Week 2: Introduction to testing strategies
▶ Getting started: the most important criteria
▶ Checking intermediate results
▶ Manual vs. automatic checking

The number 1 strategy
▶ The ease and thoroughness of testing (validation, debugging) depends heavily on the internal quality of the programs.
▶ Specifically:
  ▶ Localization, lack of unnecessary repetition
  ▶ Partitionability, "divide & conquer"
  ▶ Readability / understandability

Why?

The number 2 strategy
▶ Many bugs can be spotted in a manual inspection of the source code:
  ▶ Best by someone other than the original programmer.
  ▶ Consider this simple function:
    ```java
double arraySum(double a[], int low, int high)    {double result;    for (int k=low; k<=high; ++k)       result += a[k];    return result;  }
```
▶ Do we need to run that in order to find the bug(s)?
  ▶ in Java? Why the difference?
  ▶ in C++

Therefore . . .
▶ Ideally unit-testing or integration-testing (debugging / validation / verification) should begin:
  ▶ with well-organized, clearly-coded MUTs
  ▶ after reasonable inspection (desk checking)
▶ Is that always possible? practical?
Checking intermediate results

Whenever the result of a non-trivial process (function, transaction, complete program) appears wrong, we'll want to examine intermediate results. What do we mean by "appears wrong"?

That applies both:
- to computation-intensive processes
- and complex logic / decision rules.

We hope that will show where things went wrong or started to go wrong. Does that always help?

Displaying versus automatically verifying

Suppose that printing or displaying a final or intermediate result and then examining the output would either:
- reveal a problem or
- validate a process

Then a test driver program ought to be able to make the same comparisons.
- That's more reliable.
- It's repeatable.
- It doesn't waste paper or repeated time

A little extra work up front reduces tedious repetitive work later.

Problems with examining intermediate results

1. The process may be so complicated that we can't easily determine what the expected correct result should be! What can we compare the actual result to?

2. If we print the intermediate results, we may be overwhelmed by the output volume.
   - Inexperienced programmers are often tempted to insert print/display statements everywhere!
   - The program may waste several hundred pages of paper or hours of time and still not reveal the problem. What else could we do?

Problems with examining intermediate results (continued)

3. Inserting intermediate output or validation statements may alter a program's behavior.
   - That was a common problem with assembly-language programs. Why? What's the remedy?
   - It's still a difficult and often frustrating problem with parallel processing (multi-tasking / multi-threading). Why? What's the remedy?
Supplementary reading recommendation #1

- You won't be responsible for anything in these supplementary reading sources that we didn't thoroughly discuss in class.
- Nevertheless, you should find them interesting and helpful in your future work.
  - What can a 1979 book tell us today?
  - It doesn't know about OOP or C#
  - It doesn't know about modern operating systems and networks.
  - But it still contains a lot about testing strategies.

Assignment #2

- This is a bit like assignment #1, except that you get to determine the appropriate strategy.
- It uses a sample *Angle* class as a basis.
  - This illustrates a shortcoming of the C# library.
  - You need to remember *basic high-school trigonometry*, but nothing very advanced.
  - Follows additive pattern, but range is limited.
  - What's that? To what?
- If you prefer to do it in Java instead of C#, you'll test named functions instead of C# operators.