**PRACTICAL 10**

**AIM: Intermediate code (IC) generator for arithmetic expression.**

**CODE:**

Lex.l-

%{

#include "y.tab.h"

#include <stdlib.h>

#include <string.h>

extern YYSTYPE yylval;

%}

%%

[0-9]+ { yylval.ival = atoi(yytext); return NUM; }

[a-zA-Z][a-zA-Z0-9\_]\* { yylval.sval = strdup(yytext); return ID; }

[\t ]+ ; // skip whitespace

\n { return '\n'; }

\+ { return '+'; }

- { return '-'; }

\\* { return '\*'; }

\/ { return '/'; }

\( { return '('; }

\) { return ')'; }

. { return yytext[0]; } // catch-all for any other character

%%

int yywrap() {

return 1;

}

Yacc-

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int yylex(void);

void yyerror(const char\* s);

int tempCount = 1;

char\* createTemp() {

char\* temp = (char\*)malloc(10);

sprintf(temp, "t%d", tempCount++);

return temp;

}

%}

%union {

int ival;

char\* sval;

}

%token <ival> NUM

%token <sval> ID

%type <sval> expr

%left '+' '-'

%left '\*' '/'

%%

stmt:

expr '\n' {

printf("Result: %s\n", $1);

}

;

expr:

expr '+' expr {

char\* temp = createTemp();

printf("%s = %s + %s\n", temp, $1, $3);

$$ = temp;

}

| expr '-' expr {

char\* temp = createTemp();

printf("%s = %s - %s\n", temp, $1, $3);

$$ = temp;

}

| expr '\*' expr {

char\* temp = createTemp();

printf("%s = %s \* %s\n", temp, $1, $3);

$$ = temp;

}

| expr '/' expr {

char\* temp = createTemp();

printf("%s = %s / %s\n", temp, $1, $3);

$$ = temp;

}

| '(' expr ')' {

$$ = $2;

}

| NUM {

char\* temp = (char\*)malloc(10);

sprintf(temp, "%d", $1);

$$ = temp;

}

| ID {

$$ = $1;

}

;

%%

void yyerror(const char\* s) {

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

}

int main() {

printf("Enter arithmetic expression:\n");

yyparse();

return 0;

}







