ASSIGNMENT 8

```
#include <iostream>
#include <algorithm>
using namespace std;
struct Node {
    char data;
    Node* next;
    Node(char val) : data(val), next(nullptr) {}
};

class LinkedListStack {
private:
    Node* top;

public:
    LinkedListStack() : top(nullptr) {}

    void push(char val) {
        Node* newNode = new Node(val);
        newNode->next = top;
        top = newNode;
    }

    char pop() {
        if (isEmpty()) {
            cerr << "Error: Stack is empty.\n";
            return '\0';
        }

        char poppedValue = top->data;
        Node* temp = top;
        top = top->next;
        delete temp;
        return poppedValue;
    }

    char peek() {
        if (isEmpty()) {
            cerr << "Error: Stack is empty.\n";
            return '\0';
```

```
        }
        return top->data;
    }

    bool isEmpty() {
        return (top == nullptr);
    }
};

bool Operator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
}

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

string infixToPostfix(const string& infix) {
    string input = '(' + infix + ')';
    int l = input.size();
    LinkedListStack charStack;
    string output;

    for (int i = 0; i < l; i++) {
        if (isalpha(input[i]) || isdigit(input[i]))
            output += input[i];
        else if (input[i] == '(')
            charStack.push('(');
        else if (input[i] == ')') {
            while (charStack.peek() != '(') {
                output += charStack.pop();
            }
            charStack.pop();
        }
        else {
            while (!charStack.isEmpty() && getPriority(input[i]) <=
getPriority(charStack.peek())) {
```

```
            output += charStack.pop();
        }
        charStack.push(input[i]);
    }
  }
  return output;
}

string infixToPrefix(const string& infix) {
  string reversedInfix = infix;
  reverse(reversedInfix.begin(), reversedInfix.end());

  for (char& c : reversedInfix) {
    if (c == '(')
      c = ')';
    else if (c == ')')
      c = '(';
  }

  string prefix = infixToPostfix(reversedInfix);

  reverse(prefix.begin(), prefix.end());

  return prefix;
}

string prefixToPostfix(const string& prefix) {
  LinkedListStack operandStack;

  int len = prefix.length();
  for (int i = len - 1; i >= 0; i--) {
    if (isalpha(prefix[i]) || isdigit(prefix[i])) {
      operandStack.push(prefix[i]);
    } else {
      char operand1 = operandStack.pop();
      char operand2 = operandStack.pop();
      operandStack.push(operand1 + operand2 + prefix[i]);
    }
  }
```

```
    return operandStack.pop();
}

int main() {
    char choice;
    do {
        cout << "Expression Conversion and Evaluation Program\n";
        cout << "1. Infix to Prefix\n";
        cout << "2. Prefix to Postfix\n";
        cout << "3. Prefix to Infix\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";

        cin >> choice;
        cin.ignore();

        if (choice == '4') {
            cout << "Exiting program.\n";
            break;
        }

        string expression;
        cout << "Enter the expression: ";
        getline(cin, expression);

        switch (choice) {
            case '1':
                cout << "Infix to Prefix: " << infixToPrefix(expression) << "\n";
                break;
            case '2':
                cout << "Prefix to Postfix: " << prefixToPostfix(expression) << "\n";
                break;
            case '3':
                cout << "Prefix to Infix: " << prefixToPostfix(expression) << "\n";
                break;
            default:
                cout << "Invalid choice\n";
        }

    } while (choice != '4');
```

```
    return 0;
}
```

```
Expression Conversion and Evaluation Program
1. Infix to Prefix
2. Prefix to Postfix
3. Prefix to Infix
4. Exit
Enter your choice: 1
Enter the expression: A+B
Infix to Prefix: +AB
Expression Conversion and Evaluation Program
1. Infix to Prefix
2. Prefix to Postfix
3. Prefix to Infix
4. Exit
Enter your choice: 2
Enter the expression: +*ABC
Prefix to Postfix: AB*C+
Expression Conversion and Evaluation Program
1. Infix to Prefix
2. Prefix to Postfix
3. Prefix to Infix
4. Exit
```