CS 121, Lecture Outline, May 7, 2021

• Updated course schedule and the new assignment
• Mid-Semester Survey: thank you all for your feedback!
  • Key takeaways
  • Adopted suggestions: more interactions during lectures, more lab time for student work, smaller/more frequent assignments, our own assignments
• Postfix expression Evaluation
• Stack Related Applications: check the validity of parentheses/bracket sequences
  • Single type of parentheses: ( )
  • Multiple type of parentheses: ( ) [ ] { } ...
Postfix Expression: Evaluation

5 3 + 8 * 2 -

• A: 27
• B: 48
• C: 62
• D: 118

Comment: you can use the code for evaluating postfix expressions (available on the slides for Wed., May 5) to compare your evaluation to the program output, helping you develop a better understanding of postfix expressions.
Valid Parenthesis sequences: Single Type ( )

• Input: a sequence of parentheses;
• output: true if the sequence is valid, false otherwise
  • ( ( ( ) ) ) ) valid: openingCount = 4, closingCount = 4
  • ( ( ) ( ( ) ) ) invalid, openingCount = 4, closingCount = 5
  • ( ( ( ( ) ) ) invalid, openingCount = 5, closingCount = 3

• Ideas
  • compute the number of opening parentheses and the number of closing parentheses, and check if they are the same, at the end
  • Check if the first parenthesis is an opening one and the last one is a closing one

These ideas show good observations but are not enough yet. Here is one counter example that satisfies both properties but is still invalid: ( ) ) ( ( )

→ Key observation: need to check the numbers of opening and closing parentheses throughout the processing of the parentheses, not just at the end
Valid Parenthesis sequences: Single Type ( )

Given a sequence of parentheses, determine if the sequence is valid

• Create **two counters**, openingCount and closingCount, initialized to 0
• For each parenthesis in the sequence
  • Update the appropriate counter
  • If (closingCount > openingCount)
    • return false

• //out of the loop. This means that
• //all the parentheses are processed, closingCount was never
• //greater than the openingCount
• If (the closing counter is smaller than the opening counter)
  • return false; // there are extra opening parentheses

• return true
Valid Parenthesis sequences: Single Type ( )

Given a sequence of parentheses, determine if the sequence is valid

Alternative solution: use just one counter to keep track of the number of un-matched opening parentheses so far

• Create one counter, unMatchedOpeningCount, initialized to to 0
• For each parenthesis
  • If it is an opening one, increase unMatchedOpeningCount by 1
  • else decrease unMatchedOpeningCount by 1
• If (unMatchedOpeningCount < 0)
  • return false  //an un-matched closing parenthesis
• //all the parentheses are processed, and there is no unmatched closing parenthesis
• If ( unMatchedOpeningCount > 0 )
  • return false; // unmatched opening one(s)
• return true
Valid Bracket Sequences: Multiple Types ( ), [ ], { } ... 

• ( [ { } ] { [ ] ( ) } ) valid
• ( [ { ] } ): invalid, first issue: the mismatched opening curly bracket and closing square bracket

• Key observations
  • In a valid sequence, a closing bracket needs to match the nearest, unmatched opening bracket before it: they need to have a same type
  • simply keeping track of the numbers of unmatched opening brackets is not enough. We also need to know the order of the unmatched ones.

• Use a stack to keep track of un-matched opening brackets and their order
  • The bracket at the top is the nearest unmatched opening bracket
  • In a valid sequence, the next closing bracket must have the same type as the opening bracket at the top
Valid Bracket Sequences: Multiple Types (),[],{} ...

• Create an initial empty stack of Strings
• For each bracket in the sequence
  • if it is an opening one, push it to the stack
  • If it is a closing one
    • If the stack is empty, return false; //un-matched, extra closing bracket
    • Pop the stack, and check if the popped opening bracket and the current closing bracket are the same type.
    • If they are different types, return false //mis-matched types, as in { ]

// if the stack is empty, all brackets are matched; otherwise, it indicates unmatched opening brackets, and the sequence is invalid
• return stack.isEmpty()