Data Structures and algorithms

Submitted by:

Aryaveer Singh

102304064

3D12

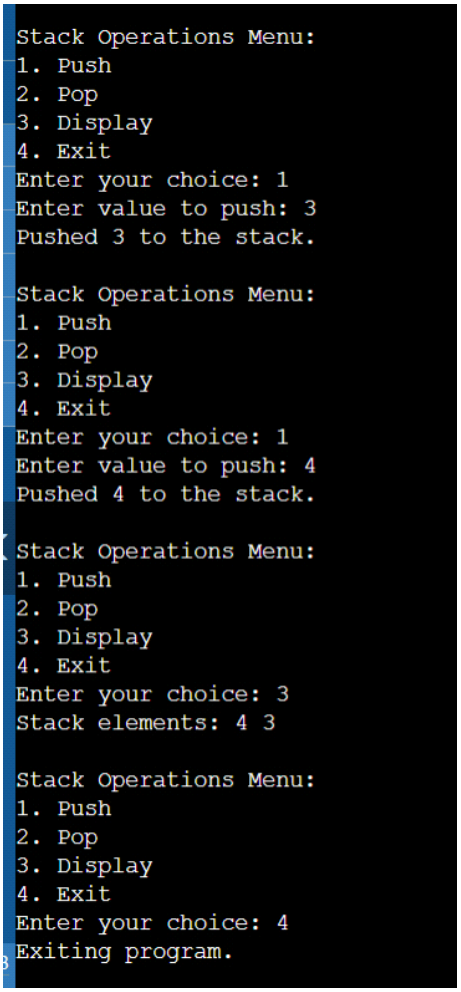
Assignment 4

Q 1: Write a menu driven program with 4 options (Push, Pop, Display, and Exit) to demonstrate the working of stacks using arrays.

```
#include <stdio.h>
#include <stdlib.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 more elements.\n");
    } else {
        stack[++top] = value;
        printf("Pushed %d to the stack.\n", value);
    }
}

void pop() {
```

```
    if (top < 0) {
        printf("Stack Underflow! Stack is empty.\n");
    } else {
        printf("Popped %d from the stack.\n", stack[top--]);
    }
}

void display() {
    if (top < 0) {
        printf("Stack is empty.\n");
    } else {
        printf("Stack elements: ");
        for (int i = top; i >= 0; i--) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}

int main() {
    int choice, value;

    do {
        printf("\nStack Operations Menu:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
```

```
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```



Q 2: Write a menu driven program with 4 options (Push, Pop, Display, and Exit) to demonstrate the working of stacks using linked-list.

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

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

struct Node *top_ll = NULL;
```

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

void pop_ll() {
    if (top_ll == NULL) {
        printf("Stack Underflow! Stack is empty.\n");
        return;
    }
    struct Node *temp = top_ll;
    top_ll = top_ll->next;
    printf("Popped %d from the stack.\n", temp->data);
    free(temp);
}

void display_ll() {
    if (top_ll == NULL) {
        printf("Stack is empty.\n");
        return;
    }
```

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

int main() {
    int choice, value;

    do {
        printf("\nLinked List Stack Operations Menu:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push_ll(value);
                break;
            case 2:
                pop_ll();
```

```
            break;
        case 3:
            display_ll();
            break;
        case 4:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
      }
   } while (choice != 4);

   return 0;
}
```

```
Linked List Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 3
Pushed 3 to the stack.

Linked List Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 2
Pushed 2 to the stack.

Linked List Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements: 2 3

Linked List Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 4
Exiting program.
```

Q 3: Write a program to convert infix expression into postfix expression using stack.

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

#define MAX_SIZE 100

char stack_infix[MAX_SIZE];
int top_infix = -1;

void push_infix(char op) {
    if (top_infix >= MAX_SIZE - 1) {
        printf("Stack Overflow\n");
        return;
    }
    stack_infix[++top_infix] = op;
}

char pop_infix() {
    if (top_infix < 0) {
        return '\0';
    }
    return stack_infix[top_infix--];
}

int precedence(char op) {
```

```
    if (op == '+' || op == '-') {
        return 1;
    }
    if (op == '*' || op == '/') {
        return 2;
    }
    return 0;
}

void infixToPostfix(char* infix, char* postfix) {
    int i, j;
    char op;

    for (i = 0, j = 0; infix[i] != '\0'; i++) {
        if (isalnum(infix[i])) {
            postfix[j++] = infix[i];
        } else if (infix[i] == '(') {
            push_infix(infix[i]);
        } else if (infix[i] == ')') {
            while (top_infix != -1 && stack_infix[top_infix] != '(') {
                postfix[j++] = pop_infix();
            }
            if (top_infix != -1 && stack_infix[top_infix] != '(') {
                printf("Invalid Expression\n");
                return;
            } else {
                pop_infix();
            }
        } else {
```

```
            while (top_infix != -1 && precedence(stack_infix[top_infix]) >= precedence(infix[i])) {
                postfix[j++] = pop_infix();
            }
            push_infix(infix[i]);
        }
    }

    while (top_infix != -1) {
        if (stack_infix[top_infix] == '(') {
            printf("Invalid Expression\n");
            return;
        }
        postfix[j++] = pop_infix();
    }
    postfix[j] = '\0';
}

int main() {
    char infix[MAX_SIZE];
    char postfix[MAX_SIZE];

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

    infixToPostfix(infix, postfix);
    printf("The postfix expression is: %s\n", postfix);

    return 0;
}
```

```
Enter an infix expression: 2+3*5
The postfix expression is: 235*+
```

Q 4: Write a program to convert infix expression into prefix expression using stack.

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

#define MAX_SIZE 100

char stack_prefix[MAX_SIZE];
int top_prefix = -1;

void push_prefix(char op) {
    if (top_prefix >= MAX_SIZE - 1) {
        printf("Stack Overflow\n");
        return;
    }
    stack_prefix[++top_prefix] = op;
}

char pop_prefix() {
    if (top_prefix < 0) {
```

```
        return '\0';
    }
    return stack_prefix[top_prefix--];
}

void reverseString(char* str) {
    int len = strlen(str);
    int i, j;
    char temp;
    for (i = 0, j = len - 1; i < j; i++, j--) {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
    }
}

int precedence_prefix(char op) {
    if (op == '+' || op == '-') {
        return 1;
    }
    if (op == '*' || op == '/') {
        return 2;
    }
    return 0;
}

void infixToPrefix(char* infix, char* prefix) {
    int i, j;
    char op;
```

```
    reverseString(infix);

    for (i = 0, j = 0; infix[i] != '\0'; i++) {
        if (infix[i] == '(') {
            infix[i] = ')';
        } else if (infix[i] == ')') {
            infix[i] = '(';
        }
    }

    for (i = 0; infix[i] != '\0'; i++) {
        if (isalnum(infix[i])) {
            prefix[j++] = infix[i];
        } else if (infix[i] == '(') {
            push_prefix(infix[i]);
        } else if (infix[i] == ')') {
            while (top_prefix != -1 && stack_prefix[top_prefix] != '(') {
                prefix[j++] = pop_prefix();
            }
            if (top_prefix != -1 && stack_prefix[top_prefix] != '(') {
                printf("Invalid Expression\n");
                return;
            } else {
                pop_prefix();
            }
        } else {
            while (top_prefix != -1 && precedence_prefix(stack_prefix[top_prefix]) >=
precedence_prefix(infix[i])) {
```

```
            prefix[j++] = pop_prefix();
        }
        push_prefix(infix[i]);
    }
  }

  while (top_prefix != -1) {
    if (stack_prefix[top_prefix] == '(') {
      printf("Invalid Expression\n");
      return;
    }
    prefix[j++] = pop_prefix();
  }
  prefix[j] = '\0';
  reverseString(prefix);
}

int main() {
  char infix[MAX_SIZE];
  char prefix[MAX_SIZE];

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

  infixToPrefix(infix, prefix);
  printf("The prefix expression is: %s\n", prefix);

  return 0;
}
```

```
Enter an infix expression: 34*5-87/4
The prefix expression is: -*345/874
```

Q 5: Write a program to evaluate a postfix expression using stack.

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

#define MAX_SIZE 100

int stack_eval[MAX_SIZE];
int top_eval = -1;

void push_eval(int value) {
    if (top_eval >= MAX_SIZE - 1) {
        printf("Stack Overflow\n");
        return;
    }
    stack_eval[++top_eval] = value;
}

int pop_eval() {
    if (top_eval < 0) {
        return -1;  Indicates error
    }
    return stack_eval[top_eval--];
}
```

```
int evaluatePostfix(char* postfix) {
    int i;
    for (i = 0; postfix[i] != '\0'; i++) {
        if (isdigit(postfix[i])) {
            push_eval(postfix[i] - '0');
        } else {
            int op2 = pop_eval();
            int op1 = pop_eval();
            switch (postfix[i]) {
                case '+': push_eval(op1 + op2); break;
                case '-': push_eval(op1 - op2); break;
                case '*': push_eval(op1 * op2); break;
                case '/': push_eval(op1 / op2); break;
                default: printf("Invalid operator\n"); return -1;
            }
        }
    }
    return pop_eval();
}

int main() {
    char postfix[MAX_SIZE];
    printf("Enter a postfix expression (single-digit operands): ");
    scanf("%s", postfix);

    int result = evaluatePostfix(postfix);
    if (result != -1) {
        printf("Result of the postfix expression: %d\n", result);
```

```
    }

    return 0;
}
```

```
Enter a postfix expression (single-digit operands): 6783+-*
Result of the postfix expression: -24
```