# CD Programs for CIE and SEE(21CS36)

## Program 1:

**a)Write a LEX program to count number of words, lines, characters and whitespaces in a given paragraph -**

```
%{

int l=0,w=0,s=0,c=0;

%}

%%

[.] l++;

[ ] s++;

[\t] s=s+3;

[a-zA-Z]+ {w++; c=c+yyleng;}

. ;

\n return 0;

%%

int main()

{

printf("Enter the string\n");

yylex();

printf("no. of
lines=%d\n\twords=%d\n\tcharacters=%d\n\tspaces=%d\n",l,w,c,s);

}
```

----------OUTPUT----------

1. Enter the string

R V College Of Engineering

no. of lines=0

words=5

characters=22

spaces=4

2. Enter the string

R V College of engineering. Department of CSE.

no. of lines=2

words=8

characters=37

spaces=7

**b) Write a YACC program to recognize strings of the form $a^n b^{n+m} c^m$, n,m>=0-**

**1b.l-**

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

%%
```

```
"a" { return 'a'; }

"b" { return 'b'; }

"c" { return 'c'; }

.    { return yytext[0]; }


\n   { return 0; }  // Handle newline character
properly


%%


int yywrap() {

    return 1;

}
```

**1b.y-**

```
%{

    #include <stdio.h>

    #include <stdlib.h>

    #include <string.h>

    void yyerror(const char *s);
```

```
    int yylex(void);
%}
%start S
%%

S: B C
 ;

B: 'a' B 'b'
 | /* empty */
 ;

C: 'b' C 'c'
 | /* empty */
 ;

%%

int main() {
    if (yyparse() == 0) {
```

```
        printf("\nValid string\n");
    }
    return 0;
}

void yyerror(const char *s) {
    fprintf(stderr, "INVALID!!!\n");
    exit(1);
}
```

**Output-**

i. abbcc-Invalid

ii. aabbbc - Valid string

## Program 2:

**a)Write a LEX program to count number of Positive & negative integers and Positive & negative fractions-**

**2a.l-**

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

```
int p = 0, n = 0, pf = 0, nf = 0;

%}

%%

[+]?[0-9]+    { p++; }

[-][0-9]+     { n++; }

[0-9]*[.][0-9]+  { pf++; }

[-][0-9]*[.][0-9]+ { nf++; }


[-][0-9]*[.]*[0-9]+[/][-][0-9]*[.]*[0-9]+ { pf++; }

[0-9]*[.]*[0-9]+[/][0-9]*[.]*[0-9]+ { pf++; }

[0-9]*[.]*[0-9]+[/][-][0-9]*[.]*[0-9]+ { nf++; }

[-][0-9]*[.]*[0-9]+[/][0-9]*[.]*[0-9]+ { nf++; }

\n { return 0; }

%%

int main() {

    printf("Enter the no.'s\n");

    yylex();

    printf("Number of positive integers = %d\n\tnegative
integers = %d\n\tpositive fractions = %d\n\tnegative
fractions = %d\n", p, n, pf, nf);

    return 0;

}
```

Output-

```
Enter the no.'s
1 2 -5 1 3
    Number of positive integers = 4
        negative integers = 1
        positive fractions = 0
        negative fractions = 0
```

**b) Write a YACC program to validate and evaluate a simple expression involving operators +,- , * and /-**

## 2b.l-

```
%{

#include "y.tab.h"

extern int yylval;

%}

%%

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

[-]       return '-';

[+]       return '+';

[*]       return '*';

[/]       return '/';

.         return yytext[0];

\n        return 0;


%%
```

**2b.y-**

```
%{
#include <stdio.h>
#include <stdlib.h>
extern int yylex();
int yyerror();
%}
%token NUM
%left '+' '-'
%left '/' '*'
%%
S: I { printf("result is %d\n", $$); };
I: I '+' I { $$ = $1 + $3; }
 | I '-' I { $$ = $1 - $3; }
 | I '*' I { $$ = $1 * $3; }
 | I '/' I {
     if ($3 == 0) {
         yyerror();
     } else {
         $$ = $1 / $3;
     }
```

```
 }
 | '(' I ')' { $$ = $2; }
 | NUM;
%%
int main() {
    yyparse();
    printf("Valid\n");
    return 0;
}
int yyerror() {
    printf("INVALID!!!!\n");
    exit(0);
}
```

Output-
i. 2+3 - Valid  The result is 5
ii. 2+ -INVALID

## Program 3:

### a) Write a LEX program to count the number of comment lines in a given C program. Also eliminate them and copy that program into a separate file.

```
%{
#include<stdio.h>
int flag=0;
int c=0;
```

```
int flg=0;

%}

%%


"//".*  { if(flag==1){fprintf(yyout," ");flg--;}else{c++;
fprintf(yyout," ");flg++;}}

"/*".*\n?"*/"?  { if(flg==1){fprintf(yyout," ");} else {flag++;
fprintf(yyout," ");c++;}}

.*"*/"  {if(flag==1){ fprintf(yyout," "); c++;flag--;}}



%%

main()

{

        yyin= fopen("v.txt","r");

        yyout = fopen("v1.txt","w");

        yylex();

        printf("Number of comment lines=%d",c);

}
```

input file v.txt

```
//jsdhg/*fjkjhgjghj
fghgfhghg*/
//jhgdjfhgj
/*mndbfjkhk /*m,jhkdf*/ljdfghlk
*/
 /*jhgds//jhgdfjgds
hfdkjkg */
```

```
kjsdfhkjhkfjh

---------------------------------OUTPUT------------------------------

no. of comment lines=4

ouput file v1.txt

kjsdfhkjhkfjh
```

**b) Write a YACC program to recognize a nested (minimum 3 levels) FOR loop statement for C language.**

**3b.l-**

```
%{

#include "y.tab.h"

%}

%%

"for"   { return FOR; }

"("     { return LPAREN; }

")"     { return RPAREN; }

"{"     { return LF; }

"}"     { return RF; }

"="     { return '='; }

"-"     { return '-'; }

"+"     { return '+'; }

">"     { return '>'; }
```

```
"<"     { return '<'; }

";"     { return ';'; }

"=="    { return EQ; }

"<="    { return LE; }

">="    { return GE; }

"+="    { return ADD_ASSIGN; }

"-="    { return SUB_ASSIGN; }

"++"    { return INC; }

"--"    { return DEC; }

[a-zA-Z]+   { return EXP; }

[0-9]+  { return NUM; }

[ \t]   { /* Ignore whitespace */ }

\n      { return 0; }

.       { /* Ignore any other character */ }

%%

int yywrap(){

 return 1;

 }
```

**3b.y-**

```
%{
#include <stdio.h>
#include <stdlib.h>
// Declare yylex function
int yylex(void);
int count = 0;
void yyerror(const char *s);
%}
%token FOR LPAREN RPAREN LF RF EXP NUM
%token EQ LE GE ADD_ASSIGN SUB_ASSIGN INC DEC
%%
S : I
;
I : FOR A B { count++; }
;
A : LPAREN E ';' E ';' E RPAREN
;
E : EXP Z NUM
  | EXP Z EXP
```

```
  | EXP U
  | /* empty */  /* Handling space as an empty rule
*/
;
Z : '='
  | '>'
  | '<'
  | LE /* Placeholder for '<=' */
  | GE /* Placeholder for '>=' */
  | EQ /* Placeholder for '==' */
  | ADD_ASSIGN /* Placeholder for '+=' */
  | SUB_ASSIGN /* Placeholder for '-=' */
;

U : INC /* Placeholder for '++' */
  | DEC /* Placeholder for '--' */
;
B : LF B RF
  | I
  | EXP
```

```
  | EXP I
  | /* empty */
;
%%
int main() {
  yyparse();
  printf("Number of nested FOR's are: %d\n", count);
  return 0;
}
void yyerror(const char *s) {
  printf("ERROR: %s\n", s);
  exit(1);
}
```

Output-

i. for(i=0;i<4;i++)

no. of nested FOR's are: 1

ii. for(i=0;i<3;i++){ for(i=2;i<n;i++)}}

ERROR!!!

iii. for(i=0;i<n;i++){for(j=0;jN7;j++)}

ERROR!!!

iv. for(i=0;i<n;i++){for(j=0;j<8;j++)}

no. of nested FOR's are: 2

## Program 4-

**a) Write a LEX program to recognize and count the number of identifiers, operators and keywords in a given input file.**

```
%{

#include<stdio.h>

int i=0,k=0,op=0;

%}

%%

auto|break|case|char|continue|do|default|const|double|else|enum|
extern|for|if|goto|float|int|long|register|return|signed|static|
sizeof|short|struct|switch|typedef|union|void|while|volatile|uns
igned {  }

("/"[^\"]*"/") { k++;}

("_"|[a-z]|[A-Z])("_"|[a-z]|[A-Z]|[0-9])* {i++;}

"#include".*  ;

"#"[a-zA-Z]+.*  ;

[;] ;

[ ] ;

[,] ;

[+*%/-] {op++;}

[\n] ;
```

```
%%

void main()

{

yyin=fopen("d.c","r");

yylex();

printf("No. of identifiers=%d\n,
keywords=%d,operators=%d",i,k,op);

}
```

input file d.c

```
#include<stdio.h>

#define max 10

int a,b,gfg;

float vbg;//int b;

/*int a*/

char gfhjk,kjhg;
```

----------OUTPUT----------

No. of identifiers=6

## b) Write a YACC program to recognize nested IF control statements(C language) and display the number of levels of nesting.

Lex Program

```
 %{

#include "y.tab.h"

%}
```

```
%%
"if" return IF;
"else" return ELSE;
[(] return LPAREN;
[)] return RPAREN;
[{] return LF;
[}] return RF;
[a-z]* return EXP;
[ ] return SPACE;
\n return 0;
%%
```

Yacc Program

```
%{
#include<stdio.h>
#include<stdlib.h>
int count=0;
%}
%token IF ELSE LPAREN RPAREN LF RF EXP SPACE
%%
S:I
;
I:IF E B {count++;}
```

```
;
E:LPAREN EXP RPAREN
;
B:LF B RF
|I
|EXP
|EXP SPACE I
|
;
%%
int main()
{
yyparse();
printf("no. of nested IF's are: %d\n",count);
}
int yyerror()
{
printf("ERROR!!!\n");
exit(0);
}
```

----------OUTPUT----------

if(abc)

no. of nested IF's are: 1

if(abc){if(abc)}

no. of nested IF's are: 2

if(ab){}

no. of nested IF's are: 1

**Program 5: Write a YACC program to recognize Declaration statement (C language) and display the number variables declared .**

**5.l -**

```
%{

#include "y.tab.h"

%}

%%

"int"    { return INT; }

"float"  { return FLOAT; }

"char"   { return CHAR; }

"double" { return DOUBLE; }

[a-zA-Z_][a-zA-Z0-9_]*  { yylval.str = strdup(yytext);
return IDENTIFIER; }

[0-9]+ {return NUM;}

"["      { return '['; }

"]"      { return ']'; }

","      { return ','; }
```

```
";"       { return ';'; }
[ \t\n]   { /* Ignore whitespace */ }
.         { /* Ignore any other characters */ }
%%
int yywrap() {
     return 1;
}
```

**5.y-**

```
%{
#include <stdio.h>
#include <stdlib.h>
int var_count = 0;
void yyerror(const char *s);
int yylex();
%}
%union {
     char *str;
}
%token <str> IDENTIFIER
%token INT FLOAT CHAR DOUBLE NUM
%%
program: declarations
```

```
     ;
declarations: declaration ';'
          | declarations declaration ';'
          ;
declaration: type var_list
          ;
type: INT
     | FLOAT
     | CHAR
     | DOUBLE
     ;
var_list: var
     | var_list ',' var
     ;
var: identifier
   | identifier '[' ']'  // Matches array without size
   | identifier '[' NUM ']' // Matches array with size
   ;
identifier: IDENTIFIER
          {
               var_count++;
          }
```

```
          ;
%%
void yyerror(const char *s) {
     fprintf(stderr, "Error: %s\n", s);
}
int main() {
     yyparse();
printf("Total number of variables declared: %d\n", var_count);
     return 0;
}
```

Input.txt:

int a[10], b, c;

float x, y;

char name[50];

double z;

**Output-**

Total number of variables declared : 7

## Program 6: YACC program that reads the C statements for an input file and converts them in quadruple three address intermediate code.

**6.l-**

```
 %{
#include "y.tab.h"
extern char yyval;
%}
NUMBER [0-9]+
LETTER [a-zA-Z]+
%%
{NUMBER} {
    yylval.sym = (char)yytext[0];
    return NUMBER;
}
{LETTER} {
    yylval.sym = (char)yytext[0];
    return LETTER;
}
\n { return 0; }
. { return yytext[0]; }
%%
```

**6.y-**

```
%{
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void ThreeAddressCode();
void triple();
void quadraple();
char AddToTable(char, char, char);
int ind = 0;
char temp = 'A';
struct incod {
    char opd1;
    char opd2;
    char opr;
};
%}
%union {
    char sym;
}
```

```
%token <sym> LETTER NUMBER

%type <sym> expr

%left '-' '+'

%right '*' '/'

%%

statement: LETTER '=' expr ';' {
AddToTable((char)$1, (char)$3, '='); }

         | expr ';'

         ;

expr: expr '+' expr { $$ = AddToTable((char)$1,
(char)$3, '+'); }

    | expr '-' expr { $$ = AddToTable((char)$1,
(char)$3, '-'); }

    | expr '*' expr { $$ = AddToTable((char)$1,
(char)$3, '*'); }

    | expr '/' expr { $$ = AddToTable((char)$1,
(char)$3, '/'); }

    | '(' expr ')' { $$ = (char)$2; }

    | NUMBER { $$ = (char)$1; }

    | LETTER { $$ = (char)$1; }

    ;
```

```
%%
yyerror(char *s) {
    printf("%s\n", s);
    exit(0);
}
struct incod code[20];
int id = 0;
char AddToTable(char opd1, char opd2, char opr) {
    code[ind].opd1 = opd1;
    code[ind].opd2 = opd2;
    code[ind].opr = opr;
    ind++;
    temp++;
    return temp;
}
void ThreeAddressCode() {
    int cnt = 0;
    temp++;
```

```
    printf("\n\n\tTHREE ADDRESS CODE\n\n");
    while (cnt < ind) {
        printf("%c =\t", temp);
        if (isalpha(code[cnt].opd1))
        printf("%c\t", code[cnt].opd1);
        else
        printf("%c\t", temp);
        printf("%c\t", code[cnt].opr);
        if (isalpha(code[cnt].opd2))
        printf("%c\t", code[cnt].opd2);
        else
        printf("%c\t", temp);
        printf("\n");
        cnt++;
        temp++;
    }
}
void quadraple() {
    int cnt = 0;
```

```
    temp++;
    printf("\n\n\tQUADRAPLE CODE\n\n");
    while (cnt < ind) {
        printf("\t%d\t%c\t", id, code[cnt].opr);
        if (isalpha(code[cnt].opd1))
        printf("%c\t", code[cnt].opd1);
        else
        printf("%c\t", temp);
        if (isalpha(code[cnt].opd2))
        printf("%c\t", code[cnt].opd2);
        else
        printf("%c\t", temp);
        printf("%c\n", temp);
        cnt++;
        temp++;
        id++;
    }
}
void triple() {
```

```
int cnt = 0, cnt1, id1 = 0;
temp++;
printf("\n\n\tTRIPLE CODE\n\n");
while (cnt < ind) {
    if (id1 == 0) {
    printf("%d\t%c\t", id1, code[cnt].opr);
    if (isalpha(code[cnt].opd1))
        printf("%c\t", code[cnt].opd1);
    else
        printf("%c\t", temp);
    cnt1 = cnt - 1;
    if (isalpha(code[cnt].opd2))
        printf("%c", code[cnt].opd2);
    else
        printf("%c\t", temp);
    } else {
    printf("%d\t%c\t", id1, code[cnt].opr);
    if (isalpha(code[cnt].opd1))
        printf("%c\t", code[cnt].opd1);
```

```
        else
            printf("%c\t", temp);
        cnt1 = cnt - 1;
        if (isalpha(code[cnt].opd2))
            printf("%d", id1 - 1);
        else
            printf("%c\t", temp);
        }
        printf("\n");
        cnt++;
        temp++;
        id1++;
    }
}
int main() {
    printf("Enter the Expression: ");
    yyparse();
    temp = 'A';
    ThreeAddressCode();
```

```
    quadraple();
    triple();
    return 0;
}
int yywrap() {
    return 1;
}
```

Output-

```
Enter the Expression: a=b+c;

        THREE ADDRESS CODE

B =     b       +       c
C =     a       =       B

        QUADRAPLE CODE

        0       +       b       c       E
        1       =       a       B       F

        TRIPLE CODE

0       +       b       c
1       =       a       0
```

**Program 7: Write a YACC program that identifies the Function Definition of C language**

**7.l-**

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

alpha [a-zA-Z]
digit [0-9]
%%
[ \t] ;
[ \n] {yylineno++;}
int return INT;
float return FLOAT;
char return CHAR;
void return VOID;
double return DOUBLE;
for return FOR;
while return WHILE;
if return IF;
else return ELSE;
printf return PRINTF;
```

```
struct return STRUCT;

return return RETURN;

\"([^\"\\]|\\.)*\" return STRING_LITERAL;

^"#include "[^\n]* return INCLUDE ;


{digit}+ return NUM;

{alpha}({alpha}|{digit})* return ID;


"<=" return LE;

">=" return GE;

"==" return EQ;

"!=" return NE;

">" return GT;

"<" return LT;

"." return DOT;

"\/\/".* ;      // Ignore single-line comments

"\/\*"(.*\n)*.*\*\/ ; // Ignore multi-line comments

. return yytext[0]; // Return unrecognized
characters

%%
```

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

**7.y-**

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

extern FILE *yyin;
extern int yylineno;
extern char *yytext;

int yylex(void);
void yyerror(char *s);
%}
%token INT FLOAT CHAR DOUBLE VOID
%token FOR WHILE IF ELSE PRINTF RETURN STRUCT
%token NUM ID INCLUDE DOT
%token STRING_LITERAL
```

```
%left '='
%left AND OR
%left '<' '>' LE GE EQ NE LT GT
%left '+' '-'
%left '*' '/'
%nonassoc UMINUS
%%
start: INCLUDE start| Function | Declaration ;
Declaration:
    Type Assignment ';'
    | Assignment ';'
    | FunctionCall ';'
    | ArrayUsage ';'
    | Type ArrayUsage ';'
    | StructStmt ';'
    | error
    ;
Assignment:
     ID '=' Assignment
```

```
| ID '=' FunctionCall

| ID '=' ArrayUsage

| ArrayUsage '=' Assignment

| ID ',' Assignment

| NUM ',' Assignment

| ID '+' Assignment

| ID '-' Assignment

| ID '*' Assignment

| ID '/' Assignment

| NUM '+' Assignment

| NUM '-' Assignment

| NUM '*' Assignment

| NUM '/' Assignment

| '\'' Assignment '\''

| '(' Assignment ')'

| '-' '(' Assignment ')'

| '-' NUM

| '-' ID

| NUM
```

```
    | ID
    ;
FunctionCall:
    ID '(' ')'
    | ID '(' Assignment ')'
    ;
ArrayUsage:
    ID '[' Assignment ']'
    ;
Function:
    Type ID '(' ArgListOpt ')' CompoundStmt
    ;
ArgListOpt:
    ArgList
    |
    ;
ArgList:
    ArgList ',' Arg
    | Arg
```

```
    ;
Arg:
    Type ID
    ;
CompoundStmt:
    '{' StmtList '}'
    ;
StmtList:
    StmtList Stmt
    |
    ;
Stmt:
    WhileStmt
    | Declaration
    | ForStmt
    | IfStmt
    | PrintFunc
    | ReturnStmt
    | ';'
```

```
    ;
ReturnStmt:
    RETURN Expr ';'
    ;
Type:
    INT
    | FLOAT
    | CHAR
    | DOUBLE
    | VOID
    ;
WhileStmt:
    WHILE '(' Expr ')' Stmt
    | WHILE '(' Expr ')' CompoundStmt
    ;
ForStmt:
    FOR '(' Expr ';' Expr ';' Expr ')' Stmt
    | FOR '(' Expr ';' Expr ';' Expr ')'
CompoundStmt
    | FOR '(' Expr ')' Stmt
```

```
    | FOR '(' Expr ')' CompoundStmt
    ;

IfStmt:
    IF '(' Expr ')' Stmt
    ;
StructStmt:
    STRUCT ID '{' Type Assignment '}'
    ;
PrintFunc:
    PRINTF '(' STRING_LITERAL ',' Expr ')' ';'
    ;
 Expr:
    Expr LE Expr
    | Expr GE Expr
    | Expr NE Expr
    | Expr EQ Expr
    | Expr GT Expr
    | Expr LT Expr
```

```
    | Assignment
    | ArrayUsage
    ;

%%
#include "y.tab.h"
int main(int argc, char *argv[]) {
    if (argc > 1) {
        yyin = fopen(argv[1], "r");
        if (!yyin) {
        perror(argv[1]);
        return 1;
        }
    }
    if (!yyparse()) {
        printf("\nParsing complete\n");
    } else {
        printf("\nParsing failed\n");
    }
```

```
    if (yyin) fclose(yyin);
    return 0;
}
void yyerror(char *s) {
    fprintf(stderr, "%d : %s %s\n", yylineno, s,
yytext);
}
```

**input.txt-**

```
#include <stdio.h>
int main() {
    int x = 5;
    int y = 10;
    int z = x+y;
    printf("Result: %d\n", z);
    return 0;
}
```

Output-

```
Parsing Complete
```

**Program 8: Write a YACC program that generates Assembly language (Target) Code for valid Arithmetic Expression.**

**8.l-**

```
%{

#include "y.tab.h"

#include <stdlib.h>

#include <string.h>

%}


DIGIT [0-9]

ID [a-zA-Z][a-zA-Z0-9]*

WS [ \t\n]

STRING \"[^"]*\"

%%


"int"       { return INT; }

"main"      { return MAIN; }

"printf"    { return PRINTF; }

{STRING}    { yylval.str = strdup(yytext); return
STRING; }
```

```
{ID}        { yylval.id = strdup(yytext); return ID;
}

{DIGIT}+    { yylval.num = atoi(yytext); return NUM;
}

"+"         { return ADD; }

"="         { return ASSIGN; }

"("         { return LPAREN; }

")"         { return RPAREN; }

";"         { return SEMI; }

","         { return COMMA; }

"{"         { return LBRACE; }

"}"         { return RBRACE; }

{WS}        ; /* ignore whitespace */


%%

int yywrap() {

    return 1;

}
```

**8.y-**

```
%{
```

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

extern int yylex();

extern int yylineno;

void yyerror(const char* s) {

   fprintf(stderr, "Error: %s at line %d\n", s,
yylineno);

    exit(1);

}

%}

%union {

    char* id;

    int num;

    char* str;

}

%token <id> ID

%token <num> NUM

%token <str> STRING
```

```
%token INT MAIN PRINTF ADD LPAREN RPAREN SEMI COMMA
LBRACE RBRACE ASSIGN

%start program

%%

program:
    INT MAIN LPAREN RPAREN LBRACE stmt_list RBRACE
    {
        printf(".data\n");
        printf(".LC0: .string \"Sum %%d\"\n");
        printf(".text\n");
        printf(".globl main\n");
        printf("main:\n");
    }
    ;


stmt_list:
    stmt
    | stmt_list stmt
    ;
stmt:
```

```
    INT ID ASSIGN NUM SEMI {
        printf("   movl $%d, %s\n", $4, $2);
    }
    | ID ASSIGN ID ADD ID SEMI {
        printf("movl %s, %%eax\n", $3);
        printf("addl %s, %%eax\n", $5);
        printf("movl %%eax, %s\n", $1);
    }
    | PRINTF LPAREN STRING COMMA ID RPAREN SEMI {
        printf("movl %s, %%edi\n", $5);  // Load
argument into %edi
        printf("movl $.LC0, %%rsi\n");   // Address
of format string into %rsi
        printf("call printf\n");        // Call
printf function
    }
    ;
%%

int main() {
```

```
    printf("Assembly code output:\n");
    yyparse();
    return 0;
}
```

**Output-**