Proyecto Intérprete

Gutiérrez Esparza, Marquina Cancino

Noviembre 2020

Índice

1. Requerimientos

El intérprete debe de ser capaz de realizar lo siguiente:

- Declarar variables de los siguientes tipos:
  - Entero
  - Flotante
  - Cadena de caracteres
  - Booleano
- Asignar valores a variables (usando el operador `:=`)
- Evaluar expresiones matemáticas. Operaciones soportadas:
  - Suma (`+`)
  - Resta (`-`)
  - Multiplicación (`*`)
  - División (`/`)
- Comparar expresiones usando los siguientes operadores:
  - Menor que (`<`)
  - Mayor que (`>`)
  - Igual que (`==`)
  - Menor o igual que (`<=`)
  - Mayor o igual que (`>=`)
  - Diferente que (`!=`)
- Implementar los siguientes operadores booleanos:
  - "y" lógico (`&&`)
  - "o" lógico (`||`)
  - "no" lógico (`!!`)
- Soportar condicionales y secuencias de control
  - `while`
  - `for`
  - `if`/`else`
  - `break`
- Soportar los siguientes valores:
  - Todos los números de 64 bits
  - Caracteres
  - Valores booleanos (`0, 1`)
- Imprimir y leer de consola

2. Reglas de sintaxis

Una línea del programa debe de contener **una y sólamente una** de las siguientes cuatro instrucciones:

1. Declaraciones
2. Asignaciones
3. Secuencias de control
4. Imprimir y leer
5. Comentario

2.1. Declaración

Los elementos requeridos en una declaración son:

1. Tipo de variable
2. Asignación (ver abajo)

2.2. Asignación

Los elementos requeridos en una asignación son:

1. Nombre de variable
2. Operador Asignación (`:=`)
3. Expresión(Matemática, Comparación, Operadores booleanos, Valor)

Es necesario checar que el resultado de la expresión coincida con la variable.
Las reglas para los nombres de las variables son:

- Los nombres de las variables están compuestos únicamente por letras del alfabeto inglés y dígitos.
- El nombre no puede empezar con un dígito.
- El nombre no puede ser una palabra reservada.
- El intérprete distingue mayúsculas y minúsculas

```
int a := 14
my_string := "hello world!"
bool flag := a || b

#Ejemplos de nombres invalidos
3pointer, a$af, break, algo.mod
```

2.3. Secuencias de control

A continuación, se presenta una descripción de las secuencias de control utilizadas en este proyecto. Algunas consideraciones generales:

- Los **elementos evaluables** por las secuencias de control se refiere a expresiones, valores o variables.
- Para todas las secuencias de control se utilizan los corchetes ("`{`","`}`") para delimitar su alcance.

2.3.1. If/else

La secuencia de control `if`/`else` es utilizada como condicional. La sintaxis es la siguiente

```
if elemento_evaluable {
    #Hacer algo
}
else {
    #Hacer algo mas
}
```

Se puede utilizar un `for` sin utilizar el `else`, pero no al revés.

2.3.2. While

La secuencia de control `while` es utilizada para ejecutar un bloque de código hasta que alguna condición sea falsa. La sintaxis es la siguiente

```
while elemento_evaluable {
    #Hacer algo
}
```

Dentro de un while se puede colocar la sentencia `break`, que sirve para romper el ciclo en ejecución.

2.3.3. For

La secuencia de control `for`/`to` se utiliza para iterar. La sintaxis es la siguiente

```
int iterador
for iterador := valor_inicio to valor_final {
    #Hacer algo
}
```

El `for` toma el valor asignado al iterador e itera desde su valor hasta llegar a un valor final (sin incluirlo). Si el `valor_inicio` es mayor que el `valor_final`, entonces contará hacia abajo. El iterador **siempre** tiene que ser declarado fuera del `for` y reasignado al comenzar el `for`. Dentro de un `for` también se puede colocar la sentencia `break` para romper el ciclo en ejecución. Ejemplos:

```
int i
for i := -1 to 4 {
    Print(i)
}
#Imprime: -1 0 1 2 3
for i := 5 to 0 {
    Print(i)
}
#Imprime: 5 4 3 2 1
for i := 1 to 100 {
    Print(i*2)
    if i == 4 {
        break
    }
}
#Imprime: 2 4 6 8
```

2.4. Leer

La función `Read()` es utilizada para lectura. `Read()` devuelve el contenido de la siguiente línea de la entrada estándar, por ello, es necesario usar la función en conjunto con una asignación o declaración.
Ejemplo:

```
int x := Read()
string s := Read()
```

2.5. Imprimir

La función `Print()` es utilizada para imprimir a la salida estándar. La sintaxis es la siguiente

```
Print(elemento_evaluable)
```

2.6. Comentarios

Para comentar una línea del programa, se utiliza el caracter `#`. Los comentarios multilínea no están soportados.

```
#Esto es un comentario, y puedo escribir lo que sea
Esto no es un comentario. Si se trata de compilar daria error

int a := 33 #Comentar de esta manera tambien funciona

#Para comentarios de varias lineas
#Hay que usar varios simbolos
```

2.7. Ejemplo de un programa

```
#Programa que obtiene el promedio de las calificaciones de n estudiantes
int n := Read()
float sum := 0
int cont := 0

for cont := 0 to n {
    float calif := Read()
    sum := sum + calif
}

Print(sum/n)
```

3. Tabla Léxica

Símbolo o palabra clave	Significado	Ejemplo	Valor
`int`	Declarar Entero	`int x:= 2`	201
`float`	Declarar Flotante	`float x := 3`	202
`string`	Declarar Cadena	`string x := hello`	203
`bool`	Declarar Booleano	`bool x := True`	200
`True`	1, Verdadero	`bool x := True`	300
`False`	0, Falso	`bool x := False`	301
`#`	Comentario	`#Comentario`	ASCII
`"`	Delimitador de cadena	`string x := "hello"`	ASCII
`:=`	Asignación	`x := 2`	400
`*`	Multiplicación	`x := 2 * 3`	ASCII
`/`	División	`x := 2 / 1`	ASCII
`+`	Suma	`x := 2 + 1`	ASCII
`-`	Resta	`x := 2 - 1`	ASCII
`<`	Menor que	`bool x := 2 < 4`	ASCII
`>`	Mayor que	`bool x := 2 > 5`	ASCII
`<=`	Menor o igual que	`bool x := 2 <= 2`	500
`==`	Igual	`bool x := 2 == 2`	502
`>=`	Mayor o igual que	`bool x := 2 >= 3`	501
`!=`	Diferente que	`if x != 2 >= 3`	503
`&&`	AND lógico(entre dos booleanos)	`bool x := True && False`	600
`||`	OR lógico	`bool x := True || False`	601
`!!`	NOT lógico	`bool x := !! False`	602
`()`	Encapsular Operaciones Matemáticas	`int x := ( 2 + 3 )* 4`	ASCII

`break`	Salir de ciclo	`while True {`
		`  break`
		`}`
			700

`while`	Iniciar ciclo	`while x {`
		`  break`
		`}`
			701

`if`	Checar condición	`if x {`
		`  x := False`
		`}`
			702

`else`	Hacer algo si la condición del `for` no se cumple	`if x {`
		`  x := False`
		`}`
		`else {`
		`  x := True`
		`}`
			703

`for`	Iterar	`int x := 0`
		`for x := 0 to 14 {`
		`  sum := sum + x`
		`}`
			704

`to`	Iterar	`int x := 0`
		`for x := 0 to 14 {`
		`  sum := sum + x`
		`}`
			705

`{}`	Delimitar el alcance del `while`, `if`	`if x {`
		`  x := False`
		`}`
			ASCII

`Read()`	Devuelve el siguiente valor de consola, delimitado por espacios	`int x := Read()`	800
`Print()`	Imprimir valor dentro de paréntesis	`Print(x)`	801