Welcome to CSE 143!

CSE 143: Computer Programming II

Course Goals

1. CSE 142 vs. CSE 143: The Big Picture
   In CSE 142, you learned how to use logic, control flow, and decomposition to write programs.
   In CSE 143, you will learn to solve more complex and larger tasks efficiently.

2. Big Learning Goals
   - Abstraction (implementation vs. client)
   - Data Structures (organizing complex data)
   - Algorithms (standard ways of completing common tasks)

   We're going to build some really cool programs. And have a lot of fun!

Support and Asking for Help

1. Resources
   - TWO sections a week
   - Tons of TAs!
   - The IPL (and my office hours!)
   - Practice-It

   Asking for help is not a sign of weakness; it's a sign of strength.

Boring Administrivia

1. Course Website
   - http://cs.uw.edu/143

2. Section
   - We have two sections a week.
   - Each section has a set of problems; turn in at least one set of problems each week for credit.

3. Grading
   - 47% programming projects, 3% section problems, 20% midterm, 30% final
   - Weekly programming projects assigned Fridays, due on Thursdays
   - 5 “free late days”; -2 points for subsequent days late; up to 3 days late on each hw

Program Correctness: Internal & External

1. What does it mean for a program to be “correct”?
   - A program is only correct if it is internally correct and externally correct.

2. What does this code do?
   ```c
   _(__,___,____){___/__<=1?_(__,___+1,___ _):!(___%__)?_(__,___+1,0):___%__==___ / __&&!____?(printf("%d	",___/__),_(__,_ __+1,0)):___%__
   >1&&___%__<___/__?_( __,1+ ___,____+!(___/__%(___%__))):___<__*__?_(__,___+1,____):0;}main(){_(100,0,0);}\n   ```
What is External Correctness?

The code does the right thing on all inputs.

What is Internal Correctness?

The code is...
- easy to read
- well documented
- well formatted
- efficient
- ...

Internal correctness matters, because:
- Do you want a job at a software engineering company?
- Do you want to ever reuse your code later?
- Do you want to ever write a large program? (Like a game, maybe)
- Important people think it does:
  - Programs must be written for people to read, and only incidentally for machines to execute. (Abelson & Sussman)

Goals for Internal Correctness

- Make non-obvious code obvious via comments.
- Document all features, limitations, design decisions.
- Make your code easy for someone else to read
- Explain what your methods, classes, etc. are supposed to do

Grading will be on both external and internal correctness!

Words Exercise

Write code to read a file and display its words in reverse order.

(Bad) Solution with Arrays

```java
String[] words = new String[1000];
int i = 0;
Scanner inp = new Scanner(new File("words.txt"));
while (inp.hasNext()) {
    String word = inp.next();
    words[i] = word;
    i++;
}
for (int j = i - 1; j >= 0; j--)
    System.out.println(words[j]);
```

Arrays are one way to store many values of the same type (int, String, DrawingPanel, etc.).

```java
int[] arr = new int[8];
arr = [12, 49, -2, 26, 2, 0, 26, 11];
```

```
Element #3 is 26
arr has size 8
```

Limitations of Arrays

- Fixed, upfront size (once you create the array, it will remain that size)
- Adding and removing can get complicated
- No methods (and weird ".length" syntax)
- Functionality for arrays is in the Arrays class:
  - Arrays.copyOf
  - Arrays.equals
  - Arrays.sort
  - Arrays.toString

Collections and Lists

Collections store many pieces of data of the same type.

In Java, collections are in the util package:

```java
import java.util.*;
```

Different collections have different properties:

- “Data ordered by indices”
- “Sorted data”
- “Data without duplicates”
- etc.

Lists

A list is a collection of elements ordered by a 0-based index.

- It supports add/remove from anywhere!
- The size isn’t fixed!
- There are multiple implementations; first, ArrayList

```java
ArrayList Mechanics

Suppose we have an ArrayList with values: [1, 2, -6]:

Step 0: [1, 2, -6] ...

Insert 5 at index 2:

Step 1: [1, 2, 5, -6] ...

Add 0 at the beginning:

Step 2: [0, 1, 2, 5, -6] ...

Get index 3:

arrayList.get(3) → 5
```
/*
  Remove All Words Ending in 's'
*/
for (int i = 0; i < allWords.size(); i++) {
  if (word.endsWith("s")) {
    allWords.remove(i);
  }
}

// This is the tricky part; since we removed a word,
// we're actually at the SAME index again! */
i--;
}

/**
 * Remove All Words Ending on 's'
 */
for (int i = 0; i < allWords.size(); i++) {
  if (word.endsWith("s")) {
    allWords.remove(i);
  }
}

// Display in Reverse Order
for (int i = allWords.size() - 1; i >= 0; i--) {
  System.out.println(allWords.get(i));
}