Universidad Austral de Chile
Conocimiento y Naturaleza

Ingeniería Civil en INFORMÁTICA UACh

Traductor dirigido por sintaxis para el lenguaje BLOCK

Compiladores - INFO165

Flex

Bison

Profesora: Maria Eliana de la Maza

Integrantes:

- Diego Vera
- Nicolás Robledo
- Sebastián Lara

Fecha de entrega: 15/11/2021

Introducción

La actividad consiste en crear un traductor dirigido por sintaxis para un lenguaje llamado BLOCK. Este lenguaje nos permite dibujar figuras geométricas mediante una serie de instrucciones ingresadas por un usuario.

El lenguaje BLOCK posee las siguientes instrucciones:

a. dibujar: primera instrucción de cualquier programa

b. fin: última instrucción de cualquier programa

c. linea(a,b,c,d): dibuja una línea entre las coordenadas (a,b) y (c,d)

d. cuadro(a,b,c,d): dibuja un rectángulo cuyo vértice superior izquierdo está en la coordenada (a,b) y el inferior derecho en la (c,d).

e. redondo(a,b,c): dibuja un círculo con centro en la coordenada (a,b) y radio c.

f. triangulo(a,b,c,d,e,f): dibuja un triángulo cuyos vértices están en las coordenadas (a,b), (c,d) y (e,f).

g. color(a): fija el color de la(s) próxima(s) figura(s) a dibujar en el color a, donde a puede ser rojo, verde, azul, amarillo o blanco

h. relleno(si/no): fija si las siguientes figuras serán rellenas o no.

i. asignar variable=color o entero o si o no

Para esto, se creó un analizador léxico y un analizador sintáctico utilizando los programas FLEX y BISON respectivamente y así poder construir un compilador para el lenguaje BLOCK. Para la parte gráfica se utilizó Miniwin.

Antes de continuar, debemos definir algunos conceptos:

¿Qué es un compilador?
Un compilador es un programa informático que traduce un programa escrito en algún lenguaje de programación, este programa se denomina código fuente y es traducido a un lenguaje de alto, medio o bajo nivel.

¿Qué es un analizador léxico?
Dentro del proceso de compilación se necesita en primera fase un analizador léxico el cual se encarga de leer los caracteres de entrada y elaborar como salida una secuencia de componentes léxicos que utiliza el analizador sintáctico para hacer el análisis.

¿Qué es un analizador sintáctico?
La fase de compilación del analizador sintáctico se encarga de procesar la secuencia de componentes léxicos (tokens) mediante alguna gramática dada para así construir una estructura de datos.

Desarrollo

Parte I. Analizador léxico

Se comenzó construyendo el analizador léxico, para esto decidimos utilizar la herramienta Flex. Flex nos permite generar analizadores léxicos a partir de una serie de expresiones regulares y así producir código en lenguaje C.

Las expresiones regulares utilizadas son las siguientes:

Palabras reservadas: "dibujar" "linea" "redondo" "cuadro" "triangulo" "color" "relleno" "azul" "amarillo" "rojo" "verde" "blanco" "si" "no" "fin"

Símbolos: "(" ")" "," "="

El proceso de compilación del analizador es el siguiente:

fig.1: estructura de la creación de un analizador léxico con FLEX

En donde:

- **lexico.l**: es el fichero que creamos consta de lo siguiente:
    - Definiciones: acá tenemos algunas declaraciones de definiciones que agregamos para una mayor simplificación.

```
ID          [A-Z]([a-z]*[0-9]+)*
NUMERO      [0-9][0-9]*
```

fig. 2: definiciones en lexico.l

- Reglas y acciones: tenemos las expresiones regulares de la forma patrón {acción}.

```
"dibujar" {return yy::trayect_parser::make_DIBUJAR(loc);}
"linea" {return yy::trayect_parser::make_LINEA(loc);}
"redondo" {return yy::trayect_parser::make_REDONDO(loc);}

"cuadro" {return yy::trayect_parser::make_CUADRO(loc);}
"triangulo" {return yy::trayect_parser::make_TRIANGULO(loc);}
"color" {return yy::trayect_parser::make_COLOR(loc);}
"relleno" {return yy::trayect_parser::make_RELLENO(loc);}
"azul" {return yy::trayect_parser::make_AZUL(loc);}
"amarillo" {return yy::trayect_parser::make_AMARILLO(loc);}
"rojo" {return yy::trayect_parser::make_ROJO(loc);}
"verde" {return yy::trayect_parser::make_VERDE(loc);}
"blanco" {return yy::trayect_parser::make_BLANCO(loc);}
"si" {return yy::trayect_parser::make_SI(loc);}
"no" {return yy::trayect_parser::make_NO(loc);}
"fin" {return yy::trayect_parser::make_FIN(loc);}

"(" {return yy::trayect_parser::make_PARA(loc);}
")" {return yy::trayect_parser::make_PARC(loc);}
"," { return yy::trayect_parser::make_COMA(loc);}
"=" {return yy::trayect_parser::make_IGUAL(loc);}
```

fig. 3: expresiones regulares en lexico.l

- **FLEX**: es la herramienta elegida que generará el código en C++ a partir del fichero .l
- **lex.cpp**: compilamos lexico.l en la terminal con 'flex lexico.l' para así tener como salida lex.cpp

Al obtener nuestro código generado lex.cpp, este contiene una función llamada yylex() la cual al ejecutarla analizará las entradas en búsqueda de coincidencias y respectivamente ejecutar el respectivo código en C++

Parte II. Analizador sintáctico

Luego de construir el analizador léxico, se construyó el analizador sintáctico sintactico.yy utilizando la herramienta Bison. El cual al compilar nos retorna un ejecutable sintactico.tab.cc con las constantes asociadas a los tokens, además de variables y estructuras de datos necesarias para el analizador léxico.

Para la construcción del traductor es necesario utilizar ambas herramientas (Flex y Bison) en conjunto gracias a la función que contiene el código generado por Bison yyparse(); la cual en sí es nuestro analizador, el cual a su vez es el encargado de llamar internamente a la función yylex() cuando necesite un token.

Nos guiamos por el siguiente formato de especificación de Bison, la cual consta de tres secciones .

sección de definiciones

```
%{
      /*delimitadores de código C*/
%}

%%
```

sección de reglas

```
%%
```

sección de funciones de usuario

En la sección de definiciones es donde definimos que usaremos la librería MiniWin para poder abrir una ventana y pintar en ella.

Para soportar una interfaz pura con analizador(y Scanner) la técnica “parsing context” es conveniente: La cual es una estructura que contiene todos los datos para intercambiar. Dado que, además de simplemente iniciar el análisis, hay varias tareas auxiliares para ejecutar (abrir el archivo para escanear, crear una instancia del analizador, etc.). Por lo tanto transformamos la estructura de contexto de análisis simple en una clase de *controlador de análisis completo(parsing driver class).*

Entonces para declarar esta clase controladora creamos el fichero draw.h. de la cual nos basamos de la siguiente estructura:

```
#ifndef DRAW_HH
# define DRAW_HH
# include <string>
# include <map>
# include "parser.hh"
```

Luego viene la declaración de la función de escaneo. Flex espera que la firma de yylex se defina en la macro YY_DECL y el analizador de C ++ espera que se declare. Podemos factorizar ambos de la siguiente manera.

```
// Darle a Flex el prototipo de yylex que queremos ...
# definir YY_DECL \
  yy :: parser :: symbol_type yylex (driver & drv)
// ... y declararlo por el bien del analizador.
  YY_DECL;
```

Por nuestra parte definimos YY_DECL en el controlador en draw.h para referenciar en el fichero lexico.l, para obtener un código más ordenado.

Por consiguiente declaramos la clase controladora draw.h con las funciones que necesitaremos para usar esta técnica.

```
class trayect_draw
{
public:
      float resultado;
      void iniciarScanner();
      void terminarScanner();

      int parse(const std::string& archivo);

      std::string file;
};
```

Y finalmente para poder llama este analizador a través de la rutina main() del fichero main.cpp

En la sección de definiciones también declaramos los métodos a utilizar y que posteriormente son creados en la sección de código adicional.
En la sección de reglas es donde definimos los tokens y los tipos de valor utilizados para las variables no terminales de nuestra gramática a utilizar.

```
/*Declaración de tokens*/

//Listado de terminales
%token <float> NUMERO
%token <std::string> ID
%token DIBUJAR
%token LINEA
%token REDONDO
%token CUADRO
%token TRIANGULO
%token COLOR
%token RELLENO
//%token ASIGNAR
%token AZUL
%token AMARILLO
%token ROJO
%token VERDE
%token BLANCO
%token SI
%token NO
%token FIN 0 "eof"

%token PARA
%token PARC
%token COMA
%token IGUAL

//Listado de no terminales

%type <float> INST
%type <float> E
%type <float> LINEAR
%type <float> CUADROR
%type <float> REDONDOR
%type <float> TRIANGULOR
%type <float> COLORR
%type <float> RELLENOR
%type <std::string> var
%type <float> constante
```

fig. 4: declaración de tokens y tipos (types)

La gramática que se adecua a los requisitos para este tipo de lenguaje es la siguiente:

```
/*Inicio de la gramatica*/
s :                     DIBUJAR INST FINALIZAR
                        ;

INST:   INST INST |
        LINEAR |
        CUADROR |
        REDONDOR |
        TRIANGULOR |
        COLORR |
        RELLENOR //|
        //ASIGNARR
        ;
LINEAR: LINEA PARA E COMA E COMA E COMA E PARC {dibujaLinea($3,$5,$7,$9);}
        ;

CUADROR: CUADRO PARA E COMA E COMA E COMA E PARC {dibujaCuadro($3,$5,$7,$9);}
        ;

REDONDOR: REDONDO PARA E COMA E COMA E PARC {dibujaRedondo($3,$5,$7);}
        ;

TRIANGULOR: TRIANGULO PARA E COMA E COMA E COMA E COMA E COMA E PARC {dibujaTriangulo($3,$5,$7,$9,$11,$13);}
        ;

COLORR: COLOR PARA ID PARA {guardaColor($3);} |
        COLOR PARA AZUL PARC {guardaAzul();} |
        COLOR PARA AMARILLO PARC {guardaAmarillo();} |
        COLOR PARA ROJO PARC {guardaRojo();} |
        COLOR PARA VERDE PARC {guardaVerde();} |
        COLOR PARA BLANCO PARC {guardaBlanco();}
        ;
RELLENOR: RELLENO PARA ID PARC {guardaRelleno($3);} |
        RELLENO PARA SI PARC {guardaBoolean("si");} |
        RELLENO PARA NO PARC {guardaBoolean("no");}
        ;

E: var | constante;

var: ID { $$ = $1;}
        ;
constante:  NUMERO { $$ = $1;}
        ;

FINALIZAR: FIN {exit (-1);}
        ;
```

fig. 5: gramática usada en el analizador sintáctico

Finalmente en la sección de funciones de usuario o código adicional es donde escribimos nuestros métodos que a su vez llaman a los métodos definidos en la librería MiniWin, por ejemplo:

```
void dibujaRedondo(float posx,float posy,float ratio){

        if (rellenar == "si"){
                miniwin::circulo_lleno( posx, posy, ratio);
                miniwin::refresca();
        }
       else if (rellenar == "no"){
                miniwin::circulo( posx, posy, ratio);
                miniwin::refresca();
       }
}
```

fig. 6: métodos de ejemplo dibujaRedondo

Los métodos utilizados son los siguientes:

```
//Declaracion de metodos
void yyerror(char *s);
void dibujaLinea(float posx1,float posy1,float posx2,float posy2);
void dibujaCuadro(float posx1,float posy1,float posx2,float posy2);
void dibujaRedondo(float posx,float posy , float ratio);
void dibujaTriangulo(float posx1,float posy1,float posx2,float posy2,float posx3,float posy3);
void guardaColor(std::string colour);
void guardaAzul();
void guardaAmarillo();
void guardaBlanco();
void guardaRojo();
void guardaVerde();
void guardaRelleno(std::string relleno);
void guardaBoolean(std::string boolean);
void asignaAzul(std::string azul);
void asignaAmarillo(std::string amarillo);
void asignaBlanco(std::string blanco);
void asignaRojo(std::string rojo);
void asignaVerde(std::string verde);
void asignaSi();
void asignaNo();
void asignaConstante();
```

fig.7: métodos utilizados en el analizador sintáctico

Donde usando la librería MiniWin los métodos tienen sus respectivas tareas:

```
dibujaLinea(a,b,c,d): dibuja una línea entre las coordenadas (a,b) y (c,d)
dibujaCuadro(a,b,c,d): dibuja un rectángulo cuyo vértice superior izquierdo
está en la coordenada (a,b) y el inferior derecho en la (c,d).
dibujaRedondo(a,b,c): dibuja un circulo con centro en la coordenada (a,b) y
radio c.
dibujaTriangulo(a,b,c,d,e,f): dibuja un triángulo cuyos vértices están en
las coordenadas (a,b), (c,d) y (e,f).
guardaColor(A): Define el color almacenado en A.
guardaAzul(): Define color Azul.
      .
      .
guardaVerde(): Define color Verde.
guardaRelleno(): Guarda en una variable global "rellenar" si se debe
rellenar la figura dibujada.
guardaBooleano(): Guarda en una variable las palabras "si" o "no".
asignaAzul():Asigna en una variable ID la palabra "azul".
      .
      .
asignaVerde():Asigna en una variable ID la palabra "verde".
asignaSi(): asigna en una variable ID la palabra "si".
asignaNo(): asigna en una variable ID la palabra "no".
asignaConstante(): Asigna en una variable ID una constante.
```

Para la parte gráfica, utilizamos una librería llamada **MiniWin**, ésta abre una ventana y en ella se pueden dibujar diversas figuras geométricas como también colorearlas.

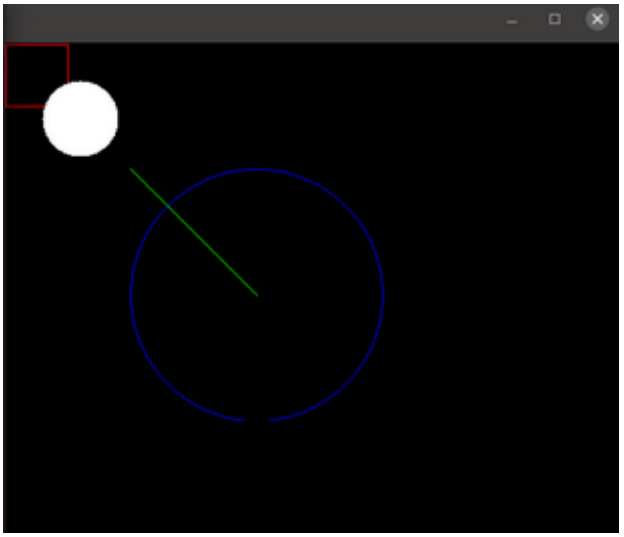


fig. 8: ventana de MiniWin

Problemas encontrados:

- Problema con ASIGNAR: Debido que para la declaración de los tipos de variables no terminales usamos api.value.type nos encontramos con un problema para separar el string de los identificadores y el valor (string o float) la cual no fuimos capaces de solucionar, por la implementación de nuestro lenguaje cuenta sin el token y la función de “asignar”, sin embargo para poder hacer un funcionamiento sin este token lo sacamos de la gramática.
- Problema con Triángulo relleno: Debido a que la librería MiniWin no cuenta con un método para dibujar un triángulo, tuvimos que dibujarlo con 3 líneas, sin embargo para triángulo relleno es bastante complicado pintar un triángulo ya que necesita bastante cálculo operacional. Por lo que alternativamente distinguimos ese triángulo relleno con una Letra R dentro de la ventana que abre MiniWin.

Especificaciones del programa

- **Nombre de los programas:**
  - lexico.l
  - sintáctico.yy
  - main.cpp
  - draw.h

- **Requerimientos básicos de hardware:**
  - Sistema operativo: Linux
  - Periféricos: Teclado, mouse, pantalla

- **Requerimientos básicos de software:**
  - Flex 2.6.4
  - Bison 3.8
  - Miniwin 0.2.1

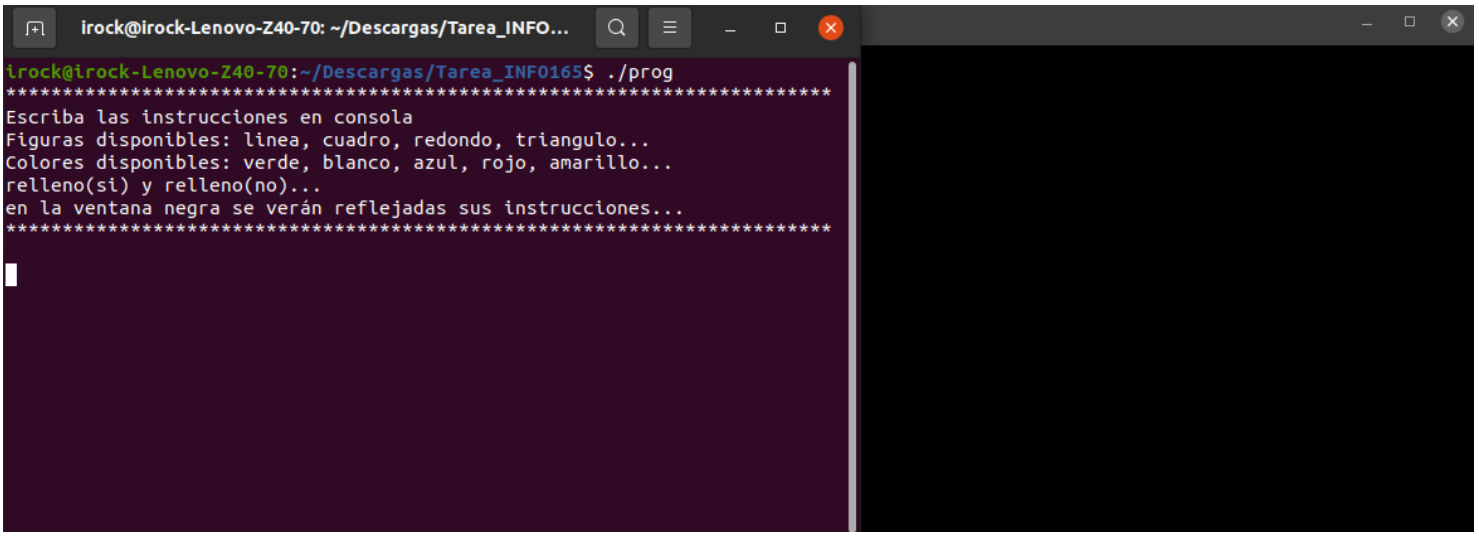
- **Compilación y Ejecución:**
    - Para compilar, de deben ejecutar los siguientes comandos en una terminal del sistema operativo:

```
user: flex lexico.l
user: bison -d sintactico.yy
user: g++ -Wall -g -DDEBUG main.cpp draw.cpp sintactico.tab.cc
lex.cpp miniwin.cpp -o prog -pthread -lX11 -lfl -std=c++11
user: ./prog
```

    - En sistemas operativos basados en Linux, desde una terminal se puede usar el archivo “makefile” para compilar todos los programas a la vez. Luego se ejecuta el comando ./prog para ejecutar

```
user: make
user: ./prog
```

- **Ejemplo de funcionamiento:**



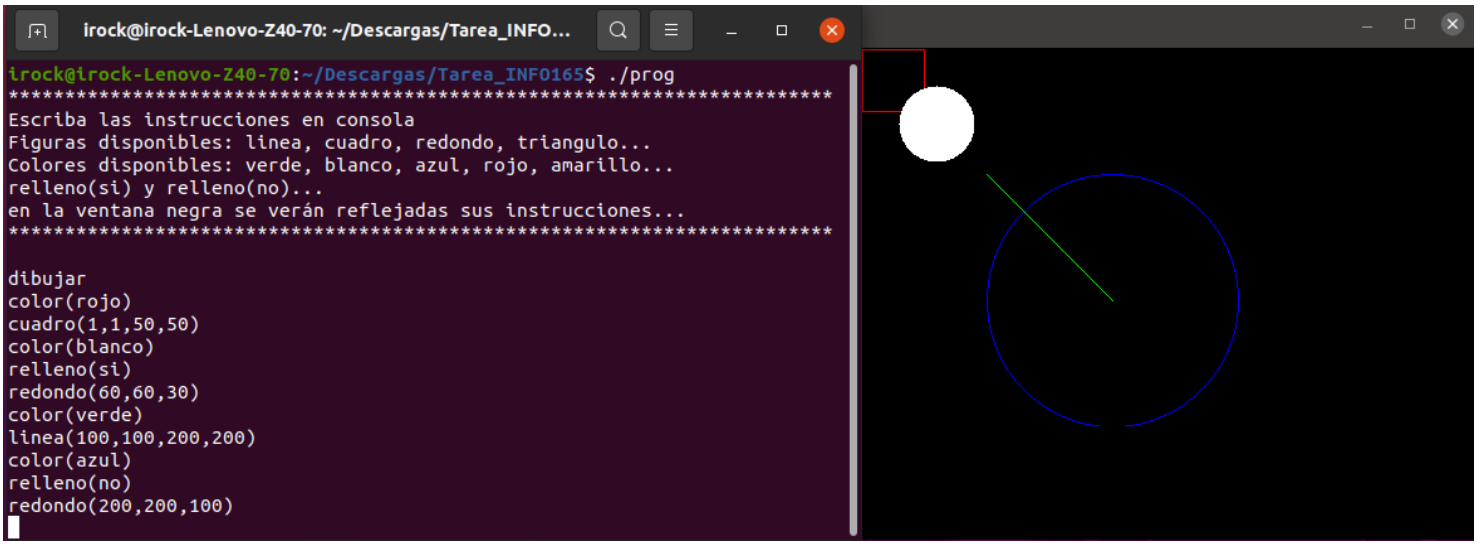


fig. 9 y fig 10: ejemplo de ejecución de programa

Conclusiones

Podemos concluir que se logró reforzar el aprendizaje visto en clases del proceso de un compilador, específicamente del análisis léxico y sintáctico al hacer uso herramientas las herramientas de programación Flex y Bison.

En general, nuestro traductor dirigido por sintaxis tiene un buen desempeño salvo por el problema mencionado anteriormente.

Algunas mejoras que se podrían hacer son por ejemplo el poder dibujar otros tipos de figuras geométricas, ya sean elipses, trapecios, etc. También se podría agregar un buen manejo de errores a la hora de ingresar instrucciones en la terminal, ya que actualmente si el usuario se equivoca con algún tipeo, el programa se caerá.

Bibliografía

- http://webdiis.unizar.es/asignaturas/LGA/material_2004_2005/Intro_Flex_Bison.pdf
- http://arantxa.ii.uam.es/~mdlcruz/docencia/compiladores/2002_2003/compiladores_02_03_yacc_bison.pdf
- https://www.gnu.org/software/bison/manual/bison.html#Value-Type
- https://www.gnu.org/software/bison/manual/html_node/Calc_002b_002b-Parsing-Driver.html

Anexo

1. **lexico.l**

```
%{
    #include <cerrno>
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include "draw.h"
    #include "sintactico.tab.hh"
    #undef yywrap
    #define yywrap() 1
    static yy::location loc;
    int lineno = 1; // initialize to 1


    //void ret_print(char *tipo_token);
    //void yyerror();
%}

%option noyywrap nounput batch debug noinput
%option outfile="lex.cpp"

%x ML_COMMENT

ID          [A-Z]([a-z]*[0-9]+)*
NUMERO      [0-9][0-9]*
DELIMITADOR [\t\r\f" "]
```

```
%%

{NUMERO}    {return yy::trayect_parser::make_NUMERO(strtol(yytext, NULL,
10),loc);}
{ID}        {return yy::trayect_parser::make_ID(yytext,loc);}

<ML_COMMENT>"\n"        { lineno += 1; }

"dibujar" {return yy::trayect_parser::make_DIBUJAR(loc);}
"linea" {return yy::trayect_parser::make_LINEA(loc);}
"redondo" {return yy::trayect_parser::make_REDONDO(loc);}

"cuadro" {return yy::trayect_parser::make_CUADRO(loc);}
"triangulo" {return yy::trayect_parser::make_TRIANGULO(loc);}
"color" {return yy::trayect_parser::make_COLOR(loc);}
"relleno" {return yy::trayect_parser::make_RELLENO(loc);}
"azul" {return yy::trayect_parser::make_AZUL(loc);}
"amarillo" {return yy::trayect_parser::make_AMARILLO(loc);}
"rojo" {return yy::trayect_parser::make_ROJO(loc);}
"verde" {return yy::trayect_parser::make_VERDE(loc);}
"blanco" {return yy::trayect_parser::make_BLANCO(loc);}
"si" {return yy::trayect_parser::make_SI(loc);}
"no" {return yy::trayect_parser::make_NO(loc);}
"fin" {return yy::trayect_parser::make_FIN(loc);}

"(" {return yy::trayect_parser::make_PARA(loc);}
")" {return yy::trayect_parser::make_PARC(loc);}
"," { return yy::trayect_parser::make_COMA(loc);}
"=" {return yy::trayect_parser::make_IGUAL(loc);}
"\n"            { lineno += 1; }

{DELIMITADOR}   {}
.               {printf("ERROR LEXICO %s en lineno %i\n",yytext,lineno+1);
            exit(0);}
<<EOF>>         {return yy::trayect_parser::make_FIN(loc);}

%%


void trayect_draw::iniciarScanner()
{
```

```
  yy_flex_debug = false;
  yyin = stdin;
}
void trayect_draw::terminarScanner()
{
  fclose(yyin);
}
```

2. **sintactico.yy**

```
%skeleton "lalr1.cc" /* -*- C++ -*- */
%require "3.0.2"
%defines
%define api.parser.class {trayect_parser}
%define api.token.constructor
%define api.namespace {yy}
%define api.value.type variant
%define parse.assert
%code requires

{

/**********************
 * Declaraciones en C *
 **********************/



 //Importacion de librerias
  #include <stdio.h>
  #include <stdlib.h>
  #include <math.h>
  #include <utility>
  #include <vector>
  #include "string.h"
  #include "miniwin.h"

  class trayect_draw;

  extern int yylex(void);
  extern char *yytext;
  extern FILE *yyin;
```

```
  //Declaracion de metodos
  void yyerror(char *s);
  void dibujaLinea(float posx1,float posy1,float posx2,float posy2);
  void dibujaCuadro(float posx1,float posy1,float posx2,float posy2);
  void dibujaRedondo(float posx,float posy , float ratio);
  void dibujaTriangulo(float posx1,float posy1,float posx2,float
posy2,float posx3,float posy3);
  void guardaColor(std::string colour);
  void guardaAzul();
  void guardaAmarillo();
  void guardaBlanco();
  void guardaRojo();
  void guardaVerde();
  void guardaRelleno(std::string relleno);
  void guardaBoolean(std::string boolean);
  void asignaAzul(std::string azul);
  void asignaAmarillo(std::string amarillo);
  void asignaBlanco(std::string blanco);
  void asignaRojo(std::string rojo);
  void asignaVerde(std::string verde);
  void asignaSi();
  void asignaNo();
  void asignaConstante();
}

%param { trayect_draw& draw }
%locations
%define parse.trace
%define parse.error verbose
%code
{
#include "draw.h"
#include <iostream>

std::string rellenar = "no";
std::vector<std::pair<std::string, float> > v;
std::vector<std::pair<std::string, std::string> > cl;
}
%define api.token.prefix {TOK}

/************************
```

```
  Declaraciones de Bison *
 ************************/


/*Declaración de tokens*/

//Listado de terminales
%token <float> NUMERO
%token <std::string> ID
%token DIBUJAR
%token LINEA
%token REDONDO
%token CUADRO
%token TRIANGULO
%token COLOR
%token RELLENO
//%token ASIGNAR
%token AZUL
%token AMARILLO
%token ROJO
%token VERDE
%token BLANCO
%token SI
%token NO
%token FIN 0 "eof"

%token PARA
%token PARC
%token COMA
%token IGUAL

//Listado de no terminales

%type <float> INST
%type <float> E
%type <float> LINEAR
%type <float> CUADROR
%type <float> REDONDOR
%type <float> TRIANGULOR
%type <float> COLORR
%type <float> RELLENOR
%type <std::string> var
```

```
%type <float> constante
//%type <float> ASIGNARR

%start s

%%



/**********************
 * Reglas Gramaticales *
 **********************/

/*Inicio de la gramatica*/
s :             DIBUJAR INST FINALIZAR
                ;

INST:   INST INST |
        LINEAR |
        CUADROR |
        REDONDOR |
        TRIANGULOR |
        COLORR |
        RELLENOR //|
        //ASIGNARR
        ;


LINEAR: LINEA PARA E COMA E COMA E COMA E PARC {dibujaLinea($3,$5,$7,$9);}
        ;

CUADROR: CUADRO PARA E COMA E COMA E COMA E PARC
{dibujaCuadro($3,$5,$7,$9);}
        ;

REDONDOR: REDONDO PARA E COMA E COMA E PARC {dibujaRedondo($3,$5,$7);}
        ;

TRIANGULOR: TRIANGULO PARA E COMA E COMA E COMA E COMA E COMA E PARC
{dibujaTriangulo($3,$5,$7,$9,$11,$13);}
        ;
```

```
COLORR: COLOR PARA ID PARA {guardaColor($3);} |
        COLOR PARA AZUL PARC {guardaAzul();} |
        COLOR PARA AMARILLO PARC {guardaAmarillo();} |
        COLOR PARA ROJO PARC {guardaRojo();} |
        COLOR PARA VERDE PARC {guardaVerde();} |
        COLOR PARA BLANCO PARC {guardaBlanco();}
        ;

RELLENOR: RELLENO PARA ID PARC {guardaRelleno($3);} |
        RELLENO PARA SI PARC {guardaBoolean("si");} |
        RELLENO PARA NO PARC {guardaBoolean("no");}
        ;
/*
ASIGNARR: ASIGNAR ID IGUAL AZUL {asignaAzul("AZUL");} |
          ASIGNAR ID IGUAL AMARILLO {asignaAmarillo("AMARILLO");} |
          ASIGNAR ID IGUAL ROJO {asignaRojo("ROJO");} |
          ASIGNAR ID IGUAL VERDE {asignaVerde("VERDE");} |
          ASIGNAR ID IGUAL BLANCO {asignaBlanco("BLANCO");} |
          ASIGNAR ID IGUAL SI {asignaSi("SI");} |
          ASIGNAR ID IGUAL NO {asignaNo("NO");} |
          ASIGNAR ID IGUAL E {asignaConstante($1,$3);}
          ;
*/

E: var | constante;

var: ID { $$ = $1;}
        ;

constante:  NUMERO { $$ = $1;}
        ;

FINALIZAR: FIN {exit (-1);}
        ;
%%

/**********************
 * Codigo C Adicional *
 **********************/

void yy::trayect_parser::error(const location_type& lugar, const
std::string& lexema)
```

```
{
        std::cout << "Error Sintactico " << lexema << std::endl;
        exit(0);

}

/*************************
  *Funciones *
 *************************/

 void dibujaLinea(float posx1,float posy1,float posx2,float posy2){

         miniwin::linea(posx1,posy1,posx2,posy2);
         miniwin::refresca();
 }

 void dibujaCuadro(float posx1,float posy1,float posx2,float posy2){

         if (rellenar == "si"){
                 miniwin::rectangulo_lleno(posx1,posy1,posx2,posy2);
                 miniwin::refresca();
         }
        else if (rellenar == "no"){
                miniwin::rectangulo(posx1,posy1,posx2,posy2);
                miniwin::refresca();
        }

 }

  void dibujaRedondo(float posx,float posy,float ratio){

         if (rellenar == "si"){
                 miniwin::circulo_lleno( posx, posy, ratio);
                 miniwin::refresca();
         }
        else if (rellenar == "no"){
                miniwin::circulo( posx, posy, ratio);
                miniwin::refresca();
        }
 }

  void dibujaTriangulo(float posx1,float posy1,float posx2,float
```

```
posy2,float posx3,float posy3){
         if (rellenar == "si"){
                 miniwin::rectangulo_lleno(posx1,posy1,posx2,posy2 );
                 miniwin::refresca();
         }
        else if (rellenar == "no"){
                miniwin::linea(posx1,posy1,posx2,posy2);
                miniwin::linea(posx2,posy2,posx3,posy3);
                miniwin::linea(posx1,posy1,posx3,posy3);
                miniwin::refresca();
        }

 }

void guardaColor(std::string colour ){
        if(colour == "AMARILLO"){
                miniwin::color(miniwin::AMARILLO);
                miniwin::refresca();
        }
        else if(colour == "ROJO"){
                miniwin::color(miniwin::ROJO);
                miniwin::refresca();
        }
        else if(colour == "VERDE"){
                miniwin::color(miniwin::VERDE);
                miniwin::refresca();
        }
        else if(colour == "BLANCO"){
                miniwin::color(miniwin::BLANCO);
                miniwin::refresca();
        }

        else if(colour =="AZUL"){
                miniwin::color(miniwin::AZUL);
                miniwin::refresca();
        }

}

void guardaRelleno(std::string relleno){

        if (relleno == "si"){
```

```
                ::rellenar = "si";
                miniwin::refresca();
        }
        else if (relleno == "no"){
                ::rellenar ="no";
                miniwin::refresca();
        }
}
void guardaBoolean(std::string boolean){
        ::rellenar = boolean;
}


void guardaRojo(){
     miniwin::color(miniwin::ROJO);
        miniwin::refresca();
}
void guardaVerde(){
     miniwin::color(miniwin::VERDE);
        miniwin::refresca();
}
void guardaAzul(){
     miniwin::color(miniwin::AZUL);
        miniwin::refresca();
}
void guardaAmarillo(){
     miniwin::color(miniwin::AMARILLO);
        miniwin::refresca();
}
void guardaBlanco(){
     miniwin::color(miniwin::BLANCO);
        miniwin::refresca();
}
```

3. **draw.h**

```
#ifndef DRAW
# define DRAW
# include <string>
# include "sintactico.tab.hh"
#define YY_DECL \
yy::trayect_parser::symbol_type yylex (trayect_draw& draw)
YY_DECL;

class trayect_draw
{
public:
  float resultado;
  void iniciarScanner();
  void terminarScanner();

  int parse(const std::string& archivo);

  std::string file;
};
#endif
```

4. **main.cpp**

```
#include <stdio.h>
#include "draw.h"
#include <iostream>
#include <string.h>
using namespace std;
int main()
{

cout<<"**********************************************************
**********"<<endl;
  cout<<"Escriba las instrucciones en consola"<<endl;
  cout<<"Figuras disponibles: linea, cuadro, redondo,
triangulo..."<<endl;
  cout<<"Colores disponibles: verde, blanco, azul, rojo,
amarillo..."<<endl;
```

```
  cout<<"relleno(si) y relleno(no)..."<<endl;
  cout<<"en la ventana negra se verán reflejadas sus
instrucciones..."<<endl;

cout<<"**********************************************************
**********"<<endl<<endl;
  trayect_draw draw;
  draw.parse("f");
  return 0;
```