Central University Of Rajasthan

Ajmer, Rajasthan

COMPILER DESIGN LAB

SUBMITTED BY

Ritvik Shukla

2020BTCSE023

Ques-1

```c
#include <stdio.h>
int fa[10][10][10], states[2][10], curr, row = 0, col = 0, sr = 0, sc = 0, th = 0, in;
char *str;
int nfa(char *string, int state)
{
    int i, j;
    for (i = 0; i <= row; i++)
    {
        if (*string)
        {
            curr = fa[state][*string - 97][i];
            if (curr == -1)
                break;
            if (nfa(string + 1, curr))
                return 1;
        }
        else
        {
            if (states[1][i] == -1)
                break;
            if (state == states[1][i])
                return 1;
        }
    }
    return 0;
}
int main()
{
    FILE *fp;
    int i, j, k, flag = 0;
    char c, ch;

    fp = fopen("Nfa_ip.txt", "r");
    for (i = 0; i < 2; i++)
        for (j = 0; j < 10; j++)
            states[i][j] = -1;
    for (i = 0; i < 10; i++)
        for (j = 0; j < 10; j++)
            for (k = 0; k < 10; k++)
                fa[i][j][k] = -1;
    while (fscanf(fp, "%d", &in) != EOF)
    {
        fscanf(fp, "%c", &c);
        if (flag)
        {
            states[sr][sc++] = in;
            if (c == '\n')
            {
                sr++;
                sc = 0;
            }
        }
        else if (c == '#')
        {
            flag = 1;
            fa[row][col][th] = in;
            printf("\nfa[%d][%d][%d]=%d", row, col, th, fa[row][col][th]);
        }
        else if (!flag)
        {
            fa[row][col][th] = in;
            printf("\nfa[%d][%d][%d]=%d", row, col, th, fa[row][col][th]);
            if (c == ',')
            {
                th++;
            }
            else if (c == '\n')
            {
                col = 0;
                row++;
                th = 0;
            }
            else if (c != ',')
            {
                col++;
                th = 0;
            }
        }
    }
    printf("\n\nEnter the string : \n");
    scanf("%s", str);
    if (nfa(&str, states[0][0]))
        printf("\nString Is Accepted");
    else
        printf("\nString Not Accepted");
    getch();
    return 0;
}
```

Ques-2

```c
#include <stdio.h>

int Fa[10][10][10], states[2][10], row = 0, col = 0, sr = 0, sc = 0, th = 0,
                                   in, stat, new_state[10][10], max_inp = -1, no_stat;
FILE *fp;

int search(int search_var)
{

    int i;
    for (i = 0; i < no_stat; i++)
        if (search_var == states[1][i])
            return 1;
    return 0;
}

int sort(int *arr, int count)
{
    int temp, i, j;
    for (i = 0; i < count - 1; i++)
    {
        for (j = i + 1; j < count; j++)
        {
            if (arr[i] >= arr[j])
            {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
    return 0;
}

int checkcon(int *arr, int *count)
{
    int i, temp, j, k, c, t, m;
    for (i = 0; i < *count; i++)
    {
        if (arr[i] > row)
        {
            temp = arr[i];
            c = 0;
            t = 0;
            while (new_state[arr[i]][t] != -1)
            {
                t++;
                c++;
            }
            for (k = 0; k <= c - 2; k++)
            {
                for (j = 9; j >= i + 1 + k; j--)
                {
                    arr[j] = arr[j - 1];
                }
            }

            t = 0;
            for (j = i; j < c; j++)
            {
                arr[j] = new_state[temp][t];
                t++;
            }
        }
    }
    c = 0;
    for (i = 0; arr[i] != -1; i++)
        c++;
    *count = c;
    return 0;
}

int remove_duplicate(int *arr, int *count)
{
    int i, j = 0;
    for (i = 1; i < *count; i++)
    {
        if (arr[i] != arr[j])
```

```
        {
            j++;
            arr[j] = arr[i];
        }
    }
    *count = j + 1;
    return 0;
}

int check(int i, int j, int c, int *name)
{
    int t, l, f;
    for (l = 0; l <= stat; l++)
    {
        t = 0;
        f = 0;
        while (Fa[i][j][t] != -1)
        {
            if (Fa[i][j][t] == new_state[l][t])
                t++;
            else
            {
                f = 1;
                break;
            }
        }
        if ((t == c) && !f)
        {
            *name = l;
            return 1;
        }
    }
    return 0;
}

int trans(int i, int j, int t, int c, int *count, int *arr)
{
    int k = 0, co, temp;
    *count = 0;

    for (k = 0; k < c; k++)
    {
        temp = Fa[i][j][k];
        co = 0;
        while (Fa[temp][t][co] != -1)
        {
            arr[*count] = Fa[temp][t][co++];
            (*count)++;
        }
    }
    return 0;
}

int nfa2dfa(int start, int end)
{
    int j, t, c, i, k, count, arr[10], name, l;
    for (i = start; i <= end; i++)
    {
        for (j = 0; j <= max_inp; j++)
        {
            c = 0;
            t = 0;
            while (Fa[i][j][t] >= 0)
            {
                t++;
                c++;
            }
            if (c > 1)
            {
                if (check(i, j, c, &name) == 0)
                {
                    for (k = 0; k < c; k++)
                    {
                        new_state[stat][k] = Fa[i][j][k];
                        for (l = 0; states[1][l] != -1; l++)
                            if (new_state[stat][k] == states[1][l] && !search(stat))
                                states[1][no_stat++] = stat;
                    }

```

```
                    for (t = 0; t <= max_inp; t++)
                    {
                        count = 0;
                        for (k = 0; k < 10; k++)
                            arr[k] = -1;
                        trans(i, j, t, c, &count, arr);

                        checkcon(arr, &count);

                        sort(arr, count);
                        remove_duplicate(arr, &count);

                        for (k = 0; k < count; k++)
                            Fa[stat][t][k] = arr[k];
                    }
                    Fa[i][j][0] = stat++;
                    for (t = 1; t < c; t++)
                        Fa[i][j][t] = -1;
                }
                else
                {
                    Fa[i][j][0] = name;
                    for (t = 1; t < c; t++)
                        Fa[i][j][t] = -1;
                }
            }
        }
    }
    return 0;
}
int main()
{

    int i, j, k, flag = 0, start, end;
    char c, ch;
    fp = fopen("Nfa_ip.txt", "r+");

    for (i = 0; i < 2; i++)
        for (j = 0; j < 10; j++)
            states[i][j] = -1;

    for (i = 0; i < 10; i++)
        for (j = 0; j < 10; j++)
            new_state[i][j] = -1;

    for (i = 0; i < 10; i++)
        for (j = 0; j < 10; j++)
            for (k = 0; k < 10; k++)
                Fa[i][j][k] = -1;

    while (fscanf(fp, "%d", &in) != EOF)
    {
        fscanf(fp, "%c", &c);

        if (flag)
        {
            states[sr][sc++] = in;
            if (c == '\n')
            {
                sr++;
                sc = 0;
            }
        }
        else if (c == '#')
        {
            flag = 1;
            Fa[row][col][th] = in;
        }
        else if (!flag)
        {
            Fa[row][col][th] = in;
            if (c == ',')
            {
                th++;
            }
            else if (c == '\n')
            {
                if (max_inp < col)
```

```
                    max_inp = col;
                col = 0;
                row++;
                th = 0;
            }
            else if (c != ',')
            {
                col++;
                th = 0;
            }
        }
    }
    no_stat = 0;
    i = 0;
    while (states[1][i++] != -1)
        no_stat++;
    stat = row + 1;
    start = 0;
    end = row;
    while (1)
    {
        nfa2dfa(start, end);
        start = end + 1;
        end = row;
        if (start > end)
            break;
    }

    printf("\n\nDFA IS : \n\n\n");
    for (i = 0; i <= max_inp; i++)
        printf("\t%d", i);
    printf("\n");
    printf("----------------------------\n");

    for (i = 0; i < stat; i++)
    {
        printf("%d-> |", i);
        for (j = 0; j <= max_inp; j++)
        {
            printf("%2d        ", Fa[i][j][0]);
        }
        printf("\n");
    }
    printf("\n\n");
    printf("Total Number Of State Is : %d \n\n", stat);
    printf("Final States Are : ");
    for (i = 0; states[1][i] != -1; i++)
        printf("%d ", states[1][i]);

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

Output

```
ritvik@ritvik-G3-3500:~/OpenCv$ gcc cd.c
ritvik@ritvik-G3-3500:~/OpenCv$ ./a.out


 DFA IS :


         0       1
----------------------------
0-> | 3          1
1-> |-1          2
2-> |-1         -1
3-> |-1          2


 Total Number Of State Is : 4

 Final States Are : 2 3
```

Ques-3

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

//Global Variables
int z = 0, i = 0, j = 0, c = 0;

// Modify array size to increase
// length of string to be parsed
char a[16], ac[20], stk[15], act[10];

// This Function will check whether
// the stack contain a production rule
// which is to be Reduce.
// Rules can be E->2E2 , E->3E3 , E->4
void check()
{
    // Copying string to be printed as action
    strcpy(ac,"REDUCE TO E -> ");

    // c=length of input string
    for(z = 0; z < c; z++)
    {
        //checking for producing rule E->4
        if(stk[z] == '4')
        {
            printf("%s4", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';

            //printing action
            printf("\n$%s\t%s$\t", stk, a);
        }
    }

    for(z = 0; z < c - 2; z++)
    {
        //checking for another production
        if(stk[z] == '2' && stk[z + 1] == 'E' &&
                                    stk[z + 2] == '2')
        {
            printf("%s2E2", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';
            stk[z + 2] = '\0';
            printf("\n$%s\t%s$\t", stk, a);
            i = i - 2;
        }

    }

    for(z=0; z<c-2; z++)
    {
        //checking for E->3E3
        if(stk[z] == '3' && stk[z + 1] == 'E' &&
                                    stk[z + 2] == '3')
        {
            printf("%s3E3", ac);
            stk[z]='E';
            stk[z + 1]='\0';
            stk[z + 1]='\0';
            printf("\n$%s\t%s$\t", stk, a);
            i = i - 2;
        }
    }
    return ; //return to main
}

//Driver Function
int main()
{
    printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");

    // a is input string
    strcpy(a,"32423");

    // strlen(a) will return the length of a to c
    c=strlen(a);
```

```
79
80      // "SHIFT" is copied to act to be printed
81      strcpy(act,"SHIFT");
82
83      // This will print Labels (column name)
84      printf("\nstack \t input \t action");
85
86      // This will print the initial
87      // values of stack and input
88      printf("\n$\t%s$\t", a);
89
90      // This will Run upto length of input string
91      for(i = 0; j < c; i++, j++)
92      {
93          // Printing action
94          printf("%s", act);
95
96          // Pushing into stack
97          stk[i] = a[j];
98          stk[i + 1] = '\0';
99
00          // Moving the pointer
01          a[j]=' ';
02
03          // Printing action
04          printf("\n$%s\t%s$\t", stk, a);
05
06          // Call check function ..which will
07          // check the stack whether its contain
08          // any production or not
09          check();
10      }
11
12      // Rechecking last time if contain
13      // any valid production then it will
14      // replace otherwise invalid
15      check();
16
17      // if top of the stack is E(starting symbol)
```

```
118     // then it will accept the input
119     if(stk[0] == 'E' && stk[1] == '\0')
120         printf("Accept\n");
121     else //else reject
122         printf("Reject\n");
123
124 }
```

Output

```
ritvik@ritvik-G3-3500:~/OpenCv$ gcc cd.c
ritvik@ritvik-G3-3500:~/OpenCv$ ./a.out
GRAMMAR is -
E->2E2
E->3E3
E->4

stack    input   action
$       32423$  SHIFT
$3       2423$  SHIFT
$32       423$  SHIFT
$324       23$  REDUCE TO E -> 4
$32E       23$  SHIFT
$32E2       3$  REDUCE TO E -> 2E2
$3E         3$  SHIFT
$3E3         $  REDUCE TO E -> 3E3
$E           $  Accept
ritvik@ritvik-G3-3500:~/OpenCv$
```

Ques-4

cd.c > main()

```c
#include<stdio.h>
#include<string.h>

char *input;
int i=0;
char lasthandle[6],stack[50],handles[][5]={")E(","E*E","E+E","i","E^E"};
//(E) becomes )E( when pushed to stack

int top=0,l;
char prec[9][9]={

                    /*input*/

            /*stack    +    -    *    /    ^    i    (    )    $  */

            /*  + */  '>', '>','<','<','<','<','<','>','>',

            /*  - */  '>', '>','<','<','<','<','<','>','>',

            /*  * */  '>', '>','>','>','<','<','<','>','>',

            /*  / */  '>', '>','>','>','<','<','<','>','>',

            /*  ^ */  '>', '>','>','>','<','<','<','>','>',

            /*  i */  '>', '>','>','>','>','e','e','>','>',

            /*  ( */  '<', '<','<','<','<','<','<','>','e',

            /*  ) */  '>', '>','>','>','>','e','e','>','>',

            /*  $ */  '<', '<','<','<','<','<','<','<','>',

                };

int getindex(char c)
{
switch(c)
    {
```

cd.c > main()

```c
    case '+':return 0;
    case '-':return 1;
    case '*':return 2;
    case '/':return 3;
    case '^':return 4;
    case 'i':return 5;
    case '(':return 6;
    case ')':return 7;
    case '$':return 8;
    }
}


int shift()
{
stack[++top]=*(input+i++);
stack[top+1]='\0';
}


int reduce()
{
int i,len,found,t;
for(i=0;i<5;i++)//selecting handles
    {
    len=strlen(handles[i]);
    if(stack[top]==handles[i][0]&&top+1>=len)
        {
        found=1;
        for(t=0;t<len;t++)
            {
            if(stack[top-t]!=handles[i][t])
                {
                found=0;
                break;
                }
            }
        if(found==1)
            {
```

cd.c > main()

```
                stack[top-t+1]='E';
                top=top-t+1;
                strcpy(lasthandle,handles[i]);
                stack[top+1]='\0';
                return 1;//successful reduction
                }
            }
      }
return 0;
}



void dispstack()
{
int j;
for(j=0;j<=top;j++)
    printf("%c",stack[j]);
}



void dispinput()
{
int j;
for(j=i;j<l;j++)
    printf("%c",*(input+j));
}



void main()
{
int j;

input=(char*)malloc(50*sizeof(char));
printf("\nEnter the string\n");
scanf("%s",input);
input=strcat(input,"$");
```

cd.c > main()

```
l=strlen(input);
strcpy(stack,"$");
printf("\nSTACK\tINPUT\tACTION");
while(i<=l)
    {
    shift();
    printf("\n");
    dispstack();
    printf("\t");
    dispinput();
    printf("\tShift");
    if(prec[getindex(stack[top])][getindex(input[i])]=='>')
        {
        while(reduce())
            {
            printf("\n");
            dispstack();
            printf("\t");
            dispinput();
            printf("\tReduced: E->%s",lasthandle);
            }
        }
    }

if(strcmp(stack,"$E$")==0)
    printf("\nAccepted;");
else
    printf("\nNot Accepted;");
}
```

Output

```
ritvik@ritvik-G3-3500:~/OpenCv$ ./a.out

Enter the string
i+(i*i)+i

STACK   INPUT   ACTION
$i      +(i*i)+i$       Shift
$E      +(i*i)+i$       Reduced: E->i
$E+     (i*i)+i$        Shift
$E+(    i*i)+i$ Shift
$E+(i   *i)+i$  Shift
$E+(E   *i)+i$  Reduced: E->i
$E+(E*  i)+i$   Shift
$E+(E*i )+i$    Shift
$E+(E*E )+i$    Reduced: E->i
$E+(E   )+i$    Reduced: E->E*E
$E+(E)  +i$     Shift
$E+E    +i$     Reduced: E->)E(
$E      +i$     Reduced: E->E+E
$E+     i$      Shift
$E+i    $       Shift
$E+E    $       Reduced: E->i
$E      $       Reduced: E->E+E
$E$             Shift
$E$             Shift
```

Ques-5

cd.c > F()

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

char input[10];
int i, error;
void E();
void T();
void Eprime();
void Tprime();
void F();
main()
{
    i = 0;
    error = 0;
    printf("Enter an arithmetic expression   :  "); // Eg: a+a*a
    gets(input);
    E();
    if (strlen(input) == i && error == 0)
        printf("\nAccepted..!!!\n");
    else
        printf("\nRejected..!!!\n");
}

void E()
{
    T();
    Eprime();
}
void Eprime()
{
    if (input[i] == '+')
    {
        i++;
        T();
        Eprime();
    }
}
void T()
```

```c
{
    F();
    Tprime();
}
void Tprime()
{
    if (input[i] == '*')
    {
        i++;
        F();
        Tprime();
    }
}
void F()
{
    if (isalnum(input[i]))
        i++;
    else if (input[i] == '(')
    {
        i++;
        E();
        if (input[i] == ')')
            i++;

        else
            error = 1;
    }

    else
        error = 1;
}
```

Output

```
ritvik@ritvik-G3-3500:~/OpenCv$ ./a.out
Enter an arithmetic expression   :  a+a*a

Accepted..!!!
ritvik@ritvik-G3-3500:~/OpenCv$
```

Ques-6

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct op
{
    char l;
    char r[20];
} op[10], pr[10];
void main()
{
    int a, i, k, j, n, z = 0, m, q;
    char *p, *l;
    char temp, t;
    char *tem;
    printf("enter no of values");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("\tleft\t");
        op[i].l = getche();
        printf("\tright:\t");
        scanf("%s", op[i].r);
    }
    printf("intermediate Code\n");
    for (i = 0; i < n; i++)
    {
        printf("%c=", op[i].l);
        printf("%s\n", op[i].r);
    }
    for (i = 0; i < n - 1; i++)
    {
        temp = op[i].l;
        for (j = 0; j < n; j++)
        {
            p = strchr(op[j].r, temp);
            if (p)
            {
                pr[z].l = op[i].l;
                strcpy(pr[z].r, op[i].r);
```

cd.c nfa.c Preview CHANGELOG.md Go for VS Code launch.json Nfa_ip.txt

cd.c > ...

```c
                z++;
            }
        }
    }
    pr[z].l = op[n - 1].l;
    strcpy(pr[z].r, op[n - 1].r);
    z++;
    printf("\nafter dead code elimination\n");
    for (k = 0; k < z; k++)
    {
        printf("%c\t=", pr[k].l);
        printf("%s\n", pr[k].r);
    }
    // sub expression elimination
    for (m = 0; m < z; m++)
    {
        temp = pr[m].r;
        for (j = m + 1; j < z; j++)
        {
            p = strstr(tem, pr[j].r);
            if (p)
            {
                t = pr[j].l;
                pr[j].l = pr[m].l;
                for (i = 0; i < z; i++)
                {
                    l = strchr(pr[i].r, t);
                    if (l)
                    {
                        a = l - pr[i].r;
                        // printf("pos: %d",a);
                        pr[i].r[a] = pr[m].l;
                    }
                }
            }
        }
    }
    printf("eliminate common expression\n");
    for (i = 0; i < z; i++)
```

cd.c nfa.c Preview CHANGELOG.md Go for VS Code launch.json Nfa_ip.txt

cd.c > ...

```c
    {
        printf("%c\t=", pr[i].l);
        printf("%s\n", pr[i].r);
    }
    // duplicate production elimination
    for (i = 0; i < z; i++)
    {
        for (j = i + 1; j < z; j++)
        {
            q = strcmp(pr[i].r, pr[j].r);
            if ((pr[i].l == pr[j].l) && !q)
            {
                pr[i].l = '\0';
                strcpy(pr[i].r, '\0');
            }
        }
        printf("optimized code");
        for (i = 0; i < z; i++)
        {
            if (pr[i].l != '\0')
            {
                printf("%c=", pr[i].l);
                printf("%s\n", pr[i].r);
            }
        }
        getch();
    }
}
```

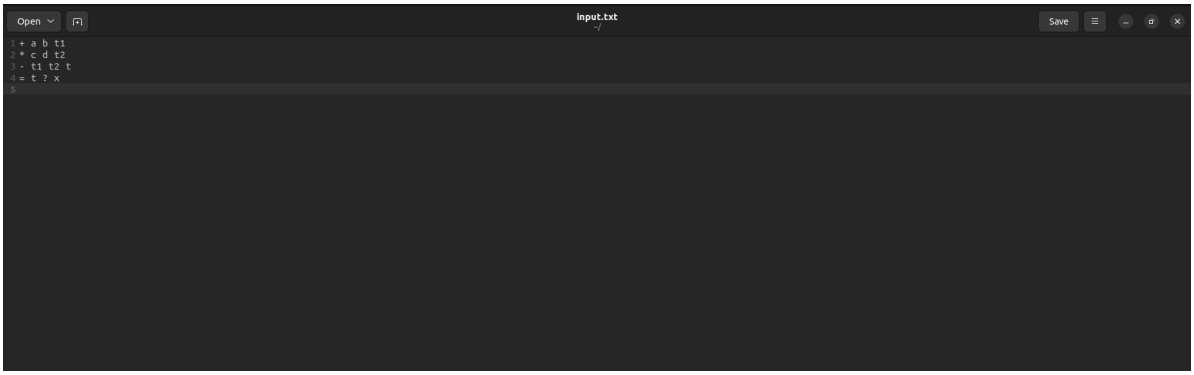
Output

```
enter no of values3
        left    1       right:  1
        left    1       right:  2
        left    3       right:  4
intermediate Code
1=1
1=2
3=4

after dead code elimination
1       =1
1       =2
3       =4
eliminate common expression
1       =1
1       =2
3       =4
optimized code1=1
1=2
3=4

--------------------------------
Process exited after 36.74 seconds with return value 4
Press any key to continue . . .
```

Ques-7

```c
#include <stdio.h>
#include <curses.h>
#include <string.h>
char op[2], arg1[5], arg2[5], result[5];
void main()
{
    FILE *fp1, *fp2;
    fp1 = fopen("input.txt", "r");
    fp2 = fopen("output.txt", "w");
    while (!feof(fp1))
    {

        fscanf(fp1, "%s%s%s%s", op, arg1, arg2, result);
        if (strcmp(op, "+") == 0)
        {
            fprintf(fp2, "\nMOV R0,%s", arg1);
            fprintf(fp2, "\nADD R0,%s", arg2);
            fprintf(fp2, "\nMOV %s,R0", result);
        }
        if (strcmp(op, "*") == 0)
        {
            fprintf(fp2, "\nMOV R0,%s", arg1);
            fprintf(fp2, "\nMUL R0,%s", arg2);
            fprintf(fp2, "\nMOV %s,R0", result);
        }
        if (strcmp(op, "-") == 0)
        {
            fprintf(fp2, "\nMOV R0,%s", arg1);
            fprintf(fp2, "\nSUB R0,%s", arg2);
            fprintf(fp2, "\nMOV %s,R0", result);
        }
        if (strcmp(op, "/") == 0)
        {
            fprintf(fp2, "\nMOV R0,%s", arg1);
            fprintf(fp2, "\nDIV R0,%s", arg2);
            fprintf(fp2, "\nMOV %s,R0", result);
        }
        if (strcmp(op, "=") == 0)
        {
            fprintf(fp2, "\nMOV R0,%s", arg1);
            fprintf(fp2, "\nMOV %s,R0", result);
        }
    }
    fclose(fp1);
    fclose(fp2);
    return;
}
```

Output


Open
input.txt
~/
Save
+ a b t1
* c d t2
- t1 t2 t
= t ? x


Open
output.txt
~/
Save
MOV R0,a
ADD R0,b
MOV t1,R0
MOV R0,c
MUL R0,d
MOV t2,R0
MOV R0,t1
SUB R0,t2
MOV t,R0
MOV R0,t
MOV x,R0
MOV R0,t
MOV x,R0