NATIONAL INSTITUTE OF TECHNOLOGY, TIRUCHIRAPPALLI

CSPC62

COMPILER DESIGN

TOPIC: C++ Compiler

LAB REPORT-2

Sub Topic : Syntax Analyzer

DONE BY:

S.No	Name	RollNo
1.	Hema Sai Dorababu K	106121065
2.	Chetan Reddy T	106121139
3.	Harshith Babu B	106121029

SYNTAX ANALYZER

Introduction

What is a Syntax Analyzer?

A syntax analyzer, also known as a parser, is a program that checks the grammatical structure of an input program written in a specific programming language. It verifies if the code follows the language's rules and constructs.

Role in Compilers:

Syntax analysis is a crucial stage in the compilation process, typically the second phase after lexical analysis.

1. **Lexical Analysis:** Breaks down the source code into smaller meaningful units called tokens (keywords, identifiers, operators, literals).
2. **Syntax Analysis:** Checks if the sequence of tokens adheres to the grammar rules of the programming language.

How it Works:

- The syntax analyzer uses a set of predefined rules, that define valid program structures.
- It reads the token stream generated by the lexical analyzer one token at a time.
- It applies the grammar rules to determine if the sequence of tokens forms a valid construct in the language.
- If a violation is found, it reports a syntax error with details like the line number and potential error message.

Benefits:

- **Early Error Detection:** Syntax analysis helps identify errors early in the development process, saving time and effort compared to debugging runtime errors.
- **Improved Code Quality:** By enforcing language rules, syntax analysis promotes well-structured and maintainable code.
- **Enables Further Compilation Stages:** A successfully parsed program can proceed to semantic analysis (checking types) and code generation (creating machine code).

Limitations:

- **Focuses on Structure:** Syntax analyzers only verify the code's structure, not its logical correctness or efficiency.
- **Language Specific:** Each language has its own grammar rules, so a syntax analyzer is designed for a particular programming language.

Applications:

Syntax analyzers are used in various contexts:

- **Compilers and Interpreters:** As a core component for translating code into machine code or executing it directly.
- **Text Editors and IDEs:** To provide real-time syntax highlighting and error checking as developers write code.
- **Data Validation:** To ensure data entered in a specific format adheres to defined rules.

Key Functionalities:

- **Lexical Analysis:** The code likely uses the `yyin` stream and `yytext` variable to handle the input source code. Functions like `yyerror` might be used for error reporting during the lexical analysis phase (not shown in the provided code).
- **Grammar Parsing:** The code utilizes the `yylex` function to retrieve tokens from the input. The `%token` section defines the recognized tokens, including keywords, operators, identifiers, and literals.
- **Abstract Syntax Tree (AST) Construction:** The parser uses a recursive descent approach to build an AST representing the program structure. The `create_Node` function creates AST nodes with labels, values, and references to child nodes.
- **Error Handling:** The `yyerror` function reports syntax errors encountered during parsing, potentially including line numbers and error messages.

Parsing Rules:

The grammar rules are defined using BNF (Backus-Naur Form) notation within the `%%` section. These rules specify how different tokens can be combined to form valid program constructs. Here are some key observations:

- The program can start with a sequence of headers, followed by a function declaration and more program statements, or just statements alone.
- Function declarations include return type, identifier, parameter list, and a compound statement containing local declarations and statements.
- Statements can be variable declarations, assignments, control flow statements (if, else, while, for), input/output statements (cout, cin), return statements, or error handling.
- Expressions are built using arithmetic operators (+, -, *, /), comparison operators (<=, >=, ==, !=, <, >), and unary operators.

1. Create a parser for your programming language:

Code Snippets:

Lexer.l:

```
%option yylineno
%{
	#include "temp2.tab.h"
	#include<stdio.h>
	#include<stdlib.h>
	#include <string.h>
	#include <stdarg.h>
	#include "TreeNode.h"
	int st[100];
	int top,count,currscope,up,declared = 0;
	char decl[20];
	int flag = 0;
	void installID(char *text,int nm,int scp);
	void display();
	struct entry
	{
		char arr[20];
		int scope;
		char dtype[10];
		int value;
	};
	struct entry symbolTable[100];

	TreeNode *createNode(char *label, int value, char *value_str, int num_children,
...) {
```

```
            TreeNode *newNode = (TreeNode *)malloc(sizeof(TreeNode));
            newNode->label = label;
            newNode->value = value;
            newNode->value_str = (char*)malloc(sizeof(char) * strlen(value_str) + 1);
            strcpy(newNode->value_str,value_str);
            newNode->num_children = num_children;
            newNode->children = malloc(sizeof(TreeNode*) * num_children);

            va_list args;
            va_start(args, num_children);
            for (int i = 0; i < num_children; i++) {
                newNode->children[i] = va_arg(args, TreeNode*);
            }
            va_end(args);
            return newNode;
        }
%}
/* regular definitions */
delim [ \t]
ws {delim}+
letter [a-zA-Z]
digit [0-9]
id {letter}({letter}|{digit})*
relop (<=|>=|==|!=|<|>)
logicalop (&&|[|][|])
leftshift (<<)
rightshi (>>)
plus [+]
minus [-]
```

```
mult [*]

div [/]

num ({digit}+)

float ({num}\.{num})

arithmeticop ({plus}|{minus}|{mult}|{div})

increment {plus}{plus}

decrement {minus}{minus}

assignop =

string (\"(\\.|[^\\"])*\")

character (\'(\\.|[^\\'])*\')

keyword (if|else|const|while|for|int|float|return|void|main|char|"long
long"|double|short|long|unsigned|signed|define|struct|enum|typedef|sizeof|sta
c|register|auto|break|case|con
nue|default|do|goto|switch|cout|cin|endl|bool|using|namespace|std|include|iostream|
vector|map|set|queue|stack|push_back|pop_back|pop|push|top|front|priority_queue)

inval (({digit}+{id}))

comment (\/\/.*|\/\*([^**]|(\*+[^**/]))*\*+\/[\n]*)


%%

[\n] {printf("newline\n");return EOL;}

#include<[^>]+> {printf( "header:%s\n",yytext); yylval.node = createNode("header",-
1,yytext,0);  return header;};

#include\"([^\"]+)\" {printf( "header:%s\n",yytext); yylval.node = createNode("header",-
1,yytext,0);  return header;};

if { printf("if\n");  yylval.node = createNode("if",-1,yytext,0);  return if_x; }

else { printf("else\n");  yylval.node = createNode("else",-1,yytext,0);  return else_x; }

while { printf("while\n"); yylval.node = createNode("while",-1,yytext,0);  return while_x; }

for {printf("for\n"); yylval.node = createNode("for",-1,yytext,0);  return for_x; }

return { printf("return \n"); yylval.node = createNode("return",-1,yytext,0);  return
return_x; }

printf { printf("printf\n"); yylval.node = createNode("printf",-1,yytext,0);  return printf_x; }

cout { printf("cout\n"); yylval.node = createNode("cout",-1,yytext,0);  return cout; }
```

```
cin { printf("cin\n"); yylval.node = createNode("cin",-1,yytext,0);  return cin; }

"<<" { printf("insert\n");  yylval.node = createNode("<<",-1,yytext,0);  return insert; }

">>" { printf("extract\n"); yylval.node = createNode(">>",-1,yytext,0);  return extract; }

{assignop} {printf("assignop:%s\n", yytext); yylval.node = createNode("assignop",-
1,yytext,0);  return assignmentop; }

{relop} { ; printf("compop:%s\n", yytext); strcpy(yylval.str,yytext); yylval.node =
createNode("comparisionop",-1,yytext,0);  return comparisionop; }

{logicalop} { printf("logicalop:%s\n", yytext); return logicalop; }

int|float|double|char|string|"long long"|short|long { printf("datatype:%s\n",
yytext);declared = 1;strcpy(decl, yytext);  yylval.node = createNode("datatype",-
1,yytext,0);  return datatype; }

{num} { printf("num:%s\n", yytext); yylval.node =
createNode("number",atoi(yytext),"NULL",0); return number; }

{increment} {  ; printf("unary:%s\n", yytext); yylval.node = createNode("unary",-
1,yytext,0);  return unary; }

{decrement} {  ; printf("unary:%s\n", yytext); yylval.node = createNode("unary",-
1,yytext,0);  return unary; }

{id} { if(declared == 1) {installID(yytext, yylineno, st[top]);}  printf("id:%s\n", yytext);
yylval.node = createNode("id",-1,yytext,0); return identifier; }

{character} { printf("character:%s\n", yytext);  yylval.node = createNode("character",-
1,yytext,0); return character;}

{string} { printf("string:%s\n", yytext); yylval.node = createNode("string",-1,yytext,0);
return string; }

\n* ;

{ws} ;

{plus} { yylval.node = createNode("plus",-1,yytext,0); return PLUS;}

{minus} { yylval.node = createNode("minus",-1,yytext,0); return MINUS;}

{mult} { yylval.node = createNode("mult",-1,yytext,0); return MUL;}

{div} { yylval.node = createNode("div",-1,yytext,0); return DIV;}

"{" { printf("Lbrace:%s\n",yytext); currscope++; yylval.node = createNode("{",-1,yytext,0);
return LBRACE; }

"}" { currscope--;printf("Rbrace:%s\n",yytext); yylval.node = createNode("}",-1,yytext,0);
return RBRACE; }
```

```
"(" {printf ("LPAREN: %s, line: %d\n", yytext, yylineno); yylval.node =
createNode("LPAREN",-1,yytext,0); return LPAREN;}

")" {printf ("RPAREN: %s, line: %d\n", yytext, yylineno); yylval.node =
createNode("RPAREN",-1,yytext,0); return RPAREN;}

"[" {printf ("LBRACKET: %s, line: %d\n", yytext, yylineno); yylval.node = createNode("[",-
1,yytext,0);  return yytext[0];}

"]" {printf ("RBRACKET: %s, line: %d\n", yytext, yylineno); yylval.node = createNode("]",-
1,yytext,0);  return yytext[0];}

";" { declared = 0; printf("semicolon :%s,line:%d\n",yytext,yylineno);  yylval.node =
createNode(";",-1,yytext,0);  return SEMICOLON; }

"," { printf("comma :%s,line:%d\n",yytext,yylineno);  yylval.node = createNode(",",-
1,yytext,0);  return COMMA; }

{comment} { };

. { return yytext[0]; }

%%


void installID(char *text, int nm, int scp)

{

        int present = 0;

        for (int i = 0; i <= up; i++)

        {

                if (strcmp(symbolTable[i].arr, text) == 0 && symbolTable[i].scope ==
currscope)

                {

                        present = 1;

                        break;

                }

        }

        if (!present)

        {

                strcpy(symbolTable[up].arr, text);

                symbolTable[up].scope = currscope;
                strcpy(symbolTable[up].dtype, decl);
                // symbolTable[up].value = up;
                up++;
        }
}

void display()
{
        printf("\nSymbol Table\n");
        printf("Symbol\t\tscope\t\tdtype\n");
        for (int i = 0; i < up; i++)
        {
                printf("%s\t\t%d\t\t%s\n", symbolTable[i].arr,symbolTable[i].scope,
symbolTable[i].dtype);
        }
}
int yywrap()
{
        return 1;
}
```

Parser.y:

```
%{
        #include "TreeNode.h"
        #include<stdio.h>
        #include<stdlib.h>
        #include<string.h>
        int yylex();
```

```
	int yyerror(const char *s);
	int yyparse();
	extern void display();
	struct entry{
		char arr[20];
		int scope;
		char dtype[10];
		int value;
	};
	extern struct entry symbolTable[100];


	TreeNode *head = NULL;
	struct TreeNode *create_Node(char *label, int value, char *value_str, int
num_children, ...) {
		struct TreeNode *newNode = (struct TreeNode *)malloc(sizeof(struct
TreeNode));
		newNode->label = label;
		newNode->value = value;
		newNode->value_str = (char*)malloc(sizeof(char) * strlen(value_str) + 1);
		strcpy(newNode->value_str,value_str);
		newNode->num_children = num_children;
		newNode->children = NULL;
		if(num_children<1) return newNode;
		newNode->children = malloc(sizeof(TreeNode*) * num_children);


		va_list args;
		va_start(args, num_children);
```

```
        for (int i = 0; i < num_children; i++) {
                newNode->children[i] = va_arg(args, struct TreeNode*);
        }

        va_end(args);

        return newNode;
}

void printTree(TreeNode *root,int level){
        if(root == NULL){
                return;
        }
        for(int i = 0;i<level;i++){
                printf("   ");
        }

        if(root->value ==-1 && strcmp(root->value_str,"NULL") == 0){
                printf("%d.%s\n",level, root->label);
        }else if(root->value == -1){
                printf("%d.%s\n", level,root->value_str);
        }else{
                printf("%d.(%s,%d)\n",level, root->label, root->value);
        }
        // printf("(%s,%d)\n", root->label,root->value);
        for(int i = 0;i<root->num_children;i++){
                printTree(root->children[i],level+1);
        }
}
```

```
%}


%union {
   int num;
   char *str;
           struct TreeNode *node;
}


%start program
%token EOL
%error-verbose
%token<node> PLUS MINUS MUL DIV number if_x else_x while_x for_x return_x printf_x
main_x assignmentop comparisionop logicalop datatype unary identifier string
character cout cin insert extract header LBRACE RBRACE LPAREN RPAREN
SEMICOLON COMMA
%type<node> E T F assignment_statement statement_list function_declaration
declaration_statement id_list insert_statement extract_statement if_statement
else_statement if_else_statement while_statement for_statement return_statement
cout_statement cin_statement statement headers parameter_list program
/* rules */
%%


program: headers function_declaration program { printf("program No: %d\n",$$); $$ =
create_Node("program", -1, "NULL", 3,$1,$2,$3);head = $$;}
| statement_list program {$$ = create_Node("program", -1, "NULL", 2,$1,$2); head = $$;}
| EOL program {$$ = create_Node("program", -1, "NULL", 1,$2); head = $$;}
| EOL {$$ = NULL;}
|;



headers: header {printf("headers1\n"); $$ = create_Node("headers", -1, "NULL",1,$1);}
```

```
| headers EOL headers {printf("headers\n"); $$ = create_Node("headers", -1,"NULL", 2,
$1, $3);}

| EOL headers {printf("headers\n"); $$ = create_Node("headers", -1, "NULL",1, $2);}

| EOL{ $$ = NULL; };


function_declaration: datatype identifier LPAREN parameter_list RPAREN LBRACE
statement_list RBRACE statement_list {printf("Function NO: %d\n",$$); $$ =
create_Node("function_declaration", -1, "NULL", 9, $1,$2,$3, $4, $5, $6, $7, $8, $9); };


parameter_list: datatype identifier COMMA parameter_list { $$ =
create_Node("parameter_list", -1, "NULL",4, $1, $2, $3, $4); }

 | datatype identifier { $$ = create_Node("parameter_list", -1, "NULL",2, $1, $2);}

| EOL {$$ = NULL;}

| { $$ = NULL; };



statement_list:

        statement statement_list {printf("statement_list\n"); $$ =
create_Node("statement_list", -1, "NULL", 2, $1, $2);}

|       EOL statement_list { $$ = create_Node("statement_list", -1, "NULL", 1, $2);}

|       EOL {$$ = NULL;}

|

;


statement: declaration_statement {printf("declaration_statement\n"); $$ =
create_Node("declaration_statement", -1, "NULL",0); }

| assignment_statement SEMICOLON {printf("assignment_statement\n"); $$ =
create_Node("assignment_statement", -1,"NULL",2, $1, $2);}

| for_statement {printf("for_statement\n"); $$ = create_Node("for_statement", -1,
"NULL",1,$1); }

| if_statement {printf("if_statement\n"); $$ = create_Node("if_statement", -1,
"NULL",1,$1); }
```

```
| if_else_statement {printf("if_else_statement\n"); $$ =
create_Node("if_else_statement", -1, "NULL",1,$1); }

| while_statement {printf("while_statement\n"); $$ = create_Node("while_statement", -1,
"NULL",1,$1); }

| cout_statement {printf("cout_statement\n"); $$ = create_Node("cout_statement", -1,
"NULL",1,$1); }

| cin_statement {printf("cin_statement\n"); $$ = create_Node("cin_statement", -1,
"NULL",1,$1); }

| return_statement {printf("return_statement\n"); $$ = create_Node("return_statement",
-1, "NULL",1,$1); }

| error SEMICOLON { $$ = create_Node("error", -1, "NULL",0); }

| EOL { $$ = NULL; }

;


if_statement: if_x LPAREN E RPAREN LBRACE statement_list RBRACE { $$ =
create_Node("if_statement", -1,"NULL", 7,$1,$2,$3,$4,$5,$6,$7);}

| if_x LPAREN E RPAREN  statement { $$ = create_Node("if_statement", -1,"NULL",
5,$1,$2,$3,$4,$5);}

| if_statement EOL  { $$ = create_Node("if_statement", -1,"NULL", 1,$1);}


else_statement: else_x LBRACE statement_list RBRACE { $$ =
create_Node("else_statement", -1, "NULL", 3, $1, $2, $3); }

 | else_x statement { $$ = create_Node("else_statement", -1, "NULL", 2, $1, $2); }

 | else_statement EOL { $$ = create_Node("else_statement", -1, "NULL", 1, $1); }

 ;


if_else_statement: if_statement else_statement { $$ =
create_Node("if_else_statement", -1, "NULL", 2, $1, $2); };


while_statement: while_x LPAREN E RPAREN LBRACE statement_list RBRACE { $$ =
create_Node("while_statement", -1, "NULL", 7, $1, $2, $3, $4, $5, $6, $7); };
```

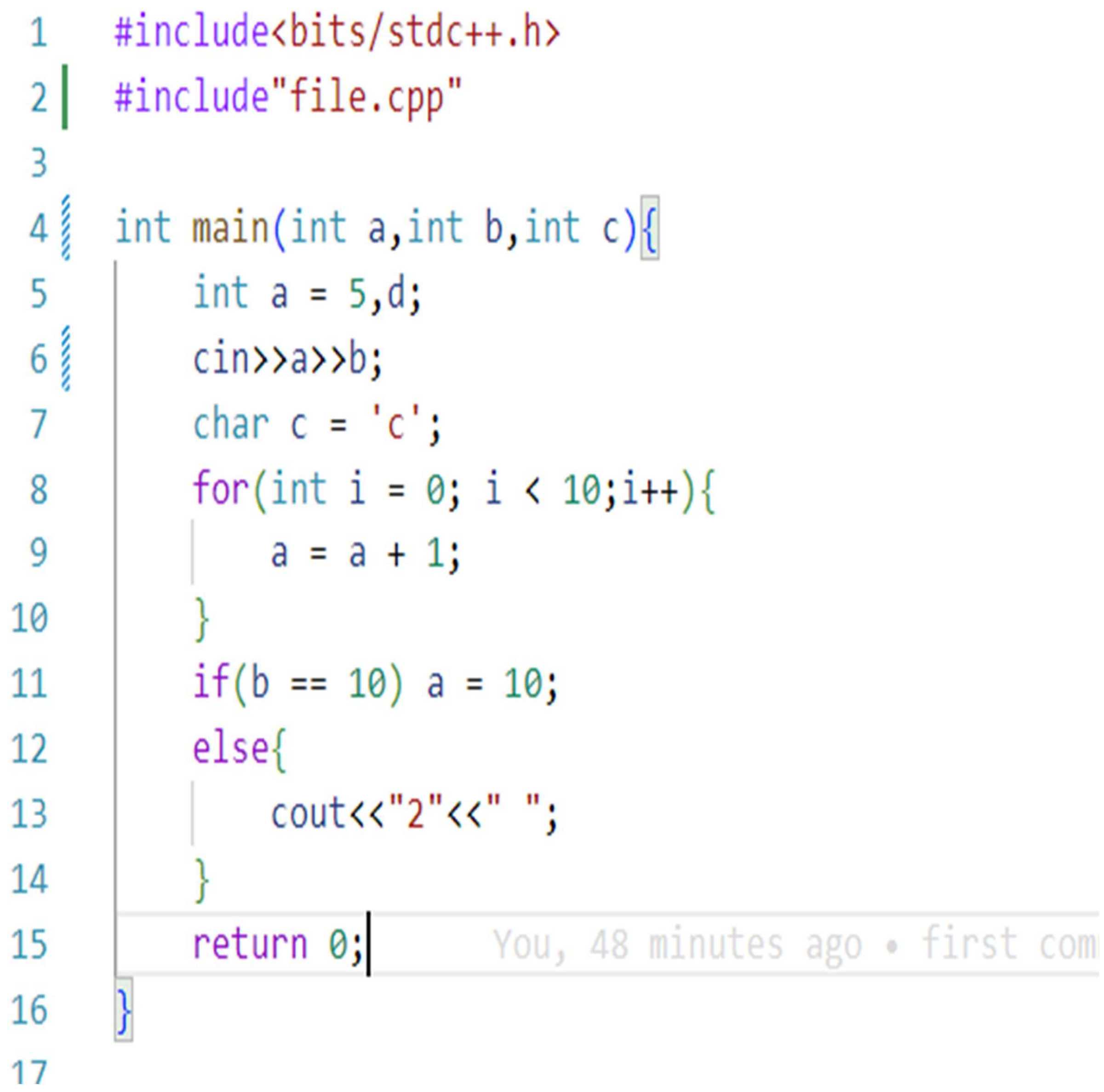
```
for_statement: for_x LPAREN declaration_statement E SEMICOLON E RPAREN LBRACE
statement_list RBRACE { $$ = create_Node("for_statement", -1, "NULL", 9, $1, $2, $3, $4,
$5, $6, $7, $8, $9); };

return_statement: return_x E SEMICOLON { $$ = create_Node("return_statement", -
1,"NULL", 2, $1, $2); }

| return_x SEMICOLON { $$ = create_Node("return_statement", -1, "NULL", 2, $1,$2); };

cout_statement: cout insert_statement SEMICOLON { $$ =
create_Node("cout_statement", -1,"NULL", 2, $1, $2); };

insert_statement: insert E insert_statement { $$ = create_Node("insert_statement", -
1,"NULL", 3, $1, $2, $3); }

| insert string insert_statement { $$ = create_Node("insert_statement", -1, "NULL", 3, $1,
$2, $3); }

| insert E { $$ = create_Node("insert_statement", -1, "NULL", 2, $1, $2); }

| insert string { $$ = create_Node("insert_statement", -1, "NULL", 2, $1, $2); }

cin_statement: cin extract_statement SEMICOLON { $$ = create_Node("cin_statement",
-1, "NULL", 3, $1, $2, $3); };

extract_statement: extract identifier extract_statement { $$ =
create_Node("extract_statement", -1, "NULL", 3, $1, $2, $3); }

| extract identifier { $$ = create_Node("extract_statement", -1, "NULL", 2, $1, $2); }

;

declaration_statement: datatype id_list SEMICOLON { $$ =
create_Node("declaration_statement", -1, "NULL", 3, $1, $2, $3); };

id_list: identifier COMMA id_list { $$ = create_Node("id_list", -1, "NULL", 3, $1, $2, $3); }

| assignment_statement COMMA id_list { $$ = create_Node("id_list", -1, "NULL", 3, $1,
$2, $3); }

| assignment_statement { $$ = create_Node("id_list", -1, "NULL", 1, $1); }
| identifier { $$ = create_Node("id_list", -1, "NULL", 1, $1); }


assignment_statement:
  F assignmentop E  { printf("assignment-statement\n");$$ =
create_Node("assignment_statement", -1, "NULL", 3, $1, $2, $3); $1->value = $3->value;
}
;


E: F assignmentop E  { $$ = create_Node("E", -1,  "NULL", 3, $1, $2, $3); $1->value = $3-
>value; }
|          E comparisionop T { $$ = create_Node("E", -1,  "NULL", 3, $1, $2, $3);
                    if (strcmp($2->value_str, "<=") == 0) {
                    $$->value = $1->value <= $3->value;
          } else if (strcmp($2->value_str, ">=") == 0) {
                    $$->value = $1->value >= $3->value;
          } else if (strcmp($2->value_str, "==") == 0) {
                    $$->value = $1->value == $3->value;
          } else if (strcmp($2->value_str, "!=") == 0) {
                    $$->value = $1->value != $3->value;
          }
          else if (strcmp($2->value_str, "<") == 0) {
                    $$->value = $1->value < $3->value;
          }
          else if (strcmp($2->value_str, ">") == 0) {
                    $$->value = $1->value > $3->value;
          }
}
|  E PLUS T { $$ = create_Node("E", $1->value+$3->value, "NULL", 3, $1, $2, $3); }
|  E MINUS T { $$ = create_Node("E", $1->value-$3->value, "NULL", 3, $1, $2, $3); }
|  T { $$ = create_Node("E", $1->value, "NULL", 1, $1); }
```

```
;


T:
        T MUL F { $$ = create_Node("T", $1->value*$3->value, "NULL", 3, $1, $2, $3);}
|       T DIV F { $$ = create_Node("T", $1->value/$3->value, "NULL", 3, $1, $2, $3);}
|       F { $$ = create_Node("T", $1->value, "NULL", 1, $1); }
;


F:
        number { $$ = create_Node("F", -1, "NULL" , 1, $1); }
| character { $$ = create_Node("F", -1, "NULL", 1, $1); }
|       LPAREN E RPAREN { $$ = create_Node("F",$2->value, "NULL", 3, $1, $2, $3); }
|  identifier { $$ = create_Node("identifier", -1, "NULL", 1, $1); }
| unary identifier { $$ = create_Node("F", -1, "NULL", 2, $1, $2); }
| identifier unary { $$ = create_Node("F", -1, "NULL", 2, $1, $2); }
;
%%
#include <ctype.h>
int yyerror(const char *s)
{
        extern int yylineno;
        // valid = 0;
        if(yylineno != 27){
                printf("Line no: %d \n The error is: %s\n",yylineno,s);
        }
}
extern FILE *yyin;
int main(int argc,char **argv){
        if(argc<2)
```

```
	{
		printf("Usage: %s <filename>\n",argv[0]);
		return 1;
	}
	FILE *fp = fopen(argv[1], "r");
	if(fp==NULL)
	{
		printf("Error: File not found\n");
		return 1;
	}
	extern char *yytext;
	yyin=fp;
	yyparse();
	printf("---------------\n");
	printTree(head,0);
	printf("---------------\n");
	display();
}
```

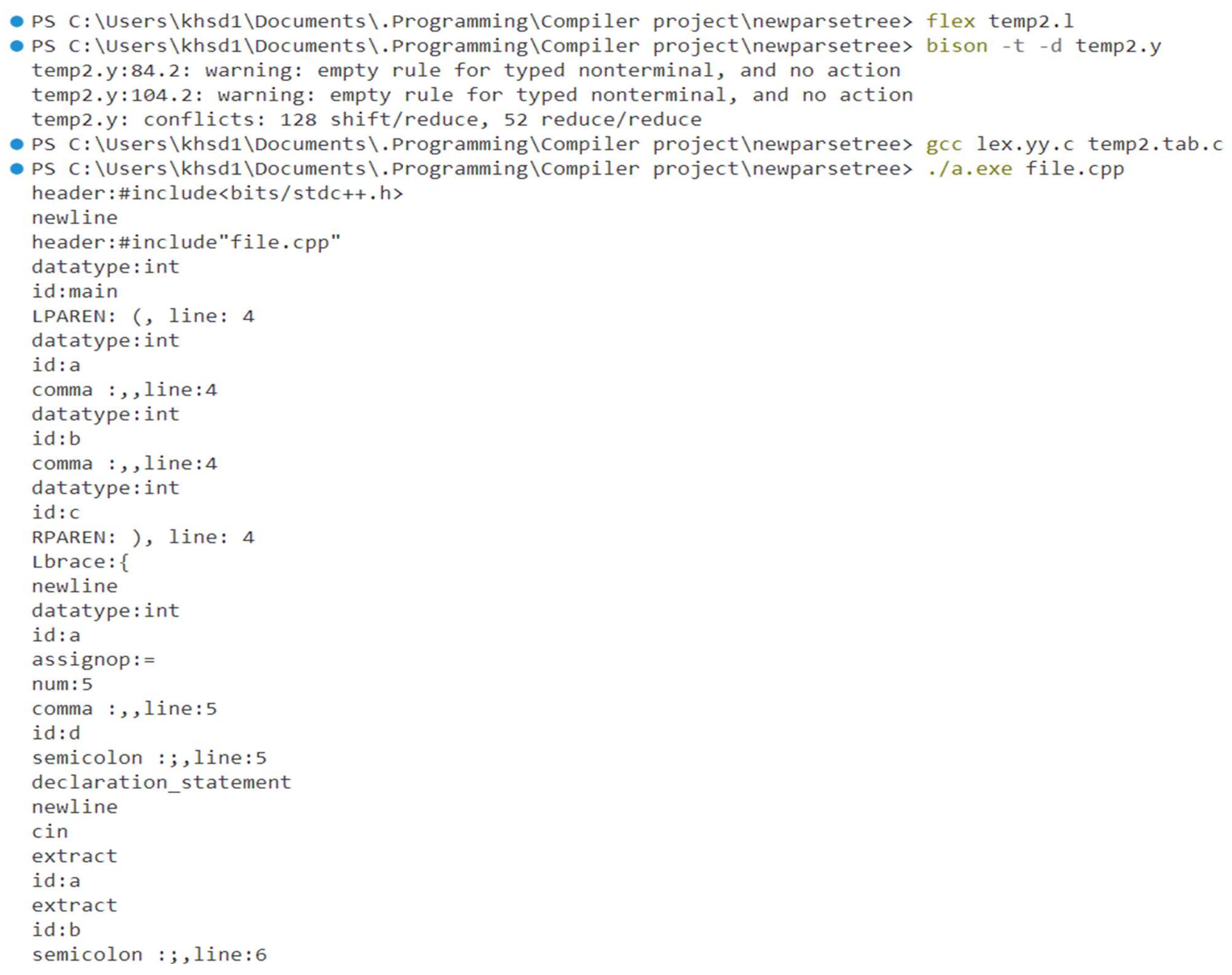
2. Show that this parser correctly parses the input token generated by your lexical analyser for any program written in your programming language as well as identifies errors.

Sample Code A:

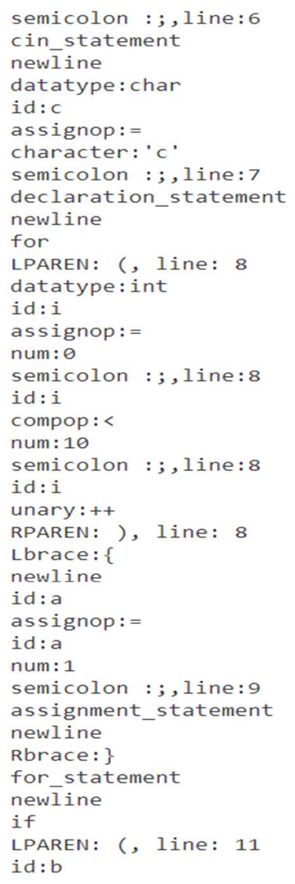


```
#include<bits/stdc++.h>
#include"file.cpp"

int main(int a,int b,int c){
    int a = 5,d;
    cin>>a>>b;
    char c = 'c';
    for(int i = 0; i < 10;i++){
        a = a + 1;
    }
    if(b == 10) a = 10;
    else{
        cout<<"2"<<" ";
    }
    return 0;        You, 48 minutes ago • first com
}

```

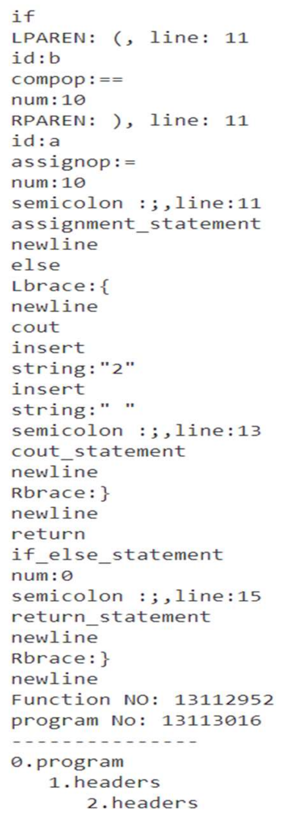
Output Parse Tree:



```
● PS C:\Users\khsd1\Documents\.Programming\Compiler project\newparsetree> flex temp2.l
● PS C:\Users\khsd1\Documents\.Programming\Compiler project\newparsetree> bison -t -d temp2.y
  temp2.y:84.2: warning: empty rule for typed nonterminal, and no action
  temp2.y:104.2: warning: empty rule for typed nonterminal, and no action
  temp2.y: conflicts: 128 shift/reduce, 52 reduce/reduce
● PS C:\Users\khsd1\Documents\.Programming\Compiler project\newparsetree> gcc lex.yy.c temp2.tab.c
● PS C:\Users\khsd1\Documents\.Programming\Compiler project\newparsetree> ./a.exe file.cpp
  header:#include<bits/stdc++.h>
  newline
  header:#include"file.cpp"
  datatype:int
  id:main
  LPAREN: (, line: 4
  datatype:int
  id:a
  comma :,,line:4
  datatype:int
  id:b
  comma :,,line:4
  datatype:int
  id:c
  RPAREN: ), line: 4
  Lbrace:{
  newline
  datatype:int
  id:a
  assignop:=
  num:5
  comma :,,line:5
  id:d
  semicolon :;,line:5
  declaration_statement
  newline
  cin
  extract
  id:a
  extract
  id:b
  semicolon :;,line:6
```



```
semicolon :;,line:6
cin_statement
newline
datatype:char
id:c
assignop:=
character:'c'
semicolon :;,line:7
declaration_statement
newline
for
LPAREN: (, line: 8
datatype:int
id:i
assignop:=
num:0
semicolon :;,line:8
id:i
compop:<
num:10
semicolon :;,line:8
id:i
unary:++
RPAREN: ), line: 8
Lbrace:{
newline
id:a
assignop:=
id:a
num:1
semicolon :;,line:9
assignment_statement
newline
Rbrace:}
for_statement
newline
if
LPAREN: (, line: 11
id:b
```

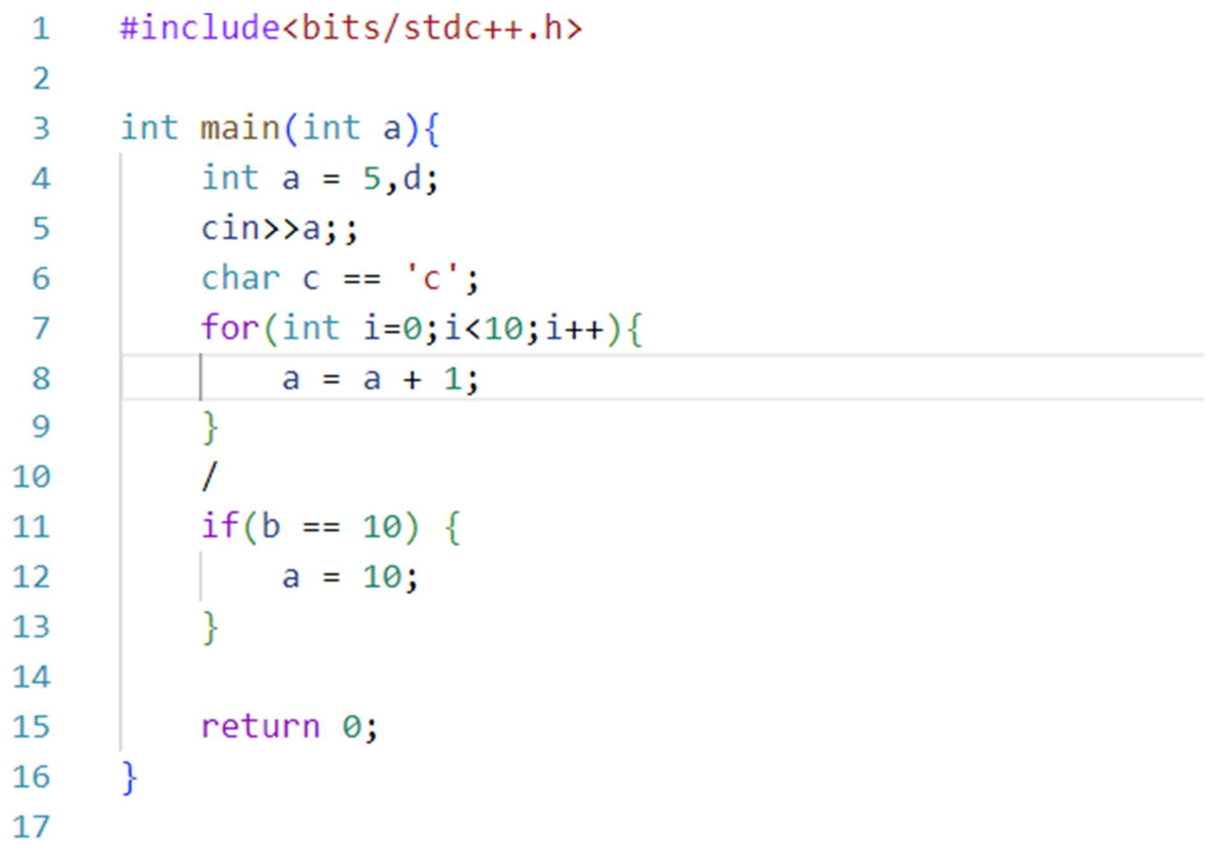


```
if
LPAREN: (, line: 11
id:b
compop:==
num:10
RPAREN: ), line: 11
id:a
assignop:=
num:10
semicolon :;,line:11
assignment_statement
newline
else
Lbrace:{
newline
cout
insert
string:"2"
insert
string:" "
semicolon :;,line:13
cout_statement
newline
Rbrace:}
newline
return
if_else_statement
num:0
semicolon :;,line:15
return_statement
newline
Rbrace:}
newline
Function NO: 13112952
program No: 13113016
---------------
0.program
   1.headers
      2.headers
      2.headers
         3.#include<bits/stdc++.h>
      2.headers
         3.#include"file.cpp"
   1.function_declaration
      2.int
      2.main
      2.(
      2.parameter_list
         3.int
         3.a
         3.,
         3.parameter_list
            4.int
            4.b
            4.,
            4.parameter_list
               5.int
               5.c
      2.)
      2.{
      2.statement_list
         3.statement_list
            4.declaration_statement
            4.statement_list
               5.statement_list
                  6.cin_statement
                     7.cin_statement
                        8.cin
                        8.extract_statement
                           9.>>
                           9.a
                           9.extract_statement
                              10.>>
                              10.b
                        8.;
                  6.statement_list
                     7.statement_list
                        8.declaration statement
```

```
8.declaration_statement
8.statement_list
   9.statement_list
      10.for_statement
         11.for_statement
            12.for
            12.(
            12.declaration_statement
               13.int
               13.id_list
                  14.assignment_statement
                     15.identifier
                        16.i
                     15.=
                     15.E
                        16.T
                           17.F
                              18.(number,0)
               13.;
            12.(E,0)
               13.E
                  14.T
                     15.identifier
                        16.i
               13.<
               13.T
                  14.F
                     15.(number,10)
            12.;
            12.E
               13.T
                  14.F
                     15.i
                     15.++
            12.)
            12.{
            12.statement_list
               13.statement_list
                  14.assignment statement

                  14.assignment_statement
                     15.assignment_statement
                        16.(identifier,-2)
                           17.a
                        16.=
                        16.(E,-2)
                           17.E
                              18.T
                                 19.identifier
                                    20.a
                           17.+
                           17.T
                              18.F
                                 19.(number,1)
                     15.;
      10.statement_list
         11.statement_list
            12.if_else_statement
               13.if_else_statement
                  14.if_statement
                     15.if_statement
                        16.if
                        16.(
                        16.(E,1)
                           17.E
                              18.T
                                 19.identifier
                                    20.b
                           17.==
                           17.T
                              18.F
                                 19.(number,10)
                        16.)
                        16.assignment_statement
                           17.assignment_statement
                              18.identifier
                                 19.a
                              18.=
                              18.E
```

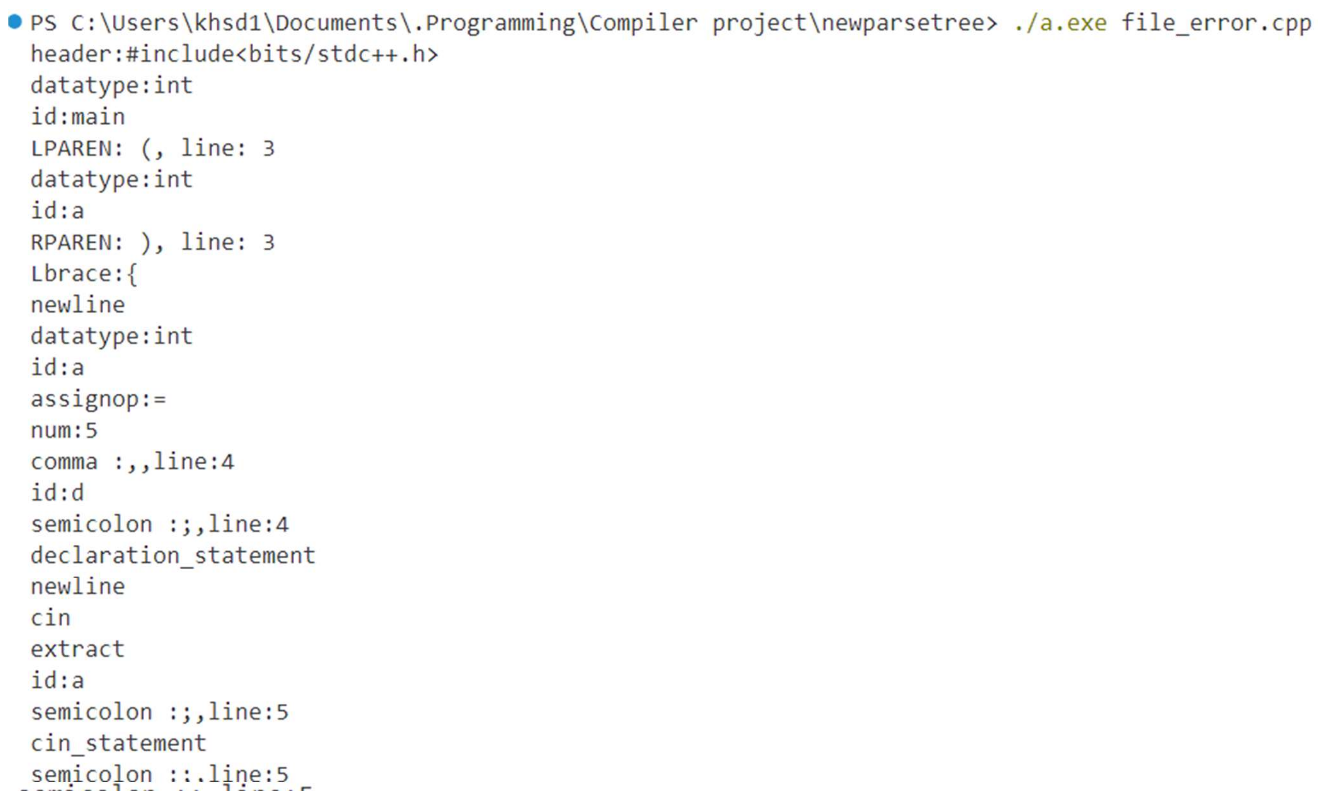
```
                                18.=
                                18.E
                                  19.T
                                    20.F
                                      21.(number,10)
                              17.;
                        14.else_statement
                          15.else_statement
                            16.else
                            16.{
                            16.statement_list
                              17.statement_list
                                18.cout_statement
                                  19.cout_statement
                                    20.cout
                                    20.insert_statement
                                      21.<<
                                      21."2"
                                      21.insert_statement
                                        22.<<
                                        22." "
                    12.statement_list
                      13.return_statement
                        14.return_statement
                          15.return
                          15.E
                            16.T
                              17.F
                                18.(number,0)
2.}
```

Sample Code B: (For Error detection)

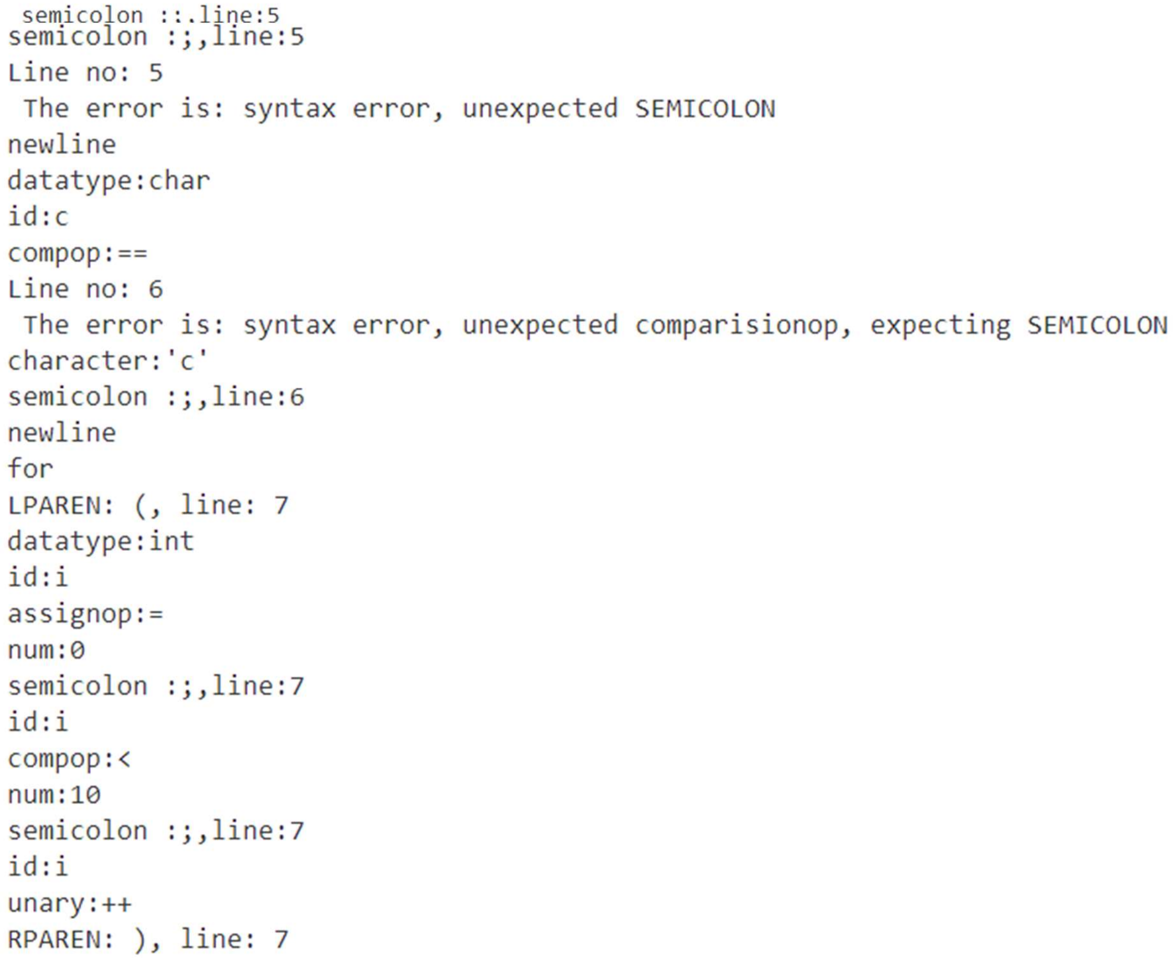


```cpp
#include<bits/stdc++.h>

int main(int a){
    int a = 5,d;
    cin>>a;;
    char c == 'c';
    for(int i=0;i<10;i++){
        a = a + 1;
    }
    /
    if(b == 10) {
        a = 10;
    }

    return 0;
}

```

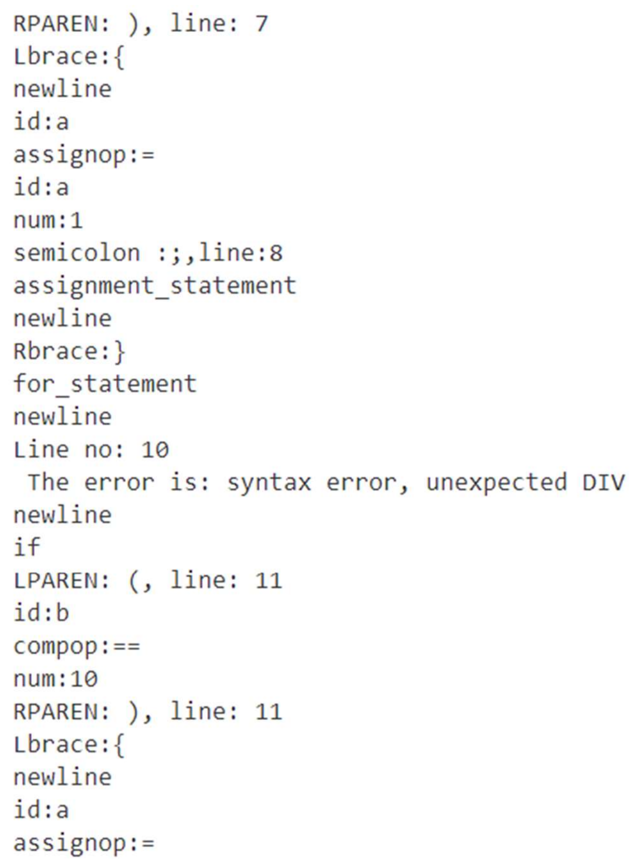
Erros Detecting Output:



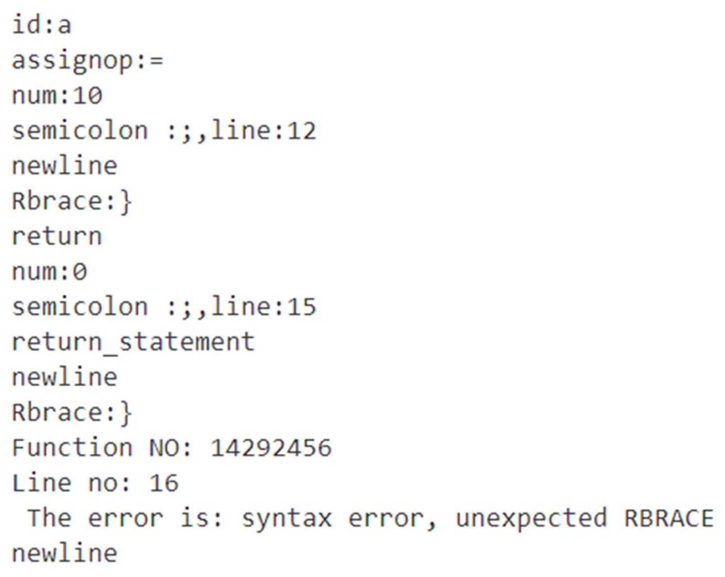
```
PS C:\Users\khsd1\Documents\.Programming\Compiler project\newparsetree> ./a.exe file_error.cpp
header:#include<bits/stdc++.h>
datatype:int
id:main
LPAREN: (, line: 3
datatype:int
id:a
RPAREN: ), line: 3
Lbrace:{
newline
datatype:int
id:a
assignop:=
num:5
comma :,,line:4
id:d
semicolon :;,line:4
declaration_statement
newline
cin
extract
id:a
semicolon :;,line:5
cin_statement
semicolon :;,line:5
```



```
semicolon :;,line:5
Line no: 5
 The error is: syntax error, unexpected SEMICOLON
newline
datatype:char
id:c
compop:==
Line no: 6
 The error is: syntax error, unexpected comparisionop, expecting SEMICOLON
character:'c'
semicolon :;,line:6
newline
for
LPAREN: (, line: 7
datatype:int
id:i
assignop:=
num:0
semicolon :;,line:7
id:i
compop:<
num:10
semicolon :;,line:7
id:i
unary:++
RPAREN: ), line: 7
```



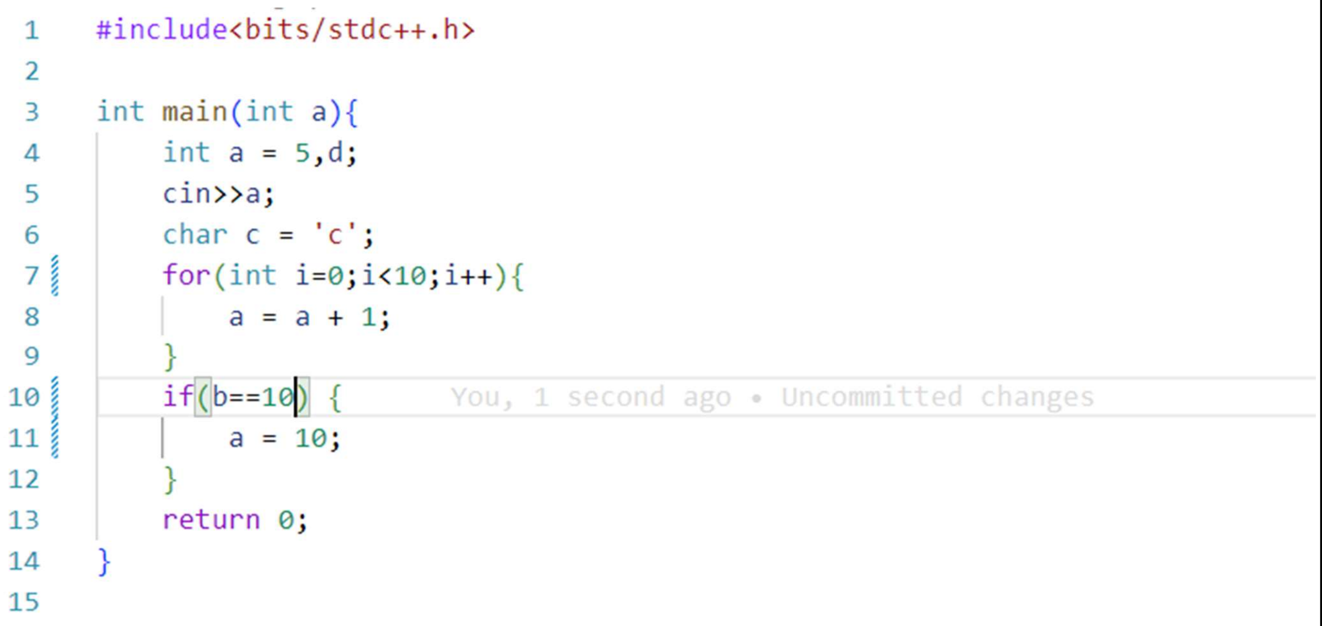
```
RPAREN: ), line: 7
Lbrace:{
newline
id:a
assignop:=
id:a
num:1
semicolon :;,line:8
assignment_statement
newline
Rbrace:}
for_statement
newline
Line no: 10
 The error is: syntax error, unexpected DIV
newline
if
LPAREN: (, line: 11
id:b
compop:==
num:10
RPAREN: ), line: 11
Lbrace:{
newline
id:a
assignop:=
```



```
id:a
assignop:=
num:10
semicolon :;,line:12
newline
Rbrace:}
return
num:0
semicolon :;,line:15
return_statement
newline
Rbrace:}
Function NO: 14292456
Line no: 16
 The error is: syntax error, unexpected RBRACE
newline
```

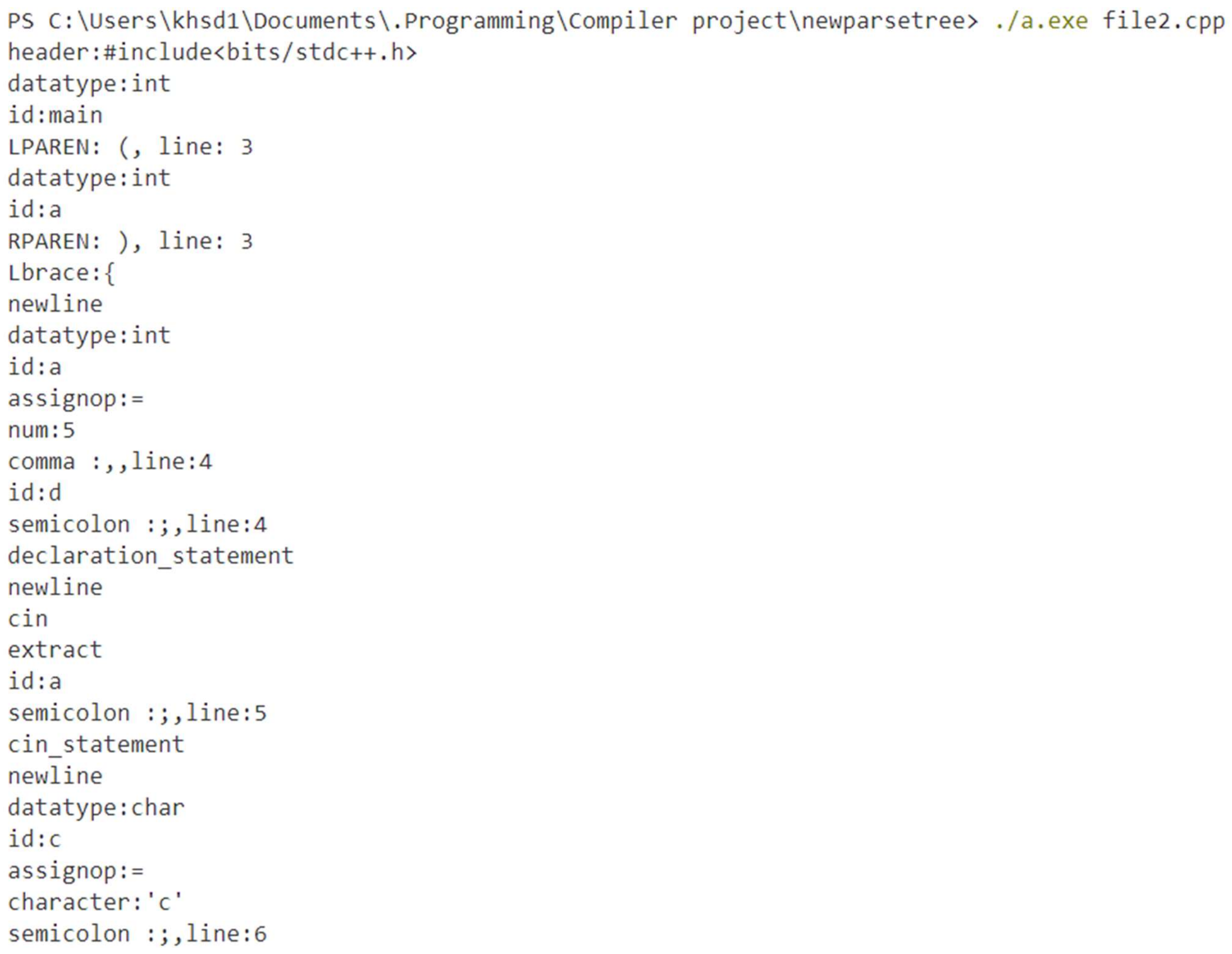
3. Write a simple program in your language with all kinds of tokens and keywords and show that your compiler is correctly detecting the tokens and errors. Parse the program using your

parser. Print step by step parsing process and draw the parse tree.

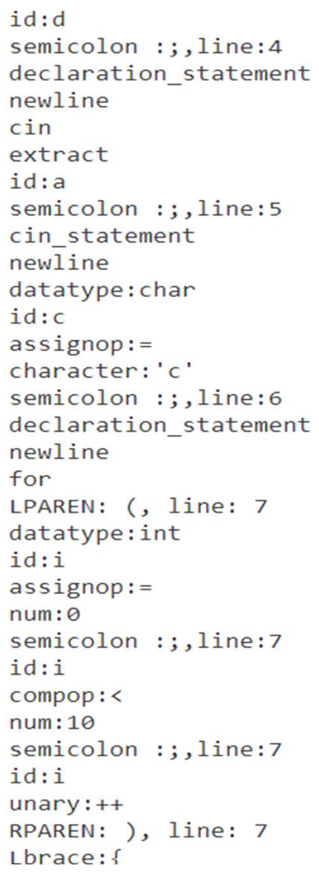


```
#include<bits/stdc++.h>

int main(int a){
    int a = 5,d;
    cin>>a;
    char c = 'c';
    for(int i=0;i<10;i++){
        a = a + 1;
    }
    if(b==10) {         You, 1 second ago • Uncommitted changes
        a = 10;
    }
    return 0;
}

```

Output:

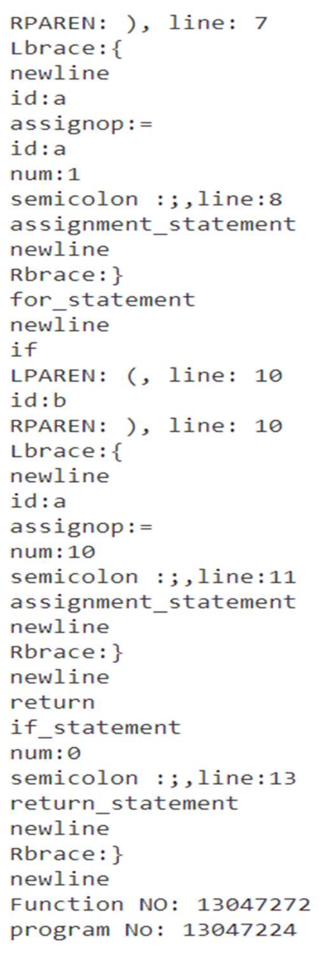


```
PS C:\Users\khsd1\Documents\.Programming\Compiler project\newparsetree> ./a.exe file2.cpp
header:#include<bits/stdc++.h>
datatype:int
id:main
LPAREN: (, line: 3
datatype:int
id:a
RPAREN: ), line: 3
Lbrace:{
newline
datatype:int
id:a
assignop:=
num:5
comma :,,line:4
id:d
semicolon :;,line:4
declaration_statement
newline
cin
extract
id:a
semicolon :;,line:5
cin_statement
newline
datatype:char
id:c
assignop:=
character:'c'
semicolon :;,line:6
```



```
id:d
semicolon :;,line:4
declaration_statement
newline
cin
extract
id:a
semicolon :;,line:5
cin_statement
newline
datatype:char
id:c
assignop:=
character:'c'
semicolon :;,line:6
```

```
declaration_statement
newline
for
LPAREN: (, line: 7
datatype:int
id:i
assignop:=
num:0
semicolon :;,line:7
id:i
compop:<
num:10
semicolon :;,line:7
id:i
unary:++
RPAREN: ), line: 7
Lbrace:{
```



```
RPAREN: ), line: 7
Lbrace:{
newline
id:a
assignop:=
id:a
num:1
semicolon :;,line:8
assignment_statement
newline
Rbrace:}
for_statement
newline
if
LPAREN: (, line: 10
id:b
RPAREN: ), line: 10
Lbrace:{
newline
id:a
assignop:=
num:10
semicolon :;,line:11
assignment_statement
newline
Rbrace:}
newline
return
if_statement
num:0
semicolon :;,line:13
return_statement
newline
Rbrace:}
newline
Function NO: 13047272
program No: 13047224
```

```
0.program
   1.headers
      2.#include<bits/stdc++.h>
   1.function_declaration
      2.int
      2.main
      2.(
      2.parameter_list
         3.int
         3.a
      2.)
      2.{
      2.statement_list
         3.statement_list
            4.declaration_statement
            4.statement_list
               5.statement_list
                  6.cin_statement
                     7.cin_statement
                        8.cin
                        8.extract_statement
                           9.>>
                           9.a
                        8.;
                  6.statement_list
                     7.statement_list
                        8.declaration_statement
                        8.statement_list
                           9.statement_list
                              10.for_statement
                                 11.for_statement
                                    12.for
                                    12.(

                                    12.for
                                    12.(
                                    12.declaration_statement
                                       13.int
                                       13.id_list
                                          14.assignment_statement
                                             15.identifier
                                                16.i
                                             15.=
                                             15.E
                                                16.T
                                                   17.F
                                                      18.(number,0)
                                       13.;
                                    12.(E,0)
                                       13.E
                                          14.T
                                             15.identifier
                                                16.i
                                       13.<
                                       13.T
                                          14.F
                                             15.(number,10)
                                    12.;
                                    12.E
                                       13.T
                                          14.F
                                             15.i
                                             15.++
                                    12.)
```

```
                                        12.)
                                        12.{
                                        12.statement_list
                                           13.statement_list
                                              14.assignment_statement
                                                 15.assignment_statement
                                                    16.(identifier,-2)
                                                       17.a
                                                    16.=
                                                    16.(E,-2)
                                                       17.E
                                                          18.T
                                                             19.identifier
                                                                20.a
                                                       17.+
                                                       17.T
                                                          18.F
                                                             19.(number,1)
                                                 15.;
                                  10.statement_list
                                     11.statement_list
                                        12.if_statement
                                           13.if_statement
                                              14.if_statement
                                                 15.if
                                                 15.(
                                                 15.E
                                                    16.T
                                                       17.identifier
                                                          18.b
                                                 15.)
                                                 15.{
                                                 15.statement_list
                                                    16.statement_list
                                                       17.assignment_statement
                                                          18.assignment_statement

                                                        17.assignment_statement
                                                           18.assignment_statement
                                                              19.identifier
                                                                 20.a
                                                              19.=
                                                              19.E
                                                                 20.T
                                                                    21.F
                                                                       22.(number,10)
                                                           18.;
                                                  15.}
                                         12.statement_list
                                            13.return_statement
                                               14.return_statement
                                                  15.return
                                                  15.E
                                                     16.T
                                                        17.F
                                                           18.(number,0)
            2.}
```