Campus Irapuato-Salamanca | División de Ingenierías

Universidad de Guanajuato

División de Ingenierías Campus Irapuato Salamanca

Proyecto 2: Analizador sintáctico

UDA: Compiladores

Impartido por: Dr. José Ruiz Pinales.

Integrantes:

José Luis Arroyo Núñez.	NUA: 390893.
Bryan Ricardo Cervantes Mancera	NUA: 146809.

Objetivo:

El objetivo de realizar este analizador sintáctico, el cual, es el paso siguiente paso a desarrollar para el compilador, con ayuda del proyecto 1, el analizador léxico, se desarrollará el analizador para el lenguaje de programación C, por lo que es necesario tener en cuenta las reglas gramaticales que este lenguaje contempla en su estructura.

Introducción:

En esta práctica por medio de modificaciones al archivo del analizador léxico “ansic.l” se pudo implementar un analizador sintáctico en el archivo “ansic.y”.

La funcionalidad de este analizador es la de recorrer la sintaxis de cualquier código en Lenguaje C, estructurando las instrucciones del código mapeando los comandos del lenguaje y desarrollando un árbol sintáctico con salida en la consola, este solamente se pone en esta parte para fines demostrativos, del funcionamiento del analizador.

Desarrollo:

El analizador se encuentra estrechamente apoyado por el analizador léxico, debido a que este analizador nos da como resultado los tokens que se encontraron en el código fuente, nosotros usaremos estos tokens, para la integración final de la gramática que necesitamos implementar.

Uno de los puntos clave que se tienen que tener en cuenta para la realización de este analizado son los nombres y tipos de tokens que se están encontrando con el analizador léxico.

Código implementado:

Código del archivo ansic.l:

```
%{
#include <stdio.h>
#include <ctype.h>
#include "ansic.tab.h"

//Se aggrean las constantes del switch
enum ATTRTYPES
{
   CHARVAL,
   INTVAL,
   DOUBLEVAL
};

void count();
void comment();
void yyerror(char *);

YYSTYPE yylval;
int lineno = 0;
/*Sirve para contar la cantidad de columnas.*/
int column = 0; /* Contiene el numero de la columna donde encontro el token. */

/*        Count: cuenta todos los carcateres que vaya encpontrnado. */
/* atol: comvierte de texto a numero */
/* strtol: convierte de cualquier base a decimal */
/* atof: convierte de texto a float */

//Guardar la cadena que resulta de la conversion
char *buffer = NULL;
int buffer_size = 0;

/*#define isdigit(x) ((x) >= '0' && (x) <= '7')*/
/*Convertir un carcater a su equivalente en decimal*/
#define hextoint(x) (isdigit((x)) ? (x) - '0' : ((x) - 'A') + 10)

/* Usamos count en todas las reglas para poder localizar con mas precicion un error.*/
%}

D                       [0-9]
L                       [a-zA-Z_]
H                       [a-fA-F0-9]
E                       [Ee][+-]?{D}+
FS                      (f|F|l|L)
IS                      (u|U|l|L)*
hex       [0-9a-fA-F]{1,2}
oct       [0-7]{1,3}
%x INSTRING INCHAR
%option noyywrap

%%

"/*"                    { comment(); /* bUSCA EL FIN DE COMENTARIO. */}
^#.*            {   /*IGNORA EL AVANZE DE LINEA*/   }
"auto"                  { count(); return(AUTO); }
"break"                 { count(); return(BREAK); }
"case"                  { count(); return(CASE); }
"char"                  { count(); return(CHAR); }
"const"                 { count(); return(CONST); }
"continue"              { count(); return(CONTINUE); }
"default"               { count(); return(DEFAULT); }
```

```
"do"                    { count(); return(DO); }
"double"                { count(); return(DOUBLE); }
"else"                  { count(); return(ELSE); }
"enum"                  { count(); return(ENUM); }
"extern"                { count(); return(EXTERN); }
"float"                 { count(); return(FLOAT); }
"for"                   { count(); return(FOR); }
"goto"                  { count(); return(GOTO); }
"if"                    { count(); return(IF); }
"int"                   { count(); return(INT); }
"long"                  { count(); return(LONG); }
"register"              { count(); return(REGISTER); }
"return"                { count(); return(RETURN); }
"short"                 { count(); return(SHORT); }
"signed"                { count(); return(SIGNED); }
"sizeof"            { count(); return(SIZEOF); }
"static"            { count(); return(STATIC); }
"struct"            { count(); return(STRUCT); }
"switch"                { count(); return(SWITCH); }
"typedef"               { count(); return(TYPEDEF); }
"union"                 { count(); return(UNION); }
"unsigned"              { count(); return(UNSIGNED); }
"void"                  { count(); return(VOID); }
"volatile"              { count(); return(VOLATILE); }
"while"                 { count(); return(WHILE); }

{L}({L}|{D})*           { count(); yylval.name = strdup(yytext); return(IDENTIFIER); /*Contiene el
lexema que fue encontrado*/}

0[xX]{H}+{IS}?                  { count(); yylval.ival = strtol(yytext, NULL, 16); yylval.type =
INTVAL; return(CONSTANT);}   /*Formato hexadecimal*/
0{D}+{IS}?              { count(); yylval.ival = strtol(yytext, NULL, 8);  yylval.type = INTVAL;
return(CONSTANT);}   /*Constante octal*/
{D}+{IS}?               { count(); yylval.ival = atol(yytext); yylval.type = INTVAL;
return(CONSTANT);}          /*Constante entera decimal.*/
L?'                 {   /* se busca el inicio de la constante char. */
                    /*Solo es para un carcater.*/
                    count();
                    buffer = malloc(1);
                    buffer_size = 1;
                    buffer[0] = 0; /*Se limpia el espacio donde esta el carcater*/
                    BEGIN(INCHAR); /*Se inicia que todos los carcateres seguidos del apostrofe se
capturen*/
                }
L?\"                {
                    count();
                    buffer = malloc(1);
                    buffer_size = 1;
                    strcpy(buffer, "");
                    BEGIN(INSTRING); /*Se inicia que todos los carcateres seguidos del apostrofe se
capturen*/
                    printf("Start of  the string\n");
                }
<INCHAR,INSTRING>\n    {
                    count();
                    yyerror("Undeterminated characters of string literal");
                    free(buffer); /* Se libera el buffer */
                    BEGIN(INITIAL); /* Se regresa al estado inicial */
                }
<INCHAR,INSTRING><<EOF>>   {
                       count();
                       yyerror("EOF in string literal"); /*Fin de archivo en cadena */
                       free(buffer); /* Se libera el buffer */
                       BEGIN(INITIAL); /* Se regresa al estado inicial */
                    }
<INCHAR,INSTRING>[^\|\|\n"'] {
```

```
                        /*Se busca cualquier carcater alfanumerico, se exculen apostorfes, comillas,
saltos de linea, carcateres doble o triples.*/
                        count();
                        buffer = realloc(buffer, buffer_size + yyleng + 1);
                        buffer_size += yyleng; /*Se incrementa el tamaño del buffer*/
                        strcat(buffer, yytext);
                        if(YY_START == INCHAR && buffer_size > 2) /*Si se agrega un carcater
demas*/
                            yyerror("Caracter o literal ilegal.");
                    }
<INSTRING>\\\n          /* ingnore this */
<INCHAR,INSTRING>\\{hex}    {
                        count();
                        int temp = 0, loop = 0;
                        for(loop=yyleng-2; loop>0; loop--) /*procesaro digitio a digito*/
                        {
                            temp <<= 4; /* Recorrimiento de 3 bits */
                            temp += hextoint(toupper(yytext[yyleng-loop]));
                        }
                        buffer = realloc(buffer, buffer_size+1);
                        buffer[buffer_size-1] = temp;
                        buffer[buffer_size] = '\0';
                        buffer_size += 1;
                        if(YY_START == INCHAR && buffer_size > 2) /*Si se agrega un carcater
demas*/
                            yyerror("Caracter o literal ilegal.");
                    }
<INCHAR,INSTRING>\\{oct}    {
                        count();
                        int temp = 0, loop = 0;
                        for(loop=yyleng-2; loop>0; loop--) /*procesaro digitio a digito*/
                        {
                            temp <<= 4; /* Recorrimiento de 3 bits */
                            temp += (yytext[yyleng-loop] - '0');
                        }
                        buffer = realloc(buffer, buffer_size+1);
                        buffer[buffer_size-1] = temp;
                        buffer[buffer_size] = '\0';
                        buffer_size += 1;
                        if(YY_START == INCHAR && buffer_size > 2) /*Si se agrega un carcater
demas*/
                            yyerror("Caracter o literal ilegal.");
                    }
<INCHAR,INSTRING>\\[^\n]    {
                        count();
                        buffer = realloc(buffer, buffer_size+1); /*Incrementa el tamaño del buffer*/
                        switch(yytext[yyleng-1])
                        {
                            case 'b' : buffer[buffer_size-1] = '\b'; break;
                            case 't' : buffer[buffer_size-1] = '\t'; break;
                            case 'n' : buffer[buffer_size-1] = '\n'; break;
                            case 'v' : buffer[buffer_size-1] = '\v'; break;
                            case 'f' : buffer[buffer_size-1] = '\f'; break;
                            case 'r' : buffer[buffer_size-1] = '\r'; break;
                            default  : buffer[buffer_size-1] = yytext[yyleng-1];
                        }
                        buffer[buffer_size] = '\0';
                        buffer_size += 1;
                        if(YY_START == INCHAR && buffer_size > 2)
                            yyerror("Ilegal lenght of characters constants");
                    }
<INCHAR,INSTRING>'      {
                    count();
                    if(YY_START == INCHAR)
                    {
                        yylval.cval = buffer[0];
```

```
                        if(buffer_size > 2)
                        {
                            yyerror("llegal lenght of characters constants");
                        }
                        yylval.type = CHARVAL;
                        free(buffer);
                        BEGIN(INITIAL);
                        return(CONSTANT);
                    }
                    buffer = realloc(buffer, buffer_size + yyleng + 1);
                    buffer_size += yyleng;
                    strcat(buffer, yytext);
                }
<INSTRING,INCHAR>\"    {
                    count();
                    if(YY_START == INSTRING)
                    {
                        yylval.str = buffer;
                        /*free(buffer);*/
                        BEGIN(INITIAL);
                        return(STRING_LITERAL);
                    }
                    buffer = realloc(buffer, buffer_size + yyleng + 1);
                    buffer_size += yyleng;
                    strcat(buffer, yytext);
                    if(buffer_size > 2)
                        yyerror("llegal lenght of characters constants");
                    printf("End of the string\n");
                }

{D}+{E}{FS}?             { count(); yylval.dval = atof(yytext); yylval.type = DOUBLEVAL;
return(CONSTANT);    /* constante floar o double. */  }
{D}*"."{D}+({E})?{FS}?        { count(); yylval.dval = atof(yytext); yylval.type = DOUBLEVAL;
return(CONSTANT);    /* constante floar o double. */}
{D}+"."{D}*({E})?{FS}?        { count(); yylval.dval = atof(yytext); yylval.type = DOUBLEVAL;
return(CONSTANT); }

"..."                   { count(); return(ELLIPSIS); }
">>="                   { count(); return(RIGHT_ASSIGN); }
"<<="                   { count(); return(LEFT_ASSIGN); }
"+="                    { count(); return(ADD_ASSIGN); }
"-="                    { count(); return(SUB_ASSIGN); }
"*="                    { count(); return(MUL_ASSIGN); }
"/="                    { count(); return(DIV_ASSIGN); }
"%="                    { count(); return(MOD_ASSIGN); }
"&="                    { count(); return(AND_ASSIGN); }
"^="                    { count(); return(XOR_ASSIGN); }
"|="                    { count(); return(OR_ASSIGN); }
">>"                    { count(); return(RIGHT_OP); }
"<<"                    { count(); return(LEFT_OP); }
"++"                    { count(); return(INC_OP); }
"--"                    { count(); return(DEC_OP); }
"->"                    { count(); return(PTR_OP); }
"&&"                    { count(); return(AND_OP); }
"||"                    { count(); return(OR_OP); }
"<="                    { count(); return(LE_OP); }
">="                    { count(); return(GE_OP); }
"=="                    { count(); return(EQ_OP); }
"!="                    { count(); return(NE_OP); }
";"                     { count(); return(';'); }
("{"|"<%")              { count(); return('{'); }
("}"|"%>")              { count(); return('}'); }
","                     { count(); return(','); }
":"                     { count(); return(':'); }
"="                     { count(); return('='); }
"("                     { count(); return('('); }
```

```
")"                         { count(); return(')'); }
("["|"<:")                  { count(); return('['); }
("]"|":>")                  { count(); return(']'); }
"."                         { count(); return('.'); }
"&"                         { count(); return('&'); }
"!"                         { count(); return('!'); }
"~"                         { count(); return('~'); }
"-"                         { count(); return('-'); }
"+"                         { count(); return('+'); }
"*"                         { count(); return('*'); }
"/"                         { count(); return('/'); }
"%"                         { count(); return('%'); }
"<"                         { count(); return('<'); }
">"                         { count(); return('>'); }
"^"                         { count(); return('^'); }
"|"                         { count(); return('|'); }
"?"                         { count(); return('?'); }

[ \t\v\n\f]                 {
count();
if(yytext[0]=='\n')
lineno++;
/* Cuando se encuentre uno de esos carcateres se checa cual es */  }

.                           { /* ignore bad characters */ }

%%

void yyerror(char *msg)
{
   printf("\n\t Errror lexico: %s en linea: %d, columna: %d\n", msg, lineno+1, column+1);
   exit(1);
}

/* Busca el fin de comentario y no escirbe nada en consola */
void comment()
{
char c, c1;

loop:
while ((c = input()) != '*' && c != 0)
;

if ((c1 = input()) != '/' && c != 0)
{
unput(c1);
goto loop;
}
}

void count()
{
int i;

for (i = 0; yytext[i] != '\0'; i++)
if (yytext[i] == '\n')
column = 0;
else if (yytext[i] == '\t')
column += 8 - (column % 8);
else
column++;

 /* ECHO; */ /*Equivale a un pronft*/
}
```

Código del archivo ansic.y:

```
%{
#include <stdio.h>
#include <stdlib.h>
extern int yylex();//ya que es una funcion en archivo externo y esta es usada en el analizador
sintactico
extern void yyerror(char *);
extern FILE *yyin;

%}
%union{
struct{
char cval;
long int ival;
double dval;
char *str;
char *name;
int type;
};
}

%token IDENTIFIER CONSTANT STRING_LITERAL SIZEOF
%token PTR_OP INC_OP DEC_OP LEFT_OP RIGHT_OP LE_OP GE_OP EQ_OP NE_OP
%token AND_OP OR_OP MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN ADD_ASSIGN
%token SUB_ASSIGN LEFT_ASSIGN RIGHT_ASSIGN AND_ASSIGN
%token XOR_ASSIGN OR_ASSIGN TYPE_NAME

%token TYPEDEF EXTERN STATIC AUTO REGISTER
%token CHAR SHORT INT LONG SIGNED UNSIGNED FLOAT DOUBLE CONST VOLATILE VOID
%token STRUCT UNION ENUM ELLIPSIS

%token CASE DEFAULT IF ELSE SWITCH WHILE DO FOR GOTO CONTINUE BREAK RETURN

//Declaracion de las prioridades para el if else
%nonassoc NO_ELSE
%nonassoc ELSE

%start translation_unit
%%

primary_expression
: IDENTIFIER {printf("primary_expression: IDENTIFIER \n");}
| CONSTANT {printf("primary_expression: CONSTANT \n");}
| STRING_LITERAL {printf("primary_expression: STRING_LITERAL \n");}
| '(' expression ')' {printf("primary_expression: '(' expression ')' \n");}
;

postfix_expression
: primary_expression {printf("postfix_expression: primary_expression \n");}
| postfix_expression '[' expression ']' {printf("postfix_expression: postfix_expression '[' expression
']' \n");}
| postfix_expression '(' ')' {printf("postfix_expression: postfix_expression '(' ')' \n");}
| postfix_expression '(' argument_expression_list ')' {printf("postfix_expression: postfix_expression
'(' argument_expression_list ')' \n");}
| postfix_expression '.' IDENTIFIER {printf("postfix_expression: postfix_expression '.' IDENTIFIER \
n");}
| postfix_expression PTR_OP IDENTIFIER {printf("postfix_expression: postfix_expression PTR_OP
IDENTIFIER \n");}
| postfix_expression INC_OP {printf("postfix_expression: postfix_expression INC_OP \n");}
| postfix_expression DEC_OP {printf("postfix_expression: postfix_expression DEC_OP \n");}
;

argument_expression_list
: assignment_expression {printf("argument_expression_list: assignment_expression \n");}
```

```
| argument_expression_list ',' assignment_expression {printf("argument_expression_list:
argument_expression_list ',' assignment_expression \n");}
;

unary_expression
: postfix_expression {printf("unary_expression: postfix_expression \n");}
| INC_OP unary_expression {printf("unary_expression: INC_OP unary_expression \n");}
| DEC_OP unary_expression {printf("unary_expression: DEC_OP unary_expression \n");}
| unary_operator cast_expression {printf("unary_expression: unary_operator cast_expression \n");}
| SIZEOF unary_expression {printf("unary_expression: SIZEOF unary_expression \n");}
| SIZEOF '(' type_name ')' {printf("unary_expression: SIZEOF '(' type_name ')' \n");}
;

unary_operator
: '&' {printf("unary_operator: '&' \n");}
| '*' {printf("unary_operator: '*' \n");}
| '+' {printf("unary_operator: '+' \n");}
| '-' {printf("unary_operator: '-' \n");}
| '~' {printf("unary_operator: '~' \n");}
| '!' {printf("unary_operator: '!' \n");}
;

cast_expression
: unary_expression {printf("cast_expression: unary_expression \n");}
| '(' type_name ')' cast_expression {printf("cast_expression: '(' type_name ')' cast_expression \n");}
;

multiplicative_expression
: cast_expression {printf("multiplicative_expression: cast_expression \n");}
| multiplicative_expression '*' cast_expression {printf("multiplicative_expression:
multiplicative_expression '*' cast_expression \n");}
| multiplicative_expression '/' cast_expression {printf("multiplicative_expression:
multiplicative_expression '/' cast_expression \n");}
| multiplicative_expression '%' cast_expression {printf("multiplicative_expression:
multiplicative_expression '%' cast_expression \n");}
;

additive_expression
: multiplicative_expression {printf("additive_expression: multiplicative_expression \n");}
| additive_expression '+' multiplicative_expression {printf("additive_expression:
additive_expression '+' multiplicative_expression \n");}
| additive_expression '-' multiplicative_expression {printf("additive_expression: additive_expression
'-' multiplicative_expression \n");}
;

shift_expression
: additive_expression {printf("shift_expression: additive_expression \n");}
| shift_expression LEFT_OP additive_expression {printf("shift_expression: shift_expression LEFT_OP
additive_expression \n");}
| shift_expression RIGHT_OP additive_expression {printf("shift_expression: shift_expression
RIGHT_OP additive_expression \n");}
;

relational_expression
: shift_expression {printf("relational_expression: shift_expression \n");}
| relational_expression '<' shift_expression {printf("relational_expression: relational_expression '<'
shift_expression \n");}
| relational_expression '>' shift_expression {printf("relational_expression: relational_expression '>'
shift_expression \n");}
| relational_expression LE_OP shift_expression {printf("relational_expression: relational_expression
LE_OP shift_expression \n");}
| relational_expression GE_OP shift_expression {printf("relational_expression: relational_expression
GE_OP shift_expression \n");}
;

equality_expression
```

```
: relational_expression {printf("equality_expression: relational_expression \n");}
| equality_expression EQ_OP relational_expression {printf("equality_expression:
equality_expression EQ_OP relational_expression \n");}
| equality_expression NE_OP relational_expression {printf("equality_expression:
equality_expression NE_OP relational_expression \n");}
;

and_expression
: equality_expression {printf("and_expression: equality_expression \n");}
| and_expression '&' equality_expression {printf("and_expression: and_expression '&'
equality_expression \n");}
;

exclusive_or_expression
: and_expression {printf("exclusive_or_expression: and_expression \n");}
| exclusive_or_expression '^' and_expression {printf("exclusive_or_expression:
exclusive_or_expression '^' and_expression \n");}
;

inclusive_or_expression
: exclusive_or_expression {printf("inclusive_or_expression: exclusive_or_expression \n");}
| inclusive_or_expression '|' exclusive_or_expression {printf("inclusive_or_expression:
inclusive_or_expression '|' exclusive_or_expression \n");}
;

logical_and_expression
: inclusive_or_expression {printf("logical_and_expression: inclusive_or_expression \n");}
| logical_and_expression AND_OP inclusive_or_expression {printf("logical_and_expression:
logical_and_expression AND_OP inclusive_or_expression \n");}
;

logical_or_expression
: logical_and_expression {printf("logical_or_expression: logical_and_expression \n");}
| logical_or_expression OR_OP logical_and_expression {printf("logical_or_expression:
logical_or_expression OR_OP logical_and_expression \n");}
;

conditional_expression
: logical_or_expression {printf("conditional_expression: logical_or_expression \n");}
| logical_or_expression '?' expression ':' conditional_expression {printf("conditional_expression:
logical_or_expression '?' expression ':' conditional_expression \n");}
;

assignment_expression
: conditional_expression {printf("assignment_expression: conditional_expression \n");}
| unary_expression assignment_operator assignment_expression {printf("assignment_expression:
unary_expression assignment_operator assignment_expression \n");}
;

assignment_operator
: '=' {printf("assignment_operator: '=' \n");}
| MUL_ASSIGN {printf("assignment_operator: MUL_ASSIGN \n");}
| DIV_ASSIGN {printf("assignment_operator: DIV_ASSIGN \n");}
| MOD_ASSIGN {printf("assignment_operator: MOD_ASSIGN \n");}
| ADD_ASSIGN {printf("assignment_operator: ADD_ASSIGN \n");}
| SUB_ASSIGN {printf("assignment_operator: SUB_ASSIGN \n");}
| LEFT_ASSIGN {printf("assignment_operator: LEFT_ASSIGN \n");}
| RIGHT_ASSIGN {printf("assignment_operator: RIGHT_ASSIGN \n");}
| AND_ASSIGN {printf("assignment_operator: AND_ASSIGN \n");}
| XOR_ASSIGN {printf("assignment_operator: XOR_ASSIGN \n");}
| OR_ASSIGN {printf("assignment_operator: OR_ASSIGN \n");}
;

expression
: assignment_expression {printf("expression: assignment_expression \n");}
```

```
| expression ',' assignment_expression {printf("expression: expression ',' assignment_expression \
n");}
;

constant_expression
: conditional_expression {printf("constant_expression: conditional_expression \n");}
;

declaration
: declaration_specifiers ';' {printf("declaration: declaration_specifiers ';' \n");}
| declaration_specifiers init_declarator_list ';' {printf("declaration: declaration_specifiers
init_declarator_list ';' \n");}
;

declaration_specifiers
: storage_class_specifier {printf("declaration_specifiers: storage_class_specifier \n");}
| storage_class_specifier declaration_specifiers {printf("declaration_specifiers:
storage_class_specifier declaration_specifiers \n");}
| type_specifier {printf("declaration_specifiers: type_specifier \n");}
| type_specifier declaration_specifiers {printf("declaration_specifiers: type_specifier
declaration_specifiers \n");}
| type_qualifier {printf("declaration_specifiers: type_qualifier \n");}
| type_qualifier declaration_specifiers {printf("declaration_specifiers: type_qualifier
declaration_specifiers \n");}
;

init_declarator_list
: init_declarator {printf("init_declarator_list: init_declarator \n");}
| init_declarator_list ',' init_declarator {printf("init_declarator_list: init_declarator_list ','
init_declarator \n");}
;

init_declarator
: declarator {printf("init_declarator: declarator \n");}
| declarator '=' initializer {printf("init_declarator: declarator '=' initializer \n");}
;

storage_class_specifier
: TYPEDEF {printf("storage_class_specifier: TYPEDEF \n");}
| EXTERN {printf("storage_class_specifier: EXTERN \n");}
| STATIC {printf("storage_class_specifier: STATIC \n");}
| AUTO {printf("storage_class_specifier: AUTO \n");}
| REGISTER {printf("storage_class_specifier: REGISTER \n");}
;

type_specifier
: VOID {printf("type_specifier: VOID \n");}
| CHAR {printf("type_specifier: CHAR \n");}
| SHORT {printf("type_specifier: SHORT \n");}
| INT {printf("type_specifier: INT \n");}
| LONG {printf("type_specifier: LONG \n");}
| FLOAT {printf("type_specifier: FLOAT \n");}
| DOUBLE {printf("type_specifier: DOUBLE \n");}
| SIGNED {printf("type_specifier: SIGNED \n");}
| UNSIGNED {printf("type_specifier: UNSIGNED \n");}
| struct_or_union_specifier {printf("type_specifier: struct_or_union_specifier \n");}
| enum_specifier {printf("type_specifier: enum_specifier \n");}
| TYPE_NAME {printf("type_specifier: TYPE_NAME \n");}
;

struct_or_union_specifier
: struct_or_union IDENTIFIER '{' struct_declaration_list '}' {printf("struct_or_union_specifier:
struct_or_union IDENTIFIER '{' struct_declaration_list '}' \n");}
| struct_or_union '{' struct_declaration_list '}' {printf("struct_or_union_specifier: struct_or_union '{'
struct_declaration_list '}' \n");}
| struct_or_union IDENTIFIER {printf("struct_or_union_specifier: struct_or_union IDENTIFIER \n");}
```

```
;

struct_or_union
: STRUCT {printf("struct_or_union: STRUCT \n");}
| UNION {printf("struct_or_union: UNION \n");}
;

struct_declaration_list
: struct_declaration {printf("struct_declaration_list: struct_declaration \n");}
| struct_declaration_list struct_declaration {printf("struct_declaration_list: struct_declaration_list
struct_declaration \n");}
;

struct_declaration
: specifier_qualifier_list struct_declarator_list ';' {printf("struct_declaration: specifier_qualifier_list
struct_declarator_list ';' \n");}
;

specifier_qualifier_list
: type_specifier specifier_qualifier_list {printf("specifier_qualifier_list: type_specifier
specifier_qualifier_list \n");}
| type_specifier {printf("specifier_qualifier_list: type_specifier \n");}
| type_qualifier specifier_qualifier_list {printf("specifier_qualifier_list: type_qualifier
specifier_qualifier_list \n");}
| type_qualifier {printf("specifier_qualifier_list: type_qualifier \n");}
;

struct_declarator_list
: struct_declarator {printf("struct_declarator_list: struct_declarator \n");}
| struct_declarator_list ',' struct_declarator {printf("struct_declarator_list: struct_declarator_list ','
struct_declarator \n");}
;

struct_declarator
: declarator {printf("struct_declarator: declarator \n");}
| ':' constant_expression {printf("struct_declarator: ':' constant_expression \n");}
| declarator ':' constant_expression {printf("struct_declarator: declarator ':' constant_expression \
n");}
;

enum_specifier
: ENUM '{' enumerator_list '}' {printf("enum_specifier: ENUM '{' enumerator_list '}' \n");}
| ENUM IDENTIFIER '{' enumerator_list '}' {printf("enum_specifier: ENUM IDENTIFIER '{'
enumerator_list '}' \n");}
| ENUM IDENTIFIER {printf("enum_specifier: ENUM IDENTIFIER \n");}
;

enumerator_list
: enumerator {printf("enumerator_list: enumerator \n");}
| enumerator_list ',' enumerator {printf("enumerator_list: enumerator_list ',' enumerator \n");}
;

enumerator
: IDENTIFIER {printf("enumerator: IDENTIFIER \n");}
| IDENTIFIER '=' constant_expression {printf("enumerator: IDENTIFIER '=' constant_expression \
n");}
;

type_qualifier
: CONST {printf("type_qualifier: CONST \n");}
| VOLATILE {printf("type_qualifier: VOLATILE \n");}
;

declarator
: pointer direct_declarator {printf("declarator: pointer direct_declarator \n");}
| direct_declarator {printf("declarator: direct_declarator \n");}
```

```
;

direct_declarator
: IDENTIFIER {printf("direct_declarator: IDENTIFIER \n");}
| '(' declarator ')' {printf("direct_declarator: '(' declarator ')' \n");}
| direct_declarator '[' constant_expression ']' {printf("direct_declarator: direct_declarator '['
constant_expression ']' \n");}
| direct_declarator '[' ']' {printf("direct_declarator: direct_declarator '[' ']' \n");}
| direct_declarator '(' parameter_type_list ')' {printf("direct_declarator: direct_declarator '('
parameter_type_list ')' \n");}
| direct_declarator '(' identifier_list ')' {printf("direct_declarator: direct_declarator '(' identifier_list ')'
\n");}
| direct_declarator '(' ')' {printf("direct_declarator: direct_declarator '(' ')' \n");}
;

pointer
: '*' {printf("pointer: '*' \n");}
| '*' type_qualifier_list {printf("pointer: '*' type_qualifier_list \n");}
| '*' pointer {printf("pointer: '*' pointer \n");}
| '*' type_qualifier_list pointer {printf("pointer: '*' type_qualifier_list pointer \n");}
;

type_qualifier_list
: type_qualifier {printf("type_qualifier_list: type_qualifier \n");}
| type_qualifier_list type_qualifier {printf("type_qualifier_list: type_qualifier_list type_qualifier \n");}
;

parameter_type_list
: parameter_list {printf("parameter_type_list: parameter_list \n");}
| parameter_list ',' ELLIPSIS {printf("parameter_type_list: parameter_list ',' ELLIPSIS \n");}
;

parameter_list
: parameter_declaration {printf("parameter_list: parameter_declaration \n");}
| parameter_list ',' parameter_declaration {printf("parameter_list: parameter_list ','
parameter_declaration \n");}
;

parameter_declaration
: declaration_specifiers declarator {printf("parameter_declaration: declaration_specifiers declarator
\n");}
| declaration_specifiers abstract_declarator {printf("parameter_declaration: declaration_specifiers
abstract_declarator \n");}
| declaration_specifiers {printf("parameter_declaration: declaration_specifiers \n");}
;

identifier_list
: IDENTIFIER {printf("identifier_list: IDENTIFIER \n");}
| identifier_list ',' IDENTIFIER {printf("identifier_list: identifier_list ',' IDENTIFIER \n");}
;

type_name
: specifier_qualifier_list {printf("type_name: pecifier_qualifier_list \n");}
| specifier_qualifier_list abstract_declarator {printf("type_name: specifier_qualifier_list
abstract_declarator \n");}
;

abstract_declarator
: pointer {printf("abstract_declarator: pointer \n");}
| direct_abstract_declarator {printf("abstract_declarator: direct_abstract_declarator \n");}
| pointer direct_abstract_declarator {printf("abstract_declarator: pointer
direct_abstract_declarator \n");}
;

direct_abstract_declarator
: '(' abstract_declarator ')' {printf("direct_abstract_declarator: '(' abstract_declarator ')' \n");}
```

```
| '[' ']' {printf("direct_abstract_declarator: '[' ']' \n");}
| '[' constant_expression ']' {printf("direct_abstract_declarator: '[' constant_expression ']' \n");}
| direct_abstract_declarator '[' ']' {printf("direct_abstract_declarator: direct_abstract_declarator '['
']' \n");}
| direct_abstract_declarator '[' constant_expression ']' {printf("direct_abstract_declarator:
direct_abstract_declarator '[' constant_expression ']' \n");}
| '(' ')' {printf("direct_abstract_declarator: '(' ')' \n");}
| '(' parameter_type_list ')' {printf("direct_abstract_declarator: '(' parameter_type_list ')' \n");}
| direct_abstract_declarator '(' ')' {printf("direct_abstract_declarator: direct_abstract_declarator '('
')' \n");}
| direct_abstract_declarator '(' parameter_type_list ')' {printf("direct_abstract_declarator:
direct_abstract_declarator '(' parameter_type_list ')' \n");}
;

initializer
: assignment_expression {printf("initializer: assignment_expression \n");}
| '{' initializer_list '}' {printf("initializer: '{' initializer_list '}' \n");}
| '{' initializer_list ',' '}' {printf("initializer: '{' initializer_list ',' '}' \n");}
;

initializer_list
: initializer {printf("initializer_list: initializer \n");}
| initializer_list ',' initializer {printf("initializer_list: initializer_list ',' initializer \n");}
;

statement
: labeled_statement {printf("statement: labeled_statement \n");}
| compound_statement {printf("statement: compound_statement \n");}
| expression_statement {printf("statement: expression_statement \n");}
| selection_statement {printf("statement: selection_statement \n");}
| iteration_statement {printf("statement: iteration_statement \n");}
| jump_statement {printf("statement: jump_statement \n");}
;

labeled_statement
: IDENTIFIER ':' statement {printf("labeled_statement: IDENTIFIER ':' statement \n");}
| CASE constant_expression ':' statement {printf("labeled_statement: CASE constant_expression ':'
statement \n");}
| DEFAULT ':' statement {printf("labeled_statement: DEFAULT ':' statement \n");}
;

compound_statement
: '{' '}' {printf("compound_statement: '{' '}' \n");}
| '{' statement_list '}' {printf("compound_statement: '{' statement_list '}' \n");}
| '{' declaration_list '}' {printf("compound_statement: '{' declaration_list '}' \n");}
| '{' declaration_list statement_list '}' {printf("compound_statement: '{' declaration_list
statement_list '}' \n");}
;

declaration_list
: declaration {printf("declaration_list: declaration \n");}
| declaration_list declaration {printf("declaration_list: declaration_list declaration \n");}
;

statement_list
: statement {printf("statement_list: statement \n");}
| statement_list statement {printf("statement_list: statement_list statement \n");}
;

expression_statement
: ';' {printf("expression_statement: ';' \n");}
| expression ';' {printf("expression_statement: expression ';' \n");}
;

selection_statement
```

```
: IF '(' expression ')' statement %prec NO_ELSE {printf("selection_statement: IF '(' expression ')'
statement \n");}
| IF '(' expression ')' statement ELSE statement {printf("selection_statement: IF '(' expression ')'
statement ELSE statement \n");}
| SWITCH '(' expression ')' statement {printf("selection_statement: SWITCH '(' expression ')'
statement \n");}
;


iteration_statement
: WHILE '(' expression ')' statement {printf("iteration_statement: WHILE '(' expression ')'
statement \n");}
| DO statement WHILE '(' expression ')' ';' {printf("iteration_statement: DO statement WHILE '('
expression ')' ';' \n");}
| FOR '(' expression_statement expression_statement ')' statement {printf("iteration_statement:
FOR '(' expression_statement expression_statement ')' statement \n");}
| FOR '(' expression_statement expression_statement expression ')' statement
{printf("iteration_statement: FOR '(' expression_statement expression_statement expression ')'
statement \n");}
;

jump_statement
: GOTO IDENTIFIER ';' {printf("jump_statement: GOTO IDENTIFIER ';' \n");}
| CONTINUE ';' {printf("jump_statement: CONTINUE ';' \n");}
| BREAK ';' {printf("jump_statement: BREAK ';' \n");}
| RETURN ';' {printf("jump_statement: RETURN ';' \n");}
| RETURN expression ';' {printf("jump_statement: RETURN expression ';' \n");}
;

translation_unit
: external_declaration {printf("translation_unit: external_declaration \n");}
| translation_unit external_declaration {printf("translation_unit: translation_unit
external_declaration \n");}
;

external_declaration
: function_definition {printf("external_declaration: function_definition \n");}
| declaration {printf("external_declaration: declaration \n");}
;

function_definition
: declaration_specifiers declarator declaration_list compound_statement
{printf("function_definition: declaration_specifiers declarator declaration_list compound_statement\
n");}
| declaration_specifiers declarator compound_statement {printf("function_definition:
declaration_specifiers declarator compound_statement \n");}
| declarator declaration_list compound_statement {printf("function_definition: declarator
declaration_list compound_statement \n");}
| declarator compound_statement {printf("function_definition: declarator compound_statement \
n");}
;
%%
int main(int argc, char *argv[])
{
if(argc>1)
{
yyin = fopen(argv[1],"rt");
}
yyparse();
}
```

Pruebas y descripciones:

Código de prueba:

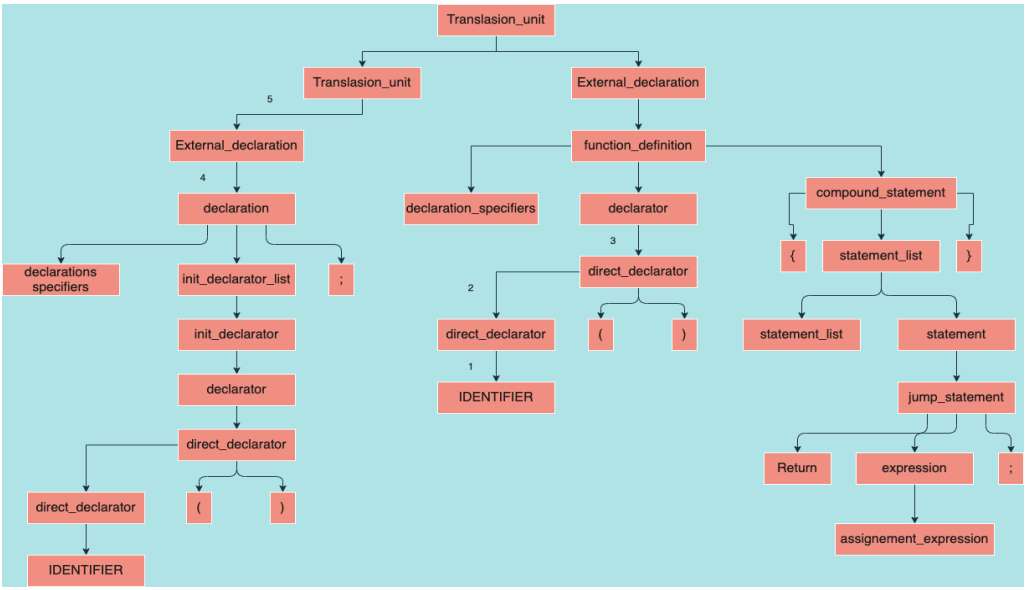
```c
/* C program to printf a sentence.*/

int printf();

int main()

{

printf("C PROGRAMMING //");

return 0;

}
```

Salida de consola:

```
Last login: Sun May  2 13:30:58 on ttys000
luisnunez.@MacBook-Air-de-Luis ~ % cd ejercicios
luisnunez.@MacBook-Air-de-Luis ejercicios % cd analizador\ sintactico
luisnunez.@MacBook-Air-de-Luis analizador sintactico % open codigo.c
luisnunez.@MacBook-Air-de-Luis analizador sintactico % ./parser codigo.c
Analizador Sintactico de ANSI C 2021 version 0.5

type_specifier: INT
declaration_specifiers:
direct_declarator: IDENTIFIER
direct_declarator: direct_declarator '(' ')'
declarator: direct_declarator
init_declarator: declaratorinit_declarator_list: init_declaratordeclaration:
declaration_specifiers init_declarator_list ';'
external_declaration: declaration
translation_unit: external_declaration
type_specifier: INT
declaration_specifiers:
direct_declarator: IDENTIFIER
direct_declarator: direct_declarator '(' ')'
declarator: direct_declarator
primary_expression: IDENTIFIER
postfix_expression: primary_expression
Start of  the string
primary_expression: STRING_LITERAL
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: assignment_expression
postfix_expression: postfix_expression '(' argument_expression_list ')'
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
```

additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement
primary_expression: CONSTANT
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
jump_statement: RETURN expression ';'
statement: jump_statement
statement_list: statement_list statement
compound_statement: '{' statement_list '}'
function_definition: declaration_specifiers declarator compound_statement
external_declaration: function_definition
translation_unit: external_declaration
luisnunez.@MacBook-Air-de-Luis analizador sintactico %

Árbol sintáctico generado:



Código de prueba:

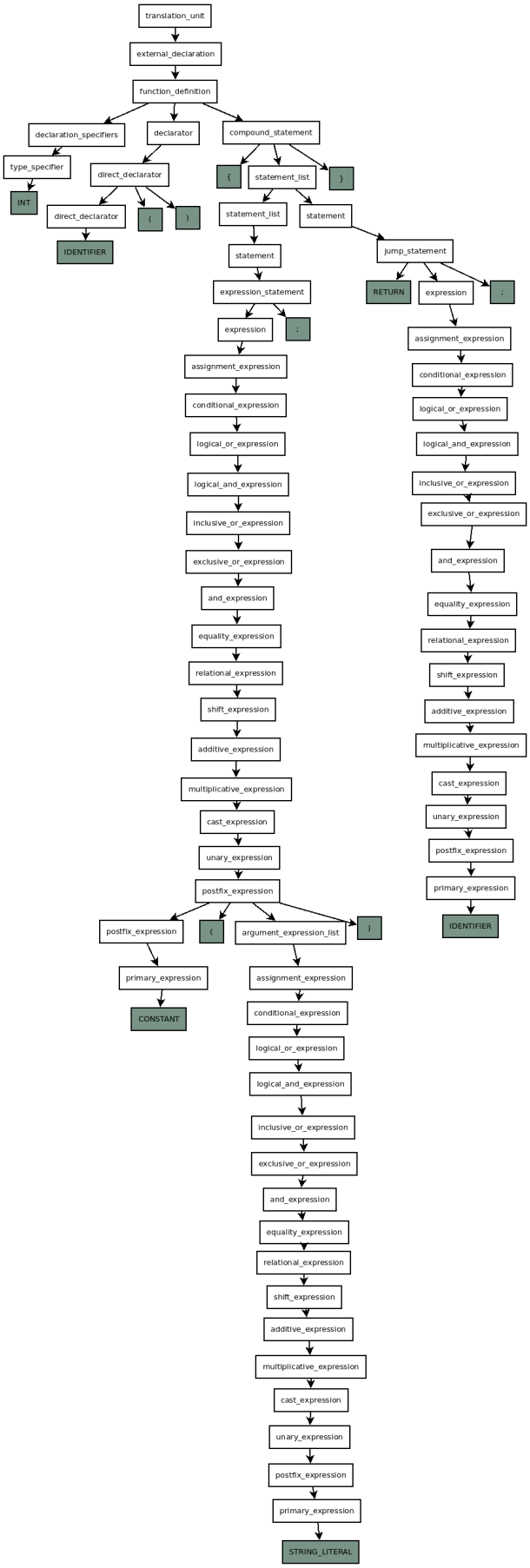
```
#include <stdio.h>

#include <stdlib.h>

int main()

{

printf("Hello world!");

return 0;

}
```

Salida de consola:

```
type_specifier: INT
declaration_specifiers: type_specifier
direct_declarator: IDENTIFIER
direct_declarator: direct_declarator '(' ')'
declarator: direct_declarator
primary_expression: IDENTIFIER
postfix_expression: primary_expression
primary_expression: STRING_LITERAL
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: assignment_expression
postfix_expression: postfix_expression '(' argument_expression_list ')'
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement
primary_expression: CONSTANT
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
```

multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
jump_statement: RETURN expression ';'
statement: jump_statement
statement_list: statement_list statement
compound_statement: '{' statement_list '}'
function_definition: declaration_specifiers declarator compound_statement
external_declaration: function_definition
translation_unit: external_declaration

Árbol sintáctico generado:



Código de prueba:

```
#include <stdio.h>

int main()

{

int vlr=1, i, ttl;

ttl=vlr+4;

for(i=0; i<ttl; i++)

{

printf("Valor numero: ",i);

}

return 0;

}
```

Salida de consola:

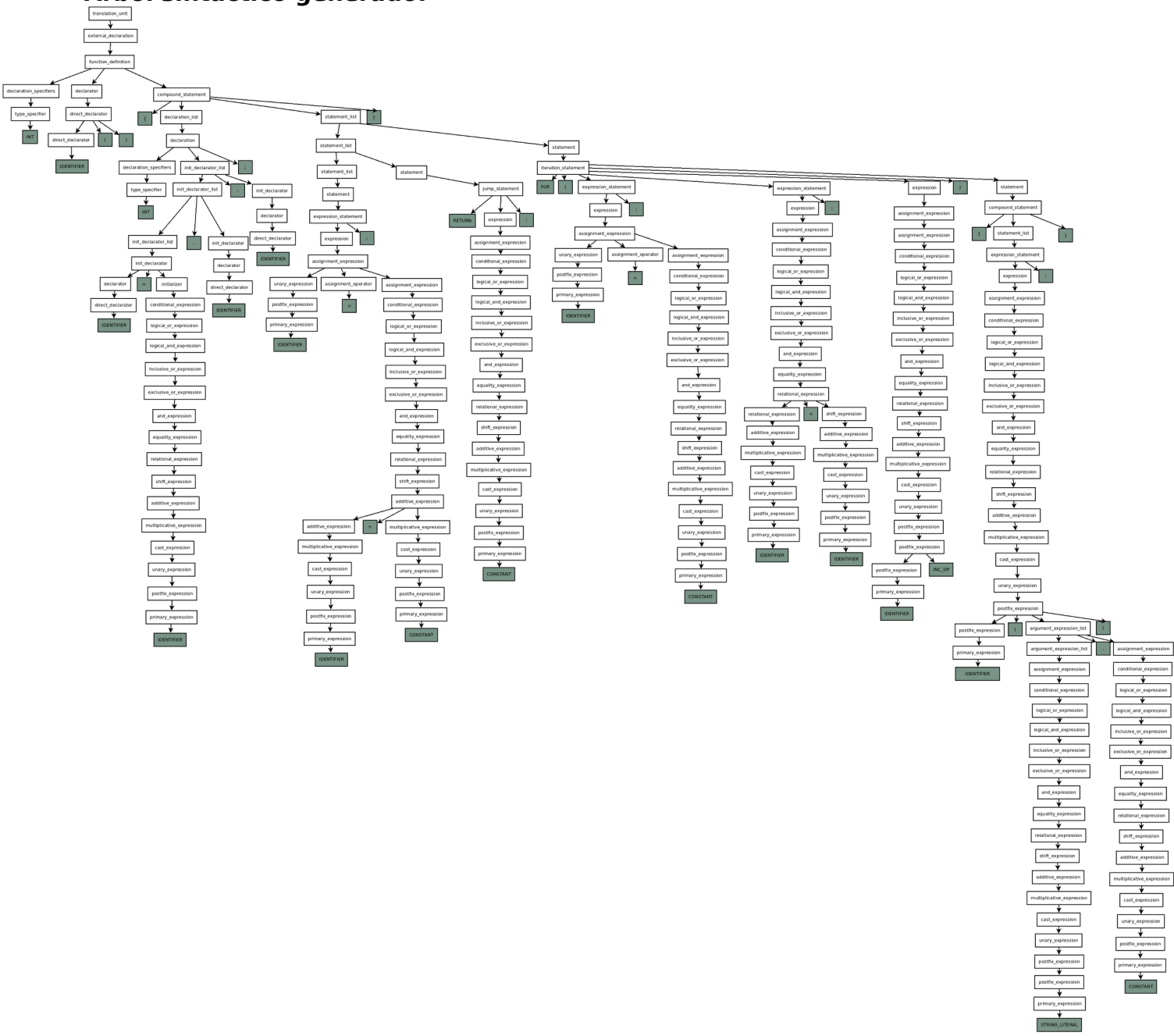
```
type_specifier: INT
declaration_specifiers: type_specifier
direct_declarator: IDENTIFIER
direct_declarator: direct_declarator '(' ')'
declarator: direct_declarator
type_specifier: INT
declaration_specifiers: type_specifier
direct_declarator: IDENTIFIER
declarator: direct_declarator
primary_expression: CONSTANT
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
initializer: assignment_expression
init_declarator: declarator '=' initializer
init_declarator_list: init_declarator
direct_declarator: IDENTIFIER
declarator: direct_declarator
init_declarator: declarator
init_declarator_list: init_declarator_list ',' init_declarator
direct_declarator: IDENTIFIER
declarator: direct_declarator
init_declarator: declarator
init_declarator_list: init_declarator_list ',' init_declarator
declaration: declaration_specifiers init_declarator_list ';'
declaration_list: declaration
primary_expression: IDENTIFIER
```

postfix_expression: primary_expression
unary_expression: postfix_expression
assignment_operator: '='
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
primary_expression: CONSTANT
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: additive_expression '+' multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
assignment_expression: unary_expression assignment_operator assignment_expression
expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
assignment_operator: '='
primary_expression: CONSTANT
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
assignment_expression: unary_expression assignment_operator assignment_expression
expression: assignment_expression
expression_statement: expression ';'
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression

additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: relational_expression '<' shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
expression_statement: expression ';'
primary_expression: IDENTIFIER
postfix_expression: primary_expression
postfix_expression: postfix_expression INC_OP
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
primary_expression: IDENTIFIER
postfix_expression: primary_expression
primary_expression: STRING_LITERAL
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: assignment_expression
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression

argument_expression_list: argument_expression_list ',' assignment_expression
postfix_expression: postfix_expression '(' argument_expression_list ')'
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement
compound_statement: '{' statement_list '}'
statement: compound_statement
iteration_statement: FOR '(' expression_statement expression_statement expression ')'
statement
statement: iteration_statement
statement_list: statement_list statement
primary_expression: CONSTANT
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
jump_statement: RETURN expression ';'
statement: jump_statement
statement_list: statement_list statement
compound_statement: '{' declaration_list statement_list '}'
function_definition: declaration_specifiers declarator compound_statement
external_declaration: function_definition
translation_unit: external_declaration

Árbol sintáctico generado:



Código de prueba:

```
#include<stdio.h>
int digito1=0;
int digito2=1;
int Resultado=0;
int main()
{
int i, digito;
printf("Ingrese el número de posición de la serie de fibonacci\n");
printf("Ingrese el número: \n");
scanf("%d", &digito);
if(digito<0)
{
printf("La serie de fibonacci no trabaja números negativos\n");
}
if(digito==0)
{
printf("0\n");
}
for(i=0; i<digito; i++)
{
digito1=digito2;
digito2=Resultado;
Resultado=digito1+digito2;
}
printf("La serie es: %d\n", Resultado);
return 0;
}
```

Salida de consola:

```
Last login: Sun May  2 13:01:07 on ttys000
luisnunez.@MacBook-Air-de-Luis ~ % cd ejercicios
luisnunez.@MacBook-Air-de-Luis ejercicios % cd analizador\ sintactico
luisnunez.@MacBook-Air-de-Luis analizador sintactico % ./parser itefibo.c
Analizador Sintactico de ANSI C 2021 version 0.5

type_specifier: INT
declaration_specifiers:
direct_declarator: IDENTIFIER
declarator: direct_declarator
primary_expression: CONSTANT
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
initializer: assignment_expression
init_declarator: declarator '=' initializerinit_declarator_list: init_declaratordeclaration:
declaration_specifiers init_declarator_list ';'
external_declaration: declaration
translation_unit: external_declaration
type_specifier: INT
declaration_specifiers:
direct_declarator: IDENTIFIER
declarator: direct_declarator
primary_expression: CONSTANT
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
initializer: assignment_expression
init_declarator: declarator '=' initializerinit_declarator_list: init_declaratordeclaration:
declaration_specifiers init_declarator_list ';'
external_declaration: declaration
translation_unit: external_declaration
type_specifier: INT
declaration_specifiers:
direct_declarator: IDENTIFIER
declarator: direct_declarator
primary_expression: CONSTANT
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
```

multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
initializer: assignment_expression
init_declarator: declarator '=' initializerinit_declarator_list: init_declaratordeclaration:
declaration_specifiers init_declarator_list ';'
external_declaration: declaration
translation_unit: external_declaration
type_specifier: INT
declaration_specifiers:
direct_declarator: IDENTIFIER
direct_declarator: direct_declarator '(' ')'
declarator: direct_declarator
type_specifier: INT
declaration_specifiers:
direct_declarator: IDENTIFIER
declarator: direct_declarator
init_declarator: declaratorinit_declarator_list: init_declaratordirect_declarator: IDENTIFIER
declarator: direct_declarator
init_declarator: declaratorinit_declarator_list: init_declarator_list ','
init_declaratordeclaration: declaration_specifiers init_declarator_list ';'
declaration_list: declaration
primary_expression: IDENTIFIER
postfix_expression: primary_expression
Start of the string
primary_expression: STRING_LITERAL
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: assignment_expression
postfix_expression: postfix_expression '(' argument_expression_list ')'
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression

expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement
primary_expression: IDENTIFIER
postfix_expression: primary_expression
Start of the string
primary_expression: STRING_LITERAL
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: assignment_expression
postfix_expression: postfix_expression '(' argument_expression_list ')'
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement_list statement
primary_expression: IDENTIFIER
postfix_expression: primary_expression
Start of the string
primary_expression: STRING_LITERAL
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: assignment_expression
unary_operator: '&'
primary_expression: IDENTIFIER
postfix_expression: primary_expression

unary_expression: postfix_expression
cast_expression: unary_expression
unary_expression: unary_operator cast_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: argument_expression_list ',' assignment_expression
postfix_expression: postfix_expression '(' argument_expression_list ')'
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement_list statement
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
primary_expression: CONSTANT
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: relational_expression '<' shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
primary_expression: IDENTIFIER
postfix_expression: primary_expression
Start of the string
primary_expression: STRING_LITERAL
postfix_expression: primary_expression

unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: assignment_expression
postfix_expression: postfix_expression '(' argument_expression_list ')'
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement
compound_statement: '{' statement_list '}'
statement: compound_statement
selection_statement: IF '(' expression ')' statement
statement: selection_statement
statement_list: statement_list statement
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
primary_expression: CONSTANT
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: equality_expression EQ_OP relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
primary_expression: IDENTIFIER

postfix_expression: primary_expression
Start of the string
primary_expression: STRING_LITERAL
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: assignment_expression
postfix_expression: postfix_expression '(' argument_expression_list ')'
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement
compound_statement: '{' statement_list '}'
statement: compound_statement
selection_statement: IF '(' expression ')' statement
statement: selection_statement
statement_list: statement_list statement
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
assignment_operator: '='
primary_expression: CONSTANT
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
assignment_expression: unary_expression assignment_operator assignment_expression
expression: assignment_expression
expression_statement: expression ';'

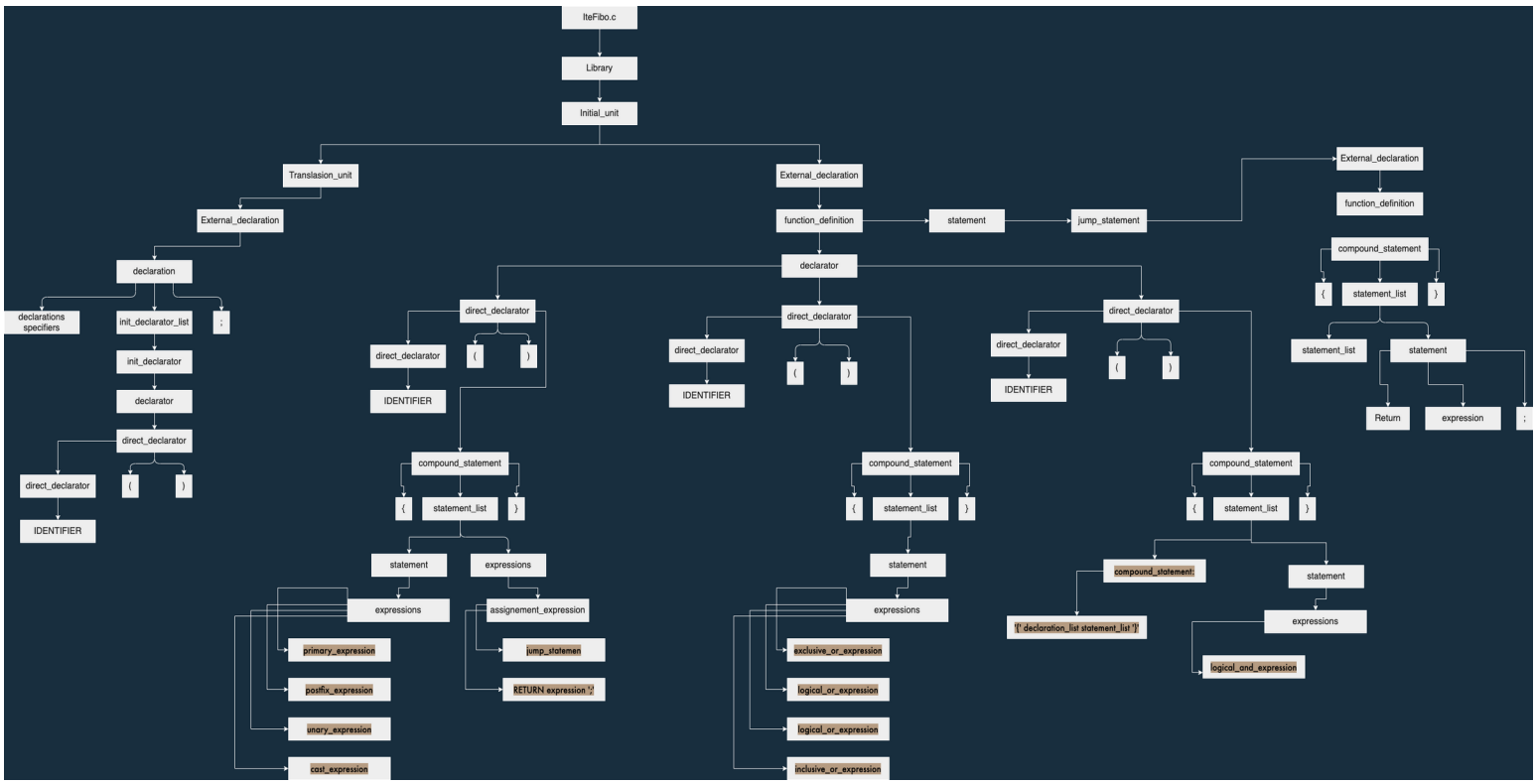
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: relational_expression '<' shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
expression_statement: expression ';'
primary_expression: IDENTIFIER
postfix_expression: primary_expression
postfix_expression: postfix_expression INC_OP
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
assignment_operator: '='
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
assignment_expression: unary_expression assignment_operator assignment_expression
expression: assignment_expression

expression_statement: expression ';'
statement: expression_statement
statement_list: statement
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
assignment_operator: '='
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
assignment_expression: unary_expression assignment_operator assignment_expression
expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement_list statement
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
assignment_operator: '='
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: additive_expression '+' multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
assignment_expression: unary_expression assignment_operator assignment_expression
expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement_list statement
compound_statement: '{' statement_list '}'
statement: compound_statement
iteration_statement: FOR '(' expression_statement expression_statement expression ')'
statement
statement: iteration_statement
statement_list: statement_list statement
primary_expression: IDENTIFIER

postfix_expression: primary_expression
Start of the string
primary_expression: STRING_LITERAL
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: assignment_expression
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: argument_expression_list ',' assignment_expression
postfix_expression: postfix_expression '(' argument_expression_list ')'
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement_list statement
primary_expression: CONSTANT
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression

inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
jump_statement: RETURN expression ';'
statement: jump_statement
statement_list: statement_list statement
compound_statement: '{' declaration_list statement_list '}'
function_definition: declaration_specifiers declarator compound_statement
external_declaration: function_definition
translation_unit: external_declaration
luisnunez.@MacBook-Air-de-Luis analizador sintactico %

Árbol sintáctico generado:



Código de prueba:

```c
#include<stdio.h>

int main()

{

int i, plus=0, numeros, terminos;

printf("¿Cuantos números desea ingresar\n");

scanf("%d", &terminos);

for(i=0; i<terminos; i++)

{

printf("Ingrese el número: \n");

scanf("%d",&numeros);

plus+=numeros;

}

printf("La suma es: %d \n", plus);

return 0;

}
```

Salida de consola:

```
Last login: Sun May  2 13:32:07 on ttys000
luisnunez.@MacBook-Air-de-Luis ~ % cd ejercicios
luisnunez.@MacBook-Air-de-Luis ejercicios % cd analizador\ sintactico
luisnunez.@MacBook-Air-de-Luis analizador sintactico % open suman.c
luisnunez.@MacBook-Air-de-Luis analizador sintactico % ./parser suman.c
Analizador Sintactico de ANSI C 2021 version 0.5

type_specifier: INT
declaration_specifiers:
direct_declarator: IDENTIFIER
direct_declarator: direct_declarator '(' ')'
declarator: direct_declarator
type_specifier: INT
declaration_specifiers:
direct_declarator: IDENTIFIER
declarator: direct_declarator
init_declarator: declaratorinit_declarator_list:
init_declaratordirect_declarator: IDENTIFIER
declarator: direct_declarator
primary_expression: CONSTANT
```

postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
initializer: assignment_expression
init_declarator: declarator '=' initializerinit_declarator_list:
init_declarator_list ',' init_declaratordirect_declarator: IDENTIFIER
declarator: direct_declarator
init_declarator: declaratorinit_declarator_list: init_declarator_list ','
init_declaratordirect_declarator: IDENTIFIER
declarator: direct_declarator
init_declarator: declaratorinit_declarator_list: init_declarator_list ','
init_declaratordeclaration: declaration_specifiers init_declarator_list ';'
declaration_list: declaration
primary_expression: IDENTIFIER
postfix_expression: primary_expression
Start of the string
primary_expression: STRING_LITERAL
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: assignment_expression
postfix_expression: postfix_expression '(' argument_expression_list ')'
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression

exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement
primary_expression: IDENTIFIER
postfix_expression: primary_expression
Start of the string
primary_expression: STRING_LITERAL
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: assignment_expression
unary_operator: '&'
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
unary_expression: unary_operator cast_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: argument_expression_list ','
assignment_expression
postfix_expression: postfix_expression '(' argument_expression_list ')'
unary_expression: postfix_expression
cast_expression: unary_expression

multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement_list statement
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
assignment_operator: '='
primary_expression: CONSTANT
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
assignment_expression: unary_expression assignment_operator
assignment_expression
expression: assignment_expression
expression_statement: expression ';'
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression

shift_expression: additive_expression
relational_expression: relational_expression '<' shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
expression_statement: expression ';'
primary_expression: IDENTIFIER
postfix_expression: primary_expression
postfix_expression: postfix_expression INC_OP
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
primary_expression: IDENTIFIER
postfix_expression: primary_expression
Start of the string
primary_expression: STRING_LITERAL
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: assignment_expression
postfix_expression: postfix_expression '(' argument_expression_list ')'
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression

additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement
primary_expression: IDENTIFIER
postfix_expression: primary_expression
Start of the string
primary_expression: STRING_LITERAL
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: assignment_expression
unary_operator: '&'
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
unary_expression: unary_operator cast_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression

argument_expression_list: argument_expression_list ','
assignment_expression
postfix_expression: postfix_expression '(' argument_expression_list ')'
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement_list statement
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
assignment_operator: ADD_ASSIGN
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
assignment_expression: unary_expression assignment_operator
assignment_expression
expression: assignment_expression
expression_statement: expression ';'
statement: expression_statement
statement_list: statement_list statement
compound_statement: '{' statement_list '}'
statement: compound_statement
iteration_statement: FOR '(' expression_statement expression_statement
expression ')' statement
statement: iteration_statement
statement_list: statement_list statement
primary_expression: IDENTIFIER

postfix_expression: primary_expression
Start of the string
primary_expression: STRING_LITERAL
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: assignment_expression
primary_expression: IDENTIFIER
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
argument_expression_list: argument_expression_list ','
assignment_expression
postfix_expression: postfix_expression '(' argument_expression_list ')'
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
expression_statement: expression ';'

```
statement: expression_statement
statement_list: statement_list statement
primary_expression: CONSTANT
postfix_expression: primary_expression
unary_expression: postfix_expression
cast_expression: unary_expression
multiplicative_expression: cast_expression
additive_expression: multiplicative_expression
shift_expression: additive_expression
relational_expression: shift_expression
equality_expression: relational_expression
and_expression: equality_expression
exclusive_or_expression: and_expression
inclusive_or_expression: exclusive_or_expression
logical_and_expression: inclusive_or_expression
logical_or_expression: logical_and_expression
conditional_expression: logical_or_expression
assignment_expression: conditional_expression
expression: assignment_expression
jump_statement: RETURN expression ';'
statement: jump_statement
statement_list: statement_list statement
compound_statement: '{' declaration_list statement_list '}'
function_definition: declaration_specifiers declarator compound_statement
external_declaration: function_definition
translation_unit: external_declaration
luisnunez.@MacBook-Air-de-Luis analizador sintactico %
```

Árbol sintáctico generado:

Conclusiones:

Esta práctica fue de ayuda para entender cómo funciona la estructura de comandos de un código en lenguaje C y como es que el analizador de un compilador mapea dichos comandos para ir generando una estructura de árbol sintáctico acorde a las expresiones lógicas y matemáticas que pueda interpretar para después utilizarlo en el generador de código intermedio.