



UNIVERSIDAD NACIONAL DE SAN AGUSTIN

PROYECTO FINAL

Desarrollo de un Compilador

Bejar Merma Angel Andres

Docente

Dr. Yuber Elmer Velazco Paredes BROWN

7 de agosto de 2020

1. Introducción

Un compilador es un programa que transforma el código fuente escrito en un lenguaje de programación (el idioma de origen) en otro lenguaje de computadora (el idioma de destino), y este último a menudo tiene una forma binaria conocido como código objeto. Al ejecutar, el compilador primero analiza todas las declaraciones del lenguaje sintácticamente uno tras otro y luego, en una o más etapas sucesivas construye el código de salida, asegurándose de que las declaraciones se mencionan correctamente en el código final.

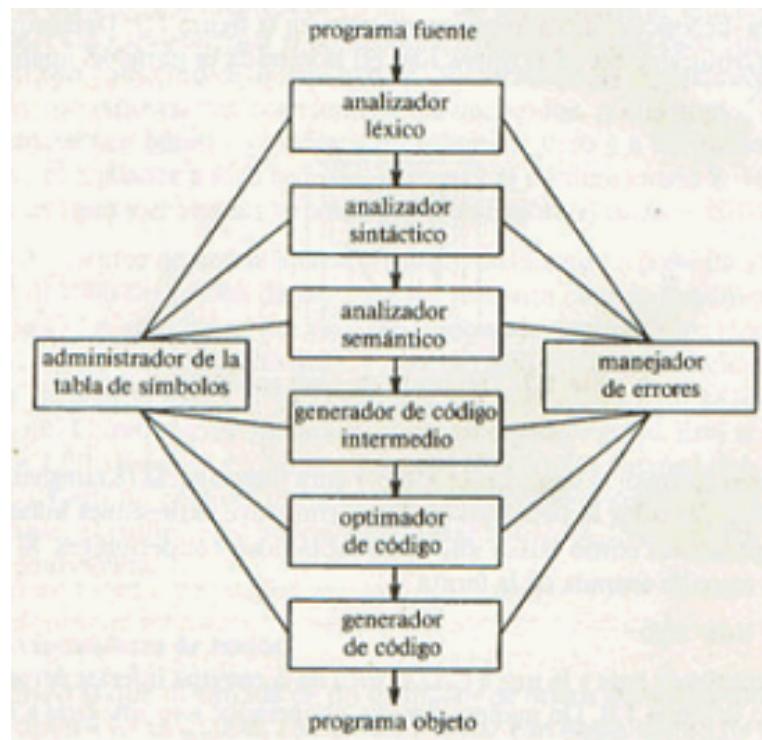


Figura 1: Esquema de un Compilador

El análisis sintáctico² convierte el texto de entrada en otras estructuras (comúnmente árboles), que son más útiles para el posterior análisis y capturan la jerarquía implícita de la entrada. Un analizador léxico crea tokens de una secuencia de caracteres de entrada y son estos tokens los que son procesados por el analizador sintáctico para construir la estructura de datos, por ejemplo un árbol de análisis o arboles.

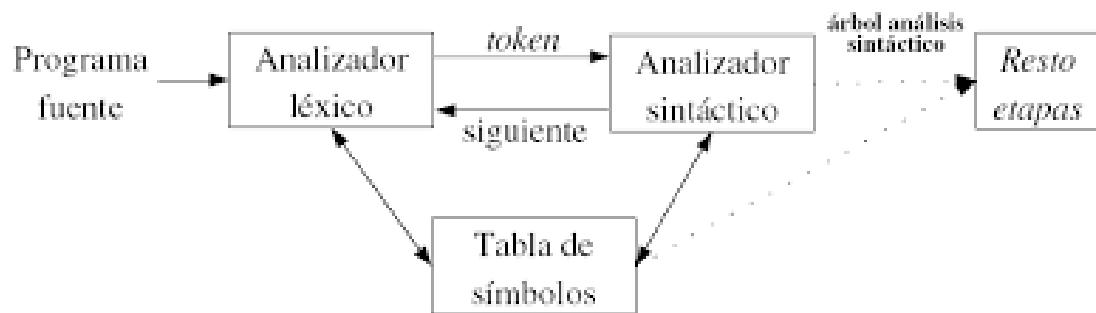


Figura 2: Esquema de funcionamiento Analisador Sintactico

Se hizo la comprobación de tipo de variable, manejo del alcance de las variables, parámetros de función verificación de tipo, número de parámetros que coinciden en la llamada a la función y la matriz verificación de dimensionalidad junto con verificación de tipo de índice de array[1].

También se manejo los casos de variables no declaradas y redeclaración de variables. El analizador semántico. también comprueba si las funciones predefinidas (como printf, scanf, gets, getchar, etc.) son redefinido en el programa.

1.1. Objetivo

Desarrollar un compilador usando YACC.

2. Marco Referencial

Se pueden distinguir Cuatro fases en el desarrollo de un compilador , el front end del Compilador son la fase léxica, fase de sintaxis, fase semántica y Generacion de código intermedio.

, el analizador semántico verificará significado real de la declaración analizada en el árbol de análisis.

El análisis semántico es principalmente un proceso en la construcción del compilador después del análisis para reunir la semántica necesaria información del código fuente. como se muestra en Figura 3.

El analizador semántico verifica si la sintaxis La estructura construida en el programa fuente deriva cualquier significado o no. Es el fase en la que el compilador agrega información semántica al árbol de análisis y construye La tabla de símbolos. El analizador semántico también se llama análisis sensible al contexto. Esta La fase realiza comprobaciones semánticas como la comprobación de tipo (comprobación de errores de tipo), o asignación definida (variable a inicializar antes del uso), rechazando incorrecto programas o emitiendo advertencias. El análisis semántico sigue lógicamente la fase de análisis, y lógicamente precede a la fase de generación de código.

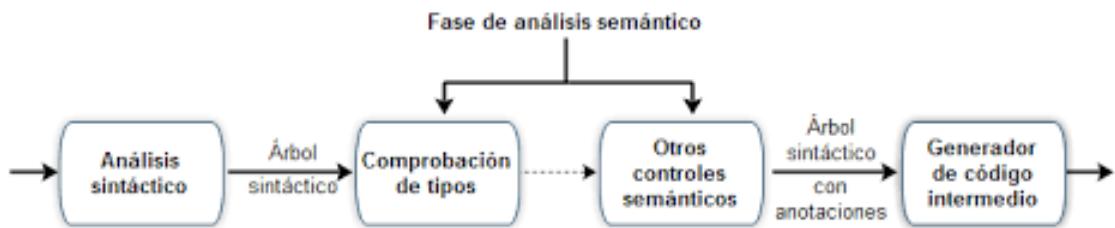


Figura 3: Esquema del Analisador Semantico

la notacion bnf es un lenguaje usado para hablar acerca de otro lenguaje al lenguaje al cual se habla se llama lenguaje objeto para gramaticas libres de contexto

2.1. YACC

Yacc es un programa que genera analizadores sintáticos^[2] su nombre significa Yet Another Compiler-Compiler, es decir otro generador de compiladores mas .Los analizadores de YACC esperan invocar a una rutina llamada 'yylex()' para encontrar el próximo token de entrada. La rutina se supone que devuelve el tipo del próximo token además de poner cualquier valor asociado en la variable global yyval.

Un archivo Lex tiene el siguiente formato:

```

%{
declaraciones en C
%}
Declaraciones de Bison/Yacc
%%
```

Reglas gramaticales
% %
Código C adicional

2.2. Lex

Flex es una herramienta para generar escáneres: programas que reconocen patrones léxicos en un texto. En otras palabras, Flex se encarga de convertir esos patrones léxicos en tokens que pueden servir para estructurar una gramática (en el caso de un compilador) o para colorear código (En caso de un coloreador de código).

Flex genera el código C equivalente a la descripción pedida por el usuario, código que está contenido en el archivo fuente “lex.yy.c”. Este archivo contiene el código fuente de un autómata finito determinista, capaz de reconocer las expresiones regulares entregadas por el usuario en el archivo de entrada.

Un archivo flex tiene el siguiente formato:

definiciones
% %
reglas
% %
código de usuario

3. Gramatica del Lenguaje

reglas de MiniC

```
%%
programC:    listaDeclC;
listaDeclC:   listaDeclC declC | ;
declC:        Tipo listaVar ';';
declC:        Tipo ID '(' listaPar ')' bloque;
Tipo:         INT | FLOAT;
listaVar:     ID ',' listaVar | ID;
listaPar:     Tipo ID ',' listaPar | Tipo ID;
bloque:       '{' listaVarLoc listaProp '}';
listaVarLoc:  Tipo listaVar ';' listaVarLoc | ;
listaProp:    listaProp prop | ;

prop:          ';' ;
prop:          IF '(' EXPRESION ')' prop;
prop:          IF '(' EXPRESION ')' prop ELSE prop;
prop:          WHILE '(' EXPRESION ')' prop;
prop:          ID '=' EXPRESION ';' ;
prop:          bloque

EXPRESION:    NUM;
EXPRESION:    EXPRESION '+' EXPRESION
EXPRESION:    EXPRESION '-' EXPRESION
EXPRESION:    EXPRESION '*' EXPRESION
EXPRESION:    EXPRESION '/' EXPRESION
EXPRESION:    '(' EXPRESION ')'
EXPRESION:    ID

%%
```

Arbol sintactico

3.1. Analisador Sintactico

El programa en YACC especifica las siguientes producciones:

- Construcción del bucle: while, for, do-while
- Tipo de datos :(signed/unsigned) int, float
- Operación Aritmética y Relacional
- Estructura de Datos :Array
- Funciones definidas por el usuario.
- Palabras clave del lenguaje C
- Comentarios de una o varias líneas
- Identificadores y errores constantes
- Selección de sentencias if-else(anidadas)

3.2. Analisador Semantico

El analizador semántico creado para el subconjunto de lenguaje C informa varios errores presentes (si existe) y tiene las siguientes funcionalidades:

- Verificaciones de variables no declaradas
- Verificaciones para la redeclaración de variables
- Verificación de tipo de variables
- Compatibilidad de operaciones matemáticas.
- Comprueba si el tipo de retorno de una función coincide con el tipo de datos del variable / constante que se devuelve
- Comprueba si el número de parámetros en una llamada de función coincide con el número de parámetros en la definición de la función
- Comprueba si los tipos de datos de los parámetros en la llamada a la función y su definición coinciden

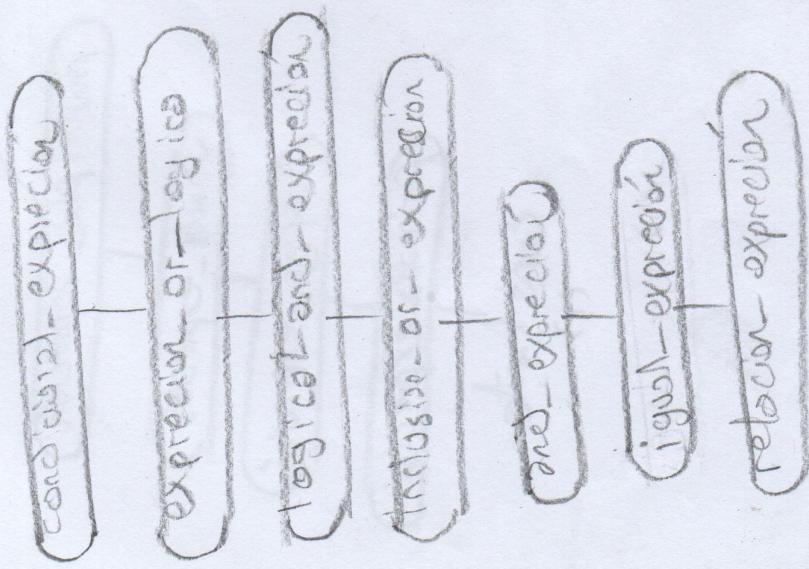
- Comprueba el alcance de las variables e informa un error si una variable está fuera del alcance
- Identifica la dimensionalidad de las matrices y comprueba si el índice de una matriz es un entero positivo o no

Arbol sintactico para la Gramatica

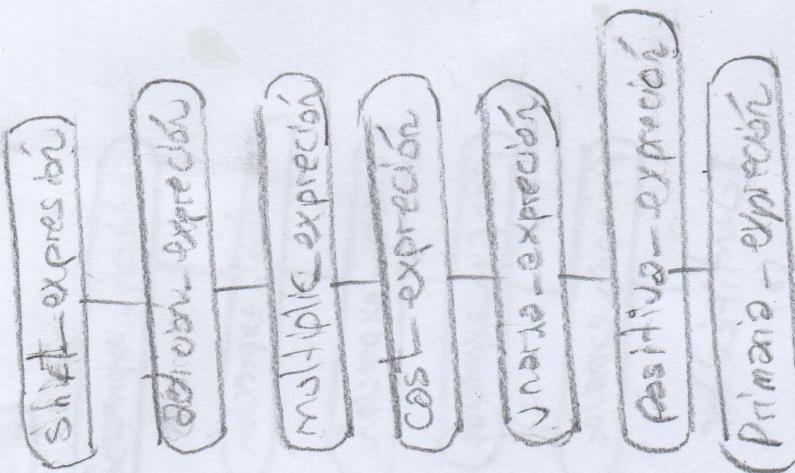
```
2 #include<stdio.h>
3 #define x 3
4 int main()
5 {
6     int a=4;
7     if(a<10)
8     {
9         a = a + 1;
10    }
11    else
12    {
13        a = a + 2;
14    }
15    return;
16 }
```

Figura 4: Gramatica

Forma 1



Forma 2



Pág 1

constante

!atificador

función

variable

declarar

initializar

int

=

!nt

identificador

directo

int

declaración

int

declaración

int

función definida

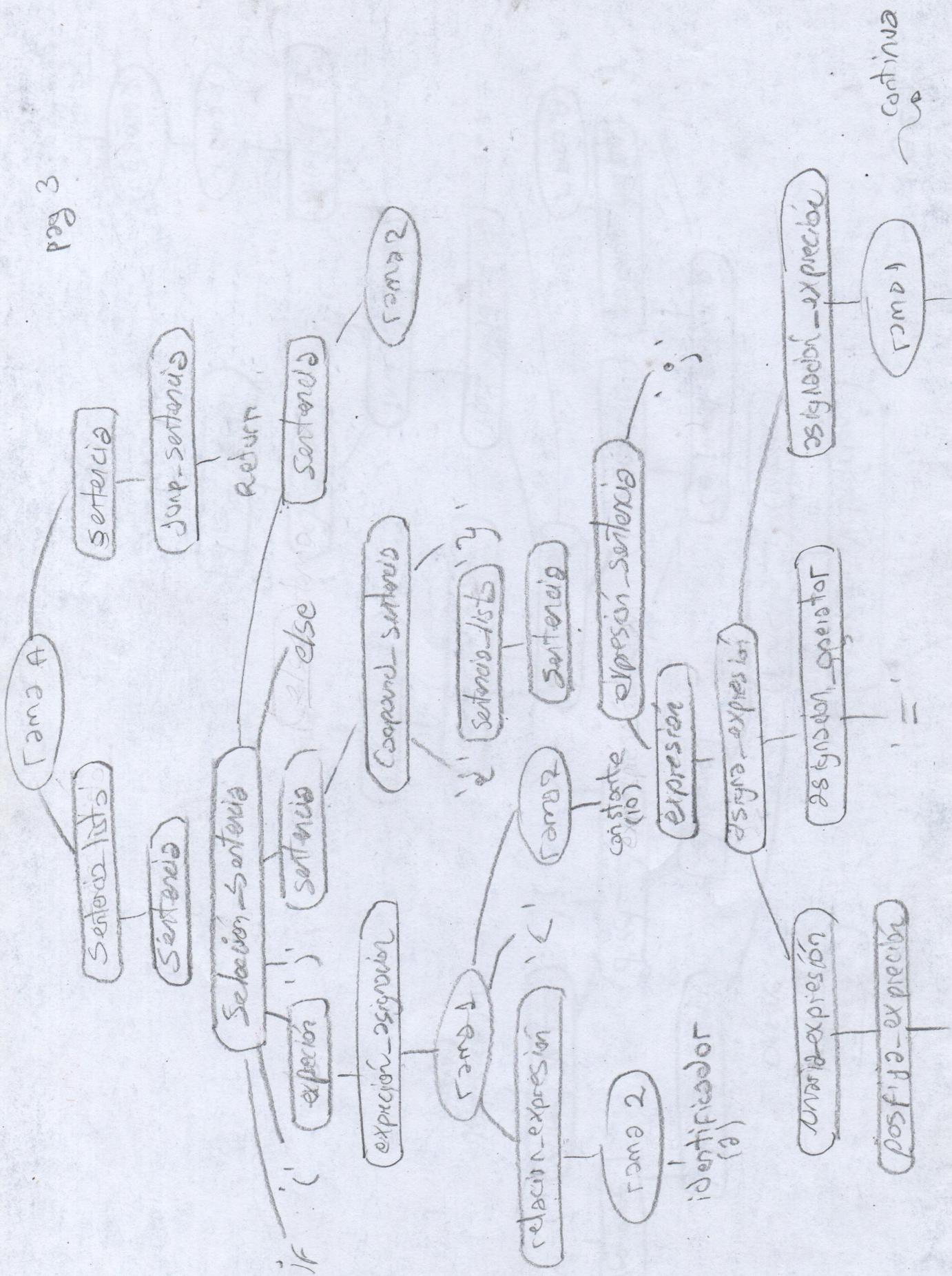
int

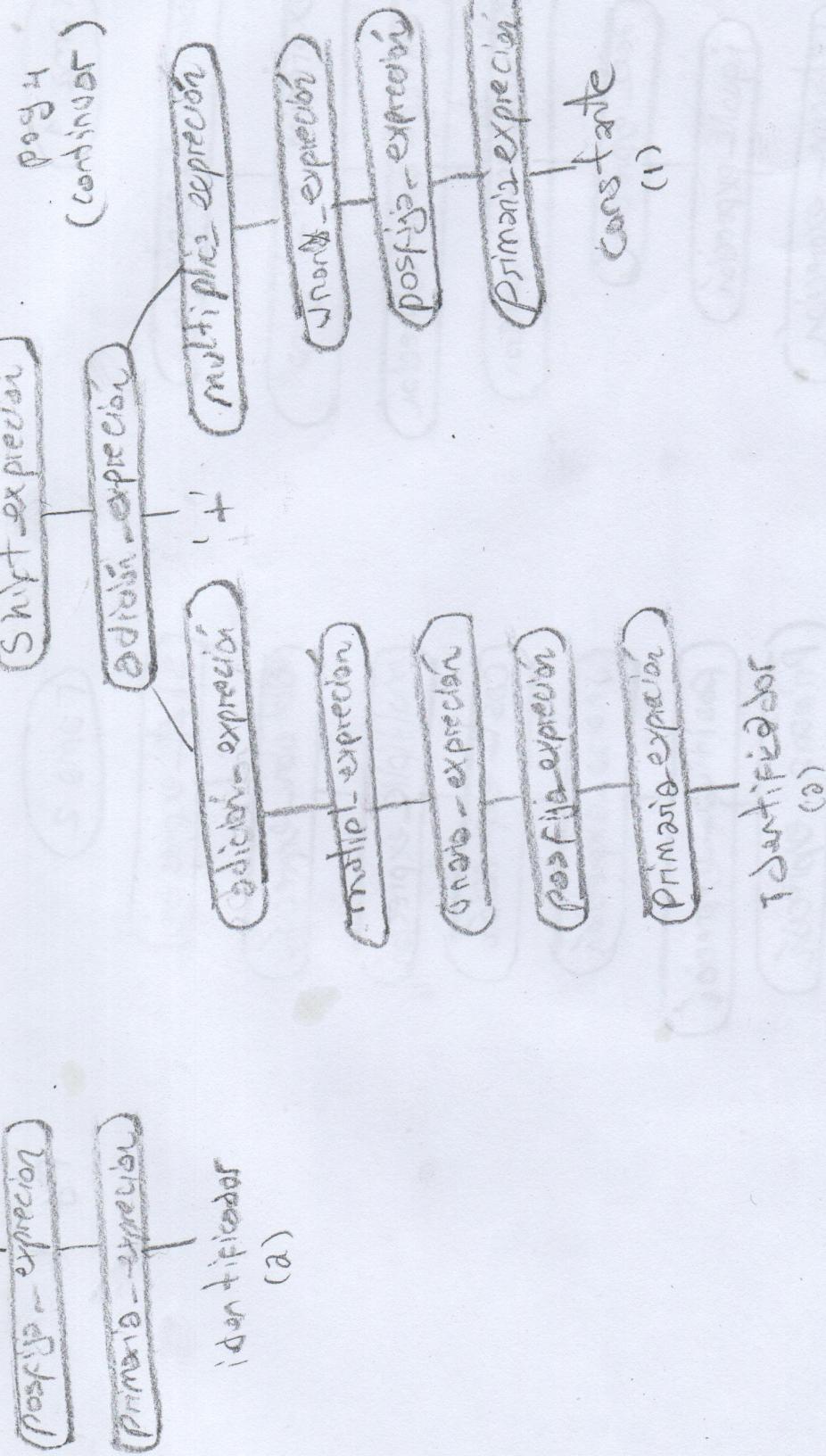
externa - declaración

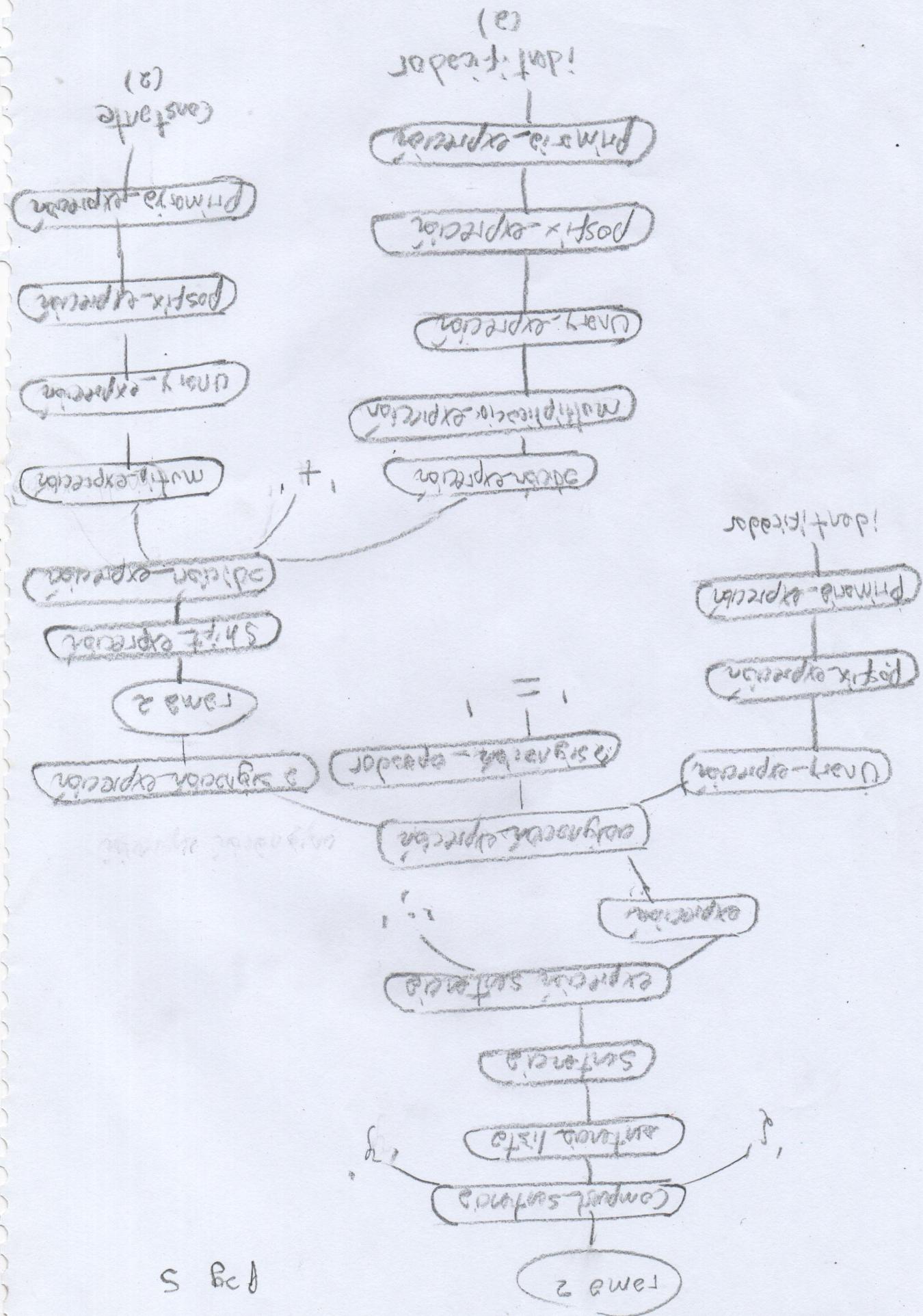
int

begin

Prog 2







4. Desarrollo

Analisador sintactico

El propósito del análisis o análisis sintáctico es verificar que tenemos^[4] Una secuencia válida de tokens. Los tokens son una secuencia válida de símbolos, palabras clave, identificadores, etc. El analizador debe ser capaz de manejar el número infinito de posibles programas válidos que se le pueden presentar. La forma habitual de definir el idioma es Especifica una gramática.

Una gramática es un conjunto de reglas (o producciones) que especifica la sintaxis del idioma (es decir, qué es una oración válida en el idioma). Puede haber más de una gramática para un idioma dado. El analizador analiza el código fuente (token flujo) en contra de las reglas de producción para detectar cualquier error en el código. El resultado de esto La fase es un árbol de análisis.

La sección de declaraciones de un archivo yacc puede consistir en lo siguiente:

- **%token**:identifica los nombres de token que acepta el archivo yacc
- **%start**: identifica un nombre no terminal para el símbolo de inicio axioma
- **%right**:identifica tokens que son asociativos por la derecha con otros tokens [3]
- **%left**:identifica tokens que son asociativos a la izquierda con otros tokens
- **% nonassoc** :identifica tokens que no son asociativos con otros tokens
- **% union** :especifica la colección de distintos tipos de datos de la variable global yyval y los atributos \$1 \$2 .

La sección de reglas consiste en la gramática libre de contexto utilizada para generar el análisis árbol. Una regla general tiene la siguiente estructura:

```
no terminal
: sentencia form
| sentencia form
..... .
| sentencia form
;
```

Analisador semantico

El analizador semántico se construye agregando subrutinas y funciones C para reglas gramaticales definidas en la fase del analizador de sintaxis del compilador. El símbolo La tabla de la fase de sintaxis se actualiza en la fase semántica. La tabla de símbolos contiene las siguientes columnas:

- Serial No: representa el número de entradas en la tabla de símbolos
- Identificador: especifica el nombre de las variables que son identificadores
- Alcance: especifica el alcance de las variables
- Valor: representa el valor matemático de una variable si se inicializa / define
- Tipo: especifica el tipo de datos de la variable
- Dimensión: especifica la dimensión si el identificador es una matriz
- Tipo de parámetro: especifica los tipos de datos de argumentos / parámetros de función
- Lista de parámetros: especifica los nombres de los parámetros de función

4.1. Importante

Después del análisis, si hay errores, se muestran los números de línea de esos errores. junto con un mensaje de .eror de análisis.en el terminal. De lo contrario, un .ánalysis completo.El mensaje se muestra en la consola. Cuando una gramática no está cuidadosamente pensada, el analizador generado a partir de la gramática puede enfrentar dos tipos de dilemas.

4.2. Desplazamiento-Reducción Conflicto

Se han leído suficientes términos y se puede reconocer una regla gramatical de acuerdo con los términos acumulados En esta situación, el analizador puede hacer una reducción. Si, sin embargo, hay También es otra regla gramatical que exige que se acumulen más términos y el aspecto El token adelantado es justo lo que esperaba la segunda regla gramatical. En esta situación, el analizador También puede hacer una operación de turno. Este dilema que enfrenta el analizador se llama Desplazamiento / Reduce el conflicto.

Resolver Conflicto

El comando yacc está construido con dos reglas internas para resolver estas dos ambigüedades, a veces llamado las reglas de desambiguación.

1. Yacc resuelve el conflicto Descplazamiento / Reduce Conflicto a favor de la operación de turno. En llano Inglés, esto solo significa que coincide con la entrada más larga posible.
2. yacc resuelve el conflicto Reducccion / Reducccion a favor de la primera regla gramatical.

En general, siempre que sea posible aplicar reglas de desambiguación para producir una correcta analizador, también es posible reescribir las reglas gramaticales para que se lean las mismas entradas Pero no hay conflictos. Por esta razón, es importante escribir una gramática inequívoca sin conflictos

Las acciones semanticas se incluyen en la definicion de la gramatica donde se utiliza expresiones de tipo c para darle funcionalidad al lenguaje.

```
start : Funcion start
       | PREPROC start
       | Declaracion start
       |
       ;
Funcion : Tipo ID '(' ')' CompoundStmt {
           if ($1!=retornarfunc_tipo(ct))
           {
               printf("\nError : Tipos no coinciden : Line %d\n",printline());
               errc++;
           }
       }
```

Figura 5: Programas de prueba MINIC

La gramatcia para analisador semantico tiene una accion semantica guarda el resutado en Tipo de la funcion que verifican si los tipos son iguales

```

62             printf("\nError : No coinciden los tipos : Line %d\n",printline())
63         errc++;
64     }
65     if (!(strcmp($2,"printf") && strcmp($2,"scanf") && strcmp($2,"getc") &&
66           strcmp($2,"gets") && strcmp($2,"getchar") && strcmp
67           ($2,"puts") && strcmp($2,"putchar") && strcmp($2,"clearerr") && strcmp($2,"getw") &&
68           strcmp($2,"putw") && strcmp($2,"putc") && strcmp($2,"rewind") &&
69           strcmp($2,"sprint") && strcmp($2,"sscanf") && strcmp($2,"remove") &&
70           strcmp($2,"fflush")))
71         {printf("Error : Redeclaracion de %s : Linea %d\n",
72         $2,printline());errc++;}
73     else
74     {
75         insertar($2,FUNCION);
76         insertar($2,$1);
77         for(j=0;j<=k;j++)
78         {insertarp($2,plist[j]);}
79         k=-1;
80     }
81 }
82 ;

```

Figura 6: Programas de prueba MINIC

Se comparan las tokens con las declaraciones previas para ver si hay una redeclaracion.

```

405 struct Tsimbolo
406 {
407     int sno;
408     char token[100];
409     int tipo[100];
410     int paratipo[100];
411     int tn;
412     int pn;
413     float valorf;
414     int index;
415     int alcance;
416 }tSimbolo[100];
417 int n=0,arr[10];
418 int tnp;
419
420 int retornarfunc_tipo(int ct)
421 {
422     return arr[ct-1];
423 }
424 void almaretorono( int ct, int retornartipo )
425 {
426     arr[ct] = retornartipo;
427     return;
428 }

```

Figura 7: Programas de prueba MINIC

la estructura que albergara la tabla de simbolos En el analisador sintactico se

hace uso de una estructura tabla hash mientras que en el analisador semantico se utiliza un estructura de 9 campos para la tabla de simbolos. Se guarda el lexema atributos para determinar si un determinado valor es un entero un flotante etc.

```

455 int buscar(char *a)
456 {
457     int i;
458     for(i=0;i<n;i++)
459     {
460         if( !strcmp( a, tSimbolo[i].token) )
461             return 0;
462     }
463     return 1;
464 }
```

Figura 8: Programas de prueba MINIC

Se busca el token para luego ser insertado en la tabla

```

535 void insertar(char *name, int tipo)
536 {
537     int i;
538     if(buscar(name))
539     {
540         strcpy(tSimbolo[n].token,name);
541         tSimbolo[n].tn=1;
542         tSimbolo[n].pn=0;
543         tSimbolo[n].tipo[tSimbolo[n].tn-1]=tipo;
544         tSimbolo[n].sno=n+1;
545         n++;
546     }
547     else
548     {
549         for(i=0;i<n;i++)
550         {
551             if(!strcmp(name,tSimbolo[i].token))
552             {
553                 tSimbolo[i].tn++;
554                 tSimbolo[i].tipo[tSimbolo[i].tn-1]=tipo;
555                 break;
556             }
557         }
558     }
559     return;
560 }
```

Figura 9: Programas de prueba MINIC

Funcion insertar que recibe un puntero char que apunta al campo nombre para insertar en la tabla y un tipo del token.

```

602 void insertar_index(char *name,int ind)
603 {
604     int i;
605     for(i=0;i<n;i++)
606     {
607         if(!strcmp(name,tSimbolo[i].token) && tSimbolo[i].tipo[0]==273)
608         {
609             tSimbolo[i].index = atoi(ind);
610         }
611     }
612 }
```

Figura 10: Programas de prueba MINIC

Funcion insertar que recibe un puntero char que apunta al campo nombre para insertar en la tabla y un tipo del token.

La gramatica para el analisador ver [7.2.1](#)yyparse () lee una secuencia de pares de token / valor de yylex (), que debe suministrarlo. La funcion se puede definir .En los ejemplos, optado por dejar esta tarea a Lex.

cadaa llamada yylex devuelve un valor que representa un tipo de token y poner el valor en yyval.Esto le dice a yacc que tipo de token a leido por defecto yyval es de tipo int

5. Resultados

Para ejecutar

```

lex parser.l
yacc parser.y
gcc y.tab.c -ll -w
./a.out test1.c
```

archivo y.tab.h

```
33 #ifndef YY_Y_TAB_H_INCLUDED
34 # define YY_Y_TAB_H_INCLUDED
35 /* Debug traces. */
36 #ifndef YYDEBUG
37 # define YYDEBUG 0
38 #endif
39 #if YYDEBUG
40 extern int yydebug;
41#endif
42
43 /* Token type. */
44 #ifndef YYTOKENTYPE
45 # define YYTOKENTYPE
46 enum yytokentype
47 {
48     ID = 258,
49     IF = 259,
50     ELSE = 260,
51     NUM = 261,
52     REAL = 262,
53     WHILE = 263,
54     INT = 264,
55     FLOAT = 265,
56     VAR = 266,
57     FUNCION = 267,
58     NOT = 268,
59     IGUAL = 269,
60     NOIGUAL = 270,
61     MENORIGUAL = 271
62 };
63#endif
64 /* Tokens. */
65 #define ID 258
66 #define IF 259
67 #define ELSE 260
68 #define NUM 261
69 #define REAL 262
70 #define WHILE 263
71 #define INT 264
72 #define FLOAT 265
73 #define VAR 266
74 #define FUNCION 267
75 #define NOT 268
76 #define IGUAL 269
77 #define NOIGUAL 270
78 #define MENORIGUAL 271
79
80 /* Value type. */
81 #if ! defined YYSTYPE && ! defined YYSTYPE_IS_DEFINED
82 #define YYSTYPE int
83 # define YYSTYPE_IS_TRIVIAL 1
84 # define YYSTYPE_IS_DEFINED 1
85#endif
86
87
88 extern YYSTYPE yylval;
89
90 int yyparse (void);
```

Figura 11: Archivo y.tab.h

Las acciones que hace el Compilador para reconocer la gramatica ingresada sus respectivas representacion en codigo ASCII para luego ser interpretada.

The figure consists of three vertically stacked screenshots of a terminal window. Each screenshot shows the command `./a.out<test4.cpp` or `./a.out<test3.cpp` being run, followed by the program's output and a prompt.

Screenshot 1 (Top):

```

1 int main(int x){
2 float a;
3
4
5 }chao
                                         angel@CSangel: ~/Descargas/DESCARGASYCapturas
                                         Archivo Editar Ver Buscar Terminal Ayuda
error de sintaxis syntax error
angel@CSangel:~/Descargas/DESCARGASYCapturas$ ./a.out<test4.cpp
main 258 269 0
x 258 274 0
a 259 274 0
Programa en ejecucion:
angel@CSangel:~/Descargas/DESCARGASYCapturas$ 

```

Screenshot 2 (Middle):

```

1 int main(int x){
2 float a;
3 a=1.1;
4 while(a<8.1){
5     a=a+1.1;
6 }
7 }
8 return 0;
9
10 }chao
                                         angel@CSangel: ~/Descargas/DESCARGASYCapturas
                                         Archivo Editar Ver Buscar Terminal Ayuda
angel@CSangel:~/Descargas/DESCARGASYCapturas$ ./a.out<test4.cpp
main 258 269 0
x 258 274 0
a 259 274 0
1.1 265 265 1.1
8.1 265 265 8.1
_T0 274 0 0
_T1 274 0 0
0 264 264 0
0) MOVER a 1.1
1) MENOR _T0 = a < 8.1
2) SALTARF _T0 6
3) SUMAR _T1 = a + 1.1
4) MOVER a _T1
5) SALTAR 1
Programa en ejecucion:
angel@CSangel:~/Descargas/DESCARGASYCapturas$ 

```

Screenshot 3 (Bottom):

```

MINICa.y x test3.cpp x
                                         Archivo Editar Ver Buscar Terminal Ayuda
angel@CSangel:~/Descargas/DESCARGASYCapturas$ ./a.out<test3.cpp
1
2 int main(int x){
3 int a;
4 a=a+4;
5 return a;
6 }chao
                                         angel@CSangel:~/Descargas/DESCARGASYCapturas$ 
main 258 269 0
x 258 274 0
a 258 274 0
4 264 264 5.60519e-45
_T0 274 0 0
0) SUMAR _T0 = a + 4
1) MOVER a _T0
Programa en ejecucion:
angel@CSangel:~/Descargas/DESCARGASYCapturas$ 

```

Figura 12: Programas de prueba MINIC

```

1 int main(int x){
2 float a;
3 a=3.1;
4 if(a==3.1)
5     return 1;
6
7 }adios

```

```

Archivo Editar Ver Buscar Terminal Ayuda
angel@CSangel:~/Descargas/DESCARGASYCapturas$ ./a.out<prog2.cpp
main 258 269 0
x 258 274 0
a 259 274 0
3.1 265 265 3.1
1 264 264 1.4013e-45
0) MOVER a 3.1
Programa en ejecucion:
angel@CSangel:~/Descargas/DESCARGASYCapturas$ █

```

```

1 int main(int x){
2 float a;
3 a=3.1;
4 if(a==3.1){
5     return 1;
6 }else {
7     a=a+1.1;
8 }
9 }adios

```

```

Archivo Editar Ver Buscar Terminal Ayuda
angel@CSangel:~/Descargas/DESCARGASYCapturas$ ./a.out<prog2.cpp
main 258 269 0
x 258 274 0
a 259 274 0
3.1 265 265 3.1
1 264 264 1.4013e-45
1.1 265 265 1.1
_T0 274 0 0
0) MOVER a 3.1
1) SUMAR _T0 = a + 1.1
2) MOVER a _T0
Programa en ejecucion:
angel@CSangel:~/Descargas/DESCARGASYCapturas$ █

```

```

1 int main(int a)
2 {
3 int a;
4
5 for( a=0;a<4;a=a+1)
6     print(a);
7
8 }adios

```

```

Archivo Editar Ver Buscar Terminal Ayuda
angel@CSangel:~/Descargas/DESCARGASYCapturas$ ./a.out<test6.cpp
main 258 269 0
a 258 274 0
0 264 264 0
4 264 264 5.60519e-45
_T0 274 0 0
1 264 264 1.4013e-45
_T1 274 0 0
0) MOVER a 0
1) MENOR _T0 = a < 4
2) SUMAR _T1 = a + 1
3) MOVER a _T1
4) IMPRIMUN a
Programa en ejecucion:
0.00
angel@CSangel:~/Descargas/DESCARGASYCapturas$ █

```

programas que acepta el compilador

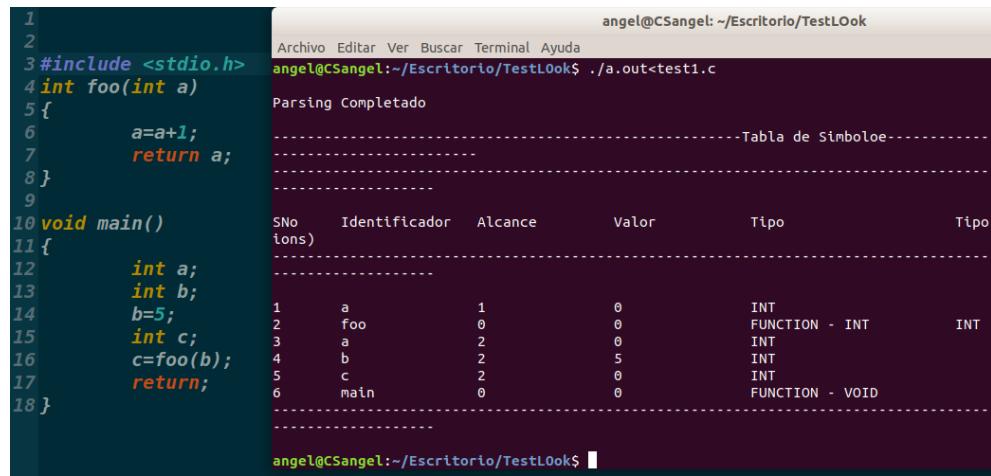
Como resultado de una abstraccion del lenguaje como proceso de reconocimiento del programa se obtiene como resultado una tabla de simbolos que como consecuencia genera un codigo para la etapa de ejecucion del codigo.

Analisis sintactico



```
test1.c
1 //without error - nested if-else
2 #include<stdio.h>
3 #define x 3
4 int main()
5 {
6     int a=4;
7     if(a<10)
8     {
9         a = a + 1;
10    }
11    else
12    {
13        a = a + 2;
14    }
15    return;
16 }
```

Figura 13: Programa de prueba test1.c



```
1
2
3 #include <stdio.h>
4 int foo(int a)
5 {
6     a=a+1;
7     return a;
8 }
9
10 void main()
11 {
12     int a;
13     int b;
14     b=5;
15     int c;
16     c=foo(b);
17     return;
18 }
```

angel@CSangel:~/Escritorio/TestLook\$./a.out<test1.c

Archivo Editar Ver Buscar Terminal Ayuda

angel@CSangel:~/Escritorio/TestLook\$./a.out<test1.c

Parsing Completado

-----Tabla de Simbolos-----

SNo	Identificador	Alcance	Valor	Tipo	Tipo
1	a	1	0	INT	
2	foo	0	0	FUNCTION - INT	INT
3	a	2	0	INT	
4	b	2	5	INT	
5	c	2	0	INT	
6	main	0	0	FUNCTION - VOID	

angel@CSangel:~/Escritorio/TestLook\$

Figura 14: terminal ubuntu analisis semanticoo

```
CSangel  rapidin (CSangel) ✘
angel@CSangel:~/eclipse-Renovate/rapidin$ lex parser.l
angel@CSangel:~/eclipse-Renovate/rapidin$ yacc parser.y
angel@CSangel:~/eclipse-Renovate/rapidin$ gcc y.tab.c -ll -w
angel@CSangel:~/eclipse-Renovate/rapidin$ ./a.out test1.c

Parsing completo

-----Tabla de Simbolo-----
-----Tabla Constante-----

Token | Token Tipo
-----|-----
a      | INT
main   | procedimiento
-----|-----
Valor | Dato Tipo
-----|-----
1      | INT
5      | INT
1      | INT
2      | INT

angel@CSangel:~/eclipse-Renovate/rapidin$
```

Figura 15: terminal Eclipse

```
angel@CSangel:~/eclipse-Renovate/rapidin$ ./a.out
int main(int x){
float a,b,c,d;

Linea 2 : syntax error ,
Variable no declarada d : Linea 2
Variable no declarada c : Linea 2
Variable no declarada b : Linea 2
Error : Variable no declaradab : Linea 2
a=1.1;

Error : Variable no declaradaa : Linea 3
b=2.2;

Error : Variable no declaradab : Linea 4
c=3.3;

Error : Variable no declaradac : Linea 5
a=a+5.0; print(a);

Error : Variable no declaradaaa : Linea 6
Vaariable no Declarada a : Line 6
NO declarado Funcion print : Linea 6

Error : Variable no declaradaprint : Linea 6
b=a-b; print(b);
c
Error : Tipos no Coincidan : Linea 7
Error : Variable no declaradab : Linea 7
Vaariable no Declarada b : Line 7
NO declarado Funcion print : Linea 7

Error : Variable no declaradaprint : Linea 7
=c*2.0; print(c);
d
Error : Variable no declaradac : Linea 8
Vaariable no Declarada c : Line 8
NO declarado Funcion print : Linea 8

Error : Variable no declaradaprint : Linea 8
=a/2.0; print(d);

Error : Variable no declaradad : Linea 9
Vaariable no Declarada d : Line 9
NO declarado Funcion print : Linea 9

Error : Variable no declaradaprint : Linea 9
}chao

Error : No coinciden los tipos : Line 10
```

Figura 16: El resultado de yacc y lex de programa de prueba de MINICa

En la imagen anterior el compilador nos avisa cuando hay un error y nos muestra la linea donde se produjo el error

```
2 #include<stdio.h>
3
4 int foo(int a)
5 {
6     a=a+1;
7     return a;
8 }
9
10 void main()
11 {
12     int a,b,c;
13     int b;
14     b=5;
15     int c;
16     c=foo(b);
17     return;
18 }
```

Figura 17: programa de prueba para el anterior test1.c

Analisador semantico

Después del análisis, si hay errores, se muestran los números de línea de esos errores. junto con un .eror de análisis.en el terminal. De lo contrario, un .análisis completo.^{El} mensaje se muestra en la consola. La tabla de símbolos con almacenado y actualizado los valores siempre se muestran, independientemente de los errores.

5.0.1. Archivos generados en yacc

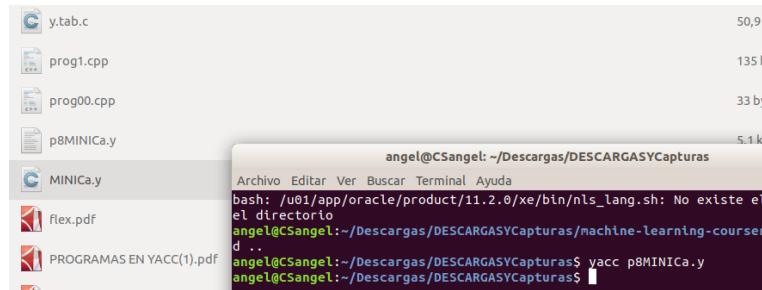


Figura 18: Programa de prueba test1.c

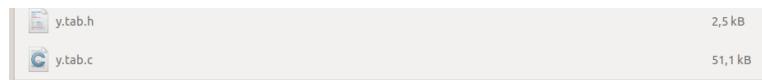


Figura 19: Programa de prueba test1.c

Para usar Lex con Bison/YACC tanto en Windows como en Linux, uno especifica la opción ‘-d’ de yacc para instruirle a que genere el fichero ’y.tab.h’ que contiene las definiciones de todos los ’en la entrada de Bison. Entonces este archivo se incluye en el analizador de Flex.

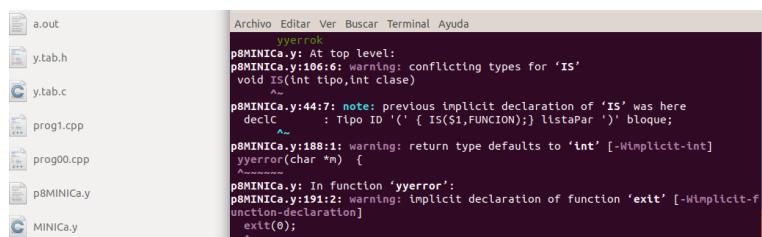


Figura 20: Programa de prueba test1.c

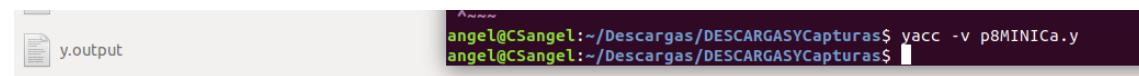


Figura 21: Archivo de la tabla de análisis Sintáctico

The figure shows a file explorer on the left and a code editor on the right. The file explorer lists several files and folders:

- John Carey, Shreyans Doshi, F
- Lecture slides
- machine-learning-coursera-m
- a.out
- MINICa.y
- p8MINICa.y
- prog00.cpp** (highlighted)
- prog1.cpp
- y.output
- y.tab.c
- y.tab.h

The code editor displays the contents of the `prog00.cpp` file:

```
1 int main(int a)
2 {
3     int b;
4 }
5 chao
```

Figura 22: Programa de prueba prog00.cpp

The figure shows a file explorer on the left and a code editor on the right. The file explorer lists several files and folders:

- ▶ John Carey, Shreyans Doshi, F
- ▶ Lecture slides
- ▶ machine-learning-coursera-m
- ▶ a.out
- ▶ MINICa.y
- ▶ p8MINICa.y
- ▶ prog00.cpp
- prog1.cpp** (highlighted)
- ▶ y.output
- ▶ y.tab.c
- ▶ y.tab.h

The code editor displays the contents of the `prog1.cpp` file:

```
1 int main(int x){
2     float a,b,c,d;
3     a=1.1;
4     b=2.2;
5     c=3.3;
6     a=a+5.0; print(a);
7     b=a-b;    print(b);
8     c=c*2.0;  print(c);
9     d=a/2.0;  print(d);
10 }chao
```

Figura 23: Programa de prueba prog1.cpp

5.0.2. Parte de la tabla generada de MINICa.y

```
John Carey, Shreyans Doshi, F
Lecture slides
machine-learning-coursera-m
a.out
MINICa.y
p8MINICa.y
prog0.cpp
prog1.cpp
y.output
y.tab.c
y.tab.h

1 Terminales no usados en la gramática
2
3 FORMATO
4 DO
5 FUNCION
6 LEENUM
7 CENTERO
8 CFLOAT
9 VAR
10
11
12 Estado 92 conflictos: 1 desplazamiento/reducción
13 Estado 95 conflictos: 13 desplazamiento/reducción
14
15
16 Gramática
17
18 0 $accept: programaC $end
19
20 1 programaC: listaDeclC
21
22 2 listaDeclC: listaDeclC declC
23 3 | %empty
24
25 4 declC: Tipo listaVar ';'
26
27 5 @{$1: %empty
28
29 6 declC: Tipo ID '(' @{$1 listaPar ')' bloque
30
31 7 Tipo: INT
32 8 | FLOAT
33
34 9 @{$2: %empty
```

Figura 24: Resultado de la tabla de simbolos

```

93 $end (0) 0
94 '(' (40) 6 22 23 26 27 28
95 ')' (41) 6 22 23 26 27 28
96 '*' (42) 41
97 '+' (43) 39
98 ',' (44) 10 13
99 '-' (45) 40
100 '/' (47) 42
101 ':' (58) 43
102 ';' (59) 4 17 20 27
103 '<' (60) 35
104 '=' (61) 48 50
105 '>' (62) 36
106 '?' (63) 43
107 '[' (91) 49 50
108 ']' (93) 49 50
109 '{' (123) 15
110 '}' (125) 15
111 error (256)
112 INT (258) 7
113 FLOAT (259) 8
114 FORMATO (260)
115 ID (261) 6 10 11 13 14 44 48 49 50
116 IF (262) 22 23
117 ELSE (263) 23
118 NUM (264) 45
119 REAL (265) 46

191 Estado 1
192
193     0 $accept: programaC . $end
194
195     $end desplazar e ir al estado 3
196
197
198 Estado 2
199
200     1 programaC: listaDeclC .
201     2 listaDeclC: listaDeclC . declC
202
203     INT desplazar e ir al estado 4
204     FLOAT desplazar e ir al estado 5
205
206     $default reduce usando la regla 1 (programaC)
207
208     declC ir al estado 6
209     Tipo ir al estado 7
210
211
212 Estado 3
213
214     0 $accept: programaC $end .
215
216     $default aceptar      29
217
218
219 Estado 4
220
221     7 Tipo: INT .
222

```

The screenshot shows a terminal window titled "Terminal" with the command "dom 20:52 ●". The window displays the following text:

```
Actividades Terminal ●
Abrir ▾ Guardar
parser.y × semantica.y × minic.y × programa2.cpp × tabla.y × prog00.cpp × prog1.cpp ×
prog00.cpp
~/Descargas/DESCARGAS Y Capturas
Angel@CSangel: ~/Descargas/DESCARGAS Y Capturas
Archivo Editar Ver Buscar Terminal Ayuda
yyerror(char *m) {
^~~~~~
p8MINICA.y: In function 'yyerror':
p8MINICA.y:191:2: warning: implicit declaration of function 'exit' [-Wimplicit-function-declaration]
    exit(0);
^~~~~~
p8MINICA.y:191:2: warning: incompatible implicit declaration of built-in function 'exit'
p8MINICA.y:191:2: note: include <stdlib.h> or provide a declaration of 'exit'
p8MINICA.y: At top level:
p8MINICA.y:194:1: warning: return type defaults to 'int' [-Wimplicit-int]
main() {
^~~~~~
angel@CSangel:~/Descargas/DESCARGAS Y Capturas$ ./a.out prog00.cpp
int main(int a)
{
    int b;
}
chao
        main 264 267 0
        a 264 266 0
        b 264 266 0
angel@CSangel:~/Descargas/DESCARGAS Y Capturas$
```

The terminal shows the output of the program, which includes several warnings about implicit declarations and function definitions. The final output shows the variable values: main is 264, 267, 0; a is 264, 266, 0; and b is 264, 266, 0.

Figura 26: El resultado

```

1718    '<'           desplazar e ir al estado 59
1719    '>'           desplazar e ir al estado 60
1720    '+'           desplazar e ir al estado 61
1721    '-'           desplazar e ir al estado 62
1722    '*'           desplazar e ir al estado 63
1723    '/'           desplazar e ir al estado 64
1724    ')'           desplazar e ir al estado 103
1725
1726
1727 Estado 103
1728
1729 27 prop: FOR '(' expr ';' expr ';' expr ')' . prop
1730
1731  ID      desplazar e ir al estado 31
1732  IF      desplazar e ir al estado 32
1733  NUM     desplazar e ir al estado 33
1734  REAL    desplazar e ir al estado 34
1735  WHILE   desplazar e ir al estado 35
1736  FOR     desplazar e ir al estado 36
1737  IMPRINUM desplazar e ir al estado 37
1738  NOT     desplazar e ir al estado 38
1739  ';'     desplazar e ir al estado 39
1740  '{'     desplazar e ir al estado 22
1741
1742  bloque  ir al estado 41
1743  prop    ir al estado 104
1744  expr    ir al estado 43
1745
1746
1747 Estado 104
1748
1749 27 prop: FOR '(' expr ';' expr ';' expr ')' prop .
1750
1751 $default reduce usando la regla 27 (prop)

```

Figura 25: Parte final de minica.y

```

Actividades Terminal *
Abrir parser.y x semantica.y x
dom 20:55 ● angel@CSangel: ~/Descargas/DESCARGAS Y Capturas
Archivo Editar Ver Buscar Terminal Ayuda
angel@CSangel:~/Descargas/DESCARGAS Y Capturas$ ./a.out prog1.cpp
int main(int x){
float a,b,c,d;
a=1.1;
b=2.2;
c=3.3;
a=a+5.0; print(a);
b=a-b; print(b);
c=c*2.0; print(c);
d=a/2.0; print(d);
}chao
main 258 269 0
x 258 274 0
a 259 274 0
b 259 274 0
c 259 274 0
d 259 274 0
1.1 265 265 1.1
2.2 265 265 2.2
3.3 265 265 3.3
5.0 265 265 5
T0 274 0 0
_T1 274 0 0
2.0 265 265 2
_T2 274 0 0
_T3 274 0 0
0) MOVER a 1.1
1) MOVER b 2.2
2) MOVER c 3.3
3) SUMAR _T0 = a + 5.0
4) MOVER a _T0
5) IMPRINUM a
6) RESTAR _T1 = a - b
7) MOVER b _T1
8) IMPRINUM b
9) MULTIPLICAR _T2 = c * 2.0
10) MOVER c _T2
11) IMPRINUM c
12) DIVIDIR _T3 = a / 2.0
13) MOVER d _T3
14) IMPRINUM d
Programa en ejecución:
6.10
3.90
6.60
3.05
angel@CSangel:~/Descargas/DESCARGAS Y Capturas$ 

```

Figura 27: El resultado de MINICa.y

Se obtiene como resultado una tabla de simbolos que como consecuencia genera un codigo para la etapa de ejecucion del programa las caracteres se obtienen evalundolos en su codigo ASCII para ser conocidos en la maquina. Hay tokens que no tiene que ser definidos porque ya se encuentran en los 256 caracteres ASCII. Puede adecuarse este codigo para los procesadores que tienen un tipo lenguaje tecnico

```
#include <stdio.h>
int main()
{
    int a=4;
    int b=9;
    a=10;
    return 0;
}

^C
angel@CSangel:~/eclipse-Renovate/rapidin$ ./a.out<test2.cpp

Parsing Completado
-----Tabla de Simbolos-----
-----SNo Identificador Alcance      Valor      Tipo-----ions)-----1      a          1          10        INT2      b          1          9         INT3      main       0          0         FUNCTION - INT-----angel@CSangel:~/eclipse-Renovate/rapidin$
```

Figura 28: El resultado del parsing completado

6. Conclusiones

Analisador sintactico

El analizador léxico y el analizador de sintaxis para un subconjunto de lenguaje C, que incluyen declaraciones de selección, declaraciones compuestas, declaraciones de iteración, declaraciones de salto, Se generan funciones definidas por el usuario y expresiones primarias. Es importante observar que Pueden producirse conflictos (desplazamiento-reducción y reducción-reducción) en caso de analizador de sintaxis si no se toma el cuidado adecuado al especificar la gramática libre de contexto para el idioma. Nosotros siempre debe especificar una gramática inequívoca para que el analizador funcione correctamente.

El analizador léxico, el analizador de sintaxis y el analizador semántico para un subconjunto de C lenguaje, que incluye declaraciones de selección, declaraciones com-

puestas, iteración Se generan declaraciones (para, mientras y do-while) y funciones definidas por el usuario. Es importante definir una gramática inequívoca en la fase de análisis de sintaxis. El analizador semántico realiza una verificación de tipo, informa varios errores, como variable no declarada, tipo no coincidente, errores en la llamada a la función (número y tipos de datos de los parámetros no coinciden) y errores en la indexación de la matriz.

Referencias

- [1] <https://github.com/Shivananda199/Mini-C-Compiler>.
- [2] <http://epaperpress.com/lexandyacc/download/LexAndYaccTutorial.pdf>.
[Lex and Yacc Tutorial by Tom Nieman].
- [3] <https://www.programiz.com/c-programming/precedence-associativity-operators>.
- [4] Ravi Sethi Jeffrey D. Ullman Alfred V. Aho Monica S. Lam. *Compiler Design Semantic Analysis*. Pearson, 2007.