DAY 7 LAB

1.. Write a C program to implement infix, prefix and postfix notations for arithmetic expressions using stack

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 10

char stack[MAX][MAX];

int top = -1;

void push(char *item) {

   if (top == MAX - 1) {

      printf("Stack Overflow\n");

      return;

   }

   top++;

   strcpy(stack[top], item);

}

char *pop() {

   if (top == -1) {

      printf("Stack Underflow\n");

      exit(1);

   }

   return stack[top--];

}

int isOperand(char ch) {

   return (ch >= '0' && ch <= '9') || (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');

}

int precedence(char ch) {

   if (ch == '+' || ch == '-') {

      return 1;

   } else if (ch == '*' || ch == '/') {
        return 2;
    } else {
        return 0;
    }
}
void infixToPostfix(char *infix, char *postfix) {
}
void infixToPrefix(char *infix, char *prefix) {
}
int main() {
    char infix[MAX], postfix[MAX], prefix[MAX];
    printf("Enter an infix expression: ");
    scanf("%s", infix);
    infixToPostfix(infix, postfix);
    printf("Postfix expression: %s\n", postfix);
    infixToPrefix(infix, prefix);
    printf("Prefix expression: %s\n", prefix);
    return 0;
}
```

OUTPUT:

```
Enter an infix expression: a+b*c(d/v)
Postfix expression:
Prefix expression: @
```

2. Write a C program to check if the parentheses in an expression are balanced using a stack. Extend the program to handle multiple types of parentheses (e.g., {}, [], ()).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100
```

```
char stack[MAX][MAX];
int top = -1;
void push(char *item) {
    if (top == MAX - 1) {
        printf("Stack Overflow\n");
        return;
    }
    top++;
    strcpy(stack[top], item);
}
char *pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
        exit(1);
    }
    return stack[top--];
}
int isOperand(char ch) {
    return (ch >= '0' && ch <= '9') || (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');
}
int precedence(char ch) {
    if (ch == '+' || ch == '-') {
        return 1;
    } else if (ch == '*' || ch == '/') {
        return 2;
    } else {
        return 0;
    }
}
void infixToPostfix(char *infix, char *postfix) {
}
```

```
void infixToPrefix(char *infix, char *prefix) {
}


int main() {
    char infix[MAX], postfix[MAX], prefix[MAX];


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


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


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


    return 0;
}
```

OUTPUT:

```
Enter an infix expression: a+b*c(u/v)
Postfix expression: @——————————————————————
Prefix expression:
```

3. Write a program to evaluate a postfix expression using a stack. The program should handle basic arithmetic operators (+, -, *, /).

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define MAX_SIZE 100
int stack[MAX_SIZE];
int top = -1;
void push(int item) {
    if (top >= MAX_SIZE - 1) {
        printf("Stack Overflow\n");
        exit(1);
    } else {
        stack[++top] = item;
    }
}
int pop() {
    if (top < 0) {
        printf("Stack Underflow\n");
        exit(1);
    } else {
        return stack[top--];
    }
}
int evaluatePostfix(char* exp) {
    int i = 0, operand1, operand2, result;
    char ch;
    while ((ch = exp[i++]) != '\0') {
        if (isdigit(ch)) {
            push(ch - '0');
        } else {
            operand2 = pop();
            operand1 = pop();
            switch(ch) {
                case '+':
                    push(operand1 + operand2);
                    break;
                case '-':
```

```
                push(operand1 - operand2);
                break;
            case '*':
                push(operand1 * operand2);
                break;
            case '/':
                push(operand1 / operand2);
                break;
          }
       }
    }
    result = pop();
    return result;
}
int main() {
    char exp[] = "82/3-";
    printf("Result of the postfix expression evaluation: %d\n", evaluatePostfix(exp));
    return 0;
}
```

OUTPUT:

```
Result of the postfix expression evaluation: 1
```

4. Write a C program to solve the Tower of Hanoi problem using recursion.

```
#include <stdio.h>
void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod) {
    if (n == 1) {
        printf("Move disk 1 from rod %c to rod %c\n", from_rod, to_rod);
        return;
    }
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
    printf("Move disk %d from rod %c to rod %c\n", n, from_rod, to_rod);
    towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}
int main() {
    int n = 3;
    towerOfHanoi(n, 'A', 'C', 'B');
    return 0;
}
```

OUTPUT:

```
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C
```