**Name**: Karthikeyan A

**Reg No**: 3122 21 5001 041

—----------------------------------------------------------------------------------------------------------------------------

**Assignment-03: Implementation of Recursive Decent Parser** -------------------------------------------------------------------------------------------------------------------------------

**Aim:**

Implement Recursive Decent Parser for the given grammar (left recursion eliminated)

**Code:**

**parser.c**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

char input[100];

int pos = 0;

// Forward declarations for each non-terminal function

void E();

void EPrime();

void T();

void TPrime();

void F();

void error()

{

printf("Error: syntax error\n");

exit(1);

}

// Check if the current character matches the expected token

void match(char expected)

{

if (input[pos] == expected)

{

pos++;

}

else

{

error();

}

}

// Non-terminal functions for each rule

void E()

{

T();

EPrime();

}

void EPrime()

{

if (input[pos] == '+')

{

match('+');

T();

EPrime();

}

// epsilon case - do nothing

}

void T()

{

F();

TPrime();

}

void TPrime()

{

if (input[pos] == '\*')

{

match('\*');

F();

TPrime();

}

// epsilon case - do nothing

}

void F()

{

if (input[pos] == '(')

{

match('(');

E();

match(')');

}

else if (strncmp(&input[pos], "id", 2) == 0)

{

pos += 2; // Advance position for "id"

}

else

{

error();

}

}

int main()

{

printf("Implementation of Recursive Descent Parser\n");

printf("Enter expression: ");

scanf("%s", input);

pos = 0;

E();

// Check if we've reached the end of input

if (input[pos] == '\0')

{

printf("Input is valid.\n");

}

else

{

printf("Input is invalid.\n");

}

return 0;

}

**parser.y**

%{

#include <stdio.h>

#include <stdlib.h>

void yyerror(char \*s);

int yylex(void);

extern int yy\_scan\_string(const char \*str);

%}

%token ID

%%

E : T EPrime ;

EPrime : '+' T EPrime | /\* epsilon \*/ ;

T : F TPrime ;

TPrime : '\*' F TPrime | /\* epsilon \*/ ;

F : '(' E ')' | ID ;

%%

void yyerror(char \*s) {

fprintf(stderr, "Error: %s\n", s);

}

int main(void) {

char input[100];

printf("Enter expression: ");

fgets(input, sizeof(input), stdin); // Read a full line of input

// Set the input for the lexer

yy\_scan\_string(input);

if (yyparse() == 0) {

printf("Input is valid.\n");

} else {

printf("Input is invalid.\n");

}

return 0;

}

**scanner.l**

%{

#include "y.tab.h"

int yywrap(void) { return 1; }

%}

%%

id { return ID; }

"+" { return '+'; }

"\*" { return '\*'; }

"(" { return '('; }

")" { return ')'; }

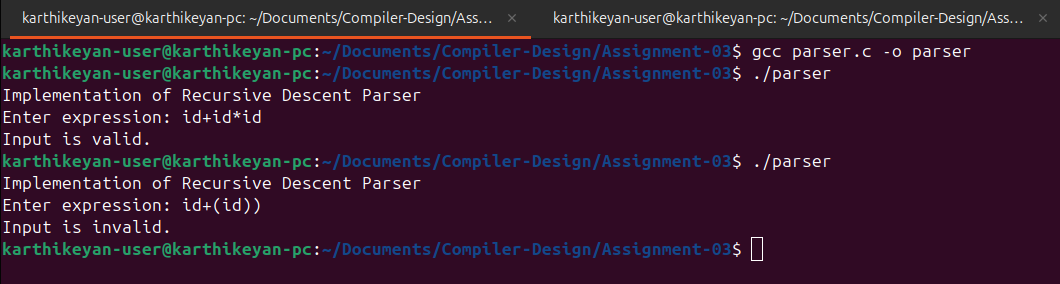
[ \t\n]+ ;

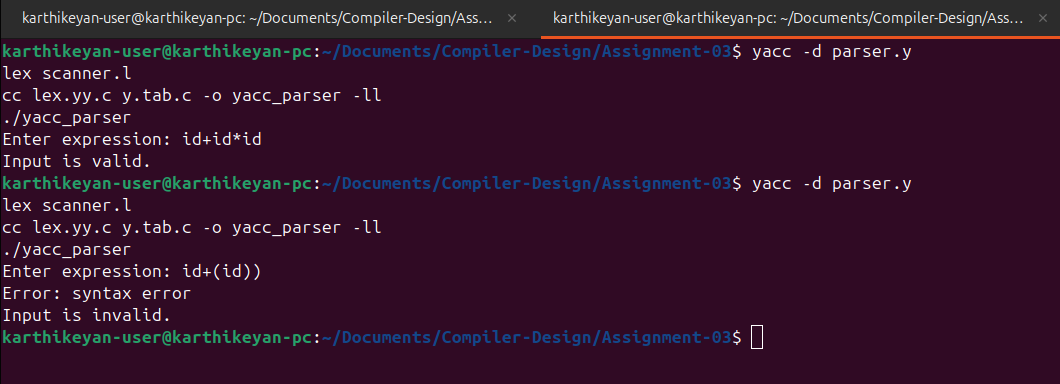
. { return yytext[0]; }

<<EOF>> { return 0; }

%%

**Output:**





**Learning Outcome**:

1. Understand Recursive Descent Parsing: Learn how recursive functions can parse input strings based on grammar rules.
2. Eliminate Left Recursion: Gain the ability to modify grammars to eliminate left recursion for top-down parsing.
3. Debugging and Error Handling: Learn to track the parsing process step-by-step and handle syntax errors.
4. Grammar and Parsing Relationship: Understand the connection between formal grammars and parsing techniques used in compilers