Gustavo Gutierrez Navarro *14/08/2023*

Revisión: Mario Alejandro Castro Lerma

Presenta tres programas del lenguaje usado actualmente que ilustren los siguientes puntos:

- Si la variable es declarada en el bloque actual, usamos esta declaración.

Primer programa:

Código:

```
(deffun (suma x)
        (defvar y 2)
        (+ x y))

(deffun (resta x)
        (defvar y 1)
        (- x y))

(suma 3)
(resta 1)
```

Resultado:
5 0

Explicación:
A pesar de que se usen variables con el mismo nombre, cada bloque es independiente del otro. El bloque de suma usa x – 3 & y – 2 mientras que resta x – 1 & y – 1.

Es correcto, cada bloque utiliza su “y” independientemente de la otra que se encuentra en el programa.

Segundo programa:

Código:

```
(defvar x 1)
(deffun (aux)
        (defvar x 5)
        x)
(aux)
x
```

Resultado:
5 1

Explicación:
A pesar de existir dos declaraciones de una variable con el mismo nombre (x) y que una este descrita en el bloque primordial (Primer bloque del programa el cual engloba todo el algoritmo), la manera en la que funciona el lenguaje es siempre dar prioridad a las variables locales, es decir una variable con el mismo nombre siempre se impondrá o hará sombra a otra variable con el mismo nombre que provenga de un bloque padre.

Es correcto, cada variable utiliza su valor definido en su respectivo bloque, la función aux utiliza x = 5, mientras que el bloque superior utiliza x = 1.

Tercer programa:

Código:

```
(deffun (first)
        (defvar x 3)
        (deffun (second)
                (defvar x 2)
                (deffun (first)
                        (defvar x 1)
                        x)
                (first)
                x)
        (second)
        x)
(first)
```

Resultado:

3

Explicación:

La ultima variable que sale de la llamada a función (se podría ver como el return de la función) es la variable x en donde su valor corresponde a 3 según el bloque donde se encuentra (el bloque de la función first).

Es correcto, ya que, aunque existen múltiples asignaciones a una variable llamada "x", en ningún momento hay ambigüedad ya que estas se encuentran contenidas en sus propios bloques y finalmente se utiliza la "x" de la función en el bloque más superior.

- De lo contrario, buscamos en el bloque donde aparece el bloque actual, continuando recursivamente.

Primer programa:

Código:

```
(defvar x 99)
(deffun (main)
        (defvar y 66)
        x)
(main)
```

Resultado:

99

Explicación:

La variable x no se encuentra dentro de la función, por lo tanto, lo busca afuera de esta y lo encuentra en su bloque padre con el valor de 99.

Es correcto, ya que la función "main" no encuentra la variable "x" en su propio bloque y se dirige al bloque superior para buscarla.

Segundo programa:

Código:

```
(defvar z 9)
(deffun (resta x y)
        (defvar z 6)
        (- x y))
(resta 5 2)
z
```

Resultado:
3 9
Explicación:
El primer resultado viene de las variables pasadas a la función como parámetro mientras que el z proviene del bloque de afuera.
Es correcto, ya que la función "resta" busca la variable "z" en el bloque superior por que no se encuentra una variable "z" en el bloque de la función.

Tercer programa:

Código:

```
(defvar z 4)
(deffun (first)
        (defvar x 3)
        (deffun (second)
                (defvar y 2)
                (deffun (first)
                        (defvar z 1)
                        y)
                (first)
                z)
        (second)
        z)
(first)
```

Resultado:
4
Explicación:
La búsqueda de las variables siempre se da para arriba, es decir buscando en los bloques padres, y nunca en los inferiores o hijos, en este caso el bloque padre corresponde a z – 4 mientras que del lado de los hijos es z – 1.
Es correcto, ya que la función "first" hace referencia a la variable "z" la cual se encuentra en el bloque superior y no dentro de la función.

- Si el bloque actual ya es el bloque primordial y aun no hemos encontrado una declaración correspondiente, la variable no está ligada.

Primer programa:

Código:

```
(deffun (main) x)
(main)
(defvar x 5)
```

Resultado:
error
Explicación:
En la función se intenta llamar a la variable x que a pesar de estar reservada en memoria, esta al no tener todavía un valor asignado, provoca que el programa explote.
Es correcto, el momento en que se llama la variable x, esta no esta ligada a ningún valor y por lo tanto da error.

Segundo programa:

Código:

```
(deffun (main)
        (defvar x 5))
(main)
x
```

Resultado:
error
Explicación:
Como se mencionó anteriormente, la búsqueda de variables va hacia el padre, por lo tanto al estar la variable x asignada dentro de un bloque hijo, provoca que la variable x no exista en el ambiente del bloque primordial.
Es correcto, ya que al llamar a "x" esta no está definida en el bloque superior.

Tercer programa:

Código:

```
(deffun (first)
        (deffun (second)
                (defvar x 5)
        x))
(first)
```

Resultado:
error
Explicación:
Es el mismo caso que el anterior programa, pero visto desde funciones.
Es correcto, funciona igual que el código anterior

- Cada *defvar* evalúa la expresión inmediatamente y vincula la variable al valor, incluso si la variable no es usada en el resto del programa.

Primer programa:

Código:

Simplemente evalúa las expresiones en un orden de arriba hacia abajo y de izquierda a derecha.
Es correcto, ya que se evalúa las operaciones primero para vincular la variable a el valor.

Segundo programa:

Código:

```
(defvar x 5)
(defvar y (/ x 0))
x
```

Resultado:
error
Explicación:
A pesar de que la variable y no se usa en ningún momento, al momento de asignar el valor se evalúa la expresión, en este caso la expresión resulta ser un error ya que se intenta dividir por 0, esto provoca que explote el programa y no se pueda terminar de ejecutar correctamente.

Es correcto, porque la vinculación de variables se realiza primero y en este momento se realiza la operación pero causa error.

Tercer programa:

Código:

```
(defvar z 5)
(deffun (aux x)
        (defvar y (/ x 0)))
z
```

Resultado:
5
Explicación:
Si la variable nunca alcanza la línea de asignación, nunca se ejecutará la operación dentro de la misma, por lo tanto no causara error (en este caso).
Es correcto, porque no se llega en ningún momento al bloque de la función “aux”.

- Cada llamada a función evalúa los argumentos inmediatamente y vincula los valores a los parámetros formales, incluso cuando los parámetros formales no son usados en la función.

Primer programa:

Código:

```
(defvar x 2)
(deffun (main y)
    (+ y 6))

(main (/ 2 0))
```

Resultado:
error
Explicación:
El programa evalúa la operación dentro de la llamada a función para posteriormente enviar este resultado como parámetro, sin embargo al resultar en un error de división entre 0, provoca una explosión y el programa se termina abruptamente, sin leer el contenido de la función.
Es correcto, al llamar a la función ocurre el error de intentar dividir entre 0.

Segundo programa:

Código:

```
(defvar x 5)
(defvar y (+ 5 4))
(deffun (main a b)
  (* a b))

(main y (- x y))
```

Resultado:
-36
Explicación:
Simplemente se siguen el orden de las operaciones, primero realizando los parámetros establecidos.
Es correcto, se evalúan los parámetros en orden de cómo son asignados y llamados.

Tercer programa:

Código:

```
(deffun (main x) x)

(main (- (* 2 5) (/ 10 2)))
```

Resultado:
5
Explicación:
Antes de mandar a llamar la función , primero se determina el valor que será mandado, esto implica evaluar las operaciones.
Es correcto, ya que se evalúan los parámetros.

- Una secuencia de *defvar* y *deffun* vincula las variables a los valores de arriba hacia abajo y de izquierda a derecha.
- Es un error evaluar una variable antes de vincularla a un valor.

Primer programa:

Código:

```
(defvar x 5)
(deffun (main)
    (+ x y))
(defvar y 2)
(main)
```

Resultado:
7
Explicación:
La función main está justo por debajo de la asignación de la variable y, por lo tanto, al ejecutar la función esta misma puede utilizar tanto a la variable x & y por que ya se han leído sus líneas de comando.
Es correcto, ya que se realizan las operaciones de arriba hacia abajo del programa.

Segundo programa:

Código:

```
(defvar x 2)
(deffun (main)
    (+ x y))
(main)
(defvar y 1)
```

Resultado:
error
Explicación:
En este caso, la asignación ocurre justo debajo una línea de la llamada al a función por lo tanto esta última no tendrá conocimiento del valor de y por qué en la ejecución del programa todavía no se ha llegado a leer esa línea de comando,

Es correcto, ya que se vincula la "y" después de que es llamada en el programa, mostrando que el orden de lectura del programa es de arriba hacia abajo.

Tercer programa:

Código:

```
(defvar w 4)
(deffun (main x)
    (deffun (aux y)
        (+ y 5))
    (defvar z (aux x))
    z)
(main w)
```

Resultado:
9
Explicación:

Simple lectura de programa de izquierda a derecha y de arriba hacia abajo.
Es correcto, porque sigue la lectura de arriba hacia abajo y de izquierda y derecha.