Compiladores
Tarea 1

Andrea Itzel González Vargas

Facultad de Ciencias UNAM

Para esta tarea se implementó un intérprete de expresiones aritméticas con analizador sintáctico de descenso recursivo utilizando C++ y flex.

Preguntas

Definir la gramática del lenguaje con la que piensas trabajar:

```
S     := InstProg
Prog  := InstProg | ϵ
Inst  := Asig; | Exp;
Asig  := var VarAsig'
Asig' := =Exp
Exp   := TermExp'
Exp'  := +TermExp' | -TermExp' | ϵ
Term  := FactTerm'
Term' := *FactTerm' | /FactTerm' | ϵ
Fact  := Num | Var | (Exp) | -Exp
Num   := IntNum'
Num'  := .Int | ϵ
Int   := Dig | DigInt
Dig   := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Var   := MinVar'
Var'  := MayVar' | MinVar' | DigVar' | ϵ
Min   := a | b | c | ... | z
May   := A | B | C | ... | Z
```

Ejemplo:

```
var y = 9;
var x = 2 + y;
4 + 7;
var x = y;
var h0lA = 3 * 4 - (4 + 2.32) * -2.0292 - x + y;
```

¿Dónde debo insertar el código necesario en el analizador para que se realicen las acciones según mis propósitos?

Para realizar el analizador sintáctico se utilizó una función por cada símbolo no terminal (sin contar a las variables y números, ya que de éstos se encargó flex para crear los tokens). Estas funciones se llaman entre sí de acuerdo a las producciones correspondientes para cada símbolo no terminal. Es posible ir interpetando la entrada al ir entrando en cada función, haciendo que cada una regrese su evaluación y recursivamente se irá obteniendo el resultado para cada instrucción, mientras se cuide la asociatividad de las operaciones.

Si se quiere crear un árbol de sintaxis abstracta, tambien es posible hacerlo dentro de estas funciones, de manera que cada una regrese un nodo del árbol y éstos se vayan uniendo al entrar en cada función recursivamente.

Información sobre el programa

Compilación

Para ejecutar el programa basta con ir a la carpeta src y ahí dentro correr el comando make. Esto creará un archivo binario llamado interprete.

Otros comandos:

make flex: Equivalente a correr flex++ scanner.lex.
make clean: Elimina los archivos que se crean al compilar.

Ejecución

Para ejecutar el programa se debe de correr el comando:

```
.\interprete <archivo.txt>
```

donde <archivo.txt> es un archivo de texto con las instrucciones que serán ejecutadas por el intérprete. Éstas deben de seguir la gramática descrita.

El resultado de correr el programa es que en la terminal se imprime el valor final de las variables que se haya declarado en la entrada, a demás de que se crean dos archivos, arbol.txt y log.txt.

En arbol.txt se muestra impreso el árbol de sintaxis abstracta de la entrada, y en log.txt se muestra el resultado de la evaluación de cada instruccion que está separada por punto y coma.

En el caso de que no se haya seguido las reglas de la gramática en la entrada, se muestra un mensaje de error.

Ejemplo:

Si se da como entrada la siguiente:

```
var x = 3 + 2;
var y = 4;
4.3 * 24 / y;
var x = y;
```

En la terminal se imprime

```
y: 4
x: 4
```

Se crea el archivo arbol.txt

```
(SEQ, ;)
├─>(ASIG, =)
│   ├─>(VAR, x)
│   └─»(MAS, +)
│       ├─>(NUM, 3)
│       └─»(NUM, 2)
└─»(SEQ, ;)
    ├─>(ASIG, =)
    │   ├─>(VAR, y)
    │   └─»(NUM, 4)
    └─»(SEQ, ;)
        ├─>(DIV, /)
        │   ├─>(MULT, *)
        │   │   ├─>(NUM, 4.3)
        │   │   └─»(NUM, 24)
        │   └─»(VAR, y)
        └─»(SEQ, ;)
            └─>(ASIG, =)
                ├─>(VAR, x)
                └─»(VAR, y)
```

Y el archivo log.txt

```
Instrucción 1: Asignación de variable 'x' con valor 5.000000
Instrucción 2: Asignación de variable 'y' con valor 4.000000
Instrucción 3: 25.800001
Instrucción 4: Asignación de variable 'x' con valor 4.000000
```