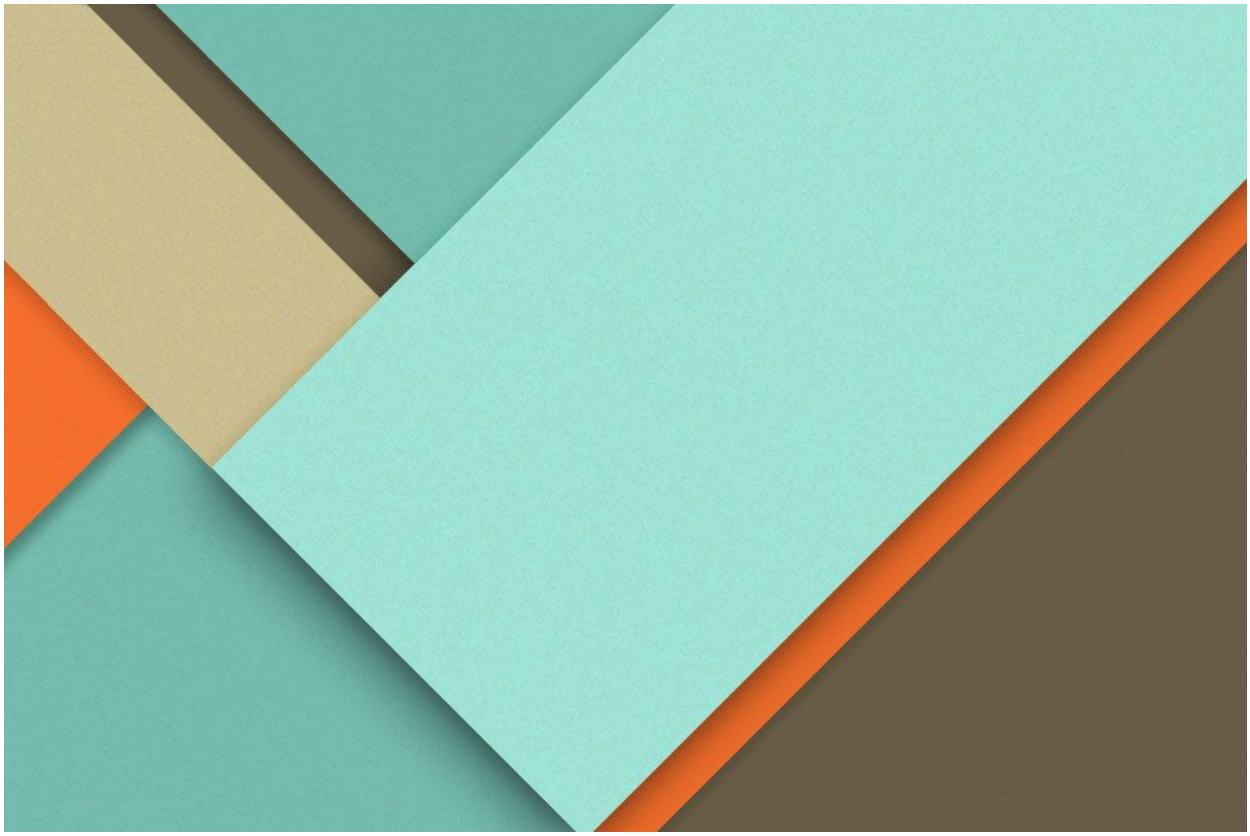


Compiler Design Project

Report-2

20.02.2018

Siddharth V, 15CO150

Akash Rao, 15CO202

Introduction

Parsing, **syntax analysis** or **syntactic analysis** is the process of analysing a string of symbols, either in natural language, computer languages or data structures, conforming to the rules of a formal grammar. It is the second phase in the process of building a compiler(immediately after the lexical analysis stage)

This project uses Yacc to build the parser with inputs from the Lex file created in the previous . Yacc is a Look Ahead Left-to-Right (LALR) parser generator, generating a parser, the part of a compiler that tries to make syntactic sense of the source code, specifically a LALR parser, based on an analytic grammar written in a notation similar to Backus–Naur Form (BNF). Yacc is supplied as a standard utility on BSD and AT&T Unix. GNU-based Linux distributions include Bison, a forward-compatible Yacc replacement.

The input to Yacc is a grammar with snippets of C code (called "actions") attached to its rules. Its output is a shift-reduce parser in C that executes the C snippets associated with each rule as soon as the rule is recognized. Typical actions involve the construction of parse trees. Using an example from Johnson, if the call `node(label, left, right)` constructs a binary parse tree node with the specified `label` and children, then the rule recognizes summation expressions and constructs nodes for them. The special identifiers `$$`, `$1` and `$3` refer to items on the parser's stack.

Yacc produces only a parser (phrase analyzer); for full syntactic analysis this requires an external lexical analyzer to perform the first tokenization stage (word analysis), which is then followed by the parsing stage proper. Lexical analyzer generators, such as Lex or Flex are widely available. The IEEE POSIX P1003.2 standard defines the functionality and requirements for both Lex and Yacc.

Goals

1. Successful parsing of C-code and inclusion of all language constructs mentioned in the abstract.
2. Expanding the symbol table attributes count to include the datatype.

Contents

Source Code

Lex code

```
%{
    #include "y.tab.h"

    #include <stdio.h>
    #include <stdlib.h>

%}



%%

"//".*\n                    { /*printf("Single line comment\n");*/ yylineno++;}
"/*"([^*]|\*+[^*/])*"*/"  { /*printf("Multi line comment\n");*/ }

\"(\\.|[^"\n\\])*\"                             {/*printf("String const:\t\t%s\n",
yytext); insert(yytext, 1, CONST_TBL);*/ strcpy(yylval.str,yytext); return STRCONST; }
0[xX][0-9a-fA-F]+                               {/*printf("Int const:\t\t%s\n", yytext);
insert(yytext, 2, CONST_TBL);*/ strcpy(yylval.str,yytext); return INTCONST; }
[0-9]+                                          {/*printf("Int const:\t\t%s\n", yytext);
insert(yytext, 2, CONST_TBL);*/ strcpy(yylval.str,yytext); return INTCONST; }
(([0-9]+)|([0-9]*\.[0-9]+)([eE][-+]?[0-9]+)?)     {/*printf("Float const:\t\t%s\n",
yytext); insert(yytext, 3, CONST_TBL);*/  strcpy(yylval.str,yytext); return FLTCONST; }

'([^'\\\n]|\\.)'                                {/*printf("Char const:\t\t%s\n", yytext);
insert(yytext, 4, CONST_TBL);*/  strcpy(yylval.str,yytext); return CHARCONST; }
'                                               {/*printf("\n Error: Unterminated
Character constant\n\n");*/ }
''                                              {/*printf("Char const:\t\t%s\n", yytext);
insert(yytext, 1, CONST_TBL);*/ }
'([^'\\\n]|\\.)+'                               {/*printf("\n Error: Character constant
too long\n\n");*/ }

"auto"              { /*printf("Keyword:\t\t%s\n", yytext);*/ return AUTO; }
"break"             { /*printf("Keyword:\t\t%s\n", yytext);*/ return BREAK; }
"case"              { /*printf("Keyword:\t\t%s\n", yytext);*/ return CASE; }
"char"              { /*printf("Keyword:\t\t%s\n", yytext);*/ strcpy(yylval.str,yytext);
return CHAR; }
"const"             { /*printf("Keyword:\t\t%s\n", yytext);*/ return CONST; }
"continue"          { /*printf("Keyword:\t\t%s\n", yytext);*/ return CONTINUE; }
"default"           { /*printf("Keyword:\t\t%s\n", yytext);*/ return DEFAULT; }
```

```
"do"                    { /*printf("Keyword:\t\t%s\n", yytext);*/ return DO; }
"double"                { /*printf("Keyword:\t\t%s\n", yytext);*/ strcpy(yylval.str,yytext);
return DOUBLE; }
"else"                  { /*printf("Keyword:\t\t%s\n", yytext);*/ return ELSE; }
"enum"                  { /*printf("Keyword:\t\t%s\n", yytext);*/ return ENUM; }
"extern"                { /*printf("Keyword:\t\t%s\n", yytext);*/ return EXTERN; }
"float"                 { /*printf("Keyword:\t\t%s\n", yytext);*/ strcpy(yylval.str,yytext);
return FLOAT; }
"for"                   { /*printf("Keyword:\t\t%s\n", yytext);*/ return FOR; }
"if"                    { /*printf("Keyword:\t\t%s\n", yytext);*/ return IF; }
"int"                   { /*printf("Keyword:\t\t%s\n", yytext);*/ strcpy(yylval.str,yytext);
return INT; }
"long"                  { /*printf("Keyword:\t\t%s\n", yytext);*/ strcpy(yylval.str,yytext);
return LONG; }
"register"              { /*printf("Keyword:\t\t%s\n", yytext);*/ return REGISTER; }
"return"                { /*printf("Keyword:\t\t%s\n", yytext);*/ return RETURN; }
"short"                 { /*printf("Keyword:\t\t%s\n", yytext);*/ strcpy(yylval.str,yytext);
return SHORT; }
"signed"                { /*printf("Keyword:\t\t%s\n", yytext);*/ return SIGNED; }
"sizeof"                { /*printf("Keyword:\t\t%s\n", yytext);*/ return SIZEOF; }
"static"                { /*printf("Keyword:\t\t%s\n", yytext);*/ return STATIC; }
"struct"                { /*printf("Keyword:\t\t%s\n", yytext);*/ return STRUCT; }
"switch"                { /*printf("Keyword:\t\t%s\n", yytext);*/ return SWITCH; }
"typedef"               { /*printf("Keyword:\t\t%s\n", yytext);*/ return TYPEDEF; }
"union"                 { /*printf("Keyword:\t\t%s\n", yytext);*/ return UNION; }
"unsigned"              { /*printf("Keyword:\t\t%s\n", yytext);*/ return UNSIGNED; }
"void"                  { /*printf("Keyword:\t\t%s\n", yytext);*/ strcpy(yylval.str,yytext);
return VOID; }
"volatile"              { /*printf("Keyword:\t\t%s\n", yytext);*/ return VOLATILE; }
"while"                 { /*printf("Keyword:\t\t%s\n", yytext);*/ return WHILE; }


[a-zA-Z_][a-zA-Z0-9_]*   { /*printf("Identifier:\t\t%s\n", yytext);*/  /*insert(yytext, 0,
SYM_TBL);*/ strcpy(yylval.str,yytext); return IDENTIFIER; }

"*/"                    { /*printf("\n Error: Unexpected end of comment\n\n");*/ }
"/*"                    { /*printf("\n Error: Unterminated Multi line comment\n\n");*/ }

"++"                    { /*printf("Operator:\t\t%s\n", yytext);*/ return INCREMENT; }
"--"                    { /*printf("Operator:\t\t%s\n", yytext);*/ return DECREMENT; }
"&&"                    { /*printf("Operator:\t\t%s\n", yytext);*/ return AND; }
"||"                    { /*printf("Operator:\t\t%s\n", yytext);*/ return OR; }
"=="                    { /*printf("Operator:\t\t%s\n", yytext);*/ return EQUAL; }
">="                    { /*printf("Operator:\t\t%s\n", yytext);*/ return GREATER; }
"<="                    { /*printf("Operator:\t\t%s\n", yytext);*/ return LESSER; }
"!="                    { /*printf("Operator:\t\t%s\n", yytext);*/ return NOTEQUAL; }

"!"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '!'; }
"%"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '%'; }
"^"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '^'; }
```

```
"&"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '&'; }
"*"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '*'; }
"("                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '('; }
")"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return ')'; }
"-"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '-'; }
"+"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '+'; }
"="                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '='; }
"{"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '{'; }
"}"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '}'; }
"|"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '|'; }
"~"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '~'; }
"["                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '['; }
"]"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return ']'; }
";"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return ';'; }

":"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return ':'; }
"<"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '<'; }
">"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '>'; }
"?"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '?'; }
","                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return ','; }
"."                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '.'; }
"/"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '/'; }
"#"                     { /*printf("Punctuator:\t\t%s\n", yytext);*/ return '#'; }

"\n"                    { /*printf("\n");*/ yylineno++; }
" "                     { /*printf(" ");*/ }
"\t"                    { /*printf("\t");*/ }

"\""                    { /*printf("\n Error: Unmatched quotation\n\n");*/ }
.                       { /*printf("\n Error: Invalid token \n\n");*/ }

%%

int yywrap()
{
    return 1;
}
```

Yacc code

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>

    #define SYM_TBL 0
    #define CONST_TBL 1

    typedef struct node
    {
        char* name;
        char* type;

        struct node* next;
    }node;

    node* sym_tbl[100];
    node* const_tbl[100];

    int hash(char* x, int M)
    {
       int i, sum;
       for (sum=0, i=0; i < strlen(x); i++)
       sum += x[i];
       return sum % M;
    }

    int lookup(char* x, int table)
    {
        int idx = hash(x, 100);

        node* t = NULL;

        if(table==0)
        {
               if(sym_tbl[idx]==NULL)
                      return 0;

               t = sym_tbl[idx];
         }
         else
         {
               if(const_tbl[idx]==NULL)
               return 0;
```

```
		t = const_tbl[idx];
	}

	while(t!=NULL)
	{
		if(strcmp(t->name, x)==0)
			return 1;
		t = t->next;
	}

	return 0;

}


void insert(char* x, char* type, int table)
{
	if(lookup(x, table))
		return;

	int idx = hash(x, 100);

	node* cell = (node*)malloc(sizeof(node));
	cell->name = (char*)malloc(strlen(x));
	cell->type = (char*)malloc(strlen(type));
	strcpy(cell->name, x);
	strcpy(cell->type, type);
	cell->next = NULL;

	node* t = NULL;

	if(table==0)
	{
		if(sym_tbl[idx]==NULL)
		{
			sym_tbl[idx] = cell;
			return;
		}

		t = sym_tbl[idx];
	}
	else
	{
		if(const_tbl[idx]==NULL)
		{
			const_tbl[idx] = cell;
			return;
		}

		t = const_tbl[idx];
```

```
         }

        while(t->next!=NULL)
        t = t->next;

        t->next = cell;
    }

    void display()
    {
        printf("\n-----------------------------\n\tSymbol
table\n-----------------------------\n");
        printf("Value\t\t-\tType\n-----------------------------\n");

        int i;

        for(i=0; i<100; i++)
        {
                if(sym_tbl[i]==NULL)
                        continue;

                node* t = sym_tbl[i];

                while(t!=NULL)
                {
                        printf("%s\t\t-\t%s\n", t->name, t->type);
                        t = t->next;
                }

        }


        printf("\n\n-----------------------------\n\tConstant
table\n-----------------------------\n");
        printf("Value\t\t-\tType\n-----------------------------\n");

        for(i=0; i<100; i++)
        {
                if(const_tbl[i]==NULL)
                        continue;

                node* t = const_tbl[i];

                while(t!=NULL)
                {
                        printf("%s\t\t-\t%s\n", t->name, t->type);
                        t = t->next;
                }

        }
```

```
    }

    extern FILE *yyin;

    int yylineno;

    char* yytext;

    void yyerror(char*);
    int error = 0;

    char datatype_str[100];
    char vars[100][1000];
    int pnt[100]={0};
    int varpt = 0;
%}

%union{
        char str[1000];
}

%right '='
%left OR
%left AND
%left '|'
%left '^'
%left '&'
%left EQUAL NOTEQUAL
%left '<' '>' LESSER GREATER
%left '+' '-'
%left '*' '/'
%left '$'
%right ELSE


%token STRCONST INTCONST FLTCONST CHARCONST
%token AUTO BREAK CASE CHAR CONST CONTINUE DEFAULT DO DOUBLE ELSE ENUM EXTERN FLOAT FOR IF
INT LONG REGISTER RETURN SHORT SIGNED SIZEOF STATIC STRUCT SWITCH TYPEDEF UNION UNSIGNED
VOID VOLATILE WHILE
%token IDENTIFIER
%token INCREMENT DECREMENT AND OR EQUAL GREATER LESSER NOTEQUAL


%%
S
    :func_def S
    |id_dec ';' S
    |id_assign_dec';' S
    |multidec';' S
    |
```

```
    ;
func_def
    :modifiers datatype IDENTIFIER '(' params_list')' '{'statement_list'}' { insert
($<str>3, $<str>2, 0);}
    |modifiers datatype IDENTIFIER '('')' '{'statement_list'}' { insert ($<str>3, $<str>2,
0);}
    |modifiers datatype IDENTIFIER '(' params_list')' ';' { insert ($<str>3, $<str>2, 0);}
    |modifiers datatype IDENTIFIER '('')' ';' { insert ($<str>3, $<str>2, 0);}
    |modifiers datatype'*' IDENTIFIER '(' params_list')' '{'statement_list'}' {char
temp[1000];strcpy(temp,$<str>2);strcat(temp,"*"); insert ($<str>4,temp, 0);}
    |modifiers datatype'*' IDENTIFIER '('')' '{'statement_list'}' {char
temp[1000];strcpy(temp,$<str>2);strcat(temp,"*"); insert ($<str>4,temp, 0);}
    |modifiers datatype'*' IDENTIFIER '(' params_list')' ';' {char
temp[1000];strcpy(temp,$<str>2);strcat(temp,"*"); insert ($<str>4,temp, 0);}
    |modifiers datatype'*' IDENTIFIER '('')' ';' {char
temp[1000];strcpy(temp,$<str>2);strcat(temp,"*"); insert ($<str>4,temp, 0);}
    ;
id_dec
    :modifiers datatype IDENTIFIER { insert ($<str>3, $<str>2, 0);}
    |modifiers datatype IDENTIFIER '['INTCONST']' { insert ($<str>3, $<str>2, 0);}
    |modifiers datatype IDENTIFIER ['']' { insert ($<str>3, $<str>2, 0);}
    |modifiers datatype '*' IDENTIFIER {char
temp[1000];strcpy(temp,$<str>2);strcat(temp,"*"); insert ($<str>4,temp, 0);}
    ;
id_assign_dec
    :modifiers datatype IDENTIFIER '=' expression { insert ($<str>3, $<str>2, 0);}
    |modifiers datatype IDENTIFIER[']' '=' '{'const_list'}' {char
temp[1000];strcpy(temp,$<str>2);strcat(temp,"*"); insert ($<str>3,temp, 0);}
    |modifiers datatype IDENTIFIER'['INTCONST']' '=' '{'const_list'}' {char
temp[1000];strcpy(temp,$<str>2);strcat(temp,"*"); insert ($<str>3,temp, 0);}
    |modifiers datatype '*' IDENTIFIER '=' expression {char temp[1000];
strcpy(temp,$<str>2); strcat(temp,"*"); insert ($<str>4,temp, 0);}
    ;
multidec
    :modifiers datatype id_chain
    {
        varpt--;
        while(varpt>=0)
        {
                if(pnt[varpt])
                {
                        char temp[1000];
                        strcpy(temp,$<str>2);
                        strcat(temp,"*");
                        pnt[varpt] = 0;
                        insert (vars[varpt--], $<str>2, 0);
                }
                else
                        insert (vars[varpt--], $<str>2, 0);
        }
```

```
    }
    ;
datatype
    :INT
    |FLOAT
    |CHAR
    |DOUBLE
    |VOID
    |LONG
    |SHORT
    ;

modifiers
    :AUTO
    |CONST
    |EXTERN
    |REGISTER
    |SIGNED
    |UNSIGNED
    |VOLATILE
    |
    ;

params_list
    :id_dec
    |id_dec ',' params_list
    ;

brackets
    :'('expression')'
    ;
expression
    :constant
    |IDENTIFIER bin_op expression
    |constant bin_op expression
    |IDENTIFIER
    |un_op IDENTIFIER
    |un_op constant
    |IDENTIFIER INCREMENT
    |IDENTIFIER DECREMENT
    |func_call bin_op expression
    |func_call
    |un_op func_call
    |brackets
    |un_op brackets
    |brackets bin_op expression
    ;
const_list
    :constant
    |constant ',' const_list
```

```
    ;
statement_list
    :statement
    |statement statement_list
    ;
constant
    :INTCONST { insert ($<str>$, "int", 1);}
    |STRCONST { insert ($<str>$, "string", 1);}
    |FLTCONST { insert ($<str>$, "float", 1);}
    |CHARCONST { insert ($<str>$, "char", 1);}
    ;
bin_op
    :'+'
    |'-'
    |'*'
    |'/'
    |AND
    |OR
    |EQUAL
    |GREATER
    |LESSER
    |NOTEQUAL
    |'%'
    |'^'
    |'|'
    |'>'
    |'<'
    ;
un_op
    :'!' %prec '$'
    |'+' %prec '$'
    |'-' %prec '$'
    |'*' %prec '$'
    |'&' %prec '$'
    |INCREMENT %prec '$'
    |DECREMENT %prec '$'
    ;
func_call
    :IDENTIFIER '('expression_list')'
    |IDENTIFIER '('')'
    ;
expression_list
    :expression
    |expression ',' expression_list
    ;
statement
    :id_dec';'
    |id_assign_dec';'
    |multidec ';'
    |conditional
```

```
    |iterative
    |assignment
    |if_block
    |RETURN expression ';'
    |func_call';'
    |IDENTIFIER INCREMENT ';'
    |IDENTIFIER DECREMENT ';'
    |'{'statement_list'}'
    |';'
    ;
id_chain
    :IDENTIFIER { strcpy(vars[varpt++],$<str>1); }
    |IDENTIFIER '=' expression { strcpy(vars[varpt++],$<str>1); }
    |'*'IDENTIFIER { strcpy(vars[varpt],$<str>1); pnt[varpt] = 1; varpt++;}
    |'*'IDENTIFIER ',' id_chain { strcpy(vars[varpt],$<str>1); pnt[varpt] = 1; varpt++;}
    |'*'IDENTIFIER '=' expression { strcpy(vars[varpt],$<str>1); pnt[varpt] = 1; varpt++;}
    |'*'IDENTIFIER '=' expression ',' id_chain { strcpy(vars[varpt],$<str>1); pnt[varpt] =
1; varpt++;}
    |IDENTIFIER '=' expression ',' id_chain { strcpy(vars[varpt++],$<str>1); }
    |IDENTIFIER ',' id_chain { strcpy(vars[varpt++],$<str>1); }
    ;
if_block
    :IF '('expression')' statement
    |IF '('expression')' '{'statement_list'}'
    ;
conditional
    :if_block
    |if_block ELSE statement
    |if_block ELSE '{'statement_list'}'
    ;
iterative
    :WHILE'('expression')' statement
    |WHILE '('expression')''{'statement_list'}'
    ;
assignment
    :IDENTIFIER '=' expression ';'
    |'*'IDENTIFIER '=' expression';'
    |IDENTIFIER'['INTCONST']' '=' expression';'
    ;


%%


void yyerror(char* s)
{
    error = 1;
    //printf("ERROR: %s\n", s);
    fprintf(stderr, "\nLINE %d: %s \nERROR: %s\n", yylineno, s, yytext);
    //exit(0);
```

```
}

int main()
{
    //yyin = fopen("test_cases/yacc/8.c", "r");
    yyin = fopen("test_cases/program.c", "r");

    yyparse();

    if(error)
        printf("\nUNSUCCESSFUL\n");
    else
        printf("\nSUCCESS!\n");

    display();

    return 0;
}
```

Implementation

- ## Lex code

  The lex code has been significantly changed. The print statements when each lexeme was encountered was removed(As now each token need not be explicitly identified when its encountered). Instead a return statement is put along with each token, which returns the token ID for each lexeme. The main function was removed as a new main has been written in the yacc file.

  "Y.tab.h" has been included in the lex file. This is a header file containing the function/variable declaration in the yacc file. This is necessary because the lex file is dependant on the yacc for the token IDs and various other components.

- ## Yacc code

  This is the code for the parser. The file has .y extension indicating its a yacc file. The symbol table, previously on lex file, has been shifted here, as parser helps in detecting identifier data type(return type in case of function names). Since we would want the values read from the source code in string form(i.e obtain "int" and not the token ID for it), we will be using the %union structure.

  The file also implemented associativity of various operators(logical operators, binary operators etc.) by using %left and %right constructs. The precedence of the operators is also implemented by the order in which the associativity was listed(precedence increases as we go down the list).

  The production rules were written with minimized shift-reduce/reduce-reduce conflicts. Symbol table operations were associated with production rules taking care of declarations. "Yyerror" displays the line number and string where the error was encountered.

- ## Symbol table

  The symbol table was shifted from lex file to yacc file. The cell in symbol table now increased to include the datatype attribute of the identifiers. The data type is identified in the yacc program and inserted into the symbol table along with the identifier name. In case of functions declaration, the return type is inserted as the datatype of the identifier. Multiple declaration is taken care of. Identifiers only inserted while declaration. Constant table operations remains unchanged.

Test Case 1: Missing semicolon

```
void main()
{
    int a = 5
}
```

```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/cd_project/C_mini_com
piler$ make
yacc -d basic.y -Wnone
lex scanner.l
cc -w y.tab.c lex.yy.c -ll -w
./a.out

LINE 7: syntax error
ERROR: }

UNSUCCESSFUL

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
a               -       int


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
5               -       int
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/cd_project/C_mini_com
piler$ s
```

Test

Test Case 2: Balancing parentheses and blocks

```c
void main()
{
    int a = 1, b = 2, c = 3;

    //Balaced paranthesis
    int d = (a + (b+c));

    //Unbalanced paranthesis
    int e = (a + (b+c);


    {
        ;//Balanced block
    }

    {{
        ;//Unbalanced block
    }

}
```

```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/cd_project/C_mini_com
piler$ make
yacc -d basic.y -Wnone
lex scanner.l
cc -w y.tab.c lex.yy.c -ll -w
./a.out

LINE 12: syntax error
ERROR: ;

UNSUCCESSFUL

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
d               -       int
a               -       int
b               -       int
c               -       int


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
1               -       int
2               -       int
3               -       int
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/cd_project/C_mini_com
piler$ s
```

Test Case 3: Incorrect Function call

```
int fun(
{
    return 5;
}

void main()
{
    int a = fun(5);

}
```

```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/cd_project/C_mini_com
piler$ make
yacc -d basic.y -Wnone
lex scanner.l
cc -w y.tab.c lex.yy.c -ll -w
./a.out

LINE 4: syntax error
ERROR: {

UNSUCCESSFUL

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/cd_project/C_mini_com
piler$
```

Test Case 4: Incorrect float value(This issue was not being taken care in lex)

```
void main()
{
    float a = 11.22.33;

}
```

```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/cd_project/C_mini_com
piler$ make
yacc -d basic.y -Wnone
lex scanner.l
cc -w y.tab.c lex.yy.c -ll -w
./a.out

LINE 6: syntax error
ERROR: .33

UNSUCCESSFUL

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
a               -       float


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
11.22           -       float
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/cd_project/C_mini_com
piler$
```

Test Case 5: Full-fledged C-program

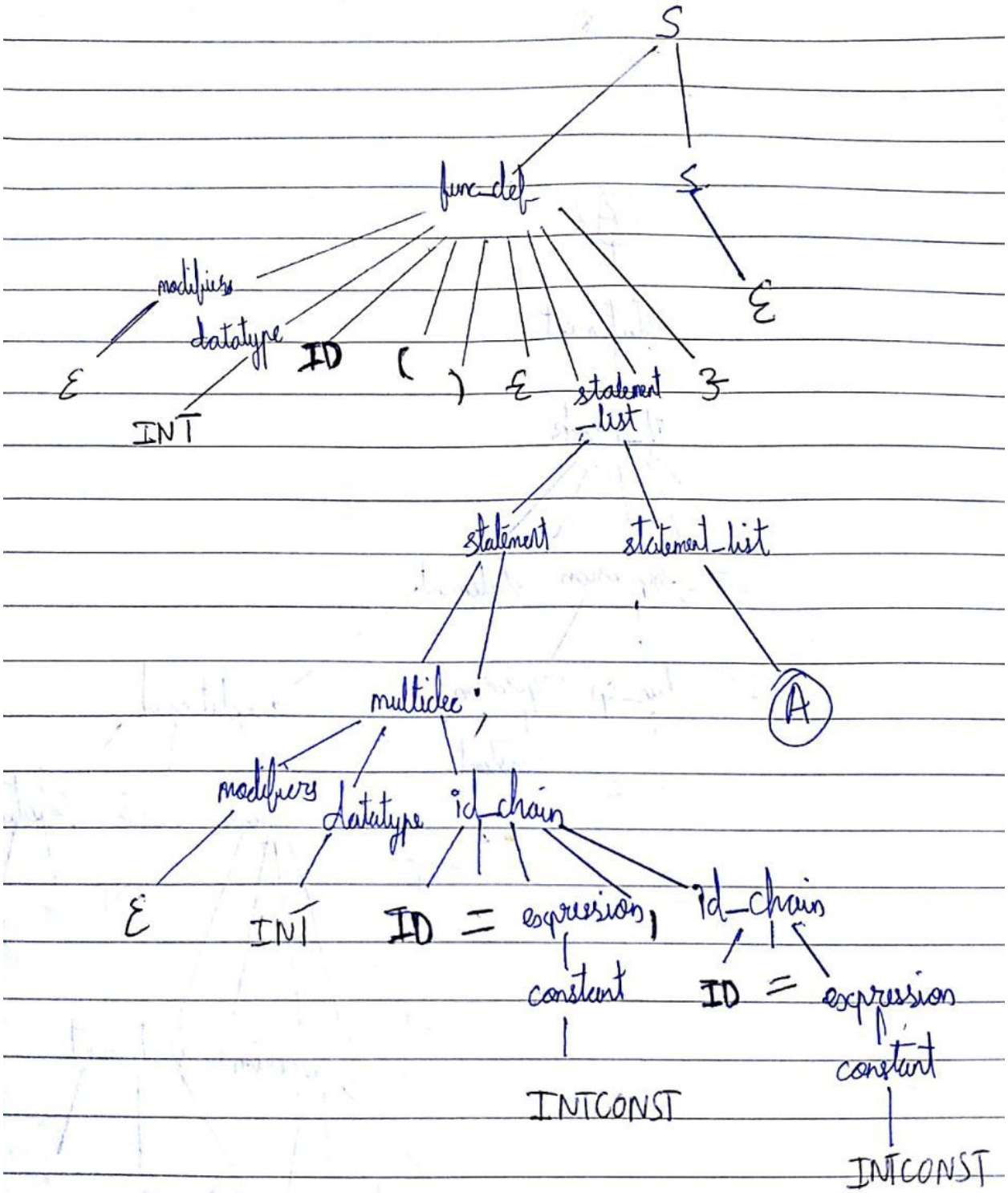
```
float compound_interest(float p, float r, float n)
{
	float interest;
	interest = p*((1 + r/100)^n - 1);
	return interest;
}

int main()
{
	float principal, rate_of_interest, time_period, result;
	printf("\nEnter The Principal Amount:\t");
	scanf("%f", &principal);
	printf("\nEnter The Interest Rate:\t");
	scanf("%f", &rate_of_interest);
	printf("\nEnter The Time Period in Years:\t");
	scanf("%f", &time_period);
	result = compound_interest(principal, rate_of_interest, time_period);
	printf("\nCompound Interest:\t%f\n", result);
	return 0;
}
```

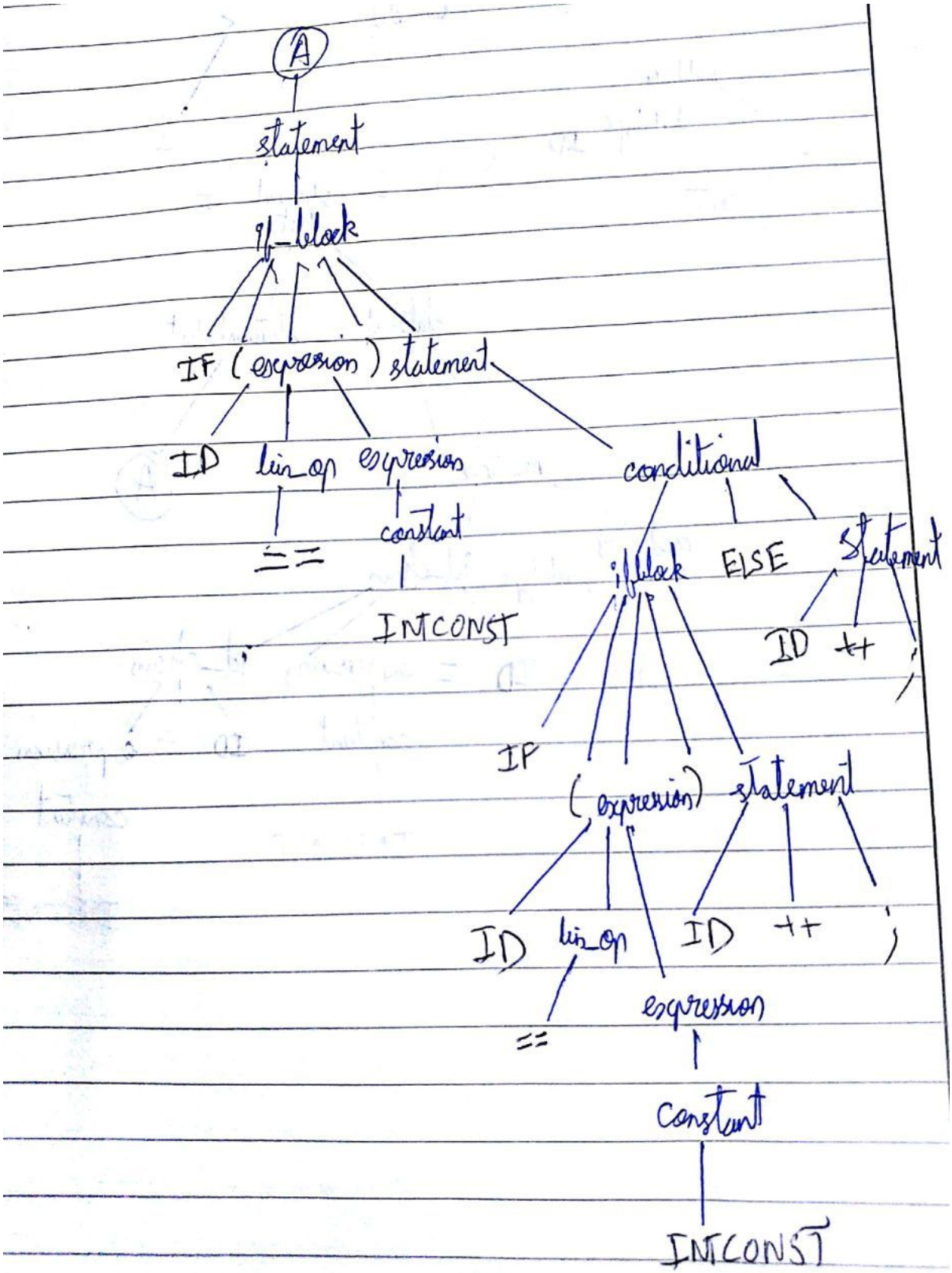
```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/cd_project/C_mini_com
piler$ make
yacc -d basic.y -Wnone
lex scanner.l
cc -w y.tab.c lex.yy.c -ll -w
./a.out

SUCCESS!

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
rate_of_interest                -       float
n               -       float
p               -       float
r               -       float
main            -       int
compound_interest               -       float
principal               -       float
time_period             -       float
result          -       float
interest                -       float


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
"%f"            -       string
"\nEnter The Time Period in Years:\t"          -       string
100             -       int
0               -       int
1               -       int
"\nEnter The Interest Rate:\t"          -       string
"\nEnter The Principal Amount:\t"               -       string
"\nCompound Interest:\t%f\n"            -       string
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/cd_project/C_mini_com
piler$
```

Test Case 6: Balanced if-else

```
int main()
{
    int a = 5,b = 4;
    if(a == 4)
        if(b == 3)
                a++;
    else
        c++;
}
```

```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/cd_project/C_mini_com
piler$ make
yacc -d basic.y -Wnone
lex scanner.l
cc -w y.tab.c lex.yy.c -ll -w
./a.out

SUCCESS!

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
main            -       int
a               -       int
b               -       int


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
3               -       int
4               -       int
5               -       int
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/cd_project/C_mini_com
piler$
```


S
func_def
S
modifiers
datatype
ID
( )
ε
statement_list
ε
ε
INT
statement
statement_list
multidec
A
modifiers
datatype
id_chain
ε
INT
ID = expression
id_chain
constant
ID = expression
INTCONST
constant
INTCONST


A
statement
if_block
IF (expression) statement
ID
bin_op
expression
==
constant
INTCONST
conditional
if_block
ELSE
statement
ID
++
;
IF
(expression)
statement
ID
bin_op
ID
++
;
==
expression
constant
INTCONST

Test Case 7: Dangling else problem

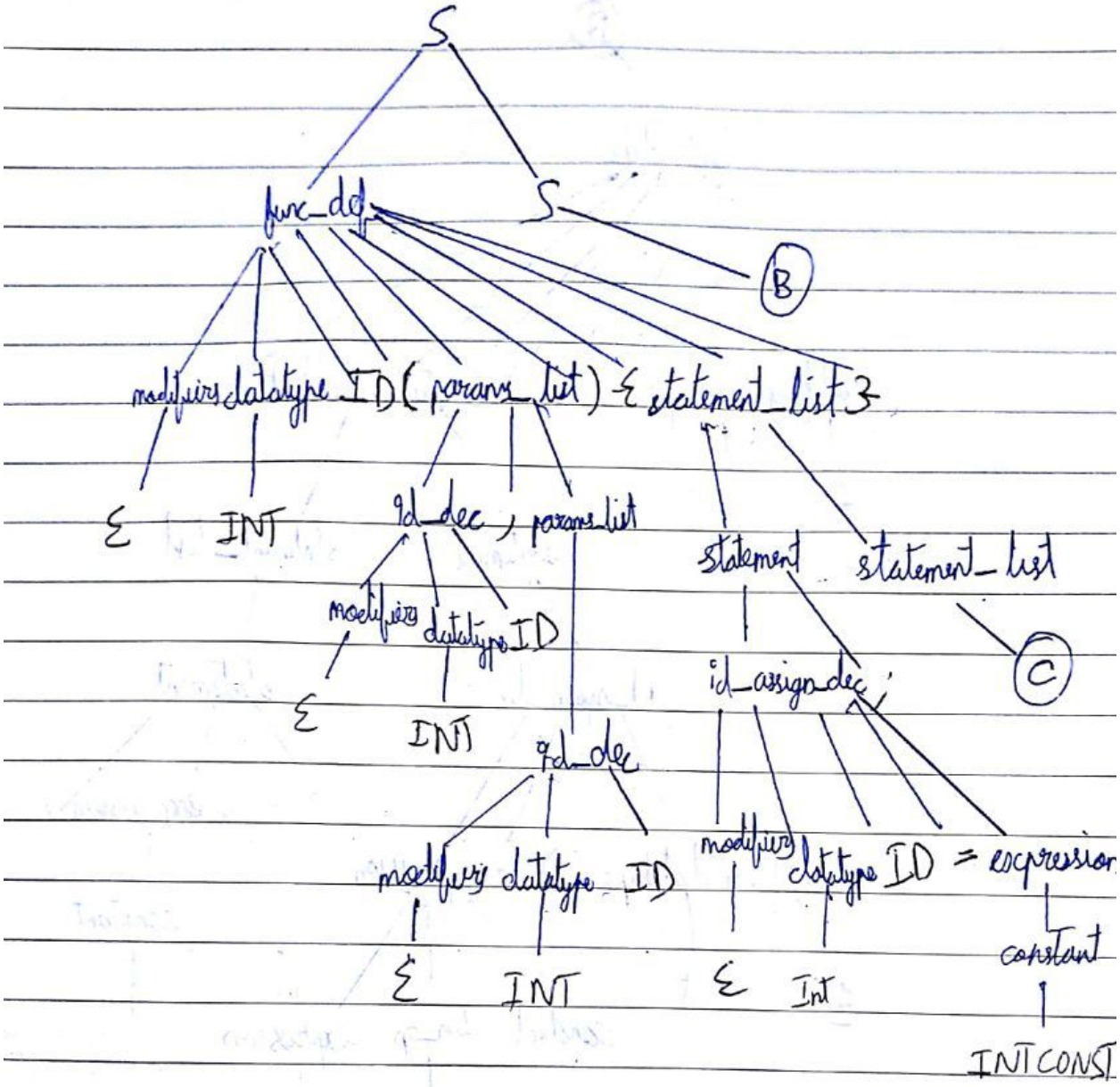
```
int main()
{
    int a = 5,b = 4;
    if(a == 4)
        else
                a++;
    else
        c++;
}
```

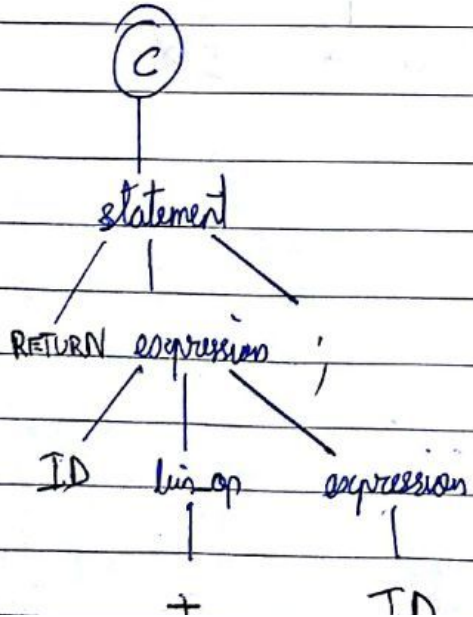
```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/cd_project/C_mini_com
piler$ make
yacc -d basic.y -Wnone
lex scanner.l
cc -w y.tab.c lex.yy.c -ll -w
./a.out

LINE 5: syntax error
ERROR: else

UNSUCCESSFUL

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
a               -       int
b               -       int


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
4               -       int
5               -       int
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/cd_project/C_mini_com
piler$ s
```

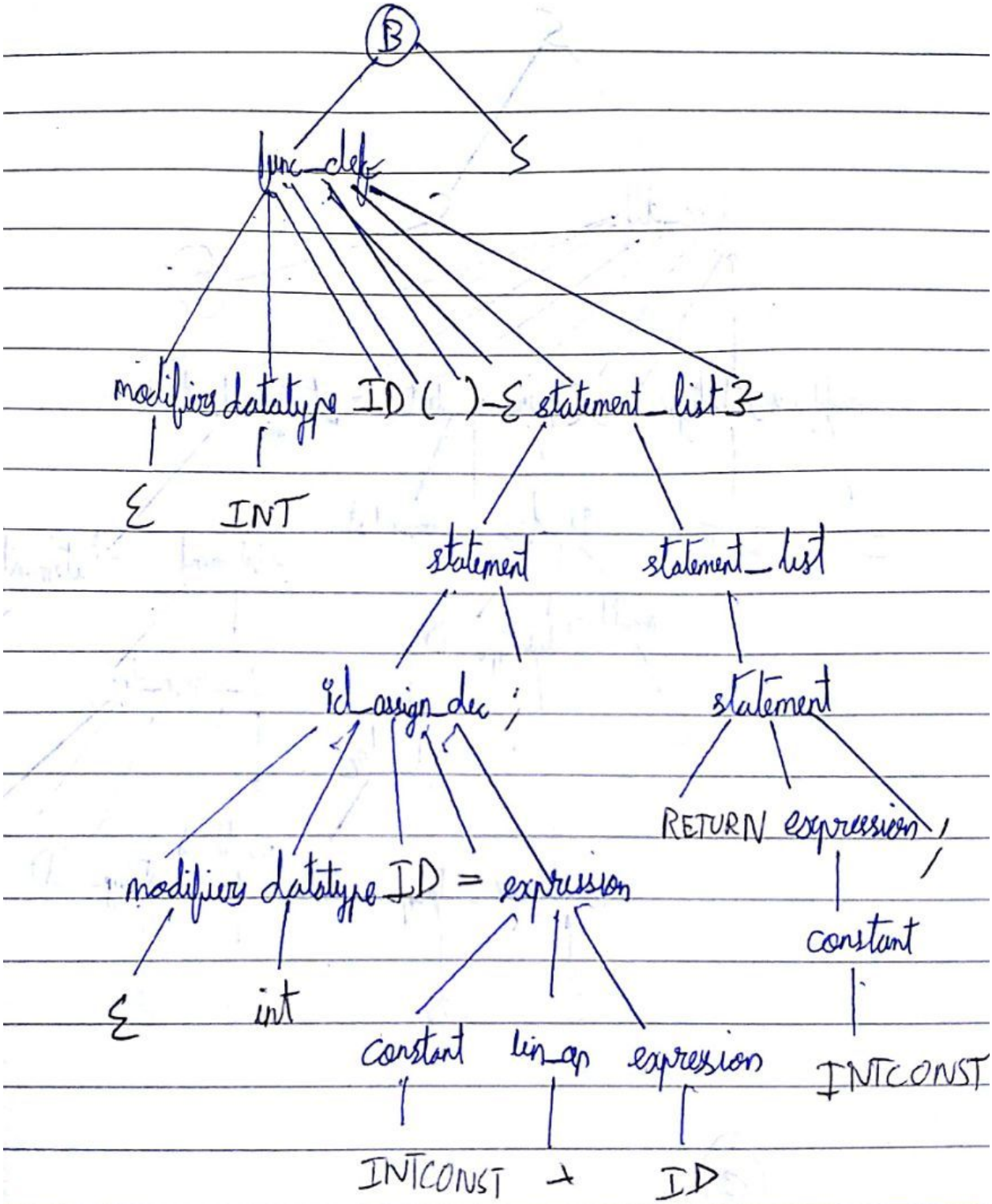
Test Case 8: Correct function call

```c
int func2(int a,int b)
{
    int c = 5;
    return  a+b;
}
int main()
{
    int a = 4+c;
    return 0;
}
```

```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/cd_project/C_mini_com
piler$ make
yacc -d basic.y -Wnone
lex scanner.l
cc -w y.tab.c lex.yy.c -ll -w
./a.out

SUCCESS!

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
main            -       int
func2           -       int
a               -       int
b               -       int
c               -       int


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
0               -       int
4               -       int
5               -       int
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/cd_project/C_mini_com
piler$
```


S
func_dcl
S
B
modifiers datatype ID (params_list) { statement_list }
ε
INT
id_dec , params_list
statement
statement_list
modifiers datatype ID
ε
INT
C
id_assign_dec ;
id_dec
modifiers datatype ID
ε
INT
modifiers datatype ID = expression
ε
Int
constant
INTCONST


C
statement
RETURN expression ;
ID
bin_op
expression
+
ID


B
func_def
modifiers datatype ID ( ) - { statement_list }
ε
INT
statement
statement_list
id_assign_dec ;
statement
modifiers datatype ID = expression
RETURN expression ;
ε
int
constant
constant
lin_op
expression
INTCONST
INTCONST
+
ID

Conclusion

At the end of the second phase of the Compiler design project, we are able to successfully parse through C programs which do not have any syntactic errors. The program was also able to identify and parse through all the C-language constructs mentioned in the abstract, that is:

- While loops
- Arrays
- Pointers
- Basic data types(int, float,char) with relevant modifiers

The symbol table contained identifier name and data type(return type in case of functions). Future works that can be done is that function parameters can be included in the symbol table as an attribute.