Compiladores – CCOMP8-1

Rodrigo Andre Cayro Cuadros

Analizador léxico propuesta “TRISTAGE”:

Tema:

Como parte del trabajo del curso de compiladores la propuesta es “tristage” un lenguaje orientado a la manipulación de imágenes, es decir que podamos modificar ciertas características de una o de varias imágenes seleccionadas previamente por usuario, facilitando las operaciones entre ellas, como por ejemplo unir varias imágenes en una o aplicarles ciertos efectos especiales, ejemplo: ESCALA de la imagen, ROTACIÓN de la imagen, RUIDO, entre otros. Para poder realizar estas funciones haremos uso de magick++.

Los tipos de datos permitidos para este lenguaje serán string que representan los nombres de los archivos, flotantes y enteros para hacer las operaciones de las imágenes. Dentro de sus operadores nos permitirán combinar imágenes y también aplicarles distintas funciones anidadas como el uso del operador “++”, también podremos comparar ambas imágenes usando el operador “==”, al igual que los objetos se asignan a las variables para así manipular las imágenes.

Funciones a implementar:

- Lectura de caracteres.
    - Getchar() devuelve el siguiente carácter de la entrada y mueve el puntero del carácter.
    - Peekchar() devuelve el siguiente carácter sin mover el puntero.
- Generación de tokens:
    - antokens(string, vector<string>&) => <TIPO_DE_TOKEN, LEXEMA>
- Funciones especiales:
    - addNoise - Añade una cantidad de ruido especificado por el usuario a la imagen:

- reduceNoise – Permite reducir el ruido de una imagen determinada, es el proceso inverso a lo que realiza la función anterior.

- colori – Coloriza una imagen usando una cantidad o valor de opacidad determinada dentro de los colores verdes, rojos y azules.

- compare – Compara dos imágenes proporcionadas por el usuario he indica si son iguales, la comparación se hace pixel por pixel.

- read - lee una imagen externa para utilizarla como objeto.

- save – Guardamos una imagen con el nombre especifico tras ya haber trabajado con ella.

- combine – Con esta función podremos combinar dos imágenes específicas, de esta manera se sobrepondrán entre ellas.

- solar – Aplica el efecto de solarize a la imagen seleccionado por el usuario.

- Tipos de tokens:
  - OP: ++, -, --, *, +, <-, ==, =
  - DELIM: ENDIF – ENDEL
  - RESERVED: IF, THEN, ELSE, PRINT
  - ID: [a-z, A-Z] [a-z, A-Z, 0-9]*
  - STR: “PALABRAS”, en el caso de tener que poner las comillas como parte de la palabra se hará uso de ¡” de esa manera: “Hola ¡’mundo¡’ “ == Hola ‘mundo’
  - NUM: (1-9)(0-9)* .(0-9)*

- Los espacios en blanco, tabulaciones, finales de línea y comentarios se consumen, pero no se devuelven nada.
  - Comentarios: % TU COMENTARIO %

Ejemplo 1 (FUNCIONAL):

```
main:
    imag = read "path.png" .
    imag2 = read "path2.png" .

    imag3 = read "path3.png" .
    imag4 = read "path4.png" .
    imag5 = read "path5.png" .

    reduceNoise imag3 11
    scale imag4 5
    imag = imag + imag2 + imag3 + imag4 + imag4 .

    save "newimg.jpg" imag
    save "imag3" imag3
    save "imag4" imag4
;
```

NO FUNCIONAL

```
main:
    imag = read "path.png" .
    imag2 = read "path2.png" .

    imag3 = read "path3.png" .
    imag4 = read "path4.png" .
    imag5 = read "path5.png" .

    reduceNoise imag3 11
    scale imag4 5
    imag = imag + imag2 + imag3 + imag4 + imag4 .

    save "newimg.jpg" imag
    save "imag3" imag3 .
    save "imag4" imag4 .
;
```

Ejemplo 2 (FUNCIONAL):

```
main:
    imag = read "path.png".
    imag2 = read "path2.png".

    imag3 = read "path3.png".

    imag = addNoise imag 30 .
    imag2 = imag + imag2 .

    imag3 = scale imag3 14 .
    imag4 = imag + imag2 + imag4 .
    save "newimg.jpg" imag4
    ;
```

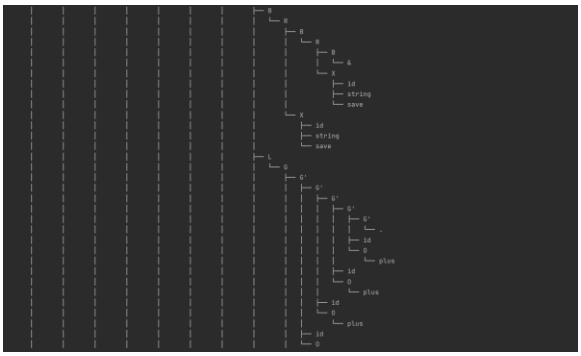
NO FUNCIONAL:

```
main:
    imag = read "path.png".
    imag2 = read "path2.png".

    imag3 = read "path3.png".

    imag = addNoise imag 30 .
    imag2 = imag + imag2 .

    imag3 = scale imag3 14 .
    imag4 = imag + imag2 + imag4 .
    save imag4 "newimg.jpg" .
    ;
```



Manejo de errores muestra el árbol del parseo.

Gramática:

```
M → main: B ;.
B → I
   | R
   | A
   | ε
A → id assign L B
L → R H
   | G
G → id G'
   | * id id
G' →O id G'
   | .
O → plus
   | minus
R → X B
X → addnoise id num
   | reducenoise id num
   | print string
   | colori id num num
```

```
    | save id
    | read string
    | rotate id num
    | scale id num.
H → plusplus R H
    | ;.
I → if Q then B Z.
Z → endif
    | else B endel.
Q → id P.
P → assignassign id J
    | less id J.
J → plusplus Q
    | minusminus Q
    | .
```

Especificación de los Operadores:

- “++” – Este operador sirve tanto para concatenar funciones como para concatenar operadores comparativos. Ej: IF some1 == some2 ++ some2 <- some3, en este caso hace referencia a un “AND”.
- “--” – Este operador sirve para concatenar operadores comparativos. Ej: IF some1 == some2 -- some2 <- some3, en este caso hace referencia a un “OR”.
- “-“ – Este operador recorta los bordes de una imagen según y cómo lo indique el usuario, está acompañado de un numero entero que indica el porcentaje a recortar.
- “*” – Este operador indica que ambas imágenes serán combinadas y se las aplicara una función determinada. Ej: some1 = * some2 some3 function1, en este caso estamos combinando en some1 las imágenes some2, some3 y con una función aplicada.
- “+” – Este operador permite combinar imágenes varias veces.
- “<-“ – Operador comparativo, este indica si ambas imágenes son diferentes, entonces se cumple con la condición.
- “==” – Operador comparativo, este indica si ambas imágenes comparadas con iguales, entonces se cumple con la condición.
- “=” – Este operador permite asignar un valor a una variable.

LINK DEL REPOSITORIO:

https://github.com/rodRigocaU/compiladores-