Post Lab

-Aaryan Sharma
-16010123012

1) **List 5 Applications of Stack Data Structures.**
   Expression Evaluation and Conversion
   Backtracking Algorithms
   Browser History Navigation
   Undo Mechanism in Text Editors
   String Reversal

2) **Convert the given Infix Expression into Postfix Expression using Stack: (A-B/C)*(D*E-F)**

   Infix to Postfix-
   Code -

```
#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX 100

char stack[MAX];

int top = -1;

int push(char x) {

   if (top >= MAX - 1) {

      printf("Stack Overflow\n");

      return 0;

   } else {

      stack[++top] = x;

      return 1;

   }

}
```

```
char pop() {
    if (top < 0) {
        printf("Stack Underflow\n");
        return -1;
    } else {
        return stack[top--];
    }
}
int precedence(char x) {
    switch (x) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        default:
            return 0;
    }
}
void infixToPostfix(char* exp) {
    char* e = exp;
    char x;
    while (*e != '\0') {
        if (isalnum(*e)) {
```

```
            printf("%c", *e);
        } else if (*e == '(') {
            push(*e);
        } else if (*e == ')') {
            while ((x = pop()) != '(') {
                printf("%c", x);
                if (top == -1) break;
            }
        } else {
            while (top != -1 && precedence(stack[top]) >= precedence(*e)) {
                printf("%c", pop());
            }
            push(*e);
        }
        e++;
    }

    while (top != -1) {
        printf("%c", pop());
    }
    printf("\n");
}
int main() {
    char exp[MAX];
    printf("Enter an infix expression: ");
    if (scanf("%99s", exp) != 1) {
        printf("Invalid input\n");
```

```
        return 1;
    }
    printf("Postfix expression: ");
    infixToPostfix(exp);
    return 0;
}
```

```
Enter an infix expression: (A-B/C)*(D*E-F)
Postfix expression: ABC/-DE*F-*


=== Code Execution Successful ===
```

Infix to Postfix-

Code-

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.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 < 0) {
        printf("Stack Underflow\n");
        return -1;
    } else {
        return stack[top--];
    }
}

int precedence(char x) {
    switch (x) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
    }
    return 0;
}

void reverse(char *exp) {
    int length = strlen(exp);
    int i;
    for (i = 0; i < length / 2; i++) {
        char temp = exp[i];
        exp[i] = exp[length - i - 1];
        exp[length - i - 1] = temp;
```

```
    }
}

void replaceParentheses(char *exp) {
    int i;
    for (i = 0; exp[i]; i++) {
        if (exp[i] == '(') {
            exp[i] = ')';
        } else if (exp[i] == ')') {
            exp[i] = '(';
        }
    }
}

void infixToPrefix(char *exp) {
    char *e, x;
    reverse(exp);
    replaceParentheses(exp);
    e = exp;

    char result[MAX];
    int resIndex = 0;

    while (*e != '\0') {
        if (isalnum(*e)) {
            result[resIndex++] = *e;
        } else if (*e == '(') {
            push(*e);
        } else if (*e == ')') {
            while ((x = pop()) != '(') {
                result[resIndex++] = x;
```

```
            }
        } else {
            while (top != -1 && precedence(stack[top]) > precedence(*e)) {
                result[resIndex++] = pop();
            }
            push(*e);
        }
        e++;
    }

    while (top != -1) {
        result[resIndex++] = pop();
    }
    result[resIndex] = '\0';

    reverse(result);
    printf("%s\n", result);
}

int main() {
    char exp[MAX];
    printf("Enter an infix expression: ");
    scanf("%s", exp);
    printf("Prefix expression: ");
    infixToPrefix(exp);
    return 0;
}
```

```
Enter an infix expression: (A-B/C)*(D*E-F)
Prefix expression: *-A/BC-*DEF


=== Code Execution Successful ===
```

3) **Explain How stack can be used in both Nested Function calls and Recursion using suitable examples for each. Further Define Activation Records used for Function Calling.**

Nested Function Calls - When a function is called, it creates a new execution context. This context includes local variables, parameters, and the return address (where to resume execution after the function returns). To keep track of these multiple execution contexts, the system uses a stack.
Pushing onto the stack: When a function is called, its activation record is pushed onto the stack. This record contains all the information about the function's context.
Popping from the stack: When a function returns, its activation record is popped off the stack, and execution resumes at the return address.

Recursion - It is a technique where a function calls itself directly or indirectly. The stack is essential for managing recursive calls.

- Base case: The recursion stops when it reaches a base case.
- Recursive case: The function calls itself with modified arguments, pushing a new activation record onto the stack.
- Unwinding: As the base case is reached, the function returns, and activation records are popped off the stack in reverse order.

An **activation record** (or stack frame) is a data structure used to store information about a function's execution context. They are stored on the call stack and used to manage function calls and returns. They allow the program to keep track of multiple function calls simultaneously and ensure that each function has its own independent environment. It typically contains:

- Return address: The address where the program should continue execution after the function returns.
- Parameters: The values passed to the function.
- Local variables: Variables declared within the function.
- Temporary variables: Variables used for intermediate calculations.
- Return value: The value returned by the function.