Week 9, 10 logistics

Monday (today): Loops (again and again)
then 2-3pm Java intensive

Tuesday: labs as usual
bring homework if you’re not done

Wednesday: Research topics in CS

Wk 10 Monday: Lecture (review)
Wednesday: Optional midterm do-over

Friday: Review for final exam
while (difficult) {
    loops.practice(examples);
}

// but can we be sure
   //this loop terminates?
Loop problems

1. Replace multiple spaces by one space in strings (“foo  bar” => “foo bar”)
2. Compute average word length in a string (with just String.charAt(), nothing fancy)
3. Print histogram of word lengths

What these have in common:
Keep track of a “state” reflecting what came before in the string. Simple state in #1 (a single boolean will do), #2 and #3 are best designed from an explicit “state machine” (aka finite state machine, finite state automaton).
Compress blanks

/**
 * Create a copy of a string where any sequences of
 * 2 or more blanks are reduced to a single blank.
 * @param before The input string
 * @return A string that differs from the input only in that
 * each sequence of more than one blank has been
 * replaced by one blank.
 */
public static String compress(String before) {

A solution (not the only one)

```java
public static String compress(String before) {
    String after = "";
    boolean spaceBefore = false;
    for (int i = 0; i < before.length(); ++i) {
        char ch = before.charAt(i);
        if (ch != ' ' || !spaceBefore) {
            after = after + ch;
        }
        spaceBefore = (ch == ' ');
    }
    return after;
}
```
/**
 * Compute the average length of words in input string.
 * No fair using Scanner, regular expressions, or other fancy
 * tools in Java libraries.
 * @param s Input string to analyze
 * @return average length of words in s
 */

public static double avgWordLength(String s) {
We can be in state “between words” or in state “in word”
If we are “between words” and encounter something other than a blank,
we enter state “in word”
If we are in state “in word” and encounter a blank,
we enter state “between words”
We can associate actions with transitions
private enum WordState { inWord, betweenWords };  
...
public static double avgWordLength(String s) {
    WordState curState = WordState.betweenWords;
Running the automaton

for (int i=0; i < s.length(); ++i) {
    char ch = s.charAt(i);
    if (curState == WordState.betweenWords) {
        if (ch == ' ') {
            /* nothing */
        } else {
            curState = WordState.inWord;
            wordLen = 1;
        }
    } else if (curState == WordState.inWord) {
        if (ch == ' ') {
            curState = WordState.betweenWords;
            nwords += 1;
            totLen += wordLen;
        } else {
            wordLen += 1;
        }
    }
}


Representing state machines

State machines can be represented directly by code, as in my example.

They can be represented by the “state pattern” using classes and inheritance to organize states.

They can be represented by tables (transition matrices)

*Transitions*[state][symbol] contains indicators of new state and needed actions.*
Word frequency histogram

/**
 * Create a simple word length frequency chart for a string.
 * No fair using Scanner, regular expressions, or other fancy tools in Java libraries.
 * @param s Input string to analyze
 */
public static void wordLenFreq(String s) {

1: **********
2: ***********************************
3: #################################################################################################
4: ###################################################
5: ***************************************************
6: ************************
7: *********************
8: ************
9: **
10: ***
Histogram in two phases:

Count word lengths & place in array

lenCounts[wordLen] += 1;

Print *’s for each element of array

for (int i=1; i <= maxLen; ++i) {
    String label = String.format("%2d: ", i);
    System.out.print(label);
    for (int star=0; star < lenCounts[i]; ++star){
        System.out.print("*");
    }
    System.out.println();
}
The same state machine, with different actions

```java
for (int i=0; i < s.length(); ++i) {
    char ch = s.charAt(i);
    if (curState == WordState.betweenWords) {
        if (ch == ' ') { /* nothing */ }
        else {
            curState = WordState.inWord;
            wordLen = 1;
        }
    } else if (curState == WordState.inWord) {
        if (ch == ' ') {
            curState = WordState.betweenWords;
            lenCounts[wordLen] += 1;
            if (wordLen > maxLen) {
                maxLen = wordLen;
            }
        } else {
            wordLen += 1;
        }
    }
}
```
Summary

Loops, loops, loops.

State machines (finite state automata) are useful in designing loops that scan a sequence (especially text)