Experiment No. : 3

Batch: A3 **Roll No.: 16010423099** **Experiment No.:3**

Aim:

a) 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.

b) WAP to convert a given infix expression into equivalent postfix form using stack implemented in part (a).

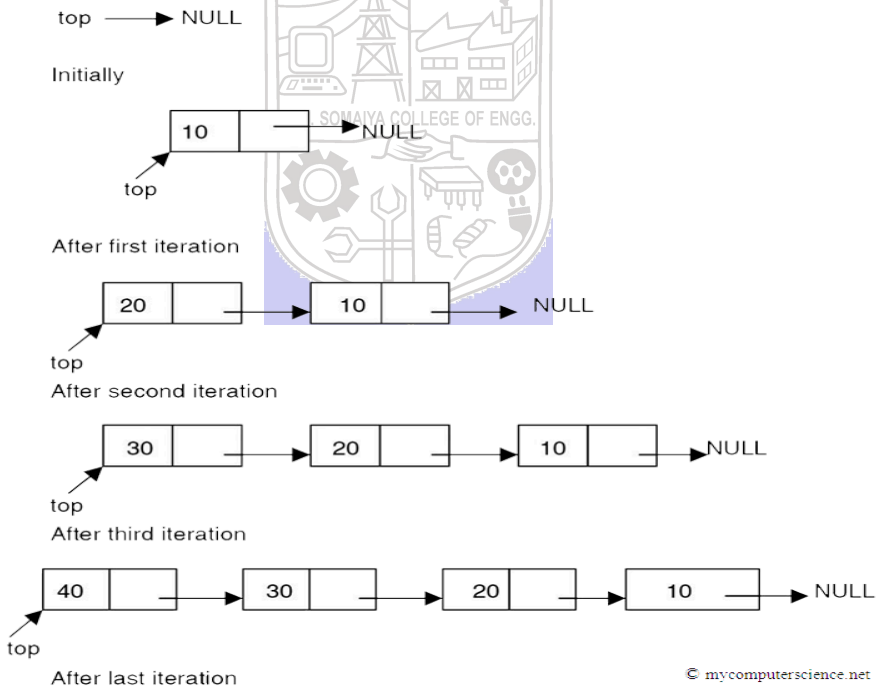
Resources Used: Turbo C editor and compiler

Theory:

a) 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

---

b) 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 :

a) 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.

b) 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:

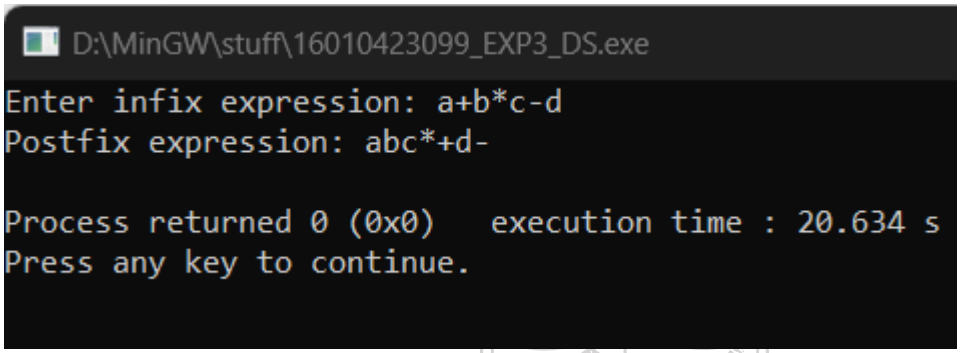
```
#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>


typedef struct Node {

    char data;

    struct Node* next;

} Node;


typedef struct Stack {

    Node* top;

} Stack;


void createStack(Stack* s) {

    s->top = NULL;

}


void push(Stack* s, char x) {

    Node* newNode = (Node*)malloc(sizeof(Node));
```

```
    newNode->data = x;
    newNode->next = s->top;
    s->top = newNode;
}

char pop(Stack* s) {
    if (s->top == NULL) {
        printf("Stack Underflow\n");
        return '\0';
    }
    Node* temp = s->top;
    char popped = temp->data;
    s->top = s->top->next;
    free(temp);
    return popped;
}

int isEmpty(Stack* s) {
    return s->top == NULL;
}

char peek(Stack* s) {
    return (s->top != NULL) ? s->top->data : '\0';
}

int precedence(char op) {
    switch (op) {
```

```
        case '+':
        case '-': return 1;
        case '*':
        case '/': return 2;
        case '^': return 3;
        default: return 0;
    }
}

void infixToPostfix(const char* infix, char* postfix) {
    Stack s;
    createStack(&s);
    int i, j = 0;
    char item;

    for (i = 0; infix[i] != '\0'; i++) {
        item = infix[i];
        if (isalnum(item)) {
            postfix[j++] = item;
        } else if (item == '(') {
            push(&s, item);
        } else if (item == '+' || item == '-' || item == '*' || item == '/' || item == '^') {
            while (!isEmpty(&s) && precedence(peek(&s)) >= precedence(item)) {
                postfix[j++] = pop(&s);
            }
            push(&s, item);
        } else if (item == ')') {
```

```
            while (!isEmpty(&s) && peek(&s) != '(') {
                postfix[j++] = pop(&s);
            }
            pop(&s);
        } else {
            printf("Oops! Invalid character in infix expression\n");
            return;
        }
    }

    while (!isEmpty(&s)) {
        char top = pop(&s);
        if (top == '(') {
            printf("Oops! Mismatched parentheses\n");
            return;
        }
        postfix[j++] = top;
    }

    postfix[j] = '\0';
}

int main() {
    char infix[100], postfix[100];

    printf("Enter infix expression: ");
    scanf("%s", infix);
```

```
    infixToPostfix(infix, postfix);

    printf("Postfix expression: %s\n", postfix);


    return 0;

}
```



Outcomes:
CO2: Apply linear and non-linear data structure in application development.

Conclusion:

Program executed successfully to convert infix expression into postfix expression using concept of singly linked list to create a dynamic stack

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