Instituto Politécnico Nacional

Escuela Superior de Cómputo

Departamento de Ciencias e Ingeniería

de la Computación

Compiladores

Proyecto Logo “Manual Técnico”

Tipo: Interprete

“Mini-logo (Tortuga)”

Profesor: Tecla Parra Roberto

Grupo: 5CM2

Integrantes del equipo:

Bernal Trani Marco Antonio

Patlani Mauricio Adriana

Ruvalcaba Flores Martha Catalina

Sandoval Hernández Eduardo

Fecha de entrega: 13/06/2022

Índice

Introducción

En el siguiente documento se presenta el proyecto llamado “Mini Logo (tortuga)”, si bien el Logo es un lenguaje de programación de alto nivel, en parte funcional, en parte estructurado; de muy fácil aprendizaje, razón por la cual suele ser el lenguaje de programación preferido para trabajar con niños y jóvenes. Fue diseñado con fines didácticos por Danny Bobrow, Wally Feurzeig, Seymour Papert y Cynthia Solomon, quienes se basaron en las características del lenguaje Lisp. Se creó con la finalidad de usarlo para enseñar programación y puede usarse para enseñar la mayoría de los principales conceptos de la programación, ya que proporciona soporte para manejo de listas, archivos y entrada/salida. Por lo tanto, el proyecto consiste en el desarrollo de un mini logo, puesto que será el diseño de una interfaz para que el usuario de instrucciones a un cursor gráfico, también conocido como tortuga virtual, con el fin de crear formas tales como círculos, cuadrados, arboles, estrellas, etc.

Requerimientos técnicos

Para la realización del proyecto Logo, se requirieron de los siguientes softwares.

Java: Java es un lenguaje de programación y una plataforma informática que fue comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarán, probablemente, a menos que tengan Java instalado, y cada día se crean más.

Java Develpment Kit (JDK): Java Development Kit es un software que provee herramientas de desarrollo para la creación de programas en Java. Puede instalarse en una computadora local o en una unidad de red. En la unidad de red se pueden tener las herramientas distribuidas en varias computadoras y trabajar como una sola aplicación.

Java Runtime Edition (JRE): Java Runtime Environment o JRE es un conjunto de utilidades que permite la ejecución de programas Java

IntelliJ IDEA: es un entorno de desarrollo integrado para el desarrollo de programas informáticos.

YACC: Es un programa para generar analizadores sintácticos. Las siglas del nombre significan Yet Another Compiler-Compiler, es decir, "Otro generador de compiladores más". Genera un analizador sintáctico basado en una gramática analítica escrita en una notación similar a la BNF.

BYACCJ: El cual es un compilador para el lenguaje YACC, pero enfocado al desarrollo en Java.

Objetivo

Implementar un intérprete para un lenguaje imperativo que implementa un subconjunto del lenguaje LOGO, el cual pueda ser usado como aprendizaje.

Expresiones regulares

Las siguientes son expresiones regulares:

- FORWARD[double];
- TURN[double];
- COLOR[double, double, double];
- PenUP[];
- PenDOWN[];
- for(expresión inicial; condición; expresión final){}
- while(condición){ declaracion(es)}
- if (condición){instrucciones}else{instrucciones}
- proc [nombre](){Bloque de instrucciones}
  Para usar parámetros se debe usar $x, donde x es el parámetro para usar, por otro lado, en las llamadas los parámetros se separan con una ‘,’.
- func [nombre](){ Bloque de instrucciones return valor; }
  Para usar parámetros se debe usar $x, donde x es el parámetro para usar, en las llamadas los parámetros se separan con una ‘,’.
- Condiciones disponibles:
- ==, comparación, entre enteros
- !=, diferente, entre enteros
- <=, operación menor o igual, entre enteros.
- >, operación mayor, entre enteros.
- >=, operación mayor o igual, entre enteros.
- &&, operación “and”, entre enteros.
- ||, operación “or”, entre enteros.

Gramática

La gramática del proyecto es la siguiente.

```
%token IF
%token ELSE
%token WHILE
%token FOR
%token COMP
%token DIFERENTES
%token MAY
%token MEN
%token MAYI
%token MENI
%token FNCT
%token NUMBER
%token VAR
%token AND
%token OR
%token FUNC
%token RETURN
%token PARAMETRO
%token PROC
%right '='
%left '+' '-'
%left '*' '/'
%left ';'
%left COMP
%left DIFERENTES
%left MAY
%left MAYI
%left MEN
```

```
%left MENI
%left '!'
%left AND
%left OR
%right RETURN
```

Figura 1. Símbolos terminales

```
%%

    list:
        | list'\n'
        | list linea '\n'
        ;

    linea: exp ';' {$$ = $1;}
        |stmt {$$ = $1;}
        |linea exp ';' {$$ = $1;}
        |linea stmt {$$ = $1;}
        ;

    exp:  VAR {
                $$ = new ParserVal(maquina.agregarOperacion("varPush_Eval"));
                maquina.agregar($1.sval);
            }
        |'-' exp {
                $$ = new ParserVal(maquina.agregarOperacion("negativo"));
            }
        |NUMBER {
                $$ = new ParserVal(maquina.agregarOperacion("constPush"));
                maquina.agregar($1.dval);
            }
```

Figura 2. Reglas gramaticales parte 1

```
| VAR '=' exp {
        $$ = new ParserVal($3.ival);
        maquina.agregarOperacion("varPush");
        maquina.agregar($1.sval);
        maquina.agregarOperacion("asignar");
        maquina.agregarOperacion("varPush_Eval");
        maquina.agregar($1.sval);
    }
| exp '*' exp {
        $$ = new ParserVal($1.ival);
        maquina.agregarOperacion("MUL");
    }
        | exp '/' exp {
        $$ = new ParserVal($1.ival);
        maquina.agregarOperacion("DIV");
    }
| exp '+' exp {
        $$ = new ParserVal($1.ival);
        maquina.agregarOperacion("SUM");
    }
| exp '-' exp {
        $$ = new ParserVal($1.ival);
        maquina.agregarOperacion("RES");
    }
|'(' exp ')' {
        $$ = new ParserVal($2.ival);
    }
```

```
  | exp COMP exp {
          maquina.agregarOperacion("EQ");
          $$ = $1;
      }
  | exp DIFERENTES exp {
          maquina.agregarOperacion("NE");
          $$ = $1;
      }
  | exp MEN exp {
          maquina.agregarOperacion("LE");
          $$ = $1;
      }
  | exp MENI exp {
          maquina.agregarOperacion("LQ");
          $$ = $1;
      }
  | exp MAY exp {
          maquina.agregarOperacion("GR");
          $$ = $1;
      }
  | exp MAYI exp {
          maquina.agregarOperacion("GE");
          $$ = $1;
      }
  | exp AND exp {
         maquina.agregarOperacion("AND");
          $$ = $1;
      }
```

Figura 3. Reglas gramaticales parte 2

```
    | exp OR exp {
            maquina.agregarOperacion("OR");
             $$ = $1;
        }
    | '!' exp {
            maquina.agregarOperacion("NOT");
            $$ = $2;
        }
    | RETURN exp { $$ = $2; maquina.agregarOperacion("_return"); }

    | PARAMETRO { $$ = new ParserVal(maquina.agregarOperacion("push_parametro")); maquina.agregar((int)$1.ival); }

    |nombreProc '(' arglist ')' { $$ = new ParserVal(maquina.agregarOperacionEn("invocar",($1.ival))); maquina.agregar(null); }
    //instrucciones tiene la estructura necesaria para la lista de argumentos
    ;

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

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

```
159     stmt:if '(' exp stop ')' '{' linea stop '}' ELSE '{' linea stop'}' {
160                             $$ = $1;
161                             maquina.agregar($7.ival, $1.ival + 1);
162                             maquina.agregar($12.ival, $1.ival + 2);
163                             maquina.agregar(maquina.numeroDeElementos() - 1, $1.ival + 3);
164                     }
165             | if '(' exp stop ')' '{' linea stop '}' nop stop{
166                             $$ = $1;
167                             maquina.agregar($7.ival, $1.ival + 1);
168                             maquina.agregar($10.ival, $1.ival + 2);
169                             maquina.agregar(maquina.numeroDeElementos() - 1, $1.ival + 3);
170                     }
171             | while '(' exp stop ')' '{' linea stop '}' stop{
172                             $$ = $1;
173                             maquina.agregar($7.ival, $1.ival + 1);
174                             maquina.agregar($10.ival, $1.ival + 2);
175                     }
176             | for '(' instrucciones stop ';' exp stop ';' instrucciones stop ')' '{' linea stop '}' stop{
177                             $$ = $1;
178                             maquina.agregar($6.ival, $1.ival + 1);
179                             maquina.agregar($9.ival, $1.ival + 2);
180                             maquina.agregar($13.ival, $1.ival + 3);
181                             maquina.agregar($16.ival, $1.ival + 4);
182                     }
183             | funcion nombreProc '(' ')' '{' linea null '}'
184             | procedimiento nombreProc '(' ')' '{' linea null '}'
185             | instruccion '[' arglist ']' ';' {
186                             $$ = new ParserVal($1.ival);
187                             maquina.agregar(null);
188                     }
189             ;
190     instruccion: FNCT {
191                     $$ = new ParserVal(maquina.agregar((Funcion)($1.obj)));
192                     }
193             ;
194
195     procedimiento: PROC { maquina.agregarOperacion("declaracion"); }
196             ;
197     funcion: FUNC { maquina.agregarOperacion("declaracion"); }
198             ;
199
200     nombreProc: VAR {$$ = new ParserVal(maquina.agregar($1.sval));}
201             ;
```

Figura 4. Reglas gramaticales parte 3

```
203     null: {maquina.agregar(null);}
204             ;
205
206     stop: {$$ = new ParserVal(maquina.agregarOperacion("stop"));}
207             ;
208
209     if: IF {
210                     $$ = new ParserVal(maquina.agregarOperacion("IF_ELSE"));
211             maquina.agregarOperacion("stop");//then
212             maquina.agregarOperacion("stop");//else
213             maquina.agregarOperacion("stop");//siguiente comando
214             }
215             ;
216
217     while: WHILE {
218                     $$ = new ParserVal(maquina.agregarOperacion("WHILE"));
219             maquina.agregarOperacion("stop");//cuerpo
220             maquina.agregarOperacion("stop");//final
221             }
222             ;
223
224     for : FOR {
225                     $$ = new ParserVal(maquina.agregarOperacion("FOR"));
226             maquina.agregarOperacion("stop");//condicion
227             maquina.agregarOperacion("stop");//instrucción final
228             maquina.agregarOperacion("stop");//cuerpo
229             maquina.agregarOperacion("stop");//final
230             }
231
232     instrucciones: { $$ = new ParserVal(maquina.agregarOperacion("nop"));}
233             |exp {$$ = $1;}
234             |instrucciones ',' exp {$$ = $1;}
235             ;
236
237
238
239 %%
```

Figura 5. Reglas gramaticales parte 4

Explicación de alto nivel

Para el desarrollo de la práctica, se requiere de conocimientos previos acerca de compiladores, puesto que el archivo funciona con una gramática y sus respectivas acciones gramaticales, así mismo, esta posee una maquina de pila la cual recibe la tabla de símbolos, además de poseer el funcionamiento de las operaciones, condiciones, decisiones, etc., todo lo que el usuario desee, siempre y cuando siga las indiaciones. Por otro lado, se tiene el archivo “Par”, el cual se usa para crear la tabla de símbolos a través de los atributos nombre (String) y un objeto (Object), además en el archivo “TablaDeSimbolos” se tiene el constructor y métodos para insertar, validar e imprimir símbolos de la tabla. Finalmente, para que el cursor trace sobre el panel se requiere que se haga uso de los comandos del lenguaje tales como FORWARD[x], TURN[x], COLOR [R, G, B], PenUP[] y PenDOWN[].

Diagrama de clases

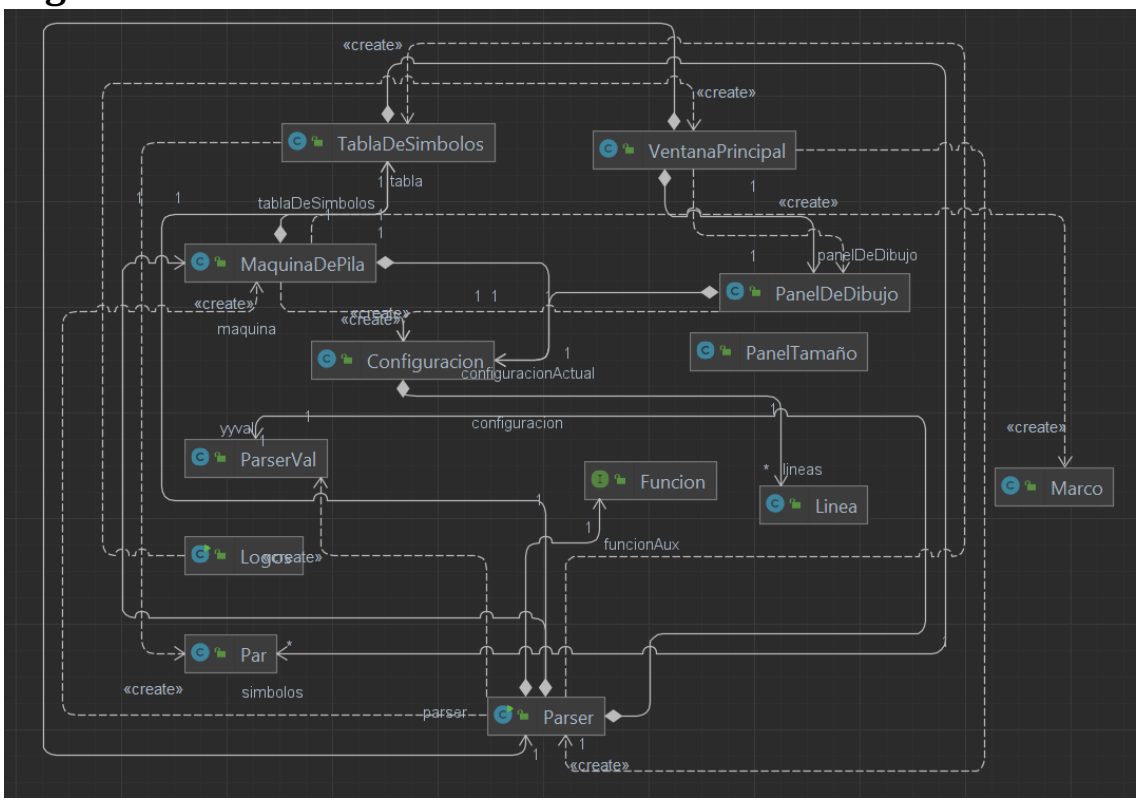


Figura 6. Diagrama de clases

Desarrollo del proyecto

El proyecto se desarrolló en base al código fuente de HOC (mini logo tortuga), tabla de símbolos de HOC3, Máquina Virtual de pila de HOC4, decisiones y ciclos de HOC5, ciclo for de HOC6, funciones, procedimientos y soporte de la recursividad del HOC7.

Código

A continuación, se mostrará los fragmentos de código más importantes del proyecto.

Máquina de Pila

```
public MaquinaDePila(TablaDeSimbolos tabla){
    configuracionActual = new Configuracion();
    contadorDePrograma = 0;
    memoria = new ArrayList<Method>();
    pila = new Stack();
    this.tabla = tabla;
    pilaDeMarcos = new Stack();
}
```

Figura 7. Máquina de pila

Tabla de símbolos

```
public class TablaDeSimbolos {
    11 usages
    ArrayList<Par> simbolos;

    //-------------------------------------------------------------//
    //          Constructor de la tabla de simbolos                //
    //-------------------------------------------------------------//
    14 usages
    public TablaDeSimbolos() { simbolos = new ArrayList<Par>(); }

    //-------------------------------------------------------------//
    //          Comprueba si existe un simbolo en la tabla         //
    //-------------------------------------------------------------//
    3 usages
    public Object encontrar(String nombre){
        for(int i = 0; i < simbolos.size(); i++)
            if(nombre.equals(simbolos.get(i).getNombre()))
                return simbolos.get(i).getObjeto();
        return null;
    }
```

Figura 8. Tabla de símbolos parte 1

```
    //-------------------------------------------------------------//
    //             Inserta un simbolo en la tabla                  //
    //-------------------------------------------------------------//
    7 usages
    public boolean insertar(String nombre, Object objeto){
        Par par = new Par(nombre, objeto);
        for(int i = 0; i < simbolos.size(); i++)
            if(nombre.equals(simbolos.get(i).getNombre())){
                simbolos.get(i).setObjeto(objeto);
                return true;
            }
        simbolos.add(par);
        return false;
    }

    //-------------------------------------------------------------//
    //         Imprime en consola los simbolos de la tabla         //
    //-------------------------------------------------------------//
    public void imprimir(){
        for(int i = 0; i < simbolos.size(); i++){
            System.out.println(simbolos.get(i).getNombre() + simbolos.get(i).getObjeto().toString());
        }
    }

}
```

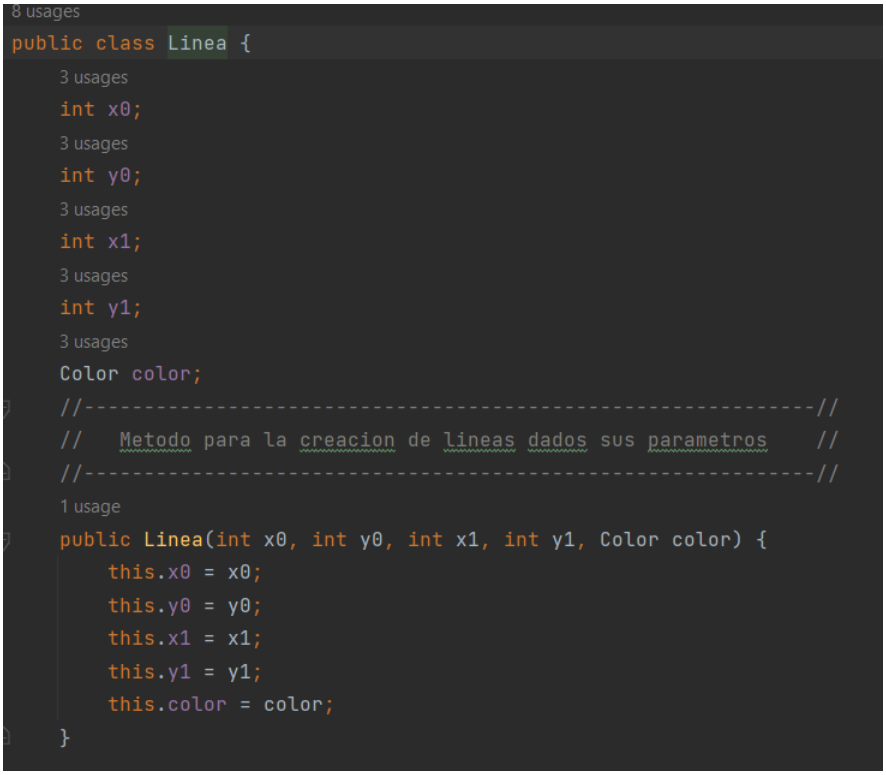
Figura 9. Tabla de símbolos parte 2

Para la realización del trazo se requiere tener la configuración de la línea, para ello se usa las coordenadas actuales.

```
2 usages
public class Configuracion {
    4 usages
    private final ArrayList<Linea> lineas;
    4 usages
    private double x;
    4 usages
    private double y;
    3 usages
    private int angulo;
    2 usages
    private Color color;

    //-------------------------------------------------------------------------------------//
    //  Constructor para la obtencion de las coordenadas actuales y las lineas a dibujar  //
    //-------------------------------------------------------------------------------------//
    19 usages
    public Configuracion(){
        x = 0.0;
        y = 0.0;
        lineas = new ArrayList<>();
    }
```

Figura 10. Configuración parte 1



```
8 usages
public class Linea {
    3 usages
    int x0;
    3 usages
    int y0;
    3 usages
    int x1;
    3 usages
    int y1;
    3 usages
    Color color;
    //-------------------------------------------------------------//
    //   Metodo para la creacion de lineas dados sus parametros    //
    //-------------------------------------------------------------//
    1 usage
    public Linea(int x0, int y0, int x1, int y1, Color color) {
        this.x0 = x0;
        this.y0 = y0;
        this.x1 = x1;
        this.y1 = y1;
        this.color = color;
    }
```

Figura 11. Configuración parte 2

```java
//-------------------------------------------------------------------------//
// Metodos para obtener los parametros necesarios para la creacion de lineas //
//-------------------------------------------------------------------------//
1 usage
public int getX0() { return x0; }
1 usage
public int getY0() { return y0; }
1 usage
public int getX1() { return x1; }
1 usage
public int getY1() { return y1; }

//-----------------------------------------------------------------------------//
//  Metodos para la obtencion y especificacion del color con el que se desea pintar  //
//-----------------------------------------------------------------------------//
public Color getColor() { return color; }
public void setColor(Color color) { this.color = color; }

//-------------------------------------------------------------//
//   Impresion en consola de los parametros de una linea       //
//-------------------------------------------------------------//
@Override
public String toString() { return "(" + x0 + "," + y0 + "," + x1 + "," + y1 + ")"; }
```

Figura 12. Configuración parte 3

Diseño de la interfaz

Se programaron los siguientes componentes de jav AWT.

- Un panel para que el cursor se desplace sobre de este y dibuje la forma.
- Un textarea para que el usuario escriba sobre de este la instrucción a dibujar.
- Un JComboBox es una lista desplegable que posee distintas opciones de formas para que sean dibujadas en el panel.
- Los botones, se tiene el primero para ejecutar la instrucción, el segundo es para limpiar el contenido del textarea, el tercer botón elimina el contenido del panel y el cuarto abre la ventana de Acerca del programa.

Figura 13. Interfaz Gráfica