Experiment No.: 3
Title: Expression Conversion and Dynamic Stack Implementation

Batch: B-1 **Roll No.: 16010422234** **Name: Chandana Ramesh Galgali**

Experiment No.:3

Aim:

a) WAP to create a stack using SLL by implementing following operations
1. Create stack, 2. Insert an element, 3. Delete an element, 4. Display top element.

b) WAP to convert a given infix expression into equivalent postfix form using stack implemented in part (a).

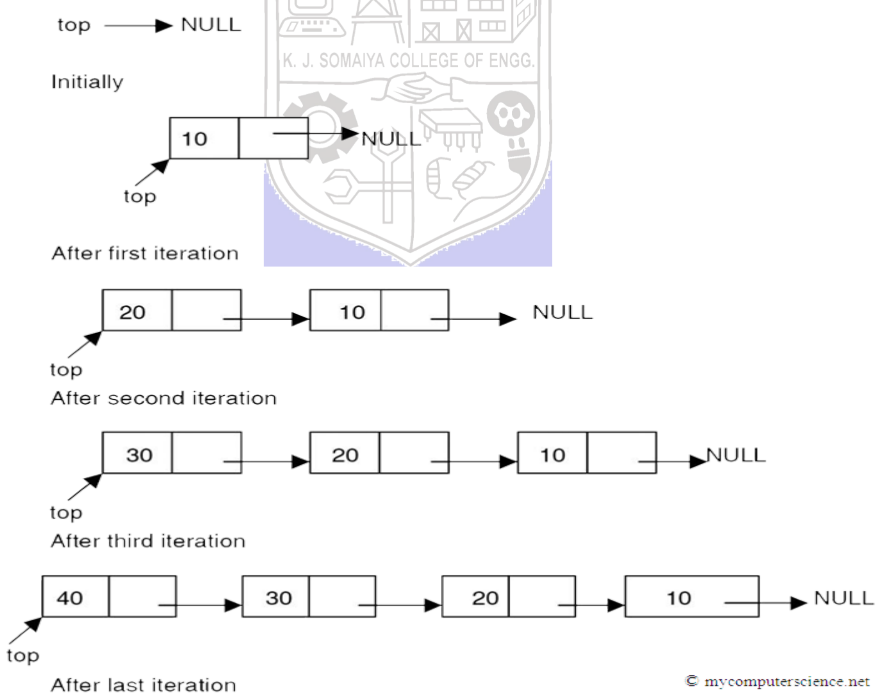
Resources Used: Turbo C/ C++/JAVA editor and compiler (online/offline)

Theory:

a) Stack :- Stack can be implemented by using an array or by using a linked list. One important feature of stack, that the last element inserted into a stack is the first element deleted. Therefore stack is also called as Last in First out (LIFO) list.

Linked List implementation –

Fig(c) shows the push operation working on stack using SLL



Fig(c) Push operation using SLL

b) Expression conversion :- Calculators employing reverse Polish notation use a stack structure to hold values. Expressions can be represented in prefix, postfix or infix notations. Conversion from one form of the expression to another form may be accomplished using a stack. Many compilers use a stack for parsing the syntax of expressions, program blocks etc. before translating into low level code. Most of the programming languages are context-free languages allowing them to be parsed with stack based machines.

Examples

Infix expression	***(2 * 5 - 1 * 2) / (11 – 9)***
Postfix Expression	***2 5 * 1 2 * - 11 9 - /***

Algorithm :

Infix String : a+b*c-d

```
1. Read an item from input infix expression
2. If item is an operand append it to postfix string
3. If item is "(" push it on the stack
4. If the item is an operator and top of the stack(tos) is also operator
    1. If the operator has higher precedence than the one on tos then push it onto
       the operator stack
    2. If the operator has lower or equal precedence than the one on tos then
        1. pop the operator on tos and append it to postfix string( repeat if tos
           is again an operator)
        2. push lower precedence operator onto the stack
5. If item is ")" pop all operators from tos one-by-one and append it to postfix
   string, until a "(" is encountered on stack. Remove "(" from tos and discard it.
6. If end of infix string, pop the items from tos one-by-one and append to the postfix
   string. If other than operator anything is encountered on the stack at this step, then
   declare input as invalid input.
```

Infix String : a+b*c-d
Postfix String : abc*+d-

Activity :

a) Implement "dynamic stack" with following functions

1. **void createStack(void) :** Create and initialize the top to NULL, creating the empty stack.
2. **void push(char x) :** Creates a node with value 'x' and inserts on top of the stack.
3. **char pop(void) :** Deletes a node from top of the stack and returns the deleted value.
4. **boolean isEmpty(void) :** Returns "1" for stack empty; "0" otherwise.
5. **char peek(void) :** Return current stack top element.

NOTE : Map appropriate methods of SLL with the methods of STACK and use.

b) Implement expression conversion

1. Implement the above algorithm given for INFIX to POSTFIX expression conversion.

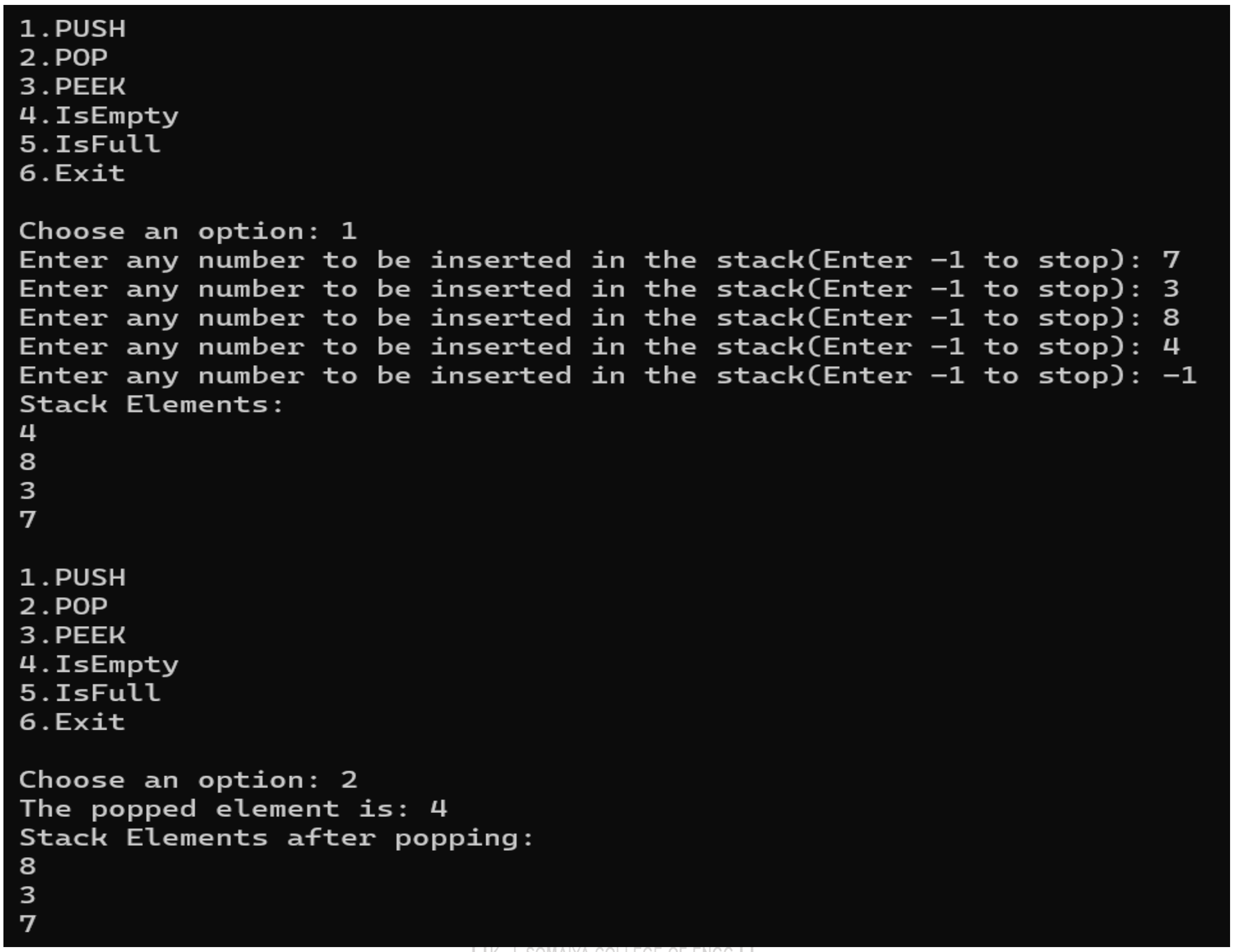
Results: A program depicting the correct behaviour of stack in conversion of expression and capable of handling all possible exceptional conditions and the same is reflected clearly in the output.

Program and Output:

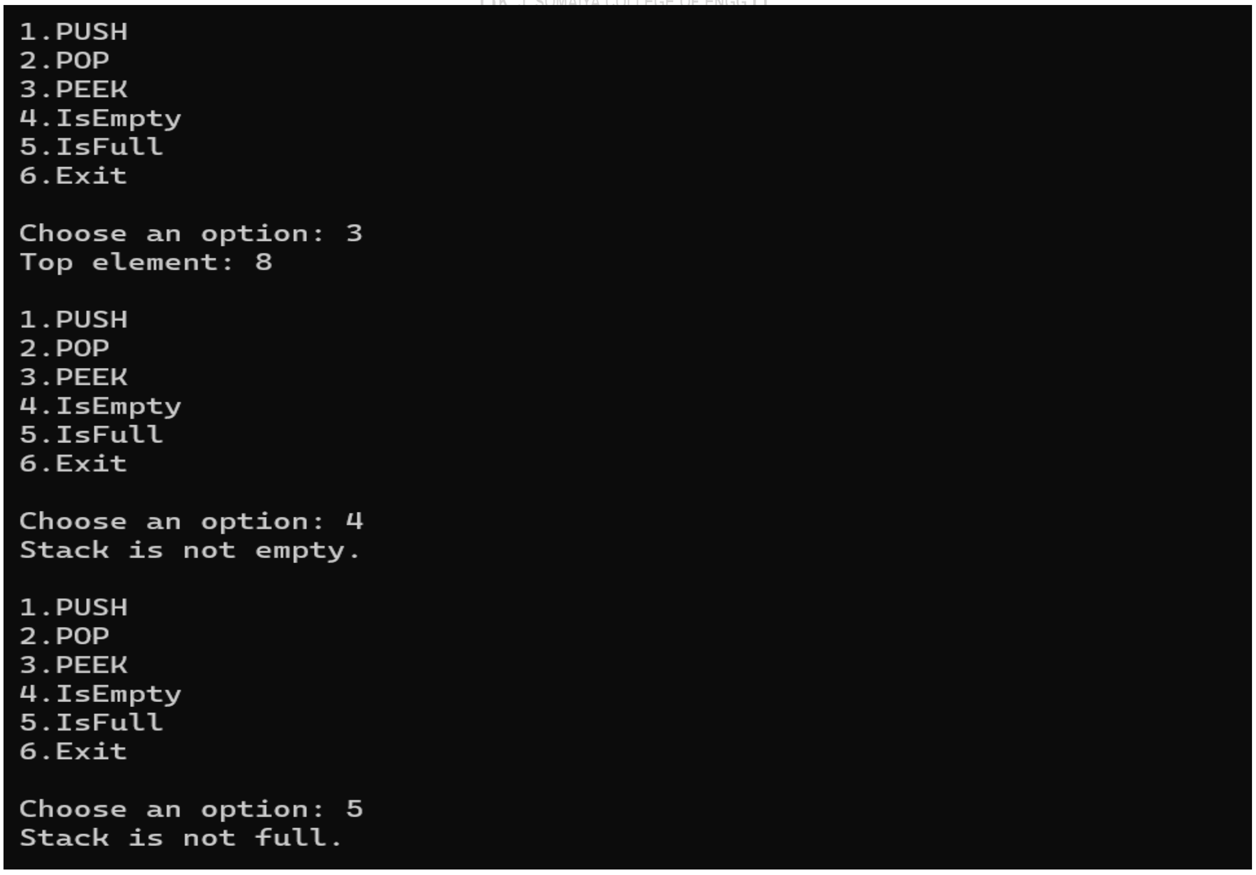
Stack Implementation using Array:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 20
struct stack{
   int TOP, a[MAX];
   }s;
int i;
int main()
{
   int choice;
   printf("\n1.PUSH\n2.POP\n3.PEEK\n4.IsEmpty\n5.IsFull\n6.Exit\n");
   printf("\nChoose an option: ");
   scanf("%d",&choice);
   switch(choice)
   {
   case 1:
      s.TOP=-1;
      push();
      main();
      break;
   case 2:
      pop();
      main();
      break;
   case 3:
      peek();
      main();
      break;
   case 4:
      if(isempty())
         printf("Stack is empty.\n");
      else
         printf("Stack is not empty.\n");
      main();
      break;
   case 5:
      if(isfull())
```

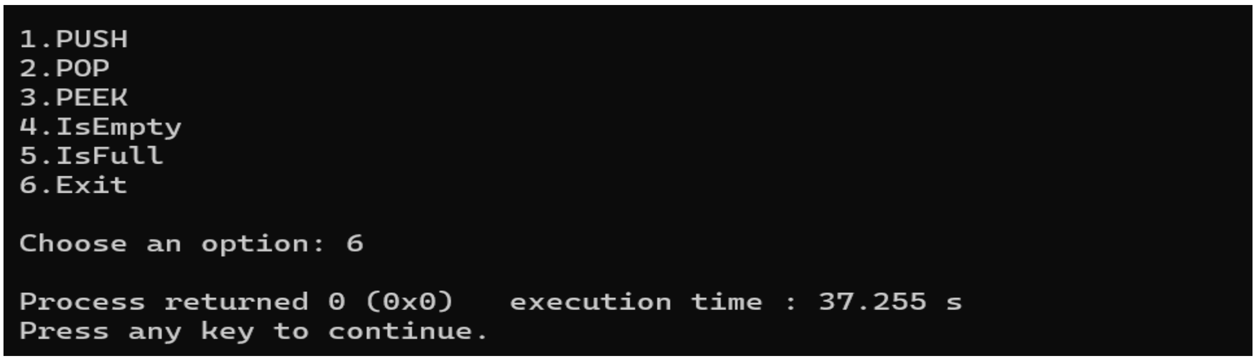
```
            printf("Stack is full.\n");
        else
            printf("Stack is not full.\n");
        main();
        break;
    case 6:
        break;
    default:
        printf("Choose a valid option!");
        break;
    }
    return 0;
}
int push()
{
    int val;
    if(s.TOP==MAX-1)
        {
            printf("Stack overflow.\n");
        }
    else
        {
            while(val!= -1)
            {
                printf("Enter any number to be inserted in the stack(Enter -1 to stop): ");
                scanf("%d",&val);
                s.TOP=s.TOP+1;
                s.a[s.TOP]=val;
            }
            printf("Stack Elements:\n");
            for(i=s.TOP;i>-1;i--)
            {
                if (s.a[i]!=-1)
                {
                    printf("%d\n",s.a[i]);
                }
            }
        }
    return s.TOP;
}
int pop()
{
    if(s.TOP==-1)
        {
            printf("Stack underflow.\n");
        }
    else
        {
            int x;
            s.TOP--;
            x = s.a[s.TOP];
            s.TOP=s.TOP-1;
            printf("The popped element is: %d\n",x);
            printf("Stack Elements after popping:\n");
            for(i=s.TOP;i>-1;i--)
            {
               printf("%d\n",s.a[i]);
            }
        }
    return s.TOP;
}
int peek()
{
    if(s.TOP==-1)
    {
       printf("Stack is empty.\n");
    }
    else
    {
       printf("Top element: %d\n",s.a[s.TOP]);
    }
}
int isempty()
{
    if(s.TOP==-1)
    {
       return 1;
    }
    else
    {
       return 0;
    }
}
int isfull()
{
     if(s.TOP==MAX-1)
    {
       return 1;
    }
    else
    {
       return 0;
    }
}
```



```
1.PUSH
2.POP
3.PEEK
4.IsEmpty
5.IsFull
6.Exit

Choose an option: 1
Enter any number to be inserted in the stack(Enter -1 to stop): 7
Enter any number to be inserted in the stack(Enter -1 to stop): 3
Enter any number to be inserted in the stack(Enter -1 to stop): 8
Enter any number to be inserted in the stack(Enter -1 to stop): 4
Enter any number to be inserted in the stack(Enter -1 to stop): -1
Stack Elements:
4
8
3
7

1.PUSH
2.POP
3.PEEK
4.IsEmpty
5.IsFull
6.Exit

Choose an option: 2
The popped element is: 4
Stack Elements after popping:
8
3
7
```



```
1.PUSH
2.POP
3.PEEK
4.IsEmpty
5.IsFull
6.Exit

Choose an option: 3
Top element: 8

1.PUSH
2.POP
3.PEEK
4.IsEmpty
5.IsFull
6.Exit

Choose an option: 4
Stack is not empty.

1.PUSH
2.POP
3.PEEK
4.IsEmpty
5.IsFull
6.Exit

Choose an option: 5
Stack is not full.
```



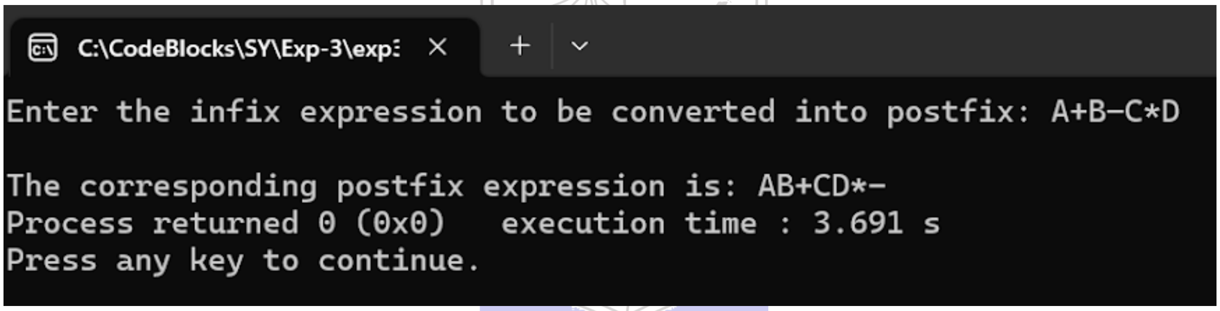
```
1.PUSH
2.POP
3.PEEK
4.IsEmpty
5.IsFull
6.Exit

Choose an option: 6

Process returned 0 (0x0)   execution time : 37.255 s
Press any key to continue.
```

Infix to Postfix:

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
#define MAX 100
struct stack{
   int TOP;
   char a[MAX];
   }s;
int main()
{
   s.TOP=-1;
   char infix[100],postfix[100];
   printf("Enter the infix expression to be converted into postfix: ");
   fgets(infix,sizeof(infix),stdin);
   infix[strcspn(infix, "\n")] = '\0';
   InfixToPostfix(infix,postfix);
   printf("\nThe corresponding postfix expression is: %s",postfix);
   return 0;
}
void push(char val)
{
   if(s.TOP==MAX-1)
      printf("\nStack overflow.");
   else
      {
         s.TOP=s.TOP+1;
         s.a[s.TOP]=val;
      }
}
char pop()
{
   char val='\0';
   if(s.TOP==-1)
      printf("\nStack underflow.");
   else
      {
         val = s.a[s.TOP];
```

```
            s.TOP=s.TOP-1;
        }
    return val;
}
int OperatorPriority(char op)
{
    if(op=='^')
        return 2;
    else if(op=='*'||op=='/'||op=='%')
        return 1;
    else if(op=='+'||op=='-')
        return 0;
    else
        return -1;
}
void InfixToPostfix(char source[],char target[])
{
    int i=0,j=0;
    char temp;
    strcpy(target,"");
    while(source[i]!='\0')
    {
        if(source[i]=='(')
        {
            push(source[i]);
            i++;
        }
        else if(source[i]==')')
        {
            while(s.TOP!=1 && s.a[s.TOP]!='(')
            {
                target[j]=pop();
                j++;
            }
            if(s.TOP==-1)
            {
                printf("\nInvalid infix expression!");
            }
            temp=pop();
            i++;
        }
        else if(isdigit(source[i])||isalpha(source[i]))
        {
            target[j]=source[i];
            j++;
            i++;
        }
                                                                              else
if(source[i]=='+'||source[i]=='-'||source[i]=='*'||source[i]=='/'||source[i]=='%'||source[i]=='^')
        {
```

```
                                                    while((s.TOP!=-1)     &&     (s.a[s.TOP]!='(')     &&
(OperatorPriority(s.a[s.TOP])>=OperatorPriority(source[i])))
            {
            target[j] = pop();
            j++;
            }
            push(source[i]);
            i++;
        }
        else
        {
            printf("\nInvalid infix expression!");
            return;
        }
    }
    while(s.TOP!=-1 && s.a[s.TOP]!='(')
    {
        target[j]=pop();
        j++;
    }
    target[j]='\0';
}
```



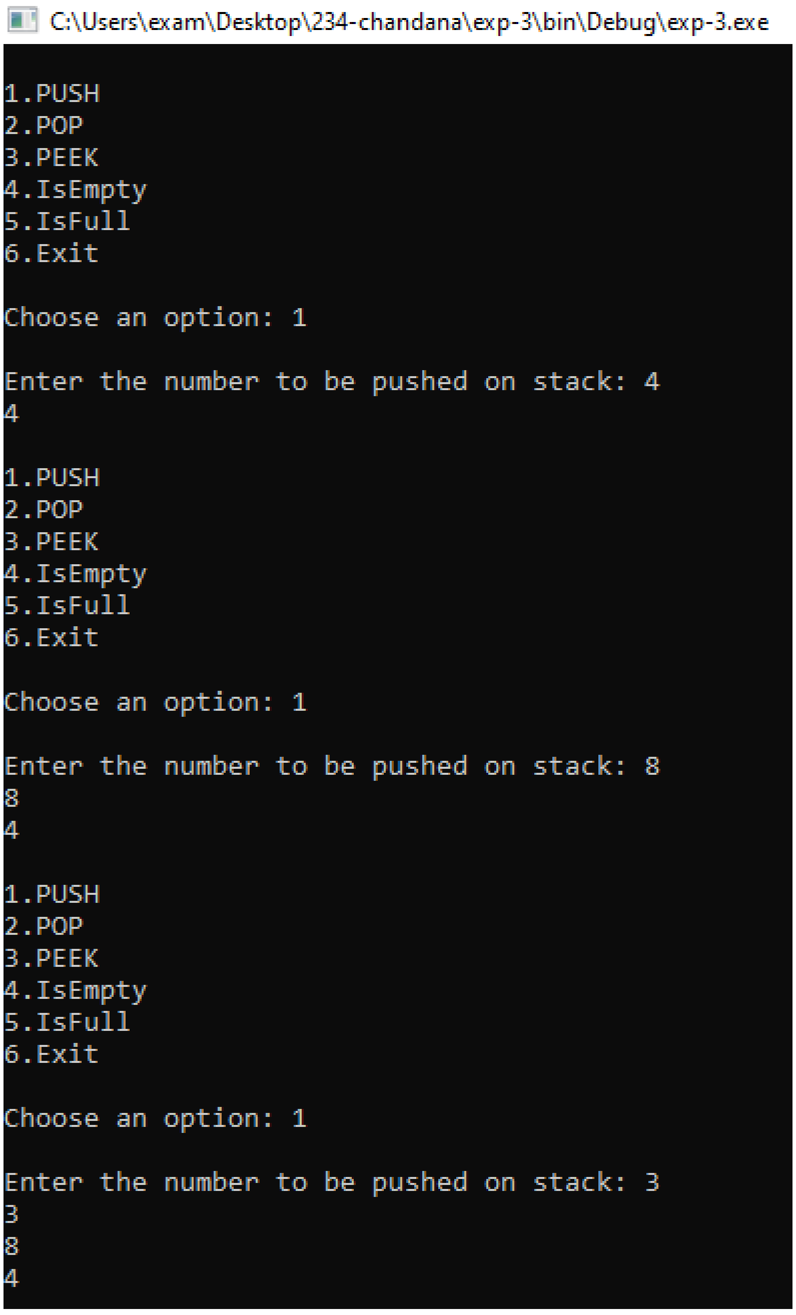
```
C:\CodeBlocks\SY\Exp-3\exp:

Enter the infix expression to be converted into postfix: A+B-C*D

The corresponding postfix expression is: AB+CD*-
Process returned 0 (0x0)   execution time : 3.691 s
Press any key to continue.
```

<u>Stack Implementation using Linked Lists</u>:

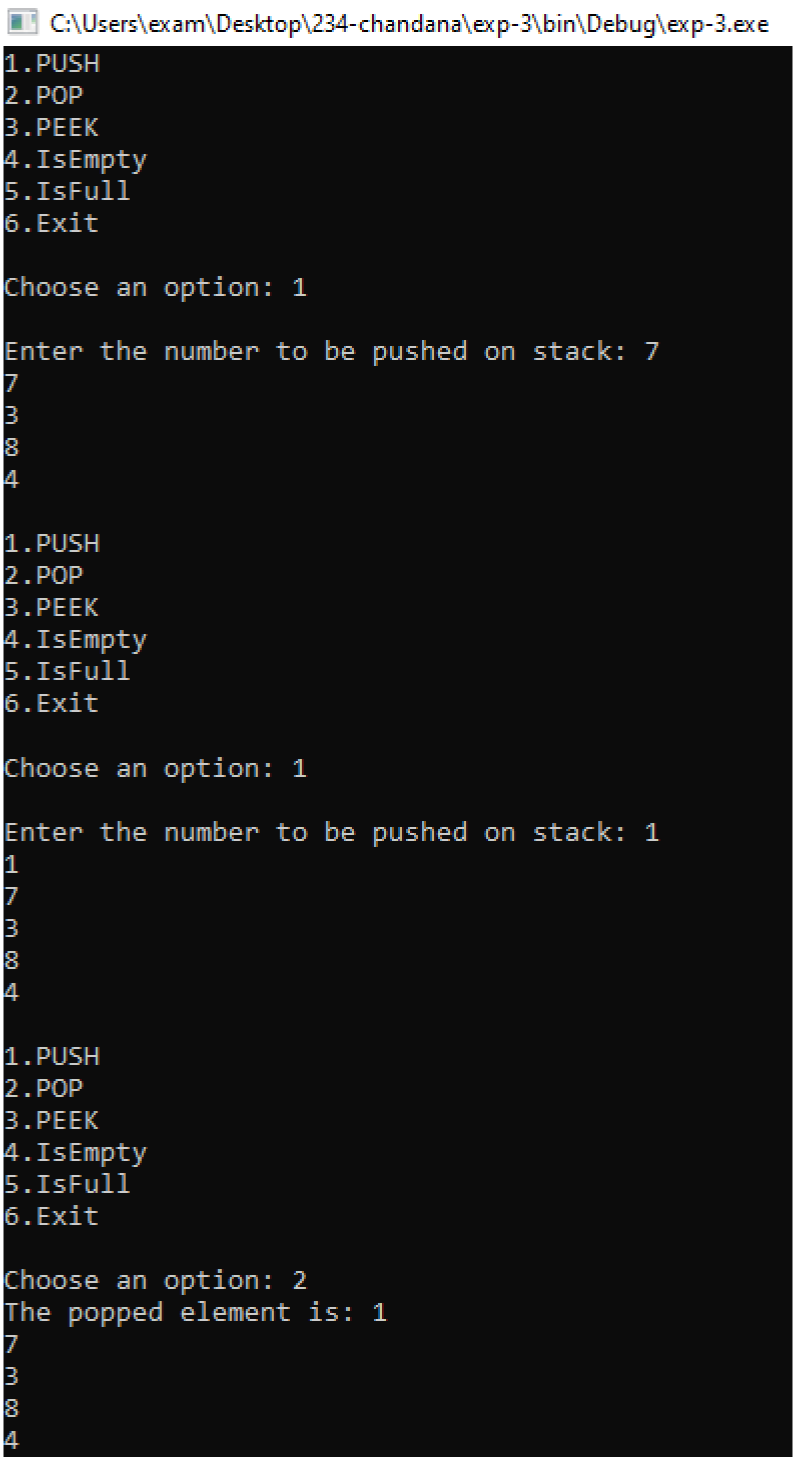
```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>
struct stack{
    int data;
    struct stack *next;
    }s;
struct stack *top = NULL;
int main()
{
    int choice, val;
    printf("\n1.PUSH\n2.POP\n3.PEEK\n4.IsEmpty\n5.IsFull\n6.Exit\n");
    printf("\nChoose an option: ");
    scanf("%d",&choice);
    switch(choice)
    {
    case 1:
        printf("\nEnter the number to be pushed on stack: ");
        scanf("%d", &val);
        top = push(top, val);
        display(top);
        main();
        break;
    case 2:
        pop();
        main();
        break;
    case 3:
        peek();
        main();
        break;
    case 4:
        if(isempty())
            printf("Stack is empty.\n");
        else
            printf("Stack is not empty.\n");
        main();
        break;
    case 5:
        printf("A stack that is dynamically implemented using linked lists can never be full.\n");
        main();
        break;
    case 6:
        break;
    default:
        printf("Choose a valid option!");
        break;
    }
    return 0;
}
struct stack *push(struct stack *top, int val)
{
    struct stack *newnode;
    newnode = (struct stack*)malloc(sizeof(struct stack));
    newnode -> data = val;
    if(isempty())
    {
        newnode -> next = NULL;
        top = newnode;
    }
    else
    {
        newnode -> next = top;
        top = newnode;
    }
```

```
   return top;
}
struct stack *display(struct stack *top)
{
   struct stack *newnode;
   newnode = top;
   if(isempty())
      printf("\n STACK IS EMPTY");
   else
   {
      while(newnode != NULL)
      {
         printf("\n %d", ptr -> data);
         newnode = newnode -> next;
      }
   }
return top;
}
struct stack *pop(struct stack *top)
{
   struct stack *ptr;
   ptr = top;
   if(isempty())
      {
         printf("Stack underflow.\n");
      }
   else
      {
         top = top->next;
         printf("The popped element is: %d\n", ptr -> data);
         free(ptr);
      }
return top;
}
int peek()
{
   if(isempty())
   {
      printf("Stack is empty.\n");
   }
   else
   {
      printf("Top element: %d\n",top -> data);
   }
}
int isempty(struct stack *top)
{
   if(top==NULL)
   {
      return 1;
```

```
    }
    else
    {
        return 0;
    }
}
```

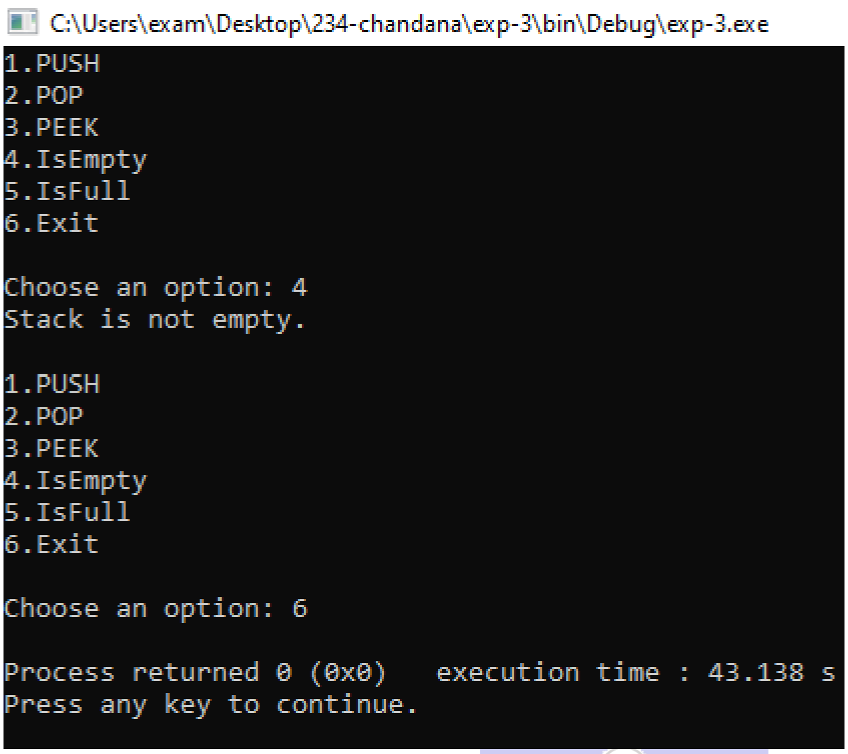


```
C:\Users\exam\Desktop\234-chandana\exp-3\bin\Debug\exp-3.exe

1.PUSH
2.POP
3.PEEK
4.IsEmpty
5.IsFull
6.Exit

Choose an option: 1

Enter the number to be pushed on stack: 4
4

1.PUSH
2.POP
3.PEEK
4.IsEmpty
5.IsFull
6.Exit

Choose an option: 1

Enter the number to be pushed on stack: 8
8
4

1.PUSH
2.POP
3.PEEK
4.IsEmpty
5.IsFull
6.Exit

Choose an option: 1

Enter the number to be pushed on stack: 3
3
8
4
```



C:\Users\exam\Desktop\234-chandana\exp-3\bin\Debug\exp-3.exe

```
1.PUSH
2.POP
3.PEEK
4.IsEmpty
5.IsFull
6.Exit

Choose an option: 1

Enter the number to be pushed on stack: 7
7
3
8
4

1.PUSH
2.POP
3.PEEK
4.IsEmpty
5.IsFull
6.Exit

Choose an option: 1

Enter the number to be pushed on stack: 1
1
7
3
8
4

1.PUSH
2.POP
3.PEEK
4.IsEmpty
5.IsFull
6.Exit

Choose an option: 2
The popped element is: 1
7
3
8
4
```

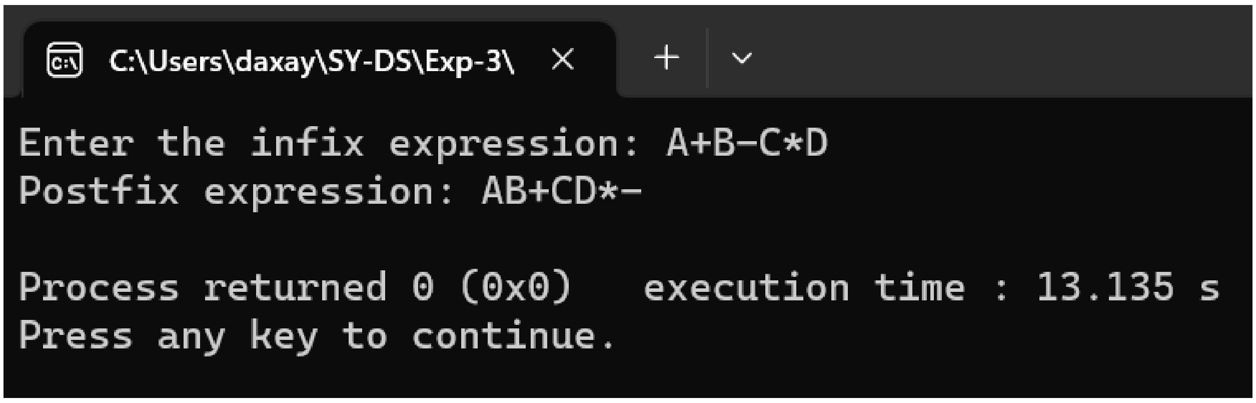


```
C:\Users\exam\Desktop\234-chandana\exp-3\bin\Debug\exp-3.exe
1.PUSH
2.POP
3.PEEK
4.IsEmpty
5.IsFull
6.Exit

Choose an option: 4
Stack is not empty.

1.PUSH
2.POP
3.PEEK
4.IsEmpty
5.IsFull
6.Exit

Choose an option: 6

Process returned 0 (0x0)   execution time : 43.138 s
Press any key to continue.
```

<u>Infix to Postfix</u>:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>
struct Node
{
   char data;
   struct Node* next;
};
struct Node* createNode(char data)
{
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}
int isEmpty(struct Node* top)
{
   return top == NULL;
}
void push(struct Node** top, char data)
{
   struct Node* newNode = createNode(data);
   newNode->next = *top;
   *top = newNode;
}
```

```
char pop(struct Node** top)
{
   if (isEmpty(*top))
   {
      printf("Stack underflow!\n");
      exit(EXIT_FAILURE);
   }
   struct Node* temp = *top;
   char data = temp->data;
   *top = (*top)->next;
   free(temp);
   return data;
}
int getPrecedence(char op)
{
   if (op == '+' || op == '-')
      return 1;
   else if (op == '*' || op == '/')
      return 2;
   else if (op == '^')
      return 3;
   else
      return 0;
}
void infixToPostfix(char* infix)
{
   struct Node* stack = NULL;
   char postfix[100];
   int i, j;
   i = j = 0;

   while (infix[i] != '\0')
   {
      if (isalpha(infix[i]))
      {
         postfix[j++] = infix[i];
      }
      else if (infix[i] == '(')
      {
         push(&stack, infix[i]);
      }
      else if (infix[i] == ')')
      {
         while (!isEmpty(stack) && stack->data != '(')
         {
            postfix[j++] = pop(&stack);
         }
         if (!isEmpty(stack) && stack->data != '(')
         {
            printf("Invalid infix expression!\n");
            return;
          }
          else
          {
            pop(&stack);
          }
        }
        else
        {
          while (!isEmpty(stack) && getPrecedence(infix[i]) <= getPrecedence(stack->data))
          {
            postfix[j++] = pop(&stack);
          }
          push(&stack, infix[i]);
        }
        i++;
      }
      while (!isEmpty(stack))
      {
        postfix[j++] = pop(&stack);
      }
      postfix[j] = '\0';
      printf("Postfix expression: %s\n", postfix);
}
int main()
{
    char infix[100];
    printf("Enter the infix expression: ");
    scanf("%s", infix);
    infixToPostfix(infix);
    return 0;
}
```



```
C:\Users\daxay\SY-DS\Exp-3\

Enter the infix expression: A+B-C*D
Postfix expression: AB+CD*-

Process returned 0 (0x0)   execution time : 13.135 s
Press any key to continue.
```

Outcomes:

Apply linear and non-linear data structure in application development.

Conclusion:

The experiment demonstrated the successful implementation of a stack using a singly linked list and the correct execution of the specified operations. Additionally, the experiment showcased the effectiveness of the stack implementation in converting infix expressions to postfix form.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

- Y. Langsam, M. Augenstin and A. Tannenbaum, “Data Structures using C”, Pearson Education Asia, 1st Edition, 2002