EXP NO-8

GENERATE THREE ADDRESS CODE FOR A SIMPLE PROGRAM USING LEX AND YACC

AIM:

To design and implement a LEX and YACC program that generates three-address code (TAC) for a simple arithmetic expression or program. The program will:

- Recognize expressions like addition, subtraction, multiplication, and division.
- Generate three-address code that represents the operations in a way that could be directly translated into assembly code or intermediate code for a compiler.

PROGRAM

LEX TOOL: ex8.l

```
%{
#include "y.tab.h"
#include <stdlib.h>
%}

%%

[0-9]+ { yylval.str = strdup(yytext); return NUMBER; }
[a-zA-Z_][a-zA-Z0-9_]* { yylval.str = strdup(yytext); return ID; }
[+\-*/=()] { return yytext[0]; }
[ \t\n] { /* Ignore whitespace */ }
. { printf("Unexpected character: %s\n", yytext); }

%%

int yywrap() {
    return 1;  // End of input
}
```

YACC TOOL: ex8.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int temp_count = 0;
char* new_temp() {
    char* temp = (char*)malloc(8);
    sprintf(temp, "t%d", temp_count++);
    return temp;
}

void emit(char* result, char* op1, char op, char* op2) {
    printf("%s = %s %c %s\n", result, op1, op, op2);
}

void emit_assign(char* id, char* expr) {
    printf("%s = %s\n", id, expr);
}
%}

%union {
    char* str;
}

%token <str> ID NUMBER
%type <str> expr term factor
%left '+' '-'
%left '*' '/'

%%

statement : ID '=' expr { emit_assign($1, $3); }
          ;

expr : expr '+' term { $$ = new_temp(); emit($$, $1, '+', $3); }
     | expr '-' term { $$ = new_temp(); emit($$, $1, '-', $3); }
     | term { $$ = $1; }
     ;

term : term '*' factor { $$ = new_temp(); emit($$, $1, '*', $3); }
     | term '/' factor { $$ = new_temp(); emit($$, $1, '/', $3); }
     | factor { $$ = $1; }
     ;
```

```
factor : '(' expr ')' { $$ = $2; }
       | NUMBER { $$ = $1; }
       | ID { $$ = $1; }
       ;

%%

int main() {
    yyparse();
    return 0;
}

void yyerror(const char* s) {
    fprintf(stderr, "Error: %s\n", s);
}
```

OUTPUT

```
Enter an expression (e.g., a = b * c + d):
a=b*c+d
t0 = b * c
t1 = t0 + d
a = t1
```

RESULT:

Thus the process effectively tokenizes the input, parses it according to defined grammar rules, and generates the corresponding Three-Address Code, facilitating further compilation or interpretation stages.

KEERTHIKA S-220701127