ST0244 – Grupo 004
Práctica

Lenguajes de programación

Reto a programar

#1: Entender la gramática y cómo se programó el compilador versión 01, para agregarle elementos al lenguaje.

Tenemos la gramática (ver siguiente diapositiva) que permite generar la visual de un programa definido en un lenguaje muy simple. Es explícito el uso del espacio entre los elementos de la gramática.

Gramática del lenguaje

$G = (N, T, P, program)$

N = {program, funDefinitionList, funDefinition, varDefList, varDef, statementList, statment, variableList, variable, numero}

<program> ::= program <funDefinitionList> endprogram

<funDefinitionList> ::= <funDefinition> <funDefinitionList> | ϵ

<funDefinition> ::= def <variable> (<varDefList>) <varDefList> <statementList> enddef

<varDefList> ::= <varDef> <varDefList> | ϵ

<varDef> ::= int <variable>

<statementList> ::= <statement> <statementList> | ϵ

<statment> ::= read <variable> | print <variable> | call <variable> (<variableList>)

<variableList> ::= <variable> <variableList> | ϵ

Ejemplo de programa válido en el lenguaje definido por la gramática anterior

```
program
def misterioso ( int x int y )
int temporal
read x
read y
print x
enddef

def main ( )
int a
int b
call misterioso ( a b )
enddef
endprogram
```

Reto # 1

Agregar lo que falta en la gramática para que el compilador en C++ funcione correctamente:

- Reconocimiento de una variable como un identificador (una variable empieza con una letra y luego van letras o números)
- Reconocimiento de números como constantes
- Expresiones aritméticas con variables o constantes numéricas y con signos ‘+’, ‘-’, ‘*’ y ‘/’
- Asignaciones (variable = expresión aritmética)
- Decisiones (if_then_else)
- Condiciones: igual, diferente, negación, y, o.
- Ciclos (ciclo while)
- Implementar el comando return para devolver una variable o una constante
- Se debe leer cualquier archivo que se mande como parámetro desde la línea de comandos
- El programa se debe modificar para que solo saque mensajes de error si la gramática encuentra un error
- Cada grupo define la sintaxis que quiera

Ayuda para la gramática

¿En cuál producción de la gramática agregarían lo siguiente (le faltan operaciones) y cómo?

<expr> ::= <term> <exprRest>

<exprRest> ::= + <term> <exprRest> | ϵ

<term> ::= <factor><termRest>

<termRest> ::= * <factor><termRest> | ϵ

<factor> ::= (<expr>) | variable | constant

Criterios de calificación

- Código documentado (con sus propios comentarios) 10%
- Gramática implementada en el lenguaje 30%
- El programa funciona (según la prueba) con los programas de prueba del profesor:
  - Que funcione un programa de prueba es el 2% de la nota de la práctica
  - El profesor tendrá 4 programas de prueba, que generan máximo el 8% de la nota de la práctica (2% por cada programa)
  - Si el programa lee diferentes nombres de archivo ***desde consola*** (2%)
- Total reto 1: 50%

Reto # 2

Una vez validado que el programa está escrito correctamente, se debe traducir a Jcoco.

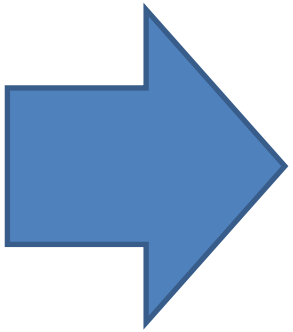
Nota: A diferencia de un compilador normal, no vamos a validar si las variables o las funciones están definidas.

El archivo generado debe tener el mismo nombre del archivo leído, pero se debe generar con extensión .casm.

Ejemplo

```
program
def divi ( int x int y )
if y != 0
then
  return x / y
endif
return 0
print x
enddef

def main ( )
def x
let x = divi ( 5 3 )
print x
enddef
endprogram
```



```
Function: divi/2
Constants: None, 0
Locals: x, y
BEGIN
          LOAD_FAST                1
          LOAD_CONST               1
          COMPARE_OP               3
          POP_JUMP_IF_FALSE   label00
          LOAD_FAST                0
          LOAD_FAST                1
          BINARY_TRUE_DIVIDE
          RETURN_VALUE
label00:  LOAD_CONST               0
          RETURN_VALUE
END
```

```
Function: main/0
Constants: None, 5, 3
Locals: x
Globals: divi, print
BEGIN
          LOAD_GLOBAL      0
          LOAD_CONST       1
          LOAD_CONST       2
          CALL_FUNCTION    2
          STORE_FAST       0
          LOAD_GLOBAL      1
          LOAD_FAST        0
          CALL_FUNCTION    1
          POP_TOP
          LOAD_CONST       0
          RETURN_VALUE
END
```

Criterios de calificación

- Código documentado (con sus propios comentarios) 10%
- El programa genera código de JCoco 15%
- El programa funciona con los programas de prueba del profesor:
  - Que funcione un programa de prueba es el 3% de la nota de la práctica
  - El profesor tendrá 5 programas de prueba, que generan máximo el 15% de la nota de la práctica (3% por cada programa)
- Sustentación 10%
- Total reto 2: 50%

Condiciones de la práctica

- Pueden ser equipos de máximo 2 personas. Informar al profesor en la clase del martes 3 de noviembre.
- Este trabajo se sustenta de la siguiente forma:
  - Se asigna un espacio de sustentación con el profesor y los estudiantes de un tiempo por definir
  - Se ejecuta el programa de cada estudiante con los archivos de prueba del profesor
  - Dependiendo de lo que observe el profesor, se pedirá explicación o cambios en el código durante la sustentación.
    - Si se llega a este punto, la nota de la sustentación será por criterio del profesor.
- Fecha de entrega: Noviembre 29
- Se puede hacer por repl.it, en su computador con compilador de g++, como deseen.

Condiciones generales

- El programa se va a probar ejecutando por línea de comandos, por ejemplo:

  ./compilador programa.244

  ./compilador es el nombre del ejecutable

  programa.244 es el nombre del archivo donde está el programa escrito en nuestro lenguaje para verificar si funciona adecuadamente.

- El nombre del archivo puede ser diferente en cada ejecución
- Si el compilador encuentra un error, debe aparecer un mensaje en pantalla de la siguiente manera:

  ERROR(número de línea): Mensaje del error encontrado