Proyecto Final

Nicolas Mauricio Rojas Mendoza - 2259460

Juan David Rojas Narvaez - 2259673

Jhonny Fernando Duque Villada - 2259398

21 de junio de 2022

1. Implementaciones realizadas en el lenguaje de programación

1.1. Especificación del lenguaje

- Números decimales
- Números flotantes
- Textos
- Listas
- Arreglos

1.2. Primitivas del lenguaje

- Primitivas numéricas
- Primitivas booleanas
- Primitivas de listas
- Primitivas de arrays
- Primitivas de cadenas

1.3. Estructuras de control

- Estructura if

2. Explicación de lo implementado

2.1. Números decimales, flotantes y primitivas numéricas

- Los números decimales están representados en la léxica como cualquier cantidad de dígitos seguidos unos de otros, así mismo su representación negativa empieza por un símbolo negativo "seguido de cualquier cantidad de dígitos.

- Los flotantes se representan como cualquier cantidad de dígitos seguido de un punto seguido de cualquier cantidad de dígitos y su versión negativa empieza con un símbolo negativo y sigue la misma estructura explicada anteriormente.

- Nota: A pesar de que el lenguaje no puede realizar operaciones numéricas con los números binarios,octales y hexadecimales si permite la entrada de los mismos asignando su correspondiente tipo de dato (octal,hexadecimal,binario) y su representación léxica consiste de la siguiente manera: Números binarios: Empieza por una "b"seguido de un 0 o 1 seguido de cualquier cantidad de ceros y unos, su versión negativa empieza con un menos y sigue la misma estructura. Números Octales: Empiezan por un "0x"seguido de un numero que puede ir del 1 hasta el 7 seguido de cualquier cantidad de números del 1 al 7. Su versión negativa inicia con un menos y sigue la misma estructura. Números Hexadecimales: Inician con un "hx"seguido de un numero o una letra, el numero puede ir del 1 hasta el 9 y la letra siguiendo el abecedario puede ir desde la ."A"hasta la "F".Su versión negativa es lo mismo solo que inicia con un menos.

  Cabe aclarar que los binarios, octales y hexadecimales están representados internamente como un string, y los decimales y flotantes como números.

- Como trabajan las primitivas numéricas:

  Las primitivas numéricas son las operaciones que el programador puede realizar para no especificar cada una de ellas se mostrara en una imagen algunas de las operaciones que se pueden realizar. Para poder de que el programador realice dichas operaciones se espera recibir en ."ev"aluar-expresionüna expresión que contenga un numero seguido de una primitiva seguido de un numero, respectivamente se realizara una evaluación del tipo de numero a partir de una función llamada ."ev"aluar-numero-exp"que retornara el respectivo tipo de numero ingresado.Luego de tener todo ello se llama una a la función ."ev"aluar-primitiva"que aplicara la respectiva operación según la primitiva numérica que se este ingresando.

```
--> (1+2)
3
--> (4*5)
20
--> (3>6)
#f
--> (2<9)
#t
--> (2==2)
#t
--> (2!=3)
#t
--> (8 pow 2)
64
--> (17 mod 5)
2
--> (4>=4)
#t
```

2.2. Cadenas

Las cadenas requieren de que el programador abra comillas ingrese un identificador seguido de cualquier cantidad de identificadores y cierre las comillas.

Cuando se ingresa una expresión de este tipo entra a la parte de evaluar-expresion en donde en el case de çadena-exp"se va a recibir lo siguiente un identificador y una lista de identificadores. Por ejemplo el programador ingresa Üniversidad del valle", el identificador seria 'Universidad y la lista de identificadores (del,valle) de modo que lo que realizamos en cadena-exp es procesar la lista de manera recursiva sacando la cabeza transformando el identificador en string y concatenando con la cola de modo que asi se va construyendo el string y por ultimo se el identificador recibido se transforma a string y se concatena con la lista de identificadores que ya ha sido respectivamente procesada.

```
--> "Hola mundo"
"Hola mundo"
--> "Universidad del valle"
"Universidad del valle"
-->
```

Las primitivas que se realizan en las cadenas han de ser ingresadas primero la primitiva de la cadena seguido de abrir un paréntesis seguido de una expresión seguido de una coma seguido de una expresión y por ultimo cierra paréntesis. Evaluar-expresión recibirá esto y aplicara la respectiva primitiva usando una función llamada .ªplicar-primitiva-string".

Las funciones son las siguientes:

- concat: Esta es una función que concatena una cadena con otra

```
--> concat("hola","mundo")
"holamundo"
```

- string-length: Es una función que retornara el tamaño de la primera cadena que encuentre

```
--> string-length("Univalle")
8
```

- elementAt: Es una función que saca un identificador en una posición dada

```
--> elementAt("Hercules",4)
"u"
```

2.3. Estructura if

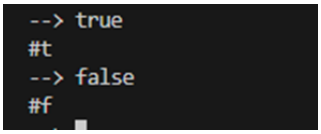
La estructura de if dentro del lenguaje de programacion tiene la siguiente forma: if expresión {expresión else expresión} cuando se recibe esta expresión por parte del programador, evaluar-expresión se ira al respectivo caso de la expresión if la cual va recibir lo siguiente: La condición, lo que hace si es verdad y lo que hace si es falso. Aprovechando la propia función if de racket podemos seguir fácilmente la misma estructura simplemente en la condición a evaluar del if de racket llamamos a evaluar-expresion enviando la condición recibida,en lo que hace si es verdad pues llamamos a evaluar-expresion enviando lo que hace si es verdad y por ultimo si no se cumple se llama a evaluar-expresion y se envía lo que hace si no es verdad.

```
--> if (5>4) {"hola" else "adios"}
"hola"
--> if (50<38) {"hola" else "adios"}
"adios"
```

2.4. Booleanos

- Implementación de booleanos

  Para la implementación de valores booleanos hacemos uso de la expresión bool-exp y de las expresiones true-exp y false-exp, las cuales son evaluadas por medio de evaluar-expresion, donde tenemos un caso para las dos anteriores, de ser true-exp se retorna el valor de verdadero de lo contrario se retorna el valor falso.



- Primitivas booleanas

  La función evaluar-booleano recibe un operador y una lista de argumentos, esta función permite evaluar por medio del operador la lista de elementos, comprobando si la lista está compuesta por un elemento, devolviendo el car, de lo contrario llama recursivamente a la función evaluar booleano con el resto de la lista.

  Para el manejo de las primitivas booleanas (and, or,, xor) se tienen las primitivas and-prim, or-prim, not-prim y xor-prim. Cada una de las expresiones evaluadas tiene la aplicación de las funciones and-logico, xor-logico y or-logico, donde se reciben dos valores y se devuelve el valor booleano dependiendo de el operador.

```
--> and(true, false) or(true, false) xor(true,true)
#f
--> #t
--> #f
```

```
--> and(true,false)
#f
-->
or(false,false)
#f
--> or(true,false)
#t
--> not(true)
#f
--> xor(true,true)
#f
--> xor(true,false)
#t
```

2.5. Listas

- Implementación listas A través de lista-exp, la cual recibe una lista de expresiones y por medio de la función eval-rands, podemos hacer un map de los elementos de una lista, como resultado obtenemos una lista de expresiones evaluadas.

  Prueba de listas:

```
Welcome to Racket v8.12 [cs].
--> list(1,2,3,4,5)
(1 2 3 4 5)
-->
```

- Primitivas listas

  Para el manejo de primitivas de listas contamos con la funcion aplicar-primitiva-lista en donde recibe la primitiva y la lista

  - rest: Si la primitiva es rest (rest-primlist) se retornará la cola de la lista.

```
--> rest(list (4,5,6,7))
(5 6 7)
```

  - firts: Si la primitiva es First (first-primList) retornará el primer elemento de la lista

```
--> first(list (1,2,3))
1
```

  - empty: Si la primitiva es empty? (empty-primlist) se retornará un valor booleano indicando si la lista está vacía.

```
--> empty?(list())
#t
-->
```

2.6. Arreglos

Dentro del manejo de los array manejamos los arreglos como vectores, para esto tenemos la definición de array-exp, la cual recibe una lista y convierte la lista a vector. Dentro de las primitivas de los arreglos tenemos prim-array-exp, que evalua la primitiva y la lista de argumentos.

- Primitivas de array

  Para el manejo de las primitivas tenemos la función aplicar-primitiva-array, la cual recibe la primitiva a utilizar (length, index, slice o setlist) acompañado del arreglo.

  - length: Para la primitiva length, por medio de vector-length y el vector, devolvemos el tamaño del mismo

    ```
    --> length(array(1,2,3,4))
    4
    ```

  - index: Para la primitiva index hacemos uso del elemento car que es el vector, y el elemento cadr que es el índice de ubicación del elemento que se busca.

    ```
    --> index(array("hola","mundo","el","yo"),3)
    "yo"
    ```

  - slice: Para la primitiva slice hacemos uso de una función auxiliar subvector, la función recibe como parámetros el vector, una posición inicial inicio y una posición final final, esta función realizará un llamado recursivo de subvector dentro del cual irá construyendo un vector desde la posición inicio hasta final por medio de cons, para finalmente retornar un subvector con las posiciones definidas del vector.

    ```
    --> slice(array("hola","mundo","el","yo",9,8,7,6,5,3),3,8)
    #("yo" 9 8 7 6 5)
    ```

  - setlist: Dentro de la primitiva setlist, tomamos el vector, la posición del valor que se requiere cambiar y el valor por el cual deseamos cambiarlo, para finalmente devolver el vector con el cambio realizado.

    ```
    --> setlist(array("hola","mundo","el","yo",9,8,7,6,5,3),3,"hasta la proxima")
    #("hola" "mundo" "el" "hasta la proxima" 9 8 7 6 5 3)
    ```