# Código fuente.y

%{

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

%}

//Se incluyen estas tres librerías ya que stdio.h es la biblioteca estándar del lenguaje C para operaciones, math.h para operaciones matemáticas y stdlib.h para la gestión de memoria dinámica.

%union{

int valor;

char valorString[32];

}

//Union, contiene los atributos de los tipos de dato que nuestros tokens van a reconocer.

%token <valor> CONSTANTE

%token <valorString> SENTENCIA

%token COMA

%token LEER

%token ESCRIBIR

%token INICIO

%token <valorString> ID

%token SUMA RESTA

%token PARENIZQUIERDO PARENDERECHO

%token PUNTOYCOMA

%token FIN

%token FDT

//Acá definimos todos los tokens del lenguaje micro con su tipo de dato en caso de ser necesario.

%left SUMA RESTA

%left ASIGNACION

//Estas son las precedencias de cada uno de los tokens, en el lenguaje micro solo debemos especificar los de suma, resta y asignación.

%type <valor> expresion

%type <valor> primaria

//Acá especificamos los types y sus tipos de dato.

%start programa

//Programa será nuestro axioma.

%%

/\*A partir de acá hacemos la definición en BNF de todos los elementos terminales.

programa: INICIO listaSentencias FIN { imprimirVariables(); }

listaSentencias: sentencia

| sentencia listaSentencias

;

sentencia: ID ASIGNACION expresion PUNTOYCOMA { escribirVariable($1,$3); }

| LEER PARENIZQUIERDO listaIdentificadores PARENDERECHO PUNTOYCOMA

| ESCRIBIR PARENIZQUIERDO listaExpresiones PARENDERECHO PUNTOYCOMA

;

listaIdentificadores: ID { escanearVariable($1); }

| listaIdentificadores COMA ID { escanearVariable($3); }

;

listaExpresiones: expresion { printf("%d\n", $1); }

| listaExpresiones COMA expresion { printf("%d\n", $3); }

;

expresion: primaria { $$=$1; /\* printf("prim: %d\n", $1);\*/ }

| primaria SUMA expresion { $$=$1+$3; /\*printf("prim + prim: %d\n", $$);\*/ }

| primaria RESTA expresion { $$=$1-$3; /\*printf("prim - prim: %d\n", $$);\*/ }

;

primaria: ID { $$ = leerVariable($1); /\*Aca lee los id\*/ }

| CONSTANTE { $$=$1; }

| PARENIZQUIERDO expresion PARENDERECHO { $$=$2; /\*printf("(exp): %d\n",$2);\*/ }

;

\*/

%%

#define MAX\_VARIABLES 30

#define MAX\_NOMBRE 32

#define NO\_ENCONTRADO 55

struct{

char nombre[MAX\_NOMBRE];

int valor;

}variables[MAX\_VARIABLES];

// Acá definimos en un struct el valor máximo de un nombre y de las variables.

extern FILE \*yyin; // Se agrega para que pueda enviarse un puntero a un archivo y ser parseado

int yyerror(char \*s) {

printf("Error: no se reconoce el programa.\n");

} //Acá, especificamos la salida ante un error.

void escanearVariable(char \*id) {

int valorVariable;

printf("\nIngrese un valor para %s:", id);

scanf("%d", &valorVariable);

escribirVariable(id, valorVariable);

} //Función auxiliar para ingresar una variable.

void modVariable(char \*Name,int Val){

int x;

x = buscarVariable(Name);

if (x!=NO\_ENCONTRADO){

variables[x].valor=Val;

}

}//Función auxiliar para modificar una variable.

int buscarVariable(char \*Name){

for(int i=0; i<MAX\_VARIABLES; i++){

if (strcmp(Name,variables[i].nombre)==0){

return i;

}

}

return NO\_ENCONTRADO;

}//Función auxiliar para buscar una variable.

void escribirVariable(char \*Name,int Val){

int x;

int i;

i = buscarVariable(Name);

if (i==NO\_ENCONTRADO){

x = buscarVariable("");

if (x!=NO\_ENCONTRADO){

// puts("entro y copia\n");

strcpy(variables[x].nombre,Name);

variables[x].valor=Val;

}

}

if(i!=NO\_ENCONTRADO){

modVariable(Name,Val);

}

}//Función auxiliar para escribir una variable, si la variable existe, la sobreescribe, sinó busca el primer espacio vacío y la guarda en el espacio correspondiente.

int leerVariable (char \*Name){

int x;

x = buscarVariable(Name);

return variables[x].valor;

} //Función auxiliar para leer una variable, se le asigna un valor a X y retorna el valor de dicha variable.

void imprimirVariables(void)

{

int i;

printf("\nVARIABLES\n");

printf("NOMBRE | Valor\n");

for(i=0; i < MAX\_VARIABLES; i++) {

if (strlen(variables[i].nombre)!= 0)

printf("%s | %d\n", variables[i].nombre, variables[i].valor);

}//Función auxiliar para mostrar por pantalla el valor de todas las variables.

}

int main(int argc, char \*argv[]) {

FILE \*punteroArchivo;

if (argc == 2) {

// viene una ruta por parámetro

punteroArchivo = fopen(argv[1],"r");

yyin = fopen(argv[1],"r");

yyparse();

fclose(punteroArchivo);

}

else {

yyparse();

}

} //En el main, llamaremos al yyparse, tanto para una ruta por parámetro, como por defecto.