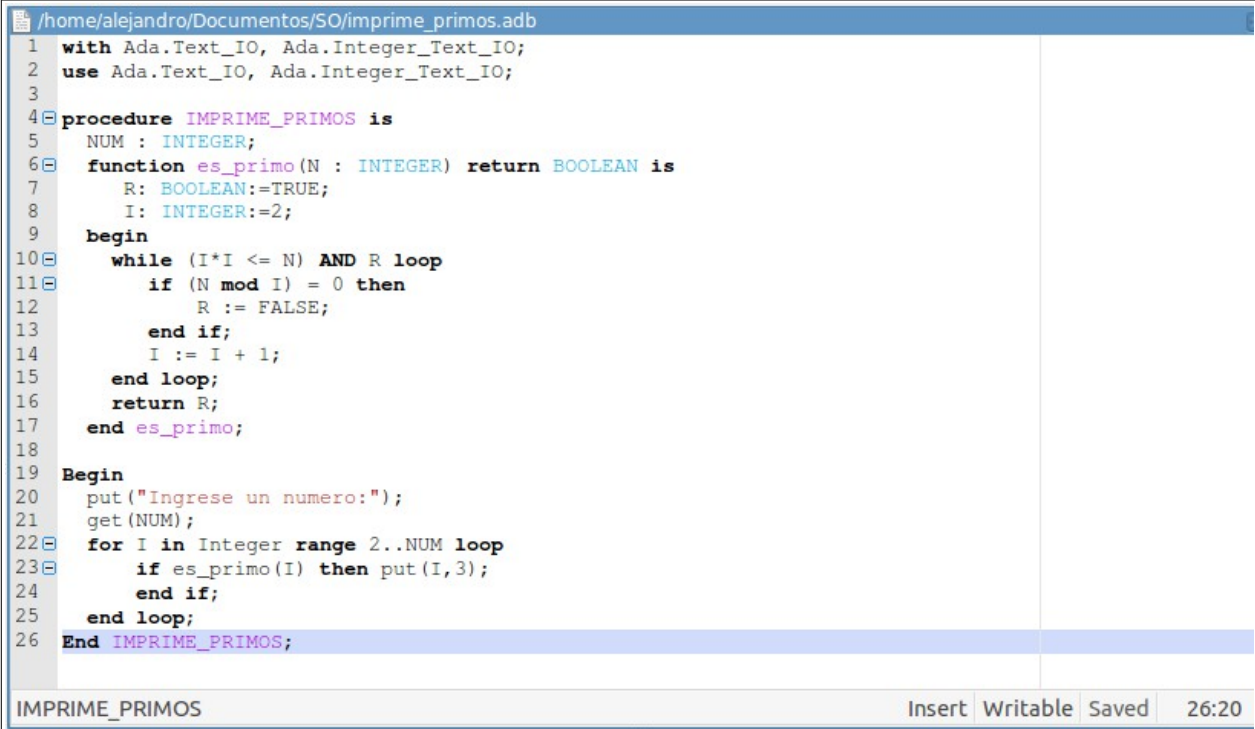


### Ejercicios

- 1.- Elabore un programa en ADA que reciba un número e imprima un mensaje si el número es primo o no.
- 2.- Elabore un programa en ADA que reciba un número e imprima todos los primos desde 2 hasta dicho número.
- 3.- Elabore un programa en ADA que reciba un número y devuelva el número Fibonacci de dicho número. No emplee la versión recursiva.
- 4.- Elabore un programa en ADA que reciba un número y devuelva el factorial de dicho número. No emplee la versión recursiva.
- 5.- Elabore un programa en ADA que reciba una cadena e imprima la cadena en forma inversa.
- 6.- Elabore un programa en ADA que reciba una cadena e imprima un mensaje diciendo si la cadena es palíndroma.
- 7.- Elabore un programa en ADA que halle el MCD de dos números, empleando el algoritmo de Euclides.
- 8.- Elabore un programa en ADA que se ingrese una número que representa la cantidad de segundos, el programa deberá imprimir a qué cantidad de horas, minutos y segundos corresponde.
- 9.- Elabore un programa en ADA que se ingresa un número e imprima su representación en base 2.
- 10.- Elabore un programa en ADA que dado un número imprima el triángulo de PASCAL. El número indica la cantidad de filas.

### Ejemplo

A continuación un pequeño programa que cuando se le ingresa un número por entrada estándar, imprime todos los números primos desde 2 hasta dicho número.



```
1 with Ada.Text_IO, Ada.Integer_Text_IO;
2 use Ada.Text_IO, Ada.Integer_Text_IO;
3
4 procedure IMPRIME_PRIMOS is
5   NUM : INTEGER;
6   function es_primo(N : INTEGER) return BOOLEAN is
7     R: BOOLEAN:=TRUE;
8     I: INTEGER:=2;
9     begin
10      while (I*I <= N) AND R loop
11        if (N mod I) = 0 then
12          R := FALSE;
13        end if;
14        I := I + 1;
15      end loop;
16      return R;
17    end es_primo;
18
19 Begin
20   put ("Ingrese un numero:");
21   get (NUM);
22   for I in Integer range 2..NUM loop
23     if es_primo(I) then put (I,3);
24   end if;
25   end loop;
26 End IMPRIME_PRIMOS;
```

```

Messages
gnatmake -d -P/home/alejandro/Documentos/SO/soperativos.gpr imprime_primos.adb
gcc-4.6 -c -I- -gnatA /home/alejandro/Documentos/SO/imprime_primos.adb
gnatbind -I- -x /home/alejandro/Documentos/SO/imprime_primos.ali
gnatlink /home/alejandro/Documentos/SO/imprime_primos.ali -o /home/alejandro/Documentos/SO/imprime_primos

[2012-09-29 11:26:09] process terminated successfully (elapsed time: 00.25s)

```

```

Run: imprime_primos
/home/alejandro/Documentos/SO/imprime_primos
Ingrese un numero:10
 2  3  5  7
[2012-09-29 11:26:15] process terminated successfully (elapsed time: 03.30s)

```

## 7.- Programación Concurrente

En Ada existen las tareas (tasks), estos son los análogos a los hilos en otros lenguajes. El programa principal se considera una tarea. Para poder emplear tareas en Ada, primero se tiene que declarar y luego se define el cuerpo de la tarea. El esquema es el siguiente:

```

task Tarea is
    -- declaración de identificadores exportados
end Tarea;

task body Tarea is
    -- sentencias y declaraciones locales
end Tarea;

```

Si la tarea no tiene identificador alguno a exportar la declaración se simplifica

```

task Tarea;

```

Las tareas se ejecutan en el momento que el procedimiento que los contiene se ejecuta. Si se desea controlar la ejecución de las tareas, esto se debe de hacer desde el procedimiento principal. A continuación un ejemplo sencillo:

```

/home/alejandro/Documentos/SO-Casa/Lab3/principal.adb
1  with Ada.Text_IO;
2  use Ada.Text_IO;
3
4  procedure Principal is
5
6      task Tarea1;
7      task Tarea2;
8
9      task body Tarea1 is
10         final : Positive := 10;
11         begin
12             for count in 1..final loop
13                 Put_Line("Tarea1");
14             end loop;
15         end Tarea1;
16
17         task body Tarea2 is
18             final : Positive := 10;
19             begin
20                 for count in 1..final loop
21                     Put_Line("Tarea2");
22                 end loop;
23             end Tarea2;
24
25         Begin
26             for count in 1..10 loop
27                 Put_Line("Tarea Principal");
28             end loop;
29         End Principal;

```

Es interesante hacer notar que al momento de ejecutarlo varias veces, se tiene diferentes salidas, como se observa a continuación.

```
Run: principal
/home/alejandro/Documentos/SO-Casa/Lab3/principal
Tarea1
Tarea1
Tarea1
Tarea1
Tarea1
Tarea1
Tarea1
Tarea1
Tarea1
Tarea1
Tarea1
Tarea Principal
Tarea Principal
Tarea Principal
Tarea Principal
Tarea Principal
Tarea Principal
Tarea Principal
Tarea Principal
Tarea Principal
Tarea2
Tarea2
Tarea2
Tarea2
Tarea2
Tarea2
Tarea2
Tarea2
Tarea2
[2013-04-27 08:03:11] process terminated successfully (elapsed time: 00.10s)
```

```
Run: principal
/home/alejandro/Documentos/SO-Casa/Lab3/principal
Tarea1
Tarea1
Tarea1
Tarea1
Tarea2
Tarea2
Tarea2
Tarea1
Tarea1
Tarea1
Tarea2
Tarea Principal
Tarea Principal
Tarea Principal
Tarea Principal
Tarea Principal
Tarea Principal
Tarea Principal
Tarea Principal
Tarea Principal
Tarea2
Tarea2
Tarea2
Tarea2
Tarea2
Tarea1
Tarea1
Tarea1
[2013-04-27 08:07:48] process terminated successfully (elapsed time: 00.10s)
```

En otros lenguajes cuando se crea un hilo, se le debe asociar el nombre de una función, que es la función que el hilo ejecutará. De forma que si se desea lanzar 10 hilos que ejecutan la misma función, el código de la función sólo se escribe una sola vez. En Ada la forma de hacer esto es declarar un tipo especial, el tipo tarea. Luego se puede declarar un arreglo de 10 elementos, donde cada uno de ellos se convertirá en una tarea.

## Elementos de sincronización

Emplearemos objetos protegidos y *rendezvous* para sincronizar la comunicación entre dos tareas.

**Objetos protegidos**, la idea básica es proteger un *objeto* (entendido como alguna estructura o conjunto de estructuras: variables, arreglos, etc) de forma que el lenguaje garantiza que los *objetos* pueden ser accedidos con exclusión mutua. Desde el punto de vista estructural tiene la forma de un *monitor* pero en lugar de emplear variables de condición emplean *guardas* que Ada llama *barreras*.

Para acceder al objeto se puede emplear: procedimientos, funciones o entradas. En cualquier caso se le acostumbra a llamar procedimientos protegidos, funciones protegidas o entradas protegidas. Los procedimientos pueden acceder en modo de lectura/escritura al *objeto*, en caso de las funciones sólo pueden acceder en modo de lectura. Por eso Ada permite que muchas funciones puedan ejecutarse concurrentemente. Pero ofrece exclusión mutua entre procedimientos y entre funciones y procedimientos.

Las entradas tienen parámetros como los procedimientos (es decir se pueden considerar como lectura / escritura) sin embargo poseen una característica adicional y es que se les puede añadir una *guarda* también llamada *barrera*. Cuando la barrera se evalúa a *falso* la tarea que la invocó se bloquea hasta que la condición se evalúe a *cierto*. Esto proporciona un potente mecanismo de sincronización.

Un tema interesante es cómo se tratan las prioridades en el caso de los accesos. Por ejemplo suponga que una tarea A está accediendo a una entrada E1, y las tareas B, C y D están bloqueadas en la entrada E2. Además la tarea F y G solicitan acceder a otra entrada. En este escenario, puesto que A está accediendo a E1 no se le interrumpe y se espera a que termine. Luego tienen prioridad las tareas que están bloqueadas haciendo cola y que se las trata como un FIFO. Por último serán atendidas el resto de tareas.

Una unidad protegida puede declararse como un tipo o como una simple instancia, pero recuerde que si la declara como un tipo debe de declarar una instancia de dicho tipo. La unidad protegida tiene dos partes la declaración y su definición. En la declaración los *objetos* a proteger van precedido de la palabra reservada *private* dentro de la unidad protegida. En la definición se desarrollan los cuerpos de los procedimientos, funciones o entradas protegidas que contienen cada unidad protegida.

Dentro de una declaración de unidad protegida no pueden haber definición de tipos, por lo que si se van a emplear tipos definidos por el usuario, estas deben ser definidos previamente.

A continuación un ejemplo de una unidad protegida declarada como un tipo:

```
protected type entero_Compartido(Valor_Inicial: Integer) is
    function Lee return Integer;
    procedure Escribe(Nuevo_Valor: Integer);
    procedure Incrementa(Por : Integer);
private
    Dato: Integer := Valor_Inicial;
end Entero_Compartido;

Mi_Dato : Entero_Compartido(42);

protected body Entero_Compartido is
    function Lee return Integer is
    begin
        ...
    end Lee;
```

```

procedure Escribe(Nuevo_Valor: Integer) is
  begin
    ...
  end Escribe;

procedure Incrementa(Por : Integer) is
  begin
    ...
  end Incrementa;
end Entero_Compartido

```

Luego desde alguna tarea se puede invocar: `Mi_Dato.Incrementa(3)`.

Ahora un ejemplo de una simple instancia:

```

type Index is mod 8;
type Buffer_Array is array(Index) of Integer;

protected Buffer is
  entry Append(I: in Integer);
  entry Take(I: out Integer);
private
  B: Buffer_Array;
  In_Ptr, Out_Ptr, Count: Index := 0;
end Buffer;

protected body Buffer is
  entry Append(I: in Integer) when Count < Index'Last is
    begin
      . . .
    end Append;

  entry Take(I: out Integer) when Count > 0 is
    begin
      . . .
    end Take;
end Buffer;

```

Una tarea cualquiera podrá invocar la entrada de la siguiente forma: `Buffer.Append(6)`.

Hay mucha información acerca de objetos protegidos en Ada, se le recomienda leer:

<http://www.iuma.ulpgc.es/users/jmiranda/gnat-rts/node25.htm>

Ada For Software Engineers 2da Ed. Ben Ari Cap 18 (Material de lectura para el curso)

**Rendezvous**, la idea de cita es muy sencilla. En Ada una cita se da entre dos tareas, si una llega primero debe esperar a la otra y viceversa. Una tarea puede recibir la petición de una cita (petición de entrada) y el procedimiento la debe esperar en un punto definido por la instrucción **accept**. Si la tarea que tiene el **accept** llega primero, entonces esta tarea se detiene en ese punto, hasta que otra tarea solicita la petición de activación. Y de forma análoga, si la tarea que hace la petición de entrada llega primero, entonces espera hasta que la tarea correspondiente llegue a su **accept**. Una vez concertada la cita la tarea que contiene el **accept** se ejecuta y la tarea que hizo la invocación se detiene hasta que termine la cita. Si una tarea desea concertar un cita, debe tener en su cuerpo definido un **accept** y además en la parte de declaración de la tarea se debe especificar las entradas (citas) que espera recibir.

Puede ser que una tarea haya concertado muchas citas, pero debido a que las ejecuciones de las tareas no son sincronizadas, estas pueden ocurrir en cualquier momento. Por ejemplo una tarea tiene 2 cuerpos de **accept**, si la primera cita no ha sido concertada se quedará esperando, a pesar de que de repente la segunda cita ya ha sido solicitada y podría ser ya concertada. En ese caso se puede usar la sentencia **select**.

Se le deja como tarea investigar acerca de la sentencia **select** para el caso de *rendezvous* en Ada.

Para una información detallada puede leer:

<http://www.iuma.ulpgc.es/users/jmiranda/gnat-rtts/node21.htm>

Ada For Software Engineers 2da Ed. Ben Ari Cap 18 (Material de lectura para el curso)

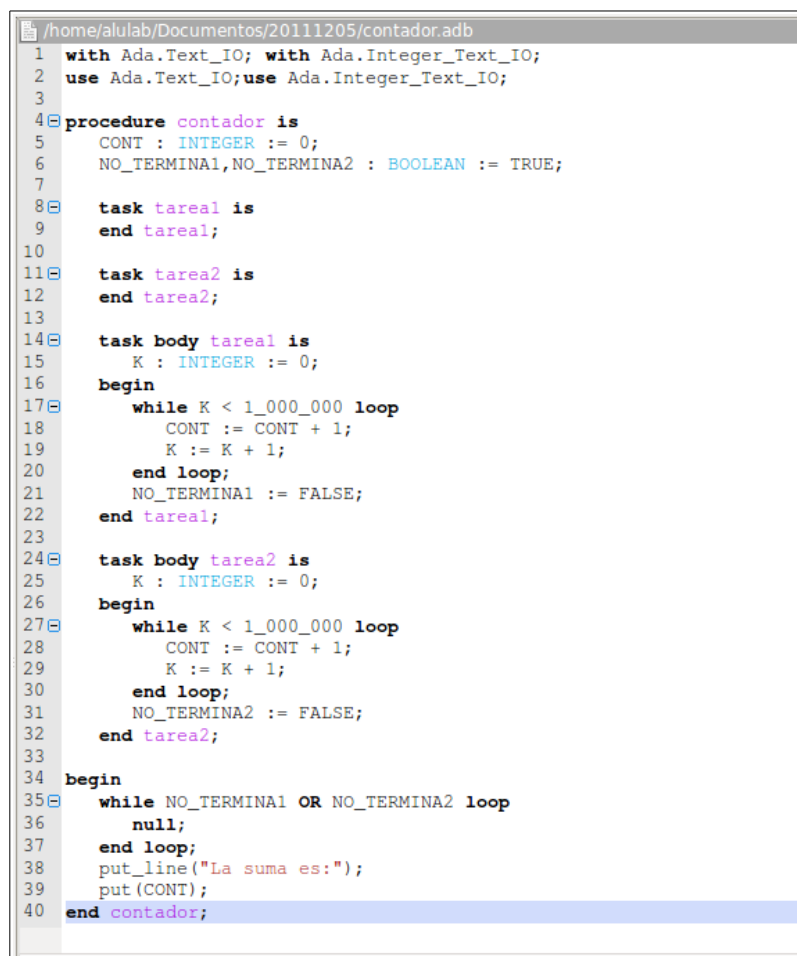
Para aclarar las ideas se presenta la solución del laboratorio del semestre 2012-2.

**PONTIFICIA UNIVERSIDAD CATOLICA DEL PERU  
FACULTAD DE CIENCIAS E INGENIERIA**

**Laboratorio Nro 2  
( 2012 - 2 )**

**1.- (3 puntos) Protegiendo el recurso compartido (archivo a entregar: *contador.adb*)**

Se tiene el siguiente programa:



```
/home/alulab/Documentos/20111205/contador.adb
1 with Ada.Text_IO; with Ada.Integer_Text_IO;
2 use Ada.Text_IO; use Ada.Integer_Text_IO;
3
4 procedure contador is
5   CONT : INTEGER := 0;
6   NO_TERMINA1, NO_TERMINA2 : BOOLEAN := TRUE;
7
8   task tarea1 is
9     end tarea1;
10
11  task tarea2 is
12    end tarea2;
13
14  task body tarea1 is
15    K : INTEGER := 0;
16    begin
17      while K < 1_000_000 loop
18        CONT := CONT + 1;
19        K := K + 1;
20      end loop;
21      NO_TERMINA1 := FALSE;
22    end tarea1;
23
24  task body tarea2 is
25    K : INTEGER := 0;
26    begin
27      while K < 1_000_000 loop
28        CONT := CONT + 1;
29        K := K + 1;
30      end loop;
31      NO_TERMINA2 := FALSE;
32    end tarea2;
33
34  begin
35    while NO_TERMINA1 OR NO_TERMINA2 loop
36      null;
37    end loop;
38    put_line("La suma es:");
39    put(CONT);
40  end contador;
```



Al ejecutarlo varias veces el resultado no siempre es el esperado.

<pre>Run: contador /home/alulab/Documentos/20111205/contador La suma es: 1211160 [2012-10-03 14:17:45] process terminated successfully (elapsed time: 00.10s)</pre>	<pre>Run: contador /home/alulab/Documentos/20111205/contador La suma es: 1731078 [2012-10-03 14:18:12] process terminated successfully (elapsed time: 00.10s)</pre>
<pre>Run: contador /home/alulab/Documentos/20111205/contador La suma es: 2000000 [2012-10-03 14:10:28] process terminated successfully (elapsed time: 00.10s)</pre>	<pre>Run: contador /home/alulab/Documentos/20111205/contador La suma es: 1502783 [2012-10-03 14:18:35] process terminated successfully (elapsed time: 00.10s)</pre>

En el programa, a modo de comentario, explique las razones de las distintos valores. Modifique el programa para que la respuesta sea siempre la correcta.

### Solución:

No hay mucho que comentar. Los objetos protegidos en Ada son parecidos al concepto de monitores excepto que su comportamiento varía si es un procedimiento, una función o una entrada. Esto está explicado en la documentación que se dejó para leer. El primer ejemplo en el material emplea entradas, pero después en otro ejemplo se emplea un procedimiento. Sin más comentario la solución:

```
/home/alejandro/SourceCode/Ada/2012-2 Lab 2 SO/solcontador.adb
1 with Ada.Text_IO; use Ada.Text_IO;
2 with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
3
4 procedure solContador is
5     NO_TERMINAL, NO_TERMINA2 : BOOLEAN := TRUE;
6
7     protected Contador is
8         function Lee return Integer;
9         procedure Incrementa;
10    private
11        Cont : INTEGER := 0;
12    end Contador;
13
14    protected body Contador is
15        procedure Incrementa is
16        begin
17            Cont := Cont + 1;
18        end Incrementa;
19        function Lee return Integer is
20        begin
21            return Cont;
22        end Lee;
23    end Contador;
24
25    task tarea1 is
26    end tarea1;
27
28    task tarea2 is
29    end tarea2;
30
31    task body tarea1 is
32        K : INTEGER := 0;
33    begin
34        while K < 1_000_000 loop
35            Contador.Incrementa;
36            K := K + 1;
37        end loop;
38        NO_TERMINAL := FALSE;
39    end tarea1;
40
41    task body tarea2 is
42        K : INTEGER := 0;
43    begin
44        while K < 1_000_000 loop
45            Contador.Incrementa;
46            K := K + 1;
47        end loop;
48        NO_TERMINA2 := FALSE;
49    end tarea2;
50
51    begin
52        while NO_TERMINAL OR NO_TERMINA2 loop
53            null;
54        end loop;
55        put_line("La suma es:");
56        put(Contador.Lee);
57    end solContador;
58
59 tarea1
```

A continuación diferentes ejecuciones:

```
Run: solcontador
/home/alulab/Documentos/ProgramasSO/solcontador
La suma es:
2000000
[2012-10-23 18:08:51] process terminated successfully (elapsed time: 00.43s)
```

```
Run: solcontador
/home/alulab/Documentos/ProgramasSO/solcontador
La suma es:
2000000
[2012-10-23 18:09:54] process terminated successfully (elapsed time: 00.57s)
```

```
Run: solcontador
/home/alulab/Documentos/ProgramasSO/solcontador
La suma es:
2000000
[2012-10-23 18:11:39] process terminated successfully (elapsed time: 00.48s)
```

## 2.- (8 puntos) Sincronizando procesos (archivo a entregar: *dac.adb*)

Se tiene el siguiente código en Ada

```
/home/alulab/Documentos/20111205/dac.adb
1  with Ada.Text_IO; with Ada.Integer_Text_IO;
2  use Ada.Text_IO; use Ada.Integer_Text_IO;
3
4  procedure dac is
5      CONT : INTEGER := 0;
6
7      task tarea1;
8      task tarea2;
9      task tarea3;
10     task tarea4;
11     task tarea5;
12
13     task body tarea1 is
14         K : INTEGER := 0;
15         begin
16             for K in Integer range 1..10 loop
17                 PUT("Tarea1");
18             end loop;
19         end tarea1;
20
21     task body tarea2 is
22         K : INTEGER := 0;
23         begin
24             for K in Integer range 1..10 loop
25                 PUT("Tarea2");
26             end loop;
27         end tarea2;
28
29     task body tarea3 is
30         K : INTEGER := 0;
31         begin
32             for K in Integer range 1..10 loop
33                 PUT("Tarea3");
34             end loop;
35         end tarea3;
36
37     task body tarea4 is
38         K : INTEGER := 0;
39         begin
40             for K in Integer range 1..10 loop
41                 PUT("Tarea4");
42             end loop;
43         end tarea4;
44
45     task body tarea5 is
46         K : INTEGER := 0;
47         begin
48             for K in Integer range 1..10 loop
49                 PUT("Tarea5");
50             end loop;
51         end tarea5;
52
53     begin
54         null;
55     end dac;
```

Que al ejecutarse varias veces, se obtienen las siguientes salidas:

```
Run: dac
/home/alulab/Documentos/20111205/dac
Tarea5Tarea5Tarea5Tarea5Tarea5Tarea5Tarea5Tarea5Tarea5Tarea4Tarea4Tarea4Tarea4Tarea4Tarea4Tarea4
Tarea4Tarea4Tarea4Tarea3Tarea3Tarea3Tarea3Tarea3Tarea3Tarea3Tarea3Tarea2Tarea2Tarea2Tarea2
Tarea2Tarea2Tarea2Tarea2Tarea2Tarea2Tarea1Tarea1Tarea1Tarea1Tarea1Tarea1Tarea1Tarea1Tarea1
[2012-10-03 14:37:31] process terminated successfully (elapsed time: 00.10s)
```

```
Run: dac
/home/alulab/Documentos/20111205/dac
Tarea1Tarea1Tarea1Tarea1Tarea1Tarea1Tarea1Tarea1Tarea1Tarea5Tarea5Tarea5Tarea5Tarea5Tarea5Tarea5
Tarea5Tarea5Tarea5Tarea4Tarea4Tarea4Tarea4Tarea4Tarea4Tarea4Tarea4Tarea3Tarea3Tarea3Tarea3
Tarea3Tarea3Tarea3Tarea3Tarea3Tarea3Tarea2Tarea2Tarea2Tarea2Tarea2Tarea2Tarea2Tarea2Tarea2
[2012-10-03 14:39:06] process terminated successfully (elapsed time: 00.10s)
```

```
Run: dac
/home/alulab/Documentos/20111205/dac
Tarea1Tarea1Tarea1Tarea1Tarea1Tarea1Tarea1Tarea1Tarea1Tarea3Tarea3Tarea3Tarea3Tarea3Tarea3Tarea3
Tarea3Tarea3Tarea5Tarea4Tarea4Tarea4Tarea4Tarea3Tarea5Tarea5Tarea4Tarea2Tarea2Tarea2Tarea2Tarea4Tarea4Tarea4
Tarea4Tarea4Tarea4Tarea2Tarea2Tarea2Tarea2Tarea2Tarea2Tarea5Tarea5Tarea5Tarea5Tarea5Tarea5Tarea5
[2012-10-03 14:41:30] process terminated successfully (elapsed time: 00.11s)
```

Haciendo uso de *rendezvous* modifique el programa para que siempre que se ejecute, cumpla los siguientes requisitos:

- La tarea1 debe terminar antes que, la tarea2 y la tarea3 empiecen.
- La tarea2 debe terminar antes que la tarea4 empiece.
- La tarea5 debe empezar después que la tarea2, y la tarea3 terminen.

Esta secuencia se debe reflejar en la impresión de los mensajes y en todas las ejecuciones.

### Solución:

```
/home/alulab/Documentos/ProgramasSO/soldac.adb
1 with Ada.Text_IO; with Ada.Integer_Text_IO;
2 use Ada.Text_IO; use Ada.Integer_Text_IO;
3
4 procedure soldac is
5   CONT : INTEGER := 0;
6
7   task tarea1;
8   task tarea2 is
9     entry uno_dos;
10  end tarea2;
11
12  task tarea3 is
13    entry uno_tres;
14  end tarea3;
15
16  task tarea4 is
17    entry dos_cuatro;
18  end tarea4;
19
20  task tarea5 is
21    entry dos_cinco;
22    entry tres_cinco;
23  end tarea5;
24
25  task body tarea1 is
26    K : INTEGER := 0;
27  begin
28    for K in Integer range 1..10 loop
29      PUT("Tarea1");
30    end loop;
31    tarea2.uno_dos;
32    tarea3.uno_tres;
33  end tarea1;
34
35  task body tarea2 is
36    K : INTEGER := 0;
37  begin
38    accept uno_dos do
39      null;
40    end;
41    for K in Integer range 1..10 loop
42      PUT("Tarea2");
43    end loop;
44    tarea4.dos_cuatro;
45    tarea5.dos_cinco;
46  end tarea2;
```

```

47
48 task body tarea3 is
49     K : INTEGER := 0;
50 begin
51     accept uno_tres do
52         null;
53     end;
54     for K in Integer range 1..10 loop
55         PUT("Tarea3");
56     end loop;
57     tarea5.tres_cinco;
58 end tarea3;
59
60 task body tarea4 is
61     K : INTEGER := 0;
62 begin
63     accept dos_cuatro do
64         null;
65     end;
66     for K in Integer range 1..10 loop
67         PUT("Tarea4");
68     end loop;
69 end tarea4;
70
71 task body tarea5 is
72     K : INTEGER := 0;
73 begin
74     accept dos_cinco do
75         null;
76     end;
77     accept tres_cinco do
78         null;
79     end;
80     for K in Integer range 1..10 loop
81         PUT("Tarea5");
82     end loop;
83 end tarea5;
84
85 begin
86     null;
87 end soldac;

```

La solución es muy sencilla, si asumimos que la aceptación de una entrada es como un *receive* y que la invocación de esta entrada es como un *send*, entonces sólo jugamos con un *send/receive*.

Observe que las salidas cumplen con los requisitos. Tenga cuidado en interpretar las exigencias, por ejemplo, decir que la tarea1 debe terminar antes que, la tarea2 y la tarea3 empiecen, no significa que la tarea3 inicie inmediatamente después de la tarea2, podría suceder que se ejecute la tarea1, luego la tarea2, la tarea4 y la tarea3, esto sigue cumpliendo lo solicitado. Es decir la tarea2 y la tarea3 se ejecutan después de la tarea1, pero en cualquier orden.

Algunas salidas:

[illegible][illegible][illegible]

**3.- (9 puntos) Cigarette smokers problem (archivo a entregar: fumadores.adb)**

The cigarette smokers problem was originally presented by Suhas Patil, who claimed that it cannot be solved with semaphores. That claim comes with some qualifications, but in any case the problem is interesting and challenging.

Four threads are involved: an agent and three smokers. The smokers loop forever, first waiting for ingredients, then making and smoking cigarettes. The ingredients are tobacco, paper, and matches.

We assume that the agent has an infinite supply of all three ingredients, and each smoker has an infinite supply of one of the ingredients; that is, one smoker has matches, another has paper, and the third has tobacco.

The agent repeatedly chooses two different ingredients at random and makes them available to the smokers. Depending on which ingredients are chosen, the smoker with the complementary ingredient should pick up both resources and proceed.

For example, if the agent puts out tobacco and paper, the smoker with the matches should pick up both ingredients, make a cigarette, and then signal the agent.

To explain the premise, the agent represents an operating system that allocates resources, and the smokers represent applications that need resources. The problem is to make sure that if resources are available that would allow one more applications to proceed, those applications should be woken up. Conversely, we want to avoid waking an application if it cannot proceed.

Based on this premise, there are three versions of this problem that often appear in textbooks:

**The impossible version:** Patil's version imposes restrictions on the solution. First, you are not allowed to modify the agent code. If the agent represents an operating system, it makes sense to assume that you don't want to modify it every time a new application comes along. The second restriction is that you can't use conditional statements or an array of semaphores. With these constraints, the problem cannot be solved, but as Parnas points out, the second restriction is pretty artificial. With constraints like these, a lot of problems become unsolvable.

**The interesting version:** This version keeps the first restriction—you can't change the agent code—but it drops the others.

**The trivial version:** In some textbooks, the problem specifies that the agent should signal the smoker that should go next, according to the ingredients that are available. This version of the problem is uninteresting because it makes the whole premise, the ingredients and the cigarettes, irrelevant. Also, as a practical matter, it is probably not a good idea to require the agent to know about the other threads and what they are waiting for. Finally, this version of the problem is just too easy.

***Elabore un programa en Ada que simule una solución para el problema de los fumadores de cigarros.***

**Solución:**

He adaptado al lenguaje Ada, la solución propuesta por *Allen Downey* en su libro *The Little Book of Semaphores*.

Simular un *mutex* en Ada no es problema, por que si se desea exclusión mutua esta se logra definiendo un objeto protegido. El problema es como sustituir *wait/signal* de los semáforos. He empleado los *rendezvous* de Ada, que no es lo mismo, por que ya sea que se acepte o que se invoque una entrada, el que llegue primero quedará bloqueado hasta que llegue el otro. Esto es diferente a un *wait/signal*. Sin embargo en el diseño del código no había diferencia por que el *signal* era la última instrucción.

El compilador emite unas advertencias (*warnings*) por que encuentra invocaciones a entradas (de hecho habrá bloqueos hasta que se concierte una cita) dentro de un objeto protegido. Pero tal como esta diseñado el programa no sucederá un interbloqueo.

El programa está escrito para bucles infinitos por lo que la ejecución se ha hecho en una terminal redireccionando la salida hacia un archivo y luego mostrando parte del archivo.

A continuación el código:

```

/home/alulab/Documentos/ProgramasSO/fumadores.adb
1 with Ada.Text_IO; with Ada.Integer_Text_IO;
2 use Ada.Text_IO; use Ada.Integer_Text_IO;
3 with Ada.Numerics.Discrete_Random;
4
5 procedure fumadores is
6
7   task agente is
8     entry agenteSem;
9   end;
10  task tiene_tabaco is
11    entry tabacoSem;
12  end;
13  task tiene_cerillos is
14    entry cerillosSem;
15  end;
16  task tiene_papel is
17    entry papelSem;
18  end;
19
20  task body agente is
21    subtype Procesos is Integer range 1..3;
22    package Random_Proc is new Ada.Numerics.Discrete_Random(Procesos);
23    use Random_Proc;
24    G:Generator;
25    D:Procesos;
26
27    hayPapel, hayCerillos, hayTabaco: BOOLEAN := False;
28  protected colocaEnMesa is
29    procedure Tabaco;
30    procedure Cerillos;
31    procedure Papel;
32  end;
33
34  protected body colocaEnMesa is
35    procedure Tabaco is
36      begin
37        put_line("colocando ---> Tabaco");
38        if hayPapel then
39          hayPapel := False;
40          tiene_cerillos.cerillosSem;
41        elsif hayCerillos then
42          hayCerillos := False;
43          tiene_papel.papelSem;
44        else
45          hayTabaco := True;
46        end if;
47      end Tabaco;

```



```

48
49 procedure Cerillos is
50 begin
51   put_line("colocando ---> Cerillos");
52   if hayPapel then
53     hayPapel := False;
54     tiene_tabaco.tabacoSem;
55   elsif hayTabaco then
56     hayTabaco := False;
57     tiene_papel.papelSem;
58   else
59     hayCerillos := True;
60   end if;
61 end Cerillos;
62
63 procedure Papel is
64 begin
65   put_line("colocando ---> Papel");
66   if hayTabaco then
67     hayTabaco := False;
68     tiene_cerillos.cerillosSem;
69   elsif hayCerillos then
70     hayCerillos := False;
71     tiene_tabaco.tabacoSem;
72   else
73     hayPapel := True;
74   end if;
75 end Papel;
76 end colocaEnMesa;
77
78 begin
79   Reset(G);
80   loop
81     D := Random(G);
82     accept agenteSem do
83       null;
84     end;
85     case D is
86       when 1 => colocaEnMesa.Cerillos;
87       when 2 => colocaEnMesa.Papel;
88       when 3 => colocaEnMesa.Tabaco;
89       when 3 => colocaEnMesa.Papel;
90       when 3 => colocaEnMesa.Tabaco;
91       when 3 => colocaEnMesa.Cerillos;
92     end case;
93   end loop;
94 end agente;
95
96 task body tiene_tabaco is
97 begin
98   loop
99     accept tabacoSem do
100       null;
101     end;
102     put_line("Tengo Tabaco => Armando cigarro");
103     agente.agenteSem;
104     put_line("Uff buen cigarro:Tenia tabaco");
105     put_Line("");
106   end loop;
107 end tiene_tabaco;
108
109 task body tiene_cerillos is
110 begin
111   loop
112     accept cerillosSem do
113       null;
114     end;
115     put_line("Tengo Cerillos => Armando cigarro");
116     agente.agenteSem;
117     put_line("Uff buen cigarro:Tenia cerillos");
118     put_Line("");
119   end loop;
120 end tiene_cerillos;
121
122 task body tiene_papel is
123 begin
124   loop
125     accept papelSem do
126       null;
127     end;
128     put_line("Tengo Papel => Armando cigarro");
129     agente.agenteSem;
130     put_line("Uff buen cigarro:Tenia papel");
131     put_Line("");
132   end loop;
133 end tiene_papel;
134
135 begin
136   agente.agenteSem;
137 end fumadores;

```

Algunos fragmentos de la salida:

```
alulab@lab-PC: ~/Documentos/ProgramasSO
colocando ---> Tabaco
colocando ---> Cerillos
        Tengo Papel => Armando cigarro
Uff buen cigarro:Tenia papel

colocando ---> Cerillos
colocando ---> Papel
        Tengo Tabaco => Armando cigarro
Uff buen cigarro:Tenia tabaco

colocando ---> Tabaco
colocando ---> Papel
        Tengo Cerillos => Armando cigarro
Uff buen cigarro:Tenia cerillos

colocando ---> Tabaco
colocando ---> Papel
        Tengo Cerillos => Armando cigarro
Uff buen cigarro:Tenia cerillos

colocando ---> Tabaco
colocando ---> Cerillos
        Tengo Papel => Armando cigarro
Uff buen cigarro:Tenia papel

colocando ---> Tabaco
colocando ---> Cerillos
        Tengo Papel => Armando cigarro
:
```

*Pando, 03 de octubre de 2012.*

*Prof: Alejandro T. Bello Ruiz.*

A continuación como ejercicio para el laboratorio calificado se le propone solucionar el laboratorio del semestre 2013-1.

**PONTIFICIA UNIVERSIDAD CATOLICA DEL PERU  
FACULTAD DE CIENCIAS E INGENIERIA**

**Laboratorio de Sistemas Operativos**

**EXCLUSIÓN MUTUA  
Y  
SINCRONIZACIÓN DE PROCESOS**

**1.- (4 + 2 = 6 puntos) Los semáforos de Dijkstra**

En 1965 E.W. Dijkstra propuso un nuevo tipo de variable, denominado semáforo. El objetivo era contar el número de señales que sirven para despertar un proceso, de forma que no se pierdan sino que sean guardadas para un uso futuro. Un semáforo podría tener el valor cero, indicando que no se guardaron señales de despertar o algún valor positivo que indica que están pendientes una o más señales de despertar. Para completar la idea se añadieron dos operaciones, la primera la denominó P (Proberen) comprueba si el valor es mayor que 0. De ser así disminuye en uno la variable y sólo continúa. Si el valor es 0 el proceso se bloquea sin completar la operación P por el momento. Estas acciones son atómicas. La segunda operación la denominó V (Verhogen), que incrementa la variable en uno, y si hay un proceso bloqueado lo despierta. En caso de que hubiesen varios procesos bloqueados se despierta cualquiera. Escriba un programa en lenguaje Ada que implemente las operaciones V y P para simular los semáforos. Para probar sus semáforos declare 5 tareas que modifican una variable global con un valor alto de forma que la suma total en diferentes ejecuciones no coincida con el valor esperado. Luego usando su implementación de semáforos debería obtener el resultado correcto.



**2.- (8 puntos) El puente Dueñas**



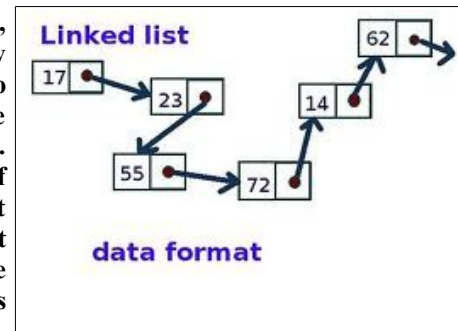
Como es de conocimiento público hace algunas semanas el puente “Dueñas” que se encuentra en la avenida Universitaria ha colapsado, construir un puente llevará demasiado tiempo. Una empresa formada por ingenieros ex alumnos de la PUCP propone una solución temporal. La idea es muy simple, y la han denominado “UnoxUno”, aprovechando que una de las vías del puente está en perfectas condiciones, y para no debilitar más la estructura se ha pensado en usar un carril, en un sentido a la vez. Si un auto viene de Este a Oeste y no hay ningún carro en el otro extremo, entonces puede pasar, pero si ya hay un carro que viene en sentido contrario debe esperar a que pase para

poder cruzar el puente. No pueden pasar dos carros en el mismo sentido a excepción de que no haya nadie en el otro extremo, la idea es que si hay muchos carros en ambos extremos, el paso de los vehículos se debe alternar estrictamente uno a uno hasta que no haya más carros.

Escriba un programa en Ada que simule esta situación, emplee 15 tareas. Cada tarea tomará de forma aleatoria un número entre 0 y 1. Si sale 0, significa que recorrerá el puente de Este a Oeste, y si es 1, significa que lo recorrerá en sentido contrario. Cada tarea debe imprimir “Tarea X inicia cruce” y después de un tiempo “Tarea X termina el cruce” (donde X toma desde 1 hasta 15) de forma que no puede haber mensajes intermedios de otras tareas. **Las tareas se deben ejecutar concurrentemente, no las ejecute secuencialmente**, usted las debe sincronizar para que se cumpla la alternancia estricta.

**3.- (6 puntos) The search-insert-delete problem** (*The Little Book of Semaphores* by Allen B. Downey)

Three kinds of threads share access to a singly-linked list: searchers, inserters and deleters. Searchers merely examine the list; hence they can execute concurrently with each other. Inserters add new items to the end of the list; insertions must be mutually exclusive to preclude two inserters from inserting new items at about the same time. However, one insert can proceed in parallel with any number of searches. Finally, deleters remove items from any where in the list. At most one deleter process can access the list at a time, and deletion must also be mutually exclusive with searches and insertions. Write code (language Ada) for searchers, inserters and deleters that enforces this kind of three-way categorical mutual exclusion.



Pando, 30 de abril de 2013.

*Prof. Alejandro T. Bello Ruiz*