CSI 33 LECTURE NOTES (Ojakian)

Topic 5: Stacks

OUTLINE
(References: 5.1, 5.2)

1. Stacks
2. Matching Paretheses
3. Postfix and Prefix Notation
4. Context-free Grammar

---

1. Intro Example
   (a) Application to parenthesis problem

   PROBLEM 1. *Program parentheses problem with one sort of parentheses (without stacks).*

   PROBLEM 2. *How would you program the parentheses problem when you have multiple types of parentheses??*
   Consider Stacks ...

2. Basic Operations of Stack
   (a) Two fundamental operations: push and pop
   (b) May have a few others: stack size and look-at-top
   (c) Program it

   PROBLEM 3. *Program the stack.*

   PROBLEM 4. *Use a stack to program the parentheses problem when you have multiple types of parentheses.*

3. Infix, Post-fix and Pre-fix
   (a) General Intuition.
      i. Infix: Standard human way to write arithmetic expressions.
      ii. Postfix: Reading left to right, the operation comes **after** its two operands.
      iii. Prefix: Reading left to right, the operation comes **before** its two operands.
   (b) Infix: Our usual way of writing mathematical expressions.
      i. NOT left to right, in that follow order of operations.
      ii. Post-fix and Pre-fix, need no parentheses and are just read left to right.
      iii. Typically, easier for a computer to work with postfix, prefix
   (c) Postfix- Read left to right, when you reach a binary operation:
      i. Perform it on the 2 most immediate values to the left, then
      ii. Replace the operation and two values by the new value

(d) Prefix - Read left to right, when you reach a binary operation:

i. Perform it on the most immediate values to the right

ii. Defer an operation till both its operands are available, then do it.

(e)

PROBLEM 5. *See* **Postfix Tutorial at webpage.** *Evaluate it by just reading it left to right.*

PROBLEM 6. *Calculate the following postfix expressions:*

i. 5 2 + 8 3 − ∗

ii. 6 13 + 3 5 − /

PROBLEM 7. *Calculate the following prefix expressions*

i. ∗ 9 + 2 6

ii. + 7 ∗ 45 + 2 0

4. Using a stack to calculate Prefix and Postfix Expressions

(a) Postfix Philosophy: Wait for an operator, then look back at two operands - i.e. operator after operands). We can call this an "Operator Triggered Stack".

i. Read left to right.

ii. If next item is an operand, push onto stack.

iii. If next item is an operator, pop last two items, evaluate, then push the result on the stack.

iv. The single number on the stack at the end is the answer

(b) Prefix Philosophy: wait for two operands, then look back at the operator - i.e. operator before operands). We can call this an "Operand Triggered Stack".

i. Read left to right.

ii. Push the next item onto the stack.

iii. If the top two items on the stack are operands, then pop the top two operands and apply the next popped operator, evaluate, then push the result on the stack.

iv. The single number on the stack at the end is the answer

(c)

PROBLEM 8. *Program a postfix evaluator using stacks.*

5. Application to Context Free Grammar

(a) What it is?

i. Have non-terminal and terminal symbols

ii. Have production rules with single left side non-terminal and right side sequence

iii. Start with a non-terminal, replacing in some fashion till only terminals are left.

iv. Question: What language does the grammar give you? (i.e. given a word in terminal symbols, can you produce it or not?)

(b)

PROBLEM 9. *Create some more rules and run it.*

PROBLEM 10. *Understand the code, in particular the the use of a Stack.*