Computer Science Engineering Department

Language Processors(CS354)

Lab Record

G CHARAN SRI
SAI SHANKAR
411922
III Year

Task1:

1) Lex program to identify and count the tokens.

Code:

LexProgram:

```
%{
int total =0 ; %}

%option noyywrap
%%
#.* {total++; fprintf(yyout,"This is Predefined function: %s\n\n",yytext);}

["|,|;(|)|{|}|.|_] {total++; fprintf(yyout,"This is Delimiter: %s\n\n",yytext);}

[[]] {total++; fprintf(yyout,"This is Delimiter: %s\n\n",yytext);}

"#"|"@"|"$"|"^"|"%"|"^"|"&" {total++; fprintf(yyout,"This is Special Characters:
%s\n\n",yytext);}

"["|"]" {total++; fprintf(yyout,"This is Delimiter: %s\n\n",yytext);}

"=" {total++; fprintf(yyout,"This is Assignment Operator: %s\n\n",yytext);}

"+"|"-"|"*"|"/" {total++; fprintf(yyout,"This is Arithmatic Operator:
%s\n\n",yytext);}

LP LabWork 1

"and"|"or"|"not"|"nand"|"xor"|"nor"|"xnor" {total++; fprintf(yyout,"This is
Logical Operators: %s\n\n",yytext);}

"<="|">="|"++"|"!="|"=="|"<"|">" {total++; fprintf(yyout,"This is Relational
Operator: %s\n\n",yytext);}

("int")|("if")|("else")|("while")|("do")|("break")|("continue")|
("double")|("float")|("return")|("EOF") {total++; fprintf(yyout,"This is Keyword:
%s\n\n",yytext);}

("return")|("char")|("case")|("sizeof")|("long")|("short")|("typedef")|
("switch")|("unsigned")|("void")|("static")|("struct")|("goto") {total+
+;fprintf(yyout,"This is Keyword:%s\n",yytext);}

[a-zA-Z_][a-zA-Z0-9_]*\( {total++; fprintf(yyout,"This is Function:
%s\n\n",yytext);}

[a-zA-Z_][a-zA-Z0-9_]* {total++; fprintf(yyout,"This is Identifier:
%s\n\n",yytext);}
```

```
[0-9]*"."[0-9]+ {total++;fprintf(yyout,"This is Fraction : %s\n\n", yytext);}

[-][0-9]*"."[0-9]+ {total++;fprintf(yyout,"This is Negative Fraction : %s\n\n",
yytext);}

[0-9]+ {total++; fprintf(yyout,"This is Integer: %s\n\n",yytext);}

"-"[0-9]+ {total++; fprintf(yyout,"This is Negative Integer: %s\n\n",yytext);}

["]([^"\\\n]|\\.|\\\n)*["] {total++; fprintf(yyout,"this is String: %s\n\n",yytext);}

"//".*" "* {total++;fprintf(yyout,"this is single line Commments:
%s\n\n",yytext);}
```

LP LabWork 2

```
\/\*(.*\n)*.*\*\/ {total++;fprintf(yyout,"this is multi line Comments:
%s\n\n",yytext);}

. {fprintf(yyout,"",yytext);}

[\t\n]+ %%

main()
{
extern FILE *yyin, *yyout;

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

yyout = fopen("Output.txt", "w"); yylex();

fprintf(yyout,"\n\n\n\n\n Total Tokens = %d",total);

return 0; }
```

2) C program to read CFG productions and store into corresponding data structures.

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
int i,j,k,l,m,n=0,o,p,nv,z=0,t,x=0;
char str[10],temp[20],temp2[20],temp3[20];
struct prod {
    char lhs[10],rhs[10][10];
```

```
int n; }pro[10];
void findter() {
for(k=0;k<n;k++) {
LP LabWork 4

if(temp[i]==pro[k].lhs[0]) {
for(t=0;t<pro[k].n;t++) {
for(l=0;l<20;l++) temp2[l]='\0';
for(l=i+1;l<strlen(temp);l++) temp2[l-i-1]=temp[l];
for(l=i;l<20;l++) temp[l]='\0';
for(l=0;l<strlen(pro[k].rhs[t]);l++) temp[i+l]=pro[k].rhs[t]
[l];
strcat(temp,temp2); if(str[i]==temp[i])
return;
else if(str[i]!=temp[i] && temp[i]>=65 && temp[i]<=90)
break; }
break; }
}
if(temp[i]>=65 && temp[i]<=90)
findter(); }
void main() {
FILE *f;
for(i=0;i<10;i++) pro[i].n=0;
f=fopen("input.txt","r");
while(!feof(f))
{
fscanf(f,"%s",pro[n].lhs);
if(n>0)
{
if( strcmp(pro[n].lhs,pro[n-1].lhs) == 0 ) {
pro[n].lhs[0]='\0';
fscanf(f,"%s",pro[n-1].rhs[pro[n-1].n]); pro[n-1].n++;
continue;
}
}
fscanf(f,"%s",pro[n].rhs[pro[n].n]);
 pro[n].n++;
n++;
}
n--;
printf("\n\nTHE GRAMMAR IS AS FOLLOWS\n\n");
for(i=0;i<n;i++)
for(j=0;j<pro[i].n;j++)
LP LabWork 5
```

```
printf("%s -> %s\n",pro[i].lhs,pro[i].rhs[j]);
while(1) {
for(l=0;l<10;l++) str[0]=NULL;
printf("\n\nENTER ANY STRING ( 0 for EXIT ) : ");
scanf("%s",str);
if(str[0]=='0') exit(1);
for(j=0;j<pro[0].n;j++) {
for(l=0;l<20;l++) temp[l]=NULL;
strcpy(temp,pro[0].rhs[j]);
m=0; for(i=0;i<strlen(str);i++) {
if(str[i]==temp[i])
m++;
else if(str[i]!=temp[i] && temp[i]>=65 && temp[i]<=90) {
findter(); if(str[i]==temp[i])
m++; }
else if( str[i]!=temp[i] && (temp[i]<65 || temp[i]>90) )
break;
}
if(m==strlen(str) && strlen(str)==strlen(temp)) {
printf("\n\nTHE STRING can be PARSED !!!");
break;
}
}
if(j==pro[0].n)
printf("\n\nTHE STRING can NOT be PARSED !!!");
}
getch();
}
```

3) a- **LeftRecursion Elimination:**

Code:

```
#include<stdio.h>
 #include<string.h>
 #define SIZE 10
 int main () {
     char non_terminal;
      char beta,alpha;
      int num;
      char production[10][SIZE];
      int index=3;
      printf("Enter Number of Productions : ");
```

```
       scanf("%d",&num);
      printf("Enter the grammar:\n");
      for(int i=0;i<num;i++){
          scanf("%s",production[i]);
      }
       for(int  i=0;i<num;i++){
           printf("\nGrammar %s",production[i]);
           non_terminal=production[i][0];
           if(non_terminal==production[i][index]) {
               alpha=production[i][index+1];
               printf(" is left recursive.\n");
               while(production[i][index]!=0 &&
production[i][index]!='|')
recursion:\n"); LP LabWork

7

    index++;
if(production[i][index]!=0) {
    beta=production[i][index+1];
    printf("Grammar without left
                    printf("%c-
>%c%c\'",non_terminal,beta,non_terminal);
                    printf("\n%c\'->%c%c\'|
E\n",non_terminal,alpha,non_terminal);
}

else

\n");
}

else

printf(" can not be reduced further
    printf(" is atlast not left recursive.\n");
index=3;
} }
```

3)b- LeftFactoring Elimination:
Code:

```
#include<stdio.h>
 #include<string.h>
 int main()
```

```
{

  char;

grammar[20],first[20],second[20],modifiedGrammar[20],newGram
mar[20],tempGrammar[20];
       int i,j=0,k=0,l=0,pos;
       printf("Enter Production : A->");
       gets(grammar);
       for(i=0;grammar[i]!='|';i++,j++)
            first[j]=grammar[i];
       first[j]='\0';
      for(j=++i,i=0;grammar[j]!='\0';j++,i++)
            second[i]=grammar[j];
       second[i]='\0';
       for(i=0;i<strlen(first)||i<strlen(second);i++)
       {
            if(first[i]==second[i])
            {
                 modifiedGrammar[k]=first[i];
                 k++;
                 pos=i+1;
} }

       for(i=pos,j=0;first[i]!='\0';i++,j++){
          newGrammar[j]=first[i];
       }
       newGrammar[j++]='|';
       for(i=pos;second[i]!='\0';i++,j++){
          newGrammar[j]=second[i];
       }
       modifiedGrammar[k]='X';
       modifiedGrammar[++k]='\0';
       newGrammar[j]='\0';
       printf("\n The modified grammar is: A-
>%s",modifiedGrammar);
       printf("\n The new grammar is: X->%s\n",newGrammar);
  }
```

Task2:

1) construct a program to find first set of given grammar

Code:

```
#include<stdio.h>
#include<ctype.h>
void FIRST(char);
int search(char);
int count,n=0,d=0;
char prodn[10][10], first[10];
char NonTerminals[10];
int main() {

    int i,choice;
    char c,ch;



    printf("Enter the no.of productions: ");
    scanf("%d",&count);
```

```
    printf("Enter the productions(epsilon=$):\n");
    for(i=0;i<count;i++)
        scanf("%s%c",prodn[i],&ch);
    for(int k=0;k<count;k++)
    {
        if(search(prodn[k][0])==1)
        {
            NonTerminals[d] = prodn[k][0];
            d++;
            n=0;
            FIRST(prodn[k][0]);
            printf("FIRST(%c)= {",prodn[k][0]);
            for(i=0;i<n;i++)
                printf("%c ",first[i]);
            printf("}\n");
} }

return 0; }

void FIRST(char c)
{
    int j;
    if(!(isupper(c)))first[n++]=c;
    for(j=0;j<count;j++)
    {
        if(prodn[j][0]==c)
        {
[2];
}

} }

else FIRST(prodn[j][2]);
int search(char s) Lp Lab Task

2

if(prodn[j][2]=='$') first[n++]='$';
else if(islower(prodn[j][2]))first[n++]=prodn[j]
{
    for(int y=0;y<=d;y++)
    {
        if(s==NonTerminals[y])
            return 0;
}
```

```
return 1; }
```

2) construct a program to !ind follow set of given grammar

Code:

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

    int i,z;
    char c,ch;
    printf("Enter the no.of productions:");
    scanf("%d",&n);
    printf("Enter the productions(epsilon=$):\n");
    for(i=0;i<n;i++)
        scanf("%s%c",a[i],&ch);
    for(int h=0;h<n;h++)
    {
        if(search(a[h][0])==1)
        {
            NonTerminals[d] = a[h][0];
            d++;
            m=0;
            follow(a[h][0]);
            printf("FOLLOW(%c) = {",a[h][0]);
            for(i=0;i<m;i++)
                printf("%c ",f[i]);


            printf("}\n");
        }
} }
```

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

} }

int search(char s)
{
    for(int y=0;y<=d;y++)
    {


        if(s==NonTerminals[y])
            return 0;
}

return 1; }
```

Task3:

Q1) write yacc specification for expression grammar

Code:

LexFile:

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

%%
[a-zA-Z_][a-zA-Z_0-9]* return id;
[0-9]+(\.[0-9]*)? return num;
[+/*] return op;
. return yytext[0];
\n return 0;
%%

int yywrap()
{
```

```
    return 1;
}
```

YaccFile:

```
%{
#include<stdio.h>
int valid = 1;
int yylex();
int yyerror();
%}

%token num id op

%%
start : id
start : id '=' s ';'
s : id x
    | num x
    | '-' num x
    | '(' s ')' x
    ;
x:  op s
    | '-' s
    |
    ;
%%

int yyerror()
{
    valid = 0;
    printf("Invalid Expression\n");
    return 0;
}

int main()
{
    printf("Enter Expression: ");
```

```
  yyparse();
    if(valid)
        printf("Valid Expression\n");
}
```

Q2)write yacc specification for more advanced desk calculator with error recovery (with ambiguous grammar and with unambiguous grammar)

Code:

LexFile:

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

%%
[0-9]+ {
    yylval = atoi(yytext);
  return NUMBER;
}
'<=' return LE;
'>=' return GE;
'!=' return NE;
'==' return EQ;
[\t] ;
[\n] return 0;

. return yytext[0];
%%
```

Yacc File:

```
%{
#include<stdio.h>
int flag=0;
int yylex();
void yyerror(const char* s);
%}

%token NAME NUMBER
%left GE LE EQ NE EE '<' '>'
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
%nonassoc UMINUS

%%
ArithmeticExpression: E {
         printf("Answer: %d\n", $$);
         return 0;
        };
 E    : E '+' E {$$ = $1 + $3;}
     | E '-' E {$$ = $1 - $3;}
     | E '*' E {$$ = $1 * $3;}
  | E '/' E {$$ = $1 / $3;}
     | E '%' E {$$ = $1 % $3;}
     | '(' E ')' {$$ = $2;}
     | NUMBER {$$ = $1;}
     | E GE E {$$ = $1 >= $3 ;}
     | E LE E {$$ = $1 <= $3 ;}
     |E NE E {$$ = $1 != $3 ;}
     | E EE E {$$ = $1 == $3 ;}
     | UMINUS E {$$ = -$1 ;}
     ;
%%
int main()
{
```

```
   yyparse();
}
void yyerror(const char* s) {}
int yywrap()
{
    return 1;
}
```

Task4:

1) Implementation of LL(1) PARSER:

Code:

```
#include<stdio.h> #include<ctype.h> #include<string.h> #include<stdlib.h>
void followfirst(char , int , int); void findfirst(char , int , int); void
follow(char c);

int count,n=0;
char calc_first[10][100];
char calc_follow[10][100];
int m=0;
char production[10][10], first[10]; char f[10];
int k;
char ck;
int e;

int main(int argc,char **argv) { int jm=0;

int km=0;
int i,choice;
char c,ch;
printf("Enter the no of Productions:"); scanf("%d",&count);
printf("\nEnter %d productions:\n\n",count); for(i=0;i<count;i++)
{
```

```
        scanf("%s%c",production[i],&ch);
    }
int kay;
char done[count];
int ptr = -1; for(k=0;k<count;k++){

for(kay=0;kay<100;kay++){

            calc_first[k][kay] = '!';
        }
}
int point1 = 0,point2,xxx; for(k=0;k<count;k++)
{

c=production[k][0];
point2 = 0;
xxx = 0;
for(kay = 0; kay <= ptr; kay++)

if(c == done[kay]) xxx = 1;

if (xxx == 1) continue;

findfirst(c,0,0);
ptr+=1;
done[ptr] = c;
printf("\n First(%c)= { ",c); calc_first[point1][point2++] = c;
for(i=0+jm;i<n;i++){

int lark = 0,chk = 0; for(lark=0;lark<point2;lark++){

if (first[i] == calc_first[point1][lark]){ chk = 1;

break; }

}
if(chk == 0){

                printf("%c, ",first[i]);
                calc_first[point1][point2++] = first[i];
} }

        printf("}\n");
        jm=n;
        point1++;
```

```
} printf("\n ---------------------------- \n\n"); char donee[count];
ptr = -1;
for(k=0;k<count;k++){

for(kay=0;kay<100;kay++){

            calc_follow[k][kay] = '!';
        }
}
point1 = 0;
int land = 0; for(e=0;e<count;e++)

{
ck=production[e][0];
point2 = 0;
xxx = 0;
for(kay = 0; kay <= ptr; kay++)

if(ck == donee[kay]) xxx = 1;

if (xxx == 1) continue;

          land += 1;
        follow(ck);
        ptr+=1;
        donee[ptr] = ck;
printf(" Follow(%c) = { ",ck); calc_follow[point1][point2++] = ck;
for(i=0+km;i<m;i++){

int lark = 0,chk = 0; for(lark=0;lark<point2;lark++){

if (f[i] == calc_follow[point1][lark]){ chk = 1;

break; }

}
if(chk == 0){

                 printf("%c, ",f[i]);
                calc_follow[point1][point2++] = f[i];
} }

          printf(" }\n\n");
        km=m;
point1++; }

char ter[10]; for(k=0;k<10;k++){
```

```
        ter[k] = '!';
    }
int ap,vp,sid = 0; for(k=0;k<count;k++){

for(kay=0;kay<count;kay++){
if(!isupper(production[k][kay]) && production[k][kay]!= '#' &&

production[k][kay] != '=' && production[k][kay] != '\0'){
                vp = 0;
for(ap = 0;ap < sid; ap++){ if(production[k][kay] == ter[ap]){

vp = 1;

break; }

}
if(vp == 0){

ter[sid] = production[k][kay];
sid ++; }

} }

}
ter[sid] = '$';
sid++;
printf("\n\t\t\t\t\t\t\t The Parsing Table :-"); printf("\n\t\t\t\t\t\t\t^^^^^^^\n");
printf("\n\t\t\t========\n"); printf("\t\t\t\t|\t");
for(ap = 0;ap < sid; ap++){

    printf("%c\t\t",ter[ap]);
}
printf("\n\t\t\t==========\n"); char first_prod[count][sid];
for(ap=0;ap<count;ap++){

int destiny = 0;
k = 2;
int ct = 0;
char tem[100]; while(production[ap][k] != '\0'){

if(!isupper(production[ap][k])){ tem[ct++] = production[ap][k]; tem[ct++] = '_';
tem[ct++] = '\0';

k++;

break; }
```

```
else{
int zap=0;

int tuna = 0; for(zap=0;zap<count;zap++){

if(calc_first[zap][0] == production[ap][k]){ for(tuna=1;tuna<100;tuna++){

if(calc_first[zap][tuna] != '!'){

                        tem[ct++] = calc_first[zap][tuna];
                    }
else break;

} break;

} }

            tem[ct++] = '_';
        }
k++; }

int zap = 0,tuna;
for(tuna = 0;tuna<ct;tuna++){

if(tem[tuna] == '#'){

zap = 1; }

else if(tem[tuna] == '_'){ if(zap == 1){

zap = 0; }

else break;

} else{

            first_prod[ap][destiny++] = tem[tuna];
        }
} }

char table[land][sid+1];
ptr = -1;
for(ap = 0; ap < land ; ap++){

for(kay = 0; kay < (sid + 1) ; kay++){

        table[ap][kay] = '!';
    }
```

```
}
for(ap = 0; ap < count ; ap++){

ck = production[ap][0];
xxx = 0;
for(kay = 0; kay <= ptr; kay++)

if(ck == table[kay][0]) xxx = 1;

if (xxx == 1) continue;

else{
ptr = ptr + 1;

        table[ptr][0] = ck;
    }
}
for(ap = 0; ap < count ; ap++){

int tuna = 0; while(first_prod[ap][tuna] != '\0'){

int to,ni=0; for(to=0;to<sid;to++){

if(first_prod[ap][tuna] == ter[to]){

ni = 1; }

}
if(ni == 1){

char xz = production[ap][0]; int cz=0;
while(table[cz][0] != xz){

cz = cz + 1; }

int vz=0;
while(ter[vz] != first_prod[ap][tuna]){

vz = vz + 1; }

table[cz][vz+1] = (char)(ap + 65); }

tuna++; }

} for(k=0;k<sid;k++){

for(kay=0;kay<100;kay++){ if(calc_first[k][kay] == '!'){

break; }
```

```
else if(calc_first[k][kay] == '#'){ int fz = 1;

while(calc_follow[k][fz] != '!'){ char xz = production[k][0]; int cz=0;
while(table[cz][0] != xz){

cz = cz + 1; }

int vz=0;
while(ter[vz] != calc_follow[k][fz]){

vz = vz + 1; }

                table[k][vz+1] = '#';
fz++; }

break; }

} }

for(ap = 0; ap < land ; ap++){ printf("\t\t\t %c\t|\t",table[ap][0]); for(kay = 1;
kay < (sid + 1) ; kay++){

if(table[ap][kay] == '!') printf("\t\t");

else if(table[ap][kay] == '#') printf("%c=#\t\t",table[ap][0]);

else{
int mum = (int)(table[ap][kay]); mum -= 65; printf("%s\t\t",production[mum]);

} }

    printf("\n");
    printf("\t\t\t---------- ");
    printf("\n");
}
int j;
printf("\n\nPlease enter the desired INPUT STRING = "); char input[100];
scanf("%s%c",input,&ch); printf("\n\t\t\t\t\t=====\n");
printf("\t\t\t\t\t\tStack\t\t\tInput\t\t\tAction");  printf("\n\t\t\t\t\t=======\n");
int i_ptr = 0,s_ptr = 1;
char stack[100];
stack[0] = '$';
stack[1] = table[0][0];
while(s_ptr != -1){

printf("\t\t\t\t\t\t");
int vamp = 0; for(vamp=0;vamp<=s_ptr;vamp++){
```

```
        printf("%c",stack[vamp]);
    }
printf("\t\t\t");
vamp = i_ptr; while(input[vamp] != '\0'){

        printf("%c",input[vamp]);
vamp++; }

printf("\t\t\t");
char her = input[i_ptr]; char him = stack[s_ptr]; s_ptr--; if(!isupper(him)){

if(her == him){ i_ptr++;

            printf("POP ACTION\n");
        }
else{
printf("\nString Not Accepted by LL(1) Parser !!\n");

exit(0); }

} else{

for(i=0;i<sid;i++){ if(ter[i] == her)

break; }

char produ[100]; for(j=0;j<land;j++){

if(him == table[j][0]){
if (table[j][i+1] == '#'){

printf("%c=#\n",table[j][0]);
produ[0] = '#';
                        produ[1] = '\0';
                    }
else if(table[j][i+1] != '!'){
int mum = (int)(table[j][i+1]); mum -= 65; strcpy(produ,production[mum]);
printf("%s\n",produ);

} else{

                        printf("\nString Not Accepted by LL(1) Parser !!\n");
exit(0); }

} }

int le = strlen(produ); le = le - 1;
if(le == 0){
```

```
continue; }

for(j=le;j>=2;j--){ s_ptr++;

                stack[s_ptr] = produ[j];
            }
} }

printf("\n\t\t\t=========\n"); if (input[i_ptr] == '\0'){

        printf("\t\t\t\t\t\t\t\tYOUR STRING HAS BEEN ACCEPTED !!\n");
    }
else

        printf("\n\t\t\t\t\t\t\t\tYOUR STRING HAS BEEN REJECTED !!\n");
    printf("\t\t\t============\n");
}

void follow(char c)

{
int i ,j;

if(production[0][0]==c){

         f[m++]='$';
     }
for(i=0;i<10;i++)

{
for(j=2;j<10;j++)

{
if(production[i][j]==c) {

if(production[i][j+1]!='\0'){

                    followfirst(production[i][j+1],i,(j+2));
                }
if(production[i][j+1]=='\0'&&c!=production[i][0]){

                     follow(production[i][0]);
                }
} }

void findfirst(char c ,int q1 , int q2)
```

```
{
int j;

if(!(isupper(c))){

        first[n++]=c;
    }
for(j=0;j<count;j++)

{
if(production[j][0]==c)

{
if(production[j][2]=='#'){

if(production[q1][q2] == '\0') first[n++]='#';

else if(production[q1][q2] != '\0' && (q1 != 0 || q2 != 0))

                {
                    findfirst(production[q1][q2], q1, (q2+1));
}

else

}
else if(!isupper(production[j][2])){

                first[n++]=production[j][2];
            }
else {

                findfirst(production[j][2], j, 3);
            }
} }

}
void followfirst(char c, int c1 , int c2)

{
int k;

if(!(isupper(c))) f[m++]=c;

else{
int i=0,j=1;

for(i=0;i<count;i++)
```

```
{
if(calc_first[i][0] == c)

break; }

while(calc_first[i][j] != '!')

{
if(calc_first[i][j] != '#'){

            f[m++] = calc_first[i][j];
        }
else{
if(production[c1][c2] == '\0'){

            follow(production[c1][0]);
        }
else{

            followfirst(production[c1][c2],c1,c2+1);
        }
}

j++; }

} }
```

2) Implementation of SLR Parser:

Code:

```
#include<stdio.h>
#include<string.h>
int i,j,k,m,n=0,o,p,ns=0,tn=0,rr=0,ch=0;
char read[15][10],gl[15],gr[15][10],temp,templ[15],tempr[15]
[10],*ptr,temp2[5],dfa[15][15];

struct states
```

```
{
char lhs[15],rhs[15][10]; int n;

}I[15];

int compstruct(struct states s1,struct states s2)

{
int t;

if(s1.n!=s2.n) return 0;

if( strcmp(s1.lhs,s2.lhs)!=0 ) return 0;

for(t=0;t<s1.n;t++)
if( strcmp(s1.rhs[t],s2.rhs[t])!=0 )

return 0; return 1;

}

void moreprod()

{
int r,s,t,l1=0,rr1=0; char *ptr1,read1[15][10];

for(r=0;r<I[ns].n;r++)

{
ptr1=strchr(I[ns].rhs[l1],'.'); t=ptr1-I[ns].rhs[l1];
if( t+1==strlen(I[ns].rhs[l1]) ) {

l1++;

continue; }

temp=I[ns].rhs[l1][t+1]; l1++; for(s=0;s<rr1;s++)

if( temp==read1[s][0] ) break;

if(s==rr1)

        {
           read1[rr1][0]=temp;
rr1++; }

else continue;

for(s=0;s<n;s++)
```

```
{
if(gl[s]==temp)

{
I[ns].rhs[I[ns].n][0]='.'; I[ns].rhs[I[ns].n][1]=NULL;
strcat(I[ns].rhs[I[ns].n],gr[s]); I[ns].lhs[I[ns].n]=gl[s];
I[ns].lhs[I[ns].n+1]=NULL; I[ns].n++;

} }

} }

void canonical(int l)

{
int t1;

char read1[15][10],rr1=0,*ptr1; for(i=0;i<I[l].n;i++)

   {
      temp2[0]='.';
ptr1=strchr(I[l].rhs[i],'.'); t1=ptr1-I[l].rhs[i];
if( t1+1==strlen(I[l].rhs[i]) )

continue; temp2[1]=I[l].rhs[i][t1+1];

temp2[2]=NULL;

for(j=0;j<rr1;j++)
if( strcmp(temp2,read1[j])==0 )

break; if(j==rr1)

{
strcpy(read1[rr1],temp2); read1[rr1][2]=NULL; rr1++;

}

else continue;

for(j=0;j<I[0].n;j++)

{
ptr=strstr(I[l].rhs[j],temp2); if( ptr )
{

templ[tn]=I[l].lhs[j]; templ[tn+1]=NULL; strcpy(tempr[tn],I[l].rhs[j]); tn++;
```

```
} }

for(j=0;j<tn;j++)

{
ptr=strchr(tempr[j],'.'); p=ptr-tempr[j]; tempr[j][p]=tempr[j][p+1];
tempr[j][p+1]='.'; I[ns].lhs[I[ns].n]=templ[j]; I[ns].lhs[I[ns].n+1]=NULL;
strcpy(I[ns].rhs[I[ns].n],tempr[j]); I[ns].n++;

}

moreprod(); for(j=0;j<ns;j++)

{



//if ( memcmp(&I[ns],&I[j],sizeof(struct states))==1 )
if( compstruct(I[ns],I[j])==1 )

{
I[ns].lhs[0]=NULL; for(k=0;k<I[ns].n;k++)

I[ns].rhs[k][0]=NULL; I[ns].n=0;

dfa[l][j]=temp2[1]; break;

} }

if(j<ns)

{
tn=0;

for(j=0;j<15;j++)

{
templ[j]=NULL;

tempr[j][0]=NULL; }

continue; }

dfa[l][j]=temp2[1]; printf("\n\nI%d :",ns); for(j=0;j<I[ns].n;j++)

            printf("\n\t%c -> %s",I[ns].lhs[j],I[ns].rhs[j]);
        getch();
ns++;
tn=0; for(j=0;j<15;j++) {
```

```
templ[j]=NULL;

tempr[j][0]=NULL; }

} }

void main()

{
FILE *f;

int l; clrscr();

for(i=0;i<15;i++)

{
l[i].n=0;

l[i].lhs[0]=NULL; l[i].rhs[0][0]=NULL; dfa[i][0]=NULL;

}

f=fopen("tab6.txt","r"); while(!feof(f))

   {
      fscanf(f,"%c",&gl[n]);
      fscanf(f,"%s\n",gr[n]);
      n++;
}

printf("THE GRAMMAR IS AS FOLLOWS\n"); for(i=0;i<n;i++)

      printf("\t\t\t\t%c -> %s\n",gl[i],gr[i]);
   l[0].lhs[0]='Z';
   strcpy(l[0].rhs[0],".S");
   l[0].n++;
   l=0;
for(i=0;i<n;i++)

{
temp=l[0].rhs[l][1]; l++; for(j=0;j<rr;j++)

if( temp==read[j][0] ) break;

if(j==rr)

{
          read[rr][0]=temp;
```

```
rr++; }

else continue;

for(j=0;j<n;j++)

{
if(gl[j]==temp)

          {
              I[0].rhs[I[0].n][0]='.';
              strcat(I[0].rhs[I[0].n],gr[j]);
              I[0].lhs[I[0].n]=gl[j];
              I[0].n++;
} }

} ns++;

printf("\nI%d :\n",ns-1); for(i=0;i<I[0].n;i++)

printf("\t%c -> %s\n",I[0].lhs[i],I[0].rhs[i]); for(l=0;l<ns;l++)

        canonical(l);
    printf("\n\n\t\tPRESS ANY KEY FOR DFA TABLE" );
    getch();
    clrscr();
printf("\t\t\tDFA TABLE IS AS FOLLOWS\n\n\n"); for(i=0;i<ns;i++)

{
printf("I%d : ",i); for(j=0;j<ns;j++)

if(dfa[i][j]!='\0')
printf("'%c'->I%d | ",dfa[i][j],j);

        printf("\n\n\n");
    }
    printf("\n\n\n\t\tPRESS ANY KEY TO EXIT");
getch(); }
```

Task5:

1. Write a program for generating a parser program using lex and yacc for a language with integer identifiers, binary arithmetic expressions and assignments. (Input is grammar and output is parser in C language).

Code:

Lex Program:

```
%{

#include<stdlib.h> #include "y.tab.c" extern int yylval; %}

%%

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

'<=' return LE; '>=' return GE; '!=' return NE; '==' return EQ; [\t] ;

[\n] return 0;
. return yytext[0];

%%
```

Yacc Program:

```
%{

#include<stdio.h> int flag=0;
int yylex();
int yyerror();

%}

%token NAME NUMBER
%left GE LE EQ NE EE '<' '>' %left '+' '-'
%left '*' '/' '%'
%left '(' ')'
%nonassoc UMINUS

%%

ArithmeticExpression: E {
        printf("Result=%d\n", $$);
return 0; };

E :E'+'E{$$=$1+$3;} | E '-' E {$$ = $1 - $3;} | E '*' E {$$ = $1 * $3;} | E '/' E
{$$ = $1 / $3;} | E '%' E {$$ = $1 % $3;} | '(' E ')' {$$ = $2;}

    | NUMBER {$$ = $1;}
    | E GE E {$$ = $1 >= $3 ;}
```

```
| E LE E {$$ = $1 <= $3 ;}
    |E NE E {$$ = $1 != $3 ;}
    | E EE E {$$ = $1 == $3 ;}
    | UMINUS E {$$ = -$1 ;}
;

%%
int main()

{
yyparse();

}

int yyerror() {return 0;} int yywrap(){

return 1; }
```

2. Write a program for generating derivation sequence for a given terminal string using parsing table.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
char s[20],stack[20];

int main()
{
char m[5][6][3]={"tb"," "," "," ","tb"," "," "," "," ","+tb"," "," "," ","n","n","fc"," "," "," ","fc"," "," "," "," ","n","*fc"," a "," ","n","n","i"," "," "," ","(e)"," "," "," "};
int size[5][6]={2,0,0,2,0,0,0,3,0,0,1,1,2,0,0,2,0,0,0,1,3,0,1,1,1,0,0,3,0,0}; int
i,j,k,n,str1,str2;
printf("\n Enter the string: ");
scanf("%s",s);
strcat(s,"$");
n=strlen(s);
stack[0]='$';
stack[1]='e';
i=1;
```

```
j=0;
printf(“\nStack    Input\n”);
printf("_____\n");
while((stack[i]!='$')&&(s[j]!='$'))
{
if(stack[i]==s[j])
{
i--;
j++;
}
switch(stack[i])
{

case 'e': str1=0; break;
case 'b': str1=1; break;

case 't': str1=2; break;
case 'c': str1=3; break;

case 'f': str1=4; break;
}
switch(s[j])

{
case 'i': str2=0;

break;
case '+': str2=1;
break;
case '*': str2=2;
break;
case '(': str2=3;
break;
case ')': str2=4;
break;
case '$': str2=5;
break;
}
if(m[str1][str2][0]=='\0')
{
printf("\nERROR");
return 0;
}
else if(m[str1][str2][0]=='n') i--;
else if(m[str1][str2][0]=='i') stack[i]='i';
```

```
else
{ for(k=size[str1][str2]-1;k>=0;k--) {
stack[i]=m[str1][str2][k];
i++;
}
i--;
}
for(k=0;k<=i;k++)
printf(" %c",stack[k]);
printf(" ");

for(k=j;k<=n;k++) printf("%c",s[k]); printf(" \n ");
}

printf("\n ACCEPTED STRING");
return 0; }
```