Ex. No.: 5 Date: 21-1-2026

Title: Implementation of Recursive Descent Parser

Aim: Write a program to implement Recursive Descent Parser for the given grammar.

Algorithm:

1. Skip whitespace characters.
2. Convert characters into tokens (id, operators, parentheses, end).
3. Set the first token as the lookahead.
4. Call parse_E (entry point).
5. In parse_E: parse T then E'.
6. In parse_E': if + or -, consume and parse T then E'; else ε.
7. In parse_T: parse F then T'.
8. In parse_T': if *, /, or %, consume and parse F then T'; else ε.
9. In parse_F: if id, consume; if (, consume, parse E, then expect).
10. If parsing succeeds and lookahead is END, accept.
11. Otherwise, reject.

Program:

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

typedef enum {
    TOK_ID, TOK_PLUS, TOK_MINUS, TOK_MUL, TOK_DIV, TOK_MOD,
    TOK_LPAREN, TOK_RPAREN, TOK_END, TOK_INVALID
} TokenType;

typedef struct {
    TokenType type;
    char lexeme[64];
} Token;

const char *input;
size_t pos;
Token lookahead;

/* --- Lexer --- */
void skip_ws() { while (isspace((unsigned char)input[pos]))
pos++; }

int is_ident_start(char c) { return isalpha((unsigned char)c)
|| c == '_'; }
int is_ident_char(char c) { return isalnum((unsigned char)c) ||
c == '_'; }
```

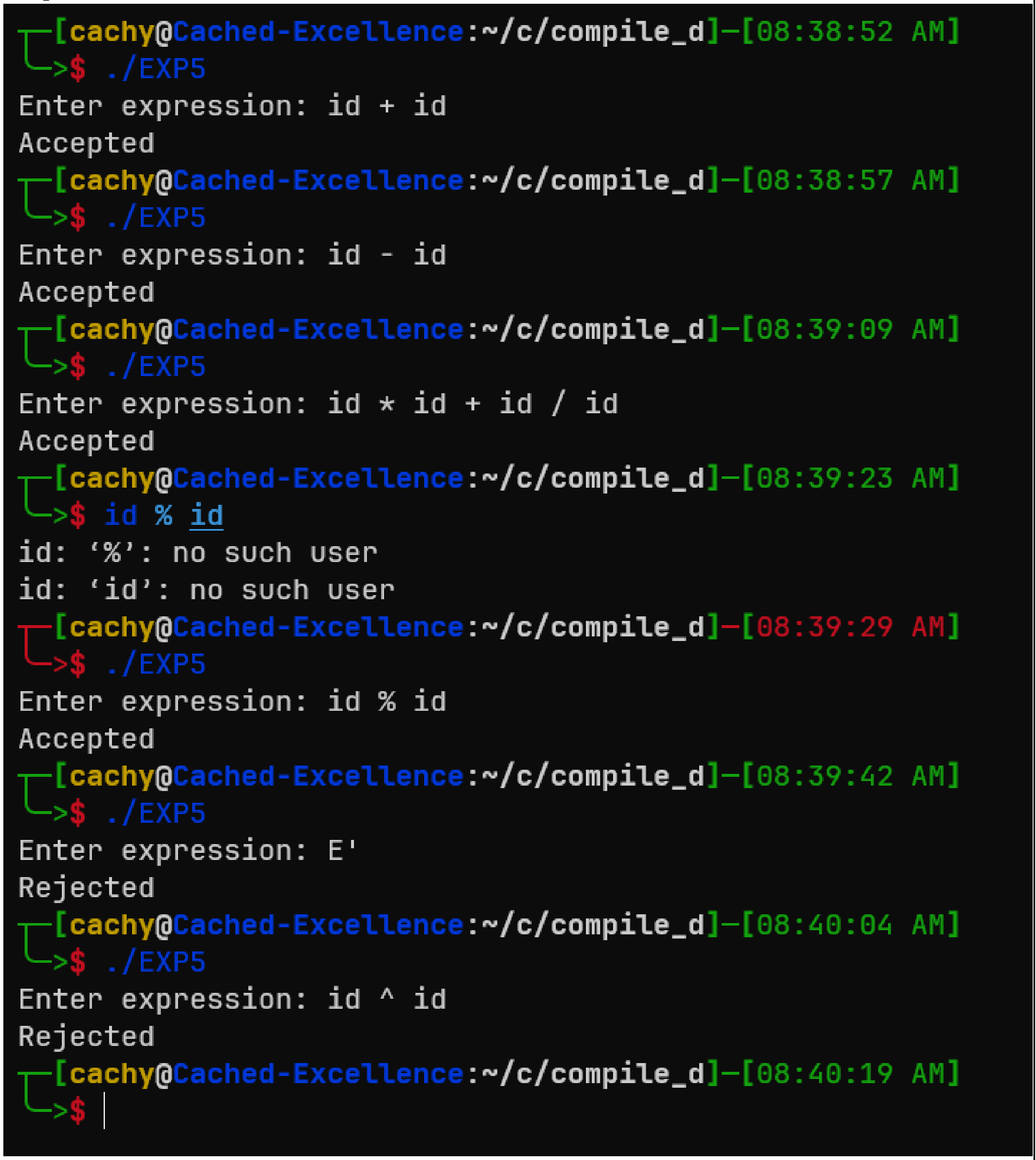
```
Token next_token() {
    skip_ws();
    Token tok = {TOK_INVALID, {0}};
    char c = input[pos];
    if (c == '\0') { tok.type = TOK_END; return tok; }

    if (is_ident_start(c)) {
        size_t start = pos++;
        while (is_ident_char(input[pos])) pos++;
        size_t len = pos - start;
        strncpy(tok.lexeme, input + start, len);
        tok.lexeme[len] = '\0';
        tok.type = TOK_ID;
        return tok;
    }

    switch (c) {
        case '+': tok.type = TOK_PLUS; break;
        case '-': tok.type = TOK_MINUS; break;
        case '*': tok.type = TOK_MUL; break;
        case '/': tok.type = TOK_DIV; break;
        case '%': tok.type = TOK_MOD; break;
        case '(': tok.type = TOK_LPAREN; break;
        case ')': tok.type = TOK_RPAREN; break;
        default:  tok.type = TOK_INVALID; break;
    }
    pos++;
    return tok;
}

void advance() { lookahead = next_token(); }
int match(TokenType t) { if (lookahead.type == t) { advance();
return 1; } return 0; }

/* --- Parser --- */
int parse_E(); int parse_Ep(); int parse_T(); int parse_Tp();
int parse_F();

int parse_E()  { return parse_T() && parse_Ep(); }
int parse_Ep() {
    if (lookahead.type == TOK_PLUS || lookahead.type ==
TOK_MINUS) {
        advance();
        return parse_T() && parse_Ep();
    }
    return 1; // epsilon
}
```

```
int parse_T()  { return parse_F() && parse_Tp(); }
int parse_Tp() {
    if (lookahead.type == TOK_MUL || lookahead.type == TOK_DIV
|| lookahead.type == TOK_MOD) {
        advance();
        return parse_F() && parse_Tp();
    }
    return 1; // epsilon
}
int parse_F() {
    if (lookahead.type == TOK_ID) { advance(); return 1; }
    if (lookahead.type == TOK_LPAREN) {
        advance();
        if (!parse_E()) return 0;
        return match(TOK_RPAREN);
    }
    return 0;
}

/* --- Driver --- */
int parse(const char *src) {
    input = src; pos = 0; advance();
    return parse_E() && lookahead.type == TOK_END;
}

int main() {
    char buffer[256];
    printf("Enter expression: ");
    fgets(buffer, sizeof(buffer), stdin);
    buffer[strcspn(buffer, "\n")] = '\0';

    if (parse(buffer))
        printf("Accepted\n");
    else
        printf("Rejected\n");
    return 0;
}
```

Result:
Program executed successfully and verified output.

Output:

Output Screenshot



```
┌─[cachy@Cached-Excellence:~/c/compile_d]─[08:38:52 AM]
└─>$ ./EXP5
Enter expression: id + id
Accepted
┌─[cachy@Cached-Excellence:~/c/compile_d]─[08:38:57 AM]
└─>$ ./EXP5
Enter expression: id - id
Accepted
┌─[cachy@Cached-Excellence:~/c/compile_d]─[08:39:09 AM]
└─>$ ./EXP5
Enter expression: id * id + id / id
Accepted
┌─[cachy@Cached-Excellence:~/c/compile_d]─[08:39:23 AM]
└─>$ id % id
id: ‘%’: no such user
id: ‘id’: no such user
┌─[cachy@Cached-Excellence:~/c/compile_d]─[08:39:29 AM]
└─>$ ./EXP5
Enter expression: id % id
Accepted
┌─[cachy@Cached-Excellence:~/c/compile_d]─[08:39:42 AM]
└─>$ ./EXP5
Enter expression: E'
Rejected
┌─[cachy@Cached-Excellence:~/c/compile_d]─[08:40:04 AM]
└─>$ ./EXP5
Enter expression: id ^ id
Rejected
┌─[cachy@Cached-Excellence:~/c/compile_d]─[08:40:19 AM]
└─>$ 
```