**INSTITUTO POLITÉCNICO NACIONAL**

**ESCUELA SUPERIOR DE CÓMPUTO**

**COMPILADORES**

**PROYECTO FINAL**

*“LOGO”*

MANUAL TECNICO

**PROF:**

Tecla Parra Roberto

**ALUMNO:**

Gutiérrez Huerta Erick Alejandro

**GRUPO:**

3CM7

**OBJETIVO**

El objetivo de este trabajo es dar una referencia técnica acerca del proyecto para que pueda ser entendido por alguna persona con los conocimientos técnicos aptos y así poder ser modificado mejorado o alguna otra acción en el software.

**INTRODUCCIÓN**

Este proyecto se realizó con la utilización de BYACC, el cual es un compilador para el lenguaje YACC que nos permite realizar este mini lenguaje de programación “Logo” para dibujar distintas figuras en un lienzo a partir de comandos de reconocimiento que el software tiene programados.

Además, se utiliza el lenguaje Java ya que este nos proporciona una buena interfaz gráfica la cual es vital para lograr una buena apreciación de las figuras y formas dibujadas, de la misma forma, su uso en el ámbito laboral es ampliamente conocido por lo que usarlo es una gran ventaja al proporcionar un ambiente simple y conocido para la mayor parte de las personas.

**DESARROLLO**

*GRAMÁTICA:*

%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

%%

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);

}

| 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("multiplicar");

}

| exp '+' exp {

$$ = new ParserVal($1.ival);

maquina.agregarOperacion("sumar");

}

| exp '-' exp {

$$ = new ParserVal($1.ival);

maquina.agregarOperacion("restar");

}

|'(' exp ')' {

$$ = new ParserVal($2.ival);

}

| exp COMP exp {

maquina.agregarOperacion("comparar");

$$ = $1;

}

| exp DIFERENTES exp {

maquina.agregarOperacion("compararNot");

$$ = $1;

}

| exp MEN exp {

maquina.agregarOperacion("menor");

$$ = $1;

}

| exp MENI exp {

maquina.agregarOperacion("menorIgual");

$$ = $1;

}

| exp MAY exp {

maquina.agregarOperacion("mayor");

$$ = $1;

}

| exp MAYI exp {

maquina.agregarOperacion("mayorIgual");

$$ = $1;

}

| exp AND exp {

maquina.agregarOperacion("and");

$$ = $1;

}

| exp OR exp {

maquina.agregarOperacion("or");

$$ = $1;

}

| '!' exp {

maquina.agregarOperacion("negar");

$$ = $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"));}

;

stmt:if\_ '(' exp stop\_ ')' '{' linea stop\_ '}' ELSE '{' linea stop\_'}' {

$$ = $1;

maquina.agregar($7.ival, $1.ival + 1);

maquina.agregar($12.ival, $1.ival + 2);

maquina.agregar(maquina.numeroDeElementos() - 1, $1.ival + 3);

}

| if\_ '(' exp stop\_ ')' '{' linea stop\_ '}' nop stop\_{

$$ = $1;

maquina.agregar($7.ival, $1.ival + 1);

maquina.agregar($10.ival, $1.ival + 2);

maquina.agregar(maquina.numeroDeElementos() - 1, $1.ival + 3);

}

| while\_ '(' exp stop\_ ')' '{' linea stop\_ '}' stop\_{

$$ = $1;

maquina.agregar($7.ival, $1.ival + 1);

maquina.agregar($10.ival, $1.ival + 2);

}

| for\_ '(' instrucciones stop\_ ';' exp stop\_ ';' instrucciones stop\_ ')' '{' linea stop\_ '}' stop\_{

$$ = $1;

maquina.agregar($6.ival, $1.ival + 1);

maquina.agregar($9.ival, $1.ival + 2);

maquina.agregar($13.ival, $1.ival + 3);

maquina.agregar($16.ival, $1.ival + 4);

}

| funcion nombreProc '(' ')' '{' linea null\_ '}'

| procedimiento nombreProc '(' ')' '{' linea null\_ '}'

| instruccion '[' arglist ']' ';' {

$$ = new ParserVal($1.ival);

maquina.agregar(null);

}

;

instruccion: FNCT {

$$ = new ParserVal(maquina.agregar((Funcion)($1.obj)));

}

;

procedimiento: PROC { maquina.agregarOperacion("declaracion"); }

;

funcion: FUNC { maquina.agregarOperacion("declaracion"); }

;

nombreProc: VAR {$$ = new ParserVal(maquina.agregar($1.sval));}

;

if\_: IF {

$$ = new ParserVal(maquina.agregarOperacion("\_if\_then\_else"));

maquina.agregarOperacion("stop");//then

maquina.agregarOperacion("stop");//else

maquina.agregarOperacion("stop");//siguiente comando

}

;

while\_: WHILE {

$$ = new ParserVal(maquina.agregarOperacion("\_while"));

maquina.agregarOperacion("stop");//cuerpo

maquina.agregarOperacion("stop");//final

}

;

for\_ : FOR {

$$ = new ParserVal(maquina.agregarOperacion("\_for"));

maquina.agregarOperacion("stop");//condicion

maquina.agregarOperacion("stop");//instrucción final

maquina.agregarOperacion("stop");//cuerpo

maquina.agregarOperacion("stop");//final

}

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

|exp {$$ = $1;}

|instrucciones ',' exp {$$ = $1;}

;

%%

*TABLA DE SÍMBOLOS*

package modelo.compilador;

import java.util.ArrayList;

import java.util.Vector;

public class TablaDeSimbolos {

ArrayList<Par> simbolos;

public TablaDeSimbolos(){

simbolos = new ArrayList<Par>();

}

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;

}

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;

}

public void imprimir(){

for(int i = 0; i < simbolos.size(); i++){

System.out.println(simbolos.get(i).getNombre() + simbolos.get(i).getObjeto().toString());

}

}

*MÁQUINA*

package modelo.compilador;

import java.awt.Color;

import java.lang.reflect.InvocationTargetException;

import java.lang.reflect.Method;

import java.util.ArrayList;

import java.util.Stack;

import modelo.Configuracion;

import modelo.Linea;

public class MaquinaDePila {

private int contadorDePrograma;

private ArrayList memoria;

private Stack pila;

private TablaDeSimbolos tabla;

private boolean stop = false;

private boolean returning = false;

private Stack<Marco> pilaDeMarcos;

private Configuracion configuracionActual;

public MaquinaDePila(TablaDeSimbolos tabla){

configuracionActual = new Configuracion();

contadorDePrograma = 0;

memoria = new ArrayList<Method>();

pila = new Stack();

this.tabla = tabla;

pilaDeMarcos = new Stack();

}

public int numeroDeElementos(){

return memoria.size() + 1;

}

//Funciones que se escriben en memoria

public int agregarOperacion(String nombre){

int posicion = memoria.size();

try{

memoria.add(this.getClass().getDeclaredMethod(nombre, null));

return posicion;

}

catch(Exception e ){

System.out.println("Error al agregar operación " + nombre + ". ");

}

return -1;

}

public int agregar(Object objeto){

int posicion = memoria.size();

memoria.add(objeto);

return posicion;

}

public void agregar(Object objeto, int posicion){

memoria.remove(posicion);

memoria.add(posicion, objeto);

}

public int agregarOperacionEn(String nombre, int posicion){

try{

memoria.add(posicion, this.getClass().getDeclaredMethod(nombre, null));

}

catch(Exception e ){

System.out.println("Error al agregar operación " + nombre + ". ");

}

return posicion;

}

//Funciones que la máquina ejecuta sobre la pila

private void sumar(){

Object matriz2 = pila.pop();

Object matriz1 = pila.pop();

pila.push((double)matriz1 + (double)matriz2);

}

private void restar(){

Object matriz2 = pila.pop();

Object matriz1 = pila.pop();

pila.push((double)matriz1 - (double)matriz2);

}

private void multiplicar(){

Object matriz2 = pila.pop();

Object matriz1 = pila.pop();

pila.push((double)matriz1 \* (double)matriz2);

}

private void negativo(){

Object matriz1 = pila.pop();

System.out.println(matriz1);

pila.push(-(double)matriz1);

}

private void constPush(){

pila.push(memoria.get(++contadorDePrograma));

}

private void varPush(){

pila.push(memoria.get(++contadorDePrograma));

}

private void varPush\_Eval(){

pila.push(tabla.encontrar((String)memoria.get(++contadorDePrograma)));

}

private void compararNot(){

Object A = pila.pop();

Object B = pila.pop();

pila.push((double)A!=(double)B);

}

private void mayor(){

double a;

double b;

Object B = pila.pop();

Object A = pila.pop(); //Se sacan en orden inverso por la forma de la pila

a = (double)A;

b = (double)B;

pila.push(a > b);

}

private void menorIgual(){

double a;

double b;

Object B = pila.pop();

Object A = pila.pop(); //Se sacan en orden inverso por la forma de la pila

a = (double)A;

b = (double)B;

pila.push(a <= b);

}

private void \_while(){

int condicion = contadorDePrograma;

boolean continua = true;

while(continua && !returning){

ejecutar(condicion + 3);

if((boolean)pila.pop()){ //lee el resultado de la condición de la pila

ejecutar((int)memoria.get(condicion+1));//Ejecuta el cuerpo

}

else{

contadorDePrograma = (int)memoria.get(condicion+2);

continua = false;

}

}

}

private void \_if\_then\_else(){

int condicion = contadorDePrograma;

ejecutar(condicion + 4); //Evalúa la condicion

boolean resultado = true;

try{

resultado = (boolean)pila.pop();

}

catch(Exception e ){

}

if(resultado){ //lee el resultado de la condición de la pila

ejecutar((int)memoria.get(condicion+1));//Ejecuta el cuerpo

}

else{

ejecutar((int)memoria.get(condicion+2));

}

contadorDePrograma = (int)memoria.get(condicion+3) - 1; //El -1 es para corregir el aumento del cp al final de cada instrucción

}

private void \_for(){

int condicion = contadorDePrograma;

ejecutar(condicion + 5); // Ejecutamos la primera parte

boolean continua = true;

while(continua && !returning){

ejecutar((int)memoria.get(condicion+1)); //evaluamos la condición

if((boolean)pila.pop()){ //lee el resultado de la condición de la pila

ejecutar((int)memoria.get(condicion+3));//Ejecuta el cuerpo

ejecutar((int)memoria.get(condicion+2));//Ejecuta la última parte del for

}

else{

contadorDePrograma = (int)memoria.get(condicion+4);

continua = false;

}

}

}

private void declaracion(){

tabla.insertar((String)memoria.get(++contadorDePrograma), ++contadorDePrograma); //Apuntamos a la primera instrucción de la función

int invocados = 0;

while(memoria.get(contadorDePrograma) != null || invocados != 0){ //Llevamos cp hasta la siguiente instrucción después de la declaración

if( memoria.get(contadorDePrograma) instanceof Method)

if(((Method)memoria.get(contadorDePrograma)).getName().equals("invocar"))

invocados++;

if( memoria.get(contadorDePrograma) instanceof Funcion)

invocados++;

if(memoria.get(contadorDePrograma) == null)

invocados--;

contadorDePrograma++;

}

}

private void invocar(){

Marco marco = new Marco();

String nombre = (String)memoria.get(++contadorDePrograma);

marco.setNombre(nombre);

contadorDePrograma++;

while(memoria.get(contadorDePrograma) != null){ //Aquí también usamos null como delimitador. Aquí se agregan los parámetros al marco

if(memoria.get(contadorDePrograma) instanceof String){

if(((String)(memoria.get(contadorDePrograma))).equals("Limite")){

Object parametro = pila.pop();

marco.agregarParametro(parametro);

contadorDePrograma++;

}

}

else{

ejecutarInstruccion(contadorDePrograma);

}

}

marco.setRetorno(contadorDePrograma);

pilaDeMarcos.add(marco);

ejecutarFuncion((int)tabla.encontrar(nombre));

}

public void ejecutar(){

//imprimirMemoria();

stop = false;

while(contadorDePrograma < memoria.size())

ejecutarInstruccion(contadorDePrograma);

}

public boolean ejecutarSiguiente(){

//imprimirMemoria();

if(contadorDePrograma < memoria.size()){

ejecutarInstruccion(contadorDePrograma);

return true;

}

return false;

}

public void ejecutar(int indice){//ejecuta hasta que se encuentra Stop

contadorDePrograma = indice;

while(!stop && !returning){

ejecutarInstruccion(contadorDePrograma);

}

stop = false;

}

public void ejecutarFuncion(int indice){

contadorDePrograma = indice;

while(!returning && memoria.get(contadorDePrograma) != null){

ejecutarInstruccion(contadorDePrograma);

}

returning = false;

contadorDePrograma = pilaDeMarcos.lastElement().getRetorno();

pilaDeMarcos.removeElement(pilaDeMarcos.lastElement());

}

public void ejecutarInstruccion(int indice){

//System.out.println("Ejecutando: " + indice);

try{

Object objetoLeido = memoria.get(indice);

if(objetoLeido instanceof Method){

Method metodo = (Method)objetoLeido;

metodo.invoke(this, null);

}

if(objetoLeido instanceof Funcion){

ArrayList parametros = new ArrayList();

Funcion funcion = (Funcion)objetoLeido;

contadorDePrograma++;

while(memoria.get(contadorDePrograma) != null){ //Aquí también usamos null como delimitador. Aquí se agregan los parámetros al marco

if(memoria.get(contadorDePrograma) instanceof String){

if(((String)(memoria.get(contadorDePrograma))).equals("Limite")){

Object parametro = pila.pop();

parametros.add(parametro);

contadorDePrograma++;

}

}

else{

ejecutarInstruccion(contadorDePrograma);

}

catch(Exception e){}

}

public Configuracion getConfiguracion(){

return configuracionActual;

}

public static class Girar implements Funcion{

@Override

public void ejecutar(Object A, ArrayList parametros) {

Configuracion configuracion = (Configuracion)A;

int angulo = (configuracion.getAngulo() + (int)(double)parametros.get(0))%360;

configuracion.setAngulo(angulo);

}

}

public static class Avanzar implements Funcion{

@Override

public void ejecutar(Object A, ArrayList parametros) {

Configuracion configuracion = (Configuracion)A;

int angulo = configuracion.getAngulo();

double x0 = configuracion.getX();

double y0 = configuracion.getY();

double x1 = x0 + Math.cos(Math.toRadians(angulo))\*(double)parametros.get(0);

double y1 = y0 + Math.sin(Math.toRadians(angulo))\*(double)parametros.get(0);

configuracion.setPosicion(x1, y1);

configuracion.agregarLinea(new Linea((int)x0,(int)y0,(int)x1,(int)y1, configuracion.getColor()));

}

}

public static class CambiarColor implements Funcion{

@Override

public void ejecutar(Object A, ArrayList parametros) {

Configuracion configuracion = (Configuracion)A;

configuracion.setColor(new Color((int)(double)parametros.get(0)%256, (int)(double)parametros.get(1)%256, (int)(double)parametros.get(2)%256));

}

}

}