Ex No 1: Lexical analyser: classify tokens

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

int isOperator(char c){
   return (c=='+'||c=='-'||c=='*'||c=='/'||c=='%'||c=='='||c=='>'||c=='<'||c=='!'||c=='&'||c=='|');
}

int isDelimiter(char c){
   return (c==' '||c=='\n'||c=='\t'||c=='('||c==')'||c=='{'||c=='}'||c=='['||c==']'||c==','||c==';'||c=='.'||c==':');
}

int isKeyword(char *c){
   const char kwds[32][10] = {"auto","break","case","char","const","continue","default","do",
      "double","else","enum","extern","float","for","goto","if","int","long","register",
      "return","short","signed","sizeof","static","struct","switch","typedef","union",
      "unsigned","void","volatile","while"};

   for(int i=0;i<32;i++)
      if(strcmp(kwds[i],c)==0) return 1;
   return 0;
}

int isValidIdentifier(char *s){
   if(!isalpha(s[0]) && s[0]!='_') return 0;
   if(isKeyword(s)) return 0;
   for(int i=1;s[i];i++)
      if(!isalnum(s[i]) && s[i]!='_') return 0;
   return 1;
}

void printTokens(char kwd[][100], char idf[][100], char lit[][100], char opr[], char delim[], int kl,int
il,int ll,int ol,int dl){
   printf("___Tokens___\n");
   int count = 1;
   printf("Keywords:\n"); for(int i=0;i<kl;i++) printf("%d. %s\n", count++, kwd[i]);
   printf("Identifiers:\n"); for(int i=0;i<il;i++) printf("%d. %s\n", count++,idf[i]);
   printf("Literals:\n"); for(int i=0;i<ll;i++) printf("%d. %s\n", count++,lit[i]);
   printf("Operators:\n"); for(int i=0;i<ol;i++) printf("%d. %c\n", count++,opr[i]);
```

```
    printf("Delimiters:\n"); for(int i=0;i<dl;i++) printf("%d. %c\n", count++,delim[i]);
}

void tokenize(char *s){
    int len = strlen(s), start=0,end=0;
    char kwd[100][100], idf[100][100], lit[100][100], opr[1000], delim[1000];
    int kl=0,il=0,ll=0,ol=0,dl=0;

    while(end<=len){
        if(!isDelimiter(s[end]) && !isOperator(s[end])) { end++; continue; }

        if(start!=end){
            char token[100]; strncpy(token,s+start,end-start); token[end-start]='\0';
            if(isKeyword(token)) strcpy(kwd[kl++],token);
            else if(isValidIdentifier(token)) strcpy(idf[il++],token);
            else strcpy(lit[ll++],token);
        }

        if(isOperator(s[end])) opr[ol++] = s[end];
        else if(isDelimiter(s[end]) && !isspace(s[end])) delim[dl++] = s[end];

        end++; start=end;
    }

    printTokens(kwd,idf,lit,opr,delim,kl,il,ll,ol,dl);
}

int main(){
    FILE *file = fopen("input.c","r");
    if(!file){ printf("Cannot open file.\n"); return 1; }

    char buffer[10000]; int idx=0; char c;
    while((c=fgetc(file))!=EOF) buffer[idx++] = c;
    buffer[idx]='\0'; fclose(file);

    tokenize(buffer);
    return 0;
}
```

Ex No: 2 : Implement lexical analyser using lex tool:

```
%{
   #include<stdio.h>
   #include<stdlib.h>
%}
%%
[a-zA-Z][a-zA-Z0-9]*    {printf("IDENTIFIER: %s\n", yytext);}
[0-9]+                  {printf("LITERAL: %s\n", yytext);}
[+\-*/=]                {printf("OPERATOR: %s\n", yytext);}
'if'|'else'|'for'|'while' {printf("KEYWORD: %s\n", yytext);}
[.;:()\[\]\{\}]         {printf("DELIMIETER: %s\n", yytext);}
.                       ;
%%

int main(){
   printf("Enter code:\n");
   yylex();
   return 0;
}

int yywrap(){
   return 1;
}
```

Ex No 3: YACC and LEX

a) **Simple calculator**

calc.l

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

%%
[ \t]+ ;
\n { return '\n'; }
[0-9]+ { yylval = atoi(yytext); return NUMBER; }
[+\-*/] { return yytext[0]; }
. ;
```

```
%%
```

calc.y:

```
%{
   #include<stdio.h>
   #include<stdlib.h>
   int yylex(void);
   int yyerror(const char *s);
%}
%token NUMBER
%left '+' '-'
%left '*' '/'

%%
stmt: expr '\n' {printf("= %d", $$);}
   | '\n'

expr: expr '+' expr {$$ = $1 + $3;}
   | expr '-' expr {$$ = $1 - $3;}
   | expr '*' expr {$$ = $1 * $3;}
   | expr '/' expr {$$ = $1 / $3;}
   | NUMBER;
%%

int main(){
   printf("Enter expression: ");
   yyparse();
   return 0;
}

int yyerror(const char *s){
   printf("Error: %s\n", s);
   return 1;
}
```

b) Validity of expression:

lex:

```
%{
   #include<stdio.h>
   #include<stdlib.h>
   #include "3avalidcalc.tab.h"
%}

%%
[ \t\n]        ;
[0-9]+          {yylval = atoi(yytext); return NUMBER;}
[+\-*/]         {return yytext[0]; }
.               ;

%%
int yywrap(){
   return 1;
}
```

yacc:

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

   int yylex(void);
   int yyerror(const char *s);
%}

%token NUMBER;
%left '+' '-'
%left '*' '/'

%%
stmt: expr    {printf("Valid\n");}
   ;

expr: expr '+' expr
   | expr '-' expr
   | expr '*' expr
   | expr '/' expr
   | NUMBER
   ;

%%
int main(){
```

```
    printf("Enter expression: \n");
    yyparse();
    return 0;
}

int yyerror(const char *s){
    printf("Error: %s\n", s);
    return 1;
}
```

c) Valid identifiers

```
%{
#include y..tab.h"
%}

letter  [_a-zA-Z]
digit   [0-9]

%%
[ \t\n]+              ;
{letter}({letter}|{digit})*   { return ALNUM; }
.                     { return yytext[0]; }
%%
int yywrap() { return 1; }


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

    int yylex(void);
    int yyerror(char *s);
%}

%token ALNUM;

%%
idf: ALNUM     {printf("Valid identifier\n");}
    ;

%%
int main(){
```

```
    printf("Enter identifier:\n");
    yyparse();
    return 0;
}

int yyerror(char *s){
    printf("Error: Invalid identifier\n");
    return 1;
}
```

Ex No 4: Three Address Code

TAC.l

```
%{
#include "y.tab.h"
%}
%%
[ \t]+ ; /* Ignore spaces/tabs */
\n {return '\n';}
[0-9]+ { yylval.sym = yytext[0]; return NUMBER; }
[a-zA-Z]+ { yylval.sym = yytext[0]; return LETTER; }
. { return yytext[0]; } /* Return single-character tokens */
%%
int yywrap() { return 1; }
```

TAC.y

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

int ind = 0;
struct code { char opd1, opd2, opr; } C[20];

char Add(char a, char b, char op);
void TAC(), Quad(), Triple();
```

```
int yylex(void);
int yyerror(char *s);
%}

%union { char sym; }
%token <sym> LETTER NUMBER
%type <sym> expr
%left '+' '-'
%left '*' '/'

%%
input : stmt '\n' { TAC(); Quad(); Triple(); }
      ;

stmt : LETTER '=' expr ';' { Add($1, $3, '='); }
     | expr ';'
     ;

expr : expr '+' expr { $$ = Add($1,$3,'+'); }
     | expr '-' expr { $$ = Add($1,$3,'-'); }
     | expr '*' expr { $$ = Add($1,$3,'*'); }
     | expr '/' expr { $$ = Add($1,$3,'/'); }
     | '(' expr ')' { $$ = $2; }
     | NUMBER { $$ = $1; }
     | LETTER { $$ = $1; }
     ;
%%

int main() {
   printf("Enter Expression:\n");
   yyparse();
   return 0;
}

int yyerror(char *s) { printf("Error: %s\n", s); exit(0); }

char Add(char a, char b, char op) {
   C[ind].opd1 = a;
   C[ind].opd2 = b;
   C[ind].opr = op;
   return '1' + ind++;
}

void printOp(char c) {
```

```
    if (c >= '1' && c <= '9') printf("t%c", c);
    else printf("%c", c);
}

void TAC() {
    printf("\n--- THREE ADDRESS CODE ---\n");
    for (int i = 0; i < ind; i++) {
        if (C[i].opr != '=') {
            printf("t%d = ", i+1);
            printOp(C[i].opd1);
            printf(" %c ", C[i].opr);
            printOp(C[i].opd2);
        } else {
            printOp(C[i].opd1);
            printf(" = ");
            printOp(C[i].opd2);
        }
        printf("\n");
    }
}

void Quad() {
    printf("\n--- QUADRUPLE CODE ---\n");
    for (int i = 0; i < ind; i++) {
        printf("%c\t", C[i].opr);
        printOp(C[i].opd1); printf("\t");
        printOp(C[i].opd2); printf("\t");
        printf("t%d\n", i+1);
    }
}

void Triple() {
    printf("\n--- TRIPLE CODE ---\n");
    for (int i = 0; i < ind; i++) {
        printf("(%d)\t%c\t", i+1, C[i].opr);
        printOp(C[i].opd1); printf("\t");
        printOp(C[i].opd2);
        printf("\n");
    }
}
```

5) Typecheck

```
%{
#include "5typecheck.tab.h"
#include <string.h>
%}

%%
[ \t\n]+              ;
"int"                 { return INT; }
"float"               { return FLOAT; }
[0-9]+\.[0-9]+        { yylval.type = 2; return NUM; }
[0-9]+                { yylval.type = 1; return NUM; }
[a-zA-Z][a-zA-Z0-9]*  { yylval.name = strdup(yytext); return ID; }
[;=+\-*/()]           { return yytext[0]; }
%%

int yywrap() { return 1; }


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

#define INT_T 1
#define FLOAT_T 2

typedef struct { char *name; int type; } Var;
Var vars[50];
int count = 0;

int get_type(char *n) {
    for(int i=0; i<count; i++)
        if(!strcmp(vars[i].name, n)) return vars[i].type;
    return -1;
}

void add_var(char *n, int t) {
    vars[count].name = strdup(n);
    vars[count++].type = t;
}

void yyerror(const char *s) { fprintf(stderr, "Error: %s\n", s); exit(1); }
int yylex(void);
%}
```

```
%union { int type; char *name; }
%token <name> ID
%token <type> NUM
%token INT FLOAT
%type <type> expr

%%
prog: stmts ;

stmts: stmt ';' | stmts stmt ';' ;

stmt: INT ID     { if(get_type($2)!=-1) yyerror("Redeclared"); add_var($2, INT_T); }
   | FLOAT ID   { if(get_type($2)!=-1) yyerror("Redeclared"); add_var($2, FLOAT_T); }
   | ID '=' expr { int t=get_type($1); if(t==-1) yyerror("Undeclared");
                if(t!=$3) yyerror("Type mismatch"); }
   ;

expr: NUM              { $$ = $1; }
   | ID              { int t=get_type($1); if(t==-1) yyerror("Undeclared"); $$ = t; }
   | expr '+' expr    { if($1!=$3) yyerror("Type mismatch"); $$ = $1; }
   | expr '-' expr    { if($1!=$3) yyerror("Type mismatch"); $$ = $1; }
   | expr '*' expr    { if($1!=$3) yyerror("Type mismatch"); $$ = $1; }
   | expr '/' expr    { if($1!=$3) yyerror("Type mismatch"); $$ = $1; }
   | '(' expr ')'     { $$ = $2; }
   ;
%%

int main() {
   printf("Enter code:\n");
   yyparse();
   printf("Success!\n");
   return 0;
}
```

6) Constant folding

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
struct expr {
char lhs;
char rhs[20];
```

```
} input[10], optimized[10];
int main() {
int n, i, j = 0;
printf("Enter number of expressions: ");
scanf("%d", &n);
for (i = 0; i < n; i++) {
printf("Enter LHS variable: ");
scanf(" %c", &input[i].lhs);
printf("Enter RHS expression: ");
scanf("%s", input[i].rhs);
}
printf("\n--- Original Code ---\n");
for (i = 0; i < n; i++) {
printf("%c = %s\n", input[i].lhs, input[i].rhs);
}
for (i = 0; i < n; i++) {
char *exp = input[i].rhs;
int num1, num2, res;
char op;
if (sscanf(exp, "%d%c%d", &num1, &op, &num2) == 3) {
switch (op) {
case '+': res = num1 + num2; break;
case '-': res = num1 - num2; break;
case '*': res = num1 * num2; break;
case '/':
if (num2 != 0) res = num1 / num2;
else res = 0;
break;
default: res = 0;
}
sprintf(optimized[j].rhs, "%d", res);
optimized[j].lhs = input[i].lhs;
j++;
} else {
strcpy(optimized[j].rhs, input[i].rhs);
optimized[j].lhs = input[i].lhs;
j++;
}
}
printf("\n--- Optimized Code (After Constant Folding) ---\n");
for (i = 0; i < j; i++) {
printf("%c = %s\n", optimized[i].lhs, optimized[i].rhs);
}
return 0;
```

```
}
```

Ex No 7: Assembly code:

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

int main() {
    int n;
    printf("Enter number of TAC statements: ");
    scanf("%d", &n);

    char lhs[10], op1[10], op2[10], op;

    for (int i = 0; i < n; i++) {
        scanf("%s = %s %c %s", lhs, op1, &op, op2);

        printf("\n--- Assembly for: %s = %s %c %s ---\n", lhs, op1, op, op2);

        printf("LOAD %s\n", op1);

        switch (op) {
            case '+': printf("ADD %s\n", op2); break;
            case '-': printf("SUB %s\n", op2); break;
            case '*': printf("MUL %s\n", op2); break;
            case '/': printf("DIV %s\n", op2); break;
            default : printf("; Unknown Operator\n");
        }

        printf("STORE %s\n", lhs);
    }

    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MAXLINES 100
#define MAXLEN 100
int isNumber(const char *s) {
```

```
 for (int i = 0; s[i]; i++)
 if (!isdigit((unsigned char)s[i])) return 0;
 return 1;
}
int main() {
 char line[MAXLEN];
 char tac[MAXLINES][MAXLEN];
 int n = 0;
 printf("Enter TAC lines:\n");
 while (fgets(line, sizeof(line), stdin)) {
 if (line[0] == '\n') break;
 line[strcspn(line, "\n")] = 0;
 strcpy(tac[n++], line);
 }
 printf("---------------------------------------\n");
 printf("\n8086 Assembly Instructions\nSTART:\n");
 for (int i = 0; i < n; i++) {
 char lhs[MAXLEN], rhs1[MAXLEN], rhs2[MAXLEN];
 char op;
 if (sscanf(tac[i], "%s = %s %c %s", lhs, rhs1, &op, rhs2) == 4) {
 printf(" MOV AX, %s\n", rhs1);
 printf(" MOV BX, %s\n", rhs2);
 switch (op) {
 case '+': printf(" ADD AX, BX\n"); break;
 case '-': printf(" SUB AX, BX\n"); break;
 case '*': printf(" IMUL BX\n"); break;
 case '/': printf(" XOR DX, DX\n DIV BX\n"); break;
 }
 printf(" MOV %s, AX\n", lhs);
 } else if (sscanf(tac[i], "%s = %s", lhs, rhs1) == 2) {
 printf(" MOV AX, %s\n", rhs1);
 printf(" MOV %s, AX\n", lhs);
 } else if (sscanf(tac[i], "goto %s", lhs) == 1) {
 printf(" JMP %s\n", lhs);
 } else if (sscanf(tac[i], "%[^:]:", lhs) == 1) {
 printf("%s:\n", lhs);
 }
 }
 printf("END START\n");
 return 0;
}
```