Taller 1 Analizador Léxico Lenguaje SR

Monitora: Laura V. Bohórquez Ramírez (lvbohorquezr@unal.edu.co)

SR es un lenguaje de programación diseñado a finales de la década de los 80's por la Universidad de Arizona en los Estados Unidos, desarrollado principalmente para programación distribuida, paralela y concurrente. Este lenguaje tiene como objetivo los sistemas distribuidos, extendiéndose desde los sistemas operativos hasta las aplicaciones.

Se recomienda revisar el siguiente material que sirve como manual de referencia:

- Página web de SR, Universidad de Arizona: https://www2.cs.arizona.edu/sr/index.html
- Papers sobre la implementación general de SR: https://www2.cs.arizona.edu/~gmt/pdf/sr-toplas.pdf
  - https://www2.cs.arizona.edu/sr/language.pdf

Grupos: 2 personas por grupo.

El objetivo para esta práctica es tomar un código fuente escrito en el lenguaje SR y realizar un análisis léxico sobre dicho código.

Entrada - Salida

La entrada para juzgar el analizador léxico está dada por un archivo que contiene código fuente escrito en lenguaje SR, que su programa debe analizar a nivel léxico para entregar la salida adecuada. Por ejemplo:

Entrada	Salida

`global sizes`
`  # matrix dimension, default 10`
`  var N := 10`
`  # number of processes, default 2`
`  var PR := 2`
`  # strip size`
`  var S : int`
`body sizes`
`  getarg(1, N); getarg(2, PR); S := N/PR`
`  if N mod PR != 0 ->`
`  write("N must be a multiple of PR"); stop (1)`
`  fi`
`end`
	`<global,1,1>`
	`<id,sizes,1,8>`
	`<var,3,4>`
	`<id,N,3,8>`
	`<tk_asig,3,10>`
	`<tk_num,10,3,13>`
	`<var,5,4>`
	`<id,PR,5,8>`
	`<tk_asig,5,11>`
	`<tk_num,2,5,14>`
	`<var,7,4>`
	`<id,S,7,8>`
	`<tk_dos_puntos,7,10>`
	`<int,7,12>`
	`<body,8,1>`
	`<id,sizes,8,6>`
	`<getarg,9,4>`
	`<tk_par_izq,9,10>`
	`<tk_num,1,9,11>`
	`<tk_coma,9,12>`
	`<id,N,9,14>`
	`<tk_par_der,9,15>`
	`<tk_punto_y_coma,9,16>`
	`<getarg,9,18>`
	`<tk_par_izq,9,24>`
	`<tk_num,2,9,25>`
	`<tk_coma,9,26>`
	`<id,PR,9,28>`
	`<tk_par_der,9,30>`
	`<tk_punto_y_coma,9,31>`
	`<id,S,9,33>`

	<tk_asig,9,35> <id,N,9,38> <tk_div,9,39> <id,PR,9,40> <if,10,4> <id,N,10,7> <mod,10,9> <id,PR,10,13> <tk_distinto,10,16> <tk_num,0,10,19> <tk_ejecuta,10,21> <write,11,6> <tk_par_izq,11,11> <tk_cadena,"N must be a multiple of PR",11,12> <tk_par_der,11,40> <tk_punto_y_coma,11,41> <stop,11,43> <tk_par_izq,11,48> <tk_num,1,11,49> <tk_par_der,11,50> <fi,12,4> <end,13,1>

Del ejemplo anterior, se pueden generalizar las reglas para realizar el análisis léxico:

- La salida consiste en una lista de tokens separados por saltos de línea, los cuales deben seguir el siguiente formato **<tipo_de_token,lexema,fila,columna>** . En el caso de las palabras reservadas, el token y el lexema son iguales. Entonces se debe imprimir el siguiente formato **<tipo_de_token,fila,columna>**. **Queda como parte del ejercicio encontrar el conjunto de palabras reservadas para SR, según la especificación dada en el manual de referencia.**

- Se deben especificar los lexemas de la siguiente manera, dependiendo de su token (recordar que los únicos valores que pueden ir fuera de comillas son los numéricos).

  <tk_cadena,lexema,fila,columna>

  <tk_num,lexema,fila,columna> **El lexema del número puede ser entero o real (con o sin signo)**

  <tk_div,lexema,fila,columna>

- Los identificadores encontrados seguirán el siguiente formato:

  <id,lexema,fila,columna>

- Los comentarios deberán ser ignorados.

- Tenga en cuenta que SR es case sensitive, por lo que una cadena como PROcess no será aceptada como palabra reservada. En cambio, la palabra process sí es considerada como palabra reservada.

- Los operadores y símbolos especiales deben imprimirse en el siguiente formato:

  <nombre_token,fila,columna>

  Donde nombre_token será determinado de la siguiente manera: tk_**nombre_token.** El nombre_token puede ser asignado libremente según su criterio. En el ejemplo anterior se utilizaron, por ejemplo: tk_coma, tk_dospuntos, tk_par_izq, tk_par_der, tk_llave_izq, tk_llave_der, tk_asig, tk_suma, ...

  Se aclaran algunos tokens que podrían generar inquietudes:

Token	Nombre
->	tk_ejecuta
[]	tk_separa

Errores

Si durante el proceso se detecta un error léxico se deberá abortar el análisis y se debe reportar el error de la siguiente manera:

>>> Error léxico(linea:X,posicion:Y)

Donde X y Y son números enteros que representan el número de línea y la posición donde fue detectado el inicio del error léxico.

Nota

- Si en la entrada de texto se detecta una cadena como esta 90.00.50, el analizador deberá detectar un tk_num 90.00, luego detectar el tk_punto, y por último, el tk_num 50. Esto ocurre porque el "." podría aparecer como un token único en este lenguaje; de lo contrario, sería un error léxico. En este caso, no es un error léxico.
- Si se detecta por ejemplo la cadena 1&23948998, el analizador deberá detectar el número 1 y luego marcar un error léxico en el &.
- Recuerde tener en cuenta casos extremos como por ejemplo: 2.5598055not3!=88&56.a , en este caso se detecta la siguiente secuencia:

  <tk_num,2.5598055,1,1>
  <id,not3,1,10>
  <tk_distinto,1,14>
  <tk_num,88,1,16>
  >>> Error lexico(linea:1,posicion:18)

- A la hora de imprimir una palabra reservada como "body", debe imprimir la misma palabra reservada en el espacio de token <body,fila,columna>.
- A la hora de realizar el análisis léxico se deberá seguir el principio de subcadena más larga, a continuación un ejemplo:

!=$	<tk_distinto,1,1> >>> Error lexico(linea:1,posicion:3)

Como se puede ver el análisis toma el token '!=' pues es el token más largo que puede tomar, y con el substring restante '$' informa un error léxico pues este carácter no está definido dentro de los tokens válidos para el lenguaje.

Ejemplos

Entrada	Salida

`resource factorial()`
`  # Esto es un comentario`
`  procedure fact(k: int) returns f: int`
`  if k < 0 -> f := -1`
`  [] k = 0 or k = 1 -> f := 1`
	`<resource,1,1>`
	`<id,factorial,1,10>`
	`<tk_par_izq,1,19>`
	`<tk_par_der,1,20>`
	`<procedure,3,4>`

```
      [] k > 1 -> f := k * fact(k-1)
      fi
   end fact

   var n: int
   writes("Cuántos factoriales hay? "); read(n)
   write()
   fa i := 1 to n ->
      write(i, "el factorial es ", fact(i))
   af
end factorial
```

```
<id,fact,3,14>
<tk_par_izq,3,18>
<id,k,3,19>
<tk_dos_puntos,3,20>
<int,3,22>
<tk_par_der,3,25>
<returns,3,27>
<id,f,3,35>
<tk_dos_puntos,3,36>
<int,3,38>
<if,4,7>
<id,k,4,10>
<tk_menorque,4,12>
<tk_num,0,4,14>
<tk_ejecuta,4,16>
<id,f,4,19>
<tk_asig,4,21>
<tk_num,-1,4,24>
<tk_separa,5,7>
<id,k,5,10>
<tk_igual,5,12>
<tk_num,0,5,14>
<or,5,16>
<id,k,5,19>
<tk_igual,5,21>
<tk_num,1,5,23>
<tk_ejecuta,5,25>
<id,f,5,28>
<tk_asig,5,30>
<tk_num,1,5,33>
<tk_separa,6,7>
<id,k,6,10>
<tk_mayorque,6,12>
<tk_num,1,6,14>
<tk_ejecuta,6,16>
<id,f,6,19>
<tk_asig,6,21>
<id,k,6,24>
<tk_multi,6,26>
<id,fact,6,28>
<tk_par_izq,6,32>
<id,k,6,33>
<tk_menos,6,34>
<tk_num,1,6,35>
<tk_par_der,6,36>
<fi,7,7>
<end,8,4>
<id,fact,8,8>
<var,10,4>
<id,n,10,8>
<tk_dos_puntos,10,9>
<int,10,11>
<writes,11,4>
<tk_par_izq,11,10>
<tk_cadena,"Cuántos factoriales hay?
",11,11>
<tk_par_der,11,38>
<tk_punto_y_coma,11,39>
<read,11,41>
<tk_par_izq,11,45>
```

	<id,n,11,46> <tk_par_der,11,47> <write,12,4> <tk_par_izq,12,9> <tk_par_der,12,10> <fa,13,4> <id,i,13,7> <tk_asig,13,9> <tk_num,1,13,12> <to,13,14> <id,n,13,17> <tk_ejecuta,13,19> <write,14,7> <tk_par_izq,14,12> <id,i,14,13> <tk_coma,14,14> <tk_cadena,"el factorial es ",14,16> <tk_coma,14,34> <id,fact,14,36> <tk_par_izq,14,40> <id,i,14,41> <tk_par_der,14,42> <tk_par_der,14,43> <af,15,4> <end,16,1> <id,factorial,16,5>

Nota:

- Revisar que el token <tk_num,-1,4,24> muestra que se debe tener en cuenta el signo de los números.

Entrada	Salida

resource main()
  import sizes, point1
  var pcap[N,N]: cap point1
  # create points
  fa i := 1 to N, j := 1 to N ->
  pcap[i,j] := create point1(i, j)
  af
  # give each point capabilities for its neighbors
  fa i := 1 to N, j := 1 to N ->
  send pcap[i,j].compute(pcap[i,1:N], pcap[1:N,j])
  af
end
	<resource,1,1>
	<id,main,1,10>
	<tk_par_izq,1,14>
	<tk_par_der,1,15>
	<import,2,4>
	<id,sizes,2,11>
	<tk_coma,2,16>
	<id,point1,2,18>
	<var,3,4>
	<id,pcap,3,8>
	<tk_cor_izq,3,12>
	<id,N,3,13>
	<tk_coma,3,14>
	<id,N,3,15>
	<tk_cor_der,3,16>
	<tk_dos_puntos,3,17>
	<cap,3,19>
	<id,point1,3,23>

	<fa,5,4> <id,i,5,7> <tk_asig,5,9> <tk_num,1,5,12> <to,5,14> <id,N,5,17> <tk_coma,5,18> <id,j,5,20> <tk_asig,5,22> <tk_num,1,5,25> <to,5,27> <id,N,5,30> <tk_ejecuta,5,32> <id,pcap,6,6> <tk_cor_izq,6,10> <id,i,6,11> <tk_coma,6,12> <id,j,6,13> <tk_cor_der,6,14> <tk_asig,6,16> <create,6,19> <id,point1,6,26> <tk_par_izq,6,32> <id,i,6,33> <tk_coma,6,34> <id,j,6,36> <tk_par_der,6,37> <af,7,4> <fa,9,4> <id,i,9,7> <tk_asig,9,9> tk_num,1,9,12> <to,9,14> <id,N,9,17> <tk_coma,9,18> <id,j,9,20> <tk_asig,9,22> <tk_num,1,9,25> <to,9,27> <id,N,9,30> <tk_ejecuta,9,32> <send,10,6> <id,pcap,10,11> <tk_cor_izq,10,15> <id,i,10,16> <tk_coma,10,17> <id,j,10,18> <tk_cor_der,10,19> <tk_punto,10,20> <id,compute,10,21> <tk_par_izq,10,28> <id,pcap,10,29> <tk_cor_izq,10,33> <id,i,10,34> <tk_coma,10,35> <tk_num,1,10,36> <tk_dos_puntos,10,37> <id,N,10,38> <tk_cor_der,10,39> <tk_coma,10,40>

	<id,pcap,10,42> <tk_cor_izq,10,46> <tk_num,1,10,47> <tk_dos_puntos,10,48> <id,N,10,49> <tk_coma,10,50> <id,j,10,51> <tk_cor_der,10,52> <tk_par_der,10,53> <af,11,4> <end,12,1>