LAB-4

4. A. Implement stack using Linked Lists.

B. Implement the Infix expression to Postfix and

Prefix conversion using Stack.

1. STACK USING LL

CODE :

#include <stdio.h>

#include <stdlib.h>

*struct* Node

{

*int* data;

*struct* Node \*next;

};

*struct* Node \*top = NULL;

*int* isEmpty(*struct* Node \**top*)

{

    if (*top* == NULL)

    {

        return 1;

    }

    return 0;

}

*int* isFull(*struct* Node \**top*)

{

*struct* Node \*p = (*struct* Node \*)malloc(sizeof(*struct* Node));

    if (p == NULL)

    {

        return 1;

    }

    return 0;

}

*struct* Node \*push(*struct* Node \**top*, *int* *value*)

{

    if (isFull(*top*))

    {

        printf("Stack Overflow\n");

    }

    else

    {

*struct* Node \*n = (*struct* Node \*)malloc(sizeof(*struct* Node));

        n->data = *value*;

        n->next = *top*;

*top* = n;

        return *top*;

    }

}

*int* pop()

{

    if (isEmpty(top))

    {

        printf("Stack Underflow\n");

    }

    else

    {

*struct* Node \*n = top;

*int* x = top->data;

        top = top->next;

        free(n);

        return x;

    }

}

*int* stackTop()

{

    return top->data;

}

*int* stackBottom(*struct* Node \**top*)

{

*struct* Node \*ptr = *top*;

    while (ptr->next != NULL)

    {

        ptr = ptr->next;

    }

    return ptr->data;

}

*void* StackDisplay(*struct* Node \**ptr*)

{

*struct* Node \*temp = *ptr*;

    while (temp != NULL)

    {

        printf("%d -> ", temp->data);

        temp = temp->next;

    }

    printf("NULL\n\n");

}

*int* main()

{

*int* choice, no;

    while (1)

    {

        printf("ENTER THE CHOICE : \n 1.Push\n 2.Pop\n 3.Display\n 4.StackTop\n 5.StackBottom\n 6.IsEmpty\n 7.IsFull\n 8.Exit");

        printf("\nPut here : ");

        scanf("%d", &choice);

        printf("\n");

        switch (choice)

        {

        case 1:

            printf("Enter the value to push : ");

            scanf("%d", &*no*);

            top = push(top, no);

            break;

        case 2:

            printf("The popped item is : %d\n", pop());

            break;

        case 3:

            StackDisplay(top);

            break;

        case 4:

            printf("StackTop is : %d\n", stackTop());

            break;

        case 5:

            printf("StackBottom is : %d\n", stackBottom(top));

            break;

        case 6:

            if (isEmpty(top))

            {

                printf(" Yes Stack Empty\n");

            }

            else

            {

                printf("Not Empty\n");

            }

            break;

        case 7:

            if (isFull(top))

            {

                printf("Yes Stack Full\n");

            }

            else

            {

                printf("Not Full\n");

            }

            break;

        case 8:

            exit(0);

            break;

        default:

            break;

        }

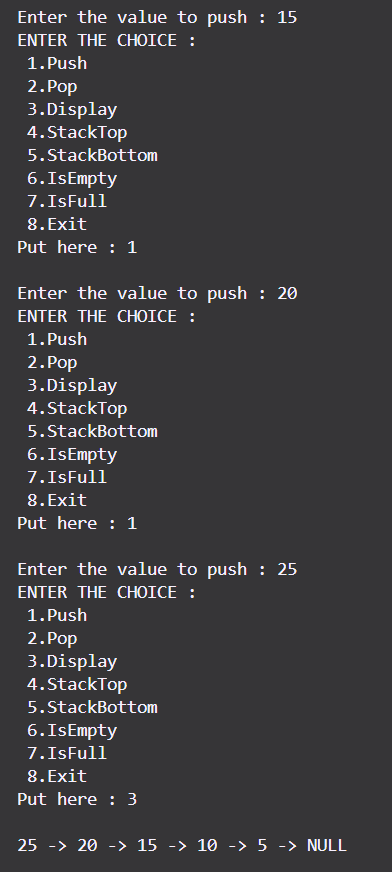
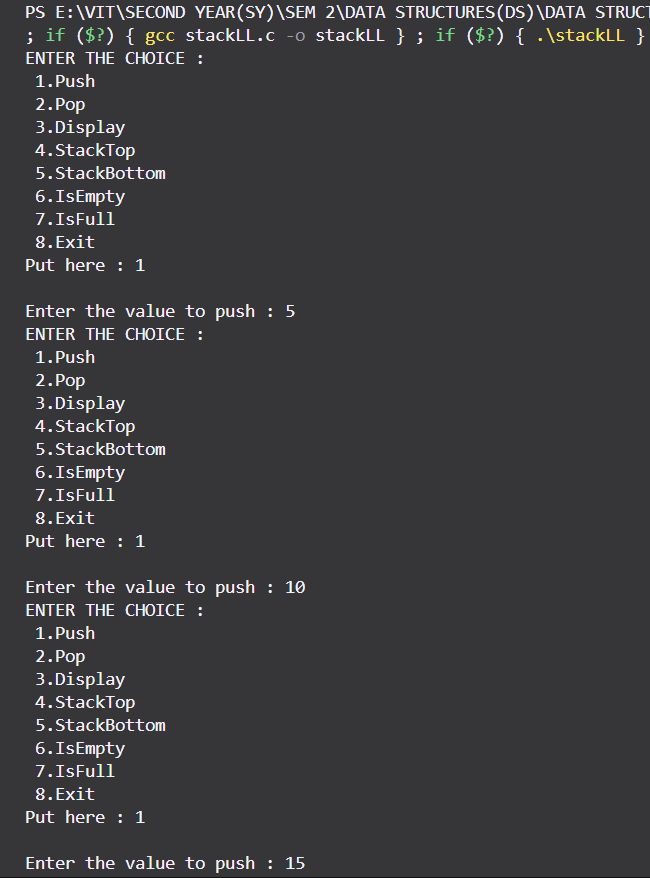
    }

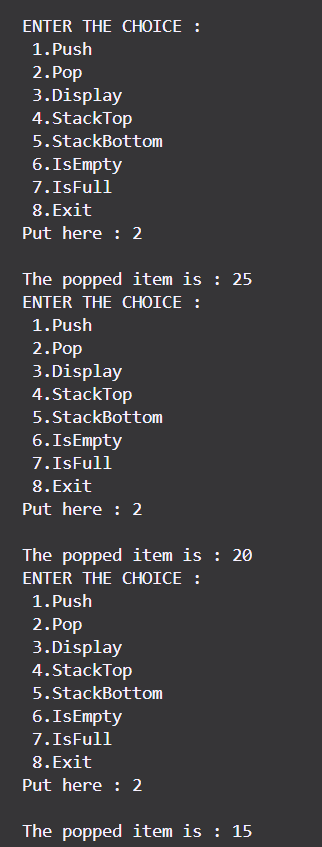
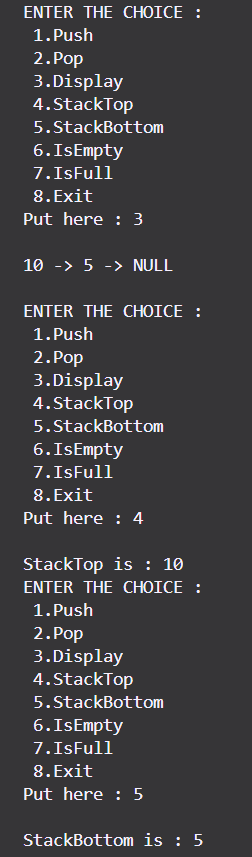
    return 0;

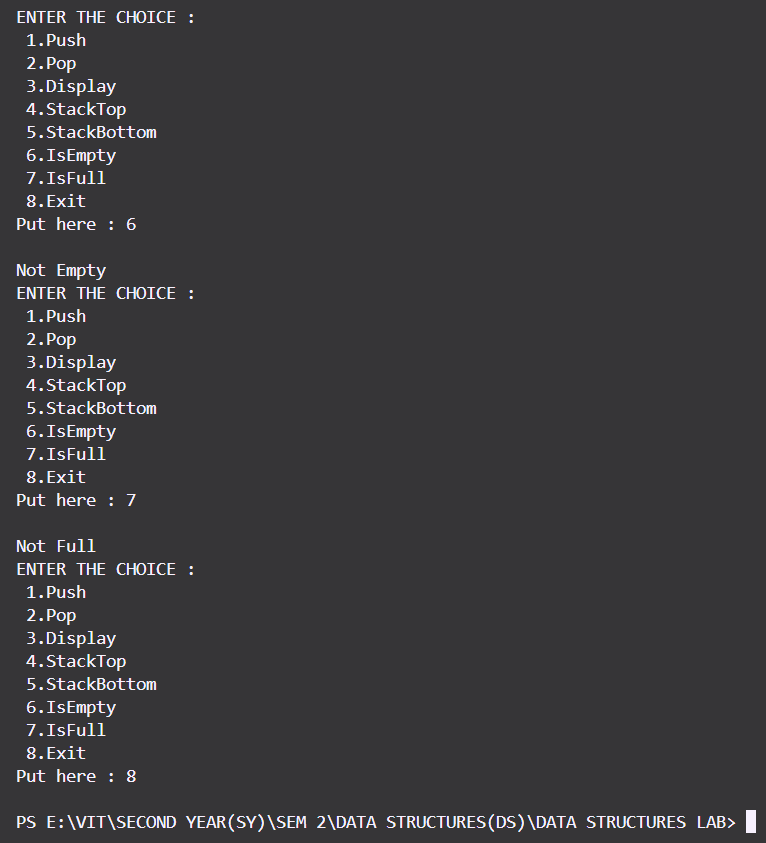
}

OUTPUT :

1. 2.







1. INFIX TO POSTFIX :

#include <stdio.h>

#include <ctype.h>

#define MAX 50

*char* stack[50];

*int* top = -1;

*void* push(*char* *elem*)

{

    if (top == MAX - 1)

    {

        printf("\nSTACK OVERFLOW\n");

    }

    else

    {

        stack[++top] = *elem*;

    }

}

*char* pop()

{

    if (top == MAX - 1)

    {

        printf("\nSTACK OVERFLOW\n");

    }

    else

    {

        return (stack[top--]);

    }

}

*int* pr(*char* *symbol*)

{

    if (*symbol* == '^')

    {

        return (3);

    }

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

    {

        return (2);

    }

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

    {

        return (1);

    }

    else

    {

        return (0);

    }

}

*int* main()

{

*char* infix[50], postfix[50], ch, elem;

*int* i = 0, k = 0;

    printf("Enter Your Infix Expression : ");

    scanf("%s", infix);

    push('#');

    while ((ch = infix[i++]) != '\0')

    {

        if (ch == '(')

            push(ch);

        else if (isalnum(ch))

            postfix[k++] = ch;

        else if (ch == ')')

        {

            while (stack[top] != '(')

                postfix[k++] = pop();

            elem = pop();

        }

        else

        {

            while (pr(stack[top]) >= pr(ch))

                postfix[k++] = pop();

            push(ch);

        }

    }

    while (stack[top] != '#')

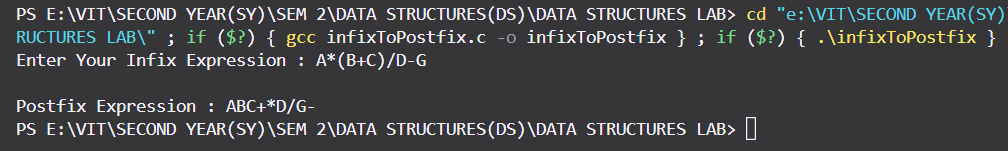
        postfix[k++] = pop();

    postfix[k] = '\0';

    printf("\nPostfix Expression : %s", postfix);

}

OUTPUT :



1. INFIX TO PREFIX

#define MAX 50

#include <stdio.h>

#include <string.h>

#include <ctype.h>

*char* s[MAX];

*int* top = -1;

push(*char* *elem*)

{

    if (top == MAX - 1)

    {

        printf("\nSTACK OVERFLOW\n");

    }

    else

    {

        s[++top] = *elem*;

    }

}

*char* pop()

{

    if (top == MAX - 1)

    {

        printf("\nSTACK OVERFLOW\n");

    }

    else

    {

        return (s[top--]);

    }

}

*int* pr(*char* *elem*)

{

    switch (*elem*)

    {

    case '#':

        return 0;

    case ')':

        return 1;

    case '+':

    case '-':

        return 2;

    case '\*':

    case '/':

        return 3;

    }

}

*int* main()

{

*char* infx[50], prfx[50], ch, elem;

*int* i = 0, k = 0;

    printf("\n\nRead the Infix Expression ? ");

    scanf("%s", infx);

    push('#');

    strrev(infx);

    while ((ch = infx[i++]) != '\0')

    {

        if (ch == ')')

            push(ch);

        else if (isalnum(ch))

            prfx[k++] = ch;

        else if (ch == '(')

        {

            while (s[top] != ')')

                prfx[k++] = pop();

            elem = pop();

        }

        else

        {

            while (pr(s[top]) >= pr(ch))

                prfx[k++] = pop();

            push(ch);

        }

    }

    while (s[top] != '#')

        prfx[k++] = pop();

    prfx[k] = '\0';

    strrev(prfx);

    strrev(infx);

    printf("\n\nGiven Infix Expn: %s Prefix Expn: %s\n", infx, prfx);

}

OUTPUT :

