Instituto Politécnico Nacional

Escuela Superior de Cómputo

Compiladores

Roberto Tecla Parra

Práctica 2

Dibujar Figuras

3CM16

Humberto Alejandro Ortega Alcocer (2016630495)

28 de Marzo del 2023

Índice

Índice **1**
Introducción **2**
Desarrollo **3**
Archivos Modificados 3
El archivo “forma.y” 3
El archivo “Maquina.java” 5
Prueba de Funcionamiento **9**
Compilación 9
Ejecución 9
Conclusión **11**
Referencias **12**

Introducción

En esta práctica realizamos la modificación al código proporcionado por el profesor *GrafiBasi* con el fin de que se puedan dibujar figuras geométricas básicas utilizando un compilador básico. Para lograr esto, lo primero será saber las figuras que queremos desarrollar, éstas son:

- Automóvil: un dibujo básico de un auto, dos ruedas y un rectángulo para la cabina.
- Casa: utilizar rectángulos (cuadrados) para dibujar una casa con un techo de forma de triángulo.
- Persona: utilizar líneas para dibujar el cuerpo y un círculo para la cabeza.

También deberá ser capaz de poder elegir el color con el cual dibujar cada figura a manera de que podamos “dibujar” libremente utilizando el proyecto.

El programa que se nos entrega inicialmente no compila, y deberemos modificar el código para que funcione. Para esto debemos modificar los contenidos de varios archivos, tanto de la gramática en YACC así como las implementaciones en Java que realizan los dibujos en sí mismo.

Desarrollo

Para esta práctica se tuvieron que modificar varios archivos para que funcionara, estos archivos fueron modificados lo menos posible a fin de que funcionara sin reescribir todo o modificar de base el programa dado.

Archivos Modificados

A continuación se mostrará las modificaciones realizadas en cada uno de los archivos.

El archivo "`forma.y`"

En este archivo se define la gramática con la que trabaja el programa, esta gramática que se presentaba inicialmente no contaba con la cantidad de tokens necesarios para leer correctamente los valores de entrada suministrados por el usuario.

Para corregirlo, se añadieron los tokens, así como se modificaron las acciones gramaticales para empujar en la pila de YACC los valores de cada token y así realizar las operaciones necesarias. El código final es:

```
%{
import java.lang.Math;
import java.io.*;
import java.util.StringTokenizer;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
%}
%token NUMBER LINE CIRCULO RECTANGULO COLOR PRINT
%start list
%%
list :
     | list ';'
     | list inst ';'   {
            maq.code("print"); maq.code("STOP"); return 1 ;
     }
     ;
inst:  NUMBER  { ((Algo)$$.obj).inst=maq.code("constpush");
                 maq.code(((Algo)$1.obj).simb);
                }
      | RECTANGULO NUMBER NUMBER NUMBER NUMBER {
                                                 maq.code("constpush");
                                                 maq.code(((Algo)$2.obj).simb);
                                                 maq.code("constpush");
                                                 maq.code(((Algo)$3.obj).simb);
                                                 maq.code("constpush");
                                                 maq.code(((Algo)$4.obj).simb);
                                                 maq.code("constpush");
                                                 maq.code(((Algo)$5.obj).simb);
                                                 maq.code("rectangulo");
                                               }
      | LINE NUMBER NUMBER NUMBER NUMBER         {
```

```
                                                maq.code("constpush");
                                                maq.code(((Algo)$2.obj).simb);
                                                maq.code("constpush");
                                                maq.code(((Algo)$3.obj).simb);
                                                maq.code("constpush");
                                                maq.code(((Algo)$4.obj).simb);
                                                maq.code("constpush");
                                                maq.code(((Algo)$5.obj).simb);
                                                maq.code("line");
                                              }
      | CIRCULO NUMBER NUMBER NUMBER             {
                                                maq.code("constpush");
                                                maq.code(((Algo)$2.obj).simb);
                                                maq.code("constpush");
                                                maq.code(((Algo)$3.obj).simb);
                                                maq.code("constpush");
                                                maq.code(((Algo)$4.obj).simb);
                                                maq.code("circulo");
                                              }
      | COLOR NUMBER { maq.code("constpush");
                maq.code(((Algo)$2.obj).simb);
                maq.code("color");
               }
      ;
%%
class Algo {
        Simbolo simb;
        int inst;
        public Algo(int i){ inst=i; }
        public Algo(Simbolo s, int i){
                simb=s; inst=i;
        }
}
public void setTokenizer(StringTokenizer st){
        this.st= st;
}
public void setNewline(boolean newline){
        this.newline= newline;
}
Tabla tabla;
Maquina maq;

StringTokenizer st;
boolean newline;
int yylex(){
String s;
int tok;
Double d;
Simbolo simbo;
   if (!st.hasMoreTokens())
      if (!newline) {
         newline=true;
         return ';';
      }
   else
      return 0;
   s = st.nextToken();
   try {
      d = Double.valueOf(s);
      yylval = new ParserVal(
             new Algo(tabla.install("", NUMBER, d.doubleValue()),0) );
      tok = NUMBER;
   } catch (Exception e){
   if(Character.isLetter(s.charAt(0))){
```

```
      if((simbo=tabla.lookup(s))==null)
         yylval = new ParserVal(new Algo(simbo, 0));
         tok= simbo.tipo;
      } else {
        tok = s.charAt(0);
      }
   }
   return tok;
}
void yyerror(String s){
   System.out.println("parser error: "+s);
}
static Parser par = new Parser(0);
static JFrame jf;
static JLabel lmuestra=new JLabel("                                    ");
static Canvas canv;
static Graphics g;
Parser(int foo){
   maq=new Maquina();
   tabla=new Tabla();
   tabla.install("line", LINE, 0.0);
   tabla.install("circulo", CIRCULO, 0.0);
   tabla.install("rectangulo", RECTANGULO, 0.0);
   tabla.install("color", COLOR, 0.0);
   tabla.install("print", PRINT, 0.0);
   maq.setTabla(tabla);
   jf=new JFrame("Calcula");
   canv=new Canvas();
   canv.setSize(600,600);
   jf.add("North", new PanelEjecuta(maq, this));
   jf.add("Center", canv);
   jf.setSize( 600, 700);
   jf.setVisible(true);
   g=canv.getGraphics();
   maq.setGraphics(g);
   jf.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
}
public static void main(String args[]){ new Parser(); }
```

El archivo “`Maquina.java`”

En este archivo se definen las funciones que realizan el trazado de los dibujos en los canvas proporcionados, las modificaciones requeridas para este archivo fueron:

1. Lo primero fue que cada función de cada figura extraiga de la pila los valores que requiere para trazar la figura. Esto se realiza con la operación pop().
2. Lo segundo fue modificar cada una de las funciones para que, utilizando estos valores, realicen el trazado de las figuras apropiadamente.

Finalmente, el archivo queda de la siguiente manera:

```
import java.awt.*;
import java.util.*;
import java.lang.reflect.*;

class Maquina {
  Tabla tabla;
  Stack pila;
```

```
Vector prog;

static int pc = 0;
int progbase = 0;
boolean returning = false;

Method metodo;
Method metodos[];
Class c;
Graphics g;
double angulo;
int x = 0, y = 0;
Class parames[];

Maquina() {
}

public void setTabla(Tabla t) {
  tabla = t;
}

public void setGraphics(Graphics g) {
  this.g = g;
}

Maquina(Graphics g) {
  this.g = g;
}

public Vector getProg() {
  return prog;
}

void initcode() {
  pila = new Stack();
  prog = new Vector();
}

Object pop() {
  return pila.pop();
}

int code(Object f) {
  System.out.println("Gen (" + f + ") size=" + prog.size());
  prog.addElement(f);
  return prog.size() - 1;
}

void execute(int p) {
  String inst;
  System.out.println("progsize=" + prog.size());
  for (pc = 0; pc < prog.size(); pc = pc + 1) {
    System.out.println("pc=" + pc + " inst " + prog.elementAt(pc));
  }
  for (pc = p; !(inst = (String) prog.elementAt(pc)).equals("STOP") && !returning;) {
    // for(pc=p;pc < prog.size();){
    try {
      // System.out.println("111 pc= "+pc);
      inst = (String) prog.elementAt(pc);
      pc = pc + 1;
      System.out.println("222 pc= " + pc + " instr " + inst);
      c = this.getClass();
      // System.out.println("clase "+c.getName());
      metodo = c.getDeclaredMethod(inst, null);
```

```
        metodo.invoke(this, null);
      } catch (NoSuchMethodException e) {
        System.out.println("No metodo " + e);
      }

      catch (InvocationTargetException e) {
        System.out.println(e);
      } catch (IllegalAccessException e) {
        System.out.println(e);
      }
    }
  }

  void constpush() {
    Simbolo s;
    Double d;
    s = (Simbolo) prog.elementAt(pc);
    pc = pc + 1;
    pila.push(new Double(s.val));
  }

  void color() {
    Color colors[] = { Color.red, Color.green, Color.blue };
    double d1;
    d1 = ((Double) pila.pop()).doubleValue();
    if (g != null) {
      g.setColor(colors[(int) d1]);
    }
  }

  void line() {
    double d4 = ((Double) pila.pop()).doubleValue();
    double d3 = ((Double) pila.pop()).doubleValue();
    double d2 = ((Double) pila.pop()).doubleValue();
    double d1 = ((Double) pila.pop()).doubleValue();

    if (g != null)
      (new Linea((int) d1, (int) d2, (int) (d3), (int) (d4))).dibuja(g);
  }

  void circulo() {
    double d3 = ((Double) pila.pop()).doubleValue();
    double d2 = ((Double) pila.pop()).doubleValue();
    double d1 = ((Double) pila.pop()).doubleValue();

    if (g != null)
      (new Circulo((int) d2, (int) d3, (int) d1)).dibuja(g);
  }

  void rectangulo() {
    double d4 = ((Double) pila.pop()).doubleValue();
    double d3 = ((Double) pila.pop()).doubleValue();
    double d2 = ((Double) pila.pop()).doubleValue();
    double d1 = ((Double) pila.pop()).doubleValue();

    if (g != null)
      (new Rectangulo((int) d1, ((int) d2), (int) d3, ((int) d4))).dibuja(g);
  }

  void print() {
    Double d;
    d = (Double) pila.pop();
    System.out.println("" + d.doubleValue());
  }
```

```
  void prexpr() {
    Double d;
    d = (Double) pila.pop();
    System.out.print("[" + d.doubleValue() + "]");
  }
}
```

Prueba de Funcionamiento

Compilación

Para compilar el programa lo primero será utilizar **byacc** que es un compilador de YACC pero que funciona con Java en lugar de C con el cual obtendremos nuestro *Parser* con el cual podremos comenzar la compilación final. El comando de compilación es:

```
$ byacc -J forma.y
```

Una vez que tengamos los archivos, entonces compilaremos el programa general:

```
$ javac *.java
```

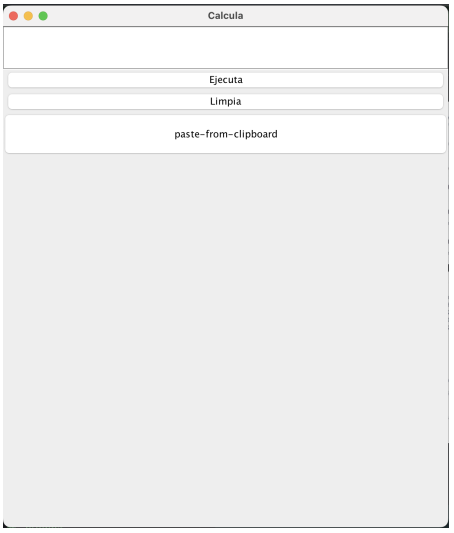
Con esto contaremos con el programa listo para ejecutarse.

Ejecución

Para ejecutar, usaremos el comando:

```
$ java Parser
```

Y se nos mostrará la siguiente ventana:

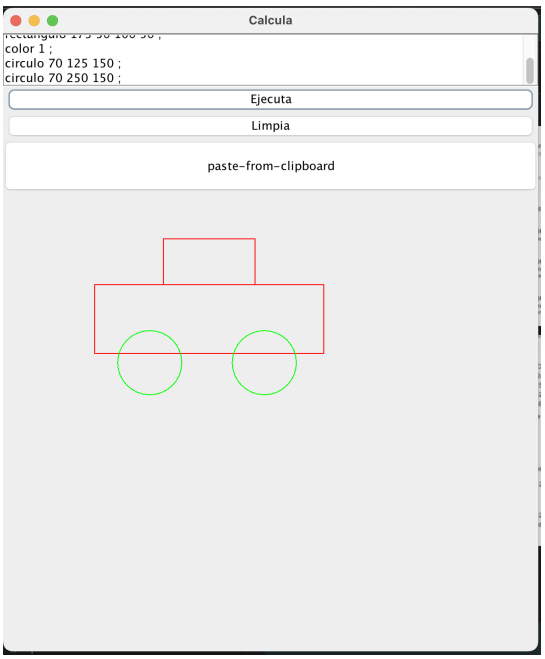


Para añadir el código de entrada usaremos el campo en la parte superior de la ventana y haremos click en “Ejecuta” con el cual se dibujará en la parte de abajo la figura que hayamos especificado. Por ejemplo, para dibujar el automóvil, el código de dibujo es este:

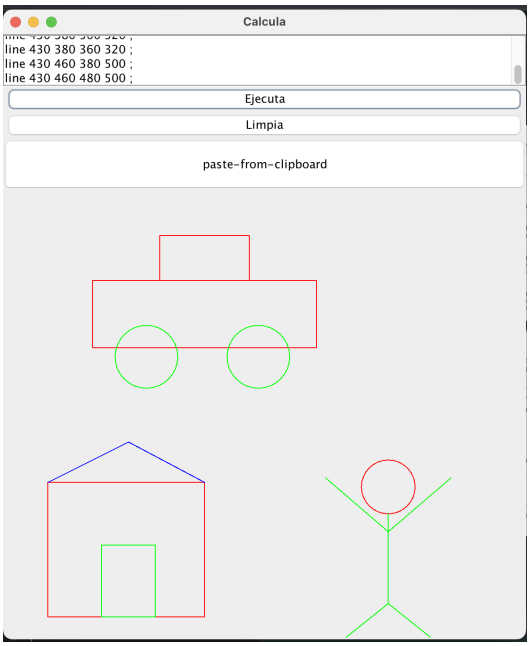
```
color 0 ;
rectangulo 100 100 250 75 ;
```

```
rectangulo 175 50 100 50 ;
color 1 ;
circulo 70 125 150 ;
circulo 70 250 150 ;
```

Y la imagen que se muestra es la siguiente:



Finalmente, dibujando los tres elementos al mismo tiempo:



Conclusión

Al realizar esta práctica aprendí que se puede utilizar YACC para otros lenguajes que no sean C, y con esto se abre la posibilidad a usar como lenguaje de bajo nivel alguno que ofrezca las funcionalidades que necesitamos, en este caso para dibujar fue Java.

El modificar la gramática y la implementación me ayudó a entender mejor cada etapa dentro del proceso de ejecución a manera de que se pudiera dibujar. Con la gramática se tuvo que aprender a modificar los tokens que definen las producciones para que pudiéramos aceptar la información necesaria para las funciones.

Al modificar la implementación final fue interesante entender cómo se pueden extraer valores de la pila para poder utilizarlos en la función y así lograr que el programa nos permita dibujar las figuras en cuestión.

Finalmente fue divertido usar el programa final para dibujar y ver cómo podríamos crear un lenguaje gráfico.

– *Humberto Alejandro Ortega Alcocer.*

Referencias

1. J.L. Scott, "A Better Lemon Squeezer: Incremental Generation of Lexers and Parsers," ACM SIGPLAN Notices, vol. 42, no. 3, pp. 95-108, 2007. doi: 10.1145/1238844.1238858

2. M. Osborn, "Parsing Expressions by Recursive Descent," Proceedings of the ACM SIGPLAN Symposium on Principles of Programming Languages, San Francisco, CA, USA, 1975, pp. 23-31. doi: 10.1145/512927.512931

3. M. J. C. Gordon, "Programming Language Theory," in Handbook of Theoretical Computer Science, J. van Leeuwen, Ed. Elsevier Science Publishers, 1990, pp. 477-567. doi: 10.1016/B978-0-444-88074-1.50014-5