PROGRAMA 2

“Analizador sintáctico”

INTEGRANTES:

- *Aguilar Castro Carlos Alfonso*
- *Bustamante Piza Karla Mireli*
- *Ugalde Vivo José Francisco*

Facultad de Ingeniería, UNAM

Grupo: 01

Compiladores

ANÁLISIS DEL PROBLEMA

Se debe desarrollar el análisis sintáctico de la gramática mostrada a continuación, deberá contener sus declaraciones correspondientes con Lex.

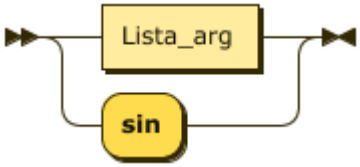
1. programa → declaraciones funciones
2. declaraciones → tipo lista_var; declaraciones
   | tipo_registro lista_var; declaraciones
   | ε
3. tipo_registro → **estructura inicio** declaraciones **fin**
4. tipo → base tipo_arreglo
5. base → **ent** | **real** | **dreal** | **car** | **sin**
6. tipo_arreglo → **(num)** tipo_arreglo | ε
7. lista_var → lista_var, **id** | **id**
8. funciones → **def** tipo **id(**argumentos**) inicio** declaraciones sentencias **fin** funciones
   | ε
9. argumentos → listar_arg | **sin**
10. lista_arg → lista_arg, arg | arg
11. arg → tipo_arg **id**
12. tipo_arg → base param_arr
13. param_arr → **( )** param_arr | ε
14. sentencias → sentencias sentencia | sentencia
15. sentencia → **si** e_bool **entonces** sentencia **fin**
    | **si** e_bool **entonces** sentencia **sino** sentencia **fin**
    | **mientras** e_bool **hacer** sentencia **fin**
    | **hacer** sentencia **mientras** e_bool;
    | **segun (**variable**) hacer** casos predeterminado **fin**
    | variable := expresion ;
    | **escribir** expresion ;
    | **leer** variable ; | **devolver;**
    | **devolver** expresion;
    | **terminar;**
    | **inicio** sentencias **fin**
16. casos → **caso num:** sentencia casos | **caso num:** sentencia
17. predeterminado → **pred:** sentencia | ε
18. e_bool → e_bool **o** e_bool | e_bool **y** e_bool | **no** e_bool | **(** e_bool **)**
    | relacional | **verdadero** | **falso**
19. relacional → relacional oprel relacional | expresion
20. oprel → > | < | >= | <= | <> | =
21. expresion → expresion oparit expresion
    | expresion **%** expresion | **(**expresion **)** | **id**
    | variable | **num** | **cadena** | **caracter** | **id(**parametros**)**
22. oparit → + | − | * | /
23. variable → dato_est_sim | arreglo
24. dato_est_sim → dato_est_sim **.id** | **id**
25. arreglo → **id (** expresion **)** | arreglo **(** expresion **)**
26. parametros → lista_param | ε
27. lista_param → lista_param, expresion | expresion

DISEÑO DE LA SOLUCIÓN

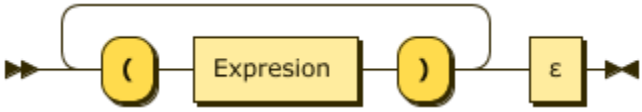
Diagramas de sintaxis:

Arg:

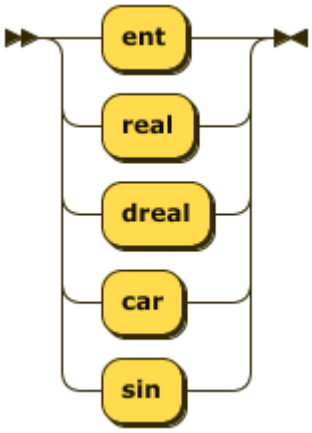
Argumentos:



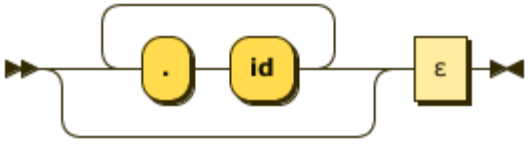
Arreglo



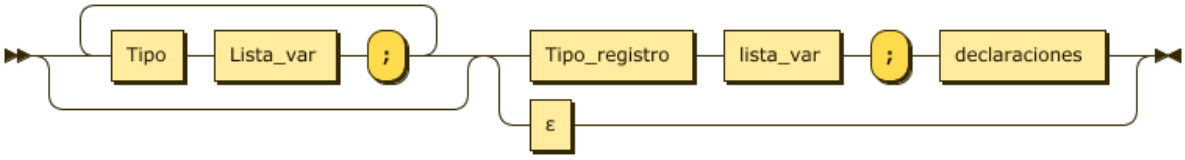
Base



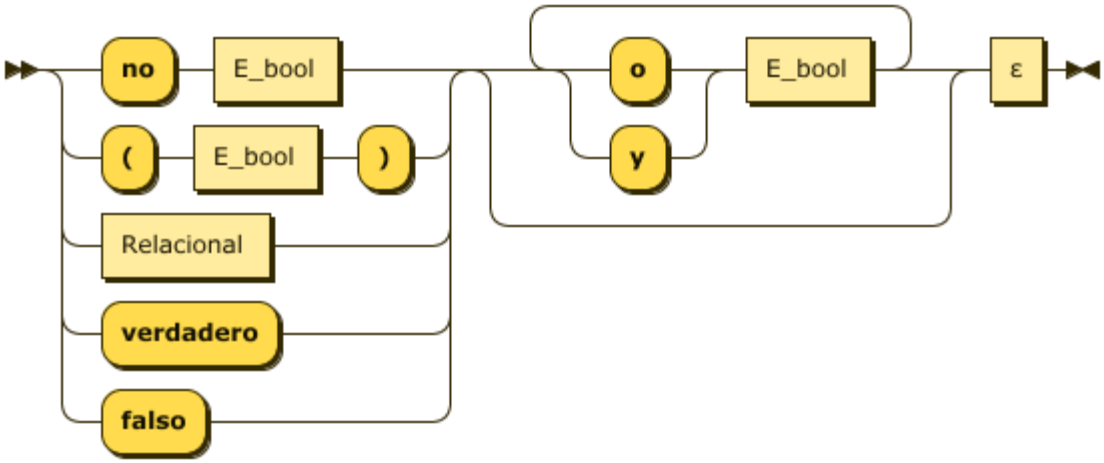
Dato_est_sim:



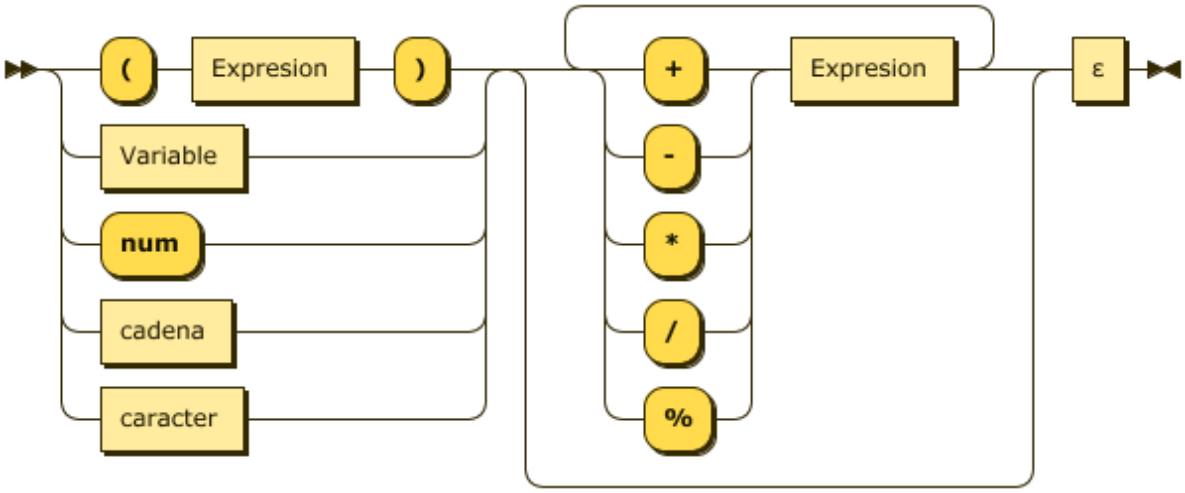
Declaraciones:



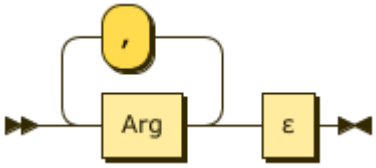
E_bool:



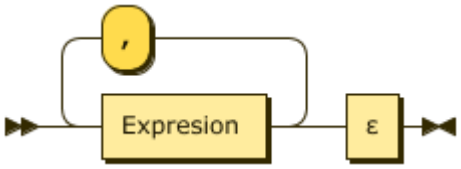
Expresión:



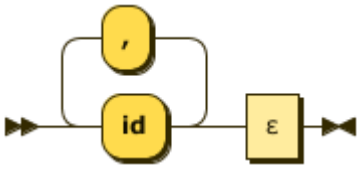
Lista_arg:



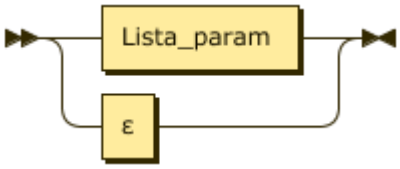
Lista_param:



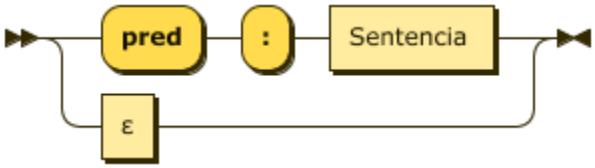
Lista_var:



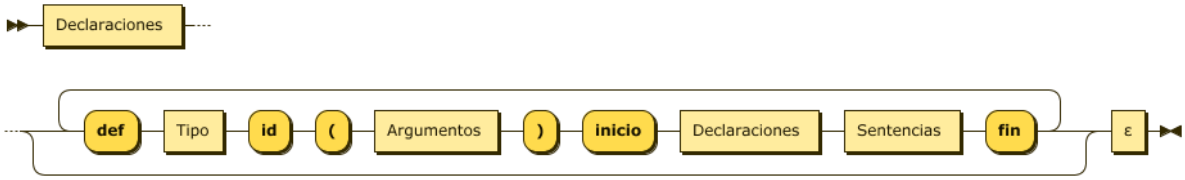
Parámetros:



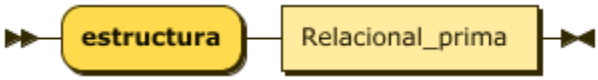
Predeterminado:



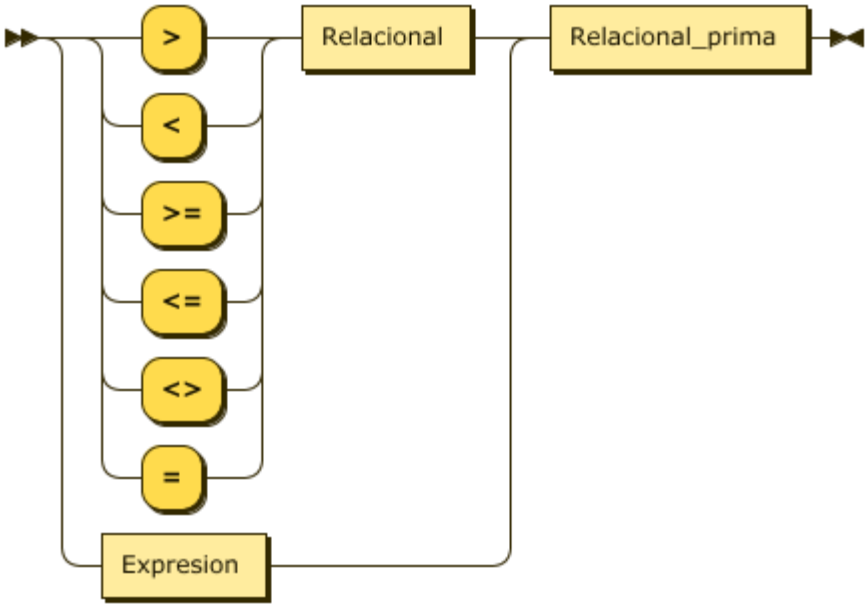
Programa:



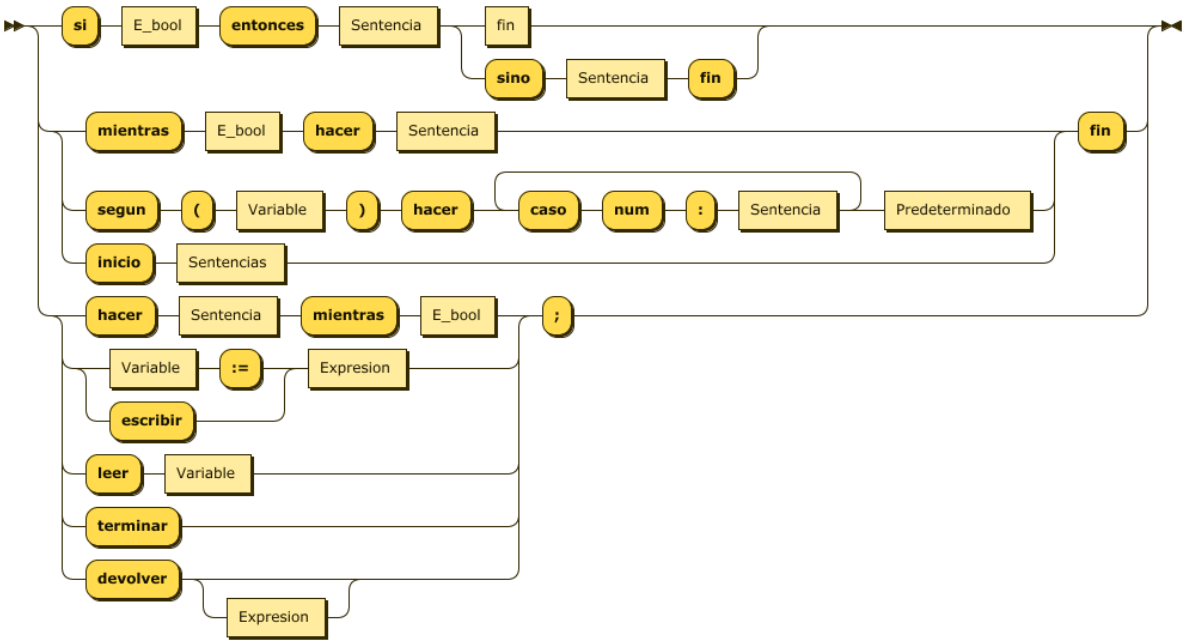
Relacional:



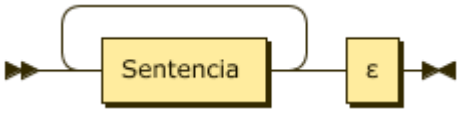
Relacional_prima:



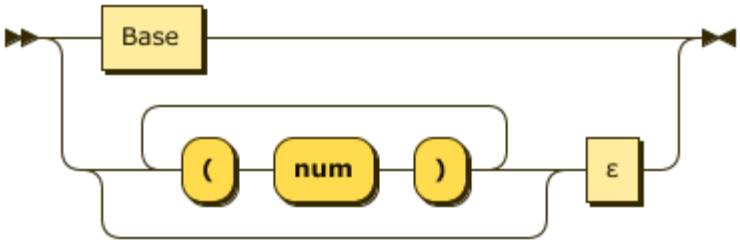
Sentencia:



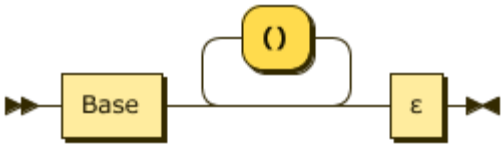
Sentencias:



Tipo:



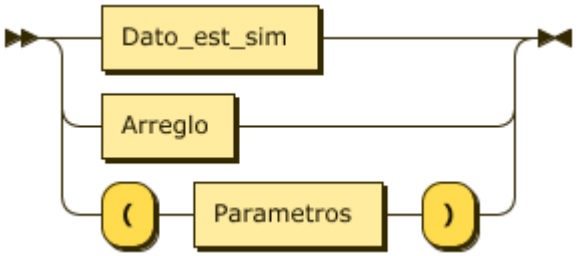
Tipo_arg:



Tipo_registro:

Variable:

Variable_comp:



Remover ambigüedad:

Para remover la ambigüedad se indicó precedencia dentro del archivo parser.y al declarar los tokens de la siguiente manera:

```
%token CASE
%token DEFAULT
%token THEN
%token PRED
%token TERM

%token<entero> NUM
%token<nreal> NREAL
%token<carac> CARAC
%token<cade> CADENA

%token FALSE TRUE

%token SCOLON COMMA COLON DCOLON

%right ASIGN ASIGN2

%left PLUS MINUS
%left MUL DIV
%left MOD

%nonassoc LPAR RPAR

%left LESS LESSEQ MORE MOREEQ NOTEQ

%left OR
%left AND
%left NOT


%start PROGRAMA
%%
```

Eliminación de recursividades izquierdas:

- lista_arg
  - O: lista_arg -> lista_arg, arg | arg
  - N: lista_arg -> arg lista_arg_prima
    lista_arg_prima -> , arg lista_arg_prima | ε
- lista_var
  - O: lista_var -> lista_var, id | id
  - N: lista_var -> id lista_var_prima
    Lista_var_prima -> ,id lista_var_prima | ε
- sentencias
  - O: sentencias -> sentencias sentencia | sentencia
  - N: sentencias -> sentencia sentencias_prima

sentencias_prima -> sentencia sentencias_prima| ε

- dato
  - O: dato_est_sim-> dato_est_sim .id | ε
  - N: dato_est_sim -> .dato_est_sim_prima | ε
    dato_est_sim_prima-> .id dato_est_sim_prima | ε
- arreglo
  - O: arreglo-> (expresión) |arreglo(expresión)
  - N: arreglo -> (expresión) arreglo_p
    arreglo_p -> (expresión) arreglo_p | ε
- lista_param
  - O: lista_param-> lista_param, expresión | expresión
  - N: lista_param -> expresión lista_param_p
  - Lista_param_p ->, expresión | ε
- e_bool
  - O: e_bool-> e_bool o e_bool | e_bool y e_bool |no e_bool | (e_bool) | relacional| T | F
  - N: e_bool -> no e_bool e_bool_p | (e_bool) e_bool_p | relacional e_bool_p | T e_bool_p | F e_bool_p
    e_bool_p -> o e_bool e_bool_p | y e_bool e_bool_p | ε
- relacional
  - O: relacional-> relacional> relacional| relacional < relacional | relacional <= relacional| relacional >= relacional | relacional <> relacional | relacional = relacional | e
  - N: relacional -> estructura relacional _p
  - relacional _p-> > relacional relacional_p | < relacional relacional_p | <= relacional relacional_p | >= relacional relacional_p | <> relacional relacional_p | = relacional relacional_p | ε

- Expresión
  - O: expresión -> expresión+expresión | expresión-expresión | expresión*expresión | expresión/expresión | expresión%expresión | (expresión) | variable | num | cadena | caracter
  - N: expresión -> (expresión) expresión_p | variable expresión_p | num expresión_p | cadena expresión_p | carácter expresión_p
    expresión_p-> + expresión expresión_p | - expresión expresión_p | * expresión expresión_p | /expresión expresión_p | %expresión expresión_p | ε

IMPLEMENTACIÓN

El programa se compone de 3 archivos: parser.y alexv3_sin.l y main.c

Dentro de parser.y observamos 3 secciones:

Declaraciones: se declara los tokens terminales, así como su precedencia con la cual eliminamos ambigüedad.

Esquema de traducción: Se definen las reglas de la gramática.

Código de usuario: se declaran las funciones con código C

Dentro de alexv3_sin.l observamos de igual manera 3 secciones que se describen de la siguiente manera:

Definiciones: Contiene declaraciones de definiciones de nombres sencillas para simplificar la especificación del escáner, y declaraciones de condiciones de arranque.

Reglas: Acciones que se tomaran al encontrar ciertas cadenas o símbolos ya sea que se hayan definido con ERs en el apartado de definiciones o se indiquen entre comillas en esta sección.

Código de usuario: se declaran las funciones con código C

Finalmente, en main.c solo guardamos la función main para el programa donde se declara la lectura del archivo de entrada.

FORMA DE EJECUTAR

1. Se deberá ejecutar el siguiente comando:

   # bison –d -y parser.y

   Este generara un .c y un .h, el .h debemos agregarlos como encabezado en nuestro archivo alexv3_sin.l.

2. A continuación, ejecutamos los siguientes comandos:

   # lex alexv3_sin.l

3. Finalmente, insertamos el siguiente comando para generar el ejecutable:

   #gcc lex.yy.c y.tab.c main.c –o prog2

4. Para probar el ejecutable podemos hacerlo de la siguiente manera:

   #./prog2 entrada.txt