Instituto Tecnológico de Costa Rica
Ingeniería en Computación
Compiladores e Intérpretes
Verano, 2024
Profesor: Allan Rodríguez Dávila

Proyecto #1
Análisis Léxico

Introducción

Un grupo de desarrolladores desea crear un nuevo lenguaje imperativo, ligero, que le permita realizar operaciones básicas para la configuración de chips, ya que esta es una industria que sigue creciendo constantemente, y cada vez estos chips necesitan ser configurados por lenguajes más ligeros y potentes. Es por esto por lo que este grupo de desarrolladores requiere desarrollar su propio lenguaje para el desarrollo de sistemas empotrados.

Proyecto a desarrollar

Este proyecto comprende la fase de Análisis Léxico para la gramática descrita en el Anexo I. Se debe desarrollar un scanner utilizando la herramienta JFlex utilizando la directiva %cup, con el fin que los tokens retornados de tipo Symbol utilizando la clase sym.

Un programa escrito para este lenguaje está compuesto por una secuencia de declaraciones de procedimientos, que contienen diferentes expresiones y asignaciones de expresiones; todo programa debe contener un único método main.

Para comprobar el desarrollo de la fase el programa a presentar deberá tomar un archivo fuente y realizar lo siguiente:

a) El sistema debe leer un archivo fuente.
b) Se debe escribir en un archivo todos los tokens encontrados, identificador asociado con el lexema.
c) Reportar y manejar los errores léxicos encontrados. Debe utilizar la técnica de Recuperación en Modo Pánico (error en línea y continúa con la siguiente).

Scanner

El desarrollo del Analizador Léxico se debe realizar utilizando la herramienta JFLex. Incluye el reconocimiento de los tokens y recuperación de errores sobre los archivos fuentes analizados. Debe reportar los errores y seguir procesando el código fuente.

Deberán analizar la especificación del lenguaje e identificar los diferentes terminales y literales permitidos, además del identificador. Para estos deberá construir patrones o definiciones regulares que permita separar el código fuente es dichos lexemas y retornar los tokens correspondientes con la siguiente información: tipo de token, lexema, línea de aparición y columna.

Tiene un valor de 82 puntos.

Aspectos técnicos

El proyecto deberá correr en java y utilizar las herramientas JFlex y Cup (para generar clases Symbol y sym). En caso de requerir librerías adicionales para compilar y ejecutar el programa, deberán especificarlo en la documentación, ya que de lo contrario se descontarán puntos en la evaluación. Se debe incluir los archivos .flex y .cup, y el proyecto en Java que utilice los archivos generados por JFlex y Cup (base mínima).

Deberán utilizar el sistema de control de versiones GitHub, el repositorio deberá ser público o incluir al profesor en el control de acceso de este.

Se valorará el aporte generado por cada estudiante, considerando, entre otras cosas, los commit generados por cada uno. Por lo que el puntaje obtenido por cada uno de los estudiantes puede ser diferenciado.

Documentación

La documentación es un aspecto de gran importancia en el desarrollo de programas, especialmente en tareas relacionadas con el mantenimiento de estos.

Para la documentación interna, deberán incluir comentarios descriptivos para cada parte, con sus entradas, salidas, restricciones y **objetivo**.

La documentación externa deberá incluir:

1. Portada.
2. Manual de usuario: **instrucciones de compilación, ejecución y uso bien detalladas**.
3. Pruebas de funcionalidad: incluir *screenshots*.
4. Descripción del problema.
5. Diseño del programa: decisiones de diseño, algoritmos usados.
6. Librerías usadas: creación de archivos, etc.
7. Análisis de resultados: objetivos alcanzados, objetivos no alcanzados, y razones por las cuales no se alcanzaron los objetivos (en caso de haberlos).
8. Bitácora (autogenerada en git, commit por usuario incluyendo comentario).

Evaluación

La evaluación se va a centrar en dos elementos: programación y documentación.

El proyecto programado tiene un valor de **10%** de la nota final, en el rubro de Proyectos.

Desglose de la evaluación del proyecto programado:

1. Documentación interna 2 ptos.
2. Documentación externa 7 ptos.
3. Funcionalidad 82 ptos (ver detalle en Proyecto a Desarrollar)
4. Revisión del proyecto (estado del sistema y correcta gestión del tiempo) 5 ptos.
5. Hora de Entrega 4 ptos.

Forma de trabajo

El trabajo se debe realizar en parejas.

Aspectos administrativos

Debe crear un archivo **.zip** (“PP1_Integrante1_Integrante2.zip”) que contenga únicamente un archivo **info.txt** y 2 carpetas llamadas **documentacion** y **programa**, en la primera deberá incluir el documento de *word* o pdf solicitado y en la segunda los archivos y/o carpetas necesarias para la implementación de este proyecto programado. El archivo **info.txt** debe contener la siguiente información (cualidades):

a. Nombre del curso
b. Número de semestre y año lectivo
c. Nombre de los Estudiantes
d. Número de carnet de los estudiantes
e. Número de proyecto programado

f. Fecha de entrega
g. Estatus de la entrega (debe ser **CONGRUENTE** con la solución entregada): [Deplorable|Regular|Buena|MuyBuena|Excelente|Superior]

Entrega

Deberá subir el archivo antes mencionado al TEC Digital en el curso de COMPILADORES E INTERPRETES GR 60, en la asignación llamada "P1" debajo del rubro de "Proyectos". En la evaluación del Proyecto el rubro de "Hora de Entrega" valdrá por 4 puntos de la nota total del proyecto, según la siguiente escala:

a. Si se entrega antes de las 11:55:55 **PM** del miércoles 18 de diciembre de 2024, 4 puntos.
b. Si se entrega antes de las 11:55:55 **AM** del jueves 19 de diciembre de 2024, 2 puntos.
c. Si se entrega antes de las 11:55:55 **PM** del jueves 19 de diciembre de 2024, 0 puntos. Después de este punto, **NO SE ACEPTARÁN** más trabajos.

Los archivos .flex y .cup pueden ser revisados en el sistema de Control de Plagio del TEC Digital.

Todo el contenido de cada proyecto debe ser 100% original y en caso de conducta fraudulenta se procederá según lo indicado en el artículo 75 del RREA. Todos los miembros del grupo deberán participar en el desarrollo del proyecto y en la revisión, con el fin de demostrar la autoría del proyecto y la exposición de este.

ANEXO I

GRAMÁTICA

La Gramática BNF para el lenguaje imperativo debe contar con las siguientes características:

a. Permitir la creación de funciones, y dentro de ellas, estructuras de control, bloques de código y sentencias de código.
    - Apertura de bloque: abrecuento
    - Cierre de bloque: cierracuento
b. Manejar los tipos de variables enteras, flotantes, booleanas, caracteres, cadenas de caracteres (string) y arreglo estático.
    - integer: rodolfo
    - float: bromista
    - bool: trueno
    - char: cupido
    - string: cometa
c. Los identificadores deben iniciar y finalizar con guion bajo (_).
d. Se permite crear arreglos unidimensionales de tipo entero o char. Además, se permite obtener y modificar sus elementos, y ser utilizados en expresiones.
    - Corchete apertura: abreempaque
    - Corchete cierre: cierraempaque
e. Permitir sentencias para creación de variables, creación y asignación de expresiones y asignación de expresiones a variables, y algunos casos, sólo expresiones sin asignación.
    - Signo asignación: entrega
f. Las expresiones permiten combinar literales, variables y/o funciones, de los tipos reconocidos en la gramática.
g. Debe permitir operadores y operandos, respetando precedencia (usual matemática) y permitiendo el uso de paréntesis.
    - Paréntesis apertura: abreregalo
    - Paréntesis cierre: cierraregalo
h. Permitir expresiones aritméticas binarias de suma, resta, división –entera o decimal según el tipo–, multiplicación, módulo y potencia. Para enteros o flotantes.
    - Suma: navidad
    - Resta: intercambio
    - División: reyes
    - Multiplicación: nochebuena
    - Módulo: magos
    - Potencia: adviento
i. Permitir expresiones aritméticas unarias de negativo (-), incremento, decremento, después del operando (postorden); esto para enteros, el negativo adicionalmente se puede aplicar a flotantes. El negativo a literales y el incremento y decremento a variables.
    - Incremento: quien

- Decremento: grinch

j. Permitir expresiones relacionales (sobre enteros y flotantes) de menor, menor o igual, mayor, mayor o igual, igual y diferente. Los operadores igual y diferente permiten adicionalmente tipo booleano.
    - Menor: snowball
    - Menor o igual: evergreen
    - Mayor: minstix
    - Mayor o igual: upatree
    - Igualdad: mary
    - Diferente: openslae
k. Permitir expresiones lógicas de conjunción, disyunción y negación.
    - Conjunción: melchor
    - Disyunción: gaspar
    - Negación: baltazar
l. Debe permitir sentencias de código para las diferentes expresiones mencionadas anteriormente y su combinación. Además, dichas expresiones pueden usarse en las condicionales y bloques de las siguientes estructuras de control.
    - Delimitador de final de expresión: finregalo
m. Debe permitir el uso de tipos y la combinación de expresiones aritméticas (binarias y unarias), relacionales y lógicas, según las reglas gramaticales, aritméticas, relacionales y lógicas del Paradigma Imperativo, por ejemplo, tomando como referencia el lenguaje C.
n. Debe permitir las estructuras de control if-[else], while, switch y for, además, permitir return y break. Las expresiones de las condiciones deberán ser valores booleanos combinando expresiones aritméticas, lógicas y relacionales.
    - if: elfo
    - else: hada
    - while: envuelve
    - for: duende
    - switch: varios
    - case: historia
    - default: ultimo
    - break: corta
    - return: envia
    - dos puntos: sigue
o. Debe permitir las funciones de leer (enteros y flotantes) y escribir en la salida estándar (cadena carácter, enteros y flotantes), se pueden escribir literales o variables, se lee a identificadores.
    - print: narra
    - read: escucha
p. Debe permitir la creación y utilización de funciones, estos deben retornar valores (entero, flotantes, char o booleanos) y recibir parámetros (con tipo). Separador de parámetros sigue siendo la coma.

q. Debe existir un único procedimiento inicial main, por medio de la cual se inicia la ejecución de los programas.
   - main: _verano_

r. Además, debe permitir comentarios de una línea (#) o múltiples líneas (_ _/).

Ejemplo de código:

```
\_ esto es un comentario inicial _/

bromista _func1_ abreregalo cupido _x22_ , cupido _x22_ cierraregalo  abrecuento
     cupido _miChar_ entrega '!' finregalo
     cometa _str1_ entrega "Hola $%&/gaspar$& cierraregalo  mundo" finregalo
     bromista _fl1_ entrega 56.6 finregalo
     cupido _arr_ abreempaque _lit_ cierraempaque  entrega   abrecuento 'c','d'
cierracuento  finregalo
    _arr_ abreempaque _lit_ cierraempaque  entrega  'c' finregalo
     trueno _mibool_  entrega  true finregalo

          _mibool_     entrega         abreregalo     3.7    nochebuena    _fl1_
quien  cierraregalo  snowball 56 melchor true   finregalo

    duende  abreregalo  _i_  entrega  10 , _i_ snowball 30 nochebuena 2 , _i_
quien  cierraregalo
     abrecuento
        envuelve abreregalo _var2_ minstix 12.2 gaspar  abreregalo 34 navidad 33
cierraregalo minstix 12 cierraregalo
         abrecuento
            _var_  entrega  _var_ intercambio 1 finregalo @semantico
            envia finregalo @sintactico
         cierracuento

        elfo  abreregalo _var_ mary 0 cierraregalo
         abrecuento
            envia finregalo
         cierracuento
        hada
         abrecuento
            envia finregalo
         cierracuento

        varios abreregalo _ch_ cierraregalo   abrecuento
            historia 1 sigue _id_ quien  finregalo
            historia 2 sigue _id_ grinch finregalo
            envia finregalo
            ultimo sigue _id_ entrega _id_ finregalo
             cierracuento

     cierracuento
    envia 1 finregalo @hola
 cierracuento

rodolfo _verano_  abreregalo   cierraregalo   abrecuento
\_
Comentario 1
_/
```

```
@comentario 2
    escucha  abreregalo  _s1_  cierraregalo   finregalo
    narra abreregalo  _b1_  cierraregalo   finregalo
    miFunc  abreregalo  _miFunc_  abreregalo   cierraregalo  ,'a'  cierraregalo
  finregalo
    envia _bl1_ finregalo
 cierracuento
```

Guía:

```
\_ esto es un comentario inicial _/

float func1 ( char x22 , char x22 )  {
     char miChar <= '!' |
     string str1 <= "Hola $%&/#$& )  mundo" |
     float fl1 <= 56.6 |
     char arr [ lit ]  <=   { 'c','d' }  |
    arr [ lit ]  <=  'c' |
     bool mibool  <=  true |

    mibool  <=   (  3.7 * fl1 ++  )  < 56 ^ true   |

    for  (  i  <=  10 , i < 30 * 2 , i ++  )
     {
        while ( var2 > 12.2 #  ( 34 + 33 ) > 12 )
         {
            var  <=  var - 1 | @semantico
            return |@sintactico
         }

        if  ( var==0 )
         {
            break |
         }
        else
         {
            return ;
         }

        switch ( ch )   {
            case 1 : id ++  |
            case 2 : id -- |
            break |
            default : id <= id |
             }

     }
    return 1 | @hola
 }

rodolfo _verano_  (   )   {
/_
Comentario 1
_/
@comentario 2
    read  (  s1  )   |
    print (  b1  )   |
    miFunc  (  miFunc  (   )  ,'a'  )   |
    return bl1 |
 }
```