Name: SUBHENDU MAJI Roll Number: 18ETCS002121

Laboratory 5

1. Questions

    1. Implement the PUSH, POP and PRINT operations on stack
    2. Write a C program to convert infix to postfix notation using stack
    3. Write a C program to evaluate a postfix expression.

2. Algorithm

2.1 Implement the PUSH, POP and PRINT operations on stack

```
step 1: start

step 2: declare global variable top = -1, and stack[max]

step 3: make a push function
      3.1 if top= max-1: print stack is full
      3.2 else:  top++ and stack[top]= item
step 4: make a pop function

      4.1 if top=-1: print stack is empty

      4.2 else: top--;

step 5: make display function to print the stack

      5.1 if top = -1: print stack is empty

      5.2 else: for (i=top; i>=0; i--): print stack[i]

step 6: call each of the function in main body

step 7: stop
```

2.2 C program to convert infix to postfix notation using stack

```
step 1: start

step 2: declare global variable top = -1, s[50]

step 3: make a stack push function

step 4: make a stack pop function

step 5: make a priority function

      5.1 if element = '(': return 0;

      5.2 if element = '+' or '-': return 1;

      5.3 if element = '*' or '/': return 2;

step 6: in main body:

      6.1 input the infix expression

step 7: iterate over each character

      7.1 if a character is alphanumeric: print it as it is

      7.2 if it is '(': push it to stack

      7.3 if it is ')': pop and print till we get '('

      7.4 else:  while (priority(s[top]) >= priority(*e))
                        { pop from the stack and print it }
                 Push the element to stack

step 8: print the last element remaining in stack

      8.1 while top! = -1: pop and print the elements

Step 9: stop
```

2.3 C program to evaluate a postfix expression.

step 1: start

step 2: declare global variable top = -1, s[50]

step 3: make a stack push function

step 4: make a stack pop function

step 5: make a print function to print stack

step 6: in main body:

6.1 iterate over each character

6.2 if the character is alphanumeric: push it to stack (after type casting it to integer (eg: 'a'-'0' = 97)

6.3 else:

6.3.1 pop last 2 element

6.3.2 according to the operator(element): do the operation and push it to stack

step 7: call print function

step 8: stop

3. Program

```c
// stack operations
#include <stdio.h>
#include <stdlib.h>

#define MAX 10

int STACK[MAX], TOP;
void display(int[]);
void PUSH(int[], int);
void POP(int[]);

void main()
{
    int ITEM = 0;
    int choice = 0;
    TOP = -1;

    while (1)
    {

        printf("Enter Choice (1: display, 2: PUSH, 3: POP, 4: Exit..:");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            display(STACK);
            break;
        case 2:
            printf("Enter Item to be insert :");
            scanf("%d", &ITEM);
            PUSH(STACK, ITEM);
            break;
        case 3:
            POP(STACK);
            break;
        case 4:
            exit(0);
        default:
            printf("\nInvalid choice.");
            break;
        }
    }
}

void display(int stack[])
{
    int i = 0;
    if (TOP == -1)
    {
        printf("Stack is Empty .\n");
        return;
    }

    printf("%d ", stack[TOP]);
    for (i = TOP - 1; i >= 0; i--)
    {
        printf("\n%d", stack[i]);
    }
    printf("\n\n");
}

void PUSH(int stack[], int item)
{
    if (TOP == MAX - 1)
    {
        printf("\nSTACK is FULL CAN't ADD ITEM\n");
        return;
    }
    TOP++;
    stack[TOP] = item;
}

void POP(int stack[])
{
    int deletedItem;
    if (TOP == -1)
    {
        printf("STACK is EMPTY.\n");
        return;
    }

    deletedItem = stack[TOP];
    TOP--;
    printf("%d deleted successfully\n", deletedItem);
    return;
}

```

Figure 1 Implement the PUSH, POP and PRINT operations on stack

```c
// infix to postfix using stack
#include <stdio.h>
#include <ctype.h>

char s[50];
int top = -1;
char push(char elem)
{
    top++;
    s[top] = elem;
    return s[top];
}

char pop()
{
    if (top == -1)
        return -1;
    else
        return (s[top--]);
}

int pr(char elem)
{
    if (elem == '(')
        return 0;
    if (elem == '+' || elem == '-')
        return 1;
    if (elem == '*' || elem == '/')
        return 2;
}

void main()
{
    char infx[50], *e, elem;

    printf("\n\nInfix Expression: ");
    scanf("%s", infx);
    e = infx;
    while ((*e != '\0'))
    {
        if (isalnum(*e))
        {
            printf("%c", *e);
        }
        else if (*e == '(')
        {
            push(*e);
        }

        else if (*e == ')')
        {
            while ((elem = pop()) != '(')
            {
                printf("%c", elem);
            }
        }
        else
        {
            while (pr(s[top]) >= pr(*e))
            {
                printf("%c", pop());
            }
            push(*e);
        }
        e++;
    }
    while (top != -1)
    {
        printf("%c", pop());
    }
}
```

Figure 2 C program to convert infix to postfix notation using stack

```c
// postfix evalution
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define max 10

int s[max];
int top = -1;

void push(int n)
{
  if (top > max)
  {
    printf("\nStack Overflow!!\n");
  }
  else
  {
    s[++top] = n;
  }
}

int pop()
{
  if (top < 0)
  {
    printf("\nStack Empty!!!\n");
  }
  else
  {
    int x = s[top--];
    return x;
  }
}

void print()
{
  printf("\n Evaluted No.: ");
  for (int i = top; i >= 0; i--)
  {
    printf("%d\n", s[i]);
  }
  printf("\n");
}

void main()
{
  char str[10], operator;
  int op1, op2;
  printf("Postfix Expression : ");
  gets(str);
  for (int i = 0; i < strlen(str); i++)
  {
    if (isalnum(str[i]))
    {
      push(str[i] - '0');
    }
    else
    {
      operator= str[i];
      op1 = pop();
      op2 = pop();
      switch (operator)
      {
      case '*':
        push(op1 * op2);
        break;
      case '/':
        push(op2 / op1);
        break;
      case '%':
        push(op1 % op2);
        break;
      case '+':
        push(op1 + op2);
        break;
      case '-':
        push(op2 - op1);
        break;
      default:
        break;
      }
    }
  }
  print();
}
```

Figure 3 C program to evaluate a postfix expression

4. Presentation of Results

```
PS D:\RUAS\sem 03\DSA lab\programs> cd "d:\RUAS\sem 03\DSA lab\programs\" ; if ($?) { gcc stack.c -o stack } ; if ($?) { .\stack }
Enter Choice (1: display, 2: PUSH, 3: POP, 4: Exit..:1
Stack is Empty .
Enter Choice (1: display, 2: PUSH, 3: POP, 4: Exit..:2
Enter Item to be insert :74
Enter Choice (1: display, 2: PUSH, 3: POP, 4: Exit..:2
Enter Item to be insert :15
Enter Choice (1: display, 2: PUSH, 3: POP, 4: Exit..:2
Enter Item to be insert :59
Enter Choice (1: display, 2: PUSH, 3: POP, 4: Exit..:3
59 deleted successfully
Enter Choice (1: display, 2: PUSH, 3: POP, 4: Exit..:1
15
74

Enter Choice (1: display, 2: PUSH, 3: POP, 4: Exit..:
```

Figure 4 output of implementing the PUSH, POP and PRINT operations on stack

```
PS D:\RUAS\sem 03\DSA lab\programs> cd "d:\RUAS\sem 03\DSA lab\programs\" ; if ($?) { gcc infix_to_postfix.c -o infix_to_postfix }
stfix }


Infix Expression: (a+b)*(c+d)
ab+cd+*
PS D:\RUAS\sem 03\DSA lab\programs>
```

Figure 5 output of C program to convert infix to postfix notation using stack

```
PS D:\RUAS\sem 03\DSA lab\programs> cd "d:\RUAS\sem 03\DSA lab\programs\" ; if ($?) { gcc postfixEval.c -o postfixEval } ;
Postfix Expression : 234*+2+

 Evaluted No.: 16
```

Figure 6 output of C program to evaluate a postfix expression

5. Conclusions

Learning happened:

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be **LIFO** (Last in First Out) or **FILO** (First in Last Out).

Mainly the following three basic operations are performed in the stack:

- **Push:** Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
- **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.

Infix Expression	Prefix Expression	Postfix Expression
A + B * C + D	+ + A * B C D	A B C * + D +
(A + B) * (C + D)	* + A B + C D	A B + C D + *
A * B + C * D	+ * A B * C D	A B * C D * +

Hence, we can see the programs are compiled successfully without any error.