



Universidad Tecnológica Centroamericana

Compiladores I

Manual de Usuario de Goddard

Nombre del catedrático:

Ing. Carlos Vallejo

Integrantes

Carlos Enrique Varela | 11711380

Marvin Fernando Estrada | 11541020

29 de septiembre de 2020

índice

Manual de usuario del Lenguaje de programación Goddard	3
Tipos de variables	3
Operadores aritméticos	3
Operadores lógicos	4
Operadores relacionales	4
Declaración de variables y asignación	4
Bloque de decisión IF	5
Bloque de iteración FOR	5
Bloque de iteración WHILE	5
Bloque de decisión SWITCH CASE	6
Output e Input	6
Return en funciones	7
Comentarios	7
Estructura básica de un programa en Goddard	7
Declaración de funciones	8
Existen tres tipos de funciones que pueden declararse, funciones simples, funciones con parámetros y funciones con retorno, las cuales pueden tener tanto parámetros como no, se crean de la siguiente manera:	8
Funciones simples	8
Funciones con parámetros	8
Funciones con retorno	9
Declaración de funciones con retorno	9
Reglas de Comparación	9
Uso de la interfaz gráfica del compilador	11

Manual de usuario del Lenguaje de programación Goddard

Tipos de variables

Los tipos de variables permitidos por el lenguaje son las siguientes:

- number: Son enteros positivos o negativos.
- boolean: Son tipos booleanos que pueden contener o un **true** o un **false**
- character: Son tipos char que pueden contener solamente un carácter encerrado entre comillas simples, por ejemplo 'C'.
- array[tipo]: Son tipos array que pueden contener array dentro de array, esto permite la creación de un array de una dimensión o de un array de dos dimensiones.
- empty: Es el tipo de variable vacío.

*Nota: No acepta strings ni tampoco comillas dobles ("")

Operadores aritméticos

Los operadores aritméticos que se pueden utilizar son los siguientes:

- Suma: se representa con el carácter +
- Resta: se representa con el carácter -
- División: se representa con el carácter /
- Multiplicación: se representa con el carácter *
- Residuo: se representa con el carácter %

Operadores lógicos

Los operadores lógicos que se pueden utilizar son los siguientes:

- Operador AND: se utiliza el símbolo **&&**
- Operador OR: se utiliza el símbolo **<>**

Operadores relacionales

- Símbolo de asignación: se utiliza el símbolo **=**
- Símbolo de igualdad: se utiliza el símbolo **==**
- Símbolo de diferencia: se utiliza el símbolo **!=**
- Símbolo de mayor: se utiliza el símbolo **>**
- Símbolo de menor: se utiliza el símbolo **<**
- Símbolo de mayor igual: se utiliza el símbolo **>=**
- Símbolo de menor igual: se utiliza el símbolo **<=**

Declaración de variables y asignación

Existen tres tipos de declaración de variables:

- Declaración simple: number n
 - Asignación: n = 10
- Declaración y asignación: number n = 10
 - Números enteros: number x
- Caracteres: char a
 - Arreglos: array[number]
 - Matrices: array[array[number]]

Bloque de decisión IF

Estructura de la condición IF es la siguiente:

```
if (condición) then {  
    ... expresión  
} else {  
    ... expresión  
}
```

Estructura de la condición IF-ELSE es la siguiente:

```
if (condición) then {  
    ... expresión  
} else if (condición) then {  
    ... expresión  
}
```

Bloque de iteración FOR

La estructura de el ciclo FOR se define de la siguiente manera:

```
for ( id = n to m | y ) then {  
    ... expresión  
}
```

Tener cuidado con el símbolo y, este indica el incremento o decremento de la variable id.

Bloque de iteración WHILE

Estructura del ciclo WHILE:

```
while (condición) then{  
    ... expresión  
}
```

Bloque de decisión SWITCH CASE

Estructura del switch case:

```
switch (id) then{  
    case (1) :{  
        ... expresión  
    };  
    case (2) :{  
        ... expresión  
    };  
    default:{  
        ... expresión  
    };  
}
```

El número de casos dentro del switch pueden ser la cantidad definida por el usuario.

Output e Input

La declaración de output recibe solamente los tipos de datos admitidos en el lenguaje, su declaración es de la siguiente manera:

```
output(ID)
```

La declaración de input se tiene que ser asignado a una variable, se declara de la siguiente manera:

```
number id = input()
```

Return en funciones

Esta declaración se usa solamente en aquellas funciones que tengan especificado el tipo de retorno, la estructura es la siguiente:

comeBack ("lo que quiero retornar")

Comentarios

Los comentarios son líneas de texto ignoradas por el compilador, constan de un símbolo que indica el inicio del comentario y otro símbolo que consta del final del mismo, se crea de la siguiente manera:

Aquí va el comentario

Estructura básica de un programa en Goddard

El lenguaje consta de una función MAIN y varias funciones extras que son opcionales, tener siempre en consideración la creación del método MAIN, sin este el programa no funcionará. Tomando en consideración lo antes mencionado una estructura básica sería de la siguiente manera:

#Declaración de la función MAIN#

```
function main() = {  
}
```

Declaración de funciones

Existen tres tipos de funciones que pueden declararse, funciones simples, funciones con parámetros y funciones con retorno, las cuales pueden tener tanto parámetros como no, se crean de la siguiente manera:

Funciones simples

Las funciones simples constan de aquellas que no tienen un tipo de retorno especificado y no tienen parámetros, se crea de la siguiente manera:

```
function name()={  
    #Este es el cuerpo de la función#  
}
```

Funciones con parámetros

Las funciones con parámetros se crean de la siguiente manera:

```
function name(id: Type, id2: Type) = {  
}
```

Donde “Type” significa el tipo de variable que espera la función.

Funciones con retorno

Las funciones con retorno se crean de la siguiente manera:

Declaración de funciones con retorno

```
function name() = type {  
    comeBack (value)  
}  
  
function name(id: Type, id2: Type) = type {  
    comeBack (value)  
}
```

En estas funciones es de importancia especificar la declaración de retorno `comeBack()`, esta declaración no se puede utilizar en las demás funciones mostradas arriba.

Reglas de Comparación

Los objetos tipo numbers se pueden evaluar con objetos del mismo tipo con los diferentes operadores relacionales. Ejemplo:

`number == number, number != number, number > number, number < number`

Los objetos tipo boolean se pueden evaluar con objetos del mismo tipo pero solo para validar el estado del objeto. Ejemplo:

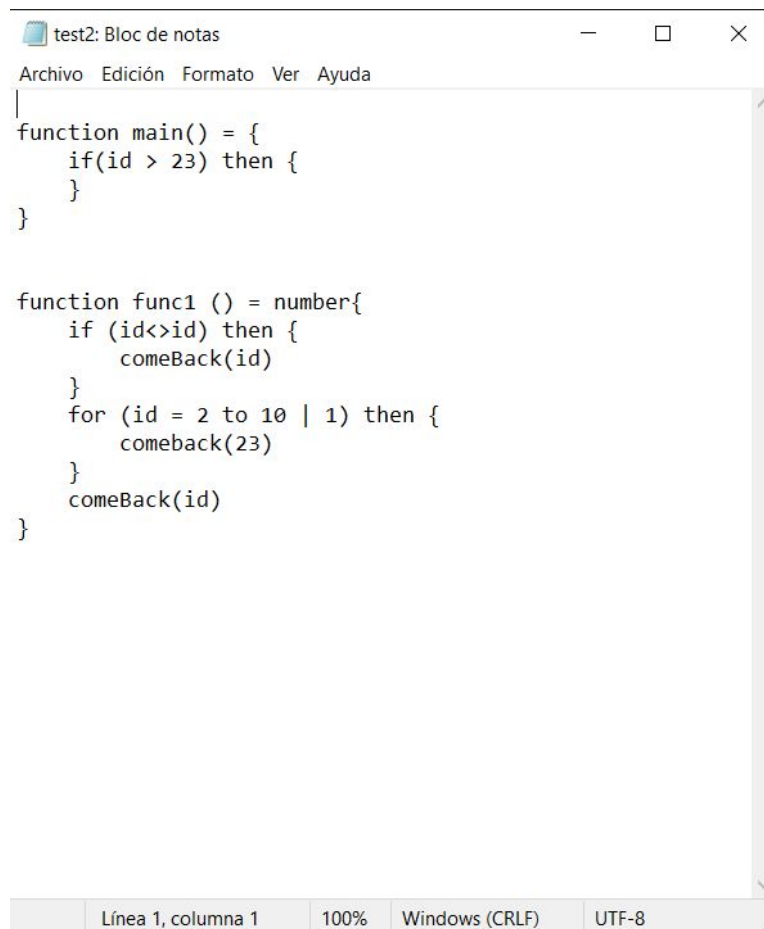
`boolean == boolean, boolean != boolean`

Ejemplo:

```
function main () = {  
    number num = 3  
    number num2 = 5  
    number num3 = 0  
    char a = 'a'  
    array[number] lista = [1,2,3,4]  
  
    if (5 < num) then {  
        output(num2)  
    } else if (5 > num) {  
        call imprimir()  
    } else {  
        output(num3)  
    }  
}  
  
function imprimir () = {  
    bool verif = true  
    output(verif)  
}
```

Uso de la interfaz gráfica del compilador

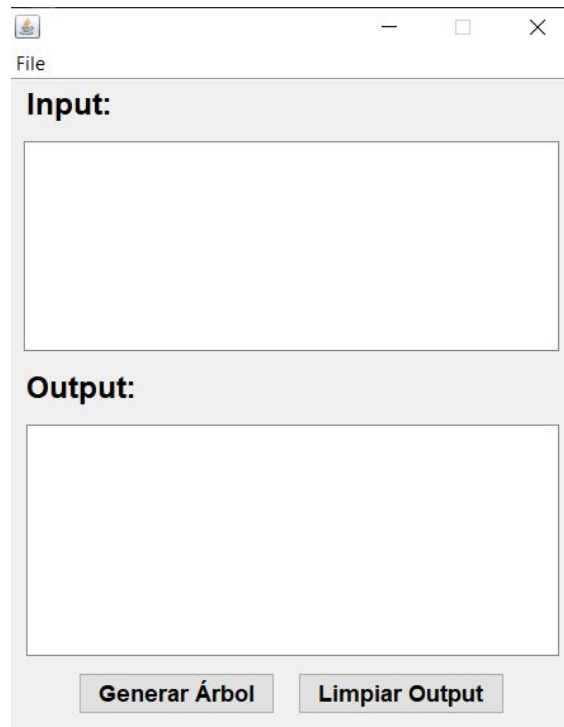
Paso #1: Crear un archivo txt donde se pueda escribir un pequeño programa para compilar.



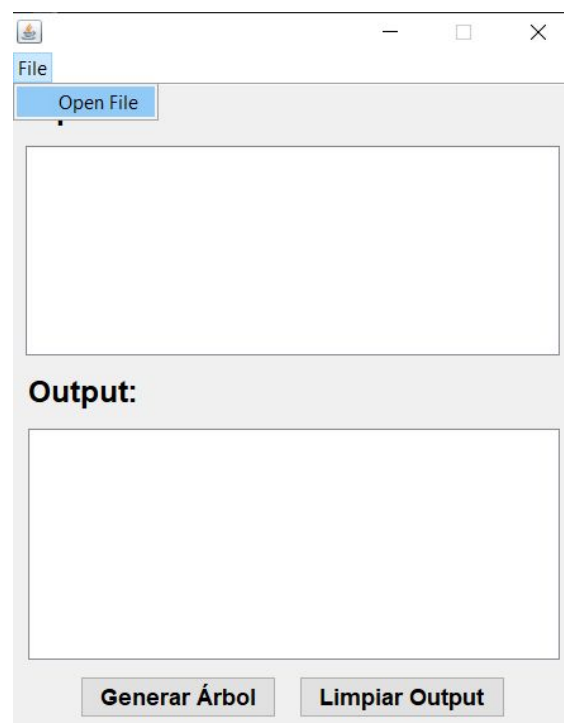
```
function main() = {  
    if(id > 23) then {  
    }  
}  
  
function func1 () = number{  
    if (id<>id) then {  
        comeBack(id)  
    }  
    for (id = 2 to 10 | 1) then {  
        comeback(23)  
    }  
    comeBack(id)  
}
```

The screenshot shows a Notepad window with the title 'test2: Bloc de notas'. The menu bar includes 'Archivo', 'Edición', 'Formato', 'Ver', and 'Ayuda'. The code is written in a Pascal-like syntax. The status bar at the bottom indicates 'Línea 1, columna 1', '100%', 'Windows (CRLF)', and 'UTF-8'.

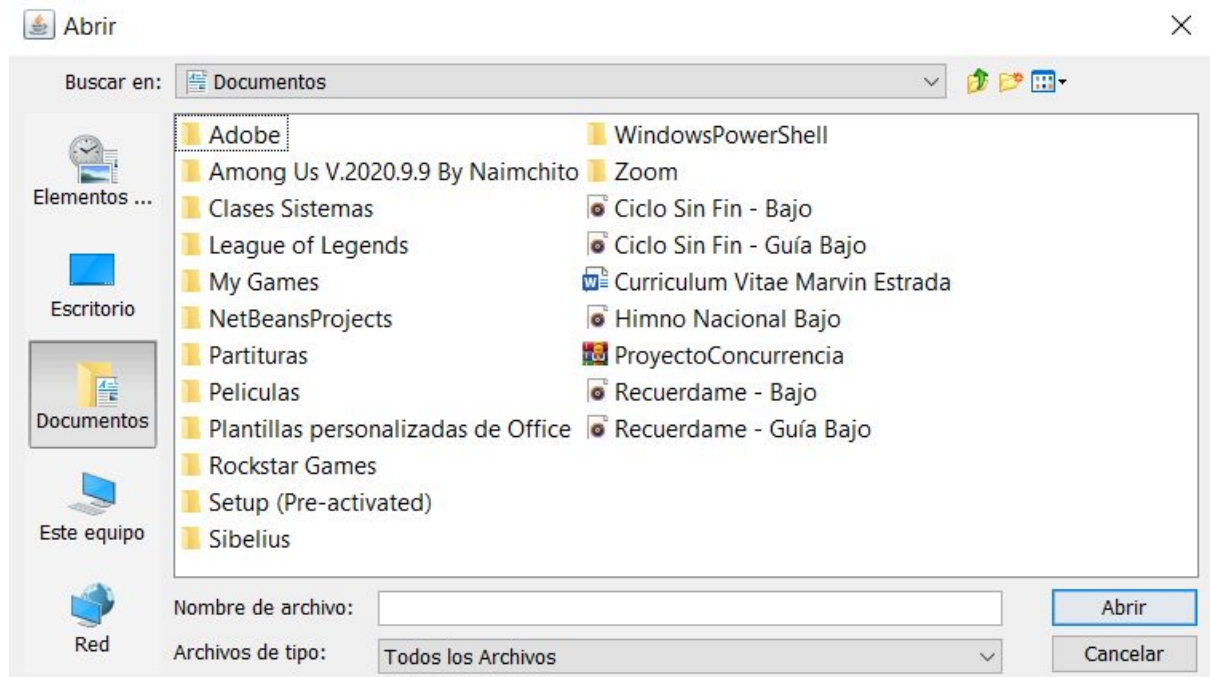
Paso #2: Abrir el GUI del programa.



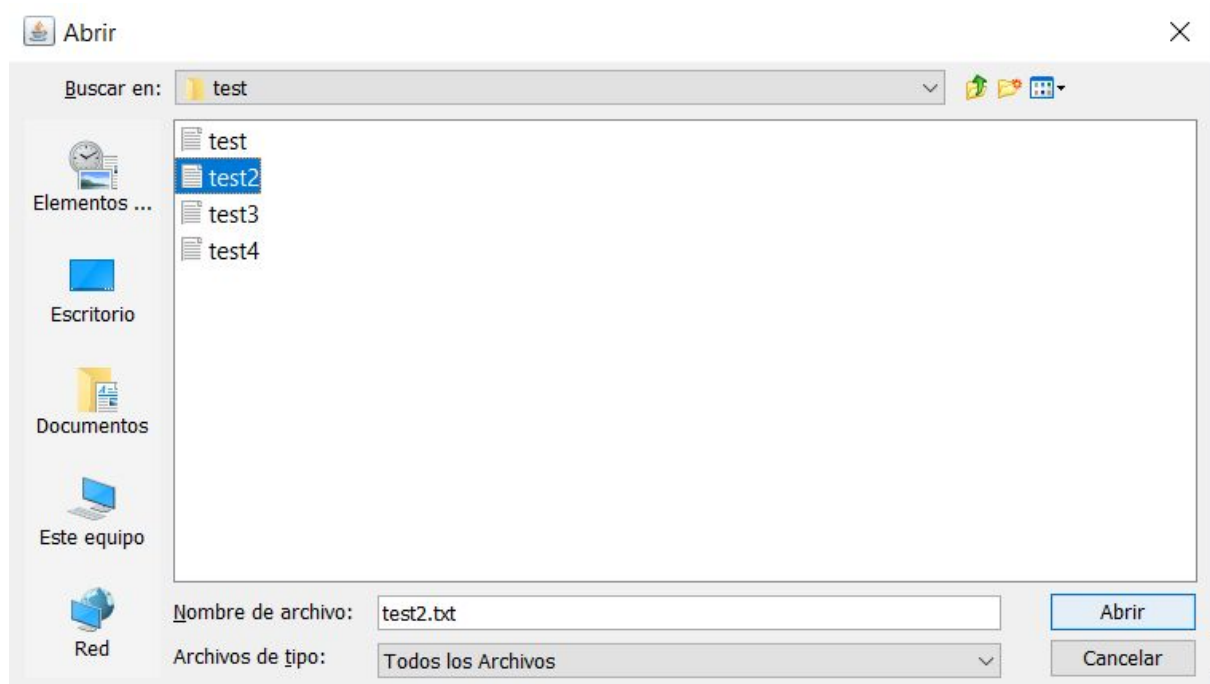
Paso #3: Hacer click en la pestaña “File” y luego en el submenú “Open File” para seleccionar el archivo previamente creado.



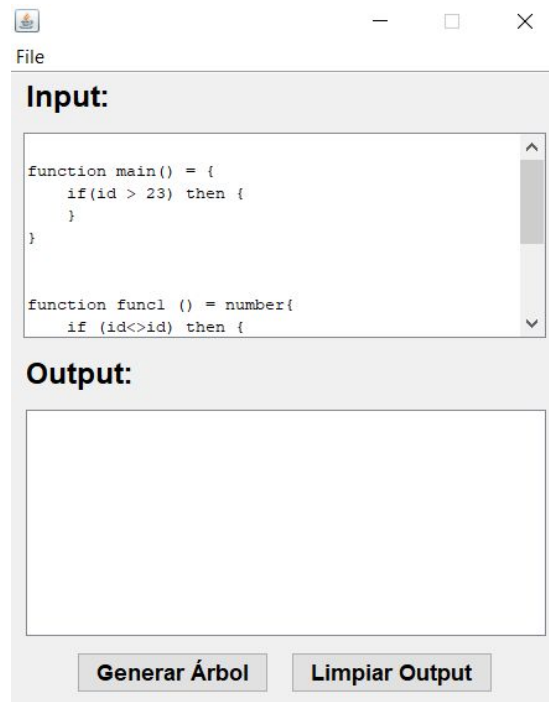
Paso #4: Cuando se abra el explorador de archivos, buscar el archivo previamente creado.



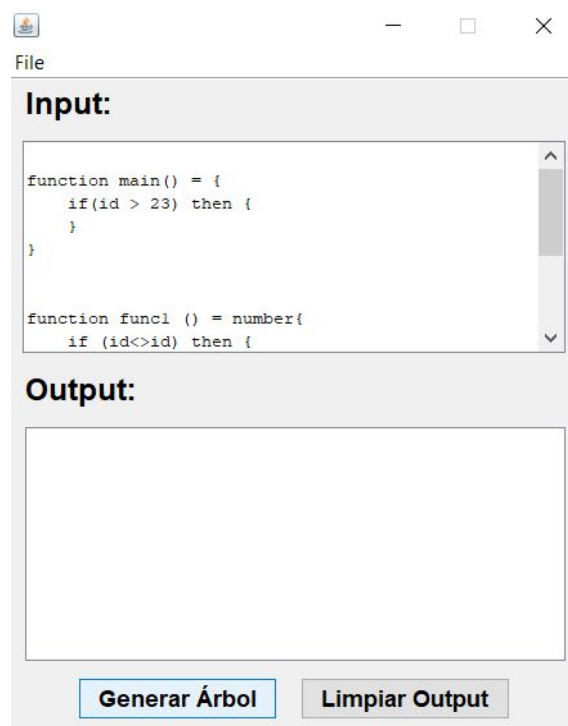
Paso #5: Hacer doble click en el archivo txt que desea abrir o seleccionar el archivo txt y luego hacer click en “Abrir”.



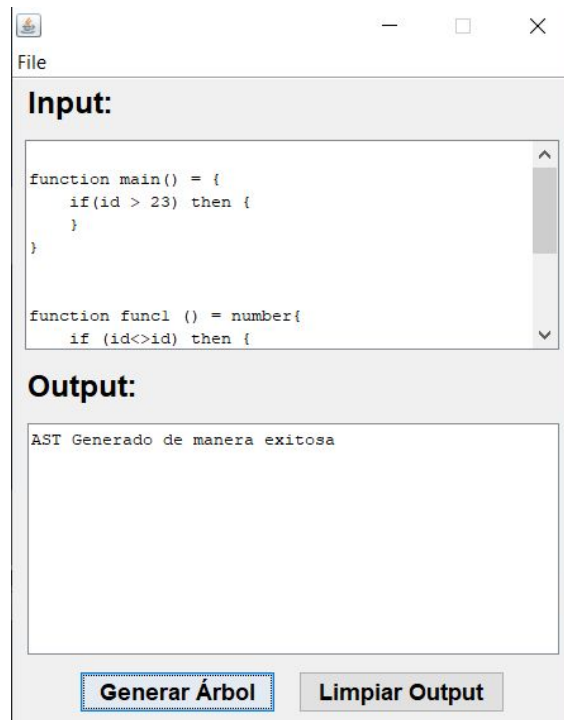
Paso #6: Verificar que el archivo correcto se ha abierto. El pequeño programa aparecerá en el cuadro de texto de “Input:” y se podrá ver todo lo contenido en el archivo.



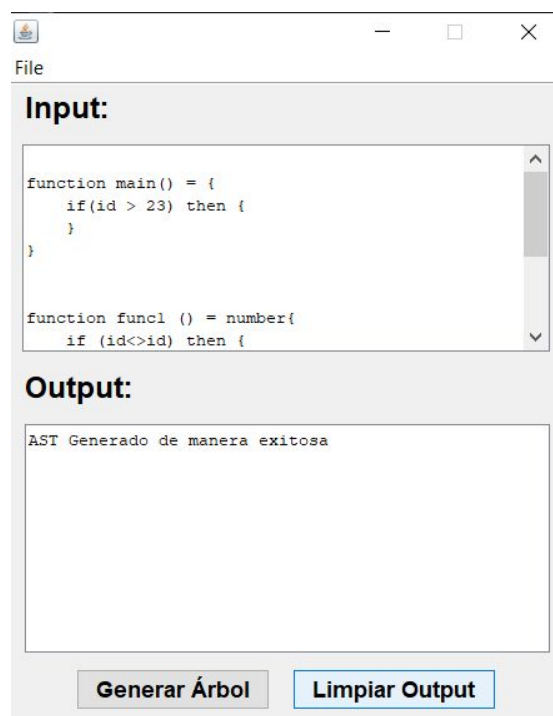
Paso #7: Cuando el archivo esté cargado, hacer click en “Generar Árbol” para correr el pequeño programa y verificar si su sintaxis está correctamente implementada.



Paso #8: Observar cuando el programa se haya compilado. Aparecerá el texto de compilación en el cuadro de texto “Output”.



Paso #9: Para limpiar el cuadro de texto “Output” hacer click en el botón de “Limpiar Output”.



Paso #10: Observar que el cuadro de texto “Output” ha limpiado el texto generado por compilación.

