11. CONVERSION OF NFA TO DFA

AIM:

To convert NFA to DFA

PROGRAM

```
#include <stdio.h>
#include <string.h>
#define STATES 256
#define SYMBOLS 20
int N_symbols; int NFA_states;
char *NFAtab[STATES][SYMBOLS];
int DFA_states; /* number of DFA states */
int DFAtab[STATES][SYMBOLS];
/*Print state-transition table.*/ void put_dfa_table(
int tab[][SYMBOLS], /* DFA table */ int nstates,    /* number of states */
int nsymbols) /* number of input symbols */
{
int i, j;
puts("STATE TRANSITION TABLE");
/* input symbols: '0', '1', ... */ printf("   | ");
for (i = 0; i < nsymbols; i++) printf(" %c ", '0'+i);
printf("\n   +--");
for (i = 0; i < nsymbols; i++) printf("    ");
printf("\n");
for (i = 0; i < nstates; i++) {
printf(" %c | ", 'A'+i); /* state */ for (j = 0; j < nsymbols; j++)
printf(" %c ", 'A'+tab[i][j]); printf("\n");
}
}
/*Initialize NFA table.*/ void init_NFA_table()
{

/*
NFA table for ex.17 at p.72
*/
NFAtab[0][0] = "12";
NFAtab[0][1] = "13";
NFAtab[1][0] = "12";
NFAtab[1][1] = "13";
NFAtab[2][0] = "4";
NFAtab[2][1] = "";
NFAtab[3][0] = "";
NFAtab[3][1] = "4";
NFAtab[4][0] = "4";
NFAtab[4][1] = "4";
NFA_states = 5;
DFA_states = 0;
N_symbols = 2;
}
/*String 't' is merged into 's' in an alphabetical order.*/ void string_merge(char *s, char *t)
{
char temp[STATES], *r=temp, *p=s;
while (*p && *t) { if (*p == *t) {
*r++ = *p++; t++;
} else if (*p < *t) {
*r++ = *p++;
} else
*r++ = *t++;
}
*r = '\0';
if (*p) strcat(r, p); else if (*t) strcat(r, t);
strcpy(s, temp);
}
/*Get next-state string for current-state string.*/
void get_next_state(char *nextstates, char *cur_states, char *nfa[STATES][SYMBOLS], int n_nfa, int
symbol)
{
int i;
char temp[STATES]; temp[0] = '\0';
for (i = 0; i < strlen(cur_states); i++) string_merge(temp, nfa[cur_states[i]-'0'][symbol]);
strcpy(nextstates, temp);
}
int state_index(char *state, char statename[][STATES], int *pn)
{
int i;
if (!*state) return -1; /* no next state */
for (i = 0; i < *pn; i++)
if (!strcmp(state, statename[i])) return i;
strcpy(statename[i], state);    /* new state-name */ return (*pn)++;
}
/* Convert NFA table to DFA table. Return value: number of DFA states.
*/
int nfa_to_dfa(char *nfa[STATES][SYMBOLS], int n_nfa, int n_sym, int dfa[][SYMBOLS])
{
char statename[STATES][STATES]; int i = 0; /* current index of DFA */ int n = 1; /* number of DFA states
*/ char nextstate[STATES];
int j;
strcpy(statename[0], "0"); /* start state */
for (i = 0; i < n; i++) {    /* for each DFA state */
for (j = 0; j < n_sym; j++) {    /* for each input symbol */ get_next_state(nextstate, statename[i], nfa, n_nfa,
j); dfa[i][j] = state_index(nextstate, statename, &n);
}}
return n; /* number of DFA states */
}
void main()
{
init_NFA_table();
DFA_states = nfa_to_dfa(NFAtab, NFA_states, N_symbols, DFAtab); put_dfa_table(DFAtab, DFA_states,
N_symbols);
}
```

Input & Output:

```
STATE TRANSITION TABLE
  |          0          1
 +--------------------------
A |          B          C
B |          D          C
C |          B          E
D |          D          E
E |          D          E
```

12. RECURSIVE DESCENT PARSER

AIM

Program to Construct a recursive descent parser for an expression.

PROGRAM

```
#include<stdio.h>
#include<string.h>
char input[100];
int i=0,error=0;
void E();
void T();
void Eprime();
void Tprime();
void F();
void main()
{
printf("Enter an arithmetic expression :\n");
gets(input);
E();
if(strlen(input)==i&&error==0)
printf("\nAccepted..!!!");
else
printf("\nRejected..!!!");
}
void E()
{
T();
Eprime();
}
void Eprime()
{
if(input[i]=='+')
{
i++;
T();
Eprime();
}
}
void T()
{
F();
Tprime();
}
void Tprime()
{
if(input[i] == '*' || input[i] == '/')
{
i++;
F();
Tprime();
}
}
```

```
void F()

{
if(input[i] == '(')
{
i++;
E();
if(input[i] == ')')
i++;
else
error = 1; // unmatched '('
}
else if(isdigit(input[i])) // numeric constant
{
i++;
while(isdigit(input[i]))
i++;
}
else if(isalpha(input[i])) // identifier
{
i++;
while(isalnum(input[i]) || input[i] == '_')
i++;
}
else
{
error = 1;
}
}
```

OUTPUT

```
1)
Enter an arithmetic expression :
sum+month*interest
Accepted..!!!
2)
Enter an arithmetic expression :
sum+avg*+interest
Rejected..!!!
3)
Enter an arithmetic expression :
sum/total*100
Accepted..!!!
```

13A. FIRST OF A GIVEN GRAMMER

AIM

Program to simulate the First of a given grammar

PROGRAM

```
#include<stdio.h>
#include<ctype.h>
void FIRST(char[],char );
void result(char[],char);
int nop;
char prod[10][10];
void main()
{
int i;
char choice;
char c;
char res1[20];
clrscr();
printf("How many number of productions ? :");
scanf(" %d",&nop);
printf("enter the production string like E=E+T\n");
for(i=0;i<nop;i++)
{
printf("Enter productions Number %d : ",i+1);
scanf(" %s",prod[i]);
}
do
{
printf("\n Find the FIRST of :");
scanf(" %c",&c);
FIRST(res1,c);
printf("\n FIRST(%c)= { ",c);
for(i=0;res1[i]!='\0';i++)
printf(" %c ",res1[i]);
printf("}\n");
printf("press 'y' to continue : ");
scanf(" %c",&choice);
}
while(choice=='y'||choice =='Y');
}
void FIRST(char res[],char c){
int i,j,k;
char subres[5];
int eps;
subres[0]='\0';
res[0]='\0';
if(!(isupper(c)))
{
result(res,c);
return ;
}
for(i=0;i<nop;i++){
if(prod[i][0]==c){
```

```
if(prod[i][2]=='$')
result(res,'$');
else{
j=2;
while(prod[i][j]!='\0'){
eps=0;
FIRST(subres,prod[i][j]);
for(k=0;subres[k]!='\0';k++)
result(res,subres[k]);
for(k=0;subres[k]!='\0';k++)
if(subres[k]=='$'){
eps=1;
break;
}
if(!eps)
break;
j++;
}}}}
return ;}
void result(char res[],char val){
int k;
for(k=0 ;res[k]!='\0';k++)
if(res[k]==val)
return;
res[k]=val;
res[k+1]='\0';
}
```

OUTPUT

```
How many number of productions ? :8
enter the production string like E=E+T
Enter productions Number 1 : E=TX
Enter productions Number 2 : X=+TX
Enter productions Number 3 : X=$
Enter productions Number 4 : T=FY
Enter productions Number 5 : Y=*FY
Enter productions Number 6 : Y=$
Enter productions Number 7 : F=(E)
Enter productions Number 8 : F=i
Find the FIRST of :X
FIRST(X)= { + $ }
press 'y' to continue : Y
Find the FIRST of :F
FIRST(F)= { ( i }
press 'y' to continue : Y
Find the FIRST of :Y
FIRST(Y)= { * $ }
press 'y' to continue : Y
Find the FIRST of :E
FIRST(E)= { ( i }
press 'y' to continue : Y
Find the FIRST of :T
FIRST(T)= { ( i }
press 'y' to continue : N
```

13B. FIND THE FOLLOW OF A GIVEN GRAMMER

AIM

Program to simulate Follow of a given grammar

PROGRAM

```
#include<stdio.h>
#include<string.h>
int nop,m=0,p,i=0,j=0;
char prod[10][10],res[10];
void FOLLOW(char c);
void first(char c);
void result(char);
void main()
{
int i;
int choice;
char c,ch;
printf("Enter the no.of productions: ");
scanf("%d", &nop);
printf("enter the production string like E=E+T\n");
for(i=0;i<nop;i++)
{
printf("Enter productions Number %d : ",i+1);
scanf(" %s",prod[i]);
}
do
{
m=0;
printf("Find FOLLOW of -->");
scanf(" %c",&c);
FOLLOW(c);
printf("FOLLOW(%c) = { ",c);
for(i=0;i<m;i++)
printf("%c ",res[i]);
printf(" }\n");
printf("Do you want to continue(Press 1 to continue....)?");
scanf("%d%c",&choice,&ch);
}
while(choice==1);
}
void FOLLOW(char c)
{
if(prod[0][0]==c)
result('$');
for(i=0;i<nop;i++)
{
for(j=2;j<strlen(prod[i]);j++)
{
if(prod[i][j]==c)
{
if(prod[i][j+1]!='\0')
first(prod[i][j+1]);
if(prod[i][j+1]=='\0'&&c!=prod[i][0])
FOLLOW(prod[i][0]);
}}}}
void first(char c)
{
int k;
if(!(isupper(c)))
result(c);
for(k=0;k<nop;k++)
{
if(prod[k][0]==c)
{
if(prod[k][2]=='$')
FOLLOW(prod[i][0]);
else if(islower(prod[k][2]))
result(prod[k][2]);
else
first(prod[k][2]);
}}}
void result(char c){
int i;
for( i=0;i<=m;i++)
if(res[i]==c)
return;
res[m++]=c;}
```

OUTPUT

```
Enter the no.of productions: 8
enter the production string like E=E+T
Enter productions Number 1 : E=TX
Enter productions Number 2 : X=+TX
Enter productions Number 3 : X=$
Enter productions Number 4 : T=FY
Enter productions Number 5 : Y=*FY
Enter productions Number 6 : Y=$
Enter productions Number 7 : F=(E)
Enter productions Number 8 : F=i
Find FOLLOW of -->X
FOLLOW(X) = { $ ) }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->E
FOLLOW(E) = {$ ) }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->Y
FOLLOW(Y) = { + $ ) }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->T
FOLLOW(T) = { +$ ) }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->F
FOLLOW(F) = { * + $ ) }
Do you want to continue(Press 1 to continue....)?2
```

14. SHIFT REDUCE PARSER

AIM

Program to Construct a Shift Reduce Parser for an expression

PROGRAM

```
#include"stdio.h"
#include"string.h"
#include<stdlib.h>
char ip_sym[15],stack[15];
int ip_ptr=0,st_ptr=0,len,i;
char temp[2],temp2[2];
char act[15];
void check();
void main()
{
printf("\n\t\t SHIFT REDUCE PARSER\n");
printf("\n GRAMMER\n");
printf("\n E->E+E\n E->E/E");
printf("\n E->E*E\n E->E-E\n E->id");
printf("\n enter the input symbol:\t");
gets(ip_sym);
printf("\n\t stack implementation table");
printf("\n stack\t\t input symbol\t\t action");
printf("\n______\t\t ____________\t\t ______\n");
printf("\n $\t\t%s$\t\t\t--",ip_sym);
strcpy(act,"shift ");
temp[0]=ip_sym[ip_ptr];
temp[1]='\0';
strcat(act,temp);
len=strlen(ip_sym);
for(i=0;i<=len-1;i++)
{
stack[st_ptr]=ip_sym[ip_ptr];
stack[st_ptr+1]='\0';
ip_sym[ip_ptr]='';
ip_ptr++;
printf("\n $%s\t\t%s$\t\t\t%s",stack,ip_sym,act);
strcpy(act,"shift ");
temp[0]=ip_sym[ip_ptr];
temp[1]='\0';
strcat(act,temp);
check();
st_ptr++;
}
st_ptr++;
check();
}
void check()
{
int flag=0;
temp2[0]=stack[st_ptr];
temp2[1]='\0';
if(islower(temp2[0]))
{
stack[st_ptr]='E';
flag=1;
}
if((!strcmp(temp2,"+"))||(!strcmp(temp2,"*"))
||(!strcmp(temp2,"/"))||(!strcmp(temp2,"-")))
{
```

```
flag=1;
}
if((!strcmp(stack,"E+E"))||(!strcmp(stack,"E/E"))
||(!strcmp(stack,"E*E"))||(!strcmp(stack,"E-E")))
{
if(!strcmp(stack,"E+E"))
{
strcpy(stack,"E");
printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip_sym);
}
else
if(!strcmp(stack,"E/E"))
{
strcpy(stack,"E");
printf("\n $%s\t\t %s$\t\t\tE->E/E",stack,ip_sym);
}
else
if(!strcmp(stack,"E-E"))
{
strcpy(stack,"E");
printf("\n $%s\t\t %s$\t\t\tE->E-E",stack,ip_sym);
}
else
{
strcpy(stack,"E");
printf("\n $%s\t\t%s$\t\t\tE->E*E",stack,ip_sym);
}
flag=1;
st_ptr=0;
}
if(!strcmp(stack,"E")&&ip_ptr==len)
{
printf("\n $%s\t\t%s$\t\t\tACCEPT",stack,ip_sym);
exit(0);
}
if(flag==0)
{
printf("\n $%s\t\t\t%s\t\t reject",stack,ip_sym);
exit(0);
}
return;
}
```

OUTPUT:

```
1)
SHIFT REDUCE PARSER GRAMMER
E->E+E
E->E/E
E->E*E
E->E-E
E->id
enter the input symbol: a+b*c
stack implementation table
stack           Input symbol           action
______           ____________          ______
$               a+b*c$                 --
$a              +b*c$                  shift a
$E              +b*c$                   E->a
$E+              b*c$                  shift +
$E+b            *c$                    shift b
$E+E            *c$                    E->b
$E              *c$                    E->E+E
$E*             c$                     shift *
$E*c            $                      shift c
$E*E            $                      E->c
$E              $                      E->E*E
$E              $                      ACCEPT
2)
SHIFT REDUCE PARSER
GRAMMER
E->E+E
E->E/E
E->E*E
E->E-E
E->id
enter the input symbol: a+b*+c
stack implementation table


stack           Input symbol           action
______           ____________          ______
$                a+b*+c$               --
$a              +b*+c$                 shift a
$E               +b*+c$                E->a
$E+             b*+c$                  shift +
$E+b            *+c$                   shift b
$E+E            *+c$                   E->b
$E              *+c$                   E->E+E
$E*             +c$                    shift *
$E*+            c$                     shift +
$E*+c           $                       shift c
$E*+E           $                      E->c
$E*+E                                  reject
```

15. INTERMEDIATE CODE GENERATION

AIM:-

To implement a program to generate intermediate code

PROGRAM

```
#include"stdio.h"
#include"string.h"
#include<stdlib.h>
int i=1,j=0,no=0,tmpch=90; char str[100],left[15],right[15];
void findopr();
void explore();
void fleft(int);
void fright(int); struct exp
{
int pos; char op;
}k[15];
void main()
{
printf("\t\tINTERMEDIATE CODE GENERATION\n\n"); printf("Enter the Expression :");
scanf("%s",str);
printf("The intermediate code:\t\tExpression\n"); findopr();
explore();
}
void findopr()
{
for(i=0;str[i]!='\0';i++)
if(str[i]==':')
{
k[j].pos=i; k[j++].op=':';
}
for(i=0;str[i]!='\0';i++)
if(str[i]=='/')
{
k[j].pos=i; k[j++].op='/';
}
for(i=0;str[i]!='\0';i++) if(str[i]=='*')
{
k[j].pos=i; k[j++].op='*';
}
for(i=0;str[i]!='\0';i++) if(str[i]=='+')
{
k[j].pos=i; k[j++].op='+';
}
for(i=0;str[i]!='\0';i++)
if(str[i]=='-')
{
k[j].pos=i; k[j++].op='-';
}
}
void explore()
{
i=1;
while(k[i].op!='\0')
{
fleft(k[i].pos);
fright(k[i].pos);
str[k[i].pos]=tmpch--;
printf("\t%c := %s%c%s\t\t",str[k[i].pos],left,k[i].op,right);
for(j=0;j <strlen(str);j++)
if(str[j]!='$')
printf("%c",str[j]);
printf("\n");
i++;
}
fright(-1); if(no==0)
{
fleft(strlen(str));
printf("\t%s := %s",right,left);
exit(0);
}
printf("\t%s := %c",right,str[k[--i].pos]);
}
void fleft(int x)
{
int w=0,flag=0; x--;
while(x!=-1&&str[x]!='+'&&str[x]!='*'&&str[x]!='='&&str[x]!='\0'&&str[x]!='-'&&str[x]!='/'&&str[x]!='
)
{
if(str[x]!='$'&& flag==0)
{
left[w++]=str[x]; left[w]='\0';
str[x]='$'; flag=1;
}
x--;
}
}
void fright(int x)
{

int w=0,flag=0; x++;
while(x!=-1&&str[x]!=
'+'&&str[x]!='*'&&str[x]!='\0'&&str[x]!='='&&str[x]!=':'&&str[x]!='-'&&str[x]!='/')

{
if(str[x]!='$'&& flag==0)
{
right[w++]=str[x]; right[w]='\0';
str[x]='$'; flag=1;
} x++;
}
}
```

OUTPUT

```
INTERMEDIATE CODE GENERATION

Enter the Expression :a:=b+c*d/e
The intermediate code:              Expression
        Z := d/e           a:=b+c*Z
        Y := c*Z                    a:=b+Y
        X := b+Y                    a:=X
        a := X
```

16. **IMPLEMENT THE BACK END OF THE COMPILER**

AIM:

To implement the back end of the compiler which takes the three address code and produces the 8086 assembly language instructions that can be assembled and run using a 8086 assembler. The target assembly instructions can be simple move, add, sub, jump.

PROGRAM

```
#include<stdio.h>
#include<stdio.h>
#include<string.h>
void main(){
char icode[10][30],str[20],opr[10];
int i=0;
printf("\n Enter the set of intermediate code (terminated by exit () \n");
do{
scanf("%s",icode[i]);
} while(strcmp(icode[i++],"exit")!=0);
printf("\n target code generation");
printf("\n************************");
i=0;
do{
strcpy(str,icode[i]);
switch(str[3])
{
case '+':
strcpy(opr,"ADD");
break;
case '-':
strcpy(opr,"SUB");
break;
case '*':
strcpy(opr,"MUL");
break;
case '/':
strcpy(opr,"DIV");
break;
}
printf("\n\tMov %c,R%d",str[2],i);
printf("\n\t%s %c,R%d",opr,str[4],i);
printf("\n\tMov R%d,%c",i,str[0]);
}while(strcmp(icode[++i],"exit")!=0);
}
```

OUTPUT:

```
Enter the set of intermediate code (terminated by exit ()
d=2/3
c=4/5
a=2*e
exit
target code generation
************************
          Mov 2,R0
          DIV 3,R0
Mov R0,d
          Mov 4,R1
          DIV 5,R1
          Mov R1,c
          Mov 2,R2
          MUL e,R2
          Mov R2,a
```

17. CONTENT BEYOND SYLLABUS

AIM: SLR Parser implementation in C

PROGRAM

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX 100
// Sample ACTION Table (hardcoded for simplicity)
char action[12][6][10] = {
/*      id     +     *     (     )     $ */
/*0*/ { "s5",  "",   "",  "s4",  "",   ""  },
/*1*/ {  "",  "s6",  "",   "",   "",  "acc"},
/*2*/ {  "",  "r2", "s7",  "",  "r2", "r2"},
/*3*/ {  "",  "r4", "r4",  "",  "r4", "r4"},
/*4*/ { "s5",  "",   "",  "s4",  "",   "" },
/*5*/ {  "",  "r6", "r6",  "",  "r6", "r6"},
/*6*/ { "s5",  "",   "",  "s4",  "",   "" },
/*7*/ { "s5",  "",   "",  "s4",  "",   "" },
/*8*/ {  "",  "s6",  "",   "",  "s11",""  },
/*9*/ {  "",  "r1", "s7",  "",  "r1", "r1"},
/*10*/{  "",  "r3", "r3",  "",  "r3", "r3"},
/*11*/{  "",  "r5", "r5",  "",  "r5", "r5"}
};
// Sample GOTO Table
int goTo[12][3] = {
/*      E   T   F */
/*0*/ {1,  2,  3},
/*1*/ {-1, -1, -1},
/*2*/ {-1, -1, -1},
/*3*/ {-1, -1, -1},
/*4*/ {8,  2,  3},
/*5*/ {-1, -1, -1},
/*6*/ {-1, 9,  3},
/*7*/ {-1, -1, 10},
/*8*/ {-1, -1, -1},
/*9*/ {-1, -1, -1},
/*10*/{-1, -1, -1},
/*11*/{-1, -1, -1}};
// Production rules
char *productions[] = {
    "",        // dummy
    "E->E+T",
    "E->T",
    "T->T*F",
    "T->F",
    "F->(E)",
    "F->id"};
// Number of RHS symbols in each production (used for popping)
int prod_len[] = {0, 3, 1, 3, 1, 3, 1};
int getSymbolIndex(char symbol[]) {
    if (strcmp(symbol, "id") == 0) return 0;
    if (strcmp(symbol, "+") == 0) return 1;
    if (strcmp(symbol, "*") == 0) return 2;
    if (strcmp(symbol, "(") == 0) return 3;
    if (strcmp(symbol, ")") == 0) return 4;
    if (strcmp(symbol, "$") == 0) return 5;
    return -1;}
int getGotoIndex(char nonterm) {
    if (nonterm == 'E') return 0;
    if (nonterm == 'T') return 1;
    if (nonterm == 'F') return 2;
    return -1;}
int main() {
    char input[MAX][10], stack[MAX][10];
    int stateStack[MAX];
    int top = 0;
    int stateTop = 0;
    int i = 0;
    // Example input: id + id * id
    // Convert to: id + id * id $
    printf("Enter input string (space separated tokens): ");
    char token[10];
    while (scanf("%s", token) == 1) {
   if (strcmp(token, "end") == 0) break;
  strcpy(input[i++], token); }
    strcpy(input[i], "$");
    // Initialize stacks
    strcpy(stack[top], "$");
    stateStack[stateTop] = 0;
    int ip = 0;
    while (1) {
        int currentState = stateStack[stateTop];
        int symbolIndex = getSymbolIndex(input[ip]);
        char *act = action[currentState][symbolIndex];
        if (strlen(act) == 0) {
            printf("\nError: Invalid string.\n");
            break; }
        if (act[0] == 's') {
            // Shift action
            int nextState = atoi(&act[1]);
            top++;
            strcpy(stack[top], input[ip]);
            stateTop++;
            stateStack[stateTop] = nextState;
            ip++;
        } else if (act[0] == 'r') {
            // Reduce action
            int prodNum = atoi(&act[1]);
            int len = prod_len[prodNum];
            // Pop stack symbols
            top -= len;
            stateTop -= len;
            // Push LHS
            char lhs[2] = {productions[prodNum][0], '\0'};
            top++;
            strcpy(stack[top], lhs);
            // Push goto
            int gotoState = goTo[stateStack[stateTop]][getGotoIndex(lhs[0])];
            if (gotoState == -1) {
                printf("\nError: Invalid GOTO.\n");
                break; }
            stateTop++;
            stateStack[stateTop] = gotoState;
            printf("Reduced using rule: %s\n", productions[prodNum]);
        } else if (strcmp(act, "acc") == 0) {
            printf("\nString accepted successfully.\n");
            break;} }
    return 0;}
```

OUTPUT:

```
student@cse-hwl-030:~$ cc slr.c
student@cse-hwl-030:~$ ./a.out
Enter input string (space separated tokens): id + id * id end
Reduced using rule: F->id
Reduced using rule: T->F
Reduced using rule: E->T
Reduced using rule: F->id
Reduced using rule: T->F
Reduced using rule: F->id
Reduced using rule: T->T*F
Reduced using rule: E->E+T

String accepted successfully.
```