Universidad de Chile
Departamento de Ciencias de la Computación
CC3102-Teoría de la computación

Tarea 2:
Lenguajes Libres del contexto

Profesor:
Gonzalo Navarro
Auxiliares:
Manuel Cáceres
Ian Letter
Integrantes:
Alexis Espinoza
Florencia Miranda
Álvaro Yáñez

1-Introducción:

Una pluma P(posx; posy; col; mod; dir) es un objeto que puede moverse por un tablero y cambiar los colores de sus casillas. La pluma tiene un color propio y puede estar alzada o en contacto con el tablero. Si la pluma se mueve estando en contacto con el tablero, las casillas que visita cambian su color al color de la pluma.

El objetivo de esta tarea es implementar un intérprete para una gramática libre del contexto que leer los movimientos de una pluma y el coloramiento sobre un tablero. Para implementar la gramática libre del contexto tendrá que hacer uso de un lexical analyzer junto de un compiler generator.

En un principio tendremos un tablero en blanco y la pluma alzada en la posición (1,1) del tablero. El programa recibe un string que representará las operaciones a realizar por la pluma y entregará como resultado el tablero pintado.

2-Descripción del programa:

El programa se dividirá en 3 partes: el lexer que lee el input y lo separa en tokens que serán utilizado por un parser que será el encargado de hacer el árbol de derivación para el lenguaje. La otra parte es el código java que tendrá las clases para la pluma y todas sus acciones, asi como un objeto tablero que se pintará.

- **Lexer:** el archivo lexer.jflex es un programa que leerá el input y reconocerá expresiones regulares. Se han creado estados, los cuales tienen ciertas acciones definidas para espresión que se reconozca.
  En el estado inicial las variables como color y dirección crearán un token según la letra que estén reconociendo. Para el resto de los macros se crearán símbolos que luego se enviarán al parser. En la figura 1 vemos una muestra de la creación de los tokens.

```
{plumaAbajo}      {return symbolFactory.newSymbol("pluma_abajo",PABAJO); }
{plumaArriba}     {return symbolFactory.newSymbol("pluma_arriba",PARRIBA); }
{numero}          {return symbolFactory.newSymbol("number",NUMBER,Integer.parseInt(yytext())); }
{color}           {return symbolFactory.newSymbol("color",COLOR,yytext()); }
{direccion}       {return symbolFactory.newSymbol("direccion",DIRECCION,yytext()); }
"if"              {return symbolFactory.newSymbol("if",IF,yytext());}
"while"           {return symbolFactory.newSymbol("while",WHILE,yytext());}
"then"            {return symbolFactory.newSymbol("then",THEN,yytext());}
"do"              {return symbolFactory.newSymbol("do",DO,yytext());}
```

Figura 1: muestra de lexer.jflex

- **Parser:** el archivo parser.cup tiene el objetivo de recibir los tokens que producirá el lexer para luego trabajar con la gramática.
  El parser tiene definidos todos los elementos de la gramática que se pide implementar. Primero se crean los terminales y no terminales, luego se define el funcionamiento de la gramática como se muestra en la siguiente figura:

```
ROOT::= OP:o {: Accion accion= (Accion) o ; accion.execute(pluma); pluma.imprimir(); :};

OP ::=  BAJARP  {: RESULT = new BajarPluma()  ;:}
        |LEVANTARP {: RESULT = new LevantarPluma() ; :}
        | COLORP COLOR:col {: String color = (String) col ;RESULT= new SetColor(color); :}
        | DIRP DIRECCION:direc {: String direct = (String) direc ; RESULT= new SetDireccion(direct); :}
        | AVANZA NUMBER:n {: RESULT =new Avanzar(); :}
```

Figura 2 creación de las reglas de la gramática en archivo parser.cup.

En el lenguaje se determinará que métodos deben seguir los no terminales para hacer funcionar la pluma con el tablero.

- **Pluma:** Para implementar la pluma y el tablero se utilizará el command pattern debido a que la información entregada por las instrucciones deben guardarse para esperar si se ejecutan o no. Esto es porque hay que construir todo el árbol sintáctico primero para saber si se ejecuta una instrucción u otra

(en el caso que haya un if <expr > then { <op1 > } else { <op2 > } depende de la expresión si se ejecuta op1 u op2).
Existen dos interfaces llamadas Accion y otra Condiciones que contienen un método *execute* que será el encargado de hacer que funcione el tablero y la pluma. En la Figura 3 se ve la implementación de la clase para AND y su implementación:

```
public class ClaseAnd implements Condiciones{
  protected Condiciones cond1;
  protected Condiciones cond2;

  public ClaseAnd(Condiciones a1, Condiciones a2){
    cond1 = a1;
    cond2 = a2;
  }

  public boolean execute(Pluma p){
    return cond1.execute(p) && cond2.execute(p);
  }
}
```

Figura 3: Clase AND y su implementación.

3-Instrucciones de compilación:

La compilación en consola es igual que cualquier compilación, el programa pedirá en la entrada que se ingrese el nombre del archivo a probar y es ahí donde se ingresa algo des estilo “input.txt” para que se ejecute el programa.

4-Ejemplos de uso:

Este es el resultado del input que se mandó. Nuestro código crea el tablero blanco pero el tiempo no alcanzó para arreglar el tema de colorear.

La siguiente figura se creó ejecutando el programa de otro grupo.

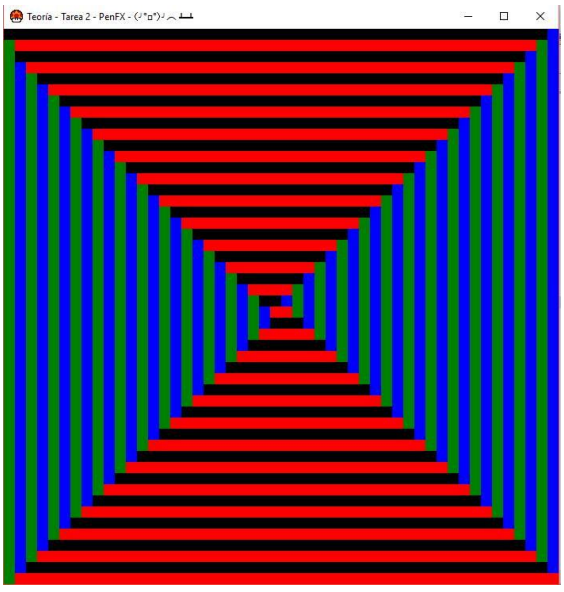

Figura4 : Resultado de la tarea.