SACAD: John Heinrichs Scholarly and Creative Activity Days

Volume 2022 Article 49

4-20-2022

Postfix Form to Infix Form

Joe Liu
Fort Hays State University, y_liu32@mail.fhsu.edu

Recommended Citation
Liu, Joe (2022) "Postfix Form to Infix Form," *SACAD: John Heinrichs Scholarly and Creative Activity Days*: Vol. 2022, Article 49.
DOI: 10.58809/HNQT3829
Available at: https://scholars.fhsu.edu/sacad/vol2022/iss2022/49

Postfix Form to Infix Form

Joe Liu
[1]Department of Computer Science, For Hays State University
Research Mentors: Dr. Hongbiao Zeng)

FORT HAYS STATE UNIVERSITY
Forward thinking. World ready.

Introduction

The mathematical expression that people are most familiar with is known as Infix form. There are also other ways to express math equations such as postfix form and prefix form. Infix expression consists of two operands and one operator such as 2+2. Postfix expression consists of two operand but with the operator at the end (2 2 +). Prefix expression is similar to postfix expression with one major difference that the operator is written before the operand (+ 2 2).

abstract

Even though infix expression is easy for humans to understand, it is hard for the computers to comprehend. According to Sayan Mahapatra from GeekforGeeks, “Prefix and Postfix expressions can be evaluated faster than an infix expression. This is because we don’t need to process any brackets or follow operator precedence rule.” Using postfix expression eliminates the problem of confusion due to precedence. For example, an infix expression such as 2$3&4. One does not know if 2$3 should be applied first or 3&4 without knowing the precedence. In addition, Postfix expression can be easily calculated with the Stack-base algorithm.

Algorithm

```
Iterate through the postfix expression:
    When encountered an operand: push it into the stack
    When encountered an operator:
     1. pop out the last two operand in the stack
     2. apply the operator
     3 .push the value back
into the stack
When there is only one value left in the stack, return the
value
```

```cpp
double evaluatePostfix(string postfix);
// Precondition: Enter a string in postfix form
// Postcondition: the value if of the postfix expression will be evaluated
double calculate(double x, double y, string op);
// Precondition: enter 2 number and a operator
// Postcondition: a value will be returned based on the input
bool isOperator(string s);
// Precondition: Enter any string
// Postcondition: true is returned if the string is an operator, fasle other
wise

int main()
{
    cout << "This program evaluate postfix expressions" << endl;
    cout << "'35 7 + 8 6 5 - * -' evalates to : " << evaluatePostfix("35 7 + 8
6 5 - * -") << endl;

    string x;
    cout << "Enter a postfix expression: ";
    getline(cin, x);
    cout << evaluatePostfix(x);
}

double evaluatePostfix(string postfix) {
    //go over each token in postfix, do accordingly
    stack<string>s;
    int begin = 0;
    int next = 0;
    //go over postfix to find all tokens
    while (next != -1 && begin < postfix.length()) { //keep going if there is
still something in the string

        //seperate the string by space (get several substring)
        next = postfix.find(" ", begin);
        string str = postfix.substr(begin, next - begin); //second element is
how long of this substring

        //if it is integer, push into the stack
        if (isdigit(str[0]))
            s.push(str);

        //if it is operator, take the top 2 value in the stack, apply the
operator and push back into the stack
        else if (isOperator(str)) {
            string op2 = s.top();
            s.pop();
            string op1 = s.top();
            s.pop();
            double val = calculate(stof(op1), stof(op2), str);
            s.push(to_string(val));
        }

        //start from the next index
        begin = next + 1;
    }
    //There will only be one value left in the stack
    return stof(s.top());
}
```

```cpp
double calculate(double x, double y, string op) {
    if (op == "+")
        return x + y;
    else if (op == "-")
        return x - y;
    else if (op == "*")
        return x * y;
    else if (op == "/")
        return x / y;
}

bool isOperator(string s) {
    //if greater than one than it has to be numbers
    if (s.length() > 1)
        return false;

    //if the opertor is found, it will return the index
within the the string (not equal to -1)
    //which will make the follwing excpression true
(return true)
    string ops = "+-*/";
    return ops.find(s) != -1;
}
```

Conclusions

Through this project, I have learned an efficient way to calculate math expression. Using various sources such as YouTube and different websites on Google, I have created an algorithm to evaluate expressions using postfix form rather than infix. To further my project, I am working on converting infix expression to postfix expression and making a calculator to evaluate Boolean expression base on my current algorithm.

Reference

Mahapatra, S. (2021, September 6). *Evaluation of Prefix Expressions*. GeeksforGeeks. Retrieved March 30, 2022, from https://www.geeksforgeeks.org/evaluation-prefix-expressions/#:~:text=Prefix%20and%20Postfix%20expressions%20can,first%2C%20irrespective%20of%20its%20priority.