



## PROYECTO HADA

### 1. OBJETIVO

Poner en práctica los conocimientos adquiridos en la asignatura, en particular algunas de las ideas de programación que derivan de los análisis léxicos, sintácticos y semánticos de los lexemas que conforman las hileras de caracteres que se usan, como código fuente, como entrada del programa.

### 2. LINEAMIENTOS

HADA es una simulación básica, ficticia, del lenguaje de programación ADA y como tal es un poco más rudimentario, careciendo de muchas de las características de su hermano mayor. Debe tenerse claro que como es una simulación si se va al Google a buscar información sobre HADA pues no se encontrará nada al respecto.

#### 2.1 PAUTAS

El proyecto es individual.

Debe ser hecho en Java. No puede usarse otro lenguaje.

El entorno de desarrollo debe ser Netbeans, oficial de la universidad.

Debe ser programado en modo carácter.

No se puede usar el modo gráfico.

#### 2.2 ENLACES

Se recomienda conocer un poco de ADA, para lo cual es importante que ingrese a los siguientes enlaces:

##### 2.2.1 HISTORIA

Un muy buen artículo de hace 40 años que no obstante explica claramente el génesis de este lenguaje; interesante establecer un paralelo, mientras se lee, con el nacimiento de Java:

<https://revistamarina.cl/revistas/1985/6/hernandez.pdf>

Una mirada a la mujer que dio su nombre al lenguaje:

<https://www.fablabsantcugat.com/blog/ada-lovelace-la-mujer-que-revolucion-la-informtica>



Los siguientes enlaces le ayudarán a conocer el lenguaje de ADA:

<https://ocw.unican.es/course/view.php?id=185&section=4>  
(en especial el apartado MC-F-002)

[http://www.gedlc.ulpgc.es/docencia/mp\\_i/GuiaAda/index.html](http://www.gedlc.ulpgc.es/docencia/mp_i/GuiaAda/index.html)

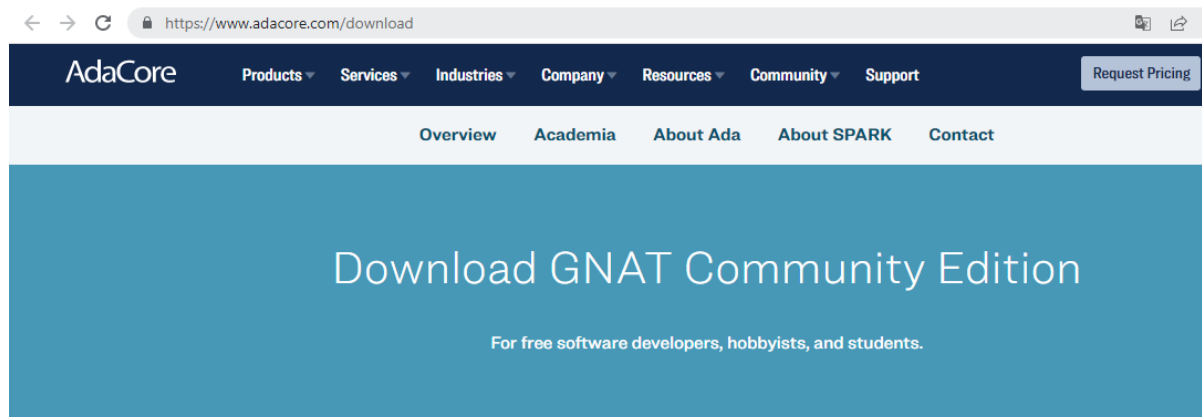
### 2.2.2 GNAT ADA

El compilador que se usará lo puede descargar en la siguiente dirección:

<https://www.adacore.com/download>

### 2.2.3 DESCARGA

De esa misma página, usando el enlace en el recuadro en rojo, puede descargar el ambiente de desarrollo, así como el compilador, según se muestra en la siguiente imagen:



x86 Windows (64 bits)

GNAT Community

[README.txt](#)

7.2 KiB

May 27 2021

SHA-1: 8c50fa137de8a288f829d37c613b0dd2d33b81e7

[gnat-2021-20210519-x86\\_64-windows64-bin.exe](#)

562.4 MiB

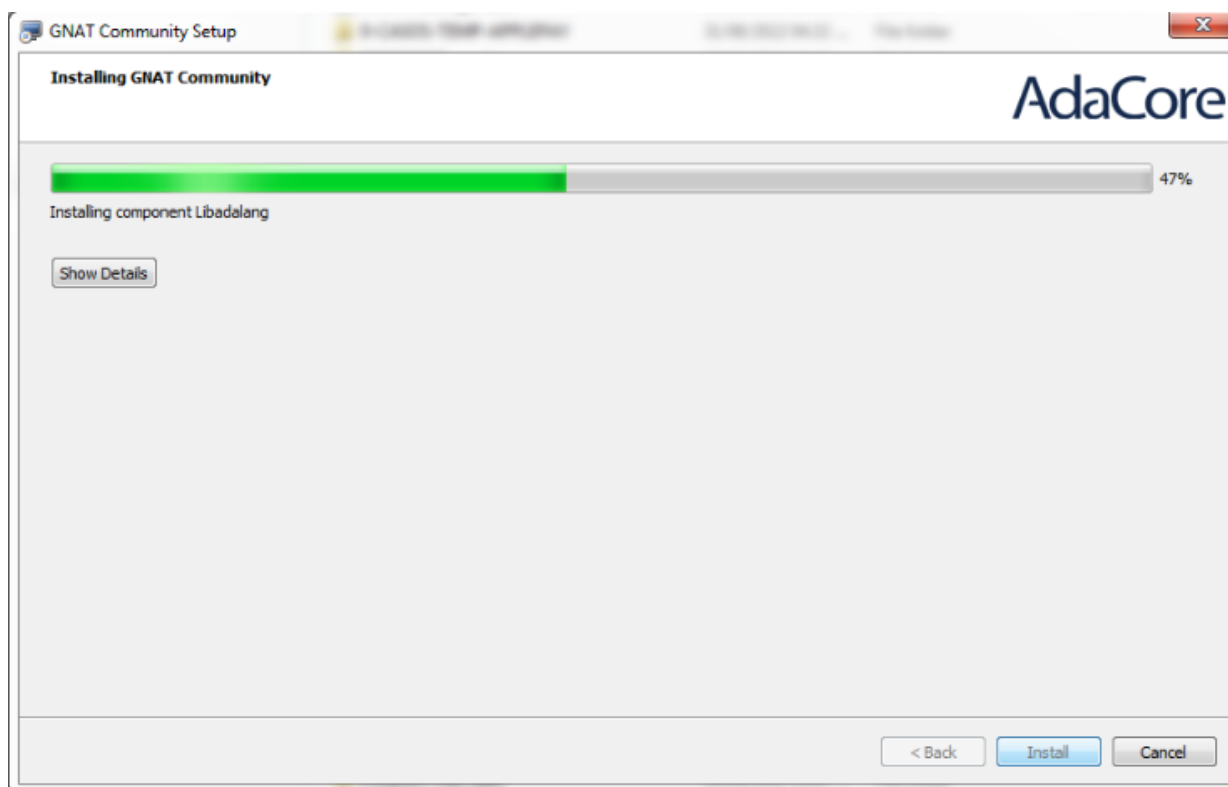
May 20 2021

SHA-1: 797dbae8bdb8a3f661dad78dd73d8e40218a68d8

Cuando se presiona el recuadro rojo se descarga el siguiente archivo de instalación:

**gnat-2021-20210519-x86\_64-windows64-bin.exe**

Se le da doble click y se inicia la instalación; deben seguirse las instrucciones del caso y se muestran pantallas como la siguiente:





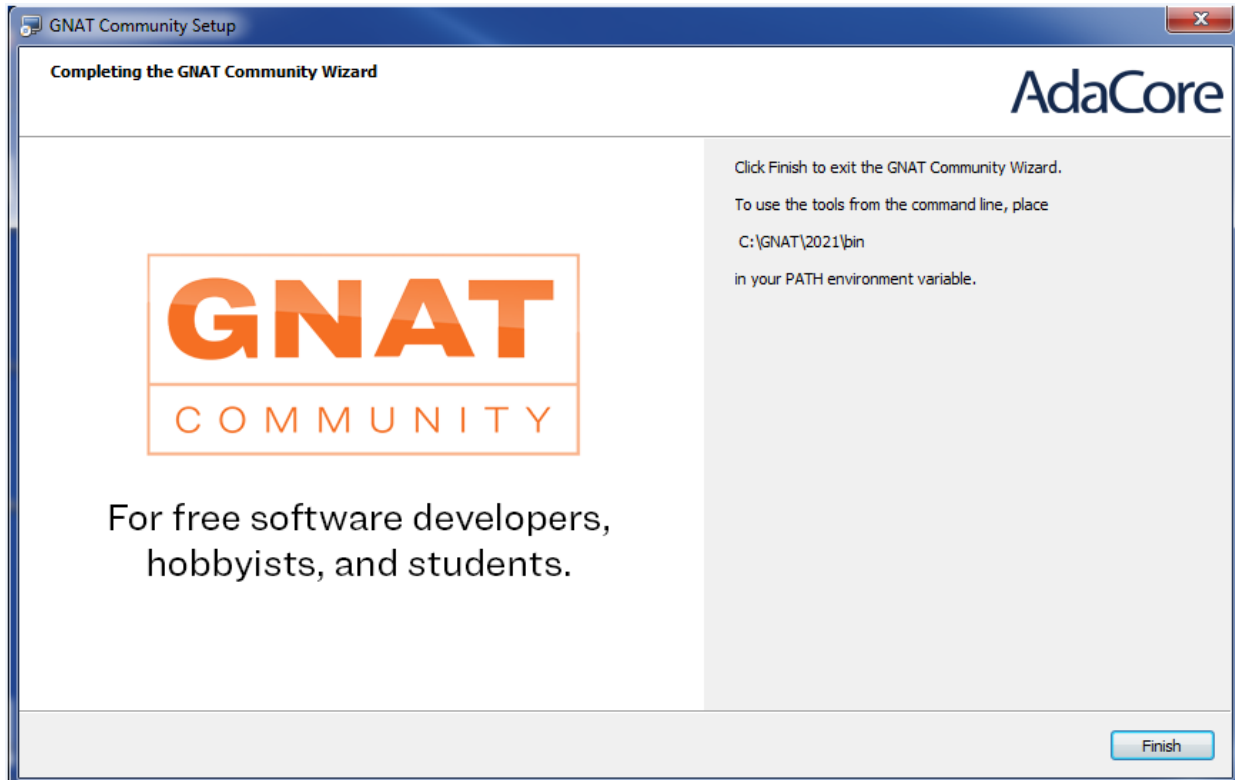
UNIVERSIDAD ESTATAL A DISTANCIA  
ESCUELA DE CIENCIAS EXACTAS Y NATURALES  
CARRERA INGENIERÍA INFORMÁTICA  
CATEDRA TECNOLOGÍA DE SISTEMAS



Compiladores

Código: 03307

Cuando termina la instalación, vemos la siguiente ventana:



Se presiona el botón **Finish** y con eso ya se tiene el entorno de programación instalado (que no es necesario usar) típicamente en la siguiente carpeta:

**C:\GNAT\2021**

así como el compilador en línea llamado **gnatmake** que sí vamos a usar en el CMD.

El compilador es el archivo llamado **gnatmake.exe** y se encuentra en la carpeta C:\GNAT\2021\bin

Para probar ese compilador podemos hacer un archivo de texto que lleve este nombre:

Hola\_Mundo.adb

y que tenga este pequeño código fuente:

```
with Ada.Text_IO;  
  
procedure Hola_Mundo is  
begin  
  Ada.Text_IO.Put_Line ( "Hola, mundo!" );  
end Hola_Mundo;
```



Lo echamos en alguna carpeta, por ejemplo en: C:\PRUEBA

Y realizamos los siguientes pasos para compilarlo (se genera el archivo con extensión .O que quiere decir código intermedio) y ejecutarlo (se usa el archivo con extensión .EXE) en el CMD.

Con eso se confirma que el compilador de GNAT está efectivamente instalado:

```
C:\Windows\System32\cmd.exe

C:\PRUEBA>DIR
Volume in drive C has no label.
Volume Serial Number is FC8F-1FCC

Directory of C:\PRUEBA

01/09/2022  08:41 p.m.      <DIR>          .
01/09/2022  08:41 p.m.      <DIR>          ..
31/08/2022  06:42 p.m.                  113 Hola_Mundo.adb
                                1 File(s)          113 bytes
                                2 Dir(s)  389.846.142.976 bytes free

C:\PRUEBA>C:\GNAT\2021\bin\gnatmake Hola_Mundo.adb
gcc -c hola_mundo.adb
gnatbind -x hola_mundo.ali
gnatlink hola_mundo.ali

C:\PRUEBA>DIR
Volume in drive C has no label.
Volume Serial Number is FC8F-1FCC

Directory of C:\PRUEBA

01/09/2022  08:42 p.m.      <DIR>          .
01/09/2022  08:42 p.m.      <DIR>          ..
31/08/2022  06:42 p.m.                  113 Hola_Mundo.adb
01/09/2022  08:42 p.m.                  2.216 hola_mundo.ali
01/09/2022  08:42 p.m.                 796.880 hola_mundo.exe
01/09/2022  08:42 p.m.                  954 hola_mundo.o
                                4 File(s)          800.163 bytes
                                2 Dir(s)  389.845.254.144 bytes free

C:\PRUEBA>Hola_Mundo
¡Hola, mundo!

C:\PRUEBA>
```



### 2.2.4 MANUAL

El manual oficial de ADA se puede obtener acá; sin embargo, se aclara que no tienen que dominar ni aprender el lenguaje ADA, ni usar el entorno de desarrollo GNAT, tampoco tienen que programar en ADA; basta con solo comprender los COMANDOS de HADA expuestos en este enunciado; este manual se cita por si quieren ampliar sus conocimientos, por lo tanto, no pierdan mucho tiempo con este documento (que por cierto es muy extenso).

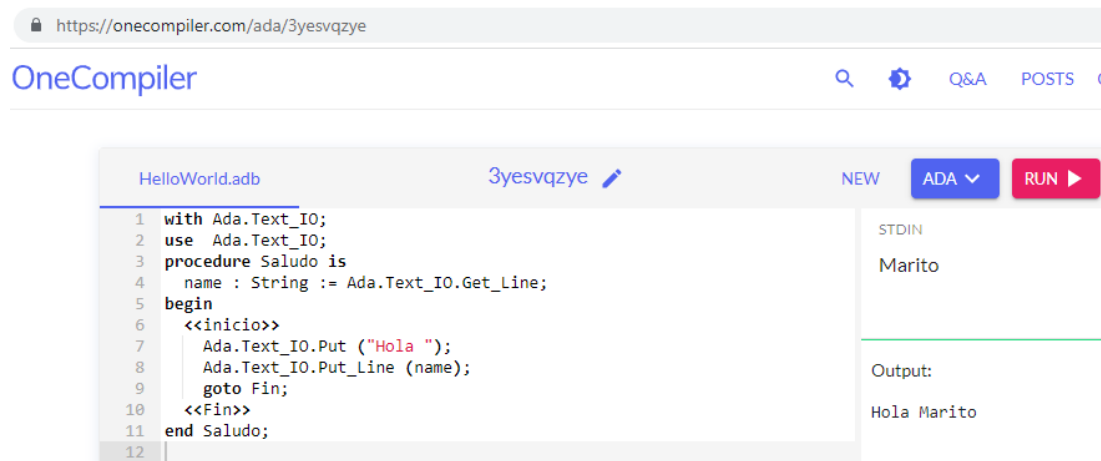
[https://www.adaic.com/resources/add\\_content/standards/05rm/RM-Final.pdf](https://www.adaic.com/resources/add_content/standards/05rm/RM-Final.pdf)

### 2.2.5 COMPILADOR ONLINE

También utilizar un compilador en línea para hacer pruebas más rápidas y entender las particularidades del lenguaje:

<https://onecompiler.com/ada/3yesvqzye>

tal y como vemos acá:



En esta pantalla debemos indicar la entrada de datos en la sección **STDIN** y luego ejecutamos el programa con el botón **RUN**, viendo la salida en la sección **Output**.

**IMPORTANTE:** cada valor en STDIN que corresponde a una variable debe estar en una sola línea y el cursor al final en una línea en blanco, para que así esos valores se asignen, de una vez, a cada una de las variables que requiere el programa; en otras palabras: se dan todos los valores de una sola vez, uno por línea, y no uno por uno, pues la interfaz no es interactiva.



### 2.2.6 LIBRO DE TEXTO

Adicionalmente, se les recuerda que este cuatrimestre no habrá libro de texto, sino que deberán usar los recursos didácticos que se indiquen en la plataforma Moodle (en la “**Documentación General de la Asignatura**”, entre otros); además, deberán revisar semana a semana el **Foro de Consultas**, los temas de estudio, las lecturas sugeridas, los materiales anexados, etc. y estar pendientes de las **sesiones virtuales** para atender dudas, las cuales se comunicarán oportunamente en la plataforma Moodle en el foro de los “**Anuncios de Interés**”. También tienen la libertad de tomar la iniciativa y buscar en el Internet material afín a los temas del curso.

Es importante destacar que el material que suministra la Universidad es muy completo y está bien diseñado para aprender los conceptos de la materia, por lo tanto, es fundamental que semana a semana lo vayan leyendo, incluso antes de iniciar a programar, para que tengan claro qué es un lexema, qué son los análisis léxico, sintáctico y semántico, la tabla de símbolos, “centinelas”, etc.

## 2.3 NOCIONES BÁSICAS

HADA es un compilador sencillo que permite compilar programas básicos hechos en un lenguaje similar al de ADA llamado HADA (porque es una versión mínima de su progenitor ADA).

Usando Netbeans deben realizar un programa en Java llamado HADA.java que implemente un compilador para HADA. Una vez programado y probado se debe generar el archivo .jar o sea el HADA.jar

### 2.3.1 EJECUCIÓN

La sintaxis para usar el compilador HADA es la siguiente, la cual se digita en la línea de comandos de MICROSOFT WINDOWS o CMD; este punto es importante porque el programa que se entrega el Tutor no lo va a probar desde dentro de Netbeans sino desde una línea de comando:

```
java -jar HADA.jar Programa.HADA
```

En donde **Programa** es el nombre del archivo de texto con extensión .HADA (si no se indica se puede asumir).

Dicho nombre (parte izquierda del punto) es un **identificador** y debe seguir las siguientes reglas:

- Tener una longitud máxima de 20 caracteres.
- No es sensible a mayúsculas/minúsculas.
- Empezar con una letra.
- Tener letras, números o guiones bajos.





No usar otros caracteres especiales.  
No empezar o terminar con guión bajo.

Ejemplos:

Correctos

CalcularInflacion.HADA  
CALCULARINFLACION.HADA  
CalcularInflacion.HADA  
Calcular\_Inflacion.HADA  
Calcular\_Inflacion1.HADA

Incorrectos

Calcular Inflacion            porque lleva un blanco.  
1CalcularInflacion.HADA    porque empieza con un dígito.  
CalcularInflacion\_.HADA    porque termina en guion bajo.  
\_CalcularInflacion.HADA    porque empieza con guion bajo.



### 2.3.2 INDEPENDENCIA FÍSICA DEL COMPILADOR

Es necesario insistir mucho en este punto: se debe entregar el archivo \*.jar listo para ser ejecutado en cualquier máquina y en cualquier carpeta sin depender de todo el ambiente de desarrollo de Netbeans. En aquella carpeta que el Tutor disponga, por ejemplo: **C:\REVISION**, debe bastar con copiar el archivo \*.jar, los archivos de entrada de la revisión y el compilador que se entrega debe generar los archivos de salida ahí mismo.

Se recalca una vez más: “ahí mismo”, no es en la raíz de C:\, ni en la raíz de D:\ ni en el escritorio de MICROSOFT WINDOWS ni es una carpeta personal de trabajo, sino en la carpeta que el Tutor disponga. En otras palabras, se desea eliminar la dependencia física del compilador hacia la máquina y que se pueda ejecutar en cualquier parte.

**Se exige esto porque a veces no se ejecuta adecuadamente el programa entregado , porque se colocó en una carpeta que no existe en la computadora del Tutor.**

Antes de hacer las entregas de los instrumentos de evaluación (tareas y proyecto), por favor, de realizar una prueba que confirme que el programa entregado esta sin problemas de la manera que se ha solicitado.

### 2.3.3 EJEMPLO DE USO DEL COMPILADOR

Si tenemos el siguiente archivo de texto que realiza el cálculo de inflaciones, llamado:

**C:\REVISION\CALCULAR\_INFLACION.HADA**

y cuyo contenido es el siguiente (ver próxima página):



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

procedure Calcular_Inflacion is
  Max_Anos  : integer;
  Ano       : integer;
  Respuesta : character;
  Factor1, Factor2, Factor3 : float;
begin
  --
  *****
  -- * Asumiendo tasas de inflacion anual de 7%, 8%, y 10%,          *
  -- * encontrar el factor por el cual cualquier moneda, tales como *
  -- * el franco, el dolar, la libra esterlina, el marco, el rublo,  *
  -- * el yen o el florin han sido devaluadas en 1, 2, ....., N anos.
  *
  --
  *****

  -- Inicio del programa

  <<inicio>>
    new_line;
    -- entrada de datos
    new_line;
    put ( "Por favor, indique la cantidad maxima de anos:");
    new_line;

  <<e10050>>
    get ( Max_Anos );
    skip_line;

    if ( Max_Anos <= 0 ) then
      goto ceroanos;
    end if;

    -- Inicializacion de variables
    Ano      := 0;
    Factor1  := 1.0;
    Factor2  := 1.0;
    Factor3  := 1.0;

    -- Calculos y salida de datos
```



```
new_line;
put ( "          Ano 7%          8%          10%" );
new_line;
for Ano in 1.. Max_Anos loop
    Factor1 := ( Factor1 *
                1.07 );
    Factor2 := ( Factor2 )
                *
                1.08;
    Factor3 := Factor3 * ( 1.10 );
    put ( Ano, 3 ) ;
    put ( Factor1, 10, 2, 0 ) ;
    put ( Factor2, 10, 2, 0 ) ;
    put ( Factor3, 10, 2, 0 ) ;
    new_line;
end loop;
```



```
new_line;
put ( "*** Fin del programa normal ***" );
new_line;

put ( "Otra vez?" );
get ( respuesta );
skip_line;
new_line;
put ( "Lo que respondio: " );
put ( respuesta);
new_line;
if ( respuesta = 'S' or respuesta = 's' ) then
    goto inicio;
else
    goto fin999;
end if;

<<ceroanos>>
put ( "*** Fin del programa porque indico 0 anos ***" );
new_line;

<<fin999>>
null;
put ( "*** Hasta luego ***" );
new_line;

end Calcular_Inflacion;

-- fin del programa
```

la manera de ejecutar el compilador HADA es la siguiente:



```
C:\REVISION\> java -jar HADA.jar CALCULAR_INFLACION.HADA
```

Tu compilador de HADA debe abrir el archivo CALCULAR\_INFLACION.HADA y empezar a leerlo línea por línea; cada una de esas líneas debe ser analizada sintáctica y semánticamente, a fin de detectar errores.

### 2.3.4 ARCHIVO DE ERRORES

HADA debe crear un nuevo archivo de salida de errores, con el mismo nombre solo que con el sufijo “-errores” y extensión txt

Para el ejemplo citado, se generará el archivo CALCULAR\_INFLACION-errores.txt

|  |  |   |
|--|--|---|
|  <p>ESCUELA DE CIENCIAS<br/>EXACTAS Y NATURALES</p> | <p>UNIVERSIDAD ESTATAL A DISTANCIA<br/>ESCUELA DE CIENCIAS EXACTAS Y NATURALES<br/>CARRERA INGENIERÍA INFORMÁTICA<br/>CATEDRA TECNOLOGÍA DE SISTEMAS</p> |  <p>UNED</p> |
|--|--|---|

Compiladores

Código: 03307

Este archivo contendrá una copia del programa en HADA, con las líneas debidamente enumeradas hasta un máximo de 99,999 líneas (con ceros a la izquierda); se puede asumir que nunca ningún programa HADA superará ese límite de líneas (99,999 o sea una menos que cien mil).

Entonces, se vería así (ver la siguiente página):



```
00001 with Ada.Text_IO, Ada.Integer_Text_IO, Ada.Float_Text_IO;
00002 use  Ada.Text_IO, Ada.Integer_Text_IO, Ada.Float_Text_IO;
00003
00004 procedure Calcular_Inflacion is
00005     Max_Anos  : integer;
00006     Ano       : integer;
00007     Respuesta : character;
00008     Factor1, Factor2, Factor3 : float;
00009 begin
00010
00011     -- * Asumiendo tasas de inflacion anual de 7%, 8%, y 10%,
00012     -- * encontrar el factor por el cual cualquier moneda, tales
00013     -- * el franco, el dolar, la libra esterlina, el marco, el
00014     -- * el yen o el florin han sido devaluadas en 1, 2, ....., N
00015     --
00016
00017     -- Inicio del programa
00018
00019     <<inicio>>
00020     new_line;
00021     -- entrada de datos
00022     new_line;
00023     put ( "Por favor, indique la cantidad maxima de anos:");
00024     new_line;
00025
00026     <<e10050>>
00027     get ( Max_Anos );
00028     skip_line;
00029
00030     if ( Max_Anos <= 0 ) then
00031         goto ceroanos;
00032     end if;
00033
00034     -- Inicializacion de variables
00035     Ano      := 0;
00036     Factor1  := 1.0;
00037     Factor2  := 1.0;
```



## Compiladores

Código: 03307

```
00038     Factor3 := 1.0;
00039
00040     -- Calculos y salida de datos
00041     new_line;
00042     put ( "          Ano 7%                8%                10%" );
00043     new_line;
00044     for Ano in 1.. Max_Anos loop
00045         Factor1 := ( Factor1 *
00046                     1.07 );
00047         Factor2 := ( Factor2 )
00048                     *
00049                     1.08;
00050         Factor3 := Factor3 * ( 1.10 );
00051         put ( Ano, 3 ) ;
00052         put ( Factor1, 10, 2, 0 ) ;
00053         put ( Factor2, 10, 2, 0 ) ;
00054         put ( Factor3, 10, 2, 0 ) ;
00055         new_line;
00056     end loop;
00057
00058     new_line;
00059     put ( "*** Fin del programa normal ***" );
00060     new_line;
00061
00062     put ( "Otra vez?" );
00063     get ( respuesta );
00064     skip_line;
00065     new_line;
00066     put ( "Lo que respondio: " );
00067     put ( respuesta);
00068     new_line;
00069     if ( respuesta = 'S' or respuesta = 's' ) then
00070         goto inicio;
00071     else
00072         goto fin999;
00073     end if;
00074
00075     <<ceroanos>>
00076     put ( "*** Fin del programa porque indico 0 anos ***" );
00077     new_line;
00078
00079     <<fin999>>
00080     null;
00081     put ( "*** Hasta luego ***" );
```



```

00082     new_line;
00083
00084 end Calcular_Inflacion;
00085
00086 -- fin del programa

```

Si no se encontraron errores quiere decir que el programa en HADA es compatible con ADA y por lo tanto puede ser ejecutado en ADA.

### 2.3.5 INVOCACIÓN DE GNAT

Entonces tu compilador HADA debe invocar automáticamente al compilador GNAT llamado **gnatmake.exe** para que el programa sea corrido. Este compilador se debe descargar desde la dirección antes citada (véase el punto 2.2.3).

Para implementar este mecanismo el estudiante deberá investigar cómo manejar las variables de ambiente de CMD necesarias para realizar esas invocaciones, así como valorar si temporalmente debe cambiar la extensión del archivo .HADA a .adb (porque .adb es la extensión de los archivos que maneja **gnatmake.exe**) o bien sacar una copia con extensión .adb para poder ser compilada en GNAT y posteriormente ejecutada.

Este es un buen artículo que explica el pase de parámetros desde CMD a un programa Java:

<https://personales.unican.es/corcuerp/java/Slides/CommandLineArguments.pdf>

En otras palabras, si el programa anterior en HADA, CALCULAR\_INFLACION.HADA, no tuviera errores, entonces así se vería la invocación del comando:

C:\REVISION\gnatmake CALCULAR\_INFLACION.adb

El programa se ejecutaría y mostraría los resultados en la ventana del CMD.

Tu programa debe hacer la invocación del compilador gnatmake.exe de manera totalmente automática y transparente para el usuario, dando la ilusión de que es tu compilador quien lo está ejecutando, según se ve en las siguientes imágenes:



```
C:\Windows\System32\cmd.exe

C:\PRUEBA>C:\GNAT\2021\bin\gnatmake Calcular_Inflacion.adb
gcc -c calcular_inflacion.adb
gnatbind -x calcular_inflacion.ali
gnatlink calcular_inflacion.ali

C:\PRUEBA>DIR
Volume in drive C has no label.
Volume Serial Number is FC8F-1FCC

Directory of C:\PRUEBA

03/09/2022  04:43 p.m.      <DIR>          .
03/09/2022  04:43 p.m.      <DIR>          ..
03/09/2022  04:43 p.m.                2.193 Calcular_Inflacion.adb
03/09/2022  04:43 p.m.                3.478 calcular_inflacion.ali
03/09/2022  04:43 p.m.            940.883 calcular_inflacion.exe
03/09/2022  04:43 p.m.                3.082 calcular_inflacion.o
                                4 File(s)          949.636 bytes
                                2 Dir(s)      388.481.855.488 bytes free
```



```
C:\Windows\System32\cmd.exe
C:\PRUEBA>Calcular_Inflacion

Por favor, indique la cantidad maxima de anos:
5

    Ano 7%      8%      10%
1     1.07     1.08     1.10
2     1.14     1.17     1.21
3     1.23     1.26     1.33
4     1.31     1.36     1.46
5     1.40     1.47     1.61

*** Fin del programa normal ***
Otra vez?S

Lo que respondio: S

Por favor, indique la cantidad maxima de anos:
3

    Ano 7%      8%      10%
1     1.07     1.08     1.10
2     1.14     1.17     1.21
3     1.23     1.26     1.33

*** Fin del programa normal ***
Otra vez?s

Lo que respondio: s

Por favor, indique la cantidad maxima de anos:
7

    Ano 7%      8%      10%
1     1.07     1.08     1.10
2     1.14     1.17     1.21
3     1.23     1.26     1.33
4     1.31     1.36     1.46
5     1.40     1.47     1.61
6     1.50     1.59     1.77
7     1.61     1.71     1.95

*** Fin del programa normal ***
Otra vez?n

Lo que respondio: n
*** Hasta luego ***
```



## 2.4 SINTAXIS DE HADA

Es importante destacar que el lenguaje de programación HADA es un subconjunto del lenguaje de programación ADA, de tal manera que para cualquier duda que se tenga se puede consultar un manual técnico de ADA (véase el punto 2.2.1), ya que los COMANDOS son los mismos, excepto que para HADA vamos a precisar las siguientes reglas específicas en los siguientes aspectos:

### 2.4.1 FORMATO DEL ARCHIVO

- Un archivo de código fuente en HADA se puede escribir en cualquier editor, siempre y cuando se grabe el contenido como “texto sin formato” (en código ASCII). Las líneas de este archivo deben terminar con ENTER o RETURN.
- El archivo puede contener líneas en blanco o bien líneas con COMANDOS de HADA o ADA.
- El formato de la línea debe ser el siguiente y se debe validar de manera estricta, reportando errores cuando no se cumple alguno de estos puntos del formato.
- Una línea física no puede tener más de 80 columnas o caracteres.
- El terminador de línea es el punto y coma.
- Si la línea empieza con PROCEDURE, BEGIN, IF, ELSE, FOR no debe terminar con punto y coma.
- Si la línea es una **etiqueta**: (véase el punto 2.5.9) no debe terminar con punto y coma.
- Una línea lógica puede tomar hasta 5 líneas físicas; la última de ellas puede terminar con punto y coma conforme a las pautas explicadas anteriormente; estas líneas son llamadas multilíneas; se reitera que pueden abarcar un máximo de 5 líneas físicas y se puede asumir que esa regla siempre se cumplirá.

Ejemplo:

```
factor2 := ( factor2 )
          *
          1.08;
```

- Toda línea lógica corresponde a un solo COMANDO HADA o ADA que, eventualmente según se explicó, debe terminar con punto y coma.
- Las líneas en blanco en cualquier parte del archivo se ignoran en cuanto a su procesamiento, o sea que no hay que hacer nada con ellas pero se debe respetar su presencia y mostrarlas en el archivo de errores.
- Una línea en blanco puede ser literalmente NULA (constituida por solo un ENTER) o tener uno o más espacios, como estas que vienen a continuación con 0, 1, 7 y 35 espacios respectivamente y en donde la fecha indica dónde terminan esos blancos:

```
<--
```

```
<--
```

```
<--
```

```
<--
```

- Decimos que el carácter blanco es el que corresponde a la espaciadora; se puede asumir que otros caracteres similares como tabuladores (TAB) o caracteres invisibles "que parecen blancos" no vendrán en el archivo.
- Los blancos o espacios "redundantes" antes, en el medio o al final de una línea se ignoran en cuanto a su procesamiento, o sea que no hay que hacer nada con ellos pero se debe respetar su presencia y mostrarlos en el archivo de errores tal y como venían en el archivo de entrada. El alumno dispondrá, a su gusto, el procesamiento que dé a estos blancos redundantes, aplicando alguna de las técnicas que se explican en el material de la asignatura.
- Un blanco es "redundante" cuando se puede eliminar y aún así procesar la línea sin ambigüedades, por ejemplo a la línea siguiente (nótese los blancos redundantes al final que terminan donde está la flecha):

Dato := ( 3.2 \* ( 4.2 - 3.1 \* 2.5 ) / 6.35 ); <--

se le pueden eliminar los blancos redundantes y procesarla como si en realidad fuera así (nótese que se eliminaron los blancos al final de la línea):

Dato:=(3.2\*(4.2-3.1\*2.5)/6.35);

- Se podrá asumir que los caracteres propios del idioma español como los signos de apertura de exclamación y de interrogación (¡, ¿), tildes, eñes, etc. no vendrán en el archivo ni siquiera en los comentarios (para no "meter ruido").

## 2.4.2 COMENTARIOS

Se debe tener claro que los comentarios como tales son un comando también.

Los comentarios se indican con dos guiones seguidos (debe validarse que vengan "pegados"); a partir de su aparición todo lo demás es comentario hasta el final de la línea física; por lo tanto, no hay caracteres que indiquen el cierre de comentario y todo comentario toma a lo más una línea física.

NOTA: pueden haber varias líneas de comentarios seguidas.

Los comentarios puede aparecer en cualquier parte del archivo.

Se puede asumir, sin embargo, que siempre empezarán en una línea nueva.

Ejemplos:

Correctos:

```
-- Inicio del programa
Factor1, Factor2, Factor3 : float;
begin
  --
  ****
  ****
```



```
-- * Asumiendo tasas de inflacion anual de 7%, 8%, y 10%,
*
-- * encontrar el factor por el cual cualquier moneda,
tales como *
-- * el franco, el dolar, la libra esterlina, el marco,
el rublo, *
-- * el yen o el florin han sido devaluadas en 1, 2,
...., N anos. *
--
*****
*****
```

Incorrecto:

```
- Inicio del programa          (porque solo aparece un
guion)
Factor1, Factor2, Factor3 : float;
```



### 2.4.3 ESTRUCTURA DEL ARCHIVO

Los archivos de código fuente en HADA siempre tendrán la siguiente estructura sin excepción; no se debe asumir y se debe verificar; si no la cumplen se deben enviar los mensajes de error respectivos:

#### 2.4.3.1 PROCEDURE NOMBRE\_PROGRAMA IS

Todo programa HADA debe empezar con la palabra reservada **PROCEDURE** (solo debe aparecer una vez y debe ser una aparición válida), llevar un **NOMBRE\_PROGRAMA** (que debe ser un identificador válido, ver el punto 2.5.5) y finalizar esa línea con la palabra reservada **IS**. Como ya se indicó, no debe terminar con punto y coma.

Antes de **PROCEDURE** solo pueden aparecer líneas en blanco, comentarios o líneas con **COMANDOS** de ADA (ver el punto 2.4.3.5).

Se puede asumir que los 3 lexemas (**PROCEDURE NOMBRE\_PROGRAMA IS**) vendrán en una misma línea física o sea no serán multilínea.

#### 2.4.3.2 SECCIÓN DE VARIABLES

Sección donde se definen las variables que utiliza el programa; es opcional.

Más adelante (ver el punto 2.5.6) se explica cómo se declaran.

Empieza luego de que aparece **PROCEDURE**.

Termina donde aparece **BEGIN**.

Fuera de esta sección no deben permitirse más declaraciones de variables.

#### 2.4.3.3 BEGIN

**COMANDO** con el que se se inicia el programa; siempre debe ser el primero y solo debe aparecer una vez y debe ser una aparición válida; no se deben permitir otros **COMANDOS** HADA mientras no haya aparecido **BEGIN**. Nótese que antes de **BEGIN** sí pueden venir líneas en blanco, comentarios o comandos ADA.

#### 2.4.3.4 SECCIÓN DE COMANDOS

Sección donde se indican los **COMANDOS** HADA que utiliza el programa; es obligatoria.

Empieza luego de **BEGIN**.

Termina con **END NOMBRE\_PROGRAMA**.

No pueden indicarse **COMANDOS** HADA o ADA después de **END NOMBRE\_PROGRAMA**, aunque sí pueden aparecer líneas en blanco o comentarios.

#### 2.4.3.5 PALABRAS RESERVADAS DE ADA

Cualquier **COMANDO** que no se reconozca como válido de HADA (pero sí de ADA) se ignorará, pero en el archivo de errores saldrá el mensaje:

**Advertencia: instrucción no es soportada por esta versión.**

Estas advertencias no se considerarán errores.





Ejemplo:

```
with Ada.Text_IO;
```

Otros ejemplos de palabras reservadas:

```
access  
case  
delay  
exception  
loop  
raise  
return  
terminate  
until  
while
```

Una vez detectados estos COMANDOS que HADA no soporta el resto de la línea se debe ignorar y por lo tanto no compilar; la idea es permitir instrucciones no soportadas por HADA pero sí por ADA. Para poder implementar esto, se sugiere llevar un arreglo o vector de todas las palabras reservadas de ADA (el alumno, si gusta, puede hacerlo de otra manera que juzgue conveniente; al final de este documento en el **ANEXO 1** se adjuntan todas las palabras reservadas de ADA). Cualquier otra cosa que no esté en esta lista y que no correspondan a COMANDOS de HADA deberá aparecer un mensaje como el siguiente:

ERROR: "xxxx" no es una instrucción válida de HADA ni de ADA.

(estos errores sí se considerarán como tales; es importante recordar que se debe invocar al compilador **gnatmake.exe**, ver punto **2.3.5**, solo si no se detectaron errores; por otro lado, si se detectaron solo advertencias o "warnings" relativas a la palabras reservadas entonces sí se debe invocar a **gnatmake.exe**).

#### 2.4.3.6 PALABRAS RESERVADAS DE HADA

Al final de este documento en el **ANEXO 2** se adjuntan todas las palabras reservadas de HADA, las cuales corresponden a los COMANDOS explicados en este documento.

#### 2.4.3.7 END NOMBRE\_PROGRAMA

Todo programa HADA debe finalizar con la palabra reservada END y llevar un NOMBRE\_PROGRAMA (que debe ser un identificador válido, ver el punto 2.5.5).

Estos dos lexemas solo deben aparecer una vez y debe ser una aparición válida.

Como ya se indicó, deben terminar con punto y coma.

Se puede asumir que los 2 lexemas (END NOMBRE\_PROGRAMA) vendrán en una misma línea física o sea no serán multilínea.

No se deben permitir otros COMANDOS después de que aparecen, aunque sí puede haber líneas en blanco o comentarios.





NOMBRE\_PROGRAMA debe ser el mismo que se utilizó en PROCEDURE.

#### 2.4.3.8 MAYÚSCULAS/MINÚSCULAS

HADA no es sensible a mayúsculas y minúsculas en cuanto a los elementos del lenguaje.

Sin embargo, lo que vaya dentro de las comillas simples o dobles (que se usan como delimitadores de textos) se debe respetar literalmente.

Ejemplos:

PUT\_LINE, put\_line, Put\_Line, PUT\_line, PuT\_LiNe son válidas y se refieren al mismo COMANDO.

FACTOR1, factor1, FACtor1, facTOR1, FaCtOr1 son válidas y se refieren al mismo identificador.

#### 2.4.3.9 UN COMANDO POR LÍNEA

HADA solo permite un COMANDO por línea lógica (la cual puede estar formada por varias líneas físicas como ya se explicó; ver el punto 2.4.1).

Se reitera que el carácter de terminación de línea siempre es el punto y coma.

Sin embargo, como ya se indicó, hay algunas excepciones en las cuales el punto y coma no debe aparecer.

Ejemplos:

Correctos:

```
-- Inicializacion de variables
Ano      := 0;
Factor1  := 1.0;
Factor2  := 1.0;
Factor3  := 1.0;

-- Calculos y salida de datos
new_line;
put ( "      Ano 7%          8%          10%" );
new_line;
```

Incorrectos (dos comandos en una misma línea marcados en negrita):

```
-- Inicializacion de variables
Ano      := 0;
Factor1  := 1.0;
Factor2 := 1.0;   Factor3 := 1.0;

-- Calculos y salida de datos
new_line;
put ( "      Ano 7%          8%          10%" );
new_line;
```



## 2.5 ELEMENTOS DEL LENGUAJE

Se exponen a continuación los principales elementos del lenguaje HADA.

### 2.5.1 AGRUPACIÓN

Los caracteres de agrupación válidos para HADA son:

( paréntesis izquierdo  
) paréntesis derecho

Ejemplo:

```
factor3 := factor3 * ( 1.10 );
```

### 2.5.2 CONSTANTES

En HADA hay cuatro tipos de constantes: caracter, texto, enteras y reales.

Las de caracter son de un solo caracter y van delimitadas por comillas simples.

Las de texto tienen hasta 50 caracteres y van delimitadas por comillas dobles.

Lo que va dentro de ambas debe respetarse en cuanto a mayúsculas y minúsculas.

Se puede asumir, “para no meter ruido”, que no llevarán caracteres del español como las tildes.

Ejemplos:

Caracter:

‘S’

‘n’

‘7’

Texto:

“HOLA”

“\$25,000.00”

“Numero de empleados”

Las numéricas pueden ser positivas o negativas, llevar decimales (el separador de decimales es el punto) y no deben llevar comas.

Hay 2 tipos de constantes numéricas:

- enteras: números enteros positivos o negativos que no tienen punto decimal y que están en el rango  
-32768 y +32767

Ejemplos:

1

-1

10

-10

- reales: números reales positivos o negativos que sí tienen punto decimal y al menos un dígito decimal y que están en el rango -32768.00 y +32767.99

Ejemplos:

1.0  
-1.0  
10.23  
-10.23

Nótese que 1. y 10. son incorrectas porque deben tener al menos un dígito decimal.

### 2.5.3 OPERADORES ARITMÉTICOS

Los siguientes son los operadores aritméticos que maneja HADA:

| Precedencia | Operador | Operación      | Ejemplo | En HADA       |
|-------------|----------|----------------|---------|---------------|
| 1           | *        | Multiplicación | XY/Z    | $X * Y / Z$   |
| 1           | /        | División       | X+Y/Z   | $X + (Y / Z)$ |
| 2           | +        | Suma           | X+Y/Z   | $(X + Y) / Z$ |
| 2           | -        | Resta          | X-Z/Y   | $(X - Y) / Z$ |

Las operaciones entre paréntesis se ejecutan primero.

Nótese de los ejemplos anteriores y de sus equivalentes en HADA el uso adecuado de los paréntesis para cambiar las reglas de precedencia de los operadores aritméticos; cuando dos de ellos tienen la misma precedencia entonces se hacen los cálculos de izquierda a derecha.

### 2.5.4. OPERADORES RELACIONALES

Los siguientes son los operadores relacionales que maneja HADA.

Nótese que no deben llevar blancos entre los dos caracteres especiales (si es del caso):

= Igual.  
/= No igual o diferente.  
< Menor que.  
> Mayor que.  
<= Menor o igual.  
>= Mayor o igual.

Estos son incorrectos porque llevan espacios:      < =      / =

Ejemplo:

```
if ( Max_Anos <= 0 ) then
    goto ceroanos;
end if;
```



### 2.5.5 IDENTIFICADORES

En HADA los identificadores se definen siguiendo las siguientes reglas:

- Tener una longitud máxima de 20 caracteres.
- No son sensibles a mayúsculas y minúsculas.
- Empezar con una letra.
- Tener letras, números o guiones bajos.
- No usar otros caracteres especiales.
- No empezar o terminar con guion bajo.
- No corresponder a una palabra reservada de HADA o de ADA.

Ejemplos:

Correctos

Max\_Anos  
ANO  
factor1  
respUESTA

Incorrectos

|            |  |
|------------|--|
| max-anos   | porque lleva un carácter especial en el medio. |
| 1factor    | porque no empieza con letra.                   |
| Factor2%   | porque lleva un carácter especial al final.    |
| _no_se_usa | porque no empieza con letra.                   |
| FLOAT      | porque es palabra reservada de HADA.           |
| null       | porque es palabra reservada de ADA.            |

### 2.5.6 DEFINICIÓN DE VARIABLES

En HADA las variables se definen siguiendo la siguiente sintaxis:

identificadores (uno o más separados por coma) : TIPO-DE-DATOS

en dónde:

- TIPO-DE-DATOS puede ser INTEGER, FLOAT o CHARACTER;
- El caracter : (dos puntos) es obligatorio y debe aparecer solo una vez antes del TIPO-DE-DATOS.

INTEGER      se usará para almacenar números enteros;  
FLOAT        se usará para almacenar números reales;  
CHARACTER se usará para almacenar un caracter;

Los identificadores de variables deben ser válidos conforme a las reglas indicadas anteriormente; si hay más de un identificador de variable se deben separar por comas; pero no deben quedar comas “guindando” (al principio, en el medio o al final); tampoco deben permitirse identificadores repetidos ni siquiera con tipos de datos diferentes.



Ejemplos:

Correctos

```
Max_Anos   : integer;
Ano        : integer;
Respuesta  : character;
Factor1, Factor2, Factor3 : float;
```

Incorrectos

```
max_anos ano   : INTEGER;           porque falta la coma como
separador de variables
,max_anos, ano : INTEGER;           porque hay una coma
"guindando".
max_anos, ano, : INTEGER;           porque hay una coma
"guindando".
FACTOR1, factor2, factor3 float;    porque falta el dos
puntos.
no_se_usa_    : CHARACTER;          porque termina con guion
bajo.
```

### 2.5.7 CALIDAD DE LOS ERRORES

HADA debe reportar al usuario los errores que detecta cuando analiza las hileras de caracteres que conforman el archivo fuente que corresponde al programa.

Los diversos mensajes de error deben tener este formato:

ERROR 999: texto del error.

Todos los errores que se vayan a manejar deben ser identificados por un código y un texto. La enumeración de los errores queda a criterio del estudiante. Los errores deben ser claros y concisos y referirse a solo una situación de error por vez, de manera que un texto como éste:

*Variable no definida o de tipo inválido.*

no es correcto pues son dos errores diferentes en un mismo mensaje.

El mensaje de error debe indicar entre paréntesis cuadrados (corchetes) cuál es el lexema que provoca el error.

Ejemplos:

```
factor1, factor2, factor3 : FFLOAT;
ERROR 025: Tipo de datos inválido [FFLOAT].
```

```
put ( Anno, 3 ) ;
```

ERROR 027: variable no definida [Anno]

GOTO zero\_anos

ERROR 030: etiqueta no existe. [zero\_anos]

Los errores deben aparecer a partir de la columna 8 de la línea respectiva en el archivo de errores, de manera que así se facilite asociar, visualmente, los errores a la línea del programa HADA; no olvidar que una sola línea puede tener varios errores y que por lo tanto todos se deben mostrar, de una vez, conforme a lo que se acaba de indicar.

También es importante recalcar, una vez más, que en el archivo de errores deben aparecer **todas las líneas del programa HADA, tanto las válidas como las inválidas**, debidamente enumeradas a 5 dígitos (con ceros a la izquierda) y con los mensajes de error asociados (si los hay) abajo, todos de una vez, uno por línea, tal y como se acaba de explicar. Esto es **IMPORTANTÍSIMO** para la revisión de las tareas y de la entrega final del proyecto.

Ejemplo:

```

00004 procedure Calcular_Inflacion is
00005   Max_Anos   : integer;
00006   Ano        : integer;
00007   Respuesta  : character;
00008   Factor1, Factor2, Factor3 : ffloat;
00009   ERROR 025: Tipo de datos inválido [FFLOAT].
00009 begin

00028   skip_line;
00029
00030   if ( Max_Anos <= 0 ) then
00031     goto cero_anos;
00032   ERROR 030: etiqueta no existe. [zero_anos]
00032   end if;
00033

00047   Factor2 := ( Factor2 )
00048           *
00049           1.08;
00050   Factor3 := Factor3 * ( 1.10 );
00051   put ( Anno, 3 ) ;
00052   ERROR 027: variable no definida [Anno]
00052   put ( Factor1, 10, 2, 0 ) ;

```

Nótese qué fácil es notar que hay errores pues las líneas enumeradas se ven interrumpidas por líneas en blanco, las cuales muestran el mensaje de error a la derecha; esto hace que el archivo de errores sea muy legible.

### 2.5.8 ASIGNACIÓN DE VARIABLES

La asignación de valores a las variables se hace mediante el signo := o “dos puntos igual”.

Ambos caracteres deben ir pegados sin blanco intermedio entre ellos.

Ver el COMANDO ASIGNAR en el punto 2.6.2.

### 2.5.9 ETIQUETAS

#### <<identificador>>

Las etiquetas son identificadores (ver el punto **2.5.5**) precedidos por << y seguidos por >>; deben ser únicas (no pueden ser repetidas) y no llevan ningún orden en particular.

<< deben aparecer pegados sin blanco intermedio entre ellos.

>> deben aparecer pegados sin blanco intermedio entre ellos.

La ausencia de << o de >> es error.

Puede asumirse que los tres lexemas de una etiqueta (<<identificador>>) aparecen en la misma línea física.

Pueden haber blancos redundantes entre <<, la etiqueta y >> así como al inicio o al final de la línea.

Después de >> no debe aparecer nada más ni siquiera un punto y coma.

Las etiquetas se usan con el COMANDO HADA llamado GOTO; nótese que una etiqueta puede usarse antes de definirse, o sea aparecer primero GOTO y más adelante la definición de la etiqueta, como, por ejemplo: la etiqueta <<ceroanos>> en el ejemplo de este enunciado.



Ejemplo:

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

procedure Calcular_Inflacion is
  Max_Anos   : integer;
  Ano        : integer;
  Respuesta  : character;
  Factor1, Factor2, Factor3 : float;
begin
  --
  *****
  *
  -- * Asumiendo tasas de inflacion anual de 7%, 8%, y 10%,
  *
  -- * encontrar el factor por el cual cualquier moneda, tales
  como *
  -- * el franco, el dolar, la libra esterlina, el marco, el
  rublo, *
  -- * el yen o el florin han sido devaluadas en 1, 2, ....., N
  anos. *
  --
  *****
  *

  -- Inicio del programa

  ...

  if ( Max_Anos <= 0 ) then
    goto ceroanos;
  end if;

  -- Inicializacion de variables
  Ano      := 0;
  Factor1  := 1.0;
  Factor2  := 1.0;
  Factor3  := 1.0;

  ...

  if ( respuesta = 'S' or respuesta = 's' ) then
    goto inicio;
```





```
        else
            goto fin999;
        end if;

    <<ceroanos>>
        put ( "*** Fin del programa porque indico 0 anos ***" );
        new_line;

    <<fin999>>
        null;
        put ( "*** Hasta luego ***" );
        new_line;

end Calcular_Inflacion;
-- fin del programa
```



## 2.6 COMANDOS

Se describen a continuación cada uno de los COMANDOS que maneja HADA.

### 2.6.1 GET

Sintaxis: **get ( Variable )**

Permite ingresar valores a las variables desde el teclado.

Se puede asumir que solo se pide una variable a la vez.

Los paréntesis son obligatorios.

La variable debe haber sido definida previamente.

Ejemplos:

```
get ( Max_Anos );
```

#### NOTA

Esta es una de las grandes ventajas técnicas de ADA, que tratamos de aprovechar en este proyecto de HADA, a saber: se deben indicar expresamente las librerías que se van a utilizar, lo cual en estos tiempos es la norma, pero en aquellos era la novedad.

¿Qué quiere decir esto?

Que si, por ejemplo, tu programa no usa variable reales o flotantes, sino solo enteras, entonces puedes quitar de estas instrucciones la última librería (en negrita), pues de esa manera el ejecutable que se obtiene es más pequeño y por lo tanto, en teoría, más eficiente:

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

Por cierto, la primera de estas librerías es la más básica de todas y entonces casi que cualquier programa de ADA (y de HADA) debe llevarla.

“use” nos permite indicar que los comandos dentro de esas librerías los vamos a utilizar sin necesidad de estar poniendo al inicio todo el nombre completo de las librerías. Por eso es que nada más escribimos “get” o “put”.

COROLARIO: siempre deben aparecer al principio los COMANDOS ADA with y use.

### 2.6.2 SKIP\_LINE

Sintaxis: **skip\_line**

Permite indicar que luego de leer una variable desde el teclado se haga un salto de línea o ENTER.

Nótese el guion bajo que separa las dos palabras.

Ejemplo:

```
skip_line;
```



### 2.6.3 ASIGNAR

Sintaxis: **variable := expresión**

Para asignar el valor de una expresión a una variable.

La variable debe haber sido definida previamente.

Se puede asumir que la expresión siempre es correcta y no se debe validar en cuanto a correctitud de los paréntesis, uso de los operadores aritméticos o relacionales, posición de los operandos, etc.

Sin embargo, se debe analizar cada uno de los identificadores y palabras reservadas que aparecen en la expresión y confirmar que todos correspondan a:

- variables previamente definidas; o
- palabras reservadas de ADA.

Es importante recordar que el orden de los análisis en todo compilador es el siguiente:

- léxico;
- sintáctico;
- semántico.

En consecuencia, los errores se deben reportar en ese orden. Veamos.

Léxico: errores de lexemas esperados y que no se encuentran o que son inválidos:

ejemplo: `jet ( Max_Anos );` es error porque `jet` no es un COMANDO válido ni es palabra reservada de ADA.

Sintáctico: lexemas mal escritos según los patrones de definición:

ejemplo: `get ( 1Max_Anos );` es error porque el identificador empieza con número

Semántico: lexemas bien escritos pero que no corresponden a objetos definidos:

ejemplo: `get ( Max_Anos );` es error si `Max_Anos` no ha sido definida.

Se recalca que este es un ejemplo; el estudiante debe determinar en todo momento qué validaciones de los tres análisis debe efectuar conforme a este enunciado.

Además, en este comando ASIGNAR se debe validar que el tipo de datos de la variable izquierda coincida con el de todas las variables que se utilicen al lado derecho, bajo las siguientes reglas:

- si la variable a la izquierda es INTEGER o FLOAT entonces todas las variables a la derecha deben ser INTEGER o FLOAT;



## Compiladores

Código: 03307

- si la variable a la izquierda es CHARACTER entonces todas las variables a la derecha deben ser CHARACTER.

Ejemplos:

```
ano      := 0;  
factor1  := 1.0;  
factor2  := 1.0;  
factor3  := 1.0;
```

...

```
factor1 := ( factor1 *  
            1.07 );  
factor2 := ( factor2 )  
            *  
            1.08;  
factor3 := abs ( factor3 * ( 1.10 ) );
```

En este caso, **abs** corresponde a una palabra reservada de ADA.

### 2.6.4 PUT

Sintaxis: **put ( variable\_tipo\_integer, tamaño )**

Sintaxis: **put ( variable\_tipo\_float, tamaño, decimales, científica )**

Sintaxis: **put ( variable\_tipo\_character )**

Sintaxis: **put ( texto )**

Para imprimir en pantalla una variable o un texto.

El par de paréntesis siempre es obligatorio.

La variable debe existir y ser del tipo esperado según la cantidad y tipo de lexemas que trae el COMANDO dentro de los paréntesis.

El separador de lexemas es la coma; no deben haber comas “guindando” tal y como se explicó en el punto 2.5.6.

“Tamaño”, “decimales” y “científica” (indica cuántos dígitos se deben mostrar en la notación científica) deben ser constantes enteras positivas entre 0 y 9 (véase el punto 2.5.2) y siempre deben aparecer dependiendo del tipo de variable que se va a imprimir.

El texto debe ir delimitado por comillas dobles tanto al inicio como al final.

El texto no puede tener más caracteres que los que almacena una constante de tipo texto (véase punto 2.5.2).

Se puede asumir que el texto a imprimir no incluirá las comillas dobles.

Ejemplos:

Correctos:

```
put ( Ano, 3 );  
put ( Factor1, 10, 2, 0 );  
put ( respuesta );
```



```
put ( "Por favor, indique la cantidad maxima de anos:");
```

Incorrectos:

```
put ( Ano 3 );           porque falta la coma.
put ( Factor1, 10, 2, 20 ); porque el lexema "científica"
                        es mayor que 9.
put ( respuestaa);       porque respuestaa no está
definida.
put ( "Por favor, indique la cantidad maxima de anos:);
                        porque faltan las comillas de
cierre.
```

### 2.6.5 NEW\_LINE

Sintaxis: **new\_line**

Imprime una línea en blanco en la pantalla.

Nótese el guion bajo que separa las dos palabras.

Ejemplo:

```
new_line;
```

### 2.6.6 FOR

Sintaxis:

```
for variable_ciclo
in
reverse
    constante_mínima o variable_mínima
..
    constante_máxima o variable_máxima
loop
    COMANDOS
end loop
```

Para hacer un ciclo de la siguiente manera:

la variable\_ciclo toma el valor mínimo (constante o variable) y se va incrementando en 1 hasta llegar al valor máximo denotado por la constante o la variable\_máxima; en ese momento el ciclo termina;

“reverse” es opcional y si se indica en lugar de incremento se hace decremento en 1 desde la constante o variable\_máxima hasta la constante o variable\_mínima.

Los dos puntos seguidos son obligatorios y deben aparecer “pegados”.

La variable\_ciclo debe haber sido definida y ser de tipo INTEGER.

La constante\_mínima debe ser entera.

La variable\_mínima debe haber sido definida y ser de tipo INTEGER.



La constante\_máxima debe ser entera.

La variable\_máxima debe haber sido definida y ser de tipo INTEGER.

Los COMANDOS dentro del ciclo pueden ser cualquiera de los ya citados o bien cualquier otro que sea propio de ADA tal y como ya se explicó.

Sin embargo, no se permiten FOR dentro de otro FOR (o sea anidados) tampoco IF dentro de FOR; se puede asumir que ambos casos no se darán.

“end loop” debe aparecer al final, solo una vez, luego del último COMANDO dentro del ciclo; ambos lexemas son obligatorios.

Se puede asumir que los lexemas desde for hasta loop (el inicio del ciclo) vendrán en una misma línea física o sea no serán multilínea.

Ejemplo:

```
for Ano in 1.. Max_Anos loop
  Factor1 := ( Factor1 *
              1.07 );
  Factor2 := ( Factor2 )
              *
              1.08;
  Factor3 := Factor3 * ( 1.10 );
  put ( Ano, 3 ) ;
  put ( Factor1, 10, 2, 0 ) ;
  put ( Factor2, 10, 2, 0 ) ;
  put ( Factor3, 10, 2, 0 ) ;
  new_line;
end loop;
```



### 2.6.7 IF

```
Sintaxis: if ( condición ) then
          COMANDOS-DE-IF
        else
          COMANDOS-DE-ELSE
        end if
```

Para hacer una decisión de la siguiente manera:

si la ( condición ) es cierta se ejecutan los COMANDOS-DE-IF;  
si la ( condición ) es falsa se ejecutan los COMANDOS-DE-ELSE.

Se debe validar que se cumpla esta sintaxis: if ( condición ) then, o sea que venga el if, que se abran y cierren los paréntesis que rodean la condición y que el then también aparezca; después del THEN nunca va punto y coma. Se puede asumir que los lexemas if ( condición ) then vendrán en una misma línea física o sea no serán multilínea.

Por otro lado, la condición como tal se puede asumir que siempre va a ser correcta y no se debe validar;  
sin embargo, se debe validar que todos los identificadores que aparezcan en la condición estén definidos (no importa el tipo) o bien que sean palabras reservadas de ADA.

Los COMANDOS-DE-IF pueden ser cualquiera de los citados en esta sección COMANDOS o bien cualquier otro que sea propio de ADA tal y como ya se explicó.

Los COMANDOS-DE-ELSE pueden ser cualquiera de los citados en esta sección COMANDOS o bien cualquier otro que sea propio de ADA tal y como ya se explicó.

if es obligatorio; debe haber al menos un COMANDO entre el if y el else o el “end if”, según corresponda.  
else es opcional; si aparece debe haber al menos un COMANDO entre el else y el “end if”.  
El else nunca debe llevar punto y coma.

Sin embargo, no se permiten if dentro de otro if (o sea anidados), tampoco for dentro de if; se puede asumir que ambos casos no se darán.  
“end if” debe aparecer al final, solo una vez, luego del último COMANDO dentro del if o del else según corresponda; ambos lexemas “end if” son obligatorios.



Ejemplo:

```
if ( respuesta = 'S' or respuesta = 's' ) then
    goto inicio;
else
    goto fin999;
end if;
```

### 2.6.8 GOTO

Sintaxis: **goto etiqueta**

Transfiere la ejecución del programa a la línea que tiene la etiqueta indicada.

La etiqueta debe haber sido definida.

Ejemplo:

```
put ( "Otra vez?" );
get ( respuesta );
skip_line;
new_line;
put ( "Lo que respondio: " );
put ( respuesta );
new_line;
if ( respuesta = 'S' or respuesta = 's' ) then
    goto inicio;
else
    goto fin999;
end if;
```

### 2.6.9 NULL

Sintaxis: **null**

Es un COMANDO que indica que “no haga nada”.

NOTA: esta es una de las características más peculiares de ADA y en consecuencia de HADA; antes los programadores simulaban el COMANDO nulo con algo poniendo un COMANDO que no era necesario, tal como:

```
Ano := 0;
```

Por ejemplo: cuando tenían un IF sin ELSE y sin embargo querían poner el ELSE solo para que se viera más legible y reflejar o reiterar que el caso ELSE no tenía procesamiento (o sea dejar claro que no era que se les había olvidado). Por estas y otras necesidades nació el COMANDO null.



Ejemplo:  
null;

### 2.6.10 END

Indicador de que los COMANDOS del programa terminaron. Siempre debe ser el último y solo debe aparecer una vez; no se deben permitir otros COMANDOS después de que aparece, aunque, como ya se indicó, sí puede haber líneas en blanco o comentarios. Véase el punto 2.4.3.7 para la sintaxis exacta.

## 2.7 ARCHIVO PARA PRUEBAS

Como ya mostramos al principio, un ejemplo de programa en HADA es el que aparece en el punto 2.3.3 y sobre el cual hemos basado todo este enunciado.

El problema es que en la vida real los programas no vienen tan "bonitos" o sea bien indentados, claramente escritos y acomodados; en realidad, por experiencia propia, como todos sabemos, los programadores somos descuidados al respecto, así que a continuación mostramos el mismo programa pero "feo", en el sentido de que no es muy ordenado; es importante que las pruebas de tu compilador las hagas usando este programa "feo", ya que así serán los archivos de prueba que usará el Tutor. Toma nota, por favor, de que las primeras líneas tienen desde 0 y hasta muchos caracteres en blanco, antes de que aparezca PROCEDURE; los mismo las líneas intermedias y las finales luego de END NOMBRE\_PROGRAMA; además, hay blancos redundantes por todo lado: antes y después de lexemas; todo eso tu programa debe procesarlo, permitirlo e ignorarlo (según sea el caso conforme a lo expuesto en este enunciado) siempre y cuando no sean errores.

Mira en la próxima página como se ve el "Feo":



```
with      Ada.Text_IO,      Ada.Integer_Text_IO,      Ada.Float_Text_IO;
USE
      ADA.Text_IO,  Ada.Integer_Text_IO,
ADA.Float_Text_IO;

      PROCEDURE      FEO      is
      Max_Anos      :      integer;
      Ano      :      integer;
      Respuesta      :      character      ;
      Factor1,Factor2,Factor3:      float      ;
BEGIN
      --
      *****
      -- * Asumiendo tasas de inflacion anual de 7%, 8%, y 10%,      *
      -- * encontrar el factor por el cual cualquier moneda, tales como
      *
      -- * el franco, el dolar, la libra esterlina, el marco, el rublo, *
      -- * el yen o el florin han sido devaluadas en 1, 2, ....., N anos.
      *
      --
      *****

      --      Inicio del programa

      <<      inicio      >>
      new_line;
      --      entrada de datos
      new_line      ;
      put      (      "Por favor, indique la cantidad maxima de anos:"      )
      ;
      new_line;

      <<      e10050>>
      get(Max_Anos);
      skip_line      ;

      if(Max_Anos<=0) then
      goto      ceroanos;
      end if;

      -- Inicializacion de variables
      Ano:=0;
```



## Compiladores

Código: 03307

```
Factor1      :=      1.0      ;
Factor2      :=      1.0      ;
Factor3 :=      1.0      ;

-- Calculos      y      salida de datos
new_line      ;
put      (      "      Ano 7%      8%      10%"      )      ;
new_line;
for      Ano      in      1 ..      Max_Anos      loop
    Factor1 :=      ( Factor1 *
                    1.07 );
    Factor2 := ( Factor2
                *
                1.08      ;
Factor3:=Factor3*(1.10);
    put (      Ano,      3 ) ;
    put (      Factor1, 10, 2, 0 ) ;
    put (      Factor2, 10, 2, 0 ) ;
    put (      Factor3, 10, 2, 0 )      ;
    new_line      ;
end      loop;

new_line;
put      ( "*** Fin del programa normal ***"      ) ;
new_line;

put (      "Otra vez?"      );
get (      respuesta      )      ;
skip_line      ;
new_line      ;
put("Lo que respondio: ");
put (      respuesta);
new_line;
if(respuesta='S' or      respuesta      =      's'      )      then
    goto inicio;
else
goto fin999;
end      if;

<<      ceroanos>>
put      ( "*** Fin del programa porque indico 0 anos ***" )      ;
new_line;

<<      fin999      >>
```



## Compiladores

Código: 03307

```
null      ;  
put ( "*** Hasta luego ***" )      ;  
    new_line;  
  
end      feo;  
  
-- fin del programa
```

Esta pantalla demuestra que FEO.adb es correcto.

La compilación:

```
C:\Windows\System32\cmd.exe  
  
C:\PRUEBA>DIR  
Volume in drive C has no label.  
Volume Serial Number is FC8F-1FCC  
  
Directory of C:\PRUEBA  
  
05/09/2022  07:27 p.m.      <DIR>          .  
05/09/2022  07:27 p.m.      <DIR>          ..  
05/09/2022  07:27 p.m.                  2.817 Feo.adb  
                1 File(s)                2.817 bytes  
                2 Dir(s)  388.194.820.096 bytes free  
  
C:\PRUEBA>C:\GNAT\2021\bin\gnatmake Feo.adb  
gcc -c feo.adb  
gnatbind -x feo.ali  
gnatlink feo.ali
```



Y la ejecución:

```
C:\Windows\System32\cmd.exe
C:\PRUEBA>Feo

Por favor, indique la cantidad maxima de anos:
3

    Ano 7%      8%      10%
  1    1.07    1.08    1.10
  2    1.14    1.17    1.21
  3    1.23    1.26    1.33

*** Fin del programa normal ***
Otra vez?s

Lo que respondio: s

Por favor, indique la cantidad maxima de anos:
5

    Ano 7%      8%      10%
  1    1.07    1.08    1.10
  2    1.14    1.17    1.21
  3    1.23    1.26    1.33
  4    1.31    1.36    1.46
  5    1.40    1.47    1.61

*** Fin del programa normal ***
Otra vez?s

Lo que respondio: s

Por favor, indique la cantidad maxima de anos:
7

    Ano 7%      8%      10%
  1    1.07    1.08    1.10
  2    1.14    1.17    1.21
  3    1.23    1.26    1.33
  4    1.31    1.36    1.46
  5    1.40    1.47    1.61
  6    1.50    1.59    1.77
  7    1.61    1.71    1.95

*** Fin del programa normal ***
Otra vez?n

Lo que respondio: n
*** Hasta luego ***
```



## 2.8 ENTREGA

1. Todo el proyecto tal y como se organizó en NETBEANS, todas las carpetas, en particular los programas fuentes en Java.

SUGERENCIA: entregar un ZIP con todo, esto es importante porque la no entrega como se está indicando el tutor no revisara lo entregado; es responsabilidad del estudiante al momento de subir la entrega y adjuntar el ZIP correcto.

2. Un manual técnico que explique cómo instalar y ejecutar el compilador HADA desde CMD (deben ser claras las instrucciones). Según lo indicado en el enunciado.

SE REITERA: MUY IMPORTANTE: la ejecución debe ser desde CMD, ya que el Tutor no entrará a Netbeans para probarlo.

SE INSISTE: si no se siguen estas recomendaciones y el programa no corre como se ha indicado entonces lamentablemente no se podrá calificar.

El Tutor analizará y revisará el código fuente del programa para verificar: **plagio**, uso de buenas técnicas de programación, indentación, identificadores significativos, modularidad, orden, etc.

3. En el manual técnico deberá venir una lista de los puntos de esa entrega que no se programaron o que no funcionan correctamente y la justificación del caso, a fin de valorar la calidad de la solución entregada. Si no se entrega este manual se asume que todo funciona al 100%

---

## 2.9 DEFENSA

Al final de la asignatura, en la última semana, una vez entregado el proyecto, se programará una cita con el Tutor para hacer una exposición virtual del proyecto y defenderlo mediante TEAMS, de manera individual, usando audio y cámara, por lo que desde ya el estudiante debe ir previendo tener esos dispositivos instalados y funcionando.

El tutor verificará el uso de buenas técnicas de programación, cumplimiento de los objetivos del proyecto **y plagio (al respecto, se advierte que el Tutor usará herramientas de software para confirmar qué tan parecidos son los programas que entregan los estudiantes con los de otros estudiantes o con los de sí mismo \_ si es repitiente \_ de los cuatrimestres anteriores)** y en caso de detectar plagio se enviarán las evidencias a la cátedra para que realice el debido proceso; se harán preguntas técnicas sobre el código fuente que el estudiante deberá explicar de manera convincente para demostrar que es el autor del programa.

## **ANEXO 1:**

### **Lista de palabras reservadas de ADA**

NOTA: la lista completa de palabras reservadas de ADA es enorme; aquí vamos a indicar las más básicas; durante el desarrollo de tu proyecto, de tus pruebas o bien cuando el Tutor esté calificando es probable que se necesite alguna palabra reservada que no esté en esta lista, entonces lo que tienes que hacer es simplemente agregarla. En ese sentido, entonces, la solución que implementes para el manejo de esta lista debe ser genérica y parametrizada, por ejemplo un vector y una variable que lleve el tamaño o cantidad de palabras reservadas.

abort  
 abs  
 abstract  
 accept  
 access  
 aliased  
 all  
 and  
 array  
 at  
 begin  
 body  
 case  
 constant  
 declare  
 delay  
 delta  
 digits  
 do  
 else  
 elsif  
 end  
 entry  
 exception  
 exit  
 for  
 function  
 generic  
 goto





UNIVERSIDAD ESTATAL A DISTANCIA  
ESCUELA DE CIENCIAS EXACTAS Y NATURALES  
CARRERA INGENIERÍA INFORMÁTICA  
CATEDRA TECNOLOGÍA DE SISTEMAS



## Compiladores

Código: 03307

if  
in  
interface  
is  
limited  
loop  
mod  
new  
not  
null  
of  
or  
others  
out  
overriding  
package  
pragma  
private  
procedure  
protected  
raise  
range  
record  
rem  
renames  
requeue  
return  
reverse  
select  
separate  
subtype  
synchronized  
tagged  
task  
terminate  
then  
type  
until  
use  
while  
with  
xor



UNIVERSIDAD ESTATAL A DISTANCIA  
ESCUELA DE CIENCIAS EXACTAS Y NATURALES  
CARRERA INGENIERÍA INFORMÁTICA  
CATEDRA TECNOLOGÍA DE SISTEMAS



Compiladores

Código: 03307

## **ANEXO 2:**

Lista de palabras reservadas de HADA

### **NOTA**

Las palabras reservadas de HADA tienen precedencia sobre las de ADA. Esto quiere decir que, por ejemplo, if se refiere al COMANDO de HADA y no al de ADA; por lo tanto la especificación de if que se debe tomar en cuenta es la que aparece en este enunciado.

begin  
character  
else  
end  
float  
for  
get  
if  
in  
integer  
is  
loop  
new\_line  
null  
procedure  
put  
skip\_line  
then

FIN