Compiler Design

Project Report - 2

Syntax analyser

By:

15CO248 - Jeshventh Raja T K

15CO229 - Zoeb Mithaiwala

CONTENTS

Introduction

What is a compiler?

A compiler is a special program that processes statements written in a high level programming language and turns them into machine language.

The input given to the compiler is a source code written in a high level language. The compiler outputs an equivalent and correct target code in machine level language. The compiler checks and outputs errors, if present. The compiler can check for syntax and semantic errors only.

There are two phases in compiling:

1) Analysis phase (Front end)
2) Synthesis phase (Back end)

Analysis phase

This phase consists of four steps:

1) Lexical analysis - Read and convert characters into words. Done by lexical/linear analyser or scanner
2) Syntax analysis - Convert the words into sentences. Done by syntax analyser or parser
3) Semantic analysis - Understand the meaning of sentences.
4) Intermediate code generation - Convert the understanding into intermediate code

Synthesis phase

This phase consists of two steps:

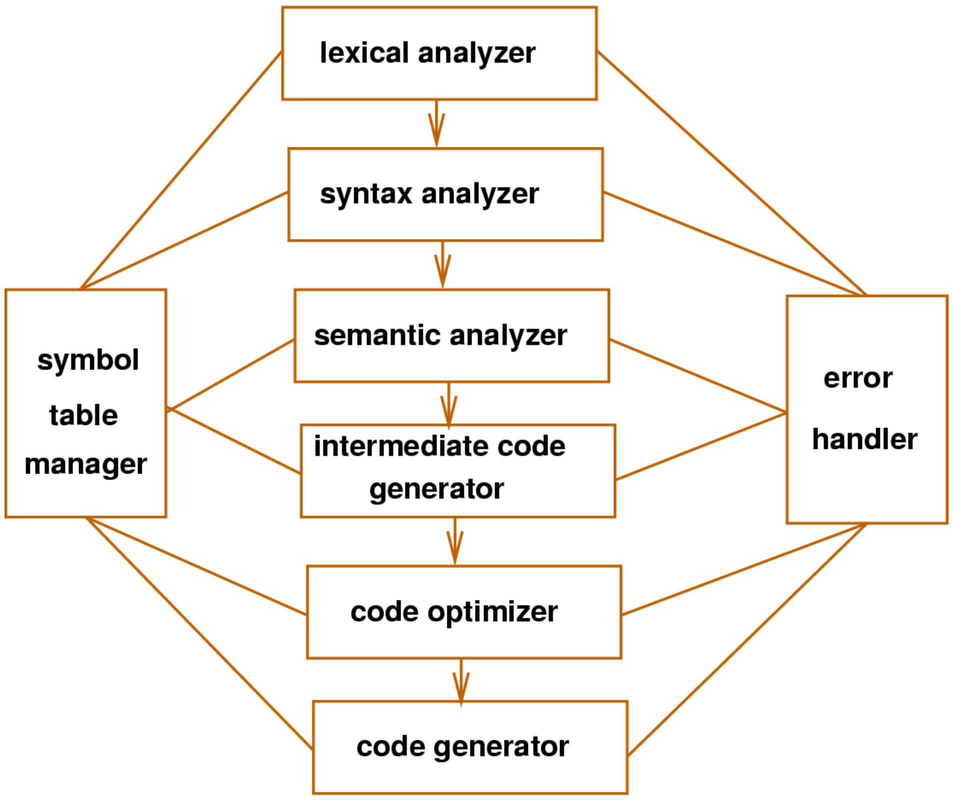
1) Code optimisation - Remove any unwanted or redundant code
2) Code generation - Convert to machine level language

There are various types of compilers:

1) Retargetable compiler - Converts H/L language to M/L language that is understood by a different machine
2) Single pass compiler - It scans an instruction, converts it to machine level code and then goes to the next instruction
3) Multipass compiler - It processes the source code or abstract syntax tree of a program several times, generating an output file after each pass

Additional functionalities of the compiler can be:

1) Error handling
2) Fast compilation
3) Efficient machine level code
4) Fast executing machine level output



Syntax analyser

Syntax analysis is the second phase in a compiler. It converts a sequence of token into sentences. The output of this phase is a parse tree.

The given stream of tokens are tried to match to a given set of rules called **grammar**. If matched, then the input is successfully parsed. Else syntax errors exist.

The grammar used is **Context free grammar.** It has four components:

1) Non-terminals
2) Terminals
3) Start symbol
4) Productions (Grammar rules)

The lexical analyser is also called as parser.

The functions of a syntax analyser are:

1) Check if the input given adheres to the language specified, that is, check for syntax errors. This is done by matching the input tokens to the grammar rules.
2) Make a parse tree of the input to be used in further stages of compiler.
3) Determine how to deal with conflicts occurring due to the grammar rules.

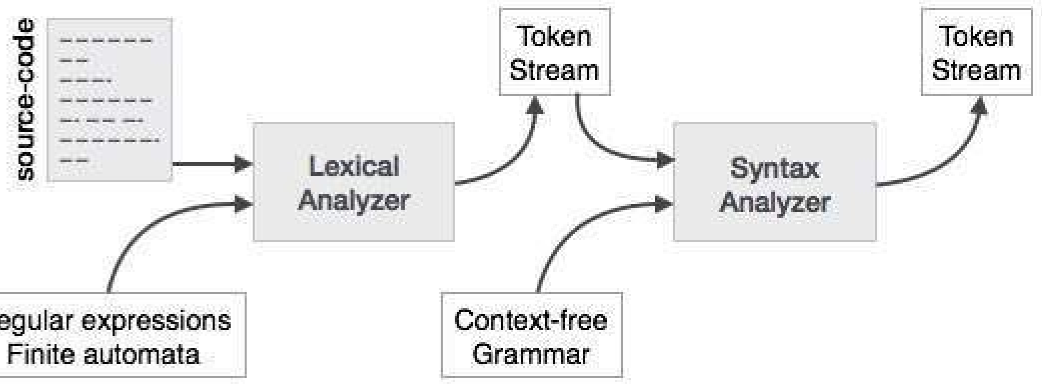
Syntax errors

1) Syntax error occurs when the given input cannot to reduced and matched to given rules of the grammar

Conflicts

1) Shift reduce conflict - This occurs when the parsed input can be directly reduced at that instant or it can be shifted(parse more tokens) and then reduced
2) Reduce reduce conflict - This occurs when the parsed input can be reduced by two different rules

The parser should either change the rules to avoid these conflicts or use a method to choose what to do when a conflict occurs..



Symbol table after syntax analysis

The symbol table consists of three columns after the lexical analysis phase.

1) Index
2) Lexeme
3) Token type

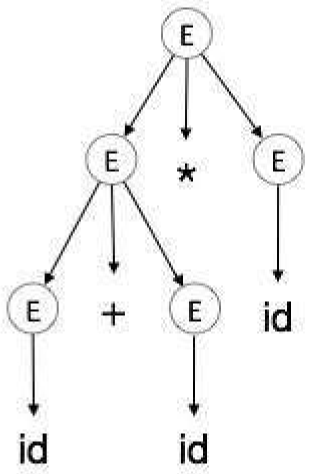
It is made using a hash table. A hash is calculated on the lexeme and chaining is used to deal with collisions.

The symbol table is filled by the parser. The tokens are inserted into the table when the sequence of tokens are reduced to basic tokens.

Parse tree

A parse tree is a graphical depiction of a derivation. It is convenient to see how strings are derived from the start symbol. The start symbol of the derivation becomes the root of the parse tree.

Parse tree for E -> id + id * id



Code - LEX

```
%{
      #include <stdio.h>
      #include <stdlib.h>
      #include <string.h>
      #include "y.tab.h"
      int lineNo=1;
%}

keyword
auto|break|case|char|const|continue|default|do|double|else|enum|extern|floa
t|for|goto|if|int|long|register|return|short|signed|sizeof|static|struct|sw
itch|typedef|union|unsigned|void|volatile|while
singleLineComment \/\/.*
multilineComment "/*"([^*]|\*+[^*/])*\*+"/"
multilineCommentError "/*"([^*]|\*+[^*/])*
str \"[^\"]*\"
strError \"[^\"]*
letter [a-zA-Z]
digit [0-9]
nonIdentifier ({digit})+(_|{letter})(_|{letter}|{digit})*
identifier (_|{letter})(_|{letter}|{digit})*
dataType int|float|double|short\ int|long\ int
argument {dataType}[\ ]*{identifier}
integer ({digit})+
float ({digit})*\.({digit})+
assignmentOperator "="
arithmeticOperator "+"|"-"|"*"|"/"|"%"
operator
"+="|"-="|"*="|"/="|"%="|">>="|"<<="|"&="|"^="|"|="|"++"|"--"|"=="|"!="|">"
|"<"|">="|"<="|"||"|"&&"|"!"|"&"|"|"|"^"|"~"|"<<"|">>"|"?"
headerFile "#include"[\ ]*"<"[\ ]*[^>]*[\ ]*">"
specialCharacters \;|\{|\}|\(|\)|\[|\]|,
whitespaces [\ ]|[\t]
%%
{multilineCommentError} {printf("ERROR: Multiline comment is not
closed\n");}
{strError} {printf("ERROR: String is not closed\n");}
{singleLineComment} {}
{multilineComment} {}
```

```
{headerFile} {printf("Header file found\n"); return HEADERFILE;}
{str} {
            char *te = (char *)malloc(strlen(yytext)-1);
            int l = strlen(yytext);
            strncpy(te, yytext + 1, l-2);
            printf("%s- String constant\n", te);  yylval.sym=te; return
STR;}
auto {printf("Keyword auto found\n");  yylval.sym="auto"; return AUTO;}
break {printf("Keyword break found\n");  yylval.sym="break"; return BREAK;}
case {printf("Keyword case found\n");  yylval.sym="case"; return CASE;}
char {printf("Keyword char found\n");  yylval.sym="keyword"; return CHAR;}
const {printf("Keyword const found\n");  yylval.sym="const"; return CONST;}
continue {printf("Keyword continue found\n");  yylval.sym="continue";
yylval.sym=""; return CONTINUE;}
default {printf("Keyword default found\n");  yylval.sym="default"; return
DEFAULT;}
do {printf("Keyword do found\n");  yylval.sym="do"; return DO;}
double {printf("Keyword double found\n");  yylval.sym="double"; return
DOUBLE;}
else {printf("Keyword else found\n");  yylval.sym="else"; return ELSE;}
enum {printf("Keyword enum found\n");  yylval.sym="enum"; return ENUM;}
extern {printf("Keyword extern found\n");  yylval.sym="extern"; return
EXTERN;}
float {printf("Keyword float found\n");  yylval.sym="float"; return FLOAT;}
for {printf("Keyword for found\n");  yylval.sym="for"; return FOR;}
goto {printf("Keyword goto found\n");  yylval.sym="goto"; return GOTO;}
if {printf("Keyword if found\n");  yylval.sym="if"; return IF;}
int {printf("Keyword int found\n"); yylval.sym="int"; return INT;}
long {printf("Keyword long found\n");  yylval.sym="long"; return LONG;}
register {printf("Keyword register found\n");  yylval.sym="register";
return REGISTER;}
return {printf("Keyword return found\n");  yylval.sym="return"; return
RETURN;}
short {printf("Keyword short found\n");  yylval.sym="short"; return SHORT;}
signed {printf("Keyword signed found\n");  yylval.sym="signed"; return
SIGNED;}
sizeof {printf("Keyword sizeof found\n");  yylval.sym="sizeof"; return
SIZEOF;}
static {printf("Keyword static found\n");  yylval.sym="static"; return
STATIC;}
struct {printf("Keyword struct found\n");  yylval.sym="struct"; return
STRUCT;}
```

```
switch {printf("Keyword switch found\n");  yylval.sym="switch"; return
SWITCH;}
typedef {printf("Keyword typedef found\n");  yylval.sym="typedef"; return
TYPEDEF;}
union {printf("Keyword union found\n");  yylval.sym="union"; return UNION;}
unsigned {printf("Keyword unsigned found\n");  yylval.sym="unsigned";
return UNSIGNED;}
void {printf("Keyword void found\n");  yylval.sym="void"; return VOID;}
volatile {printf("Keyword volatile found\n");  yylval.sym=yytext; return
VOLATILE;}
while {printf("Keyword while found\n");  yylval.sym="while"; return WHILE;}
{identifier} {printf("%s is a Identifier\n", yytext);  yylval.sym=yytext;
return ID;}
{integer} {printf("%s is an Integer constant\n", yytext);
yylval.sym=yytext; return CONSTANT;}
{float} {printf("%s is a floating point constant\n", yytext);
yylval.sym=yytext; return CONSTANT;}
{nonIdentifier} {printf("ERROR: %s is an ill formed token\n", yytext);}

"+="  {printf("%s is addition assignment operator\n", yytext);  return
ADD_ASSIGN;}
"-="  {printf("%s is subtraction assignment operator\n", yytext);  return
SUB_ASSIGN;}
"*="  {printf("%s is multiplication assignment operator\n", yytext);
return MUL_ASSIGN;}
"/="  {printf("%s is division assignment operator\n", yytext);  return
DIV_ASSIGN;}
"%="  {printf("%s is mod assignment operator\n", yytext);  return
MOD_ASSIGN;}
">>=" {printf("%s is left shift assignment operator\n", yytext);  return
LEFT_ASSIGN;}
"<<=" {printf("%s is right shift assignment operator\n", yytext);  return
RIGHT_ASSIGN;}
"&="  {printf("%s is bitwise And assignment operator\n", yytext);  return
AND_ASSIGN;}
"^="  {printf("%s is bitwise Xor assignment operator\n", yytext);  return
XOR_ASSIGN;}
"|="  {printf("%s is bitwise Or assignment operator\n", yytext);  return
OR_ASSIGN;}
"++"  {printf("%s is increment operator\n",yytext);  return INC;}
"--"  {printf("%s is decrement operator\n", yytext);  return DEC;}
"=="  {printf("%s is equal comparator\n", yytext);  return EQ;}
```

```
"!="  {printf("%s is not equal comparator\n", yytext);  return 'NE';}
">"   {printf("%s is greater than comparator\n", yytext);  return '>';}
"<"   {printf("%s is less than comparator\n", yytext);  return '<';}
">="  {printf("%s is greater than equal\n", yytext);  return GE;}
"<="  {printf("%s is less than equal\n", yytext);  return LE;}
"||"  {printf("%s is logical or operator\n", yytext);  return OR;}
"&&"  {printf("%s is logical and operator\n", yytext);  return AND;}
"!"   {printf("%s is not operator\n", yytext);  return '!';}
"&"     {printf("%s is and operator\n", yytext);  return '&';}
"|"   {printf("%s is or operator\n", yytext);  return '|';}
"^"   {printf("%s is xor operator\n", yytext);  return '^';}
"~"   {printf("%s is bitwise not\n", yytext);  return '~';}
"<<"  {printf("%s is left shift operator\n", yytext);  return LEFT;}
">>"  {printf("%s is right shift operator\n", yytext);  return RIGHT;}
"?"   {printf("%s is conditional operator\n", yytext);  return '?';}
{assignmentOperator} {printf("%s is an assignment operator\n", yytext);
return '=';}
"-"   {printf("%s is arithmetic operator\n", yytext);  return '-';}
"+"   {printf("%s is arithmetic operator\n", yytext);  return '+';}
"*"   {printf("%s is arithmetic operator\n", yytext);  return '*';}
"/"   {printf("%s is arithmetic operator\n", yytext);  return '/';}
"%"   {printf("%s is arithmetic operator\n", yytext);  return '%';}
";" {printf("%s is special character\n",yytext); return(';');}
"{"   {printf("%s is special character\n", yytext);  return '{';}
"}"   {printf("%s is special character\n", yytext);  return '}';}
"("   {printf("%s is special character\n", yytext); return '(';}
")"   {printf("%s is special character\n", yytext); return ')';}
"["   {printf("%s is special character\n", yytext);  return '[';}
"]"   {printf("%s is special character\n", yytext);  return ']';}
","   {printf("%s is special character\n", yytext);  return ',';}
{whitespaces} {}
. {printf("Error: %s is an Illegal characters\n",yytext);}
"\n" {lineNo++;}
%%
```

Code - PARSER

```
%token ID CONSTANT STR HEADERFILE

%token INC DEC LE GE EQ NE
%token AND OR LEFT RIGHT MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN ADD_ASSIGN
%token SUB_ASSIGN LEFT_ASSIGN RIGHT_ASSIGN AND_ASSIGN
%token XOR_ASSIGN OR_ASSIGN

%token AUTO BREAK CASE CHAR CONST CONTINUE DEFAULT DO DOUBLE ELSE ENUM
EXTERN FLOAT FOR GOTO IF INT LONG

%token REGISTER RETURN SHORT SIGNED SIZEOF STATIC STRUCT SWITCH TYPEDEF
UNION UNSIGNED VOID VOLATILE WHILE

%union{
      int integer;
      float floating_point;
      char *sym;
}

%start startSymbol

%{
      #include <stdio.h>
      #include <stdlib.h>
      #include <string.h>
      int size=20;
      char *alc=0;
      int globalIndex=0;
      struct symbolTable{
            int index;
            char *symbol;
            char *attribute;
            char *data;
            struct symbolTable *next;
      };
      struct symbolTable *hash[2][20];
      int i=0;
      void init()
```

```
	{
		for(;i<size;i++)
		{
			hash[0][i] = NULL;
			hash[1][i] = NULL;
		}
	}

	int hashLocation(char *sym)
	{
		int sum=0;
		size_t length = strlen(sym);
		int k=0;
		for(;k<length;k++)
		{
			sum = sum + (int)sym[k];
		}
		return sum%size;
	}

	int searchHash(int type, char *sym, int x)
	{
		struct symbolTable *temp = hash[type][x];
		while(temp!=NULL)
		{
			if (strcmp(temp->symbol, sym)==0)
			{
				return 0;
			}
			temp=temp->next;
		}
		return 1;
	}

	void addToTable(int type, char *sym, char *attr, char *dat)
	{
		int x = hashLocation(sym);
		if (searchHash(type, sym, x)==0)
			return;
		struct symbolTable *newSymbol = (struct symbolTable
*)malloc(sizeof(struct symbolTable));
		char *te = (char *)malloc(strlen(sym)+1);
```

```
            strcpy(te, sym);
            newSymbol->symbol = te;
            newSymbol->attribute = attr;
            newSymbol->data = dat;
            newSymbol->next = NULL;
            newSymbol->index = globalIndex + 1;
            globalIndex++;
            struct symbolTable *temp = hash[type][x];
            hash[type][x] = newSymbol;
            hash[type][x]->next = temp;
      }

      void display()
      {
            int k=0;
            printf("\n\nSYMBOL TABLE:\n");

printf("---------------------------------------------------------------------------------\n");
            printf("%*s\t|\t%*s\t|\t%*s\t|\t%*s\n", 10, "INDEX", 10,
"SYMBOL", 10, "ATTRIBUTE", 10, "DATATYPE");

printf("---------------------------------------------------------------------------------\n");
            for(;k<size;k++)
            {
                  struct symbolTable *temp = hash[0][k];
                  while(temp!=NULL)
                  {
                        printf("%*d\t|\t%*s\t|\t%*s\t|\t%*s\n",10,
temp->index, 10, temp->symbol, 10, temp->attribute, 10, temp->data);
                        temp = temp->next;
                  }
            }

printf("----------------------------------------------------------------------------------\n");
            k=0;
            printf("\n\nCONSTANT TABLE:\n");

printf("-------------------------------------------------------------------\n");
```

```
            printf("%*s\t|\t%*s\t|\t%*s\n", 10, "INDEX", 10, "SYMBOL", 10,
"ATTRIBUTE");

printf("---------------------------------------------------------------------------
\n");
            for(;k<size;k++)
            {
                  struct symbolTable *temp = hash[1][k];
                  while(temp!=NULL)
                  {
                        printf("%*d\t|\t%*s\t|\t%*s\n",10, temp->index, 10,
temp->symbol, 10, temp->attribute);
                        temp = temp->next;
                  }
            }

printf("---------------------------------------------------------------------------
\n");
      }
%}

%%

/*The supported datatypes*/
dataType : SHORT {alc="short";}
             | INT {alc="int";}
             | LONG {alc="long";}
             | FLOAT {alc="float";}
             | DOUBLE {alc="double";}
             | VOID {alc="void";}
             ;


      /*List of all statements*/
statement : declarationStatement
              | ifAndElseMatched
              | ifAndElseUnmatched
              | whileLoopStatement
              | expressionStatement
              | jumpStatement
              | compoundStatement
              ;
```

```
statement1 : declarationStatement
              | whileLoopStatement
              | expressionStatement
              | jumpStatement
              | compoundStatement
              ;

statements : statements statement
               | statement
               ;


      /*defining each type of statements*/

      /*Declaration statements: Include declaration of identifiers, with or
without initialisation*/
declarationStatement : declarationList ';'
                               ;

declaration : dataType identi;
declarationAndAssignment : declaration '=' consta
                                   | declaration '=' stri
                                   ;

completeDeclaration : declarationAndAssignment
                              | declaration
                              ;

argumentList : argumentList ',' completeDeclaration
                        | completeDeclaration
                        ;

declarationList : dataType identifierList
                        ;

identifierList : identifierList ',' identi
                        | identifierList ',' identi '=' stri
                        | identifierList ',' identi '=' consta
                        | identifierList ',' identi '=' identi
                        | identi
                        | identi '=' stri
                        | identi '=' consta
                        | identi '=' identi
                        ;

/*declarationStatementError : declaration { printf("Semicolon missing after
statement %s\n",$1);}
                                      | declarationAndAssignment {
printf("Semicolon missing after statement %s\n",$1);}
                                      ;*/




      /*If and else statements.*/
      /*The dangling else problem is taken care by having two types of if
and else statements: Matched and unmatched statements*/
/*ifAndElseStatement : ifAndElseMatched
                           | ifAndElseUnmatched
                           ;*/
ifAndElseMatched : IF '(' expression ')' ifAndElseMatched1 ELSE
ifAndElseMatched1
                         ;
ifAndElseMatched1 : IF '(' expression ')' ifAndElseMatched1 ELSE
ifAndElseMatched1
                          | statement1
                          ;
ifAndElseUnmatched : IF '(' expression ')' ifAndElseMatched1 ELSE
ifAndElseUnmatched
                           | IF     '(' expression ')' statement1
                           ;
/*ifAndElseStatementError : IF '(' expression ifAndElseMatched ELSE
ifAndElseMatched
                                    | IF expression ')' ifAndElseMatched
ELSE ifAndElseMatched
                                    | IF expression ')' ifAndElseMatched
ELSE ifAndElseUnmatched
                                    | IF '(' expression ifAndElseMatched
ELSE ifAndElseUnmatched
                                    | IF expression ')' statement
                                    | IF '(' expression statement
                                    ;*/
      /*While loop*/
whileLoopStatement : WHILE '(' expression ')' statement
                                    ;
/*whileLoopStatementError : WHILE expression ')' statement
{addToTable(0,"while","keyword");}
                                    | WHILE '(' expression statement
{addToTable(0,"while","keyword");}
                                    ;*/


      /*Expressions are statements with operators. The expressions are
defined taking care of the precedence of operators*/
expressionStatement : expression ';'
                              ;
expression : assignment_expression
               | expression ',' assignment_expression
               ;
primary_expression
      : identi
      | consta
      | stri
      | '(' expression ')'
      ;
postfix_expression
      : primary_expression
      | postfix_expression '[' expression ']'
      | postfix_expression '(' ')'
      | postfix_expression '(' argument_expression_list ')'
      | postfix_expression '.' identi
      | postfix_expression INC
      | postfix_expression DEC
      ;

argument_expression_list
      : assignment_expression
      | argument_expression_list ',' assignment_expression
      ;

unary_expression
      : postfix_expression
      | INC unary_expression
      | DEC unary_expression
      | unary_operator cast_expression
      | SIZEOF unary_expression
      | SIZEOF '(' dataType ')'
      ;
unary_operator
      : '&'
      | '*'
      | '+'
      | '-'
      | '~'
      | '!'
      ;
cast_expression
      : unary_expression
      | '(' dataType ')' cast_expression
      ;
multiplicative_expression
      : cast_expression
      | multiplicative_expression '*' cast_expression
      | multiplicative_expression '/' cast_expression
      | multiplicative_expression '%' cast_expression
      ;
additive_expression
      : multiplicative_expression
      | additive_expression '+' multiplicative_expression
      | additive_expression '-' multiplicative_expression
      ;
shift_expression
      : additive_expression
      | shift_expression LEFT additive_expression
      | shift_expression RIGHT additive_expression
      ;
relational_expression
      : shift_expression
      | relational_expression '<' shift_expression
      | relational_expression '>' shift_expression
      | relational_expression LE shift_expression
      | relational_expression GE shift_expression
      ;
equality_expression
      : relational_expression
      | equality_expression EQ relational_expression
      | equality_expression NE relational_expression
      ;
and_expression
      : equality_expression
      | and_expression '&' equality_expression
      ;
exclusive_or_expression
      : and_expression
      | exclusive_or_expression '^' and_expression
      ;
inclusive_or_expression
      : exclusive_or_expression
      | inclusive_or_expression '|' exclusive_or_expression
      ;
logical_and_expression
      : inclusive_or_expression
      | logical_and_expression AND inclusive_or_expression
      ;
logical_or_expression
      : logical_and_expression
      | logical_or_expression OR logical_and_expression
      ;
conditional_expression
      : logical_or_expression
      | logical_or_expression '?' expression ':' conditional_expression
      ;
assignment_expression
      : conditional_expression
      | unary_expression assignment_operator assignment_expression
      ;
assignment_operator
      : '='
      | MUL_ASSIGN
      | DIV_ASSIGN
      | MOD_ASSIGN
      | ADD_ASSIGN
      | SUB_ASSIGN
      | LEFT_ASSIGN
      | RIGHT_ASSIGN
      | AND_ASSIGN
      | XOR_ASSIGN
      | OR_ASSIGN
      ;


      /*Jump statements include continue,break and return statements*/
jumpStatement : CONTINUE ';'
                    | BREAK ';'
                    | RETURN ';'
                    | RETURN expression ';'
                    ;
/*                jumpStatementError : CONTINUE
                           | BREAK
                           | RETURN
                           ;*/


      /*Compound statements are the statements enclosed within the curly
braces*/
compoundStatement : '{' statements '}'
                          | '{' '}'
                          ;
/*compoundStatementError : '{'
                                   | '}'
                                   | '{' statements
                                   | statements '}'
                                   ;*/



startSymbol
      : external_declaration
      | startSymbol external_declaration
      ;

external_declaration
      : functionDefinition
      | declarationStatement
      | HEADERFILE
      ;
```

```
functionDefinition : declaration '(' ')' compoundStatement
                           | declaration '(' argumentList ')'
compoundStatement
                           ;

identi : ID {addToTable(0,yylval.sym,"identifier", alc);};
consta : CONSTANT {addToTable(1,yylval.sym,"constant", "");};
stri : STR {addToTable(1,yylval.sym,"string", "");};
%%

#include<stdio.h>
extern int lineNo;
int main()
{
      init();
      yyparse();
      display();
}

int yywrap()
{
      return 1;
}

void yyerror(char *msg)
{
      printf("Error: %s in line %d\n",msg, lineNo);
}
```

Explanation of implementation

The lex code identifies tokens and returns them to the parser. The lex rules are written in the form of regular expressions.

The parser code takes the tokens from lexical analyser and matched them with the rules. The parser rules are written in the form of context free grammar.

The parser contain the following main rules:

1) Start symbol - Every program has to start with external declarations of functions or variables and/or header files. Also there can be one or multiple such declaration statements, so two alternatives are added to the startSymbol rule.
    a) External Declaration - It is further divided into declaration of variables and functions and headerfiles.
        i) Function Definition - This consists of the declaration of functions
        ii) Declaration Statement - Consists of declaration of variables with or without assignment.
        iii) Headerfile - Consists of declaration of header files to be included.
2) Statement - The statement is subdivided into
    a) Declaration statement - This consists of the declaration of new variables and declaration with assignment
    b) If and else statement - This includes two types of if and else statements: Matched and unmatched. The splitting is to remove the “dangling else problem”.The rule is able to determine to which “if” the “else” belongs to.
    c) While loop - While loop statement is to distinguish the complete while loop into just one statement.
    d) Expression statement - Expression statement consists of the expressions which include arithmetic(+, -, /, *, ^), assignment, comparison expressions, etc.
    e) Jump statement - This consists of various decision jump statements like continue, break and return.
    f) Compound statement - Compound statements consists of statements individually or statements in braces ({ }), which is considered to be a single statement.

Output and screenshots

Our program identifies various types of tokens and displays them. Initially, it prints all the tokens as it encounters them. The syntax analyser calls the scanner to fetch tokens one by one and tries to match to the rules specified in the parser. The identifiers and constants are inserted into the symbol and constant table when encountered. In the end, every token is displayed in symbol and constants table. We have made various test cases depicting various functionalities of our compiler.

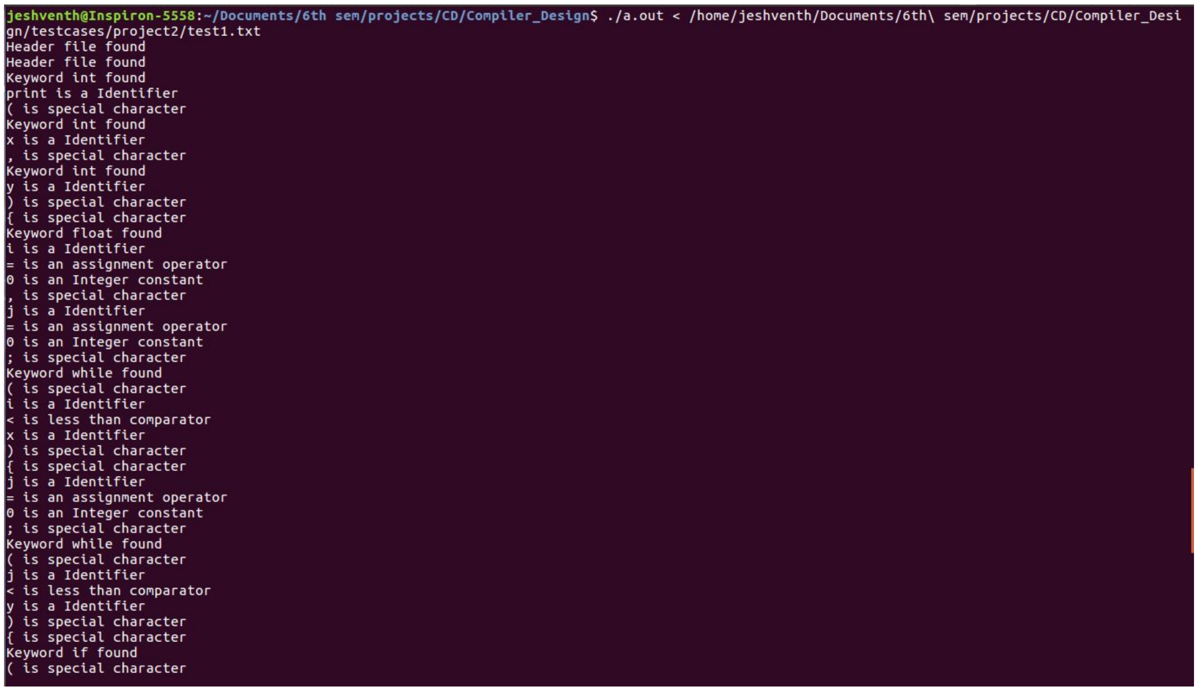
Test case 1

This test case doesn't have any lexical and syntax errors. This test case covers:

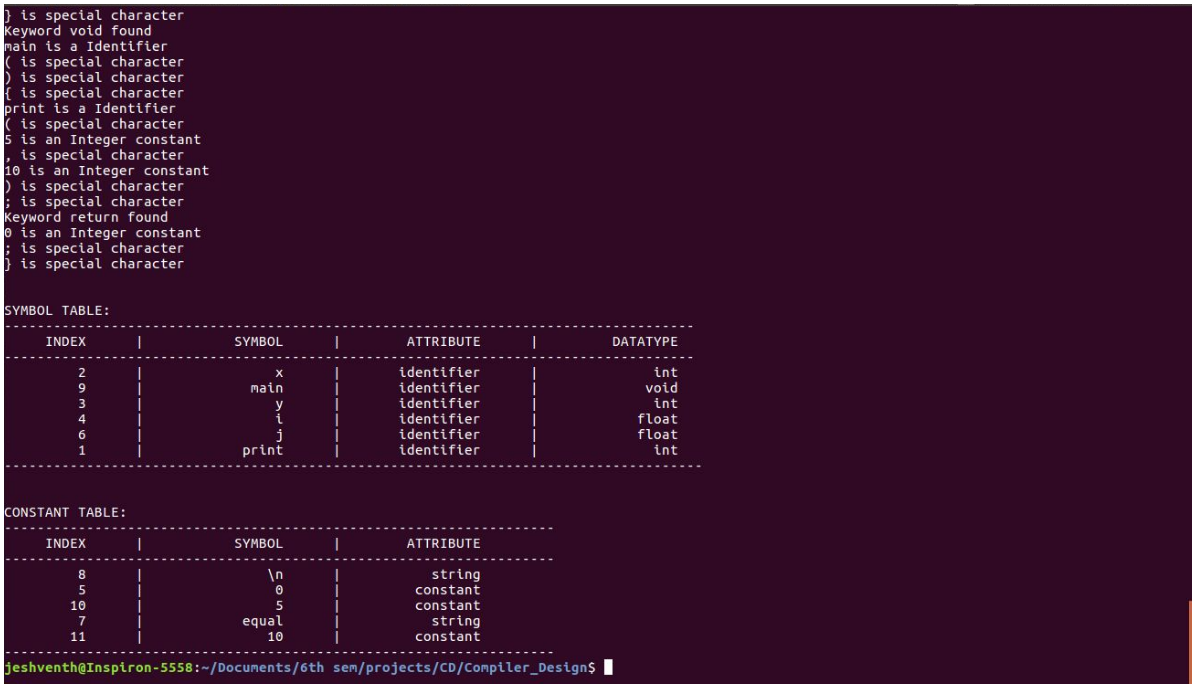
1) Header files
2) Function definition
3) While loop
4) Declaration statement
5) Compound statement
6) If and else statement

```
#include <stdio.h>
#include <string.h>

int print(int x, int y)
{
	float i=0, j=0;

	while(i<x)
	{
		j=0;
		while(j<y)
		{
```

```
                        if(i==j)
                        {
                                print("equal");
                        }

                        j++;
                }
                print("\n");
                i++;
        }
}

void main()
{
        print(5,10);
        return 0;
}
```



```
jeshventh@Inspiron-5558:~/Documents/6th sem/projects/CD/Compiler_Design$ ./a.out < /home/jeshventh/Documents/6th\ sem/projects/CD/Compiler_Desi
gn/testcases/project2/test1.txt
Header file found
Header file found
Keyword int found
print is a Identifier
( is special character
Keyword int found
x is a Identifier
, is special character
Keyword int found
y is a Identifier
) is special character
{ is special character
Keyword float found
i is a Identifier
= is an assignment operator
0 is an Integer constant
, is special character
j is a Identifier
= is an assignment operator
0 is an Integer constant
; is special character
Keyword while found
( is special character
i is a Identifier
< is less than comparator
x is a Identifier
) is special character
{ is special character
j is a Identifier
= is an assignment operator
0 is an Integer constant
; is special character
Keyword while found
( is special character
j is a Identifier
< is less than comparator
y is a Identifier
) is special character
{ is special character
Keyword if found
( is special character
```

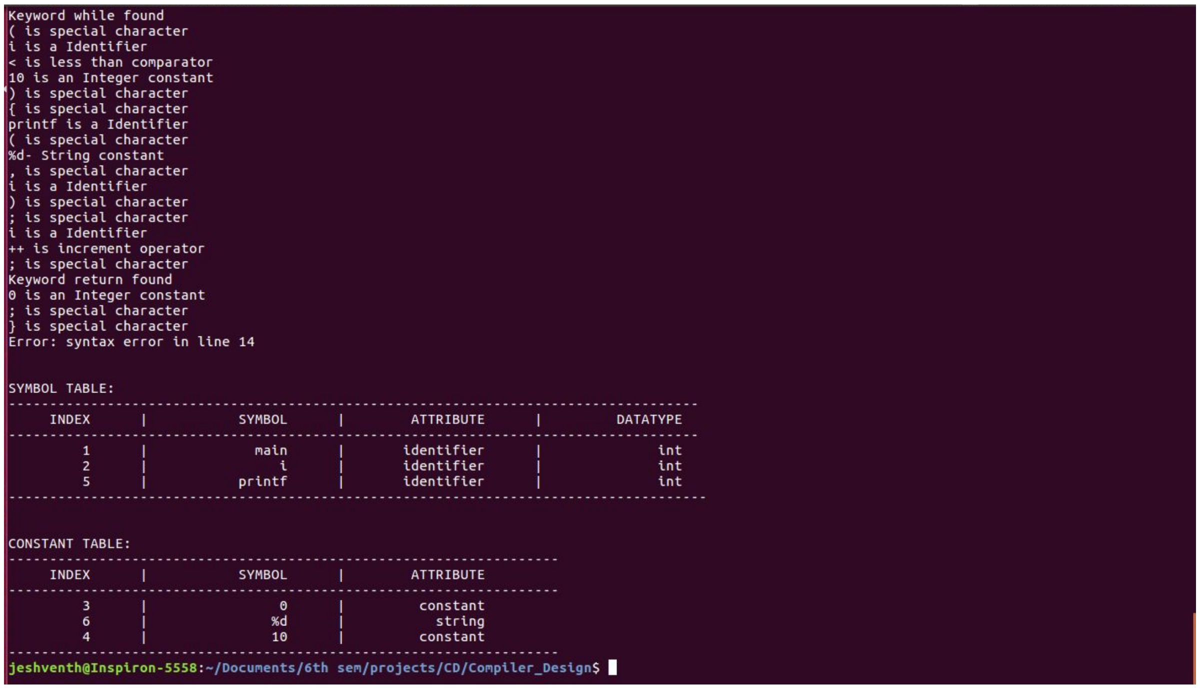


```
} is special character
Keyword void found
main is a Identifier
( is special character
) is special character
{ is special character
print is a Identifier
( is special character
5 is an Integer constant
, is special character
10 is an Integer constant
) is special character
; is special character
Keyword return found
0 is an Integer constant
; is special character
} is special character


SYMBOL TABLE:
------------------------------------------------------------------------------------
     INDEX      |           SYMBOL       |       ATTRIBUTE        |          DATATYPE
------------------------------------------------------------------------------------
         2      |                x       |      identifier        |               int
         9      |             main       |      identifier        |              void
         3      |                y       |      identifier        |               int
         4      |                i       |      identifier        |             float
         6      |                j       |      identifier        |             float
         1      |            print       |      identifier        |               int
-------------------------------------------------------------------------------------


CONSTANT TABLE:
-------------------------------------------------------------------
     INDEX      |           SYMBOL       |       ATTRIBUTE
-------------------------------------------------------------------
         8      |               \n       |          string
         5      |                0       |        constant
        10      |                5       |        constant
         7      |            equal       |          string
        11      |               10       |        constant
-------------------------------------------------------------------
jeshventh@Inspiron-5558:~/Documents/6th sem/projects/CD/Compiler_Design$
```

Test case 2

The test case has a missing curly brace. The closing curly brace isn't found and hence none of the rule match after the end of parsing. Therefore, it is a syntax error.

```c
#include <stdio.h>

//This test case has missing curly brace

int main()
{
	int i=0;
	while(i<10)
	{
		printf("%d",i);
		i++;

	return 0;
}
```

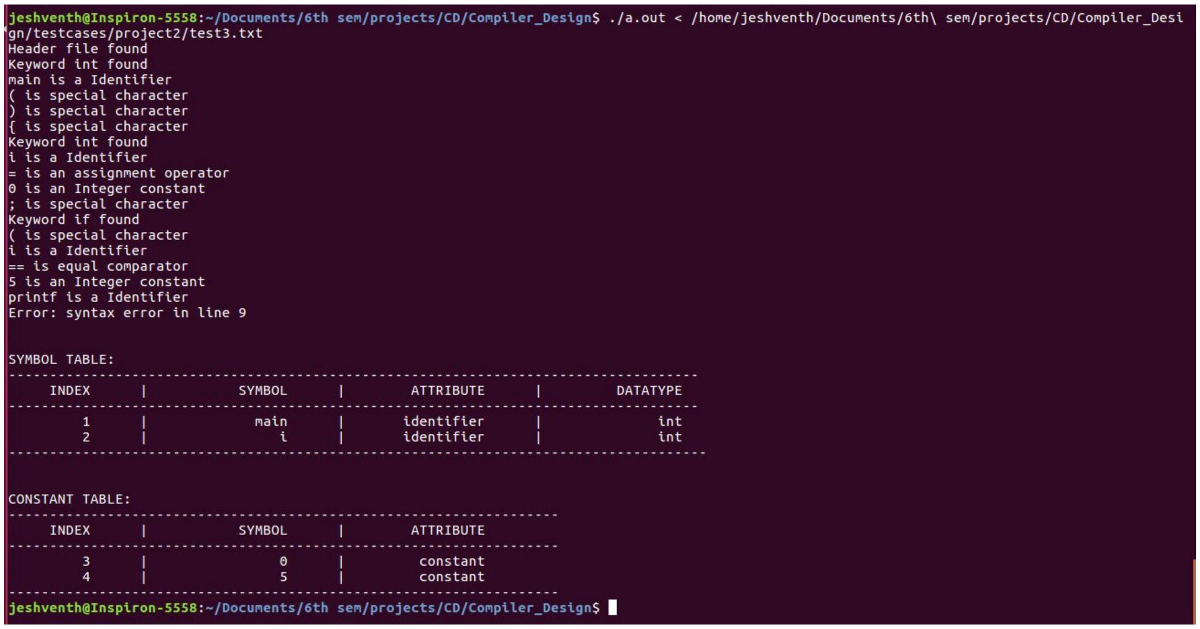


```
Keyword while found
( is special character
i is a Identifier
< is less than comparator
10 is an Integer constant
) is special character
{ is special character
printf is a Identifier
( is special character
%d- String constant
, is special character
i is a Identifier
) is special character
; is special character
i is a Identifier
++ is increment operator
; is special character
Keyword return found
0 is an Integer constant
; is special character
} is special character
Error: syntax error in line 14

SYMBOL TABLE:
-------------------------------------------------------------------------------------
     INDEX       |            SYMBOL       |          ATTRIBUTE       |          DATATYPE
-------------------------------------------------------------------------------------
         1       |              main       |        identifier        |               int
         2       |                 i       |        identifier        |               int
         5       |            printf       |        identifier        |               int
--------------------------------------------------------------------------------------


CONSTANT TABLE:
-------------------------------------------------------------------
     INDEX       |            SYMBOL       |          ATTRIBUTE
-------------------------------------------------------------------
         3       |                 0       |          constant
         6       |                %d       |            string
         4       |                10       |          constant
-------------------------------------------------------------------
jeshventh@Inspiron-5558:~/Documents/6th sem/projects/CD/Compiler_Design$
```

Test case 3

This test case has missing closing parenthesis. When printf is encountered, none of the rules will match and hence the parser reports a syntax error at that point.

```
#include <stdio.h>

//This test case has missing parenthesis

int main()
{
      int i=0;
      if(i == 5
            printf("True");
      return 0;
}
```

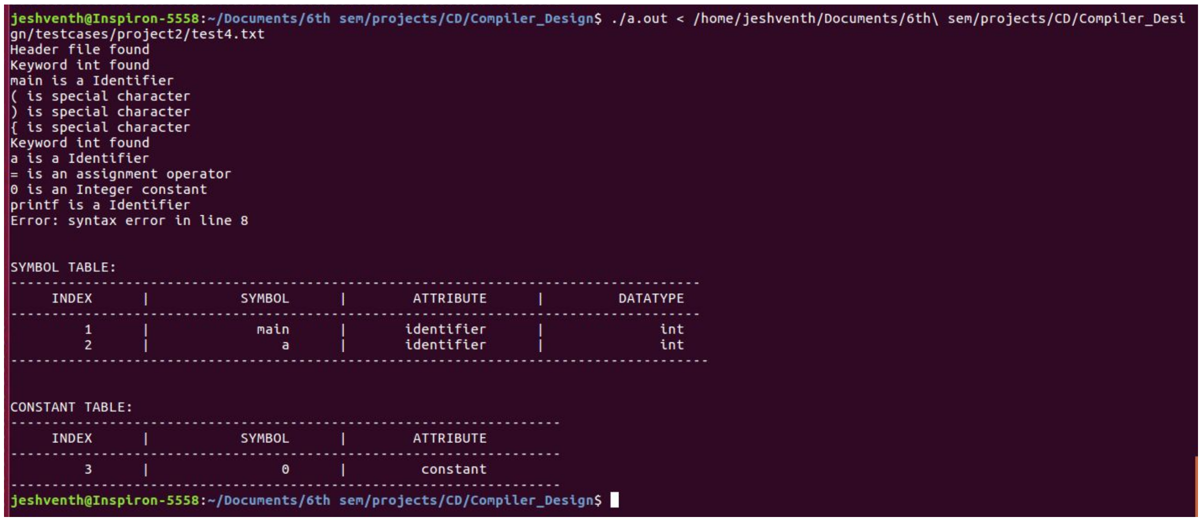


```
jeshventh@Inspiron-5558:~/Documents/6th sem/projects/CD/Compiler_Design$ ./a.out < /home/jeshventh/Documents/6th\ sem/projects/CD/Compiler_Desi
gn/testcases/project2/test3.txt
Header file found
Keyword int found
main is a Identifier
( is special character
) is special character
{ is special character
Keyword int found
i is a Identifier
= is an assignment operator
0 is an Integer constant
; is special character
Keyword if found
( is special character
i is a Identifier
== is equal comparator
5 is an Integer constant
printf is a Identifier
Error: syntax error in line 9


SYMBOL TABLE:
-----------------------------------------------------------------------------------
     INDEX      |             SYMBOL      |          ATTRIBUTE      |          DATATYPE
-----------------------------------------------------------------------------------
         1      |               main      |        identifier       |               int
         2      |                  i      |        identifier       |               int
------------------------------------------------------------------------------------


CONSTANT TABLE:
------------------------------------------------------------------
     INDEX      |             SYMBOL      |          ATTRIBUTE
------------------------------------------------------------------
         3      |                  0      |         constant
         4      |                  5      |         constant
------------------------------------------------------------------
jeshventh@Inspiron-5558:~/Documents/6th sem/projects/CD/Compiler_Design$ 
```

Test case 4

This test case has a missing semicolon. When printf is encountered, none of the rules will match and hence a syntax error is reported.

```
#include <stdio.h>

//This test case has missing semicolon

int main()
{
	int a=0
	printf("%d",a);
	return 0;
}
```

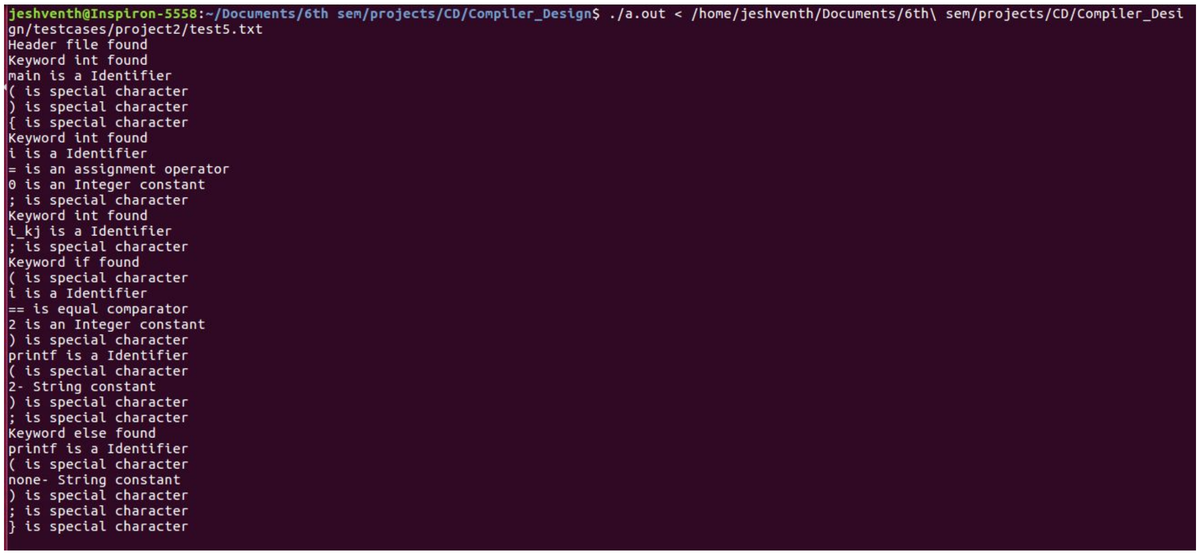


```
jeshventh@Inspiron-5558:~/Documents/6th sem/projects/CD/Compiler_Design$ ./a.out < /home/jeshventh/Documents/6th\ sem/projects/CD/Compiler_Desi
gn/testcases/project2/test4.txt
Header file found
Keyword int found
main is a Identifier
( is special character
) is special character
{ is special character
Keyword int found
a is a Identifier
= is an assignment operator
0 is an Integer constant
printf is a Identifier
Error: syntax error in line 8


SYMBOL TABLE:
-----------------------------------------------------------------------------------
     INDEX       |           SYMBOL       |          ATTRIBUTE       |          DATATYPE
-----------------------------------------------------------------------------------
         1       |             main       |         identifier       |               int
         2       |                a       |         identifier       |               int
------------------------------------------------------------------------------------


CONSTANT TABLE:
-----------------------------------------------------------------
     INDEX       |           SYMBOL       |          ATTRIBUTE
-----------------------------------------------------------------
         3       |                0       |          constant
-----------------------------------------------------------------
jeshventh@Inspiron-5558:~/Documents/6th sem/projects/CD/Compiler_Design$ █
```

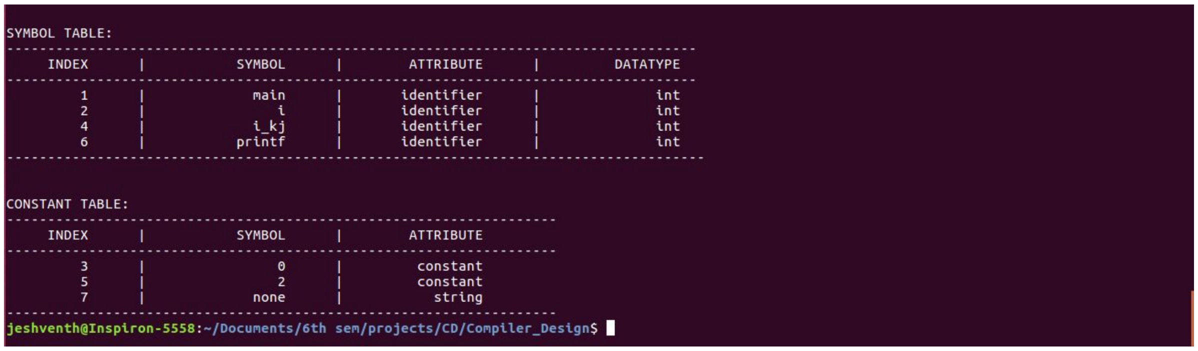
Test case 5

This test case has no errors. It demonstrates a solution for dangling else problem (Shift reduce conflict)

```
#include <stdio.h>

//This test case handles the shift reduce conflict occurring due to
dangling else problem

int main()
{
	int i = 0;
	if(i==1)
	if(i==2)
		printf("2");
	else
		printf("none");
}
```

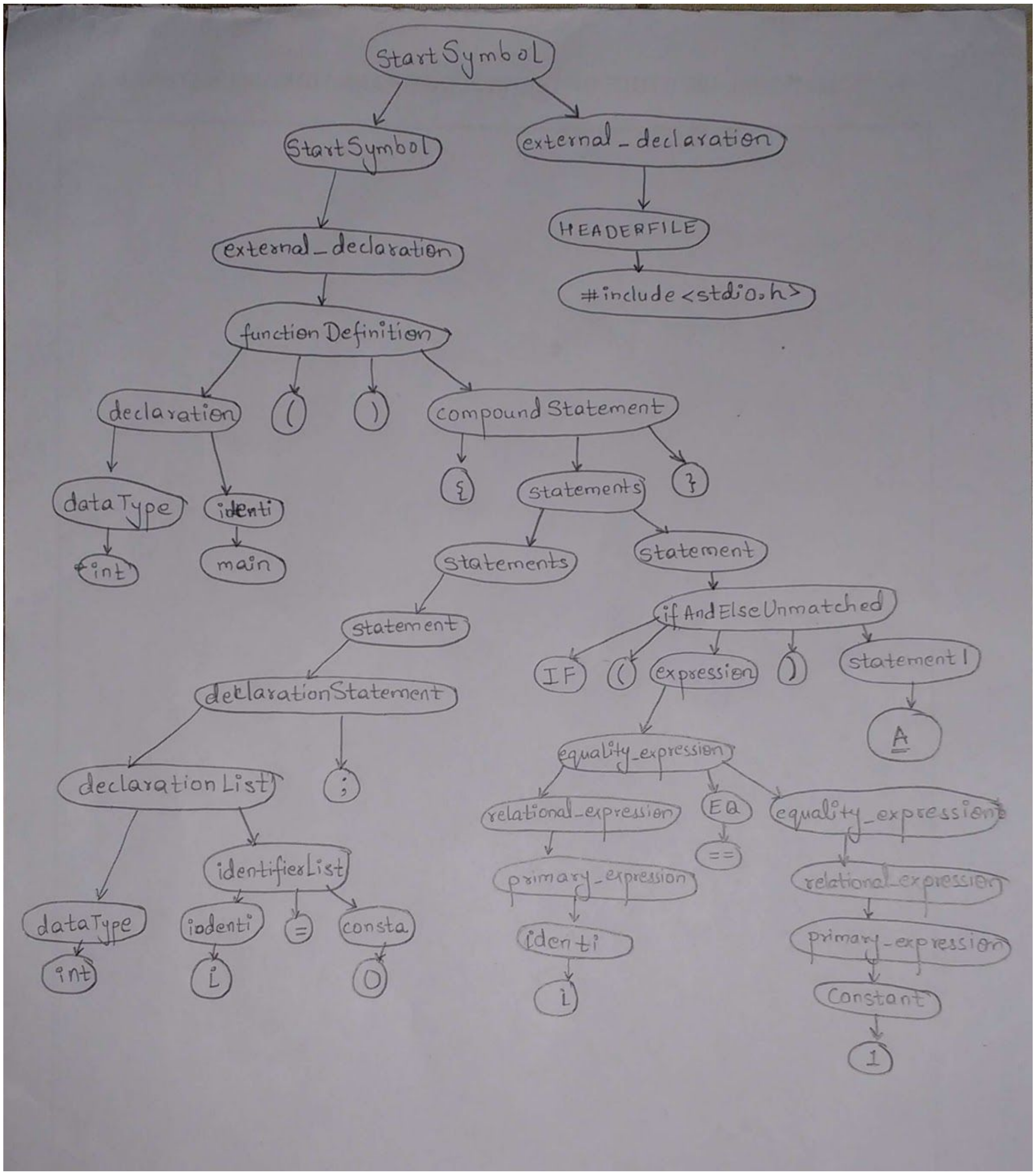


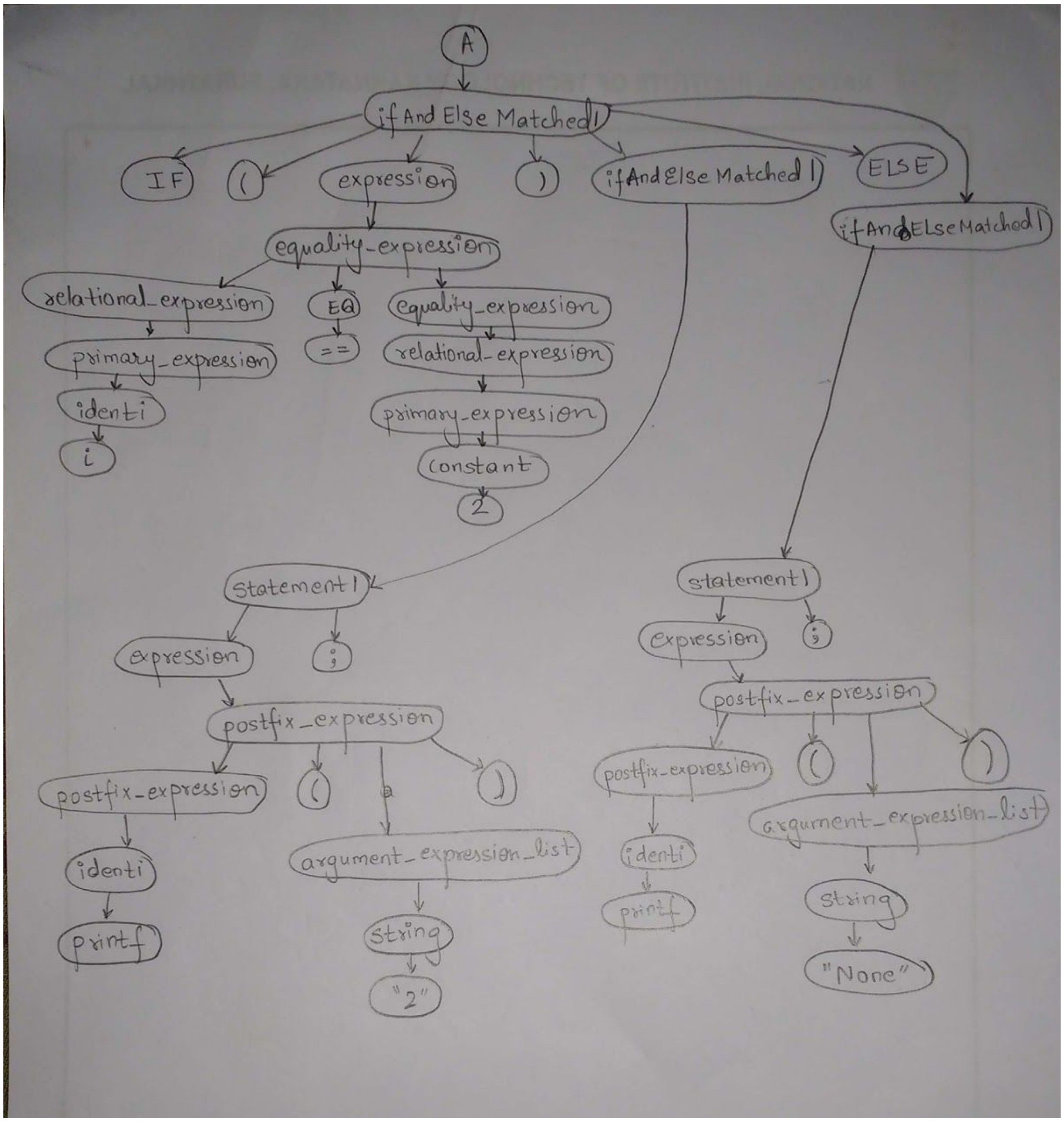
```
jeshventh@Inspiron-5558:~/Documents/6th sem/projects/CD/Compiler_Design$ ./a.out < /home/jeshventh/Documents/6th\ sem/projects/CD/Compiler_Desi
gn/testcases/project2/test5.txt
Header file found
Keyword int found
main is a Identifier
( is special character
) is special character
{ is special character
Keyword int found
i is a Identifier
= is an assignment operator
0 is an Integer constant
; is special character
Keyword int found
i_kj is a Identifier
; is special character
Keyword if found
( is special character
i is a Identifier
== is equal comparator
2 is an Integer constant
) is special character
printf is a Identifier
( is special character
2- String constant
) is special character
; is special character
Keyword else found
printf is a Identifier
( is special character
none- String constant
) is special character
; is special character
} is special character
```



```
SYMBOL TABLE:
------------------------------------------------------------------------------------
     INDEX      |           SYMBOL      |         ATTRIBUTE      |          DATATYPE
------------------------------------------------------------------------------------
         1      |             main      |        identifier      |               int
         2      |                i      |        identifier      |               int
         4      |             i_kj      |        identifier      |               int
         6      |           printf      |        identifier      |               int
-------------------------------------------------------------------------------------


CONSTANT TABLE:
-------------------------------------------------------------------
     INDEX      |           SYMBOL      |         ATTRIBUTE
-------------------------------------------------------------------
         3      |                0      |          constant
         5      |                2      |          constant
         7      |             none      |            string
-------------------------------------------------------------------
jeshventh@Inspiron-5558:~/Documents/6th sem/projects/CD/Compiler_Design$
```

Parse Tree


StartSymbol
StartSymbol
external_declaration
HEADERFILE
#include<stdio.h>
external_declaration
function Definition
declaration
(
)
compound Statement
dataType
identi
int
main
{
statements
}
statements
statement
statement
ifAndElseUnmatched
declarationStatement
IF
(
expression
)
statement1
A
equality_expression
declarationList
;
relational_expression
EQ
equality_expression
==
identifierList
primary_expression
relational_expression
dataType
iodenti
=
consta
identi
primary_expression
int
i
0
i
Constant
1


A
ifAndElseMatched1
IF
(
expression
)
ifAndElseMatched1
ELSE
ifAndElseMatched1
equality_expression
relational_expression
EQ
equality_expression
primary_expression
==
relational_expression
identi
primary_expression
i
constant
2
statement1
statement1
expression
;
expression
;
postfix_expression
postfix_expression
postfix_expression
(
)
postfix_expression
(
)
argument_expression_list
identi
argument_expression_list
identi
string
printf
string
printf
"None"
"2"