UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL RESISTENCIA

Trabajo Práctico Integrador: Intérprete de Pseudocódigo en Español

Salomón, Hilel Mauricio
Segnana, Juan Franco
Zelinka, Gonzalo

Sintaxis y Semánticas de Lenguajes
Ingeniería en Sistemas de Información
Primer Cuatrimestre 2021
Universidad Tecnológica Nacional - Facultad Regional de Resistencia
Entrega 1: 25/04/21
Entrega 2: 30/05/21

Versión 2.0

Indice

Introducción

El presente trabajo tiene como objetivo realizar un analizador léxico y sintáctico del lenguaje "Pseudocódigo" usado en la Facultad Regional de Resistencia. En esta primera entrega se presenta la gramática a utilizar.

Lenguaje de pseudocódigo

Componentes léxicos

Palabras reservadas

_div , _mod
_o, _y, _no,
leer,
escribir
si, entonces, sino, fin_si
mientras, hacer, fin_mientras
repetir, hasta_que
para, hasta, fin_para
segun, fin_segun
accion, _es, fin_accion, proceso, ambiente

Observaciones

No se distinguirá entre mayúsculas ni minúsculas.
Las palabras reservadas no se podrán utilizar como identificadores.

Identificadores

Características

Estarán compuestos por una serie de letras, dígitos y el guión bajo.
Deben comenzar por una letra
No podrán terminar con el símbolo de guión bajo, ni tener dos guión bajo seguidos.
No se distinguirá entre mayúsculas ni minúsculas.

Identificadores válidos:

dato, dato_1, dato_1_a

Identificadores no válidos:

dato, dato, dato__1

Tipos de Datos

- **Número**
  # Se utilizarán números enteros, reales de punto fijo.
  # Todos ellos serán tratados conjuntamente como números.
- **Cadena**
  # Estará compuesta por una serie de caracteres delimitados por comillas dobles:
  "Ejemplo de cadena"
  "Ejemplo de cadena con salto de línea \n y tabulador \t"
  # Deberá permitir la inclusión de la comilla doble utilizando la barra (\):
  "Ejemplo de cadena con \" comillas\" simples".
- **Operadores**
  - Operador de asignación
    # asignación: :=
  - Operadores aritméticos
    #suma: +
    # Binario: 2 + 3
    #resta: -
    # Binario: 2 - 3
    #producto: *
    #división real: /
    #división entera: _div
    #módulo: _mod
    #potencia: **
  - Operadores relacionales de números y cadenas:
    #menor que: <
    #menor o igual que: <=
    #mayor que: >
    #mayor o igual: >=
    #igual que: =
    #distinto que: <>

    Por ejemplo:
    # si A es una variable numérica y control una variable alfanumérica, se pueden generar las siguientes expresiones relacionales:
    (A >= 0) _y (control <> "stop")
  - Operadores lógicos
    #disyunción lógica: _o
    #conjunción lógica: _y
    #negación lógica: _no
    Por ejemplo: *(A >= 0) _y _no (control <> "stop")*

- **Comentarios**

  # De encabezado o descripción de file: delimitado por símbolos **/**** y ***/**

  */** Fantástico comentario de descripción de contenido de archivos puede incluir varias lineas ***/*

  # De varias líneas: delimitados por los símbolos **/*** y ***/**

  / ejemplo maravilloso*
  de un comentario
  *de tres líneas ***/*

  # De una línea: Todo lo que siga al carácter **//** o **@** hasta el final de la línea.

// ejemplo espectacular de comentario de una línea
@ Otro ejemplo de linea

- Fin de sentencia (implementación opcional)

  Punto y coma ;

  # Se utilizará para indicar el fin de una sentencia**.**

Sentencias

- Inicio y fin de Algoritmo

  Accion identificador _es
  sentencias
  fin_accion

  # Declara nombre del algoritmo a utilizar, delimita las sentencias que forman el programa

- Inicio de declaración de variables

  Ambiente
  identificador : tipo de dato

  # Declara nombre de variable de un tipo determinado

- Inicio de declaración de sentencias

  Proceso
  sentencias

  # Declara inicio de proceso o conjunto de sentencias

- Asignación

  identificador := expresión numérica

  # Declara a identificador como una variable numérica y le asigna el valor de la expresión numérica.

  # Las expresiones numéricas se formarán con números, variables numéricas y operadores numéricos.

  identificador := “expresión alfanumérica”

  # Declara a identificador como una variable alfanumérica y le asigna el valor de la expresión
  alfanumérica.

- Lectura

  Leer (identificador)

Declara a identificador como variable numérica o alfanumérica y le asigna el caracter leído.

- Escritura

  Escribir (expresión alfanumérica y/o identificadores)

  # El valor de la expresión numérica es escrito en la pantalla.

  # Se debe permitir la interpretación de comandos de saltos de línea (\n) y tabuladores (\t) que // tabs

  puedan aparecer en la expresión alfanumérica.

  escribir ("\t Introduzca el dato \n");

  escribir("escribir texto mas identificador", identificador)

- Sentencias de control

  # Sentencia condicional simple

  si (condición) entonces

  sentencias

  fin_si

  # Sentencia condicional compuesta

  si (condición) entonces

  sentencias

  sino

  sentencias

  fin_si

  # Bucle "mientras"

  mientras (condición) hacer

  sentencias

  fin_mientras

  #Una condición será una expresión relacional o una expresión lógica compuesta.

  # Bucle "repetir"

  repetir

  sentencias

  hasta_que (condición)

  # Bucle "para"

  para (identificador:=valor_inicial) hasta valor_final, incremento hacer

  sentencias

  fin_para

Gramática

Descripción de la gramática

AMBIENTE→ *ambiente*
BLOQUE_AMBIENTE → *conjunto de sentencias del ambiente*
BLOQ_AMBIENTE→ *linea dentro del ambiente*
IDENTIFICADOR→ *identificadores o numeros*
VARIABLE→ *variable*
CONSTANTE→ *constante*
TIPO_DATO→ *tipo de dato*

PROCESO→ *proceso*
CONJ_SENTENCIA→ *varias lineas de sentencias de pseudo*
SENTENCIA → *sentencia*
S_ESCRIBIR → *sentencia escribir*
S_SI → *sentencia si*
S_CICLOS → *sentencia ciclos*
S_SEGUN → *sentencia según*

SALIDA_ESC → *salida de escribir*
ENTRADA_LEER → *entrada de leer*

COMENTARIO_ENCABEZADO → *comentario de tipo encabezado*
COMENTARIO_VARIASLINEAS → *comentario de varias lineas*
COMENTARIO_LINEA → *comentario de una linea*

C_PARA → *ciclo para*clea
C_MIENTRAS → *ciclo mientras*
C_REPETIR → *ciclo repetir*

CONJ_CONDICIONES → *conjunto de condiciones*
CONJ_S_SI→ *conjunto de sentencias del SI*
CONJ_COND_SEGUN → *conjunto de condiciones SEGÚN*
ID_TIPODATO → *identificador o tipo de dato*

CONJ_OPERACIONES → *conjunto de operaciones*
OP_ARITMETICA → *operacion aritmética*
T_OP_ARITMETICO → *tipo de operador aritmético*

RELACIONALES → *relacionales*
T_RELACIONAL → *tipo de relacional*
T_OP_LOGICO → *tipo de operadores logicos*

OP_CONDICION → *operación de condición*
EXPRESION → *expresión*
CONDICION → *condición*

Producciones

Aquellas derivaciones que se encuentran con color verde, es porque permiten la inclusión de comentarios.

Σ → COMENTARIO_ENCABEZADO accion IDENTIFICADOR _es AMBIENTE fin_accion | accion IDENTIFICADOR _es AMBIENTE fin_accion

COMENTARIO_VL_L → COMENTARIO_VARIASLINEAS | COMENTARIO_LINEA

AMBIENTE → ambiente BLOQUE_AMBIENTE PROCESO | ambiente COMENTARIO_VL_L BLOQUE_AMBIENTE PROCESO

BLOQUE_AMBIENTE → VARIABLE COMENTARIO_VL_L | CONSTANTE COMENTARIO_VL_L | VARIABLE COMENTARIO_VL_L BLOQUE_AMBIENTE | CONSTANTE COMENTARIO_VL_L BLOQUE_AMBIENTE | COMENTARIO_VL_L | COMENTARIO_VL_L BLOQUE_AMBIENTE | VARIABLE BLOQUE_AMBIENTE | CONSTANTE BLOQUE_AMBIENTE | VARIABLE | CONSTANTE

- ✓ Del bloque ambiente puede derivar en asignación de variables o constantes, hasta que no exista una nueva asignación.

VARIABLE → IDENTIFICADOR: TIPO_DATO
CONSTANTE → ID = TIPO_DATO
IDENTIFICADOR → cadena
TIPO_DATO → Cadena | Numérico

PROCESO → proceso COMENTARIO_VL_L CONJ_SENTENCIA | proceso CONJ_SENTENCIA

CONJ_SENTENCIA → S_ESCRIBIR CONJ_SENTENCIA | S_ESCRIBIR COMENTARIO_VL_L CONJ_SENTENCIA | S_LEER CONJ_SENTENCIA | S_LEER COMENTARIO_VL_L CONJ_SENTENCIA | S_SI CONJ_SENTENCIA | S_SI COMENTARIO_VL_L CONJ_SENTENCIA | S_CICLOS CONJ_SENTENCIA | S_CICLOS COMENTARIO_VL_L CONJ_SENTENCIA | SENTENCIA CONJ_SENTENCIA | SENTENCIA COMENTARIO_VL_L CONJ_SENTENCIA | COMENTARIO_VL_L CONJ_SENTENCIA | S_ESCRIBIR COMENTARIO_VL_L | S_LEER COMENTARIO_VL_L | S_SI COMENTARIO_VL_L | S_CICLOS COMENTARIO_VL_L | SENTENCIA COMENTARIO_VL_L | S_ESCRIBIR | S_LEER | S_SI | S_CICLOS | SENTENCIA | COMENTARIO_VL_L

- ✓ Dentro de CONJ_SECUENCIA se derivan todas las acciones que se pueden realizar dentro del proceso del algoritmo.

S_ESCRIBIR → escribir paréntesis_abierto SALIDA_ESC paréntesis_cerrado
SALIDA_ESC → IDENTIFICADOR | “cadena” | IDENTIFICADOR, SALIDA_ESC | “cadena”, SALIDA_ESC

- ✓ Desde S_ESCRIBIR podemos obtener el “escribir” en pseudocodigo, mientras que SALIDA_ESC contiene todo lo que se puede hacer en un escribir.

S_LEER → leer paréntesis_abierto ENTRADA_LEER paréntesis_cerrado
ENTRADA_LEER → IDENTIFICADOR

SENTENCIA → IDENTIFICADOR := TIPO_DATO | IDENTIFICADOR := OP_ARITMETICA
ID_TIPODATO → IDENTIFICADOR | TIPO_DATO

CONJ_OPERACIONES → OP_ARITMETICA | RELACIONALES
OP_ARITMETICA → ID_TIPODATO T_OP_ARITMETICO ID_TIPODATO | ID_TIPODATO T_OP_ARITMETICO
T_OP_ARITMETICO → + | - | / | * | _div | _mod | **
RELACIONALES → ID_TIPODATO T_RELACIONAL ID_TIPODATO | ID_TIPODATO T_RELACIONAL ID_TIPODATO RELACIONALES
T_RELACIONAL → < | > | <= | >= | = | <>

T_OP_LOGICO → _o | _y

- ✓ En estas producciones se definen todos los tipos de operadores para poder ser usados mas adelante.

S_SI → si paréntesis_abierto CONJ_CONDICIONES paréntesis_cerrado entonces CONJ_S_SI fin_si

- ✓ En un condicional simple, se evalúa una condición o un conjunto de condiciones concatenado por operadores lógicos.

CONJ_S_SI → CONJ_SENTENCIA | CONJ_ SENTENCIA sino CONJ_S_SI

- ✓ CONJ_S_SI esta producción tiene por finalidad dar la opción de anidar secuencias condicionales alternativas.

CONJ_CONDICIONES → CONDICION | CONDICION T_OP_LOGICO CONJ_CONDICIONES

- ✓ De esta manera, se permite la comparación de verdad de una o varias condiciones con los operadores de disyunción y conjunción

CONDICION → EXPRESION T_RELACIONAL EXPRESION | _no EXPRESION

- ✓ Cada condición final es una operación que devuelve un booleano de verdadero o falso, los cuales son los operadores relacionales y el operador lógico de la negación

EXPRESION → ID_TIPODATO | ID_TIPODATO T_OP_ARITMETICO ID_TIPODATO | ID_TIPODATO T_OP_ARITMETICO EXPRESION

S_SEGUN → según paréntesis_abierto IDENTIFICADOR paréntesis_cerrado hacer CONJ_COND_SEGUN fin_segun
CONJ_COND_SEGUN→ T_RELACIONAL ID_TIPODATO: CONJ_SENTENCIA | T_RELACIONAL ID_TIPODATO: CONJ_SENTENCIA CONJ_COND_SEGUN | _otro: CONJ_SENTENCIA

S_CICLO→ C_PARA | C_MIENTRAS | C_REPETIR | CONJ_SENTENCIA

- ✓ Desde S_CICLO se derivan todas las estructuras cíclicas que permite el pseudocodigo.

C_MIENTRAS → mientras paréntesis_abierto CONJ_CONDICIONES paréntesis_cerrado hacer CONJ_SENTENCIA fin_mientras

C_REPETIR → repetir CONJ_SENTENCIA hasta_que paréntesis_abierto CONJ_CONDICIONES paréntesis_cerrado

C_PARA → para paréntesis_abierto IDT_PARA paréntesis_cerrado hasta ID_TIPODATO, ID_TIPODATO hacer CONJ_SENTENCIA fin_para | para paréntesis_abierto IDT_PARA paréntesis_cerrado hasta ID_TIPODATO, hacer CONJ_SENTENCIA fin_para
IDT_PARA → IDENTIFICADOR := ID_TIPODATO | IDENTIFICADOR

- ✓ Permitimos un tercer parámetro al PARA, donde puede variar el incremento en cada iteración.

Terminales

- accion
- _es
- fin_accion
- ambiente
- cadena
- numérico
- proceso
- escribir
- leer
- paréntesis_abierto
- paréntesis_cerrado
- _o
- _y
- si
- entonces
- fin_si
- sino
- _no
- hacer
- según
- fin_segun
- mientras
- fin_mientras
- _otro
- repetir
- hasta_que
- para
- fin_para
- +
- -
- /
- *
- _div
- _mod
- **
- =
- <
- <=
- >=
- <>

Análisis léxico

Para la realización del LEXER acordamos el uso de la librería **PLY** para el lenguaje **Python**. Para entender esta librería usamos dos principales fuentes, la documentación oficial y un ejemplo práctico sencillo de cómo hacer una calculadora usando PLY, video que facilitó bastante el proceso, ya que nos ayudó a entender la estructura principal.

Lo primero fue transcribir todos los símbolos terminales o "**tokens**" dentro de un arreglo, y más tarde definir cada token con su correspondiente **expresión regular**. Algunas fueron definidas como **funciones** y otras como variables, debido a que las primeras tienen mayor "peso" u orden para el LEXER. Los nombres comienzan siempre con una "**t**_TERMINAL" ya que la librería PLY identifica a los **t**okens de esta menara.

Para la creación de expresiones regulares nos apoyamos de la página RegExr, la cual ayudó en comprobar que se cumplan los requesitos del lenguaje en cada token.

Realizamos **dos modos de ejecución**:

- uno línea por línea, el cual simplemente se inicia el programa y puede escribir en "modo libre", mientras el lexer analiza cuando se presiona ENTER;
- y por último un modo de lectura de archivo de texto, el cual se acciona pasando como parametro "-f ruta_archivo.e" por terminal. Esta opción genera un archivo txt llamado "tokens-analizados.txt", el cual indica cada token y su valor, además de contar la cantidad total analizada y los errores léxicos.

Instrucciones de uso

- Ir a la carpeta **bin**,
- abrir terminal y ejecutar programa con el comando "**lexer.exe**".
- Para analizar un archivo de texto, pasar parámetros con "**lexer.exe -f ../prueba/HolaMundo.e**".
- Para ejecutar el archivo lexer.py se debe instalar la librería "ply", con el comando: **pip install ply**. Se pasan parámetros de la misma manera.

```
C:\Users\juans\Documents\proyectos\python\ssl-pseudocodigo\bin>lexer.exe
Pasa salir pulse: [ctrl] + [C] | O escriba _salir
>> escribir("prueba"
LexToken(ESCRIBIR,'escribir',1,0)
LexToken(PARENTESIS_ABIERTO,'(',1,8)
LexToken(CADENA,'prueba',1,9)
>>
```

```
C:\Users\juans\Documents\proyectos\python\ssl-pseudocodigo\bin>lexer.exe -f ../prueba/HolaMundo.e
LexToken(COMENTARIO_ENCABEZADO,'/** Universidad Tecnologica Nacional  Facultad Regional Resistencia  Sintaxis y Semantic
a de los Lenguajes  Ciclo:2021  Autor:   Programa basico que imprime hola mundo */',1,0)
LexToken(ACCION,'ACCION',1,171)
LexToken(IDENTIFICADOR,'HolaMundo',1,178)
LexToken(ES,'_es',1,188)
LexToken(IDENTIFICADOR,'Ambiente',1,193)
LexToken(COMENTARIO_VARIASLINEAS,'/* Definimos el ambiente  variables que usamos  */',1,204)
```

```
C:\Users\juans\Documents\proyectos\python\ssl-pseudocodigo\bin>lexer.exe -f ../prueba/Errores.e
LexToken(ESCRIBIR,'escribir',1,0)
LexToken(PARENTESIS_ABIERTO,'(',1,8)
Caracter ilegal! : '"'.
En linea: 1. Posición: 9
LexToken(IDENTIFICADOR,'esta',1,10)
LexToken(IDENTIFICADOR,'cadena',1,15)
LexToken(IDENTIFICADOR,'no',1,22)
LexToken(IDENTIFICADOR,'estÃ',1,25)
Caracter ilegal! : '¡'.
En linea: 1. Posición: 29
LexToken(IDENTIFICADOR,'cerrada',1,31)
Caracter ilegal! : '\'.
En linea: 1. Posición: 38
Caracter ilegal! : '"'.
En linea: 1. Posición: 39
LexToken(PARENTESIS_CERRADO,')',1,40)
(!) Se exportó un .txt con los tokens analizados.

C:\Users\juans\Documents\proyectos\python\ssl-pseudocodigo\bin>
```

tokens-analizados.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
TOKEN | VALOR
-------------
1- ESCRIBIR: escribir
2- PARENTESIS_ABIERTO: (
3- IDENTIFICADOR: esta
4- IDENTIFICADOR: cadena
5- IDENTIFICADOR: no
6- IDENTIFICADOR: estÃ
7- IDENTIFICADOR: cerrada
8- PARENTESIS_CERRADO: )
-------------
Total de tokens válidos analizados: 8.
Total de tokens NO válidos: 4.
```