Aim ↬ To perform basic operations on Stacks.

Objectives ↬

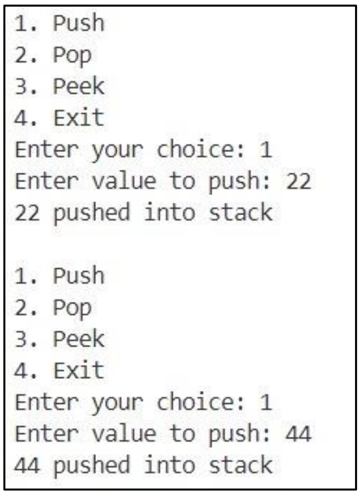
i. Write a program to implement stacks using Arrays.
ii. Write a program to implement Recursion using Stacks using one example each:
    a. Tail Recursion
    b. Non-tail recursion
    c. Nested Recursion (Ackermann's Function)
    d. Indirect Recursion
iii. Write a program to convert an Infix Expression to its equivalent Postfix notation using Stack.
iv. Write a program to evaluate a Postfix expression using Stack.
v. Write a program to implement the Tower of Hanoi problem using Stack.

Software Required ↬ Visual Studio Code

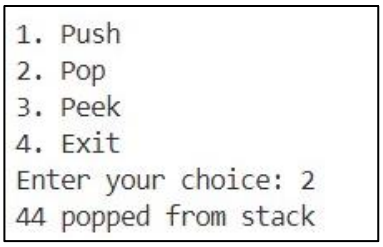
Code 1 ↬

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

#define MAX 5

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

int isFull() {
    return top == MAX - 1;
}

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

void push(int value) {
    if (isFull()) {
        printf("Stack Overflow\n");
    } else {
        top++;
        stack[top] = value;
        printf("%d pushed into stack\n", value);
    }
}

int pop() {
    if (isEmpty()) {
        printf("Stack Underflow\n");
        return -1;
    } else {
        int value = stack[top];
        top--;
        return value;
    }
}

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

int main() {
    int choice, value;

    while (1) {
        printf("\n1. Push\n2. Pop\n3. Peek\n4. Exit\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                value = pop();
                if (value != -1) {
                    printf("%d popped from stack\n", value);
                }
                break;
            case 3:
                value = peek();
                if (value != -1) {
                    printf("Top element is %d\n", value);
                }
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

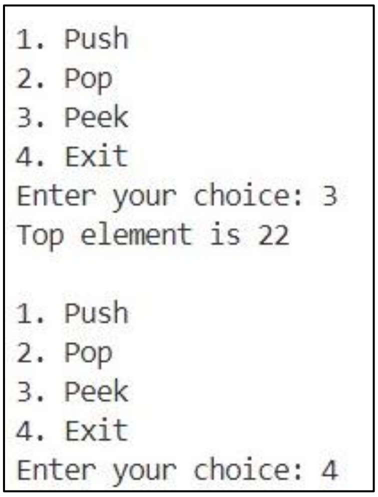
Output ↬



```
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 1
Enter value to push: 22
22 pushed into stack

1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 1
Enter value to push: 44
44 pushed into stack
```



```
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 2
44 popped from stack
```



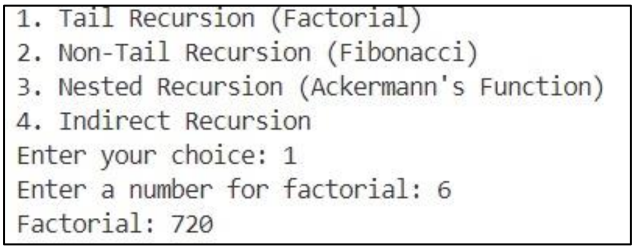
```
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 3
Top element is 22

1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 4
```

Code 2 ↬

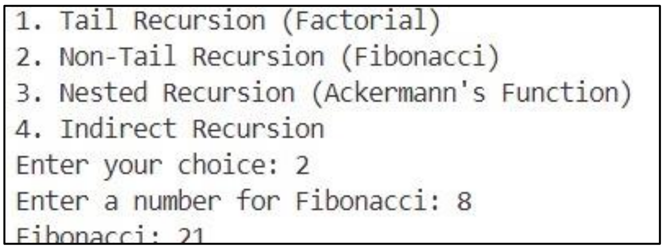
```
#include <stdio.h>

int tailRecursionFactorial(int n, int result) {
    if (n == 0) {
        return result;
    }
    return tailRecursionFactorial(n - 1, n * result);
}

int nonTailRecursionFibonacci(int n) {
    if (n <= 1) {
        return n;
    }
    return nonTailRecursionFibonacci(n - 1) + nonTailRecursionFibonacci(n - 2);
}

int nestedRecursionAckermann(int m, int n) {
    if (m == 0) {
        return n + 1;
    } else if (n == 0) {
        return nestedRecursionAckermann(m - 1, 1);
    }
    return nestedRecursionAckermann(m - 1, nestedRecursionAckermann(m, n -
1));
}

void indirectRecursionFunctionA(int n);
void indirectRecursionFunctionB(int n);

void indirectRecursionFunctionA(int n) {
    if (n > 0) {
        printf("%d ", n);
        indirectRecursionFunctionB(n - 1);
    }
}
```

```
void indirectRecursionFunctionB(int n) {
    if (n > 1) {
        printf("%d ", n);
        indirectRecursionFunctionA(n / 2);
    }
}

int main() {
    int choice, n, m;

    printf("1. Tail Recursion (Factorial)\n2. Non-Tail Recursion (Fibonacci)\n");
    printf("3. Nested Recursion (Ackermann's Function)\n4. Indirect Recursion\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter a number for factorial: ");
            scanf("%d", &n);1
            printf("Factorial: %d\n", tailRecursionFactorial(n, 1));
            break;
        case 2:
            printf("Enter a number for Fibonacci: ");
            scanf("%d", &n);
            printf("Fibonacci: %d\n", nonTailRecursionFibonacci(n));
            break;
        case 3:
            printf("Enter values for Ackermann's Function (m and n): ");
            scanf("%d %d", &m, &n);
            printf("Ackermann's Function: %d\n", nestedRecursionAckermann(m, n));
            break;
        case 4:
            printf("Enter a number for indirect recursion: ");
            scanf("%d", &n);
            printf("Indirect Recursion Output: ");
            indirectRecursionFunctionA(n);
            printf("\n");
            break;
        default:
            printf("Invalid choice.\n");
    }

    return 0;
}
```

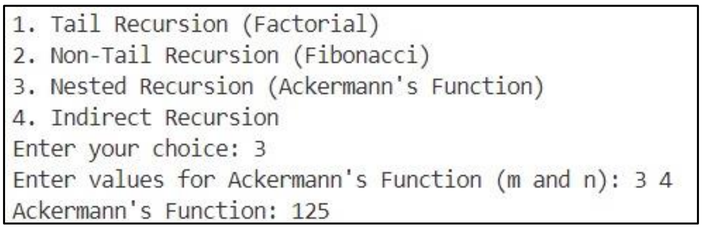
Output ↬



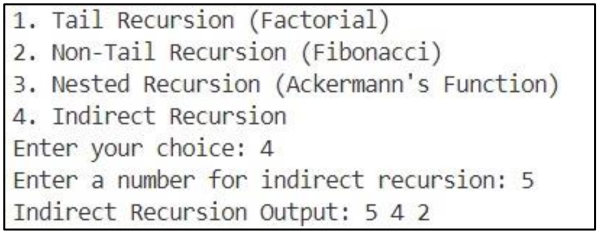
```
1. Tail Recursion (Factorial)
2. Non-Tail Recursion (Fibonacci)
3. Nested Recursion (Ackermann's Function)
4. Indirect Recursion
Enter your choice: 1
Enter a number for factorial: 6
Factorial: 720
```



```
1. Tail Recursion (Factorial)
2. Non-Tail Recursion (Fibonacci)
3. Nested Recursion (Ackermann's Function)
4. Indirect Recursion
Enter your choice: 2
Enter a number for Fibonacci: 8
Fibonacci: 21
```



```
1. Tail Recursion (Factorial)
2. Non-Tail Recursion (Fibonacci)
3. Nested Recursion (Ackermann's Function)
4. Indirect Recursion
Enter your choice: 3
Enter values for Ackermann's Function (m and n): 3 4
Ackermann's Function: 125
```



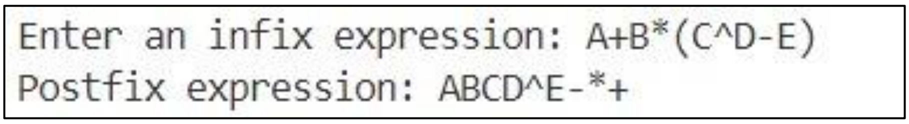
```
1. Tail Recursion (Factorial)
2. Non-Tail Recursion (Fibonacci)
3. Nested Recursion (Ackermann's Function)
4. Indirect Recursion
Enter your choice: 4
Enter a number for indirect recursion: 5
Indirect Recursion Output: 5 4 2
```

Code 3 ↬

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

#define MAX 100

char stack[MAX];
int top = -1;

void push(char x) {
    if (top == MAX - 1) {
        printf("Stack Overflow\n");
    } else {
        stack[++top] = x;
    }
}
```

```
char pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
        return -1;
    } else {
        return stack[top--];
    }
}

int precedence(char x) {
    if (x == '+' || x == '-') {
        return 1;
    } else if (x == '*' || x == '/') {
        return 2;
    } else if (x == '^') {
        return 3;
    } else {
        return 0;
    }
}

int isOperator(char x) {
    return (x == '+' || x == '-' || x == '*' || x == '/' || x == '^');
}

void infixToPostfix(char* exp) {
    int i = 0;
    char x;

    while (exp[i] != '\0') {
        if (isalnum(exp[i])) {
            // If the character is an operand, add it to the output
            printf("%c", exp[i]);
        } else if (exp[i] == '(') {
            // If the character is '(', push it to stack
            push(exp[i]);
        } else if (exp[i] == ')') {
            // If the character is ')', pop and output until '(' is found
            while ((x = pop()) != '(') {
                printf("%c", x);
            }
        } else if (isOperator(exp[i])) {
            // If the character is an operator
            while (top != -1 && precedence(stack[top]) >= precedence(exp[i])) {
                printf("%c", pop());
            }
            push(exp[i]);
        }
        i++;
    }

    // Pop all the operators from the stack
    while (top != -1) {
        printf("%c", pop());
    }
}

int main() {
    char exp[MAX];
    printf("Enter an infix expression: ");
    scanf("%s", exp);

    printf("Postfix expression: ");
    infixToPostfix(exp);

    return 0;
}
```

Output ↬



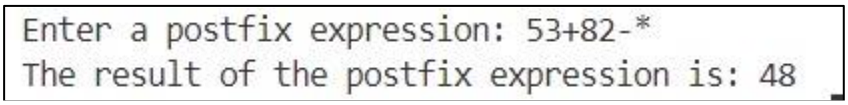
```
Enter an infix expression: A+B*(C^D-E)
Postfix expression: ABCD^E-*+
```

Code 4 ↬

```
#include <stdio.h>
#include <ctype.h>  // For isdigit() function

#define MAX 100

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

// Function to push an element onto the stack
void push(int x) {
    if (top == MAX - 1) {
        printf("Stack Overflow\n");
    } else {
        stack[++top] = x;
    }
}

// Function to pop an element from the stack
int pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
        return -1;
    } else {
        return stack[top--];
    }
}

// Function to evaluate the postfix expression
int evaluatePostfix(char* exp) {
    int i = 0;
    int val1, val2, result;

    while (exp[i] != '\0') {
        // If the character is an operand (digit), push it to the stack
        if (isdigit(exp[i])) {
            push(exp[i] - '0'); // Convert char to int
```

```c
        } else {
            // If the character is an operator, pop two operands and perform the operation
            val2 = pop();
            val1 = pop();

            switch (exp[i]) {
                case '+':
                    result = val1 + val2;
                    break;
                case '-':
                    result = val1 - val2;
                    break;
                case '*':
                    result = val1 * val2;
                    break;
                case '/':
                    result = val1 / val2;
                    break;
            }
            push(result); // Push the result back onto the stack
        }
        i++;
    }

    // The final result will be at the top of the stack
    return pop();
}

int main() {
    char exp[MAX];
    printf("Enter a postfix expression: ");
    scanf("%s", exp);

    int result = evaluatePostfix(exp);
    printf("The result of the postfix expression is: %d\n", result);
    return 0;
}
```

Output ↬



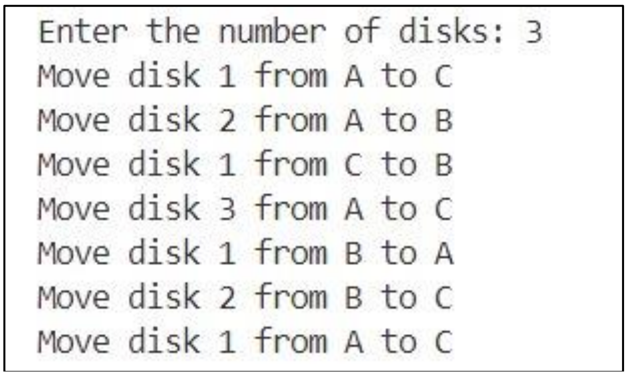
```
Enter a postfix expression: 53+82-*
The result of the postfix expression is: 48
```

Code 5 ↬

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

#define MAX 100

struct Stack {
    int top;
    int items[MAX];
};

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

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

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

void push(struct Stack* s, int item) {
    if (!isFull(s)) {
        s->items[++(s->top)] = item;
    }
}
```

```
int pop(struct Stack* s) {
    if (!isEmpty(s)) {
        return s->items[(s->top)--];
    }
    return -1;
}

void moveDisk(char fromPeg, char toPeg, int disk) {
    printf("Move disk %d from %c to %c\n", disk, fromPeg, toPeg);
}

void moveBetweenPoles(struct Stack* src, struct Stack* dest, char s, char d) {
    int pole1TopDisk = isEmpty(src) ? -1 : pop(src);
    int pole2TopDisk = isEmpty(dest) ? -1 : pop(dest);

    if (pole1TopDisk == -1) {
        push(src, pole2TopDisk);
        moveDisk(d, s, pole2TopDisk);
    } else if (pole2TopDisk == -1) {
        push(dest, pole1TopDisk);
        moveDisk(s, d, pole1TopDisk);
    } else if (pole1TopDisk > pole2TopDisk) {
        push(src, pole1TopDisk);
        push(src, pole2TopDisk);
        moveDisk(d, s, pole2TopDisk);
    } else {
        push(dest, pole2TopDisk);
        push(dest, pole1TopDisk);
        moveDisk(s, d, pole1TopDisk);
    }
}

void towerOfHanoi(int num_of_disks, struct Stack* src, struct Stack* aux, struct
Stack* dest) {
    int total_moves = pow(2, num_of_disks) - 1;
    char s = 'A', d = 'C', a = 'B';
    if (num_of_disks % 2 == 0) {
        char temp = d;
        d = a;
        a = temp;
    }

    for (int i = num_of_disks; i >= 1; i--) {
        push(src, i);
    }

    for (int i = 1; i <= total_moves; i++) {
        if (i % 3 == 1)
            moveBetweenPoles(src, dest, s, d);
        else if (i % 3 == 2)
            moveBetweenPoles(src, aux, s, a);
        else if (i % 3 == 0)
            moveBetweenPoles(aux, dest, a, d);
    }
}

int main() {
    int num_of_disks;
    printf("Enter the number of disks: ");
    scanf("%d", &num_of_disks);

    struct Stack src, aux, dest;
    initStack(&src);
    initStack(&aux);
    initStack(&dest);

    towerOfHanoi(num_of_disks, &src, &aux, &dest);

    return 0;
}
```

Output ↬



```
Enter the number of disks: 3
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

Result ↬

The programs demonstrated:

- **Stack Operations Using Arrays**: Successful implementation of push, pop, and peek operations.
- **Recursion Using Stacks**: Effective examples of tail recursion (factorial), non-tail recursion (Fibonacci), nested recursion (Ackermann’s function), and indirect recursion.
- **Infix to Postfix Conversion**: Accurate conversion of infix expressions to postfix notation.
- **Postfix Evaluation**: Correct evaluation of postfix expressions.
- **Tower of Hanoi**: Solved using stacks to manage disk movements

Conclusion ↬

The experiment effectively illustrated stack operations, recursion, expression conversion, and problem-solving techniques, enhancing understanding of stack data structures and their applications lists, improving understanding of dynamic memory management, and list manipulations.

Precautions ↬

- Validate inputs to avoid errors.
- Manage memory properly to prevent leaks.
- Handle empty or single-element cases correctly.
- Implement error handling to avoid invalid states.