NAME – AARYAN BAIRAGI

ROLL NUMBER – 27002

CLASS – SE IT B

#include <iostream>

using namespace std;

#include<malloc.h>

struct Node {

    char data;

    struct Node \*prev;

};

struct Node \*tail = NULL;

void Push (char c){

    if(tail == NULL){

        struct Node \*newnode;

        newnode = (struct Node\*)malloc(sizeof(struct Node));

        newnode -> data = c;

        newnode -> prev = NULL;

        tail = newnode;

    }

    else {

        struct Node \*newnode;

        newnode = (struct Node\*)malloc(sizeof(struct Node));

        newnode -> data = c;

        newnode -> prev = tail;

        tail = newnode;

    }

}

void Pop () {

    struct Node \*last\_node = tail;

    tail = tail -> prev;

    free(last\_node);

}

void Display () {

    struct Node \*ptr = tail;

    while (ptr != NULL) {

        cout<<ptr -> data;

        ptr = ptr -> prev;

    }

}

string reverse (string infix) {

    string new\_infix;

    for (int n = infix.length()-1; n>=0; n--) {

        new\_infix.push\_back(infix[n]);

    }

    return new\_infix;

}

bool isOperator (char c) {

    if (c == '+' || c == '-' || c == '/' || c == '\*' || c == '^') {

        return true;

    }

    else {

        return false;

    }

}

int precedence (char c) {

    if (c == '^')

    return 3;

    else if (c == '\*' || c == '/')

    return 2;

    else if (c == '+' || c == '-')

    return 1;

    else

    return -1;

}

string InfixToPostfix (string infix) {

    string postfix;

    for (int i = 0; i < infix.length(); i++) {

        if ((infix[i] >= 'a' && infix[i] <= 'z') || (infix[i] >= 'A' && infix[i] <= 'Z' )) {

            postfix += infix[i];

        }

        else if (infix[i] == '('){

            Push(infix[i]);

        }

        else if (infix[i] == ')'){

            while (tail -> data != '(' &&  tail != NULL) {

                postfix += tail -> data;

                Pop();

            }

            if (tail -> data == '(') {

                Pop();

            }

        }

        else if (isOperator(infix[i])) {

            if (tail == NULL) {

                Push (infix[i]);

            }

            else {

                if (precedence(infix[i]) >  precedence(tail -> data)) {

                    Push (infix[i]);

                }

                else {

                    while ((tail != NULL) && (precedence(infix[i]) <= precedence(tail -> data))) {

                        postfix =+ tail -> data;

                        Pop ();

                    }

                    Push (infix[i]);

                }

            }

        }

    }

    while (tail != NULL) {

        postfix += tail -> data;

        Pop ();

    }

    return postfix;

}

string InfixToPrefix (string rev) {

    string prefix;

    string infix = reverse(rev);

    for (int i = 0; i < infix.length(); i++) {

        if ((infix[i] >= 'a' && infix[i] <= 'z') || (infix[i] >= 'A' && infix [i] <= 'Z')) {

            prefix += infix[i];

        }

        else if (infix[i] == ')') {

            Push(infix[i]);

        }

        else if (infix[i] == '(') {

            while ((tail -> data != ')') && (tail != NULL)) {

                prefix += tail -> data;

                Pop();

            }

            if (tail -> data == ')') {

                Pop();

            }

        }

        else if (isOperator(infix[i])) {

            if (tail == NULL) {

                Push(infix[i]);

            }

            else {

                if (precedence(infix[i]) >= precedence(tail -> data)) {

                    Push(infix[i]);

                }

                else {

                    while ((tail != NULL) && (precedence(infix[i]) < precedence(tail -> data))) {

                        prefix += tail -> data;

                        Pop();

                    }

                    Push(infix[i]);

                }

            }

        }

    }

    while (tail != NULL) {

        prefix += tail -> data;

        Pop();

    }

    return reverse(prefix);

}

int main () {

    string Infix;

    cout<<"Enter the Infix Expression: ";

    cin >> Infix;

    cout << "\nThe Prefix Expression is: ";

    cout << InfixToPrefix(Infix);

    cout << "\nThe Postfix Expression is: ";

    cout << InfixToPostfix(Infix);

}