

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 para 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 deben 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("Ilegal 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("Ilegal 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 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 '}'

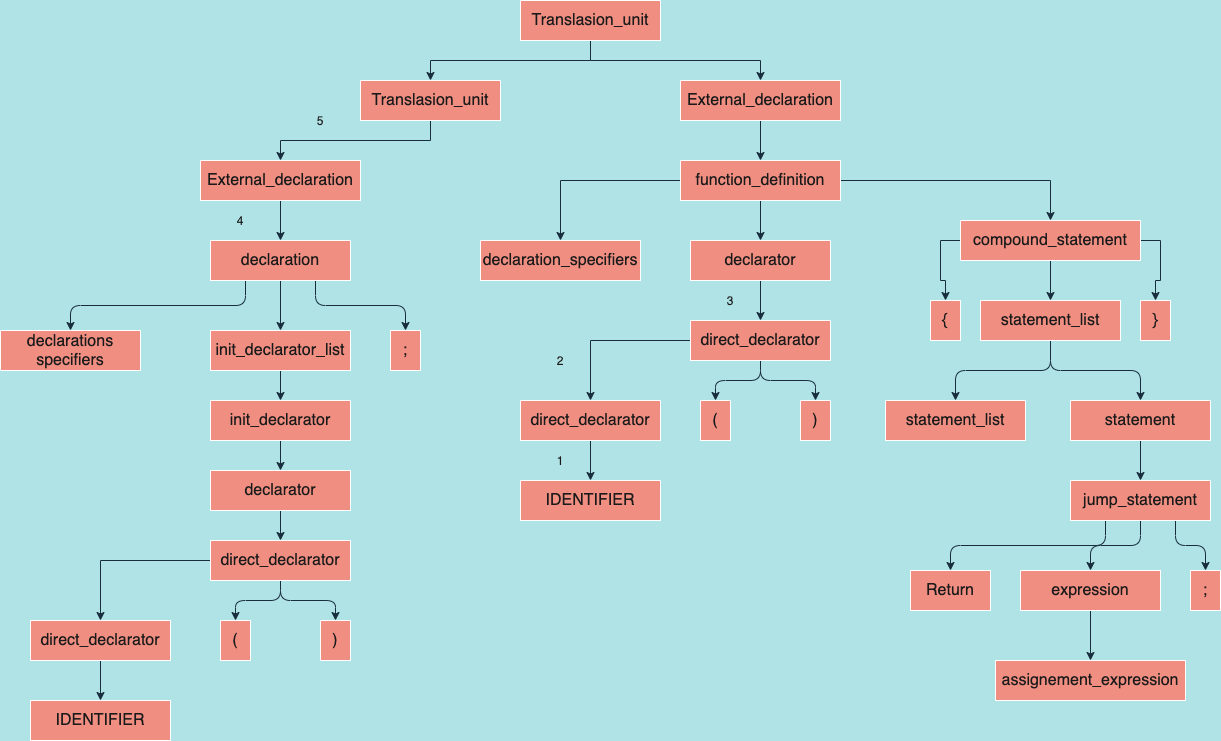
function\_definition: declaration\_specifiers declarator compound\_statement

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

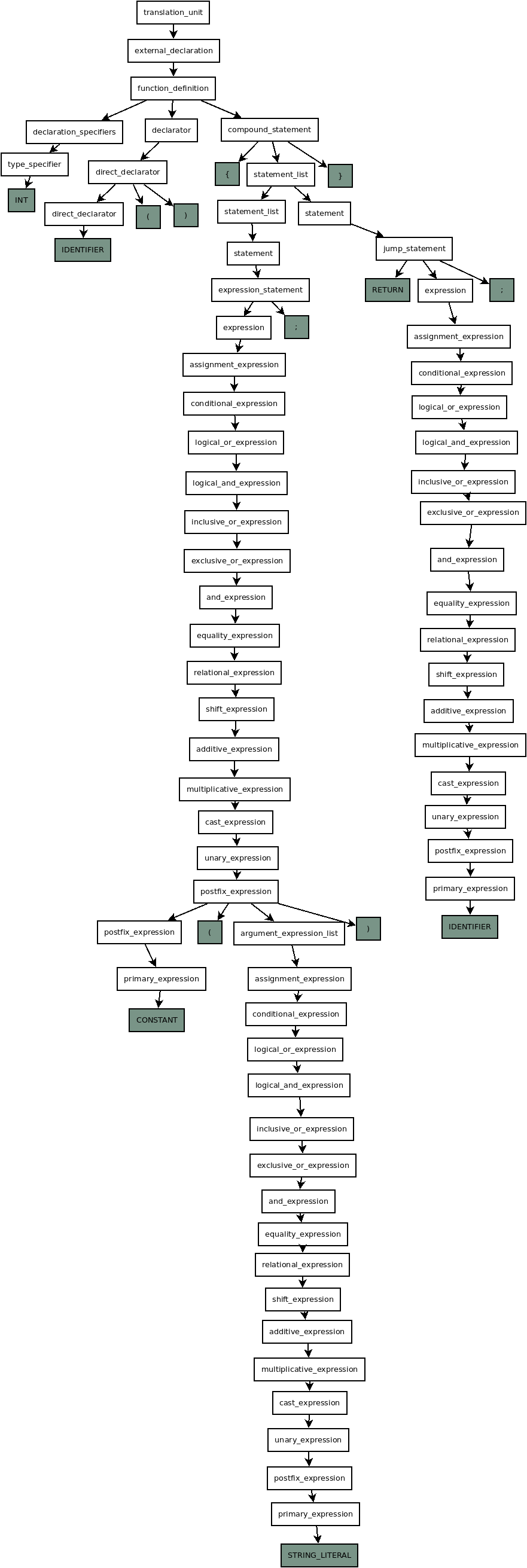
compound\_statement: '{' statement\_list '}'

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

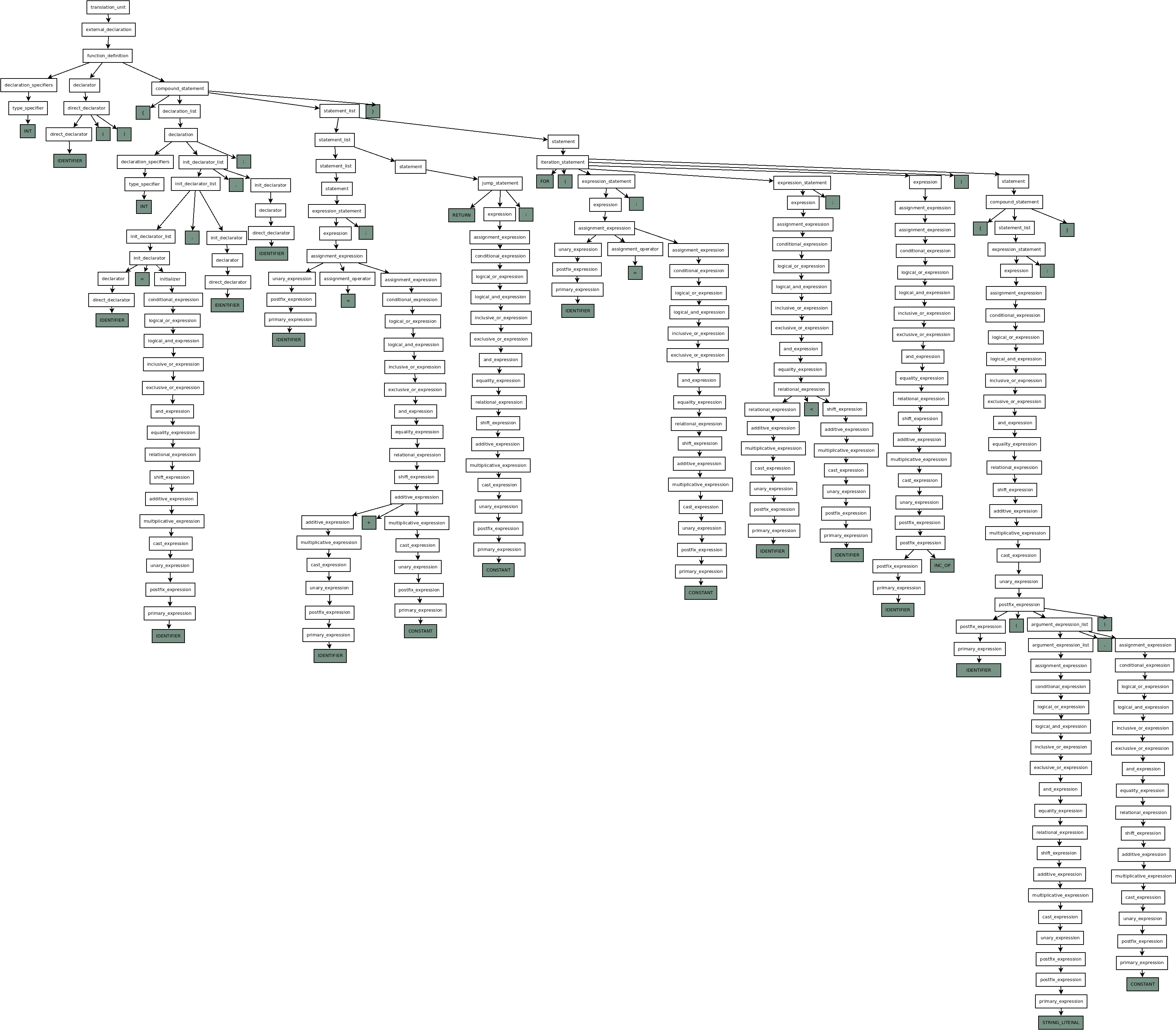
compound\_statement: '{' declaration\_list statement\_list '}'

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 '}'

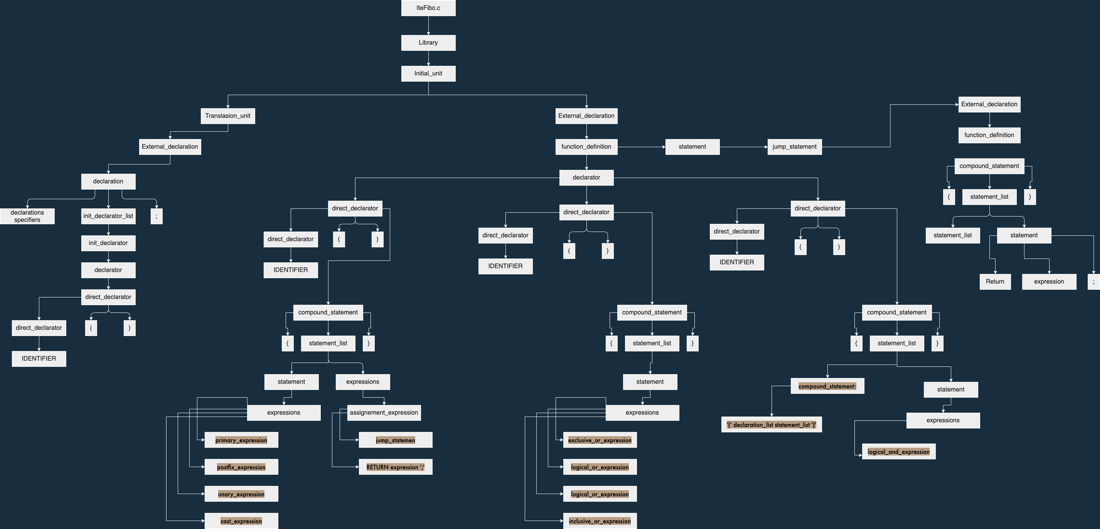
function\_definition: declaration\_specifiers declarator compound\_statement

external\_declaration: function\_definition

translation\_unit: external\_declaration

luisnunez.@MacBook-Air-de-Luis analizador sintactico %

**Árbol sintáctico generado:**



**Código de prueba:**

*#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 '}'

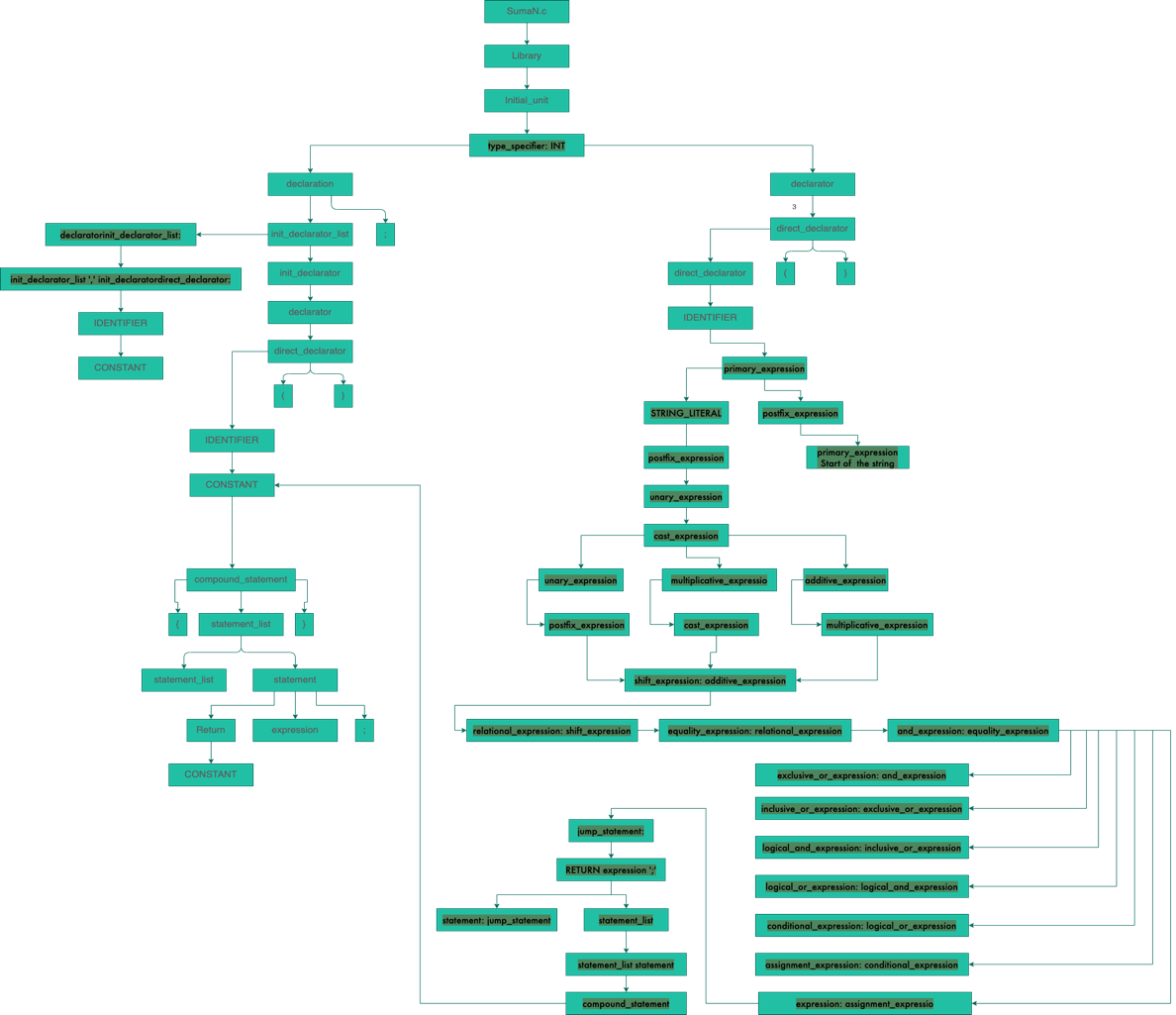
function\_definition: declaration\_specifiers declarator compound\_statement

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.