EXP 1: TOKENS IN C:

CODE:

```
%{
#include<stdio.h>
%}
%%
[ \t\n]          ;
int|float|char|if|else|double|long|switch|break|void      {printf("KEYWORDS\n");}
[a-z][a-z0-9]*  {printf("ID\n");}
","              {printf("COMMA\n");}
";"              {printf("SEMI COLON\n");}
.                {printf("%c\n",yytext[0]);}
%%
int main()
{
printf("Enter any input:");
yylex();
}
```

SAMPLE INPUT AND OUTPUT:

```
Enter any input:
int
KEYWORDS
hello
ID
;
SEMICOLON
$
$
```

EXP 2: BRANCHING ST.(IF..ELSE):

CODE:

(i)LEX PROG:

```
%{
#include<stdio.h>
#include "y.tab.h"
%}
%%
[ \t\n]          ;
"if"             return IF;
"else"           return ELSE;
int|float|char|double|long|switch|break|void      {return KEY;}
[0-9]+           return CONST;
[a-z][a-z0-9]*  {return ID;}
[ > < = ]        return OP;
"("              return OB;
")"              return CB;
"{"              return OC;
```

```
"}"             return CC;
","             {return C;}
";"             {return SC;}
.               {printf("%c\n",yytext[0]);}
%%
```

(ii)YACC PROG:

```
%{
#include<stdio.h>
#include "y.tab.h"
%}
%token IF ELSE KEY OP OB CB OC CC C SC CONST ID;
%start S;
%%
S: stmt {printf("Valid");};
stmt: IF OB exp CB OC body CC ELSE OC body CC;
exp: ID OP ID | ID OP CONST ;
body: KEY ID OP CONST SC ;
%%
yyerror()
{
printf("Error");
}
main()
{
yyparse();
}
```

SAMPLE INPUT:

```
if(a<10)
{
int b=10;
}
else
{
int b=20;
}
```

EXP 3: LOOPING ST.(WHILE):

CODE:

(i)LEX PROG:

```
%{
#include<stdio.h>
#include "y.tab.h"
%}
%%
"printf" return PRINTF;
"while" return WHILE;
"(" return OP;
```

```
[0-9]+ return NUM;
[a-zA-Z][a-zA-Z0-9_]* return ID;
"<" return LT;
">" return GT;
"==" return DEQ;
\"  return QUOT;
"!=" return NEQ;
"+" return ADD;
"-" return SUB;
")" return CP;
"{" return OB;
"=" return EQ;
";" return SC;
"}" return CB;
%%
int yywrap()
{
return 1;//end of line
}
```

(ii)YACC PROG:

```
%{
#include<stdio.h>
%}
%token WHILE OP ID LT GT DEQ NEQ ADD SUB CP OB EQ QUOT PRINTF TEXT SC NUM CB;
%start S
%%
S:Loop {printf("The syntax is valid");};
Loop: WHILE OP condn CP OB body CB;
condn: ID LT ID
    |ID GT ID
    |ID DEQ ID
    |ID NEQ ID
    ;
body:ID EQ ID SC body
   |ID EQ NUM SC body
   |ID ADD ADD SC body
   |ID SUB SUB SC body
   |PRINTF OP QUOT text QUOT CP SC body |
   ;
text:text | ID
   ;
%%
yyerror()
{
printf("Invalid syntax");
}
main()
{
```

```
yyparse();
}
```
SAMPLE INPUT:
```
while(a<b)
{
printf("loop");
a=10;
a++;
}
```

EXP 4A: ARRAY WITH FOR LOOP:
CODE:
(i)LEX PROG:
```
%{
#include<stdio.h>
#include "y.tab.h"
%}
%%
[ \t\n] ;
"#include<stdio.h>" return HEADER;
"main" return MAIN;
"int" return INT;
"for" return FOR;
[a-zA-Z][a-zA-Z0-9_]* return ID;
[0-9]+ return NUM;
"[" return LSQ;
"]" return RSQ;
"(" return OP;
")" return CP;
"{" return OB;
"}" return CB;
"<" return LT;
"+" return ADD;
"-" return SUB;
"," return COM;
"=" return EQ;
";" return SC;
. return 0;
%%
int yywrap()
{
return 1;
}
```
(ii)YACC PROG:
```
%{
#include<stdio.h>
```

```
%}
%%
%token INT ID NUM LSQ RSQ EQ SC OB CB COM OP CP LT ADD SUB FOR HEADER MAIN;
%start S;
S: header main OB arrayDec CB {printf("array");} |header main OB loop CB {printf("loop");} |header main
OB arrayDec loop CB {printf("Valid Program");};
header: HEADER;
main: INT MAIN OP CP;
arrayDec : INT ID LSQ num RSQ SC
    | INT ID LSQ num RSQ EQ num SC
    | INT ID LSQ RSQ EQ initialVal SC
    ;
loop: condn OB body CB
  ;
condn: FOR OP initialDec SC condFor SC incre CP;
initialDec: INT ID EQ num;
condFor: ID LT num;
incre: ID ADD ADD | ID SUB SUB;
body: ID LSQ ID RSQ EQ num SC;
num: NUM
  ;
initialVal: OB num COM num COM num CB
       ;
%%
yyerror()
{
printf("Invalid array declaration");
}
int main()
{
yyparse();
}
```

SAMPLE INPUTS:

TYPE 1

```
#include<stdio.h>
int main()
{
int marks[5];
for(int i=0;i<10;i++)
{
marks[i]=10;
}
}
```

TYPE 2

```
#include<stdio.h>
int main()
{
int marks[5]=10;
```

```
}
```
TYPE 3
```
#include<stdio.h>
int main()
{
for(int i=0;i<10;i++)
{
marks[i]=10;
}
}
```

EXP 4B: PROCEDURE CALLS:

CODE:

(i)LEX PROG:
```
%{
#include<stdio.h>
#include "y.tab.h"
%}
%%
[ \t\n] ;
"#include<stdio.h>" return HEADER;
"main" return MAIN;
"int" return INT;
"printf" return PRINTF;
"return" return RETURN;
[0-9][0-9]* return NUM;
[a-zA-Z][a-zA-Z0-9_%:]* return ID;
"(" return OP;
")" return CP;
"{" return OB;
"}" return CB;
"," return COM;
";" return SC;
"+" return ADD;
"-" return SUB;
"*" return MUL;
"/" return DIV;
"=" return EQ;
\" return QUOT;
. return 0;
%%
int yywrap()
{
return 1;
}
```
(ii)YACC PROG:
```
%{
#include<stdio.h>
```

```
%}
%token HEADER MAIN INT PRINTF ID NUM OP CP OB CB COM SC RETURN ADD SUB MUL DIV QUOT EQ;
%start S;
%%
S: funcDef mainFunc {printf("Valid Program");} | mainFunc funcDef {printf("Valid Program");};
mainFunc: INT MAIN OP CP OB body CB;
body:PRINTF OP QUOT text QUOT COM ID CP SC body| INT ID SC body
   | ID EQ ID OP NUM COM NUM CP SC body
   | RETURN NUM SC body |
   ;
funcDef: INT ID OP parameters CP OB bodyFunc CB;
parameters: parameters COM INT ID
        | INT ID
        |
        ;
bodyFunc: RETURN ID opr ID SC;
opr: ADD | SUB | MUL | DIV ;
text: text text | ID;
%%
yyerror()
{
printf("Invalid syntax");
}
int main()
{
yyparse();
}
```

SAMPLE INPUT:

```
int sum(int a,int b)
{
return a+b;
}
int main()
{
int add;
add=sum(10,10);
printf("Sum is:%d",add);
return 0;}
```

EXP 5: FIRST, FOLLOW SETS:

CODE:

```
#include<stdio.h>
#include<math.h>
#include<string.h>
 #include<ctype.h>
 #include<stdlib.h>
int n,m=0,p,i=0,j=0; char a[10][10],f[10]; void follow(char c); void first(char c);
int main(){
```

```
int i,z; char c,ch;
printf("Enter the no of productions:\n"); scanf("%d",&n);
printf("Enter the productions:\n"); for(i=0;i<n;i++) scanf("%s%c",a[i],&ch);
do{ m=0;
printf("Enter the elemets whose first & follow is to be found:"); scanf("%c",&c);
first(c); printf("First(%c)={",c); for(i=0;i<m;i++) printf("%c",f[i]);
printf("}\n");
strcpy(f," "); m=0;
follow(c); printf("Follow(%c)={",c);
for(i=0;i<m;i++) printf("%c",f[i]);
printf("}\n");
printf("Continue(0/1)?");
scanf("%d%c",&z,&ch);
}while(z==1);

return(0);
}
void first(char c)
{
int k; if(!isupper(c)) f[m++]=c; for(k=0;k<n;k++)
{ if(a[k][0]==c)//TERMINAL
{
if(a[k][2]=='$')
follow(a[k][0]);
else if(islower(a[k][2]))//TERMINAL
f[m++]=a[k][2];
else
first(a[k][2]);
}
}
}
void follow(char c)
{ if(a[0][0]==c)
f[m++]='$';
for(i=0;i<n;i++)
{
for(j=2;j<strlen(a[i]);j++)
{
if(a[i][j]==c)
{
if(a[i][j+1]!='\0')
first(a[i][j+1]); if(a[i][j+1]=='\0' && c!=a[i][0]) follow(a[i][0]);
}
}
}
}
```

OUTPUT:

```
Enter the no of productions:
5
Enter the productions:
S=aABb
A=c
A=0
B=d
B=0
Enter the elemets whose first & follow is to be found:S
First(S)={a}
Follow(S)={$}
Continue(0/1)?1
Enter the elemets whose first & follow is to be found:A
First(A)={c0}
Follow(A)={d0}
Continue(0/1)?1
Enter the elemets whose first & follow is to be found:B
First(B)={d0}
Follow(B)={b}
Continue(0/1)?0

--------------------------------
Process exited after 237.9 seconds with return value 0
Press any key to continue . . .
```

EXP 6:LL(1) PARSER:

CODE:

```
table={'E':{'id':'TR','(':'TR'},'R':{'+':'+TR',')':'e','$':'e'},'T':{'id':'FY','(':'FY'},'Y':{'+':'e','*':'*FY',')':'e','$':'e'},'F':{'id':'id','(':'(E)'}
}
inp='id + id * id $'
w=inp.split(' ')
i=0
word=w[i]
stack=[]
stack.append('$')
stack.append('E')
focus=stack[1]
terminal=['*','+','*','id','(',')','e','-','$']
while(focus):
   print('f',focus)
   print('w',word)
   if(focus=='$' and word=='$'):
      print('input string is valid')
      break
   elif(focus in terminal):
      if(focus == word):
         print('reduce')
         stack.pop()
         s=len(stack)-1
         focus=stack[s]
         i=i+1
         word=w[i]
   else:
      e=stack.pop()
      print(table[e])
      if(word not in table[e]):
         print('error')
         break;
      if(table[e][word]):
```

```
            right=table[e][word]
            if(right=='id'):
                stack.append('id')
            else:
                for j in range(len(right)-1,-1,-1):
                    if(right[j]!='e'):
                        stack.append(right[j])
        print(stack)

    s=len(stack)-1
    focus=stack[s]
```

OUTPUT:

```
('f', 'E')
('w', 'id')
{'(': 'TR', 'id': 'TR'}
['$', 'R', 'T']
('f', 'T')
('w', 'id')
{'(': 'FY', 'id': 'FY'}
['$', 'R', 'Y', 'F']
('f', 'F')
('w', 'id')
{'(': '(E)', 'id': 'id'}
['$', 'R', 'Y', 'id']
('f', 'id')
('w', 'id')
reduce
('f', 'Y')
('w', '+')
{')': 'e', '+': 'e', '*': '*FY', '$': 'e'}
['$', 'R']
('f', 'R')
('w', '+')
{')': 'e', '+': '+TR', '$': 'e'}
['$', 'R', 'T', '+']
('f', '+')
('w', '+')
reduce
('f', 'T')
('w', 'id')
{'(': 'FY', 'id': 'FY'}
['$', 'R', 'Y', 'F']
('f', 'F')
('w', 'id')
{'(': '(E)', 'id': 'id'}
['$', 'R', 'Y', 'id']
('f', 'id')
('w', 'id')
reduce
('f', 'Y')
('w', '*')
{')': 'e', '+': 'e', '*': '*FY', '$': 'e'}
['$', 'R', 'Y', 'F', '*']
```

```
('f', '*')
('w', '*')
reduce
('f', 'F')
('w', 'id')
{'(': '(E)', 'id': 'id'}
['$', 'R', 'Y', 'id']
('f', 'id')
('w', 'id')
reduce
('f', 'Y')
('w', '$')
{')': 'e', '+': 'e', '*': '*FY', '$': 'e'}
['$', 'R']
('f', 'R')
('w', '$')
{')': 'e', '+': '+TR', '$': 'e'}
['$']
('f', '$')
('w', '$')
input string is valid
```

EXP 7: LR(1) PARSER:

CODE:

```
ACTION = {0 : {'id' : 5, '(' : 4},1 : {'+' : 6, '$' : '*'}, 2 : {'+' : -2, '*' : 7, ')' : -2, '$' : -2},3 : {'+' : -4, '*' : -4, ')' : -4, '$' : -4}, 4 :
{'id' : 5, '(' : 4},5 : {'+' : -6, '*' : -6, ')' : -6, '$' : -6},6 : {'id' : 5, '(' : 4},7 : {'id' : 5, '(' : 4},8 : {'+' : 6, ')' : 11},9 : {'+' : -1, '*' : 7,
')' : -1, '$' : -1},10 : {'+' : -3, '*' : -3, ')' : -3, '$' : -3},11 : {'+' : -5, '*' : -5, ')' : -5, '$' : -5}}
GOTO = {0 : {'E' : 1, 'T' : 2, 'F' : 3},4 : {'E' : 8, 'T' : 2, 'F' : 3},6 : {'T' : 9, 'F' : 3},7 : {'F' : 10}}
GRAMMAR = [None,('E' , ['E', '+', 'T']),('E' , ['T']),('T' , ['T' , '*', 'F']),('T' , ['F']),('F' , ['(', 'E', ')']),('F' , ['id'])]
sentence = input("Enter the statement: ").strip().split()
stack = ['$', 0]
i = 0
while True:
    if i >= len(sentence) or sentence[i] not in ACTION[stack[-1]]:
        print("Failed!!")
        break
    if ACTION[stack[-1]][sentence[i]] == '*':
        print("Success!!")
        break
    elif ACTION[stack[-1]][sentence[i]] < 0:
        A, B = GRAMMAR[-ACTION[stack[-1]][sentence[i]]]
        for _ in range(2*len(B)):
            stack.pop()
        stack.append(A)
        stack.append(GOTO[stack[-2]][A])
    else:
        stack.append(sentence[i])
        stack.append(ACTION[stack[-2]][sentence[i]])
        i += 1
```

SAMPLE INPUT AND OUTPUT:

```
Enter the statement: id + id * id + ( id + id ) $
Success!!
Enter the statement: ( id + id ) * id $
```

```
Success!!
Enter the statement: ( ( id + id ) * ( id + id ) * id ) * ( id + id ) $
Success!!
Enter the statement: ( id + id ) * id * id + id + id $
Success!!
Fail Cases
Enter the statement: ( id + id ) * id * id + id  id $
Failed!!
```

EXP 8: IF TO SWITCH:

CODE:

(i) LEX PROG:

```
%{
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}
%%
[ /t] ;
"if" return IF;
"else" return ELSE;
"printf" return PRINTF;
[a-zA-Z%][a-zA-Z0-9+]* {yylval = strdup(yytext); return ID;}
"{" return OB;
"}" return CB;
"(" return OP;
")" return CP;
[0-9]+ {yylval = atoi(yytext); return NUM;}
"==" return EQ;
";" return SC;
"," return COM;
\" {yylval = strdup(yytext);return QUOT;}
. return yytext[0];
%%
```

(ii) YACC PROG:

```
%{
#include<stdio.h>
int cnt=0;
%}
%token NUM IF ELSE ELIF PRINTF ID OB CB OP CP SC COM QUOT EQ
%start S
%%
S: if elif  | if else {printf("VALID PROG!");} ;
if: IF OP ID EQ NUM CP {printf("switch(%s)\n{\ncase %d:",$3,cnt);} text;
elif:elif elif | ELSE IF OP ID EQ NUM CP {cnt++; printf("case %d:",cnt);} text | else;
else: ELSE {printf("default:");} def ;
text: PRINTF OP QUOT ID QUOT COM ID CP SC {printf("printf(%s%s%s,%s);",$3,$4,$5,$7);};
def: PRINTF OP QUOT ID QUOT CP SC {printf("printf(%s%s%s);\n}",$3,$4,$5);};
%%
yyerror()
{
printf("Error!");
}
```

```
main()
{
yyparse();
}
```

INPUT FILE: (inp.txt)

```
if(a==0)
printf("%d",a);
else if(a==1)
printf("%d",a+1);
else if(a==2)
printf("%d",a+2);
else
printf("default");
```

OUTPUT:

```
guest-ybih0m@admin123-VirtualBox:~$ ./a.out <inp.txt
switch(a)
{
case 0:
printf("%d",a);
case 1:
printf("%d",a+1);
case 2:
printf("%d",a+2);

default:printf("default");
}
```

EXP 9A: THREE ADDRESS CODE:

CODE:

(i)LEX PROG:

```
%{
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}
%%
[ \t\n] ;
[0-9]+ {yylval=strdup(yytext); return NUM;}
[_a-zA-Z][_0-9a-zA-Z]* {yylval=strdup(yytext); return ID;}
"+" {yylval=strdup(yytext);return ADD;}
"-" {yylval=strdup(yytext);return SUB;}
"*" {yylval=strdup(yytext);return MUL;}
"/" {yylval=strdup(yytext);return DIV;}
">" {yylval=strdup(yytext);return GT;}
"<" {yylval=strdup(yytext);return LT;}
"=" {yylval=strdup(yytext);return EQ;}
";" {return SC;}
"," {return C0M;}
"{" {return OB;}
```

```
"}" {return CB;}
"(" {return OP;}
")" {return CP;}
.  return yytext[0];
%%
```

(ii)YACC PROG:

```
%{
#include "y.tab.h"
#include <stdio.h>
count=1;
%}

%token ID FS GT LT PL AK BS MI EQ AD OR XR MD IV QM CN SC CM OB CB OP CP IC

%%
S: statements {printf("\n\nvalid!!\n\n");} ;
statements: statement statements | statement ;
statement: ID EQ E {printf("%s = %s\n", $1, $3);};

E: E PL T {printf("t%d = %s %s %s\n", count, $1, $2, $3); sprintf($$, "t%d", count); count++;} | value {$$ = $1;}
| E MI T {printf("t%d = %s %s %s\n", count, $1, $2, $3); sprintf($$, "t%d", count); count++;} | value {$$ = $1;}
| T ;

T: T AK F {printf("t%d = %s %s %s\n", count, $1, $2, $3); sprintf($$, "t%d", count); count++;} | value {$$ = $1;}
| T BS F {printf("t%d = %s %s %s\n", count, $1, $2, $3); sprintf($$, "t%d", count); count++;} | value {$$ = $1;}
| F ;

F: OP E CP {$$ = $2;} | value {$$ = $1;};

value: IC |  ID {$$ = $1;};
operator: PL | AK | MI | BS {$$ = $1;};
%%

void yyerror(){
printf("Invalid Syntax!");
}
int main(){
yyparse();
return 0;
}
```

INPUT FILE:(inp.txt)

```
a=b+c*d
```

OUTPUT:

```
t1 = c * d
t2 = b + t1
a = t2
valid!!
```

EXP 9B: POSTFIX EXP:

CODE:

(i)LEX PROG:

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

```
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+ {yylval = atoi(yytext); return NUM;}
[a-z][a-zA-Z]* {yylval=strdup(yytext);return ID;}
[\n\t] return 0;
. return yytext[0];
%%
yywrap() {return 1;}
```

(ii)YACC PROG:

```
%{
#include <stdio.h>
#include <stdlib.h>
%}
%token NUM ID
%left '+''-'
%left '*''/'
%%
start:T;
T: T'+'T {printf("+");}
| T'-'T {printf("-"); }
| T'*'T {printf("*"); }
| T'/'T {printf("/");}
| NUM {printf("%d",$1);}
| ID {printf("%s",$1);}
;
%%
int main()
{
printf("\nEnter expression:");
yyparse();
return 0;
}
int yyerror()
{
printf("Error");
}
```

INPUT:

a+b*c

OUTPUT:

abc*+

EXP 10: COMMON SUB EXP ELIMINATION:

CODE:

```
import java.io.*;
import java.util.*;
import java.lang.*;
class mod
{
public static void main(String args[])throws IOException
{
String s,temp;
```

```
Scanner sc= new Scanner(System.in);
System.out.println("Enter the size");
int size=sc.nextInt();
String arr[][]=new String[size][2];
int flag=0,index=0;
BufferedReader br=new BufferedReader(new InputStreamReader(new FileInputStream("input.txt")));
for(;(s=br.readLine())!=null;flag=0)
{
arr[index][0]=s.substring(0,s.indexOf("="));
arr[index][1]=s.substring(s.indexOf("=")+1);
index++;
}
for(int i=1;i<arr.length;i++)
{
for(int j=i-1;j>=0;j--)
{
if(arr[i][1].equals(arr[j][1]))
{
arr[i][1]=arr[j][0];
break;
}
}
}
for(int i=0;i<arr.length;i++)
{
System.out.println(arr[i][0]+"="+arr[i][1]);
}
}
}
```

INPUT FILE:(input.txt)

```
a=b
b=a+e*f
c=b
d=a+e*f
```

OUTPUT:

```
Enter the size
4
a=b
b=a+e*f
c=a
d=b
```

EXP 11: CODE GENERATION(machine lang):

CODE:

(i) LEX PROG:

```
%{
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}
%%
```

```
[0-9]+ {yylval=strdup(yytext); return IC;}
[_a-zA-Z][_0-9a-zA-Z]* {yylval=strdup(yytext); return ID;}
"+" {yylval=strdup(yytext);return PL;}
"-" {yylval=strdup(yytext);return MI;}
"*" {yylval=strdup(yytext);return AK;}
"/" {yylval=strdup(yytext);return BS;}
"\\" {yylval=strdup(yytext);return FS;}
">" {yylval=strdup(yytext);return GT;}
"<" {yylval=strdup(yytext);return LT;}
"=" {yylval=strdup(yytext);return EQ;}
"&" {yylval=strdup(yytext);return AD;}
"|" {yylval=strdup(yytext);return OR;}
"^" {yylval=strdup(yytext);return XR;}
"%" {yylval=strdup(yytext);return MD;}
"~" {yylval=strdup(yytext);return IV;}
"?" {yylval=strdup(yytext);return QM;}
":" {yylval=strdup(yytext);return CN;}
";" {return SC;}
"," {return CM;}
"{" {return OB;}
"}" {return CB;}
"(" {return OP;}
")" {return CP;}
%%
```

(ii)YACC PROG:

```
%{
#include "y.tab.h"
#include <stdio.h>
count=1;
%}

%token ID FS GT LT PL AK BS MI EQ AD OR XR MD IV QM CN SC CM OB CB OP CP IC

%%
S: statements {printf("\n\nvalid!!\n\n");} ;
statements: statement statements | statement ;
statement: ID EQ E SC {printf("STR[%s], %s\n\n", $1, $3);};
E: value operator value {
$$=$1;
if (!strcmp($2, "+"))
printf("ADD %s, %s\n", $1, $3);
else if (!strcmp($2, "-"))
printf("SUB %s, %s\n", $1, $3);
else if (!strcmp($2, "*"))
printf("MUL %s, %s\n", $1, $3);
else if (!strcmp($2, "/"))
printf("DIV %s, %s\n", $1, $3);
};
value: IC {printf("LOAD R%d, %s\n", count, $1); sprintf($$, "R%d", count++);}
| ID {printf("LOAD R%d, [%s]\n", count, $1); sprintf($$, "R%d", count++);};
operator: PL | AK | MI | BS {$$ = $1;};
%%
void yyerror(){
```

```
printf("Invalid Syntax!");
}
int main(){
yyparse();
return 0;
}
```

INPUT FILE (input.txt):

```
a=b+c;
c=a*d;
```

OUTPUT:

```
LOAD R1, [b]
LOAD R2, [c]
ADD R1, R2
STR[a], R1
LOAD R3, [a]
LOAD R4, [d]
MUL R3, R4
STR[c], R3
valid!!
```

EXP 12: LOCAL LIST SCHEDULING:

CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<fstream>
#include<list>
#include<queue>
#include<process.h>
using namespace std;
#define size 9
int rk=0;int k=0;
int cycle;

struct node{
        int num;
        int op;
   int op1;
   int op2;
   int result;
   int latency;
   int state;
}input[size+1];

struct readylist{
        int num;
   int op;
   int latency;
   }Rlist[size+1],Alist[size+1];

void In(){
        fstream in;
   int step=0,num=0,op=0,op1=0,op2=0,result=0,latency=0,count=1;
        char  filename[20];
```

```
        int i;
   printf("\nEnter Input filename(ex: DFG1.txt): ");
        scanf("%s",filename);
                in.open(filename);
   if(!in){
                printf("can not open this filename\n");
        exit(1);
        }
   while( in>>num>>op>>op1>>op2>>result>>latency ){
        input[count].num=num;
      input[count].op=op;
      input[count].op1=op1;
      input[count].op2=op2;
      input[count].result=result;
      input[count].latency=latency;
        count++;
   }
        in.close();

        for(int i=1;i<=size;i++)
        input[i].state=0;
}

void sortlist(readylist Rlist[size])
{       readylist t;
        for(int i=1;i<=rk;i++)
        for(int j=1;j<=rk;j++)
        if(Rlist[i].latency<Rlist[j].latency)
        {       t=Rlist[i];
                Rlist[i]=Rlist[j];
                Rlist[j]=t;
        }
}


void list_scheduling(){
        int i,j;
   //to detect data dependency
   for(i=1;i<=size;i++){
      for( j=i-1;j>=1;j--){
        //finding no predecessor node
        if(input[i].op1==input[j].result||input[i].op2==input[j].result)
          break;
        }
      if(j==0)
        {
        rk++;
        Rlist[rk].op=i;Rlist[rk].latency=input[i].latency;Rlist[rk].num=input[i].num;
                        }

   }

        cycle=1;
```

```
while(rk>0||k>0)
{
for(int e=1;e<=k;e++){
        int h=Alist[e].op;
        if((input[h].state+input[h].op)<=cycle)
        {
        int v=1;
        while(v<=k)
                {Alist[v]=Alist[v+1];v++;}
        k--;
        //finding successors node
        for(i=h+1;i<=size;i++){
if(input[h].result==input[i].op1||input[h].result==input[i].op2)
        {
                int m;
                        for( m=1;m<=rk;m++)
                        {               if(Rlist[m].num==i||input[i].state>0) break;

                        }
                if(m>rk)
                {
        int y;int flag=1,pc=0;
                for( y=i-1;y>=1;y--){
        //finding predecessor node
                if(input[i].op1==input[y].result||input[i].op2==input[y].result)
                { ++pc;
                                        if(input[y].state+input[y].op<cycle&&pc<2)
                                        flag=0;
                }
}
                if(flag==1)
                {       rk++;
                Rlist[rk].op=input[i].num;Rlist[rk].latency=input[i].latency;Rlist[rk].num=i;
                sortlist(Rlist);
                break;
                }
}
                }
        }

        }
}

        if(rk>0)
        {  sortlist(Rlist);
                input[Rlist[rk].num].state=cycle;
                Alist[++k].op=Rlist[rk].num;
                --rk;
        }
        cycle++;
}
```

```
for(int i=1;i<=size;i++)
        for(int j=1;j<=size;j++)
        if(input[i].state<input[j].state)
        {       node t=input[i];
                input[i]=input[j];
                input[j]=t;
        }
}

int main(){

   In();
   printf("Input given");
   printf("\nnumber\toperation ");
   for(int i=1;i<=size;i++)
                printf("\n%d\t%d %d %d\t",input[i].num,input[i].op,input[i].op1,input[i].result);
   list_scheduling();
   printf("\n\nScheduler output");
  printf("\nstate\toperation ");
   for(int i=1;i<=size;i++)
                printf("\n%d\t%d",input[i].state,input[i].num);
   system("pause");
        return 0;
}
```

INPUT FILE:(dfg2.txt)

```
1 3 1 2 6 13
2 1 6 6 6 10
3 3 1 3 7 12
4 2 6 7 6 9
5 3 1 4 8 10
6 2 6 8 6 7
7 3 1 5 7 8
8 2 6 7 6 5
9 3 6 0 1 3
```

Inputs are in the form:
operationNum>>NumberOfCyclesInOperation>>op1(reg)>>op2(reg)>>result(reg)>>latency(total num of cycles)
Eg: loadAI rarp, @a => r1 is written as 1 3 1 2 6 13
1(opnNum) 3(num of cycles for loadAI) 1(reg num for rarp) 2(reg num for a) 6(stored in reg..named as 6) 13(total weight)

OUTPUT:

```
Enter Input filename(ex: DFG1.txt): dfg2.txt
Input given
number  operation
1       3 1 6
2       1 6 6
3       3 1 7
4       2 6 6
5       3 1 8
6       2 6 6
7       3 1 7
8       2 6 6
9       3 6 1

Scheduler output
state   operation
1       1
2       3
3       5
4       2
5       4
6       7
7       6
9       8
11      9Press any key to continue . . .
```

LOCAL LIST SCHEDULING: (Using Python)

```
from collections import defaultdict, Counter, deque
from heapq import *

# get delay of each instruction
delay = {}
print("Enter the operations and cost: ")
while True:
    s = input()
    if not s: break
    i, c = s.split()
    delay[i] = int(c)

# get dependency graph as input
successors = defaultdict(list)
predecessors = defaultdict(list)
in_degree =  Counter()
out_degree = Counter()
print("Enter the edges: ")
while True:
    s = input()
    if not s: break
    u, v = s.split()
    successors[u].append(v)
    predecessors[v].append(u)
    in_degree[v] += 1
    out_degree[u] += 1

# calculate priority
queue = deque([i for i in predecessors if not successors[i]])
priority = Counter()
while queue:
    node = queue.popleft()
    priority[node] = max(priority[node], delay[node])
    for predecessor in predecessors[node]:
        priority[predecessor] = max(priority[predecessor], priority[node]+delay[predecessor])
        out_degree[predecessor] -= 1
        if not out_degree[predecessor]:
            queue.append(predecessor)

# local list scheduling
cycle = 1
ready = [(-priority[op], op) for op in delay if not in_degree[op]]
heapify(ready)
```

```
active = []
while ready or active:
    to_remove = set()
    for t,op in active:
        if t+delay[op] <= cycle:
            to_remove.add(op)
            for successor in successors[op]:
                in_degree[successor] -= 1
                if not in_degree[successor]:
                    heappush(ready, (-priority[successor], successor))
    active = [(t, op) for t,op in active if op not in to_remove]
    print(f"{cycle} [{' '.join(op for p,op in ready)}]      [{f' '.join(op for t,op in active)}]")
    if ready:
        p,op = heappop(ready)
        active.append((cycle, op))
    cycle += 1
print(f"Total cycles: {cycle}")

"""
Enter the operations and cost:
a 3
b 1
c 3
d 2
e 3
f 2
g 3
h 2
i 3

Enter the edges:
a b
b d
c d
d f
e f
f h
g h
h i

1 [a c e g]      []
2 [c g e]      [a]
3 [e g]      [a c]
4 [b g]      [c e]
5 [d g]      [e]
6 [g]      [d]
7 [f]      [g]
8 []      [g f]
9 [h]      []
10 []      [h]
11 [i]      []
12 []      [i]
13 []      [i]
14 []      []
Total cycles: 15
"""
```

EXTRA PROGRAMS:

1.EXPRESSION EVALUATION:

LEX PROG:

```
%{
#include<stdio.h>
#include "y.tab.h"
%}
%%
[ \t\n] ;
[0-9]+ {yylval=atoi(yytext); return NUM;}
[A-Za-z][A-Za-z][0-9]*     return ID;
"+" return ADD;
"-" return SUB;
"*" return MUL;
"/" return DIV;
.           return yytext[0];
%%
```

YACC PROG:

```
%{
#include<stdio.h>
%}
%token ID NUM ADD SUB MUL DIV
%start G
%%
G : E      {printf("Expression value : %d\n",$1);exit(0);};
E : E ADD T{ $$ = $1 + $3;}| E SUB T{ $$ = $1 - $3;}| T{ $$ = $1;};
T : T MUL F{ $$ = $1 * $3;}| T DIV F{ $$ = $1 / $3;}| F{ $$ = $1;};
F : NUM{ $$ = $1;}| '('E')'{ $$ = $2;};
%%
int yyerror(){printf("error.\n");}
main()
{
yyparse();
}
```

INPUT FILE:(inp.txt):

```
2+5*8
```

OUTPUT:

```
Expression value : 42
```

2.FUNCTION:

LEX PROG:

```
%{
#include "y.tab.h"
%}
%%
[ \t\n]   ;
void|int|if|while          return kw;
swap     return swap;
[a-zA-Z]+          return id;
```

```
[0-9]+   return num;
","      return com;
";"      return sc;
"("      return op;
")"      return cp;
"{"      return ob;
"}"      return cb;
"+"|"="|"*"|"<"|">="|"++"      return opr;
.        ;
%%
```

YACC PROG:

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
%token kw swap ifs whiles id num com sc op cp ob cb opr
%start S
%%
S:stmt   {printf("valid");};
stmt: kw swap op arg cp ob body cb;
arg: kw id|arg com kw id ;
body: decl cond1 ob st1 loop st2 cb;
decl: kw id sc;
cond1:kw op id opr id opr id cp;
st1:id opr sc id opr id sc;
loop:kw op id opr num cp;
st2:id opr id opr id sc;
%%
yyerror()
{
printf("error");
}
int main(){
yyparse();
}
```

INPUT FILE(smp.txt):

```
void swap(int a,int b,int c)
{ int t;
if(j>=x+y){
j++;
x=j;
while(x<100)
y=x*j;
}
}
```

3.Lex program to count the number of words, characters, Special Symbol, blank spaces and lines.

LEX PROG:

```
%{
#include <stdio.h>
#include <string.h>
int words = 0, symbols = 0, chars = 0, blankspaces = 0, lines = 0;
%}
%%
[a-zA-Z]+          {words++; chars += strlen(yytext);}
[!@#$%^&*\\()\-=_+]    {symbols++;chars++;}
" "        {blankspaces++;}
"\n"       {lines++;}
%%
int main()
{
        yylex();
        printf("words: %d\n", words);
        printf("chars: %d\n", chars);
        printf("symbols: %d\n", symbols);
        printf("blankspaces: %d\n", blankspaces);
        printf("lines: %d\n", lines);
}
```

INPUT FILE:

```
HELLO THIS
IS SAMPLE INPUT $ $
$ % HELLO
```

OUTPUT:

```
words: 6
chars: 31
symbols: 4
blankspaces: 7
lines: 3
```

4. Use YACC to Convert Binary to Decimal

```
/* C2: Use YACC to Convert Binary to Decimal (including fractional numbers). */
File: C2.y
/* definition section*/
%{
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
//#define YYSTYPE double
void yyerror(char *s);
float x = 0;
%}
// creating tokens whose values are given by lex
%token ZERO ONE POINT
// following a grammer rule which is converting binary number
to decimal number (float value)
%%
L: X POINT Y {printf("%f",$1+x);}
| X {printf("%d", $$);}
X: X B {$$=$1*2+$2;}
```

```
| B {$$=$1;}
Y: B Y {x=$1*0.5+x*0.5;}
| {;}
B:ZERO {$$=$1;}
|ONE {$$=$1;};
%%
// main function
int main()
{
printf("Enter the binary number : ");
// calling yyparse function which execute grammer rules
and lex
while(yyparse());
printf("\n");
}
// if any error
void yyerror(char *s)
{
fprintf(stdout,"\n%s",s);
}
1
```

File: C2.l

```
/* definitions */
%{
// including required header files
#include<stdio.h>
#include<stdlib.h>
#include"y.tab.h"
// declaring a external variable yylval
extern int yylval;
%}
/* rules
if 0 is matched ,make yylval to 0 and return ZERO which
is variable in Yacc program
if 1 is matched ,make yylval to 1 and return ONE which
is variable in Yacc program
if . is matched ,return POINT which is variable in Yacc
program if line change , return 0
otherwise ,ignore*/
%%
0 {yylval=0;return ZERO;}
1 {yylval=1;return ONE;}
"." {return POINT;}
[ \t] {;}
\n return 0;
%%
```

5. Expression values evaluation (Desktop calculator)

/* C3: Use YACC to implement: Expression values evaluation (Desktop calculator). */

File C3.y

```
/* definition section*/
%{
#include <stdio.h>
#include <ctype.h>
int x[5],y[5],k,j[5],a[5][10],e,w;
%}
// creating tokens whose values are given by lex
%token digit
// following a grammer rule which is printing the digit
```

```
first then solving
// the expression of additiion
,subtraction,multiplication,and power .
%%
S : E { printf("\nAnswer : %d\n",$1); }
;
E : T { x[e]=$1; } E1 { $$=x[e]; }
;
E1 : '+' T { w=x[e]; x[e]=x[e]+$2; printf("Addition Operation
%d and %d : %d\n",w,$2,x[e]); } E1 { $$=x[e]; }
| '-' T { w=x[e]; x[e]=x[e]-$2; printf("Subtraction Operation
%d and %d : %d\n",w,$2,x[e]); } E1 { $$=x[e]; }
| { $$=x[e]; }
;
T : Z { y[e]=$1; } T1 { $$=y[e]; }
;
T1 : '*' Z { w=y[e]; y[e]=y[e]*$2; printf("Multiplication
Operation of %d and %d : %d\n",w,$2,y[e]); } T1 { $$=y[e];
} | { $$=y[e]; }
;
Z : F { a[e][j[e]++]=$1; } Z1 { $$=$3; }
;
Z1 : '^' Z { $$=$2; }
| { for(k=j[e]-1;k>0;k--) { w=a[e][k-1]; a[e][k
1]=powr(a[e][k-1],a[e][k]); printf("Power Operation %d ^ %d :
%d\n",w,a[e][k],a[e][k-1]); } $$=a[e][0]; j[e]=0; } ;
F : digit { $$=$1; printf("Digit : %d\n",$1); }
| '(' { e++; } E { e--; } ')' { $$=$3; }
1
;
%%
int main()
{
//initializing all the variables to zero
for(e=0;e<5;e++) { x[e]=y[e]=0; j[e]=0; }
e=0;
// takes input as a expression
printf("Enter an expression\n");
yyparse();
return 0;
}
// if any error yyerror will be called
yyerror()
{
printf("NITW Error");
}
// when the input is finished yywrap is called to exit the
code int yywrap()
{
return 1;
}
// power function to calculate m ^ n
int powr(int m,int n)
{
int ans=1;
while(n) { ans=ans*m; n--; }
return ans;
}
```

File C3.l

```
/* definitions */
```

```
%{
// including required header files
#include "y.tab.h"
#include <stdlib.h>
// declaring a external variable yylval
extern int yylval;
%}
%%
//If the token is an Integer number,then return it's
value. [0-9]+ {yylval=atoi(yytext);return digit;}
//If the token is space or tab,then just ignore
it. [\t] ;
//If the token is new line,return 0.
[\n] return 0;
//For any other token, return the first character read since
the last match.
. return yytext[0];
%%
```

6. Implement a Program to Accept a Context Free Grammar and to print the First set in Top Down Parsing using Python

```
class LLparser:
def __init__(self,grammar):
self.grammar=grammar
self.first={}

def ac_compute_first(self,symbol,vis=None):
if vis == None:
vis=set()
if symbol in vis:
return set()

vis.add(symbol)

first_set=set()

productions=self.grammar[symbol]

for production in productions:
if len(production) == 0:
first_set.add(' ');
else:
f_s=production[0]

if f_s.isupper():
d_s=f_s
i=1
while True:
assets,boolean=self.get_E(d_s)

if len(assets)>0:
```

```
first_set|=assets
if boolean:
if i<len(production):
d_s=production[i]
i+=1

else:
break
else:
if d_s.islower():
first_set.add(d_s)
break

first_set|=self.ac_compute_first(f_s,vis)
else:
first_set.add(f_s)

return first_set

def get_E(self,symbol):

m_set=set()
bol_val=False

if symbol.islower():
bol_val=False

productions=self.grammar[symbol]

for production in productions:
if production[0]=='e':
bol_val=True
if production[0].islower():
m_set.add(production[0])

return m_set,bol_val

def compute_first(self):
for non_term in self.grammar:
self.first[non_term]=self.ac_compute_first(non_term)

def get_first(self,symbol):
return self.first[symbol]

grammar={}
n=int(input("Enter number of productions"))

productions=input("Enter productions").split()

for production in productions:
prods=production[2::].split(",")
grammar[production[0]]=prods
```

```
parser=LLparser(grammar)

parser.compute_first()

for key in grammar:
print(f"First of {key}:",parser.get_first(key))

// code end
```

```
Enter number of productions4
Enter productionsS=ABC A=a,e B=b,e C=c,e
First of S: {'b', 'e', 'c', 'a'}
First of A: {'e', 'a'}
First of B: {'b', 'e'}
First of C: {'e', 'c'}
```