/*Implement stack as an abstract data type using singly linked list and use this ADT for conversion of infix expression to prefix*/

```
#include <iostream>
#include <string>

using namespace std;

// Node class for singly linked list
class Node {
public:
    char data;
    Node* next;

    Node(char value) {
        data = value;
        next = NULL;
    }
};

// Stack class using singly linked list
class Stack {
private:
    Node* top;

public:
    Stack() {
        top = NULL;
    }
```

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

    char pop() {
        if (isEmpty()) {
            return '\0';
        }
        char value = top->data;
        Node* temp = top;
        top = top->next;
        delete temp;
        return value;
    }

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

// Function to check if a character is an operand
bool isOperand(char c) {
    return (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z');
}

// Function to check the precedence of operators
int precedence(char c) {
```

```
    if (c == '+' || c == '-') return 1;
    if (c == '*' || c == '/') return 2;
    return 0;
}

// Function to convert infix expression to prefix expression
string infixToPrefix(string infix) {
    Stack stack;
    string prefix = "";
    int length = infix.length();

    for (int i = length - 1; i >= 0; i--) {
        char c = infix[i];

        if (isOperand(c)) {
            prefix = c + prefix;
        } else if (c == ')') {
            stack.push(c);
        } else if (c == '(') {
            while (!stack.isEmpty() && stack.pop() != ')') {
                prefix = stack.pop() + prefix;
            }
        } else {
            while (!stack.isEmpty() && precedence(c) < precedence(stack.pop())) {
                prefix = stack.pop() + prefix;
            }
            stack.push(c);
        }
    }
```

```
    while (!stack.isEmpty()) {
        prefix = stack.pop() + prefix;
    }

    return prefix;
}

int main() {
    string infixExpression;
    cout << "Enter infix expression: ";
    cin >> infixExpression;

    string prefixExpression = infixToPrefix(infixExpression);
    cout << "Prefix expression: " << prefixExpression << endl;

    return 0;
}
```