Lexer

Mario Alejandro Castro Lerma

Octubre 2023

1 Debug

Activa el "Debug" para las funciones del programa, lo cual hace que se imprima que los tokens que se leen del string.

2 read-char*

Recibe un input para después leer el input dado usando una función llamada read-char de racket y lo guarda en una variable llamada ch, la cual es impresa al final de la función. Si esta activo debug entonces imprime el carácter leído.

3 peek-char*

Sigue el mismo procedimiento de read-char*, pero en lugar de leer el carácter, solo lo observa y si esta activo el debug entonces imprime el valor observado.

4 char-digit?

Recibe ch y después revisa que ch se encuentre entre los valores 0 o 9 inclusivo, es decir que también puede ser 0 y 9, utiliza la función char¡=?.

5 char-varletter?

Recibe una variable llamada ch donde esperamos un carácter, después verifica que el carácter dado pertenezca a los caracteres validos de nuestro lexer (x,y,z), si es uno de estos entonces devuelve la lista a partir del carácter encontrado. Si no se encuentra el caráracter entonces solo devuelve falso.

6 char-delimiter?

Recibe la variable ch el cual esperamos que sea caracter, si ch es eof (usando eof-object?) o un espacio en blanco (usando char-whitespace?) entonces revisa si ch es "(" o ")", si es alguno de estos entonces regresa la lista apartir del caracter encontrado, si no entonces regresa falso.

7 lex-path

Determina el path del archivo que se leera, si lo que se lee es cadena entonces se mantiene con el default el cual es "¡unknown¿".

8 lex-line

Es la linea en la que se encuentra actualmente el lexer, default = 0.

9 lex-col

Es la columna en la que se encuentra el lexer actualmente, default = 0.

10 struct token

Aunque no es funcion, es una parte importante del lexer. Crea la estructura de los tokens los cuales contienen: type value line col

11 token-open-paren

Crea un token utilizando una estructura definida anteriormente, para definir un token de un parentesis abierto, se crea con los siguientes atributos: 'open-paren falso lex-line lex-col.

12 token-close-paren

Al igual que antes, se crea un token pero ahora para un parentesis cerrado, con la siguiente estructura: 'close-paren falso lex-line lex-col.

13 token-binop

Recibe una variable llamada type la cual determina el tipo de operador para el token, se crea una estructura tipo token: 'binop value lex-line lex-col.

14 token-number

Recibe una variable llamada value y col, los cuales determinan el valor y columna donde pertenece el token, se crea una estructura tipo token: 'number value lex-line col.

15 token-number/+

Error encontrado: Se intenta reutilizar el nombre de la estructura token, generando errores ya que la palabra token ya esta definida para la estructura de los tokens que creamos. Nuevo nombre para donde se encontraba token: token-original.

Esta función recibe un token con la variable token-original, donde se espera un token de un numero para después copiar este token igualmente en todos los aspectos excepto en la columna, donde restamos 1 al valor original para retroceder un espacio en las columnas para poder eliminar el símbolo + de un token numero en una función que se encuentra mas adelante ya que no es necesario (lex-sum-or-number).

16 token-number/-

Error encontrado: Al igual que la función anterior, se intenta utilizar el nombre de la estructura token. Nuevo nombre: token-original.

Al igual que la función anterior, esta recibe una variable llamada token-original, copiamos este token creando un token nuevo con los mismos valores excepto que restamos 1 en el valor original de la columna para retroceder un espacio y esta vez agregar el símbolo - ya que este afecta el valor numérico del token.

17 token-identifier

Recibe las variables symbol y col. Esta función crea un token de "identifier", symbol se utiliza en la estructura para definir el nombre del identificador y col se utiliza para poder especificar donde inicia el string de el nombre del identificador. Se crea una estructura token con la siguiente forma: 'identifier symbol lex-line col.

18 token-define

Recibe la variable col. Esta función crea un token para "define", se crea la estructura del token con la siguiente forma: 'define #f lex-line col

19 lex-open-paren

Esta función recibe la variable chars, Esta función lee el carácter dado, crea el token del paréntesis abierto y después aumenta el valor de la columna en 1, finalmente utiliza stream-cons para producir un stream donde el token del paréntesis abierto es agregado primero y después se llama lex para crear los tokens del resto de la expresión que estamos leyendo.

20 lex-close-paren

Recibe la variable chars, Esta función realiza algo parecido a la anterior, sin embargo ahora agrega un token de un paréntesis cerrado.

21 lex-whitespace

Recibe la variable chars, Esta función lo que hace es revisar si al haber un espacio en blanco, este es un salto de linea o solamente un espacio, Primero lee el carácter, despues revisa si es un salto de linea, si es salto de linea entonces modificamos los valores de token, a linea agregamos 1 y a col la volvemos 0. Si no es salgo de linea entonces solo aumentamos 1 a col y continuamos lex con el resto de caracteres.

22 lex-sum-or-number

Error encontrado: las opciones posibles de la función no contemplan el que se de "+" como token y que después no hayan números. Cuando suceda esto el lexer debe tomar "+" como operador cuando no hay mas números. El error esta en que cuando se revisa si el siguiente carácter es un numero y se da un eof, char-digit? da un error. Solución: agregué una nueva condición a tomar "+" como parte de un numero, además de tener que ser numero, ahora también tiene que ser diferente de eof, de esta forma en un salto de linea podremos estar seguros de que el "+" no es parte de un numero, también es necesario revisar el eof antes de que revisemos que sea numero ya que al revisar si es numero obtenemos un error.
Recibe la variable chars como argumento. Esta función tiene como objetivo decidir si el carácter "+" esta siendo utilizado como operador o para indicar si un numero es positivo. Primero lee el carácter actual, después revisa el siguiente valor (usando peek-char*). Una vez sabiendo el valor que sigue se decide que hacer, si es dígito entonces se llama token-number/+ para ignorar el "+" que es parte del numero y se continua leyendo el stream de tokens usando stream-cons. En caso de que el carácter siguiente no sea un dígito entonces se decide que "+" es un operador y por lo tanto se agrega al stream, después se continua leyendo el resto de caracteres. En los 2 casos se aumenta el numero de columna.

23 lex-negative-number

Recibe chars como argumento. El objetivo de esta función es saber si se quiere agregar un numero negativo y en caso contrario dar un error ya que no admitimos la operación de -. Primero leemos el carácter, luego obtenemos el siguiente carácter (usando peek-char*) y la columna actual. Con estos valores, revisamos si el carácter actual es un numero, si es numero entonces agregamos 1 a la columna y creamos un token con el numero completo con el símbolo "-" al inicio. Si no se cumple la condición inicial entonces regresamos un error donde indicamos que la operación menos no esta soportada en el lexer.

24 lex-mult

Recibe chars como argumento. El objetivo de esta función es determinar si hay un espacio después del carácter "*" para saber si es correcto agregarlo, primero lee el carácter, después revisa el siguiente carácter (con peek-char*) y finalmente revisa si este carácter es un delimitador para agregar el token de la operación "*" además de agregar 1 a la columna actual, si no es un delimitador el carácter siguiente entonces regresa un error donde indica que se espera un delimitador después del símbolo de multiplicación.

25 lex-identifier-or-keyword

Error encontrado: Al leer el strport para comparar con "define" había error, por lo tanto fue necesario cambiar la forma de escritura en strport, la solución encontrada fue display ya que está escribe los tipos que utilizan caracteres de forma directa o "cruda", es decir sin modificarlas de ninguna forma.

Recibe chars como argumento. El objetivo de esta función es discernir si los caracteres actuales forman parte de un identificador o de una palabra clave. Esta función primero define una nueva función la cual su trabajo es leer los caracteres que recibe y aumentar la columna en 1 hasta que se llegue a un delimitador para después verificar si se trata de un identificador o que se llegue a un valor que no pertenezca a los identificadores para regresar un #f.
Después de tener la función para leer los valores alfanuméricos, utilizamos un let* para definir el valor de la columna actual, strport con open-output-string para poder acumular lo que escribamos ahi en una cadena, y finalmente una funcion is-identifier? para poder revisar si algo es un identificador.
Con todo esto preparado, utilizamos un condicional primero para revisar si el caracter es parte de un identificador, si este es parte de un identificador entonces creamos un token correspondiente con el caracter y lo agregamos. En la siguiente condicion revisamos si la str que tenemos (la creamos con strport) es igual a "define", si se cumple entonces creamos un token y lo agregamos. En cualquier otro caso entonces regesamos un error donde indicamos que esperabamos un define o identifier.

26 zero-char-val

Recibe un carácter y lo convierte a integer.

27 lex-plain-number

Recibe chars como argumento. El objetivo de esta función es convertir caracteres que recibamos que sean digitos a su forma numerica o integer. Primero se define una función para construir integers, primero agrega 1 a la columna, despues vemos el siguiente valor en la cadena y a partir de esto hacemos un cond, si tenemos un delimitador entonces multiplicamos por 10 este valor y le sumamos el valor que teniamos anteriormente, si esta condicion no se cumple entonces revisamos si el caracter que estamos viendo es digito para entonces construir un integer multiplicando este valor por 10 y sumandole el valor que teniamos anteriormente, si no se cumple ninguno de estos entonces regresamos un error. Una vez construida la funcion que se usa para convertir los valores a integers, lo unico que se necesita es convertir a entero el caracter y a este agregarlo como token al stream con el valor obtenido y el numero de col correcto.

28 lex-end-of-file

Recibe chars como argumento, esta funcion cierra el "port" de caracteres y vacia el stream que podria haber quedado. Sirve para cerrar el archivo.

29 signal-lex-error

Recibe message, line y col como argumentos. Regresa un error el cual te indica de que es ('lex), en que parte (lex-path), en que linea (line), en que columna (col) y con un mensaje (message).

30 lex

Recibe chars como argumento. Esta funcion tiene el objetivo de elegir que hacer dependiendo del caracter que se va a leer, primero revisa que caracter sigue (con peek-char*) y después con un condicional revisa cada uno de los posibles caracteres que puede hacer nuestro lexer. Cuando coincide con uno entonces los caracteres son mandados a la función correspondiente.

31 lex-from-file

Recibe path como argumento. Esta funcion define los valores iniciales para leer un archivo de texto, primero definiendo el "path" del archivo, despues la linea

inicial y la columna. Finalmente llama la funcion lex con el archivo de texto correspondiente.

32 lex-from-string

Recibe str como argumento. Define los valores iniciales para leer a partir de una cadena, al ser una cadena el "path" se define como "¡unknown¿", la linea inicial es 1 y la columna 0. Finalmente se manda a llamar a lex con la cadena correspondiente y la funcion para abrir strings.