**1) Write a program to implement a lexical analyser.**

**LEX FILE**

%{

#include<stdio.h>

#include<stdlib.h>

%}

ws [ \t\n\r]

digit [0-9]

alpha [A-Za-z]

num {digit}+

id {alpha}({alpha}|{digit})\*

ops [+\-\*/\%\\\^]

%%

main {printf("Entry Point:%s\t",yytext);}

int|float|char {printf("Data-Type:%s\t",yytext);}

if|else {printf("Conditional:%s\t",yytext);}

switch|case {printf("Conditional:%s\t",yytext);}

for {printf("Loop:%s\t",yytext);}

{ws}\* {printf(" ");}

{id} {printf("Identifier:%s\t",yytext);}

{num} {printf("Number:%s\t",yytext);}

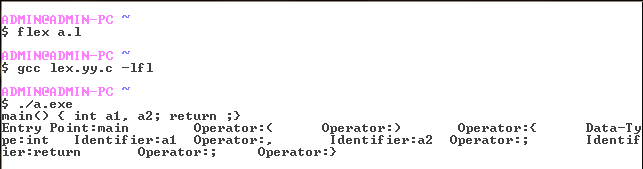
[()\[\]{};,] {printf("Operator:%s\t",yytext);}

{ops} {printf("Operator:%s\t",yytext);}

. {printf("Error:%s\t",yytext);}

%%

int main(){yylex();}



**3) Write a program to implement a simple calculator.**

**LEX FILE**

%{

#include <stdlib.h>

#include "y.tab.h"

%}

%%

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

[-+=/\*\n] {return \*yytext;}

%%

**YACC FILE**

%{

#include<stdio.h>

%}

%token NUM

%left '+' '-'

%left '\*' '/'

%%

prog :prog stmnt '\n'

|

;

stmnt :expr '=' {printf("%d\n",$1);}

;

expr :NUM

|expr '+' expr {$$ = $1 + $3;}

|expr '-' expr {$$ = $1 - $3;}

|expr '\*' expr {$$ = $1 \* $3;}

|expr '/' expr {$$ = $1 / $3;}

;

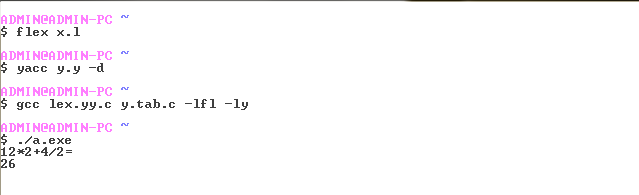
%%

int main(void) {

yyparse();

return 0;

}



**2) Write a program to implement a parser generator.**

**LEX FILE**

%{

#include"y.tab.h"

%}

D [0-9]

A [a-zA-Z]

id {A}({A}|{D})\*

%%

int|float|char {return TYPE;}

if {return IF;}

else {return ELSE;}

for|while {return LOOP;}

{D}\* {return NUM;}

{id} {return ALPHA;}

[ \t]\* {}

[-+()=/\*\n] {return \*yytext;}

[;,{}><] {return \*yytext;}

%%

**YACC FILE**

%{

#include<stdio.h>

%}

%token NUM ALPHA TYPE IF ELSE LOOP

%%

prog :prog func '\n' {printf("correct");}

|'\n'

|

;

func :def '(' para ')' '\n' '{' exprs '}'

|'\n'

;

cond :ALPHA ops NUM

|ALPHA ops ALPHA

|NUM ops NUM

|NUM ops ALPHA

;

ops :'=''='

|'='

|'<''='

|'>''='

|'<'

|'>'

;

def :TYPE ALPHA

;

para :init

|

;

init :def

|init ',' init

;

exprs :exprs expr '\n'

|exprs '\n'

|

;

expr :NUM

|init ';'

|expr '+' expr ';'

|expr '-' expr ';'

|expr '\*' expr ';'

|expr '/' expr ';'

|ALPHA '=' expr ';'

|IF '(' cond ')' '{' expr '}' ELSE '{' expr '}'

|IF '(' cond ')' '{' expr '}'

|';'

;

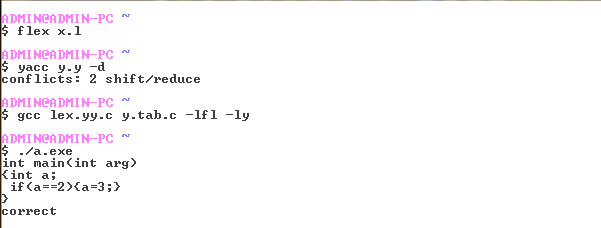
%%

int main(void) {

yyparse();

return 0;

}



**4) Generate three address codes for select C statements.**

#include<stdio.h>

#include<string.h>

main()

{

char str[10];

int i = 0, j;

char op;

printf(“\nEnter the expression:”);

scanf("%s",str);

op=str[0];

while((op!='0')||(op!='+')||(op!='-')||(op!=’\*’)||(op!=’/’))

op=str[++i];

switch(op)

{

case '+':printf("MOV R2,%c\n",str[i-1]);

printf("MOV R3,%c\n",str[i+1]);

printf("ADD R1,R2,R3\n");

printf("STORE R1,%c\n",str[i-3]);

break;

case '-':printf("MOV R2,%c\n",str[i-1]);

printf("MOV R3,%c\n",str[i+1]);

printf("SUB R1,R2,R3\n");

printf("STORE R1,%c\n",str[i-3]);

break;

case '\*':printf("MOV R2,%c\n",str[i-1]);

printf("MOV R3,%c\n",str[i+1]);

printf("MUL R1,R2,R3\n");

printf("STORE R1,%c\n",str[i-3]);

break;

case '/': printf("MOV R2,%c\n",str[i-1]);

printf("MOV R3,%c\n",str[i+1]);

printf("DIV R1,R2,R3\n");

printf("STORE R1,%c\n",str[i-3]);

break;

default: printf("ERROR\n");

break;

}

}

**OUTPUT**

Enter the expression: x=a+b

MOV R2,a

MOV R3,b

ADD R1,R2,R3

STORE R1,x