COMPUTER LAB

NAME: SOUMYADIP MAITY

ROLL NO.: 22053029

SEC: CSE-49

YEAR: 2022-23

ASSIGNMENT – 6

1. Write a menu driven program to perform the following operations of a stack using array by using suitable user defined functions for each case.

a) Check if the stack is empty
b) Display the contents of stack
c) Push
d) Pop

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

#defineMAX_SIZE10

struct Stack {
   int items[MAX_SIZE];
   int top;
};
void initialize(struct Stack *stack) {
   stack->top = -1;
}
int isEmpty(struct Stack *stack) {
   return stack->top == -1;
}
int isFull(struct Stack *stack) {
   return stack->top == MAX_SIZE - 1;
}
void push(struct Stack *stack, int value) {
   if (isFull(stack)) {
      printf("Stack is full. Cannot push %d.\n", value);
   } else {
      stack->top++;
      stack->items[stack->top] = value;
      printf("Pushed %d onto the stack.\n", value);
   }
}
void pop(struct Stack *stack) {
   if (isEmpty(stack)) {
      printf("Stack is empty. Cannot pop.\n");
   } else {
      int poppedValue = stack->items[stack->top];
      stack->top--;
      printf("Popped %d from the stack.\n", poppedValue);
   }
}
void display(struct Stack *stack) {
```

```
    if (isEmpty(stack)) {
        printf("Stack is empty.\n");
    } else {
        printf("Stack contents:\n");
        for (int i = 0; i <= stack->top; i++) {
            printf("%d ", stack->items[i]);
        }
        printf("\n");
    }
}

int main() {
    struct Stack stack;
    initialize(&stack);

    int choice, value;

    while (1) {
        printf("\nStack Menu:\n");
        printf("1. Check if the stack is empty\n");
        printf("2. Display the contents of stack\n");
        printf("3. Push an element onto the stack\n");
        printf("4. Pop an element from the stack\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                if (isEmpty(&stack)) {
                    printf("Stack is empty.\n");
                } else {
                    printf("Stack is not empty.\n");
                }
                break;
            case 2:
                display(&stack);
                break;
            case 3:
                printf("Enter the value to push: ");
                scanf("%d", &value);
                push(&stack, value);
                break;
            case 4:
                pop(&stack);
                break;
            case 5:
```

```
            printf("Exiting the program.\n");
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
      }
   }

   return 0;
}
```

2. Write a menu driven program to perform the following operations of a stack using linked list by using suitable user defined functions for each case.

a) Check if the stack is empty
b) Display the contents of stack
c) Push
d) Pop

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

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

struct Stack {
   struct Node* top;
};

void initialize(struct Stack* stack) {
   stack->top = NULL;
}

int isEmpty(struct Stack* stack) {
   return stack->top == NULL;
}
void push(struct Stack* stack, int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   if (newNode == NULL) {
      printf("Memory allocation failed. Cannot push %d.\n", value);
      return;
   }

   newNode->data = value;
```

```
    newNode->next = stack->top;
    stack->top = newNode;
    printf("Pushed %d onto the stack.\n", value);
}

void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty. Cannot pop.\n");
        return;
    }

    struct Node* poppedNode = stack->top;
    stack->top = stack->top->next;
    int poppedValue = poppedNode->data;
    free(poppedNode);
    printf("Popped %d from the stack.\n", poppedValue);
}

void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty.\n");
        return;
    }

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

int main() {
    struct Stack stack;
    initialize(&stack);

    int choice, value;

    while (1) {
        printf("\nStack Menu:\n");
        printf("1. Check if the stack is empty\n");
        printf("2. Display the contents of stack\n");
        printf("3. Push an element onto the stack\n");
        printf("4. Pop an element from the stack\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
```

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

        switch (choice) {
            case 1:
                if (isEmpty(&stack)) {
                    printf("Stack is empty.\n");
                } else {
                    printf("Stack is not empty.\n");
                }
                break;
            case 2:
                display(&stack);
                break;
            case 3:
                printf("Enter the value to push: ");
                scanf("%d", &value);
                push(&stack, value);
                break;
            case 4:
                pop(&stack);
                break;
            case 5:
                printf("Exiting the program.\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

3. WAP to convert an infix expression into its equivalent postfix notation using stack.

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

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

void initializeCharStack(struct CharStack** stack) {
    *stack = NULL;
}
```

```
void pushChar(struct CharStack** stack, char ch) {
   struct CharStack* newNode = (struct CharStack*)malloc(sizeof(struct CharStack));
   if (newNode == NULL) {
      printf("Memory allocation failed. Cannot push character.\n");
      exit(1);
   }
   newNode->data = ch;
   newNode->next = *stack;
   *stack = newNode;
}
char popChar(struct CharStack** stack) {
   if (*stack == NULL) {
      printf("Character stack is empty. Cannot pop character.\n");
      exit(1);
   }
   char ch = (*stack)->data;
   struct CharStack* temp = *stack;
   *stack = (*stack)->next;
   free(temp);
   return ch;
}
int isOperator(char ch) {
   return (ch == '+' || ch == '-' || ch == '*' || ch == '/');
}
void infixToPostfix(char* infixExpression) {
   struct CharStack* charStack;
   initializeCharStack(&charStack);

   int infixIndex = 0;
   int postfixIndex = 0;
   char postfixExpression[100];

   while (infixExpression[infixIndex] != '\0') {
      char currentChar = infixExpression[infixIndex];

      if (isalnum(currentChar)) {
         postfixExpression[postfixIndex++] = currentChar;
      } else if (isOperator(currentChar)) {
         while (charStack != NULL && isOperator(charStack->data) && charStack->data
!= '(') {
            postfixExpression[postfixIndex++] = popChar(&charStack);
         }
         pushChar(&charStack, currentChar);
      } else if (currentChar == '(') {
         pushChar(&charStack, currentChar);
      } else if (currentChar == ')') {
```

```
            while (charStack != NULL && charStack->data != '(') {
                postfixExpression[postfixIndex++] = popChar(&charStack);
            }
            if (charStack == NULL || charStack->data != '(') {
                printf("Invalid infix expression. Mismatched parentheses.\n");
                exit(1);
            } else {
                popChar(&charStack);
            }
        }

        infixIndex++;
    }

    while (charStack != NULL) {
        if (charStack->data == '(') {
            printf("Invalid infix expression. Mismatched parentheses.\n");
            exit(1);
        }
        postfixExpression[postfixIndex++] = popChar(&charStack);
    }

    postfixExpression[postfixIndex] = '\0';

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

int main() {
    char infixExpression[100];

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

    infixToPostfix(infixExpression);

    return 0;
}
```

4. WAP to reverse a stack with using extra stack.

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

struct Stack {
    int data;
    struct Stack* next;
};
```

```
void initialize(struct Stack** stack) {
   *stack = NULL;
}
int isEmpty(struct Stack* stack) {
   return stack == NULL;
}
void push(struct Stack** stack, int value) {
   struct Stack* newNode = (struct Stack*)malloc(sizeof(struct Stack));
   if (newNode == NULL) {
      printf("Memory allocation failed. Cannot push %d.\n", value);
      exit(1);
   }
   newNode->data = value;
   newNode->next = *stack;
   *stack = newNode;
}
int pop(struct Stack** stack) {
   if (isEmpty(*stack)) {
      printf("Stack is empty. Cannot pop.\n");
      exit(1);
   }
   int value = (*stack)->data;
   struct Stack* temp = *stack;
   *stack = (*stack)->next;
   free(temp);
   return value;
}
void display(struct Stack* stack) {
   struct Stack* current = stack;
   while (current != NULL) {
      printf("%d ", current->data);
      current = current->next;
   }
   printf("\n");
}
void reverseStack(struct Stack** originalStack) {
   struct Stack* reversedStack = NULL;

   while (!isEmpty(*originalStack)) {
      int value = pop(originalStack);
      push(&reversedStack, value);
   }

   *originalStack = reversedStack;
}

int main() {
```

```
    struct Stack* stack;
    initialize(&stack);
    push(&stack, 1);
    push(&stack, 2);
    push(&stack, 3);
    push(&stack, 4);

    printf("Original Stack: ");
    display(stack);

    reverseStack(&stack);

    printf("Reversed Stack: ");
    display(stack);

    return 0;
}
```

5. Write a program to represent a polynomial equation of single variable using single linked list and perform the addition of two polynomial equations.

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

struct Term {
    int coefficient;
    int exponent;
};
struct Node {
    struct Term term;
    struct Node* next;
};
struct Term createTerm(int coefficient, int exponent) {
    struct Term term;
    term.coefficient = coefficient;
    term.exponent = exponent;
    return term;
}

struct Node* createNode(struct Term term) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->term = term;
```

```
    newNode->next = NULL;
    return newNode;
}
void insertTerm(struct Node** poly, struct Term term) {
    struct Node* newNode = createNode(term);
    if (*poly == NULL) {
        *poly = newNode;
    } else {
        struct Node* current = *poly;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
}
void inputPolynomial(struct Node** poly) {
    int n, coefficient, exponent;
    printf("Enter the number of terms in the polynomial: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        printf("Enter coefficient and exponent of term %d: ", i + 1);
        scanf("%d %d", &coefficient, &exponent);
        insertTerm(poly, createTerm(coefficient, exponent));
    }
}
struct Node* addPolynomials(struct Node* poly1, struct Node* poly2) {
    struct Node* result = NULL;

    while (poly1 != NULL || poly2 != NULL) {
        struct Term term;
        if (poly1 != NULL && poly2 != NULL) {
            if (poly1->term.exponent == poly2->term.exponent) {
                term.coefficient = poly1->term.coefficient + poly2->term.coefficient;
                term.exponent = poly1->term.exponent;
                poly1 = poly1->next;
                poly2 = poly2->next;
            } else if (poly1->term.exponent > poly2->term.exponent) {
                term = poly1->term;
                poly1 = poly1->next;
            } else {
                term = poly2->term;
                poly2 = poly2->next;
            }
        } else if (poly1 != NULL) {
            term = poly1->term;
            poly1 = poly1->next;
```

```
        } else {
            term = poly2->term;
            poly2 = poly2->next;
        }
        insertTerm(&result, term);
    }

    return result;
}
void displayPolynomial(struct Node* poly) {
    if (poly == NULL) {
        printf("0\n");
        return;
    }

    while (poly != NULL) {
        printf("%dx^%d", poly->term.coefficient, poly->term.exponent);
        poly = poly->next;
        if (poly != NULL) {
            printf(" + ");
        }
    }
    printf("\n");
}
void freeLinkedList(struct Node* poly) {
    struct Node* current = poly;
    while (current != NULL) {
        struct Node* temp = current;
        current = current->next;
        free(temp);
    }
}

int main() {
    struct Node* poly1 = NULL;
    struct Node* poly2 = NULL;
    struct Node* result = NULL;

    printf("Enter the first polynomial equation:\n");
    inputPolynomial(&poly1);

    printf("Enter the second polynomial equation:\n");
    inputPolynomial(&poly2);

    result = addPolynomials(poly1, poly2);

    printf("Result of addition:\n");
```

```
    displayPolynomial(result);
    freeLinkedList(poly1);
    freeLinkedList(poly2);
    freeLinkedList(result);

    return 0;
}
```