***** Refer DSA Lab5 and Lab6 instructions videos *****

STACK CONCEPTS

Objectives:

In this lab, student will be able to:

- Understand stack as a data structure
- Design programs using stack concepts

I. SOLVED EXERCISE:

1) Write a c program to check if the given parenthesized expression has properly matching open and closing parenthesis.

Description: We use a stack to check the expression for matching opening and closing parenthesis. Here, the expression is scanned from start to end if an opening brace is encountered then it is pushed on to the stack. When a closing parenthesis is encountered a pop operation is performed. Ideally, if the number of opening and closing braces matches, then the stack will be empty after checking the entire expression.

Algorithm:

```
Step1: Set balanced to true
Step2: Set symbol to the first character in current expression
Step3: while (there are still more characters AND expression is still balanced)
            if (symbol is an opening symbol)
                Push symbol onto the stack
            else if (symbol is a closing symbol)
                if the stack is empty
                    Set balanced to false
                Else
                    Set openSymbol to the character at the top of the stack
                    Pop the stack
                    Set balanced to (symbol matches openSymbol)
```

```
            Set symbol to next character in current expression
Step4:  if (balanced)
            Write "Expression is well formed."
        else
            Write "Expression is not well formed."
  Step5: stop
```

Program:

File name: stack_operations.h

```
# define MAX 10
# define true 1
# define false 0
/* Structure definition */
typedef struct
{
        char item[MAX];
        int top;
}stack;
void push(stack *ps,char x);
char pop(stack *ps);
int empty(stack *ps);
/* Push operation */
void push(stack *ps,char x)
{
        if (ps->top!=MAX-1)
        {
                ps->top++;
                ps->item[ps->top]=x;
        }
}
```

```
 /* Pop operation */
char pop(stack *ps)
{
        if(!empty(ps))
          return(ps->item[ps->top--]);
}

/* Stack empty operation */
int empty(stack *ps)
{
        if (ps->top==-1)
                return(true);
        else
                return(false);
}
```

File name: check_expr.c

```
#include <stdio.h>
#include <stdlib.h>
#include "stack_operations.h"
void main()
{
        char expn[25],c,d;
        int i=0;
        stack s;
        s.top=-1;
        printf("\n Enter the expression: ");
        gets(expn);
        while((c=expn[i++])!='\0')
        {
                if(c=='(')
                        push(&s,c);
                else
                        if(c==')')
```

```
                        {
                                d=pop(&s);
                                if(d!='(')
                                {
                                printf("\n Invalid Expression");
                                break;
                                }
                        }
        }
        if(empty(&s))
                printf("\n Balanced Expression");
        else
                printf("\n Not a Balanced  Expression");
}
```

Sample Input and Output

Run 1:

Enter the expression: (a+b)+(c*d*(a-b)

Not a Balanced Expression

Run 2:

Enter the expression: (a+b)+(c*d*(a-b))

Balanced Expression

STACK APPLICATIONS

Objectives:

In this lab, student will be able to:

- Identify the need for Stack data structure in a given problem.
- Develop c programs applying stack concepts

I. SOLVED EXERCISE:

1) Program for evaluation of postfix expression in C

Algorithm for Evaluation of Postfix Expression

Create an empty stack and start scanning the postfix expression from left to right.

- If the element is an operand, push it into the stack.
- If the element is an operator **O**, pop twice and get A and B respectively. Calculate B**O**A and push it back to the stack.
- When the expression is ended, the value in the stack is the final answer.

Evaluation of a postfix expression using a stack is explained in below example (fig. 2):

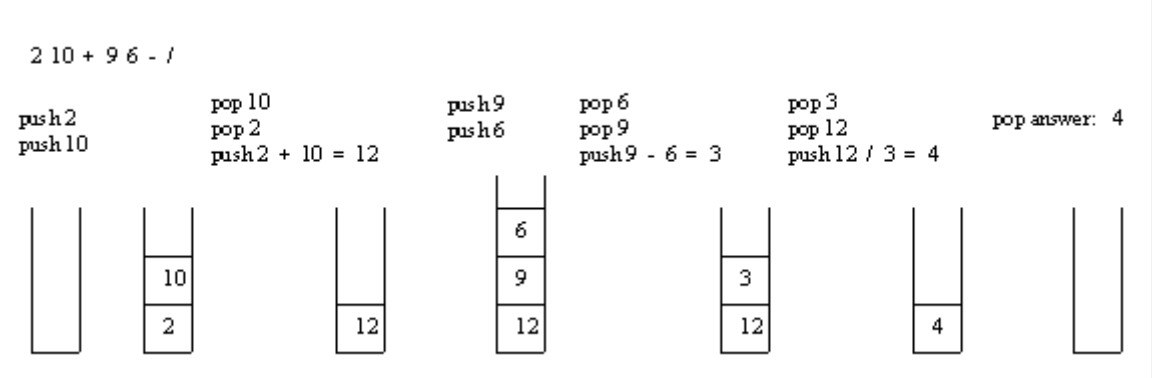

Figure 2: Illustrates the evaluation of a postfix expression using a stack

File name: eval_postfix_fun.h

```
#define MAX 20

typedef struct stack
{
   int data[MAX];
   int top;
}stack;

void init(stack *);
int empty(stack *);
```

```
int full(stack *);
int pop(stack *);
void push(stack *,int);
int evaluate(char x,int op1,int op2);

int evaluate(char x,int op1,int op2)
{
   if(x=='+')
      return(op1+op2);
   if(x=='-')
      return(op1-op2);
   if(x=='*')
      return(op1*op2);
   if(x=='/')
      return(op1/op2);
   if(x=='%')
      return(op1%op2);
}

void init(stack *s)
{
   s->top=-1;
}

int empty(stack *s)
{
   if(s->top==-1)
      return(1);

   return(0);
}

int full(stack *s)
{
   if(s->top==MAX-1)
      return(1);

   return(0);
}

void push(stack *s,int x)
{
   s->top=s->top+1;
   s->data[s->top]=x;
}

int pop(stack *s)
{
   int x;
   x=s->data[s->top];
```

```
    s->top=s->top-1;
    return(x);
}
```

File name:eval_postfix_expr.c

```
#include<stdio.h>
#include "eval_postfix_fun.h"
int main()
{
    stack s;
    char x;
    int op1,op2,val;
    init(&s);
    printf("Enter the expression(eg: 59+3*)\nsingle digit operand and operators
            only:");

    while((x=getchar())!='\n')
    {
        if(isdigit(x))
            push(&s,x-'0');        /*x-'0' for removing the effect of ascii  */
        else
        {
            op2=pop(&s);
            op1=pop(&s);
            val=evaluate(x,op1,op2);
            push(&s,val);
        }
    }
    val=pop(&s);
    printf("\nvalue of expression=%d",val);
    return 0;
}
```

Sample Input and Output:

Enter the expression(eg: 59+3*)
single digit operand and operators only: 12+3*
value of expression= 9

Questions for Lab3

Write a 'C' program to:

1) Implement a menu driven program to define a stack of characters. Include push, pop and display functions. Also include functions for checking error conditions such as underflow and overflow (ref. figure 1) by defining isEmpty and isFull functions. Use these function in push, pop and display functions appropriately. Use type defined structure to define a STACK containing a character array and an integer top. Do not use global variables.

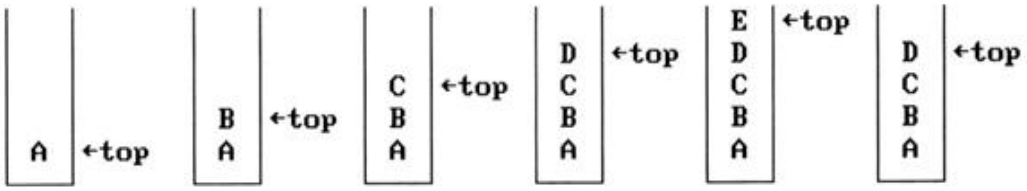

Figure 1: Inserting and deleting elements in a stack

2) Convert a given decimal number to binary using stack.
3) Determine whether a given string is palindrome or not using stack.

Questions for Lab4

Write a C program to:

1) Evaluate a given prefix expression using stack.
2) Convert an infix expression to prefix.
3) Implement two stacks in an array.