Lenguajes de Programación, 2017-2

Práctica 8: Orientación a Objetos

Karla Ramírez Pulido José Ricardo Rodríguez Abreu
Manuel Soto Romero

Fecha de inicio: 19 de mayo de 2017
Fecha de término: 5 de junio de 2017

Objetivos

- Implementar un intérprete para el Lenguaje de Programación Forth[1] usando un lenguaje orientado a objetos aprovechando el uso de clases, herencia y polimorfismo.

Desarrollo de la práctica

Forth es un lenguaje de programación basado en una pila que se compone de una lista de comandos básicos, aunque dicha lista puede ir creciendo conforme el programa se ejecuta.

Lista de comandos:

`exit`	Termina la ejecución del programa.
`help c`	Si no se pasa el parámetro `c`, muestra la lista de comandos actual con su descripción; en otro caso, muestra información sobre el comando `c` pasado como parámetro.
`+,-,*,/`	Aplican operaciones aritméticas de la siguiente manera: se sacan los dos valores del tope de la pila, se aplica la operación y se regresa el resultado a la pila.
`drop`	Elimina el elemento del tope de la pila.
`dup`	Duplica el elemento del tope de la pila.
`over`	Duplica el elemento en el tope de la pila, pero almacena el resultado como el tercer elemento en la misma.
`swap`	Intercambia las posiciones de los dos últimos elementos de la pila.
`push n`	Pone un número `n` en el tope de la pila.
`n`	Pone el número `n` en el tope de la pila. Es azúcar sintáctica de `push n`.
`show`	Muestra la pila actual.
`: id c+`	Define un nuevo comando con el nombre `id` como la composición de los comandos existentes `c+`. Después, si se escribe el `id` en el intérprete, se ejecutarán los comandos especificados en `c+` en orden. Una vez definido este comando `help` debe poder listarlo.

Ejercicios

Se debe implementar un intérprete para el lenguaje de programación Forth que se compone de los comandos descritos anteriormente. La gramática del lenguaje en EBNF se muestra a continuación[1]:

```
<expr> ::= <command> | <definition>

<command> ::= exit
            | help <command>*
            | <op>
            | drop
            | dup
            | over
            | swap
            | push <num>
            | <num>
            | show

<definition> ::= : <id> <command>+

<id> := b | .. | z | A | ... | Z | aa | ab | ... | aaa | ...
        (Cualquier combinación de caracteres alfanuméricos
              con al menos uno alfabético)

<num> ::= ... | -2 | - 1 | 0 | 1 | 2 | ...

<op> ::= + | - | * | /
```

Ejemplo. Ejecutar el siguiente programa debe producir la pila con los elementos `[4, 2]` (donde el elemento 4 se encuentra en el tope de la pila). Todos los programas de Forth regresan siempre la pila final.

```
push 2
push 2
+
dup
2
swap
-
swap

: incr 1 +
incr
incr
+
```

[1]Del inglés Extended Backus–Naur Form

La implementación se debe realizar un lenguaje orientado a objetos (el lenguaje utilizado podrá ser el que prefiera el equipo)[2]. Para la implementación, se deben incluir, al menos:

1. (2 pts.) Una clase `Lexer` que realice el análisis léxico de un programa.

2. (3 pts.) Una clase `Parser` que realice el análisis sintáctico de un programa[3].

3. (3 pts.) Una clase `Forth` que realice el análisis semántico y la ejecución del intérprete ante el usuario[4]. El intérprete debe poder recibir cadenas desde la terminal (interactuando con el usuario) o leerlas desde un archivo.

4. (2 pts.) Además se debe incluir un manual de usuario dónde se explique cómo compilar y ejecutar el intérprete y explicar **detalladamente** qué ventajas y desventajas se encontraron al implementar el intérprete usando un lenguaje orientado a objetos. Cada clase debe ir debidamente documentada, explicando de forma clara y detallada el uso de cada método mediante comentarios.

Puntos extra

1. (2 pts.) Como ejercicio extra se puede diseñar y programar una interfaz gráfica para el intérprete, se tomará en cuenta la creatividad.

2. (2 pts.) Definir un macro que genere la sintaxis necesaria para dar el comportamiento a una estructura de repetición `for-each`. El macro `for-each` debe tener la siguiente forma:

   ```
   (for-each t x in e do cuerpo)
   ```

   - El parámetro `t` representa el tipo que debe tener al elemento a iterar.
   - El parámetro `x` representa el identificador de cada elemento de la estructura.
   - El parámetro `e` representa la estructura que contiene a los elementos `x` de tipo `t`.
   - El parámetro `cuerpo` representa las instrucciones que se deben ejecutar en cada iteración. El número de instrucciones puede ser uno o más.

   Nota: Los símboloss `do` e `in` son palabras reservadas de la estructura `for-each`.

   Ejemplo:

   ```
   > (define lista '(1 2 3 4 5))
   > (for-each number? x in lista do (display (number->string x)))
   1
   2
   3
   4
   5
   ```

[2]Si se escoge un lenguaje multiparadigma, preguntar primero al ayudante de laboratorio
[3]Se sugiere que la representación intermedia sean árboles de sintáxis abstracta y aprovechar el mecanismo de herencia.
[4]Esta es la clase principal

Referencias

[1] Ben Greenman, *Forth*, En la Web. Consultado el 11 de mayo de 2017. <http://docs.racket-lang.org/forth/index.html>.