```
//==============================================================================
// Name        : 21465_Pract10.cpp
// Author      : Chaitanya Paraskar
// Roll No.    : 21465
// Aim         : Implement C++ program for expression conversion as infix to
postfix and its evaluation using stack based on given
//               conditions : 1. Operands and operator, both must be single
character.
//                            2. Input Postfix expression must be in a desired
format.
//                            3. Only '+', '-', '*' and '/ ' operators are
expected.
//==============================================================================

#include <iostream>
using namespace std;

class Stack
{
private:
    char arr[20];
    int top = -1;

public:
    void display()
    {
        cout << "Stack : ";
        for (int i = 0; i < 20; i++)
        {
            char ch = arr[i];
            cout << ch << " ";
        }
        cout << "\n";
    }

    void push(char ch)
    {
        if (top < 19)
        {
            top++;
            arr[top] = ch;
        }
        else
        {
            cout << "Stack is Full !!" << endl;
        }
    }

    char pop()
    {
        char ch = arr[top];
        top--;
        return ch;
    }
    char getTop()
    {
        return arr[top];
    }

    bool isEmpty()
    {
        if (top == -1)
        {
            return true;
        }

        return false;
    }

    void clear()
    {
        while (this->top != -1)
            this->pop();
    }
};

class Node
{
public:
    char key;
    int val;
    Node *next;
};

class LinkedList
{
private:
    Node *start;

public:
    void add(char var, int val)
    {
        Node *n = new Node();
        n->key = var;
        n->val = val;

        n->next = start;
        start = n;
    }

    int getVal(char ch)
    {
        Node *ptr = this->start;

        while (ptr != NULL)
        {
            if (ptr->key == ch)
```

```
                break;

            ptr = ptr->next;
        }

        return ptr->val;
    }

    bool in(char n)
    {
        Node *ptr = this->start;

        while (ptr != NULL)
        {
            if (ptr->key == n)
                return true;
            ptr = ptr->next;
        }

        return false;
    }

    int getFirst()
    {
        return this->start->val;
    }

    void display()
    {
        cout << "LinkedList ";
        Node *ptr = this->start;

        while (ptr != NULL)
        {
            cout << "-> (" << ptr->key << ", " << ptr->val << ")";
            ptr = ptr->next;
        }
        cout << " -> NULL" << endl;
    }
};

class Expr
{
private:
    string inex = "";
    string postex = "";
    Stack *stack;
    LinkedList *list;
    static char custChar;

public:
    Expr()
    {
        this->stack = new Stack();
        this->list = new LinkedList();

        cout << "Enter Your Equation : ";
        getline(cin, this->inex);

        cout << "Infix Expression : " << this->inex << endl;
        this->toPost();
        cout << "Postfix Expression : " << this->postex << endl;
        this->eval();
    }

    static int prec(char ch)
    {
        int pr = 0;

        switch (ch)
        {
        case '*':
            pr = 2;
            break;
        case '-':
            pr = 1;
            break;
        case '+':
            pr = 1;
            break;
        case '/':
            pr = 2;
            break;
        case '%':
            pr = 2;
            break;
        case '^':
            pr = 3;
            break;
        }

        return pr;
    }

    void assoc(char ch)
    {
        if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '%')
        {
            char c = this->stack->pop();
            this->postex += c;
            this->stack->push(ch);
        }
        else if (ch == '^')
        {
            this->stack->push(ch);
        }
    }
    void toPost()
    {
        int l = this->inex.length(), i = 0;

        while (i < l)
        {

            char ch = this->inex[i];

            if (ch == '(')
            {
                this->stack->push(ch);
            }
            else if (ch == ')')
            {
                while (this->stack->getTop() != '(')
                {
                    char c = this->stack->pop();
                    postex += c;
                }

                this->stack->pop();
            }
            else if (isalpha(ch))
            {
                postex += ch;

                if (!this->list->in(ch))
                {
                    int val;

                    cout << "Enter Value for variable " << ch << " : ";
                    cin >> val;

                    list->add(ch, val);
                }
            }
            else
            {
                int pch = Expr::prec(ch);
                int ptop = Expr::prec(this->stack->getTop());

                if (pch > ptop)
                {
                    this->stack->push(ch);
                }
                else if (pch < ptop)
                {
                    char c = this->stack->pop();
                    postex += c;

                    //  This Line causes the loop to reiterate for same char in
the expr
                    //  without incrementing value of i.
                    continue;
                }
                else
                {
                    this->assoc(ch);
                }
            }

            cout << "Char : " << ch << endl;
            this->stack->display();
            this->list->display();

            i++;
        }

        while (!this->stack->isEmpty())
        {
            char c = this->stack->pop();

            postex += c;
        }
    }

    void eval()
    {
        this->stack->clear();

        int l = this->postex.length(), i = 0;

        while (i < l)
        {
            char ch = this->postex[i];

            if (isalpha(ch))
            {
                this->stack->push(ch);
            }
            else
            {
                char a = this->stack->pop(), b = this->stack->pop();

                int aval = this->list->getVal(a);
                int bval = this->list->getVal(b);

                int res = Expr::perfOp(aval, bval, ch);

                this->list->add(Expr::custChar, res);
                this->stack->push(Expr::custChar);
                Expr::custChar = (char)((int)Expr::custChar + 1);
            }

            cout << "Char : " << ch << endl;
            this->stack->display();
            this->list->display();
```

```cpp
            i++;
        }

        int res = this->list->getFirst();

        cout << "Evaluation of this Equation is : " << res << endl;
    }

    static int perfOp(int a, int b, char ch)
    {
        int res = 0;

        switch (ch)
        {
        case '*':
            res = b * a;
            break;
        case '-':
            res = b - a;
            break;
        case '+':
            res = b + a;
            break;
        case '/':
            res = b / a;
            break;
        case '%':
            res = b % a;
            break;
        case '^':
            res = b ^ a;
            break;
        }

        return res;
    }
};

char Expr::custChar = 'A';

int main()
{
    Expr *a = new Expr();
    delete a;
    return 0;
}

/*

Output:

$ g++ Pract10.cpp -o out && ./out
Enter Your Equation : (a+b)
```

```
***********************************************
Infix Expression : (a+b)
***********************************************
Char : a
Stack : (
LinkedList  -> NULL
Enter Value for variable a : 1
***********************************************
Char : +
Stack : (
LinkedList -> (a, 1) -> NULL
***********************************************
Char : b
Stack : ( +
LinkedList -> (a, 1) -> NULL
Enter Value for variable b : 2
***********************************************
Char : )
Stack : ( +
LinkedList -> (b, 2)-> (a, 1) -> NULL
***********************************************
Postfix Expression : ab+
***********************************************
Char : a
Stack : a
LinkedList -> (b, 2)-> (a, 1) -> NULL
***********************************************
Char : b
Stack : a b
LinkedList -> (b, 2)-> (a, 1) -> NULL
***********************************************
Char : +
Stack : A b
LinkedList -> (A, 3)-> (b, 2)-> (a, 1) -> NULL
***********************************************
Evaluation of this Equation is : 3
***********************************************

*/
```