CSC 101 Lecture Notes Week 3

Problem, Problem, Problem, Solving

I. There are several problems we're trying to solve.

A. The *user's* problem.

B. The *testing* problem.

C. The *debugging* problem.

II. **Solving the users's problem.**

A. Learn to think *algorithmically*, with *step-by-step* solutions.

B. For bigger problems, learn to decompose into smaller *sub-problems*.

C. Learn at least one programming language.

D. Practice, practice, practice.

III. **Solving the testing problem.**

A. First get program to compile.

B. Then come up with test cases.

C. Each test case has:

1. program inputs
2. expected program outputs

Solving the testing problem, cont'd

D. Run program to see if actual output matches the expected output.

E. When it doesn't match, *debug*.

IV. Solving the debugging problem

A. Determine which part(s) of the program is(are) causing the problem.

B. Analyze nature of failure:

Solving the debugging problem, cont'd

1. you're not executing parts of a program you were expecting to;

Solving the debugging problem, cont'd

1. you're not executing parts of a program you were expecting to;

2. you're not executing parts of a program in the order you were expecting to;

Solving the debugging problem, cont'd

1. you're not executing parts of a program you were expecting to;

2. you're not executing parts of a program in the order you were expecting to;

3. there's some flaw in your program logic, i.e., the way the algorithm was translated into C code.

V. **Specific techniques for generating test cases.**

A. Choose typical test cases, e.g., spec examples.

B. Create tests for a *range* of legal input values.

Techniques for generating test cases, cont'd

1. at *lower boundaries*
2. at *upper boundaries*
3. one above *lower boundaries*
4. one below *upper boundaries*
5. different combinations of inputs:

Techniques for generating test cases, cont'd

a. one input varying, all others fixed

b. varying combinations of inputs that focus on a particular aspect of the problem

Techniques for generating test cases, cont'd

C. Create tests for a *range* of output values.

1. at *lower boundary*
2. at *upper boundary*
3. one above *lower boundary*
4. one below *upper boundary*

Techniques for generating test cases, cont'd

5. Different combinations of expected output:

   a. one output varying, all others fixed

   b. varying combinations of inputs to produce outputs for a particular aspect of the problem

Techniques for generating test cases, cont'd

D. Create tests that exercise program logic:

1. each conditional expression is evaluated fully

2. *loops* are fully exercised (coming up)

Techniques for generating test cases, cont'd

E. Have enough tests to *fully cover* program.

F. When the program does error handling, create test cases that produce each form of error.

VI. **Specific techniques for program debugging.**

A. Program *inspection*.

1. By yourself.
2. A lab partner *(for lab programs)*
3. One or more other colleagues.

Note that for CSC 101 class purposes, you should not have your programming assignments inspected, since you're working on these by yourself.

Techniques for debugging, cont'd

B. Compare your program to one that works, or used to work:

1. Useful during incremental development.

2. Save each working version as you develop.

3. When something breaks, compare to previous working version.

Techniques for debugging, cont'd

C. Isolate buggy program location:

1. Use particular test case results.

2. Put in intermediate print statements.

3. Use a program debugger to set break points.

Techniques for debugging, cont'd

D. Write a program *test driver*

1. Effective when a program has many functions, that each do non-trivial computations.

2. We'll look at this starting next week.