Program 1

1. Write a Lex program to accept a C program and do error detection & correction for the following.

a) Check for un-terminated string constant in the input C program. i.e A string constant begins with double quotes and extends for more than one line. Intimate the error line numbers and the corrective actions to user.

```
%{

#include<stdio.h>

int c=0;

FILE *fp;

%}

%%

\n { c++; }

["][a-zA-Z0-9]*["] {ECHO; printf(" Valid String in line number %d\n ",c+1);}

["][a-zA-Z0-9]* { ECHO; printf(" InValid String in line number %d\n ",c+1);}

.;

%%

void main()

{

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

yylex();

fclose(yyin);

}
```

Program 2

2. Write a Lex program to Check for valid arithmetic expressions in the input C program. Report the errors in the statements to user.

```
%{

#include<stdio.h>

int c=0;

FILE *fp;

%}

operator [-+*/]

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

number [0-9]+

expression ({identifier}|{number}){operator}({identifier}|{number})

%%

\n { c++; }

^"#".+ ;

^("int "|"float "|"char ").+ ;

"void main()" ;

{identifier}"="({expression}+";") { printf("Valid expression in line no :%d\t",c+1);ECHO;printf("\n");}

{identifier}"="({number}|{identifier}";") { printf("Valid expression in line no
:%d\t",c+1);ECHO;printf("\n");}

({number}|([0-9]*[a-zA-Z0-9-]+))"="{expression}+ { printf("InValid expression in line no :
%d;Lvalue should satisfy the identifier rules\n",c+1);ECHO;printf("\n");}

{identifier}"=;" { printf("InValid expression in line no : %d; R-value required; Expression is
neededat right hand side of assignment operation\n",c+1);ECHO;printf("\n");}

{operator}{operator}+ {printf(" Invalid expression in line no: %d;More than one operator can't
beused in expression consequetively",c+1);ECHO;printf("\n");}

.|\n;
```

```
%%

void main()

{

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

yylex();

fclose(yyin);

}
```

program 3
3. Write a Lex program to accept a C program and do the following error detection & correction.
a) Check for the valid usages of numerical constants in the input C program. Intimate the invalid usages to user.

```
%{

#include<stdio.h>

int c=0;

%}

number [0-9]+(".")?[0-9]*

invalid [0-9]+(".")[0-9]*((".")[0-9]*)+

%%

\n {c++;}

{number} {printf("\nValid number in line number %d : ",c+1);ECHO;printf("\n");}

{number}[a-zA-Z0-9_]+ {printf("\nInvalid number in line number %d: Number followed with alphabets is invalid",c+1);ECHO;printf("\n");}

{invalid} {printf("\nInvalid number in line number %d: Number with more than one decimal points is invalid",c+1);ECHO;printf("\n");}
```

```
. ;

%%

void main()

{

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

yylex();

fclose(yyin);

}
```

program 4
4. Write a Lex program to accept a C program and do the following error detection & correction.
a) Check for valid declarative statements in your program. Intimate the invalid statements along with their line numbers to users.

```
%{

#include<stdio.h>

int c=0;

%}

%s DECLARE VAR

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

number [0-9]+[.]?[0-9]*

string ("\"")([a-zA-Z0-9]+)("\"")

%%

\n {c++;}

"int "|"float " {BEGIN DECLARE;}
```

```
<DECLARE>{identifier}("="{number})? {BEGIN VAR;}

<DECLARE>{identifier}("="{string}) {BEGIN VAR; printf("\n Invalid variable declaration in line no %d; string can't be assigned to integer or float variable:",c+1);ECHO;printf("\n");}

<VAR>";" {BEGIN 0;}

<VAR>{identifier}("="{number})? {}

<VAR>{identifier}("="{string}) {printf("\n Invalid variable declaration in line no %d; string can't be assigned to integer or float variable:",c+1);ECHO;printf("\n");}

<VAR>\n {BEGIN 0; c++;}

<VAR>"," {BEGIN DECLARE;}

<VAR>[,][,]+ {printf("\n Invalid usage of more than one comma in declaration in line no %d",c+1);BEGIN DECLARE;ECHO;printf("\n");}

.;

%%

void main()

{

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

yylex();

fclose(yyin);

}
```

program 5
5. Write a Lex program to accept a C program and do the following error detection & correction.
a) Check for the valid if statement in the input C program. Report the errors to users.

```
%{

#include<stdio.h>

int c=0,bc=0,fc=0;
FILE *fp;

%}

%s IF OPENP CLOSEP OPENF

%%

\n { c++; }

"if" {BEGIN IF;ECHO;bc=0;}

<IF>\n {c++;ECHO;printf("\n");}

<IF>"(" {BEGIN OPENP;ECHO;bc++;}

<IF>")" {BEGIN CLOSEP;ECHO;bc--;}

<OPENP>")" {ECHO;bc--;BEGIN CLOSEP;}

<OPENP>"(" {ECHO;bc++;}

<OPENP>. {ECHO;}

<CLOSEP>"{" {if(bc==0) {printf("condn is valid in line no %d\n",c+1);}

else printf("condn invalid in line no %d;Paranthesis mismatch in condn\n",c+1);

BEGIN OPENF;ECHO;printf("\n");fc++;}

<CLOSEP>"(" {BEGIN OPENP;bc++;ECHO;}

<CLOSEP>")" {ECHO;bc--;}

<CLOSEP>. {ECHO;}

<CLOSEP>\n {ECHO;printf("\n");c++;}

<OPENF>"}" {fc--;if(fc==0) BEGIN 0;;ECHO;printf("\n");}

<OPENF>. {ECHO;}

<OPENF>\n {ECHO;c++;}
```

```
.|\n ;

%%

void main()

{

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

yylex();

fclose(yyin);

}
```

Program 6
6. Write a Lex program to accept a C program and do the following error detection & correction.
a) Check for un- terminated multi line comment statement in your C program.

```
%{

#include<stdio.h>

int c=0,oc=0;

FILE *fp;

%}

%s COMMENT

%%

\n {c++;}

"/*" {BEGIN COMMENT;printf("\n comment begins in line no : %d\n",c+1);ECHO;oc=1;}

<COMMENT>"*/" {BEGIN 0;ECHO;oc=0;printf(": Comment ends in line no %d\n",c+1);}

<COMMENT>\n {c++;printf("\n");ECHO;}
```

```
<COMMENT>. {ECHO;}

. ;

%%

void main() {

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

yylex();

fclose(yyin);

if(oc==1)

{

printf("\n comment is not closed till the end of file!");

}

}
```

source.txt

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

/*dfddf*/

void main()

{

/*vbhfghfgh

dfhfgh
fghgfhfg

fghfh */

int a,b=78;

if((a<5&&j<9)

{ a=a+h;

g=6+7;

a=a+b;

printf("\n");

} /*

if(a<n) {

h=j+k;

}

if(a<n))

{

g=h+k;

}

}
```

program 7

7. Write Yacc program to accept a statement and do the following error detection.
a) Check for valid arithmetic expressions in the input C statement. Report the errors in the statements to user. Evaluate the arithmetic expression.

YACC

```
%{
```

```
#include<stdio.h>

#include<stdlib.h>

int yylex();

void yyerror();

%}


%token id num


%left '+' '-'

%left '/' '*' '%'



%%


stmt : expression { printf("\n valid exprn");}

        ;

expression : '(' expression ')'  { $$=$2;}

        | expression '+' expression {printf("\nplus recog!"); $$=$1+$3;printf("\n %d",$$);}

| expression '+' { printf ("\n Syntax error: Right operand is missing ");}

| expression '-' expression {printf("\nminus recog!");$$=$1-$3;printf("\n %d",$$);}

| expression '-' { printf ("\n Syntax error: Right operand is missing ");}

| expression '*' expression {printf("\nMul recog!");$$=$1*$3;printf("\n %d",$$);}

| expression '*' { printf ("\n Syntax error: Right operand is missing ");}

| expression '/' expression {printf("\ndivision recog!");if($3==0) printf("\ndivision cant be done, as
divisor is zero.");
```

```
                              else {$$=$1+$3;printf("\n %d",$$);}}

| expression '/' { printf ("\n Syntax error: Right operand is missing ");}

| expression '%' expression {printf("\n Modulos recog!"); $$=$1%$3;printf("\n %d",$$);}

| expression '%' { printf ("\n Syntax error: Right operand is missing ");}

| id { $$=$1;}

| num { $$=$1;}

;

%%

void main()

{

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

yyparse();

}

void yyerror()

{

printf(" Invalid arithmetic Expression\n"); exit(1);

}
```

LEX

```
%{

#include "y.tab.h"

#include<stdio.h>

#include<ctype.h>
extern int yylval;

int val;

%}


%%


[a-zA-Z][a-zA-Z0-9]*      {printf("\n enter the value of variable
%s:",yytext);scanf("%d",&val);yylval=val;return id;}

[0-9]+[.]?[0-9]*     {yylval=atoi(yytext);return num;}

[ \t]     ;

\n     {return 0;}

.     {return yytext[0];}

```

%%

program 8
8. Write Yacc program to accept a statement and do the following error detection.
a) Check for the valid relational expression and evaluate the expression
YACC

```
%{

#include<stdio.h>

#include<stdlib.h>

void yyerror();

int yylex();

%}

%token id num
```

```
%%

stmt: expression { printf("\n valid relational exprn");}

    ;

expression : '(' expression ')' {$$=$2;}

     | expression '<' expression {printf("\nless than recog!");($$=$1<$3);printf("\n %d",$$);}

| expression '<' { printf ("\n Syntax error: Right operand is missing ");exit(0);}

| expression '>' expression {printf("\ngreater than recog!");($$=$1>$3);printf("\n %d",$$);}

| expression '>' { printf ("\n Syntax error: Right operand is missing ");exit(0);}

| expression '<''=' expression {printf("\nless than or equal recog!");$$=($1<=$4);printf("\n
%d",$$);}

| expression '<''=' { printf ("\n Syntax error: Right operand is missing ");exit(0);}

| expression '>''=' expression {printf("\ngreater than or equal!");$$=($1>=$4);printf("\n %d",$$);}

| expression '>''=' { printf ("\n Syntax error: Right operand is missing ");exit(0);}

| expression '!''=' expression {printf("\nNot equal recog!");$$=($1!=$4);printf("\n %d",$$);}

| expression '!''=' { printf ("\n Syntax error: Right operand is missing ");exit(0);}

| expression '=''=' expression {printf("\ndouble equal recog!");$$=($1==$4);printf("\n %d",$$);}

| expression '=''=' { printf ("\n Syntax error: Right operand is missing");exit(0);}

| id {$$=$1;}

| num {$$=$1;}

;

%%
```

```
void main()

{

printf(" Enter relational expression\n");

yyparse();

}

void yyerror()

{

printf(" Invalid relational expression\n"); exit(1);

}
```

lex

```
%{

#include "y.tab.h"

#include<stdio.h>

#include<ctype.h>

extern int yylval;

int val;

%}


%%


[a-zA-Z][a-zA-Z0-9]*      {printf("\n enter the value of variable
%s:",yytext);scanf("%d",&val);yylval=val;return id;}

[0-9]+[.]?[0-9]*     {yylval=atoi(yytext);return num;}
```

```
[ \t]     ;

\n     {return 0;}

.     {return yytext[0];}


%%
```

program 9
9. Write Yacc program to accept a statement and do the following error detection.
a) Check for the valid logical expression and evaluate the expression
YACC

```
%{

#include<stdio.h>

#include<stdlib.h>

void yyerror();

int yylex();

%}

%token id num


%%


stmt: expression { printf("\n valid logical exprn : evaluated result is %d",$1);}

      ;

expression : '(' expression ')' { $$=$2;printf("\n value : %d",$$);}

           | expression '&''&' expression {printf("\nlogical and recog!");$$=(($1)&&($4));printf("\n
%d",$$);}

| expression '&''&' {printf("Syntax error: Right operand is missing ");exit(0);}

| expression '|''|' expression {printf("\nlogical or recog!");$$=($1||$4);printf("\n %d",$$);}
| expression '|''|' {printf("Syntax error: Right operand is missing ");exit(0);}

| '!' expression {printf("\nlogical not recog!");$$=!($2);printf("\n %d",$$);}

| '!' {printf("Syntax error: Right operand is missing ");exit(0);}

| expression '<' expression {printf("\nless than recog!");$$=($1<$3);printf("\n %d",$$);}

| expression '<' { printf ("\n Syntax error: Right operand is missing ");exit(0);}

| expression '>' expression {printf("\ngreater than recog!");$$=($1>$3);printf("\n %d",$$);}

| expression '>' { printf ("\n Syntax error: Right operand is missing ");exit(0);}

| expression '<''=' expression {printf("\nless than or equal recog!");$$=($1<=$4);printf("\n
%d",$$);}

| expression '<''=' { printf ("\n Syntax error: Right operand is missing ");exit(0);}

| expression '>''=' expression {printf("\ngreater than or equal!");$$=($1>=$4);printf("\n %d",$$);}

| expression '>''=' { printf ("\n Syntax error: Right operand is missing ");exit(0);}

| expression '!''=' expression {printf("\nNot equal recog!");$$=($1!=$4);printf("\n %d",$$);}

| expression '!''=' { printf ("\n Syntax error: Right operand is missing ");exit(0);}

| expression '=''=' expression {printf("\ndouble equal recog!");$$=($1==$4);printf("\n %d",$$);}

| expression '=''=' { printf ("\n Syntax error: Right operand is missing");exit(0);}

| id { $$=$1;}

| num { $$=$1;}

;


%%

void main()

{
    printf(" Enter logical expression\n");

    yyparse();

}

void yyerror()

{

    printf(" Invalid logical expression\n");

    exit(1);

}
```

lex

```
%{

#include "y.tab.h"

#include<stdio.h>

#include<ctype.h>

extern int yylval;

int val;

%}


%%


[a-zA-Z][a-zA-Z0-9]*      {printf("\n enter the value of variable
%s:",yytext);scanf("%d",&val);yylval=val;return id;}

[0-9]+[.]?[0-9]*     {yylval=atoi(yytext);return num;}

[ \t]     ;
```

```
\n    {return 0;}

.    {return yytext[0];}


%%
```