# **Assignment 6** Dated Jan 13th, 2025

## **Problem Statement**

A program in C to evaluate postfix expression of a math expression given by the user.

## **Algorithm**

### Input

initStack() is used here to take input from the user as a string and convert it to a stack.

### Output

evaluatePostfix() is used to evaluate the necessary postfix expression and the result is stored in a character array which will further be displayed in the main function.

**Algorithm for createNode()**

**Step 1:** Start.  
**Step 2:** Input an integer data for the new node.  
**Step 3:** Allocate memory for a new node of type Node.  
**Step 4:** If memory allocation fails, display an error message and terminate the program.  
**Step 5:** Assign data to the data field of the new node.  
**Step 6:** Set the next pointer of the new node to NULL.  
**Step 7:** Return the newly created node.  
**Step 8:** Stop.  
**Step 9:** [End of function createNode defined at Step 1.]

**Algorithm for push()**

**Step 10:** Start.  
**Step 11:** Input a pointer to the top of the stack (top) and an integer data.  
**Step 12:** Call createNode(data) to create a new node and store the result in newNode.  
**Step 13:** Set newNode->next to the current \*top.  
**Step 14:** Update \*top to point to newNode.  
**Step 15:** Display a message indicating that data has been pushed onto the stack.  
**Step 16:** Stop.  
**Step 17:** [End of function push defined at Step 10.]

**Algorithm for isEmpty()**

**Step 18:** Start.  
**Step 19:** Input a pointer to the top of the stack (top).  
**Step 20:** If top == NULL, return 1 (stack is empty). Otherwise, return 0 (stack is not empty).  
**Step 21:** Stop.  
**Step 22:** [End of function isEmpty defined at Step 18.]

**Algorithm for pop()**

**Step 23:** Start.  
**Step 24:** Input a pointer to the top of the stack (top).  
**Step 25:** Call isEmpty(\*top). If the result is 1, display an underflow message and return -1.  
**Step 26:** Declare a temporary pointer temp and set it to \*top.  
**Step 27:** Update \*top to point to (\*top)->next.  
**Step 28:** Store the data value of temp in a variable popped.  
**Step 29:** Free the memory allocated for temp.  
**Step 30:** Display a message indicating the popped value.  
**Step 31:** Return popped.  
**Step 32:** Stop.  
**Step 33:** [End of function pop defined at Step 23.]

**Algorithm for peek()**

**Step 34:** Start.  
**Step 35:** Input a pointer to the top of the stack (top).  
**Step 36:** Call isEmpty(top). If the result is 1, display an empty stack message and return -1.  
**Step 37:** Return top->data.  
**Step 38:** Stop.  
**Step 39:** [End of function peek defined at Step 34.]

**Algorithm for display()**

**Step 40:** Start.  
**Step 41:** Input a pointer to the top of the stack (top).  
**Step 42:** Call isEmpty(top). If the result is 1, display an empty stack message and stop.  
**Step 43:** Declare a pointer temp and set it to top.  
**Step 44:** Display a message "Stack elements:".  
**Step 45:** While temp != NULL, perform the following:

* **Step 45.1:** Print temp->data.
* **Step 45.2:** Update temp to temp->next.  
  **Step 46:** Print a newline.  
  **Step 47:** Stop.  
  **Step 48:** [End of function display defined at Step 40.]

**Algorithm for displayMenu()**

**Step 49:** Start.  
**Step 50:** Display the available stack operations.   
**Step 51:** Display a prompt for user choice.  
**Step 52:** Stop.  
**Step 53:** [End of function displayMenu defined at Step 49.]

**Algorithm for main()**

**Step 54:** Start.  
**Step 55:** Declare a pointer stack and initialize it to NULL.  
**Step 56:** Declare integers choice and value.  
**Step 57:** Enter an infinite loop to handle user input:

* **Step 57.1:** Call displayMenu().
* **Step 57.2:** Input the user choice and store it in choice.
* **Step 57.3:** Perform actions based on the value of choice:

**Case 1:** Call push(&stack, value) after prompting the user for value.

**Case 2:** Call pop(&stack).

**Case 3:** Display the result of peek(stack).

**Case 4:** Call display(stack).

**Case 5:** Display an exit message and break the loop using goto end.

**Default Case:** Display an invalid choice message.

**Step 58:** Label end to exit the loop and display a final thank-you message.  
**Step 59:** Stop.  
**Step 60:** [End of function main defined at Step 54.]

## **Source Code**

#include <ctype.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 8192

// Stack structure definition

typedef struct {

int data[MAX];

int top;

} Stack;

// Function to initialize the stack

void initStack(Stack\* s)

{

s->top = -1;

}

// Function to check if the stack is empty

int isEmpty(Stack\* s)

{

return s->top == -1;

}

// Function to push an element onto the stack

void push(Stack\* s, int value)

{

if (s->top == MAX - 1) {

printf("Stack overflow\n");

return;

}

s->data[++s->top] = value;

}

// Function to pop an element from the stack

int pop(Stack\* s)

{

if (isEmpty(s)) {

printf("Stack underflow\n");

exit(1);

}

return s->data[s->top--];

}

// Function to evaluate a postfix expression

int evaluatePostfix(char\* seq)

{

Stack stack;

initStack(&stack);

for (int i = 0; i < strlen(seq); i++) {

char ch = seq[i];

// If the character is a digit, push it onto the stack

if (isdigit(ch)) {

push(&stack, ch - '0');

}

// If the character is an operator, pop two elements, apply the operator, and push the result

else if (ch == '+' || ch == '-' || ch == '\*' || ch == '/') {

int a = pop(&stack);

int b = pop(&stack);

// Handling division by zero

if (a == 0) {

printf("error: Divide by zero is impossible.\n");

exit(1);

}

switch (ch) {

case '+':

push(&stack, b + a);

break;

case '-':

push(&stack, b - a);

break;

case '\*':

push(&stack, b \* a);

break;

case '/':

push(&stack, b / a);

break;

}

}

}

// The final result is the only element left in the stack

return pop(&stack);

}

int main()

{

char expression[MAX];

printf("Enter a postfix expression: ");

fgets(expression, MAX, stdin);

// Remove newline character from input

expression[strcspn(expression, "\n")] = '\0';

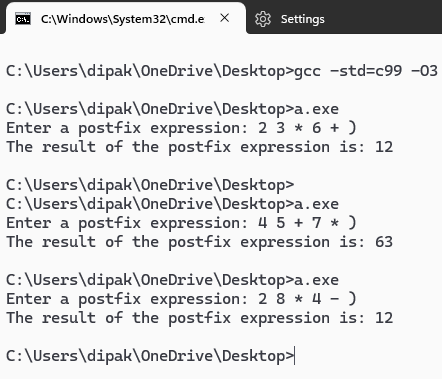
int result = evaluatePostfix(expression);

printf("The result of the postfix expression is: %d\n", result);

return 0;

}

## **Output**



### Discussion

Global variables should be used to the least. However, it has been applied here to reduce the complexity of using pointers and tricky lines.

**Teacher’s signature**