Compiladores, Sem:2019-1, 3CV5, Proyecto, 17 de junio de 2019

Manual Técnico para Mini-Logo

Piste Gómez Cristian Jovani

Escuela Superior de Cómputo
Instituto Politécnico Nacional,México
piste.gomez@icloud.com

Índice

1. Introducción **3**

2. Funcionamiento y clases **3**

3. Descripción **5**
3.1. Symbol y SymbolData 5
3.2. Linea 7
3.3. CurrentState 8
3.4. Code 9
3.5. Parser 9
3.6. ParseVal 9
3.7. logoyacc 10

1. Introducción

Este manual es para facilitar la utilización y el entendimiento de los usuarios que hagan uso de este proyecto el cual es un interprete que realiza diferentes acciones (figuras, trazos, etc.) en un lienzo mediante instrucciones que el usuario deberá proporcionarle en un campo de texto.
Este interprete fue desarrollado principalmente en ***Java*** y en ***YACC/BYACC/J***

2. Funcionamiento y clases

Para el funcionamiento del interprete el cual como ya se dijo se codifico ***Java*** y en una version mejorada de ***YACC*** la cual soporta ***Java*** la cual se llama ***BYACC/J*** (Berkeley YACC/Java). Cabe mencionar que la estructura se dividió en varias clases para facilitar el funcionamiento y obtener la modularidad, las clases en las que se dividió el código son las siguientes.

1. **Main.java:** Función principal, solo instancia un objeto de la clase *GUI*.
2. **GUI.java:** Contiene el código de la interfaz gráfica para el usuario final: botones, campos de texto y pantalla de dibujado.
3. **Symbol.java:** Tabla donde se almacenara los símbolos. Es una parte fundamental dentro de un compilador.
4. **SymbolData.java**: Contiene algunos métodos que utilizara en la clase *Symbol*, principalmente son métodos para extraer información de un símbolo (getters y setters).
5. **Marco.java:** Es una especie de *"pila"* la cual comenzara a almacenar todas las instrucciones y sus diferentes valores.
6. **Linea.java:** En esta clase se dibujan las lineas que se necesiten crear con el comando *AVANZAR[n]*.
7. **JPanelDibujo.java:** Contiene el campo de dibujado que se manda a llamar en la clase *GUI*, como su nombre lo dice es donde la tortuga se moverá para trazar las lineas y mostrar las diferentes formas.
8. **Funcion.java:** Es una interfaz que se implementara en algunas clases/metodos en los que se vaya a necesitar.
9. **CurrentState.java:** Controlara el estado de la tortuga cuando tenga algún cambio efectuado con los comandos.

10. **Code.java:** Son métodos que se aplican a los atributos de la clase *Marco* para asignar parámetros o retornarlos.

11. **Parser.java:** En este código se tiene los autómatas con los cuales se apoyara el interprete. Se genera al compilar el archivo ***.y***

12. **ParserVal.java:** Al igual que la clase *Parser* esta también se genera al compilar el ***.y*** y contiene principalmente métodos que utilizara la clase *Parser* para extraer información de los valores que vaya manejando.

13. **logoyacc.y:** Este código contiene algunos métodos pero principalmente destaca por contener la gramática y parte de la semántica que tendrá el interprete.

Algunas clases como *Main, GUI* y *Funcion* se omitira su explicación ya que contienen prácticamente solo una linea de código (*Main y Funcion*) ya sea para instanciar o para delcarar una interfaz que utilizaran otras clases o porque solo muestra la interfaz gráfica (*GUI*) o el panel de dibujo (*JPanelDibujo*). Dicho esto comenzaremos con una breve explicación.

3. Descripción

3.1. Symbol y SymbolData

La clase *Symbol* contiene los métodos necesarios para crear una tabla de símbolos que muchas veces es necesaria por un compilador para almacenar alugnas variables. Su estrucutra mas basica es la que almacena el nombre de la variable y el valor, para ello se utilizo un *ArrayList* y con los metodos *Insert* y *lookup* con los cuales se agregara a la tabla un simbolo y se hará una consulta respectivamente como se ve en la figura 1.

```
11 public class Symbol {
12
13       ArrayList<SymbolData> simbolos;
14
15       public Symbol(){
16           simbolos = new ArrayList<SymbolData>();
17       }
18
19       public Object lookup(String nombre){
20           for(int i = 0; i < simbolos.size(); i++)
21               if(nombre.equals(simbolos.get(i).getNombre()))
22                   return simbolos.get(i).getObjeto();
23           return null;
24       }
25
26       public boolean insert(String nombre, Object objeto){
27           SymbolData par = new SymbolData(nombre, objeto);
28           for(int i = 0; i < simbolos.size(); i++)
29               if(nombre.equals(simbolos.get(i).getNombre())){
30                   simbolos.get(i).setObjeto(objeto);
31                   return true;
32               }
33           simbolos.add(par);
34           return false;
35       }
36
37       public void print(){
38           for(int i = 0; i < simbolos.size(); i++){
39               System.out.println(simbolos.get(i).getNombre() + simbolos.get(i).getObjeto().toString());
40           }
41       }
42
43   }
```

Figura 1: Symbol.java

Mientras que con *SymbolData* se tendrán algunos métodos que utilizara la clase *Symbol* para asignar o consultar valores en sus atributos como se puede ver en la siguiente figura la muestra los métodos para obtener el nombre de un atributo y el valor asi como para hacer las asignaciones de los mismos.

```
 9  public class SymbolData {
10
11      private String nombre;
12      private Object objeto;
13
14      public SymbolData(String nombre, Object objeto){
15          this.nombre = nombre;
16          this.objeto = objeto;
17      }
18
19      public String getNombre() {
20          return nombre;
21      }
22
23      public void setNombre(String nombre) {
24          this.nombre = nombre;
25      }
26
27      public Object getObjeto() {
28          return objeto;
29      }
30
31      public void setObjeto(Object objeto) {
32          this.objeto = objeto;
33      }
34
35  }
36
```

Figura 2: SymbolData.java

3.2. Linea

Con esta clase se trazaran las lineas que sean producto del comando *AVANZA[n];*, el cual le enviara los parámetros de las coordenadas y el objeto para el color, las coordenadas estarán separadas las horizontales (x_0, x_1) y las verticales (y_0, y_1) y estas a su vez se enviaran a la clase *GUI*

```
19      public Linea(int x0, int y0, int x1, int y1, Color color) {
20          this.x0 = x0;
21          this.y0 = y0;
22          this.x1 = x1;
23          this.y1 = y1;
24          this.color = color;
25
26      }
27
28      public int getX0() {
29          return x0;
30      }
31
32      public void setX0(int x0) {
33          this.x0 = x0;
34      }
35
36      public int getY0() {
37          return y0;
38      }
39
40      public void setY0(int y0) {
41          this.y0 = y0;
42      }
43
44      public int getX1() {
45          return x1;
46      }
47
48      public void setX1(int x1) {
49          this.x1 = x1;
50      }
51
52      public int getY1() {
53          return y1;
54      }
```

Figura 3: Linea.java

3.3. CurrentState

Con la clase *CurrentState* se piensa guardar los estados en los que se llegue a encontrar la tortuga cuando realice los movimientos para que no haya alguna interrupción o tenga algún reseteo a no ser que el usuario final presione el botón de *Limpiar Dibujo* el cual borrara todos los trazos que existan en el panel y mandara a la tortuga a su posición inicial.

```
20      public CurrentState(){
21          x = 0.0;
22          y = 0.0;
23          lineas = new ArrayList<Linea>();
24          color = Color.WHITE;
25      }
26
27      public void agregarLinea(Linea linea){
28          lineas.add(linea);
29      }
30
31      public void setPosicion(double x, double y){
32          this.x = x;
33          this.y = y;
34      }
35
36      public void limpiar(){
37          lineas.clear();
38      }
39
40      public ArrayList<Linea> getLineas() {
41          return lineas;
42      }
43
44      public double getX() {
45          return x;
46      }
47
48      public double getY() {
49          return y;
50      }
51
52      public int getAngulo() {
53          return angulo;
54      }
55
56      public void setAngulo(int angulo) {
57          this.angulo = angulo;
58      }
59
60      public Color getColor() {
61          return color;
62      }
```

Figura 4: CurrentState.java

3.4. Code

Code sera una especie de "expresiones regulares" las cuales regresaran algunas "etiquetas" que se reconocerán en el ***.y*** y entraran en la semántica con ayuda de %token. Tambien cuenta con algunos metodos los cuales contienen el funcionamiento de los comandos básicos (*GIRAR, AVANZAR, COLOR*) que serán reconocidos por los autómatas.

```
360 ▼   public static class Girar implements Funcion{
361         @Override
362 ▼       public void ejecutar(Object A, ArrayList parametros) {
363             CurrentState configuracion = (CurrentState)A;
364             int angulo = (configuracion.getAngulo() + (int)(double)parametros.get(0))%360;
365             configuracion.setAngulo(angulo);
366         }
367     }
368
369 ▼   public static class Avanzar implements Funcion{
370         @Override
371 ▼       public void ejecutar(Object A, ArrayList parametros) {
372             CurrentState configuracion = (CurrentState)A;
373             int angulo = configuracion.getAngulo();
374             double x0 = configuracion.getX();
375             double y0 = configuracion.getY();
376             double x1 = x0 + Math.cos(Math.toRadians(angulo))*(double)parametros.get(0);
377             double y1 = y0 + Math.sin(Math.toRadians(angulo))*(double)parametros.get(0);
378             configuracion.setPosicion(x1, y1);
379             configuracion.agregarLinea(new Linea((int)x0,(int)y0,(int)x1,(int)y1, configuracion.getColor()));
380         }
381     }
382
383 ▼   public static class CambiarColor implements Funcion{
384         @Override
385 ▼       public void ejecutar(Object A, ArrayList parametros) {
386             CurrentState configuracion = (CurrentState)A;
387             configuracion.setColor(new Color((int)(double)parametros.get(0)%256, (int)(double)parametros.get(1)%256, (int)(double)parametros.get(2)%256));
388         }
389     }
```

Figura 5: Code.java

3.5. Parser

Este código es generado al compilar el ***.y***. Esta clase contendrá los autómatas que definirán el correcto funcionamiento del programa, en el se hará uso de los métodos de la clase *Code* para reconocer cuando como y hacia donde avanzar y rotar o de que color debe pintarse

3.6. ParseVal

Con *ParseVal* se tendrán algunos métodos que utilizara la clase *Parser* esto con el fin de interactuar con las variables de *BYACC/J* tales como *dval, sval, ival*, etc que son propios del analizador sintáctico.

3.7. logoyacc

En *logoyacc* contaremos con toda la gramática y la semántica de nuestro programa.

```
32  %%
33      list:
34          | list'\n'
35          | list linea '\n'
36          ;
37
38      linea:   exp ';' {$$ = $1;}
39          |stmt {$$ = $1;}
40          |linea exp ';' {$$ = $1;}
41          |linea stmt {$$ = $1;}
42          ;
```

Figura 6: logoyacc.y

```
201     instrucciones: { $$ = new ParserVal(machine.agregarOperacion("nop"));}
202         |exp {$$ = $1;}
203         |instrucciones ',' exp {$$ = $1;}
204         ;
```

Figura 7: logoyacc.y

```
arglist:
    |exp {$$ = $1; machine.agregar("Limite");}
    |arglist ',' exp {$$ = $1; machine.agregar("Limite");}
    ;

nop:    {$$ = new ParserVal(machine.agregarOperacion("nop"));}
    ;

stmt:   if '(' exp stop ')' '{' linea stop '}' ELSE '{' linea stop'}' {
            $$ = $1;
            machine.agregar($7.ival, $1.ival + 1);
            machine.agregar($12.ival, $1.ival + 2);
            machine.agregar(machine.numeroDeElementos() - 1, $1.ival + 3);
    }
    | if '(' exp stop ')' '{' linea stop '}' nop stop{
            $$ = $1;
            machine.agregar($7.ival, $1.ival + 1);
            machine.agregar($10.ival, $1.ival + 2);
            machine.agregar(machine.numeroDeElementos() - 1, $1.ival + 3);
        }
    | while '(' exp stop ')' '{' linea stop '}' stop{
            $$ = $1;
            machine.agregar($7.ival, $1.ival + 1);
            machine.agregar($10.ival, $1.ival + 2);
        }
    | for '(' instrucciones stop ';' exp stop ';' instrucciones stop ')' '{' linea stop '}' stop{
            $$ = $1;
            machine.agregar($6.ival, $1.ival + 1);
            machine.agregar($9.ival, $1.ival + 2);
            machine.agregar($13.ival, $1.ival + 3);
            machine.agregar($16.ival, $1.ival + 4);
    }
    | funcion nombreProc '(' ')' '{' linea null '}'
    | procedimiento nombreProc '(' ')' '{' linea null '}'
    | instruccion '[' arglist ']' ';' {
            $$ = new ParserVal($1.ival);
            machine.agregar(null);
    }
    ;
```

Figura 8: logoyacc.y

```
158     »   »   ;
159 ▼   »   instruccion: FNCT {
160     »   »   »   $$ = new ParserVal(machine.agregar((Funcion)($1.obj)));//Llamada a funcion
161     »   »   }
162     »   »   ;
163
164 ▼   »   procedimiento: PROC { machine.agregarOperacion("declaracion"); }
165     »   »   ;
166 ▼   »   funcion: FUNC { machine.agregarOperacion("declaracion"); }
167     »   »   ;
168     »   »
169 ▼   »   nombreProc: VAR {$$ = new ParserVal(machine.agregar($1.sval));}
170     »   »   ;
171     »   »
172 ▼   »   null: {machine.agregar(null);}
173     »   »   ;
174     »   »
175 ▼   »   stop: {$$ = new ParserVal(machine.agregarOperacion("stop"));}
176     »   »   ;
177
178 ▼   »   if: IF {
179     »   »   $$ = new ParserVal(machine.agregarOperacion("IF_ELSE"));
180     »           machine.agregarOperacion("stop");//then
181     »           machine.agregarOperacion("stop");//else
182     »           machine.agregarOperacion("stop");//siguiente comando
183     »   »   }
184     »   »   ;
185
186 ▼   »   while: WHILE {
187     »   »   »   $$ = new ParserVal(machine.agregarOperacion("WHILE"));
188     »           machine.agregarOperacion("stop");//cuerpo
189     »           machine.agregarOperacion("stop");//final
190     »   »   }
191     »   »   ;
192
193 ▼   »   for : FOR {
194     »   »   »   $$ = new ParserVal(machine.agregarOperacion("FOR"));
195     »           machine.agregarOperacion("stop");//condicion
196     »           machine.agregarOperacion("stop");//instrucción final
197     »           machine.agregarOperacion("stop");//cuerpo
198     »           machine.agregarOperacion("stop");//final
199     »   »   }
```

Figura 9: logoyacc.y

```
exp:    VAR {      $$ = new ParserVal(machine.agregarOperacion("varPush_Eval"));
            machine.agregar($1.sval);
        }
        |'-' exp {
            $$ = new ParserVal(machine.agregarOperacion("negativo"));
        }
        |NUMBER {
            $$ = new ParserVal(machine.agregarOperacion("constPush"));
            machine.agregar($1.dval);
        }
        | VAR '=' exp {
            $$ = new ParserVal($3.ival);
            machine.agregarOperacion("varPush");
                                machine.agregar($1.sval);
                                machine.agregarOperacion("asignar");
                                machine.agregarOperacion("varPush_Eval");
            machine.agregar($1.sval);
        }
        | exp '*' exp {
            $$ = new ParserVal($1.ival);
            machine.agregarOperacion("MUL");
        }
        | exp '+' exp {
            $$ = new ParserVal($1.ival);
            machine.agregarOperacion("SUM");
        }
        | exp '-' exp {
            $$ = new ParserVal($1.ival);
            machine.agregarOperacion("RES");
        }
        |'(' exp ')' {
            $$ = new ParserVal($2.ival);
        }
        | exp COMP exp {
            machine.agregarOperacion("EQ");
            $$ = $1;
        }
        | exp DIFERENTES exp {
            machine.agregarOperacion("NE");
            $$ = $1;
        }
```

Figura 10: logoyacc.y

```
85      | exp MEN exp {
86          machine.agregarOperacion("LE");
87          $$ = $1;
88        }
89      | exp MENI exp {
90          machine.agregarOperacion("LQ");
91          $$ = $1;
92        }
93      | exp MAY exp {
94          machine.agregarOperacion("GR");
95          $$ = $1;
96        }
97      | exp MAYI exp {
98          machine.agregarOperacion("GE");
99          $$ = $1;
100       }
101     | exp AND exp {
102         machine.agregarOperacion("AND");
103         $$ = $1;
104       }
105     | exp OR exp {
106         machine.agregarOperacion("OR");
107         $$ = $1;
108       }
109     | '!' exp {
110         machine.agregarOperacion("NOT");
111         $$ = $2;
112       }
113     | RETURN exp { $$ = $2; machine.agregarOperacion("_return"); }
114     
115     | PARAMETRO { $$ = new ParserVal(machine.agregarOperacion("push_parametro")); machine.agregar((int)$1.ival); }
116     
117     |nombreProc '(' arglist ')' { $$ = new ParserVal(machine.agregarOperacionEn("invocar",($1.ival))); machine.agregar(null); } //instrucciones tiene la
        estructura necesaria para la lista de argumentos
118     ;//Lamada a Procedimiento
```

Figura 11: logoyacc.y