

**Department of CSE**

**Compiler Lab (CSE 352)**

**Lab Report 06**

Construction of Parsing Table for a Grammar of Top Down Parser.

**Submitted By :**

**Fahima Abida Chowdhury**

ID: 0432220005101135

Semester: Spring-2025

Batch: 52(6B1)

**Submitted To :**

#### [Md. Ismail](https://uits.edu.bd/md-ismail/)

Lecturer

Department of CSE

University of Information Technology and Sciences

**Experiment No.: 06**

**Experiment Name:** Construction of Parsing Table for a Grammar of Top Down Parser.

**Problem Statement**

The goal of this lab is to design and implement a C program to construct a parsing table for a given grammar that is suitable for an LL(1) top-down parser. The parsing table helps guide the parser in syntactic analysis by providing the appropriate production rule based on the current non-terminal and the lookahead terminal.

**Grammar Used**

The grammar for which the parsing table is constructed is:

E → T E'

E' → + T E' | ε

T → F T'

T' → \* F T' | ε

F → id | ( E )

**3. Implementation**

**C Code**

#include <stdio.h>

#include <string.h>

#define NON\_TERMINALS 5

#define TERMINALS 6

char \*nonTerminals[NON\_TERMINALS] = {"E", "E'", "T", "T'", "F"};

char \*terminals[TERMINALS] = {"id", "+", "\*", "(", ")", "$"};

char parsingTable[NON\_TERMINALS][TERMINALS][20];

void constructParsingTable() {

int i, j;

for (i = 0; i < NON\_TERMINALS; i++) {

for (j = 0; j < TERMINALS; j++) {

strcpy(parsingTable[i][j], "error");

}

}

if (1) {

strcpy(parsingTable[0][0], "T E'");

strcpy(parsingTable[0][3], "T E'");

}

if (1) {

strcpy(parsingTable[1][1], "+ T E'");

strcpy(parsingTable[1][4], "ε");

strcpy(parsingTable[1][5], "ε");

}

if (1) {

strcpy(parsingTable[2][0], "F T'");

strcpy(parsingTable[2][3], "F T'");

}

if (1) {

strcpy(parsingTable[3][2], "\* F T'");

strcpy(parsingTable[3][1], "ε");

strcpy(parsingTable[3][4], "ε");

strcpy(parsingTable[3][5], "ε");

}

if (1) {

strcpy(parsingTable[4][0], "id");

strcpy(parsingTable[4][3], "( E )");

}

}

void printParsingTable() {

int i, j;

printf("LL(1) Parsing Table:\n\n\t");

for (j = 0; j < TERMINALS; j++) {

printf("%s\t", terminals[j]);

}

printf("\n");

for (i = 0; i < NON\_TERMINALS; i++) {

printf("%s\t", nonTerminals[i]);

for (j = 0; j < TERMINALS; j++) {

printf("%s\t", parsingTable[i][j]);

}

printf("\n");

}

}

int main() {

constructParsingTable();

printParsingTable();

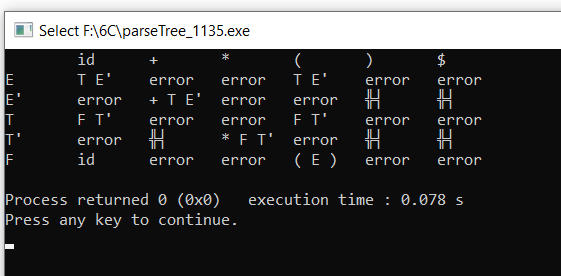
return 0;

}

**Input**

This program uses hardcoded grammar rules and terminal/non-terminal sets, so no external input is needed. The grammar is fixed within the code.

**Output**

****

**Conclusion**

This lab report successfully demonstrates the construction of a parsing table for an LL(1)grammar using a C program. The table shows which production rules to apply for each combination of non-terminal and lookahead terminal. This table is fundamental in implementing a predictive parser that performs syntactic analysis without backtracking.

The code is modular and easily extendable to other LL(1) grammars by updating the grammar rules and terminal/non-terminal lists. Understanding how parsing tables work is crucial for compiler design and syntax-directed translation.