22AIE112 Data Structure And Algorithms
Labsheet 5
Stack

Date : 9/7/2023 **Roll No: AM.EN.U4AIE22009**

1.Implement the following operations in stack data structure: Push, pop, peek using Array

```
#include <stdio.h>

#define MAX_SIZE 100

int stack[MAX_SIZE];
int top = -1;

void push(int value) {
if (top == MAX_SIZE - 1) {
printf("Stack overflow: Cannot push element\n");
return;
}
top++;
stack[top] = value;
printf("Pushed %d onto the stack\n", value);
}

void pop() {
if (top == -1) {
printf("Stack underflow: Cannot pop element\n");
return;
}
int poppedValue = stack[top];
top--;
printf("Popped %d from the stack\n", poppedValue);
}

int peek() {
if (top == -1) {
printf("Stack is empty\n");
return -1;
}
return stack[top];
}

void displayStack() {
if (top == -1) {
printf("Stack is empty\n");
return;
```

```
}
printf("Stack elements: ");
for (int i = 0; i <= top; i++) {
printf("%d ", stack[i]);
}
printf("\n");
}

int main() {
int choice, value;
do {
printf("\nStack Operations\n");
printf("1. Push\n");
printf("2. Pop\n");
printf("3. Peek\n");
printf("4. Display Stack\n");
printf("0. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 0:
printf("Exiting the program\n");
break;
case 1:
printf("Enter the value to push: ");
scanf("%d", &value);
push(value);
break;
case 2:
pop();
break;
case 3:
printf("Top of the stack: %d\n", peek());
break;
case 4:
displayStack();
break;
default:
printf("Invalid choice\n");
}
} while (choice != 0);
return 0;
}
```

```
Stack Operations
1. Push
2. Pop
3. Peek
4. Display Stack
0. Exit
Enter your choice:
1
Enter the value to push: 100
Pushed 100 onto the stack

Stack Operations
1. Push
2. Pop
3. Peek
4. Display Stack
0. Exit
Enter your choice: 1
Enter the value to push: 200
Pushed 200 onto the stack

Stack Operations
1. Push
2. Pop
3. Peek
4. Display Stack
0. Exit
Enter your choice: 2
Popped 200 from the stack

Stack Operations
1. Push
2. Pop
3. Peek
4. Display Stack
0. Exit
Enter your choice: 1
Enter the value to push: 300
Pushed 300 onto the stack

Stack Operations
1. Push
2. Pop
3. Peek
4. Display Stack
0. Exit
Enter your choice: 3
Top of the stack: 300

Stack Operations
1. Push
2. Pop
3. Peek
4. Display Stack
0. Exit
Enter your choice:
```

2.Implement the following operations in stack data structure: Push, pop, peek using Linked list

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
int data;
struct Node* next;
};

struct Node* top = NULL;

void push(int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
if (newNode == NULL) {
printf("Memory allocation failed. Unable to push element.\n");
return;
}
newNode->data = value;
newNode->next = top;
top = newNode;
printf("Pushed %d onto the stack\n", value);
}

void pop() {
if (top == NULL) {
printf("Stack underflow: Cannot pop element\n");
return;
}
struct Node* temp = top;
int poppedValue = temp->data;
top = top->next;
free(temp);
printf("Popped %d from the stack\n", poppedValue);
}

int peek() {
if (top == NULL) {
printf("Stack is empty\n");
return -1;
}
return top->data;
}



void displayStack() {
if (top == NULL) {
```

```
printf("Stack is empty\n");
return;
}
printf("Stack elements: ");
struct Node* current = top;
while (current != NULL) {
printf("%d ", current->data);
current = current->next;
}
printf("\n");
}

int main() {
int choice, value;
do {
printf("\nStack Operations\n");
printf("1. Push\n");
printf("2. Pop\n");
printf("3. Peek\n");
printf("4. Display Stack\n");
printf("0. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 0:
printf("Exiting the program\n");
break;
case 1:
printf("Enter the value to push: ");
scanf("%d", &value);
push(value);
break;
case 2:
pop();
break;
case 3:
printf("Top of the stack: %d\n", peek());
break;
case 4:
displayStack();
break;
default:
printf("Invalid choice\n");
}
} while (choice != 0);
return 0;
}
```

```
1   #include <stdio.h>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
1. Push
2. Pop
3. Peek
4. Display Stack
0. Exit
Enter your choice: 1
Enter the value to push: 200
Pushed 200 onto the stack

Stack Operations
1. Push
2. Pop
3. Peek
4. Display Stack
0. Exit
Enter your choice: 1
Enter the value to push: 300
Pushed 300 onto the stack

Stack Operations
1. Push
2. Pop
3. Peek
4. Display Stack
0. Exit
Enter your choice: 2
Popped 300 from the stack

Stack Operations
1. Push
2. Pop
3. Peek
4. Display Stack
0. Exit
Enter your choice: 3
Top of the stack: 200

Stack Operations
1. Push
2. Pop
3. Peek
4. Display Stack
0. Exit
Enter your choice: 4
Stack elements: 200 100

Stack Operations
1. Push
2. Pop
3. Peek
4. Display Stack
0. Exit
Enter your choice:
```

3.Implement a function getminElement() to return the minimum element in a stack.

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
int data;
int min;
struct Node* next;
};

struct Node* top = NULL;

void push(int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
if (newNode == NULL) {
printf("Memory allocation failed. Unable to push element.\n");
return;
}
newNode->data = value;
newNode->next = top;
top = newNode;
if (top->next == NULL || value < top->next->min)
top->min = value;
else
top->min = top->next->min;
printf("Pushed %d onto the stack\n", value);
}

void pop() {
if (top == NULL) {
printf("Stack underflow: Cannot pop element\n");
return;
}
struct Node* temp = top;
int poppedValue = temp->data;
top = top->next;
free(temp);
printf("Popped %d from the stack\n", poppedValue);
}

int peek() {
if (top == NULL) {
printf("Stack is empty\n");
return -1;
}
return top->data;
}
```

```
int getMin() {
if (top == NULL) {
printf("Stack is empty\n");
return -1;
}
return top->min;
}

struct Node* copyStack() {
if (top == NULL) {
printf("Stack is empty. Cannot copy.\n");
return NULL;
}
struct Node* originalCurrent = top;
struct Node* duplicateTop = NULL;
struct Node* duplicateCurrent = NULL;
while (originalCurrent != NULL) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
if (newNode == NULL) {
printf("Memory allocation failed. Unable to copy element.\n");
return NULL;
}
newNode->data = originalCurrent->data;
newNode->next = NULL;
if (duplicateTop == NULL) {
duplicateTop = newNode;
duplicateCurrent = newNode;
} else {
duplicateCurrent->next = newNode;
duplicateCurrent = duplicateCurrent->next;
}
originalCurrent = originalCurrent->next;
}
return duplicateTop;
}

void displayStack(struct Node* stack) {
if (stack == NULL) {
printf("Stack is empty\n");
return;
}
printf("Stack elements: ");
struct Node* current = stack;
while (current != NULL) {
printf("%d ", current->data);
current = current->next;
}
printf("\n");
}
```

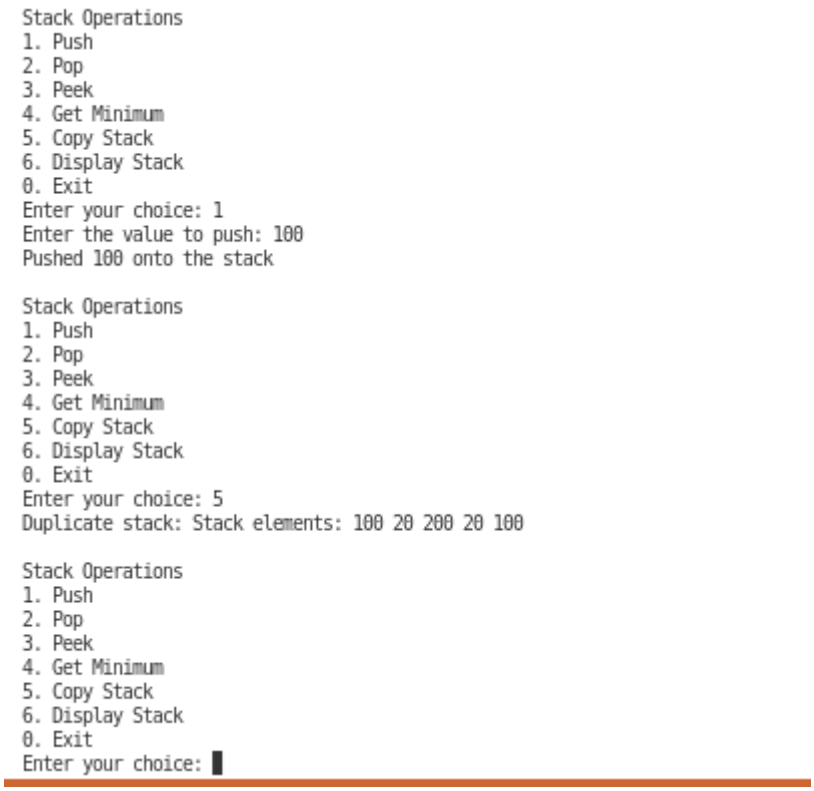
```c
int main() {
int choice, value;
do {
printf("\nStack Operations\n");
printf("1. Push\n");
printf("2. Pop\n");
printf("3. Peek\n");
printf("4. Get Minimum\n");
printf("5. Copy Stack\n");
printf("6. Display Stack\n");
printf("0. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 0:
printf("Exiting the program\n");
break;
case 1:
printf("Enter the value to push: ");
scanf("%d", &value);
push(value);
break;
case 2:
pop();
break;
case 3:
printf("Top of the stack: %d\n", peek());
break;
case 4:
printf("Minimum element: %d\n", getMin());
break;
case 5: {
struct Node* duplicateStack = copyStack();
printf("Duplicate stack: ");
displayStack(duplicateStack);
break;
}
case 6:
displayStack(top);
break;
default:
printf("Invalid choice\n");
}
} while (choice != 0);
return 0;
}
```

```
Stack Operations
1. Push
2. Pop
3. Peek
4. Get Minimum
5. Copy Stack
6. Display Stack
0. Exit
Enter your choice: 4
Minimum element: 20
```

4.Implement a copyStack() function to return a duplicate stack of original stack.

```c
struct Node* copyStack() {
    if (top == NULL) {
        printf("Stack is empty. Cannot copy.\n");
        return NULL;
    }

    struct Node* originalCurrent = top;
    struct Node* duplicateTop = NULL;
    struct Node* duplicateCurrent = NULL;

    while (originalCurrent != NULL) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        if (newNode == NULL) {
            printf("Memory allocation failed. Unable to copy element.\n");
            return NULL;
        }

        newNode->data = originalCurrent->data;
        newNode->next = NULL;

        if (duplicateTop == NULL) {
            duplicateTop = newNode;
            duplicateCurrent = newNode;
        } else {
            duplicateCurrent->next = newNode;
            duplicateCurrent = duplicateCurrent->next;
        }

        originalCurrent = originalCurrent->next;
    }

    return duplicateTop;
}
```

Output



5.Implement a fucntion to reverse an input string using stack

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_SIZE 100

struct Node {
char data;
struct Node* next;
};

struct Node* top = NULL;

void push(char value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
if (newNode == NULL) {
printf("Memory allocation failed. Unable to push element.\n");
return;
}
newNode->data = value;
newNode->next = top;
top = newNode;
}

char pop() {
```

```
if (top == NULL) {
printf("Stack underflow: Cannot pop element\n");
return '\0';
}
struct Node* temp = top;
char poppedValue = temp->data;
top = top->next;
free(temp);
return poppedValue;
}

void reverseString(char* str) {
int length = strlen(str);
for (int i = 0; i < length; i++) {
push(str[i]);
}
for (int i = 0; i < length; i++) {
str[i] = pop();
}
}

int main() {
char input[MAX_SIZE];
printf("Enter a string: ");
fgets(input, sizeof(input), stdin);
input[strcspn(input, "\n")] = '\0';
printf("Original string: %s\n", input);
reverseString(input);
printf("Reversed string: %s\n", input);
return 0;
}
```

Output

6.Given a stack with push(), pop(), isEmpty() operations. Write a function that takes a stack with suitable push, pop and isEmpty operations and modifies it such that the middle element is removed.

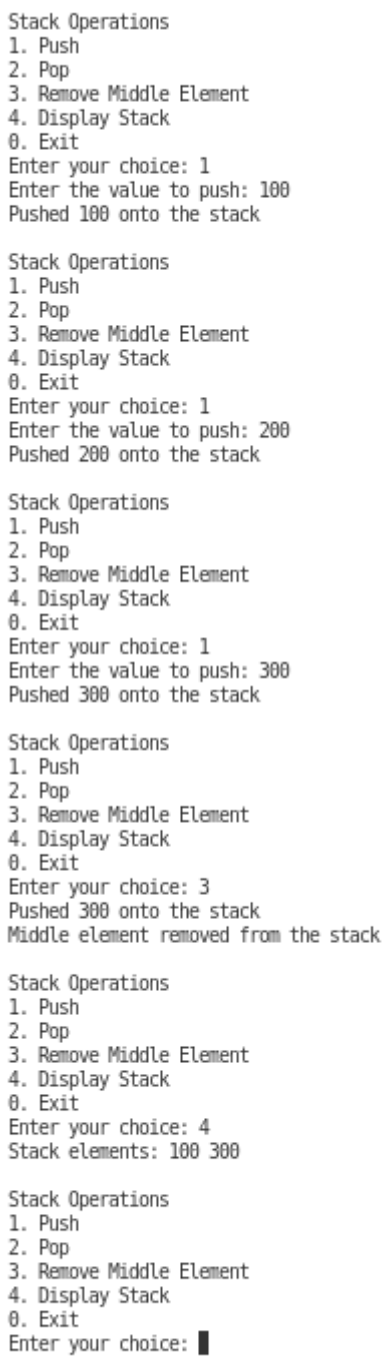
Example: Input : Stack[] = [1, 2, 3, 4, 5] Output : Stack[] = [1, 2, 4, 5] Input : Stack[] = [1, 2, 3, 4, 5, 6] Output : Stack[] = [1, 2, 4, 5, 6]

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX_SIZE 100

int stack[MAX_SIZE];
int top = -1;

void push(int value) {
if (top == MAX_SIZE - 1) {
printf("Stack overflow: Cannot push element\n");
return;
}
top++;
stack[top] = value;
printf("Pushed %d onto the stack\n", value);
}

int pop() {
if (top == -1) {
printf("Stack underflow: Cannot pop element\n");
return -1;
}
int poppedValue = stack[top];
top--;
return poppedValue;
}

bool isEmpty() {
return (top == -1);
}

void removeMiddle() {
int size = top + 1;
int middle = size / 2;
int* aux = (int*)malloc(middle * sizeof(int));
if (aux == NULL) {
printf("Memory allocation failed\n");
return;
}
```

```
for (int i = 0; i < middle; i++) {
aux[i] = pop();
}
pop();

for (int i = middle - 1; i >= 0; i--) {
push(aux[i]);
}
free(aux);
}

void displayStack() {
if (isEmpty()) {
printf("Stack is empty\n");
return;
}
printf("Stack elements: ");
for (int i = 0; i <= top; i++) {
printf("%d ", stack[i]);
}
printf("\n");
}

int main() {
int choice, value;
do {
printf("\nStack Operations\n");
printf("1. Push\n");
printf("2. Pop\n");
printf("3. Remove Middle Element\n");
printf("4. Display Stack\n");
printf("0. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 0:
printf("Exiting the program\n");
break;
case 1:
printf("Enter the value to push: ");
scanf("%d", &value);
push(value);
break;
case 2:
printf("Popped %d from the stack\n", pop());
break;
case 3:
removeMiddle();
printf("Middle element removed from the stack\n");
break;
```

```
case 4:
displayStack();
break;
default:
printf("Invalid choice\n");
}
} while (choice != 0);
return 0;
}
```

Output



7.Implement two stacks using a single array. One stack should grow upward from the bottom of the array, while the other stack should grow downward from the top of the array. Design the following operations for each stack:

a) push1(value): Inserts the given value into the first stack.

b) push2(value): Inserts the given value into the second stack.

c) pop1(): Removes and returns the top element from the first stack.

d) pop2(): Removes and returns the top element from the second stack.

e) The class should handle stack overflow and underflow conditions, raising appropriate exceptions when necessary.

```
#include <stdio.h>
#include <stdbool.h>

#define MAX_SIZE 100

int stack[MAX_SIZE];
int top1 = -1;
int top2 = MAX_SIZE;

void push1(int value) {
if (top1 + 1 == top2) {
printf("Stack overflow: Cannot push element into the first stack\n");
return;
}
top1++;
stack[top1] = value;
printf("Pushed %d into the first stack\n", value);
}

void push2(int value) {
if (top2 - 1 == top1) {
printf("Stack overflow: Cannot push element into the second stack\n");
return;
}
top2--;
stack[top2] = value;
printf("Pushed %d into the second stack\n", value);
}

int pop1() {
if (top1 == -1) {
printf("Stack underflow: Cannot pop element from the first stack\n");
return -1;
}
int poppedValue = stack[top1];
top1--;
```

```
return poppedValue;
}

int pop2() {
if (top2 == MAX_SIZE) {
printf("Stack underflow: Cannot pop element from the second stack\n");
return -1;
}
int poppedValue = stack[top2];
top2++;
return poppedValue;
}

void displayStacks() {
printf("First Stack: ");
if (top1 == -1) {
printf("Empty");
} else {
for (int i = 0; i <= top1; i++) {
printf("%d ", stack[i]);
}
}
printf("\n");
printf("Second Stack: ");
if (top2 == MAX_SIZE) {
printf("Empty");
} else {
for (int i = MAX_SIZE - 1; i >= top2; i--) {
printf("%d ", stack[i]);
}
}
printf("\n");
}

int main() {
int choice, stackNum, value;
while (true) {
printf("\nStack Operations\n");
printf("1. Push to Stack 1\n");
printf("2. Push to Stack 2\n");
printf("3. Pop from Stack 1\n");
printf("4. Pop from Stack 2\n");
printf("5. Display Stacks\n");
printf("0. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 0:
printf("Exiting the program\n");
return 0;
case 1:
```

```
printf("Enter the value to push: ");
scanf("%d", &value);
push1(value);
break;
case 2:
printf("Enter the value to push: ");
scanf("%d", &value);
push2(value);
break;
case 3:
value = pop1();
if (value != -1) {
printf("Popped %d from the first stack\n", value);
}
break;
case 4:
value = pop2();
if (value != -1) {
printf("Popped %d from the second stack\n", value);
}
break;
case 5:
displayStacks();
break;
default:
printf("Invalid choice\n");
}
}
}
```

Output

```
0. Exit
Enter your choice: 1
Enter the value to push: 200
Pushed 200 into the first stack

Stack Operations
1. Push to Stack 1
2. Push to Stack 2
3. Pop from Stack 1
4. Pop from Stack 2
5. Display Stacks
0. Exit
Enter your choice: 2
Enter the value to push: 100
Pushed 100 into the second stack

Stack Operations
1. Push to Stack 1
2. Push to Stack 2
3. Pop from Stack 1
4. Pop from Stack 2
5. Display Stacks
0. Exit
Enter your choice: 4
Popped 100 from the second stack

Stack Operations
1. Push to Stack 1
2. Push to Stack 2
3. Pop from Stack 1
4. Pop from Stack 2
5. Display Stacks
0. Exit
Enter your choice: 2
Enter the value to push: 300
Pushed 300 into the second stack

Stack Operations
1. Push to Stack 1
2. Push to Stack 2
3. Pop from Stack 1
4. Pop from Stack 2
5. Display Stacks
0. Exit
Enter your choice: 2
Enter the value to push: 400
Pushed 400 into the second stack

Stack Operations
1. Push to Stack 1
2. Push to Stack 2
3. Pop from Stack 1
4. Pop from Stack 2
5. Display Stacks
0. Exit
Enter your choice: 5
First Stack: 100 200
Second Stack: 300 400

Stack Operations
1. Push to Stack 1
2. Push to Stack 2
3. Pop from Stack 1
4. Pop from Stack 2
5. Display Stacks
0. Exit
Enter your choice:
```

8.Write a program to implement the conversions between infix and prefix/postfix expressions, as well as evaluates prefix and postfix expressions. Your program should utilize the stack data structure to perform these operations efficiently. Design the following functionalities:

a. infix_to_prefix(expression): This method takes an infix expression as input and converts it to its equivalent prefix notation. The infix expression will consist of arithmetic operators (+, -, *, /), parentheses (,), and operands (single-digit integers).

b. infix_to_postfix(expression): This method takes an infix expression as input and converts it to its equivalent postfix notation.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>

#define MAX_SIZE 100

struct Stack {
int top;
char items[MAX_SIZE];
};

void initializeStack(struct Stack* stack) {
stack->top = -1;
}

bool isStackEmpty(struct Stack* stack) {
return (stack->top == -1);
}

bool isStackFull(struct Stack* stack) {
return (stack->top == MAX_SIZE - 1);
}

void push(struct Stack* stack, char value) {
if (isStackFull(stack)) {
printf("Stack overflow: Cannot push element\n");
return;
}

stack->top++;
stack->items[stack->top] = value;
```

```
}

char pop(struct Stack* stack) {
if (isStackEmpty(stack)) {
printf("Stack underflow: Cannot pop element\n");
return '\0';
}

char poppedValue = stack->items[stack->top];
stack->top--;
return poppedValue;
}

char peek(struct Stack* stack) {
if (isStackEmpty(stack)) {
return '\0';
}

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

int getPrecedence(char operator) {
switch (operator) {
case '+':
case '-':
return 1;
case '*':
case '/':
return 2;
default:
return 0;
}
}

bool isOperator(char symbol) {
return (symbol == '+' || symbol == '-' || symbol == '*' || symbol == '/');
}

bool isOperand(char symbol) {
return (isalpha(symbol) || isdigit(symbol));
}

char* infix_to_prefix(char* expression) {
int length = strlen(expression);
struct Stack operatorStack;
struct Stack resultStack;

initializeStack(&operatorStack);
initializeStack(&resultStack);

for (int i = length - 1; i >= 0; i--) {
```

```
char symbol = expression[i];

if (symbol == ' ' || symbol == '\t') {
continue;
}

if (isOperand(symbol)) {
push(&resultStack, symbol);
} else if (symbol == ')') {
push(&operatorStack, symbol);
} else if (symbol == '(') {
while (peek(&operatorStack) != ')' && !isStackEmpty(&operatorStack)) {
char poppedOperator = pop(&operatorStack);
push(&resultStack, poppedOperator);
}

if (peek(&operatorStack) == ')') {
pop(&operatorStack);
}
} else if (isOperator(symbol)) {
while (!isStackEmpty(&operatorStack) && getPrecedence(symbol) <
getPrecedence(peek(&operatorStack))) {
char poppedOperator = pop(&operatorStack);
push(&resultStack, poppedOperator);
}

push(&operatorStack, symbol);
}
}

while (!isStackEmpty(&operatorStack)) {
char poppedOperator = pop(&operatorStack);
push(&resultStack, poppedOperator);
}

char* prefixExpression = (char*)malloc((resultStack.top + 2) * sizeof(char));
int index = 0;

while (!isStackEmpty(&resultStack)) {
prefixExpression[index] = pop(&resultStack);
index++;
}

prefixExpression[index] = '\0';

return prefixExpression;
}

char* infix_to_postfix(char* expression) {
int length = strlen(expression);
struct Stack operatorStack;
```

```
char* postfixExpression = (char*)malloc((length + 1) * sizeof(char));
int postfixIndex = 0;

initializeStack(&operatorStack);

for (int i = 0; i < length; i++) {
char symbol = expression[i];

if (symbol == ' ' || symbol == '\t') {
continue;
}

if (isOperand(symbol)) {
postfixExpression[postfixIndex] = symbol;
postfixIndex++;
} else if (symbol == '(') {
push(&operatorStack, symbol);
} else if (symbol == ')') {
while (peek(&operatorStack) != '(' && !isStackEmpty(&operatorStack)) {
char poppedOperator = pop(&operatorStack);
postfixExpression[postfixIndex] = poppedOperator;
postfixIndex++;
}

if (peek(&operatorStack) == '(') {
pop(&operatorStack);
}
} else if (isOperator(symbol)) {
while (!isStackEmpty(&operatorStack) && getPrecedence(symbol) <=
getPrecedence(peek(&operatorStack))) {
char poppedOperator = pop(&operatorStack);
postfixExpression[postfixIndex] = poppedOperator;
postfixIndex++;
}

push(&operatorStack, symbol);
}
}

while (!isStackEmpty(&operatorStack)) {
char poppedOperator = pop(&operatorStack);
postfixExpression[postfixIndex] = poppedOperator;
postfixIndex++;
}

postfixExpression[postfixIndex] = '\0';

return postfixExpression;
}

int main() {
```

```
int _caseOperator;
char expression[MAX_SIZE];
printf("Enter the infix expression: ");
fgets(expression, sizeof(expression), stdin);
expression[strcspn(expression, "\n")] = '\0';

printf("How would you wish to evaluate the given expression: \n");
printf("1.infix to prefix\n");
printf("2.infix to postfix\n");

scanf("%d",&_caseOperator);

switch(_caseOperator){
case 1:
char* prefixExpression = infix_to_prefix(expression);
printf("Prefix expression: %s\n", prefixExpression);
free(prefixExpression);
break;
case 2:
char* postfixExpression = infix_to_postfix(expression);

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

free(postfixExpression);

printf("exp");
break;
default:
printf("invalid operation");

}


return 0;
}
```

Output for infix to prefix

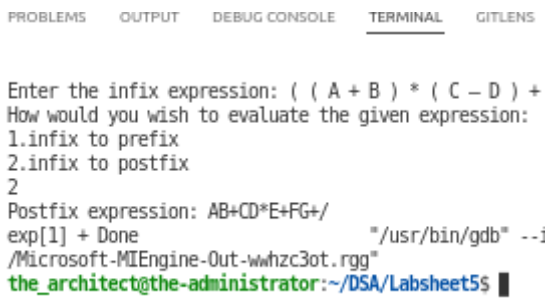
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS

Enter the infix expression: (A+B)/(C-D)
How would you wish to evaluate the given expression:
1.infix to prefix
2.infix to postfix
1
Prefix expression: /+AB-CD
[1] + Done                    "/usr/bin/gdb" --inte
crosoft-MIEngine-Out-tzzd5tgg.vea"
the_architect@the-administrator:~/DSA/Labsheet5$
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS

Enter the infix expression: ( ( A + B ) * ( C – D ) +
How would you wish to evaluate the given expression:
1.infix to prefix
2.infix to postfix
1
Prefix expression: /+*+ABCDE+FG
[1] + Done                    "/usr/bin/gdb" --inte
crosoft-MIEngine-Out-yxkvgoxk.z42"
the_architect@the-administrator:~/DSA/Labsheet5$
```

Output for infix to postfix

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS

Enter the infix expression: a+b-c
How would you wish to evaluate the given expression:
1.infix to prefix
2.infix to postfix
2
Postfix expression: ab+c-
exp[1] + Done                 "/usr/bin/gdb" --i
/Microsoft-MIEngine-Out-fzmqlght.3sm"
the_architect@the-administrator:~/DSA/Labsheet5$
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS

Enter the infix expression: a+b*c
How would you wish to evaluate the given expression:
1.infix to prefix
2.infix to postfix
2
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS

Enter the infix expression: (A+B)/(C-D)
How would you wish to evaluate the given expression:
1.infix to prefix
2.infix to postfix
2
Postfix expression: AB+CD-/
exp[1] + Done                 "/usr/bin/gdb" --i
/Microsoft-MIEngine-Out-zmbfjhbi.hyl"
the_architect@the-administrator:~/DSA/Labsheet5$
```



c. evaluate_prefix(expression): This method takes a prefix expression as input and evaluates it, returning the result.

d. evaluate_postfix(expression): This method takes a postfix expression as input and evaluates it, returning the result.

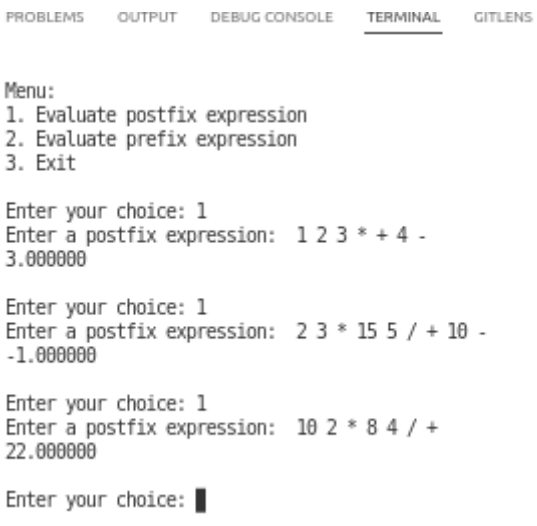
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX_SIZE 100

typedef struct {
int top;
double items[MAX_SIZE];
} Stack;

void initializeStack(Stack *s) {
s->top = -1;
}

int isEmpty(Stack *s) {
return s->top == -1;
}

int isFull(Stack *s) {
return s->top == MAX_SIZE - 1;
}
```

```
void push(Stack *s, double value) {
if (isFull(s)) {
printf("Stack overflow!\n");
exit(1);
}
s->items[++(s->top)] = value;
}

double pop(Stack *s) {
if (isEmpty(s)) {
printf("Stack underflow!\n");
exit(1);
}
return s->items[(s->top)--];
}

double evaluatePostfix(char *expression) {
Stack stack;
initializeStack(&stack);
int len = strlen(expression);
int i;
double operand1, operand2;

for (i = 0; i < len; i++) {
if (isdigit(expression[i])) {
double num = 0;
while (isdigit(expression[i])) {
num = num * 10 + (expression[i] - '0');
i++;
}
i--;
push(&stack, num);
} else if (expression[i] != ' ') {
operand2 = pop(&stack);
operand1 = pop(&stack);
switch (expression[i]) {
case '+':
push(&stack, operand1 + operand2);
break;
case '-':
push(&stack, operand1 - operand2);
break;
case '*':
push(&stack, operand1 * operand2);
break;
case '/':
push(&stack, operand1 / operand2);
break;
default:
printf("Invalid operator!\n");
exit(1);
```

```
}
}
}
return pop(&stack);
}

double evaluatePrefix(char *expression) {
Stack stack;
initializeStack(&stack);
int len = strlen(expression);
int i;
double operand1, operand2;

for (i = len - 1; i >= 0; i--) {
if (isdigit(expression[i])) {
double num = 0;
while (isdigit(expression[i])) {
num = num * 10 + (expression[i] - '0');
i--;
}
i++;
push(&stack, num);
} else if (expression[i] != ' ') {
operand1 = pop(&stack);
operand2 = pop(&stack);
switch (expression[i]) {
case '+':
push(&stack, operand2 + operand1);
break;
case '-':
push(&stack, operand2 - operand1);
break;
case '*':
push(&stack, operand2 * operand1);
break;
case '/':
push(&stack, operand2 / operand1);
break;
default:
printf("Invalid operator!\n");
exit(1);
}
}
}
return pop(&stack);
}

int main() {
int caseOperator;
char expression[100];
double result;
```

```
printf("Menu:\n");
printf("1. Evaluate postfix expression\n");
printf("2. Evaluate prefix expression\n");
printf("3. Exit\n");

while (1) {
printf("\nEnter your choice: ");
scanf("%d", &caseOperator);

switch (caseOperator) {
case 1:
printf("Enter a postfix expression: ");
scanf(" %[^\n]", expression);
printf("Result: %lf\n", evaluatePostfix(expression));
break;
case 2:
printf("Enter a prefix expression: ");
scanf(" %[^\n]", expression);
printf("Result: %lf\n", evaluatePrefix(expression));
break;
case 3:
printf("Exiting...\n");
exit(0);
default:
printf("Invalid operator\n");
break;
}
}

return 0;
}
```

Output for postfix evaluation



Output for prefix evaluation