Administrative Details

- Lab 6 is today
  - Postscript interpreter
Last Time

- Discussed iterators (Ch 8)
  - Used for data structure traversal
    - Overcome tension between generality and efficiency
  - Reviewed the Iterator interface
    - next() and hasNext()
  - Reviewed the AbstractIterator class
    - Leaves get(), next(), hasNext(), and reset() undefined (as indicated by “abstract” label in javadocs)
Today’s Outline

• Work through one more iterator example
• Review postfix for today’s lab
• Quick review of switch statement syntax
• Begin ordered structures (Ch 11)
  • An interesting twist on Lists and Vectors
Warmup: More Iterator Examples

• In addition to our “typical” iterators, we can also make specialized iterators
  • Filtering Iterators (cool example in textbook)
  • ReverseIterator
    • Task: given an iterator as input, construct an iterator to traverse the elements in reverse order
Converting Expressions

• We (i.e., humans) primarily use “infix” notation to evaluate expressions
  • \((x+y)*z\)

• Computers use “postfix” (also called Reverse Polish) notation
  • \(xy+z^*\)
  • Operators appear after operands
  • Parentheses not necessary
Converting Expressions

Example: $x*y+z*w$

Conversion

1) Add full parentheses to preserve order of operations
   $(x*y)+(z*w)$

2) Move all operators (+-*/) after operands
   $(xy*)(zw*)+$

3) Remove parentheses
   $xy*zw*+$
Evaluating Arithmetic Expressions

• Computer processes use stacks to evaluate arithmetic expressions

• Example: $x*y+z$
  • First rewrite as $xy*z+$
  • Then:
    • push $x$
    • push $y$
    • mult (pop twice, multiply, push result)
    • push $z$
    • add (pop twice, add, push result)
Use Stack to Evaluate Postfix Exp

• While there are input “tokens” (i.e., symbols) left:
  • Read the next token from input.
  • If the token is a value, push it onto the stack.
  • Else, the token is an operator that takes n arguments.
    • (It is known a priori that the operator takes n arguments.)
    • If there are fewer than n values on the stack \(\rightarrow\) error.
    • Else, pop the top n values from the stack.
      – Evaluate the operator, with the values as arguments.
      – Push the returned result, if any, back onto the stack.

• If there is only one value on the stack, that value is the result of the calculation.
• Else if there are more values in the stack w/o operators, there are too many input values \(\rightarrow\) error.
Example

- \((x*y)+(z/w)\)
- Convert:
  - \(xy*zw/+\)
- Evaluate:
  - Push \(x\)
  - Push \(y\)
  - Mult (Pop \(y\), Pop \(x\), Push \(x*y\))
  - Push \(z\)
  - Push \(w\)
  - Divide (Pop \(w\), Pop \(z\), Push \(z/w\))
  - Add (Pop \(x*y\), Pop \(z*w\), Push \((x*y)+(z/w)\))
  - One value left, so we’re done.
Lab 6

- **Reader.java**
  - Use an Iterator to walk through tokens one at a time
  - Multiple constructors – use the right one for the task

- **Token.java**
  - “Wrapper” type for all of the tokens you will encounter
  - `token.kind()`: NumberKind, BooleanKind, SymbolKind, ProcedureKind
  - (all of the built-in postscript commands are symbols)

- **SymbolTable.java**
  - Key-value store

- Example usage in lab and in Javadoc on webpage
- Use these to help implement Interpreter.java
Switch Statement

- **General structure:**

```java
switch (byte|short|char|int|String|Enum) {
    case __:
        ...
        break;
    case __:
        ...
        break;
    default:
        ...
}
```

Without `break;` code "falls through" to next case.
Moving on...
Ordered Structures

• Until now, we have not required a specific *ordering* to the data stored in our structures
  • If we wanted the data ordered/sorted, we had to do it ourselves

• We often want to keep data ordered
  • Allows for faster searching
  • Easier data mining - easy to find best/worst/average/median values*
Ordering Structures

• The key to establishing order is being able to compare objects and rank them

• We already know how to compare two objects…how?
  • Comparators and `compare(Object a, Object b)`
  • Comparable interface and `compareTo(Object that)`

• What are the advantages of each?
An Aside: Natural Comparators

• NaturalComparators bridge the gap between Comparators and Comparables

```java
class NaturalComparator<E extends Comparable<E>> implements Comparator<E> {
    public int compare(E a, E b) {
        return a.compareTo(b);
    }
}
```