

Proyecto Final EDAlI: Intérprete

Gutiérrez Esparza, Marquina Cancino

Noviembre 2020

Resumen

Este proyecto final para la materia «Estructuras de Datos y Algoritmos II» consiste en un intérprete de instrucciones. El programa compila y ejecuta un archivo de texto que contiene instrucciones a manera de lenguaje de programación. En este documento se define la sintaxis del programa y se muestran las normativas generales que puede entender el intérprete.

Índice

1. Requerimientos	2
2. Reglas de sintaxis	3
2.1. Declaración	3
2.2. Asignación	3
2.3. Secuencias de control	4
2.3.1. If/else	4
2.3.2. While	4
2.3.3. For	4
2.4. Leer	5
2.5. Imprimir	5
2.6. Comentarios	5
2.7. Ejemplo de un programa	6
3. Tabla Léxica	7
4. Errores	8
4.1. Errores en el análisis léxico	8
4.2. Errores en el análisis sintáctico	9
4.3. Errores en el análisis semántico	9
4.4. Errores en la evaluación de expresiones	9

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
- 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 solamente 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, while, 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  
}
```

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`. 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
```

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 daría error

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

#Para comentarios de varias líneas
#Hay que usar varios símbolos
```

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
float calif := 0

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

Print(sum/n)
```

3. Tabla Léxica

Símbolo o palabra clave	Significado	Ejemplo	Valor
<code>int</code>	Declarar Entero	<code>int x := 2</code>	201
<code>float</code>	Declarar Flotante	<code>float x := 3</code>	202
<code>string</code>	Declarar Cadena	<code>string x := hello</code>	203
<code>bool</code>	Declarar Booleano	<code>bool x := True</code>	200
<code>True</code>	1, Verdadero	<code>bool x := True</code>	300
<code>False</code>	0, Falso	<code>bool x := False</code>	301
<code>#</code>	Comentario	<code>#Comentario</code>	ASCII
<code>"</code>	Delimitador de cadena	<code>string x := "hello"</code>	ASCII
<code>:=</code>	Asignación	<code>x := 2</code>	400
<code>*</code>	Multiplicación	<code>x := 2 * 3</code>	ASCII
<code>/</code>	División	<code>x := 2 / 1</code>	ASCII
<code>+</code>	Suma	<code>x := 2 + 1</code>	ASCII
<code>-</code>	Resta	<code>x := 2 - 1</code>	ASCII
<code><</code>	Menor que	<code>bool x := 2 < 4</code>	ASCII
<code>></code>	Mayor que	<code>bool x := 2 > 5</code>	ASCII
<code><=</code>	Menor o igual que	<code>bool x := 2 <= 2</code>	500
<code>==</code>	Igual	<code>bool x := 2 == 2</code>	502
<code>>=</code>	Mayor o igual que	<code>bool x := 2 >= 3</code>	501
<code>!=</code>	Diferente que	<code>if x != 2 >= 3</code>	503
<code>&&</code>	AND lógico(entre dos booleanos)	<code>bool x := True && False</code>	600
<code> </code>	OR lógico	<code>bool x := True False</code>	601
<code>!!</code>	NOT lógico	<code>bool x := !! False</code>	602
<code>()</code>	Encapsular Operaciones Matemáticas	<code>int x := (2 + 3) * 4</code>	ASCII
<code>while</code>	Iniciar ciclo	<pre>while x { x := False }</pre>	701
<code>if</code>	Checar condición	<pre>if x { x := False }</pre>	702

<code>else</code>	Hacer algo si la condición del <code>for</code> no se cumple	<pre> if x { x := False } else { x := True } </pre>	703
<code>for</code>	Iterar	<pre> int x := 0 for x := 0 to 14 { sum := sum + x } </pre>	704
<code>to</code>	Iterar	<pre> int x := 0 for x := 0 to 14 { sum := sum + x } </pre>	705
<code>{}</code>	Delimitar el alcance del <code>while</code> , <code>if</code>	<pre> if x { x := False } </pre>	ASCII
<code>Read()</code>	Devuelve el siguiente valor de consola, delimitado por espacios	<pre> int x := Read() </pre>	800
<code>Print()</code>	Imprimir valor dentro de paréntesis	<pre> Print(x) </pre>	801

4. Errores

Durante la compilación o ejecución del programa, los siguientes errores pueden ocurrir.

4.1. Errores en el análisis léxico

1. **No se pudo abrir el archivo:** El programa no encontró el archivo especificado o no tiene permiso para leerlo.

2. **Token imprevisto:** El programa recibió un token que no reconoce. Ejemplo: `a = b`, puesto que se necesita usar el símbolo `:=` para señalar asignación.

4.2. Errores en el análisis sintáctico

1. **Error sintáctico:** Puede ocurrir si alguna instrucción está mal planteada o le faltan elementos. Ejemplo:

```
for i := 9{  
    #Hacer algo  
}  
#Esto da error, puesto que el for no esta completamente definido
```

4.3. Errores en el análisis semántico

1. **Asignación entre tipos no compatibles:** Ocurre cuando se trata de asignar un valor a una variable con tipos incompatibles. Ejemplo: `float x := "Hola"`.
2. **El tipo "variable" no existe:** Ocurre si se trata de declara una variable de un tipo distinto a los aquí considerados (`int`, `float`, `bool`, `string`). Cabe destacar que este error debería de ser manejado por el análisis sintáctico.
3. **Variable no declarada:** Ocurre cuando se trata de acceder a una variable que no ha sido previamente declarada.
4. **Doble declaración de variable:** Ocurre cuando se trata de volver a declarar una variable previamente declarada. Cabe destacar que todas las variables son globales, por lo que el scope no las afecta (una variable declarada dentro de un `if/else` será válida fuera de él).
5. **Error de ejecución:** Si el programa incurre en un error de ejecución, lo reporta. Podría ocurrir que un error de ejecución se de y no pase por este sistema, esto es por la forma en la que se manejan este tipo de errores en C++.

4.4. Errores en la evaluacion de expresiones

1. **Los paréntesis no coinciden:** Error que se da cuando, dentro de una expresión, hay paréntesis que no abren o cierran. Ejemplo:

```
int a := 12*(134-2)) #Falta parentesis que abre  
float y := 32-(19 #Falta parentesis que cierra
```

2. **Expresion invalida:** Si lo que se dicta como una expresión es inválida, este error ocurre. Se da cuando se tienen cosas que no son expresiones, como `14-1 1`, o cuando se tratan de hacer operaciones con `string`.