Name-Arya Dubey

Registration Number-20BCE0908

Faculty: Professor Gopinath M.P.

APPLICATION OF STACKS

(INFIX TO PREFIX AND POSTFIX)

1)INFIX TO POSTFIX

```
#include<iostream>
using namespace std;
int precedence(char m)
{
  if(m == '^')
  return 3;
  else if(m == '*' || m == '/')
  return 2;
  else if(m == '+' || m == '-')
  return 1;
}
void infix_to_postfix(string t)
{
  stack<char> s;
  int l = t.length();
  string ans;
```

```
for(int i = 0; i < l; i++)
{
  if((t[i] >= 'a' && t[i] <= 'z') || (t[i] >= 'A' && t[i] <= 'Z'))
      ans+=t[i];
  else if(t[i] == '(')
      s.push('(');

  else if(t[i] == ')')
  {
    while(s.top() != '(')
    {
      char c = s.top();
      ans += c;
      s.pop();

    }
    if(s.top() == '(')
    {
      char c = s.top();
      s.pop();
    }
  }
  else{
    while(s.empty()== false && precedence(t[i]) <= precedence(s.top()))
    {
      char c = s.top();
      ans += c;
      s.pop();
```

```
    }
    s.push(t[i]);
  }
 }

 while(s.empty() == false)
 {
  char c = s.top();
  ans += c;
  s.pop();

 }

 cout << ans << endl;
}
int main()
{
 string s = "a+b*c";
 infix_to_postfix(s);
 return 0;
}
```

OUTPUT

abc*+

---

2)INFIX TO PREFIX

```
#include <iostream>
#include <stack>
#include <algorithm>

using namespace std;

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 InfixToPrefix(stack<char> s, string infix)
```

```
{
    string prefix;
    reverse(infix.begin(), infix.end());

    for (int i = 0; i < infix.length(); i++) {
        if (infix[i] == '(') {
            infix[i] = ')';
        }
        else if (infix[i] == ')') {
            infix[i] = '(';
        }
    }
    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] == '(') {
            s.push(infix[i]);
        }
        else if (infix[i] == ')') {
            while ((s.top() != '(') && (!s.empty())) {
                prefix += s.top();
                s.pop();
            }

            if (s.top() == '(') {
                s.pop();
            }
        }
```

```
else if (isOperator(infix[i])) {
    if (s.empty()) {
        s.push(infix[i]);
    }
    else {
        if (precedence(infix[i]) > precedence(s.top())) {
            s.push(infix[i]);
        }
        else if ((precedence(infix[i]) == precedence(s.top()))
            && (infix[i] == '^')) {
            while ((precedence(infix[i]) == precedence(s.top()))
                && (infix[i] == '^')) {
                prefix += s.top();
                s.pop();
            }
            s.push(infix[i]);
        }
        else if (precedence(infix[i]) == precedence(s.top())) {
            s.push(infix[i]);
        }
        else {
            while ((!s.empty()) && (precedence(infix[i]) < precedence(s.top()))) {
                prefix += s.top();
                s.pop();
            }
            s.push(infix[i]);
        }
    }
}
```

```
    }

    while (!s.empty()) {
        prefix += s.top();
        s.pop();
    }

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

int main()
{

    string infix, prefix;
    cout << "Enter a Infix Expression :" << endl;
    cin >> infix;
    stack<char> stack;
    cout << "INFIX EXPRESSION: " << infix << endl;
    prefix = InfixToPrefix(stack, infix);
    cout << endl
         << "PREFIX EXPRESSION: " << prefix;

    return 0;
}
```

```
Enter a Infix Expression :
a+b*c/d*e-f
INFIX EXPRESSION: a+b*c/d*e-f

PREFIX EXPRESSION: -+a*/*bcdef

...Program finished with exit code 0
Press ENTER to exit console.
```