**Batch: B-1 Roll No.: 16010122104**

**Experiment / assignment / tutorial No.**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title:**  Implementation of Stack applications. |

**Objective:** To implement applications of stack

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| 1 | Explain the different data structures used in problem solving |

**Books/ Journals/ Websites referred:**

1. *Fundamentals Of Data Structures In C –* Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications –* Jean Paul Tremblay,

Paul G. Sorenson

1. *Data Structures A Pseudo Approach with C –* Richard F. Gilberg & Behrouz A. Forouzan
2. [*https://www.cprogramming.com/tutorial/computersciencetheory/stack.html*](https://www.cprogramming.com/tutorial/computersciencetheory/stack.html)
3. [*https://www.geeksforgeeks.org/stack-data-structure-introduction-program/*](https://www.geeksforgeeks.org/stack-data-structure-introduction-program/)
4. [*https://www.thecrazyprogrammer.com/2013/12/c-program-for-array-representation-of-stack-push-pop-display.html*](https://www.thecrazyprogrammer.com/2013/12/c-program-for-array-representation-of-stack-push-pop-display.html)

**Assigned Stack application**:

 Implement infix to postfix conversion and evaluation of postfix using stack implemented statically.

**Algorithm:**

**Infix to Postfix Conversion Algorithm**

1. Create a stack to hold operators and operands.
2. Initialize an index i and an empty postfix expression string.
3. Iterate through the infix expression character by character: a. If the character is an alphanumeric character (operand), append it to the postfix expression. b. If the character is an open parenthesis '(', push it onto the stack. c. If the character is a close parenthesis ')', pop operators from the stack and append them to the postfix expression until an open parenthesis '(' is encountered. Pop and discard the '('. d. If the character is an operator: i. While the stack is not empty and the precedence of the operator on the stack is greater than or equal to the precedence of the current operator, pop the operator from the stack and append it to the postfix expression. ii. Push the current operator onto the stack.
4. After processing all characters, pop any remaining operators from the stack and append them to the postfix expression.
5. The postfix expression is now the result.

**Postfix Expression Evaluation Algorithm**

1. Create a stack to hold operands.
2. Initialize an index i.
3. Iterate through the postfix expression character by character: a. If the character is a digit, convert it to an integer and push it onto the operand stack. b. If the character is an operator: i. Pop the top two operands from the operand stack (operand2 and operand1). ii. Apply the operator to operand1 and operand2 to get the result. iii. Push the result back onto the operand stack.
4. After processing all characters in the postfix expression, the operand stack contains the final result.
5. Pop the result from the operand stack, which is the result of the postfix expression.

**Example:**

**Sourcecode:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MAX\_SIZE 100

// Stack definition

struct Stack {

char items[MAX\_SIZE];

int top;

};

// Initialize the stack

void initialize(struct Stack \*stack) {

stack->top = -1;

}

// Check if the stack is empty

int isEmpty(struct Stack \*stack) {

return stack->top == -1;

}

// Check if the stack is full

int isFull(struct Stack \*stack) {

return stack->top == MAX\_SIZE - 1;

}

// Push an item onto the stack

void push(struct Stack \*stack, char item) {

if (!isFull(stack)) {

stack->items[++stack->top] = item;

} else {

printf("Stack overflow\n");

}

}

// Pop an item from the stack

char pop(struct Stack \*stack) {

if (!isEmpty(stack)) {

return stack->items[stack->top--];

} else {

printf("Stack underflow\n");

return '\0';

}

}

// Get the top item from the stack without removing it

char peek(struct Stack \*stack) {

if (!isEmpty(stack)) {

return stack->items[stack->top];

} else {

printf("Stack is empty\n");

return '\0';

}

}

// Check if the character is an operator

int isOperator(char c) {

return (c == '+' || c == '-' || c == '\*' || c == '/');

}

// Get the precedence of an operator

int getPrecedence(char c) {

if (c == '+' || c == '-')

return 1;

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

return 2;

return 0; // For other characters (operands)

}

// Convert infix expression to postfix

void infixToPostfix(char infix[], char postfix[]) {

struct Stack stack;

initialize(&stack);

int i, j;

char c, popped;

for (i = 0, j = -1; infix[i]; i++) {

c = infix[i];

if (isalnum(c)) {

postfix[++j] = c;

} else if (c == '(') {

push(&stack, c);

} else if (c == ')') {

while (!isEmpty(&stack) && peek(&stack) != '(') {

popped = pop(&stack);

postfix[++j] = popped;

}

if (!isEmpty(&stack) && peek(&stack) != '(') {

printf("Invalid expression\n");

return;

} else {

pop(&stack); // Remove the '(' from the stack

}

} else if (isOperator(c)) {

while (!isEmpty(&stack) && getPrecedence(c) <= getPrecedence(peek(&stack))) {

popped = pop(&stack);

postfix[++j] = popped;

}

push(&stack, c);

}

}

while (!isEmpty(&stack)) {

popped = pop(&stack);

postfix[++j] = popped;

}

postfix[++j] = '\0';

}

// Evaluate postfix expression

int evaluatePostfix(char postfix[]) {

struct Stack stack;

initialize(&stack);

int i, operand1, operand2, result;

for (i = 0; postfix[i]; i++) {

char c = postfix[i];

if (isdigit(c)) {

push(&stack, c - '0'); // Convert char digit to integer

} else if (isOperator(c)) {

operand2 = pop(&stack);

operand1 = pop(&stack);

switch (c) {

case '+':

result = operand1 + operand2;

break;

case '-':

result = operand1 - operand2;

break;

case '\*':

result = operand1 \* operand2;

break;

case '/':

if (operand2 == 0) {

printf("Division by zero\n");

return 0;

}

result = operand1 / operand2;

break;

}

push(&stack, result);

}

}

return pop(&stack);

}

int main() {

char infix[MAX\_SIZE], postfix[MAX\_SIZE];

printf("Enter an infix expression: ");

scanf("%s", infix);

infixToPostfix(infix, postfix);

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

int result = evaluatePostfix(postfix);

printf("Result of expression: %d\n", result);

return 0;

}

**Output Screenshots:**

A screenshot of a computer code

Description automatically generated

**Conclusion:**

We implemented applications of stack.