CSC115 Lecture 14

An Application: Infix & Postfix Expressions

Searching: Linear & Binary

Sorting

An Application: Algebraic Expressions:

in Infix & Postfix form

Which calculator buttons are needed to calculate: 17 x (13 – 107) ?

eofdreams.com/calculator.html

www.amazon.com

The HP Calculator uses: Postfix Notation, which does not need brackets or Equals signs. . . .

Algebraic Expressions

- Three languages for algebraic expressions
  - Infix expressions
    - An operator appears between its operands
    - Example: a + b
  - Prefix expressions
    - An operator appears before its operands
    - Example: + a b
  - Postfix expressions
    - An operator appears after its operands
    - Example: a b +

We are only interested in Infix and Postfix. . . . Prefix is just interesting!

 6-3

Algebraic Expressions

- To convert a fully parenthesized infix expression to a prefix form
  - Move each operator to the position marked by its corresponding open parenthesis
  - Remove the parentheses
  - Example
    - Infix expression: ((a + b) * c)
    - Prefix expression: * + a b c

 6-4

Algebraic Expressions

- To convert a fully parenthesized infix expression to a postfix form
  - Move each operator to the position marked by its corresponding closing parenthesis
  - Remove the parentheses
  - Example
    - Infix form: ((a + b) * c)
    - Postfix form: a b + c *

Algebraic Expressions

- Prefix and postfix expressions
  - Never need
    - Precedence rules
    - Association rules
    - Parentheses
  - Have
    - Simple grammar expressions
    - Straightforward recognition and evaluation algorithms

Fully Parenthesized Expressions

- To avoid ambiguity, infix notation normally requires
  - Precedence rules
  - Rules for association
  - Parentheses
- Fully parenthesized expressions do not require
  - Precedence rules
  - Rules for association

Evaluating Postfix Expressions

- A postfix calculator
  - Requires you to enter postfix expressions
    - Example: 2, 3, 4, +, *
  - When an operand is entered, the calculator
    - Pushes it onto a stack
  - When an operator is entered, the calculator
    - Applies it to the top two operands of the stack
    - Pops the operands from the stack
    - Pushes the result of the operation on the stack

Evaluating Postfix Expressions

Key entered	Calculator action		Stack (bottom to top)
2	push 2		2
3	push 3		2 3
4	push 4		2 3 4
+	operand2 = pop stack	(4)	2 3
	operand1 = pop stack	(3)	2
	result = operand1 + operand2	(7)	2
	push result		2 7
*	operand2 = pop stack	(7)	2
	operand1 = pop stack	(2)	
	result = operand1 * operand2	(14)	
	push result		14

Figure 7-8

The action of a postfix calculator when evaluating the expression 2 * (3 + 4)

Converting Infix Expressions to Equivalent Postfix Expressions

- An infix expression can be evaluated by first being converted into an equivalent postfix expression
- Facts about converting from infix to postfix
  - Operands always stay in the same order with respect to one another
  - An operator will move only "to the right" with respect to the operands
  - All parentheses are removed

Converting Infix Expressions to Equivalent Postfix Expressions

ch	stack (bottom to top)	postfixExp	
a		a	
–	–	a	
(	– (	a	
b	– (	ab	
+	– (+	ab	
c	– (+	abc	
*	– (+ *	abc	
d	– (+ *	abcd	
)	– (+	abcd*	Move operators
	– (	abcd*+	from stack to
	–	abcd*+	postfixExp until " ("
/	– /	abcd*+	
e	– /	abcd*+e	Copy operators from
		abcd*+e/–	stack to postfixExp

Figure 7-9

A trace of the algorithm that converts the infix expression *a* - (*b* + *c* * *d*)/*e* to postfix form

Application: Algebraic Expressions

- When the ADT stack is used to solve a problem, the use of the ADT's operations should not depend on its implementation
- To evaluate an infix expressions
  - Convert the infix expression to postfix form
  - Evaluate the postfix expression

Review Assignment #4

- Using Connex

Searching

- Sequential search
  - Starts at the beginning of the collection
  - Looks at every item in the collection in order until the item being searched for is found

Execution time is? O(N) .

- Binary search
  - Repeatedly halves the collection and determines which half could contain the item
  - Uses a divide and conquer strategy

Execution time is? O(log N) .

Binary Search

- A high-level binary search

```
if (anArray is of size 1) {
  Determine if anArray's item is equal to value
}
else {
  Find the midpoint of anArray
  Determine which half of anArray contains value
  if (value is in the first half of anArray) {
    binarySearch (first half of anArray, value)
  }
  else {
    binarySearch(second half of anArray, value)
  } // end if
} // end if
```


Binary Search

- Implementation issues:
  - How will you pass "half of `anArray`" to the recursive calls to `binarySearch`?
  - How do you determine which half of the array contains value?
  - What should the base case(s) be?
  - How will `binarySearch` indicate the result of the search?

Sorting Algorithms & Execution Time

View some Animations:

- http://www.sorting-algorithms.com/

Lets Write One

. . . .And Count the Execution Time