lab-3

Date
Page

1) Implementation of ~~array~~ stack using array to show push, pop and display operations.

```
#include<stdio.h>
#include<stdlib.h>
# define size 5
void push(int);
void pop();
void display();
int stack[size], top=-1;
void main(){
int op,n;
    printf("Enter the operation\n 1.push\n
        2.pop\n 3.display 4.-1 to stop\n");
    scanf("%d", &op);
    while(1){
        if(op==-1){
            printf("operation completed
                \n");
            break;
        }
        else{
        switch(op){
            case 1: printf("Enter the
                value\n");
                scanf("%d",&n);
                push(n);
                break;
            case 2: pop();
                break;
```

```
                case 3: display();
                        break();
                default: printf("wrong
                         choice\n");
                }
            }
        }
    }
void push(int n){
    if(top==size-1){
        printf("stack overflow condition
                \n");
    }
    else{
        top++;
        stack[top]=n;
        printf("PUSH() operation is
                successfull \n");
    }
}
void pop(){
    if(top==-1){
        printf("stack underflow
                condition\n");
    }
    else{
    printf("%d is POP()ed successfully
                stack[top]);
    top--;
    }
}
```

Date
Page

```
void display(){
    if(top==-1){
        printf("stack is empty\n");
    }
    else{
        for(int i=top; i>=0; i--){
            printf("%d\t", stack[i]);
        }
    }
}
```

Smita
NLP
1/1/2024.

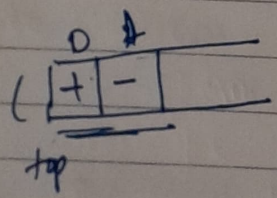
2) Program to convert infix to postfix expression using stack.

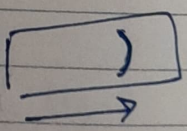
```
#include <stdio.h>
#include < string.h>
int index1=0, pos=0, top=-1
char symbol, temp, infix[20], postfix[20],
                    stack[20]
void push(char);
char pop();
void infixtopostfix();
int pred(char);

void main(){
    printf("Enter the infix expression\n");
    scanf("%s", infix);
    infixtopostfix();
    printf("Infix expression = %s\n", infix);
    printf("Postfix expression = %s\n", postfix);
}

void infixtopostfix(){
    symbol = infix
    length = strlen(infix);
    push('#');
    while(index1 < length){
        symbol = infix[index1];
        switch(symbol);
            case '(': push(symbol);
                      break;
            case ')': temp = pop();
                      while(temp != '('){
                      postfix[pos] = temp;
                      pos++;
                      temp = pop();}
```

```
                break;
            case '+':
            case '-':
            case '*':
            case '/':
            case '^': while( pred[stack[top]]
                        >= pred(symbol))
                    {
                        temp=pop();
                        postfix[pos]=temp;
                        pos++;
                    }
                    push(symbol);
                    break;
            default: push
                    postfix[pos]=symbol;
                    pos++;
                    break;
        }
    }


void push(char symbol){
    top++;
    stack[top]=symbol;
}

char pop(){
    temp=stack[top];
    top--;
    return temp;
}
```

```
int pred(char symbol){
    int p;
    switch(symbol){
        case '^': p=3;
                  break;
        case '*':
        case '/': p=2;
                  break;
        case '+':
        case '-': p=1;
                  break;
        case '(': p=0;
                  break;
        case '#': p=-1
                  break;
    }
    return p;
```

Entered 3 1/1/2024.





A+B*(C+D)/F+D*E)

Stack

(

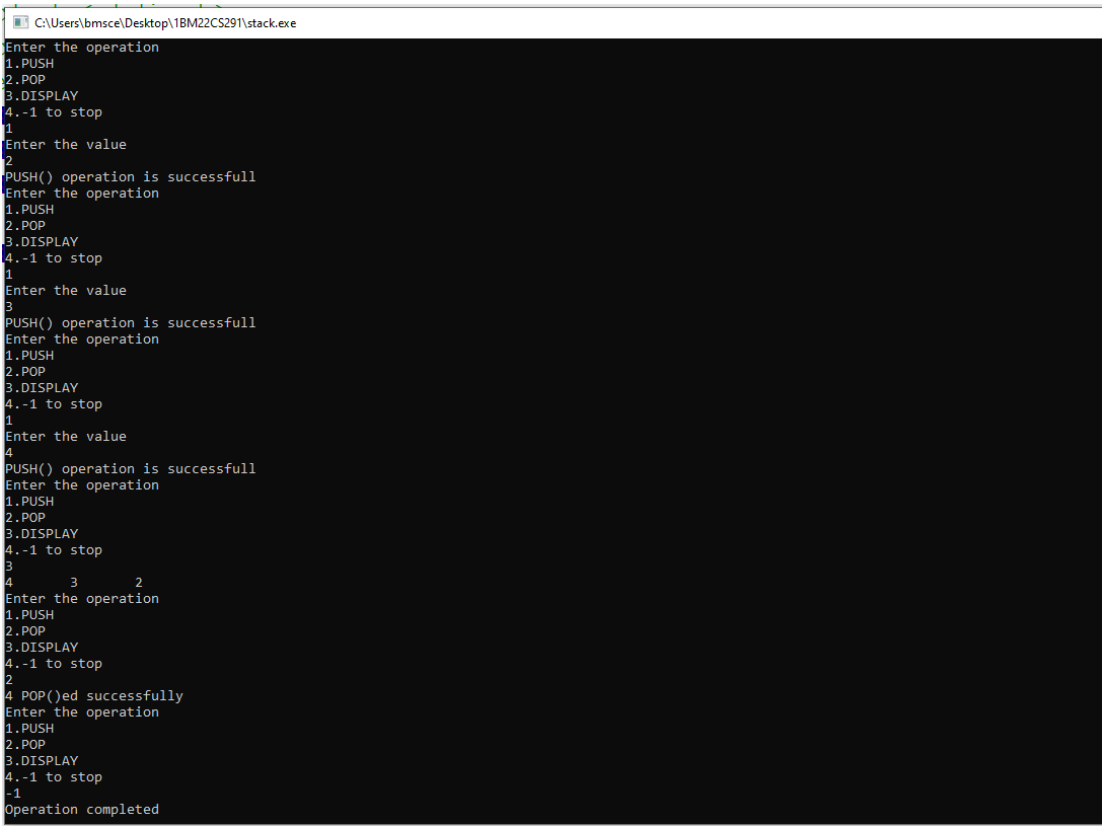
(+*(+)/

(+*/(+

ABCD+*F/+DE

*

Stack implementation using array output:



Infix to postfix using stack output: