Name : Rapariya Dhruv Dineshbhai
Div : B
Roll No : 3162
Subject : SS ( Practical )

Q-1

**Implement Recursive Descent parser. (check a-b/c and a//b)**

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

struct treenode{
   char info;
   struct treenode *left;
   struct treenode *right;
}*temp,*a,*b,*c,*d,*temp1,*root;

typedef struct treenode node;
node *proc_e(char input[]);
node *proc_t(char input[]);
node *proc_v(char input[]);

void traversal(node *temp);

int ssm=0;

void main()
{
```

```
    char input[20];
    ssm=0;
    clrscr();
    printf("\n3162 Rapariya Dhruv D.\n");
    printf("Enter String");
    scanf("%s",&input);
    root=proc_e(input);
    printf("Parser Tree:");
    traversal(root);

    getch();
}

node *proc_e(char input[])
{
    char ch;
    a = proc_t(input);
    while(input[ssm]=='+' || input[ssm] == '-')
    {
    ch = input[ssm];
    ssm++;
    b=proc_t(input);
    temp=(struct treenode*)malloc(sizeof(struct treenode));
    temp->info = ch;
    temp->left = a;
    temp->right = b;
    a = temp;
    }
    return a;
}
```

```
node *proc_t(char input[])
{
   char ch;
   c=proc_v(input);
   ssm = ssm + 1;
   while(input[ssm]=='*' || input[ssm]=='/')
   {
   ch = input[ssm];
   ssm++;
   d=proc_v(input);
   temp = (node*)malloc(sizeof(node));
   temp->info = ch;
   temp->left = c;
   temp->right = d;
   c = temp;
   ssm = ssm + 1;
   }
   return c;
}

node *proc_v(char input[])
{
   if(isalpha(input[ssm]))
   {
      temp = (node*)malloc(sizeof(node));
      temp->info = input[ssm];
      temp->left = NULL;
      temp->right = NULL;
      return temp;
   }
   else
```

```
    {
    printf("Error %c",input[ssm]);
    getch();
    exit(0);
    }
    // return c;
}

void traversal(node *temp1)
{
    if(temp1!=NULL)
    {
        printf("%c",temp1->info);
        traversal(temp1->left);
        traversal(temp1->right);
    }
}
```


Q-2

**Implement Operator Precedence Parser. (check (a-b)*c and a//c)**

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

struct Node
{
   char info;
   struct Node *left;
   struct Node *right;
```

```
};

struct Stack{
   char info;
   struct Node *next;
};

struct Stack st[10];

int top = -1,ssm = 0;
int i,j;

char table[9][9] = {
   {'_','+','*','-','/','^','(',')','$'},
   {'+','>','<','>','<','<','<','>','>'},
   {'*','>','>','>','>','<','<','>','>'},
   {'-','>','<','>','<','<','<','>','>'},
   {'/','>','>','>','>','<','<','>','>'},
   {'^','>','>','>','>','>','<','>','>'},
   {'(','<','<','<','<','<','<','=',' '},
   {')','>','>','>','>','>',' ','>','>'},
   {'$','<','<','<','<','<','<',' ','='},
};

char s[30];
```

```
struct Node* makenode(char info,struct Node* l,struct Node* r)
{
   struct Node *temp = (struct Node*)malloc(sizeof(struct Node));
   temp->info = info;
   temp->left = l;
   temp->right = r;
   return temp;
}

char check()
{
   int i,j;
   for(i=1;i<9;i++)
   {
      if(table[i][0] == st[top].info)
      {
         break;
      }
   }

   for(j=1;j<9;j++)
   {
      if(table[0][j] == s[ssm])
      {
         break;
      }
```

```
    }

    if(table[i][j] == ' ')
    {
        printf("Error : Invalid Expression");
        getch();
        exit(0);
    }

    return table[i][j];
}

void inorder(struct Node *ptr)
{
    if(ptr!=NULL)
    {
        inorder(ptr->left);
        printf("%c",ptr->info);
        inorder(ptr->right);
    }
}

int parse()
{
    char priority;
    st[++top].info = s[ssm];
```

```
   while(1)
   {
      if(s[++ssm] == '$' || s[ssm] == '(' || s[ssm] == ')' || s[ssm] == '+' || s[ssm] == '*' || s[ssm]
== '-' || s[ssm] == '/' || s[ssm] == '^')
      {
         if(s[ssm] == ')' && st[top].info == '(')
         {
            printf("Error : Invalid Expression");
            getch();
            exit(0);
         }

         if( (s[ssm] == '+' || s[ssm] == '*' || s[ssm] == '-' || s[ssm] == '/' || s[ssm] == '^') && (
s[ssm+1] == '+' || s[ssm+1] == '*' || s[ssm+1] == '-' || s[ssm+1] == '/' || s[ssm+1] == '^'))
         {
            printf("Error : Invalid Expression");
            getch();
            exit(0);
         }

         priority = check();

         while (priority == '>')
         {
            st[--top].next = makenode(st[top+1].info,st[top].next,st[top+1].next);
            priority = check();
```

```
    }

    if(priority == '<')
    {
        st[++top].info = s[ssm];
    }
    else
    {
        if(st[top].info == '$' && !top)
        {
            return 1;
        }
        if(st[top].info == '$' && top)
        {
            return 0;
        }
        if(st[top].info == '(')
        {
            st[--top].next = st[top+1].next;
        }
    }

}
else
{
    st[top].next = makenode(s[ssm],NULL,NULL);
```

```
        }
    }
}

int main()
{
    clrscr();
    printf("\n3162 Rapariya Dhruv D.\n");
    printf("Enter Input : ");
    scanf("%s",s);

    if(parse())
    {
        printf("Done\n");
        inorder(st[top].next);
    }
    else
    {
        printf("Not done");
    }

    getch();
    return 0;
}
```

**3162** **Rapariya Dhruv Dineshbhai**

```
3162 Rapariya Dhruv D.
Enter Input : $(a-b)*c$
Done
a-b*c
```

```
3162 Rapariya Dhruv D.
Enter Input : $a//c$
Error : Invalid Expression
```