

**Batch:** A-4 **Roll No.:** 16010422211  **Experiment No.:3**

**Aim:**

1. 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.

1. 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)

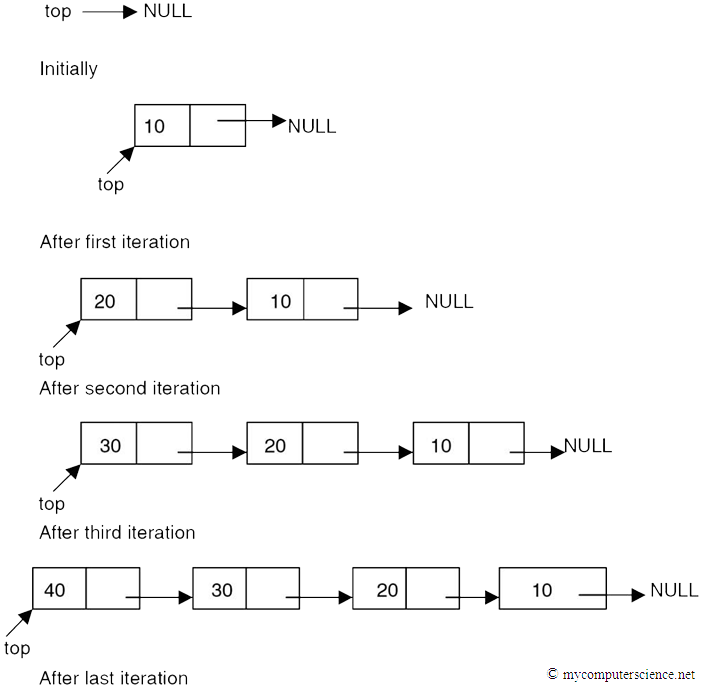
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Theory:**

1. **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**

1. **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 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 :**

1. Implement “dynamic stack” with following functions
   1. **void createStack(void)** **:** Create and initializes 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.

1. **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 reflecting clearly in the output.

**Program and Output:**

a)

#include <stdio.h>

#include <stdlib.h>

// Define a structure for the node of the singly linked list

struct Node {

int data;

struct Node\* next;

};

struct Node\* top = NULL;

// Creating an empty stack

void createStack() {

top = NULL;

printf("Stack created and initialized.\n");

}

// Insert an element at the top of the stack

void push(int value) {

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

if (newNode == NULL) {

printf("Memory allocation failed. Stack is full.\n");

return;

}

newNode->data = value;

newNode->next = top;

top = newNode;

printf("%d is pushed onto the stack.\n", value);

}

// Delete an element from the top of the stack

void pop() {

if (top == NULL) {

printf("Stack is empty. Cannot pop.\n");

return;

}

struct Node\* temp = top;

top = top->next;

int deleteValue = temp->data;

free(temp);

printf("%d is popped from the stack.\n", deleteValue);

}

// Display the top element of the stack

void displayTop() {

if (top == NULL) {

printf("Stack is empty. No top element to display.\n");

} else {

printf("Top element: %d\n", top->data);

}

}

int main() {

int choice, value;

while (1) {

printf("\nStack Operations using Singly Linked List:\n");

printf("1. Create stack\n");

printf("2. Insert an element (push)\n");

printf("3. Delete an element (pop)\n");

printf("4. Display top element\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

createStack();

break;

case 2:

printf("Enter the element to push onto the stack: ");

scanf("%d", &value);

push(value);

break;

case 3:

pop();

break;

case 4:

displayTop();

break;

case 5:

printf("Exiting the program.\n");

exit(0);

default:

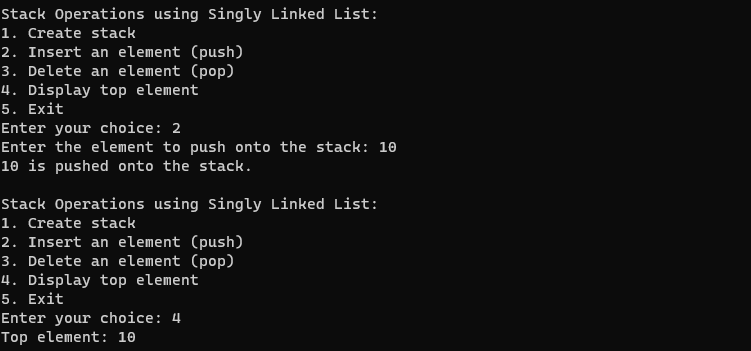
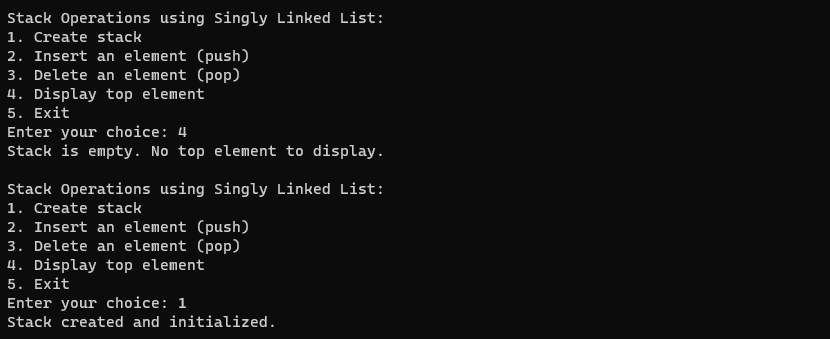
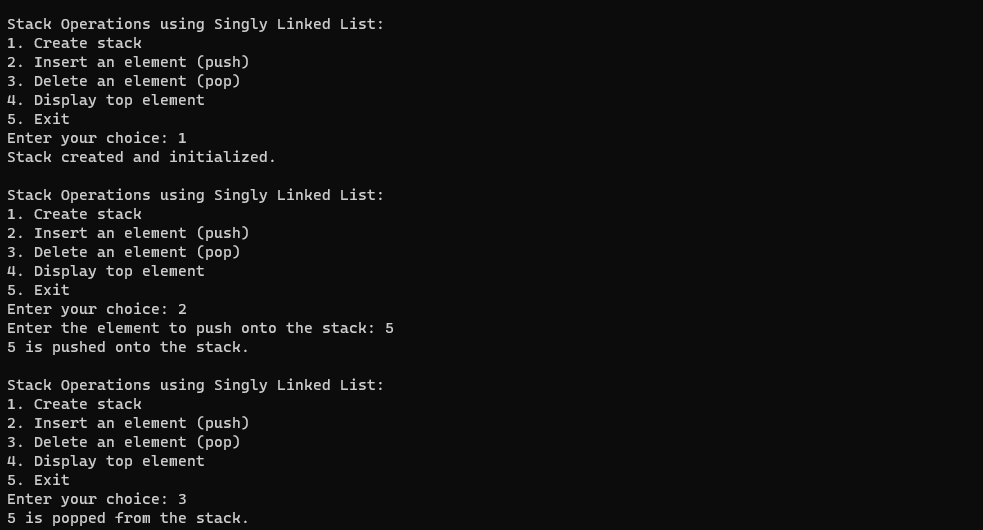
printf("Invalid choice. Please try again.\n");

}

}

return 0;

}

****

b)

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

char infix[100];

char prefix[100];

int i, j = 0;

struct node {

char id;

struct node \*next;

} \*top, \*p, \*newnode;

void createStack() {

top = NULL;

printf("stack is created\n");

}

bool isEmpty() {

return top == NULL;

}

void push(char x) {

newnode = (struct node \*)malloc(sizeof(struct node));

newnode->id = x;

if (isEmpty()) {

newnode->next = NULL;

top = newnode;

} else {

newnode->next = top;

top = newnode;

}

}

void pop() {

if (isEmpty())

printf("stack underflow\n");

else {

p = top;

top = top->next;

prefix[j++] = p->id;

free(p);

}

}

void splPop() {

if (isEmpty())

printf("stack underflow\n");

else {

p = top;

top = top->next;

free(p);

}

}

int P\_check(char a) {

switch (a) {

case '^':

return 3;

case '/':

case '\*':

return 2;

case '+':

case '-':

return 1;

default:

return -1;

}

}

void infixToPrefix() {

printf("Enter the Infix expression: ");

scanf("%s", infix);

int len = strlen(infix);

for (i = len-1; i>=0; i--) {

if (infix[i] == ')')

push(infix[i]);

else if (infix[i] == '(') {

while (top != NULL && top->id != ')')

pop();

if (top != NULL && top->id == ')')

splPop();

} else if (P\_check(infix[i]) > 0) {

while (top != NULL && P\_check(infix[i]) < P\_check(top->id))

pop();

push(infix[i]);

} else

prefix[j++] = infix[i];

}

while (top != NULL)

pop();

}

void printPrefix(){

int len = strlen(prefix);

for(i=len-1;i>=0;i--)

printf(" %c",prefix[i]);

}

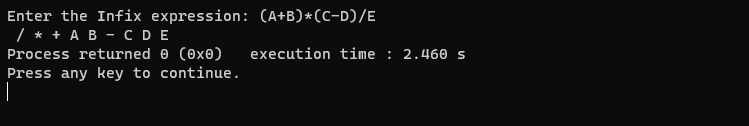
int main() {

infixToPrefix();

printPrefix();

return 0;

}

****

**Outcomes:** CO2. Dynamic STACK in the implementation of Expression Conversion.

**Conclusion**: Learnt how to implement a stack (create, insert, delete and display) & how to convert an infix expression to postfix.

.

**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