CSE1033.1

Data Structure Assignment II – Stack

Submission To:

Md. Siyam Sajeeb Khan

Lecturer and Coordinator,

Department of CSE,

Southeast University

Submitted By:

Anik Kumar Chakrabortty

ID: 2017000000037

Batch: 45, Section: 1

Semester: Summer 2018

Prefix to Postfix with Tree:

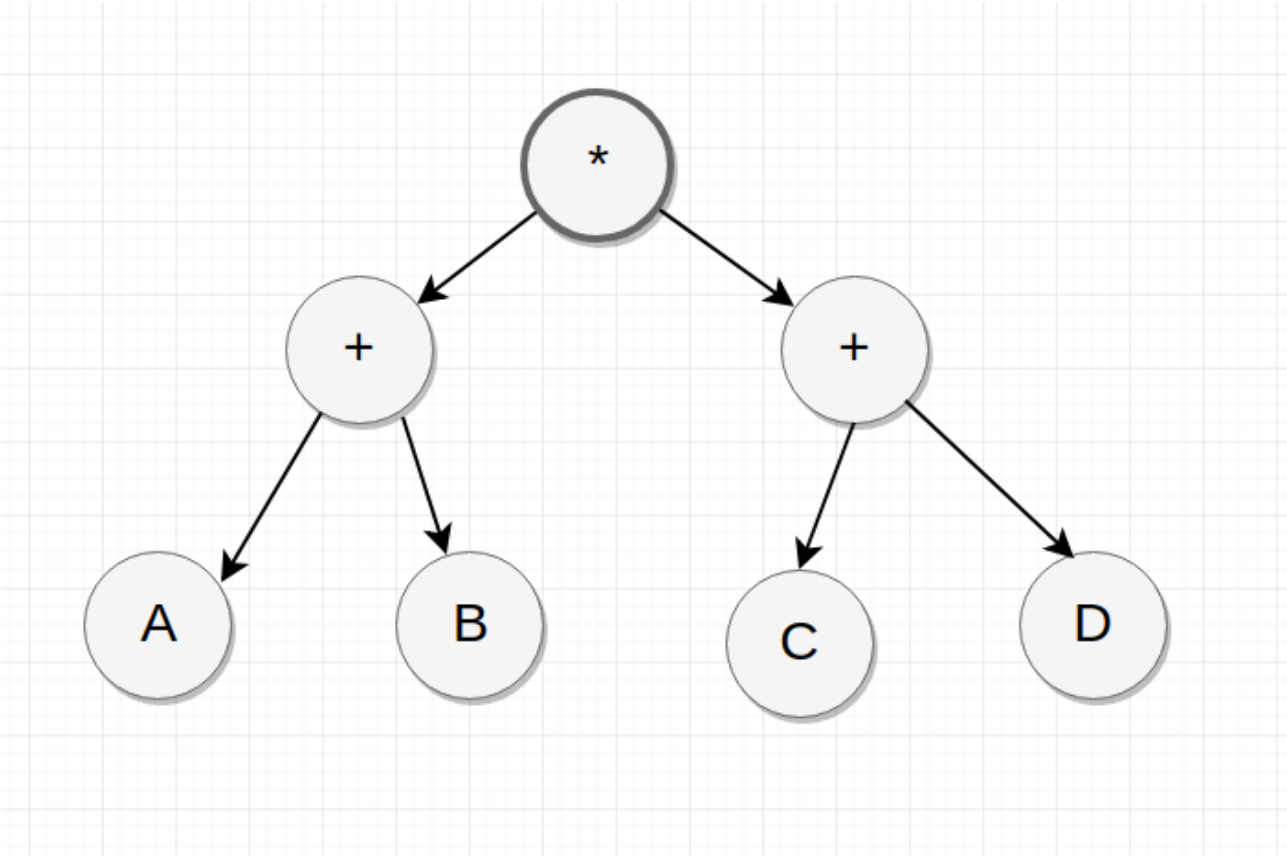
Let’s take an example of a infix equation first.

(A+B) * (C+D)

As it is known to us, prefix expressions places the operators first then comes the operands. So, converting infix to prefix will give this exact expression:

Prefix: *+AB+CD

Now we can easily construct a tree based on this:



It is quite easy to derive a postfix expression from this tree. Visiting the tree from left-bottom will help us construct a postfix expression. Which will come down to this:

Postfix: AB+CD+*

(Continued on next page with implementation)

Implementation:

The main idea behind the implementation of Prefix to Postfix- First we will read the expression from right to left. If the specific index is operand then we will push it to Stack. Otherwise, we will pop two operands from the Stack, create a string by adding the two operands and the operator after them. Then push it back to stack.

C++ Code:

```
string prefixTopostfix(string exp){
stack<string>st;  // using STL stack.
// iterating expression right to left by each character
for(int i=exp.length()-1; i>=0 i—){
      if(isOp(exp[i]==TRUE){   // isOp external function will check whether the index is operator or not
            string one = st.top();
            st.pop();
            string two = st.top();
            st.pop();
            string build = one+two+exp[i];
            st.push(build);
      }else{
      string chartoStr = string(1, exp[i]); // converting that char to string when its right to do so.
      st.push(chartoStr);
      }
      return st.top();
}
```

(Continued on next page with Postfix to Prefix Implementation)

```
string postfixToprefix(string exp){
stack<string>st;  // using STL stack.
// iterating expression right to left by each character
for(int i=0; i<exp.length(); i--){
      if(isOp(exp[i]==TRUE){   // isOp external function will check whether the index is operator or not
            string one = st.top();
            st.pop();
            string two = st.top();
            st.pop();
            string build =exp[i]+ one+two;
            st.push(build);
      }else{
      string chartoStr = string(1, exp[i]); // converting that char to string when its right to do so.
      st.push(chartoStr);
      }
      return st.top();
}
```

(END of content)