**WEEK 3 – STACKS**

Write a menu driven C program to implement stack using one dimensional array. Perform the operations on the stack (i) Push (ii) Pop(iii) Peek (ii) Is empty (iv) Isfull (v) Display.

**PROGRAM**

#include<stdio.h>

int stack[5];

int top=-1;

void push(int x)

{

    if(top==4)

    printf("\nStack Full!!!\nFurther elements cannot be pushed!!!\n");

    else

    {

        top++;

        stack[top]=x;

        printf("\nPush successful\n");

    }

}

void pop()

{

    if(top==-1)

    {

        printf("\nUnderflow !! Stack Empty\n");

    }

    else

    {

        printf("Poped Element: %d ",stack[top]);

        top--;

    }

}

void peek()

{

    if(top==-1)

    {

    printf("\nStack Empty!!\n");

    }

    else

    printf("Element at the top: %d",stack[top]);

}

void isempty()

{

    if(top==-1)

    printf("\nThe Stack is empty\n");

    else

    {

        printf("\nStack is not empty\n");

    }

}

void isfull()

{

    if(top==4)

    printf("\nThe stack is Full !!\n");

    else

    {

        printf("\nThe Stack is not full !!!\n");

    }

}

void display()

{

    int i;

    if(top==-1)

    {

        printf("\nUnderflow !! Stack Empty\n");

    }

    else

    {

        printf("Elements of the stack are:\n");

        for(i=top;i>-1;i--)

        printf("%d \n",stack[i]);

        printf("\n");

    }

}

void main()

{

    int resp,x;

    printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|||| STACK MENU ||||\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

    do

    {

    printf("\n1. Push\n2. Pop\n3.Peek\n4. Isempty\n5. Is Full\n6. Display the Stack\n0. Exit\n\nYour Choice:");

    scanf("%d",&resp);

    switch (resp)

    {

    case 1:

        printf("Enter the Element to be pushed:");

        scanf("%d",&x);

        push(x);

        break;

    case 2:

        pop();

        break;

    case 3:

        peek();

        break;

    case 4:

        isempty();

        break;

    case 5:

        isfull();

        break;

    case 6:

        display();

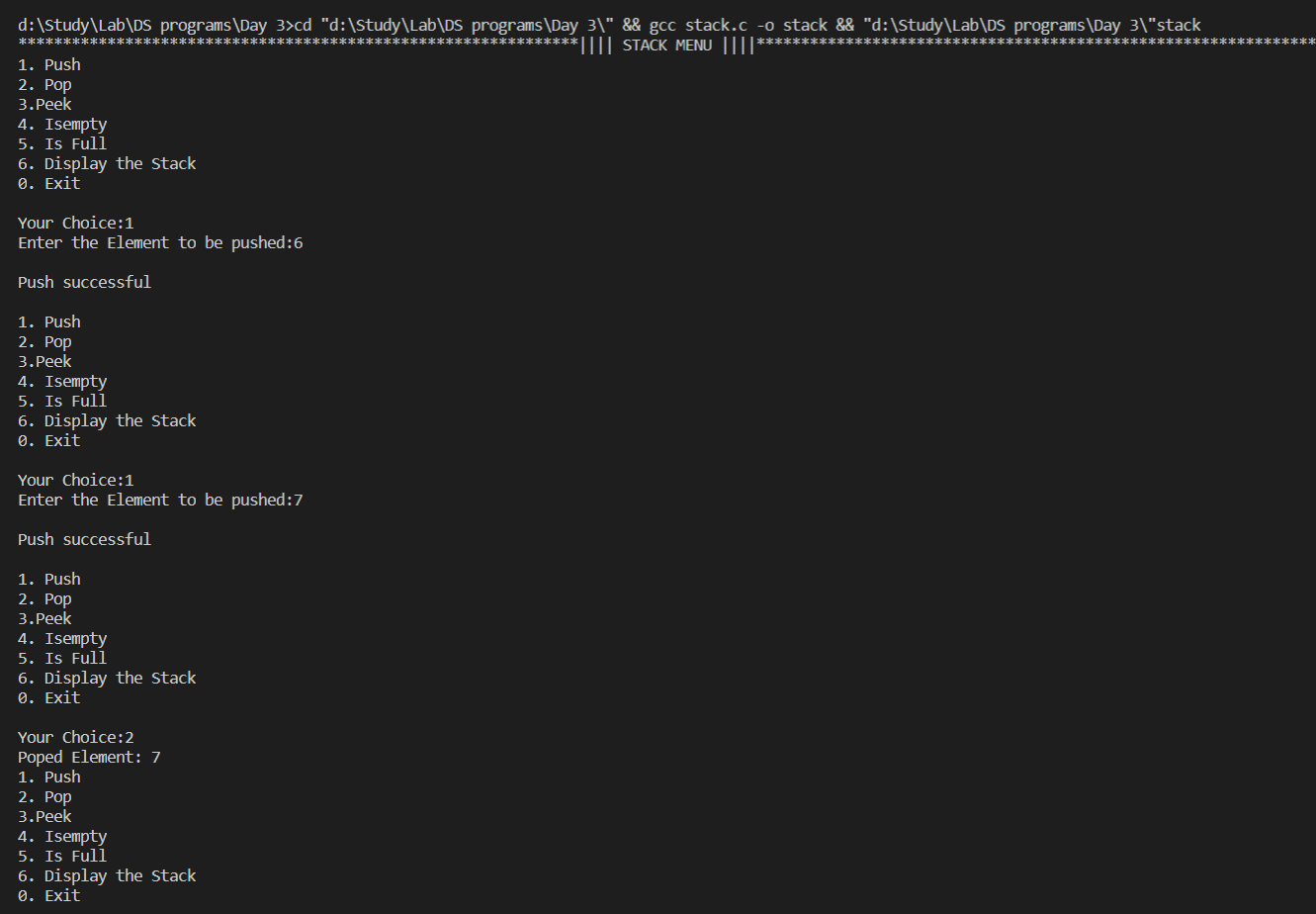
    default:

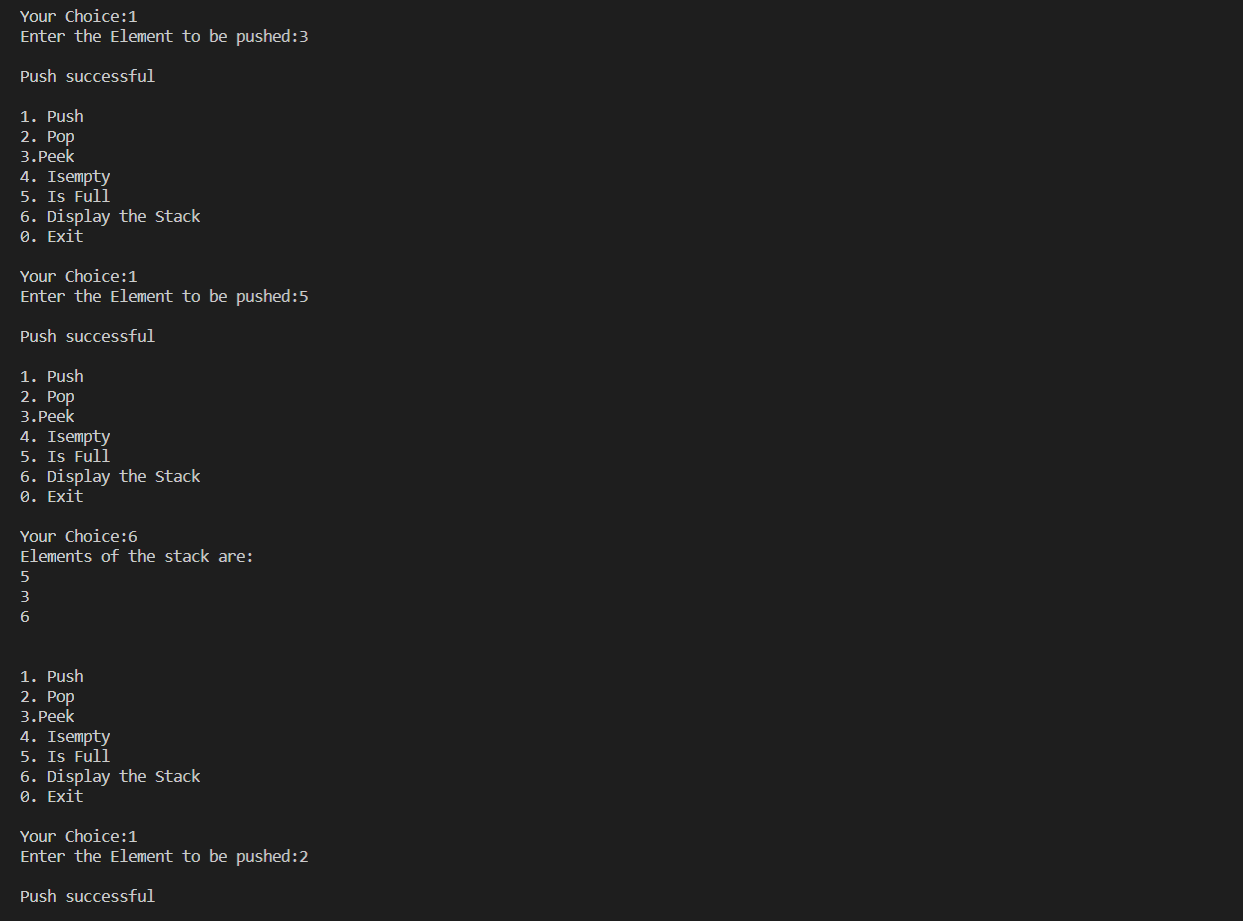
        break;

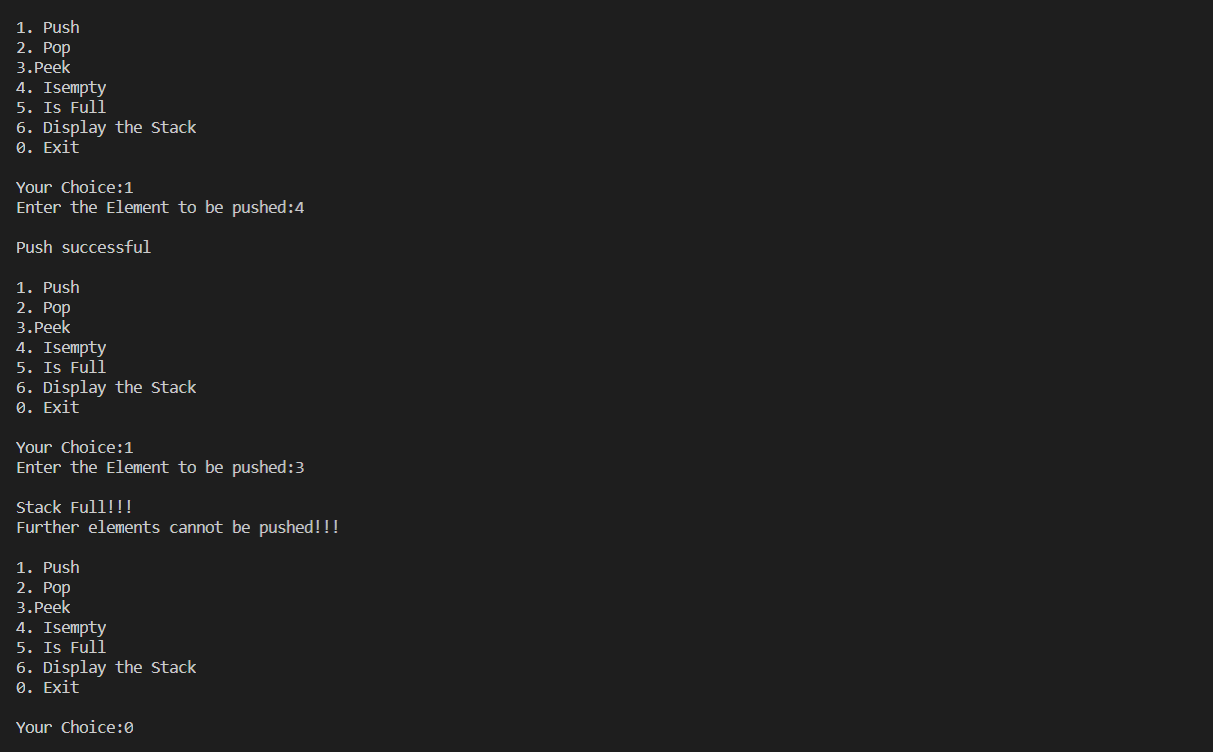
    }

    }while(resp!=0);

}

**OUTPUT**

****

****

Write a menu driven C program to covert infix to postfix expression and evaluate it using stack.

**PROGRAM**

/\*

    Jans Johnson

    S3 cse Beta

    rollno:24

    U1903096

\*/

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

#define SIZE 100

char stack[SIZE];

int top = -1;

void push(char item)

{

    if(top >= SIZE-1)

    {

        printf("\nStack Overflow.");

    }

    else

    {

        top = top+1;

        stack[top] = item;

    }

}

char pop()

{

    char item ;

    if(top <0)

    {

        printf("stack under flow: invalid infix expression");

        getchar();

        exit(1);

    }

    else

    {

        item = stack[top];

        top = top-1;

        return(item);

    }

}

int is\_operator(char symbol)

{

    if(symbol == '^' || symbol == '\*' || symbol == '/' || symbol == '+' || symbol =='-')

    {

        return 1;

    }

    else

    {

    return 0;

    }

}

int precedence(char symbol)

{

    if(symbol == '^')

    {

        return(3);

    }

    else if(symbol == '\*' || symbol == '/')

    {

        return(2);

    }

    else if(symbol == '+' || symbol == '-')

    {

        return(1);

    }

    else

    {

        return(0);

    }

}

void InfixToPostfix(char infix\_exp[], char postfix\_exp[])

{

    int i, j;

    char item;

    char x;

    push('(');

    strcat(infix\_exp,")");

    i=0;

    j=0;

    item=infix\_exp[i];

    while(item != '\0')        /\* run loop till end of infix expression \*/

    {

        if(item == '(')

        {

            push(item);

        }

        else if( isdigit(item) || isalpha(item))

        {

            postfix\_exp[j] = item;              /\* add operand symbol to postfix expr \*/

            j++;

        }

        else if(is\_operator(item) == 1)        /\* means symbol is operator \*/

        {

            x=pop();

            while(is\_operator(x) == 1 && precedence(x)>= precedence(item))

            {

                postfix\_exp[j] = x;                  /\* so pop all higher precendence operator and \*/

                j++;

                x = pop();                       /\* add them to postfix expresion \*/

            }

            push(x);

            /\* because just above while loop will terminate we have

            oppped one extra item

            for which condition fails and loop terminates, so that one\*/

            push(item);                 /\* push current oprerator symbol onto stack \*/

        }

        else if(item == ')')         /\* if current symbol is ')' then \*/

        {

            x = pop();                   /\* pop and keep popping until \*/

            while(x != '(')                /\* '(' encounterd \*/

            {

                postfix\_exp[j] = x;

                j++;

                x = pop();

            }

        }

        else

        { /\* if current symbol is neither operand not '(' nor ')' and nor

            operator \*/

            printf("\nInvalid infix Expression.\n");        /\* the it is illegeal  symbol \*/

            getchar();

            exit(1);

        }

        i++;

        item = infix\_exp[i]; /\* go to next symbol of infix expression \*/

    }

    if(top>0)

    {

        printf("\nInvalid infix Expression.\n");        /\* the it is illegeal  symbol \*/

        getchar();

        exit(1);

    }

    if(top>0)

    {

        printf("\nInvalid infix Expression.\n");        /\* the it is illegeal  symbol \*/

        getchar();

        exit(1);

    }

    postfix\_exp[j] = '\0'; /\* add sentinel else puts() fucntion \*/

    /\* will print entire postfix[] array upto SIZE \*/

}

int stackeval[20];

int topeval = -1;

void pusheval(int x)

{

    stackeval[++topeval] = x;

}

int popeval()

{

    return stackeval[topeval--];

}

void evaluate()

{

    int n1,n2,n3,num;

    int i=0;

    char postfix\_exp[SIZE];

    printf("\nEnter Postfix expression : ");

    scanf("%s",postfix\_exp);

    while(postfix\_exp[i] != '\0')

    {

        if(isalpha(postfix\_exp[i]))

        {

            printf("Enter value for %c:",postfix\_exp[i]);

            scanf("%d",&num);

            pusheval(num);

        }

        else

        {

            n1 = popeval();

            n2 = popeval();

            switch(postfix\_exp[i])

            {

            case '+':

            {

                n3 = n1 + n2;

                break;

            }

            case '-':

            {

                n3 = n2 - n1;

                break;

            }

            case '\*':

            {

                n3 = n1 \* n2;

                break;

            }

            case '/':

            {

                n3 = n2 / n1;

                break;

            }

            }

            pusheval(n3);

        }

        i++;

    }

    printf("\nThe result of expression %s  =  %d\n\n",postfix\_exp,popeval());

}

/\* main function begins \*/

int main()

{

    char infix[SIZE], postfix[SIZE];

    int choice;

    printf("\n\n\n1. Convert infix to postfix\n2. Evaluate postfix expression\n\nYour choice:");

    scanf("%d",&choice);

    if(choice==1)

    {

        printf("\nEnter Infix expression : ");

        gets(infix);

        InfixToPostfix(infix,postfix);                   /\* call to convert \*/

        printf("Postfix Expression: ");

        puts(postfix);                     /\* print postfix expression \*/

    }

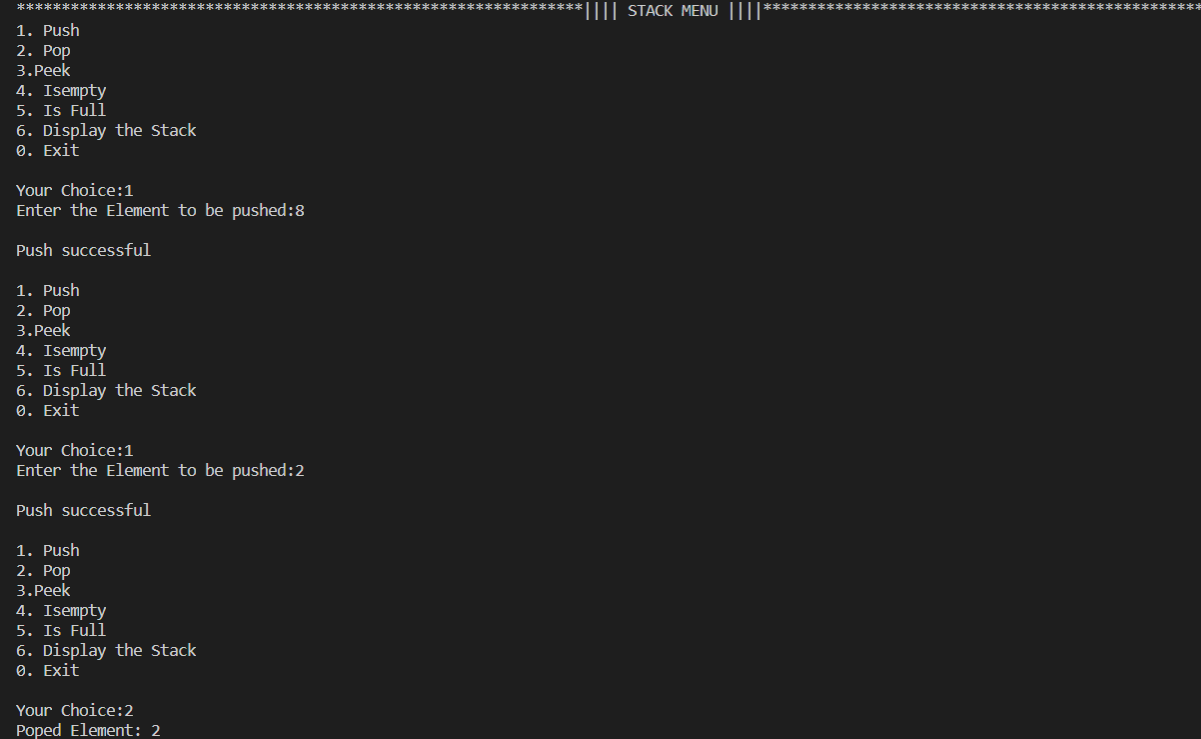
    else if(choice==2)

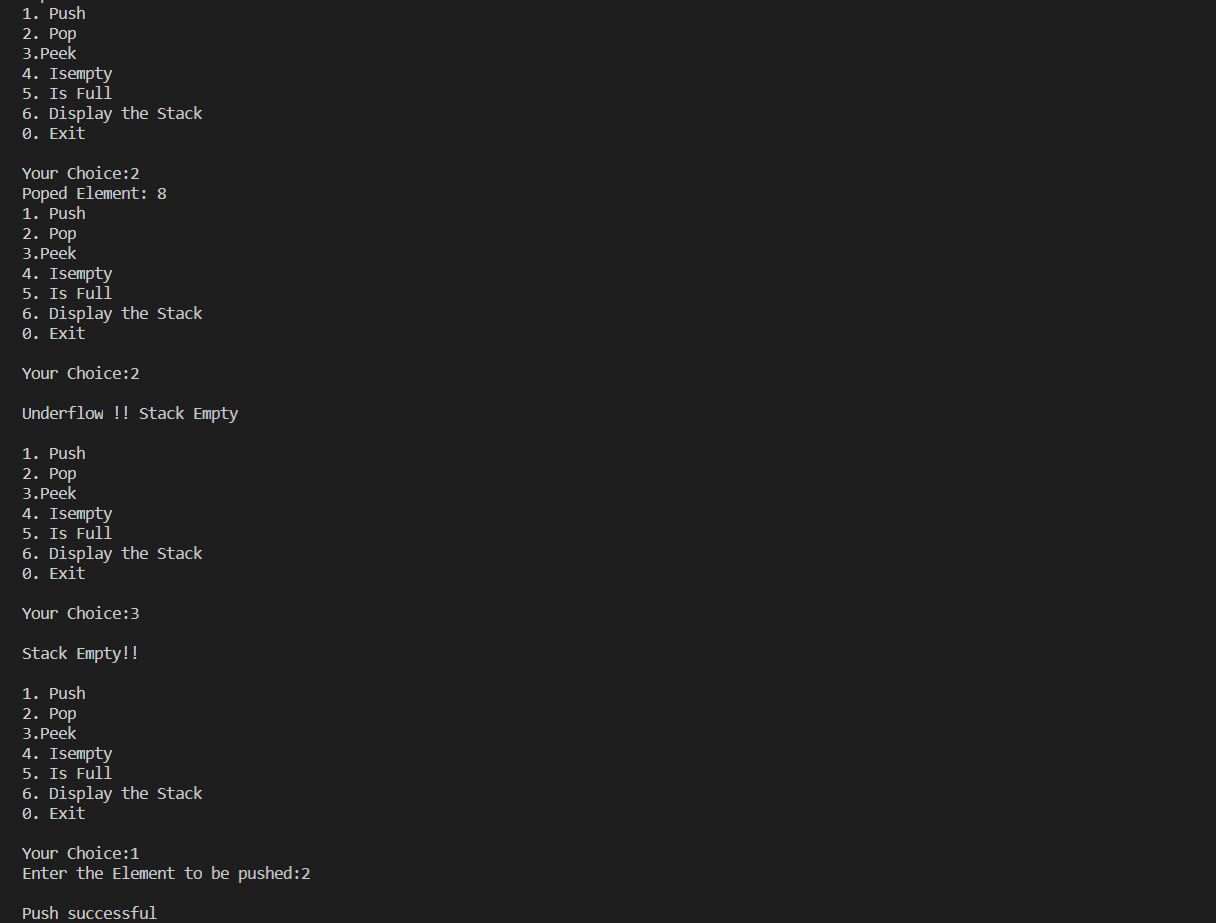
    evaluate();

    return 0;

}

**OUTPUT**

****

****

****

****