Intro to program testing

- we generally want to be sure that our programs work correctly
- the only way to be sure is to test the programs on actual data
- the test cases we use should check the program handles valid data correctly, and also that it performs error checking and handling correctly
- ideally, we should think of a collection of test cases before (or at least independently of) the actual code we're writing -- nearly every statement in the requirements/specifications for a program should give us more ideas on test cases we should create and use

Input and expected output

- for each test case we think of, we should have:
  - a reason for the test case (why we want to test that case in particular)
  - the exact input data that the user would supply for the test case
  - the exact output we think the program should display for the test case

Manual vs automated testing

- we could manually type in and check each test case every time we want to test an updated version of the program, but that is slow/error prone
- we can instead create a file to hold the input data, another to hold the expected output, and eventually store the actual program output in a third

```
./myprogx < inputfile > actualoutput
```

- we can compare the actualoutput with the expectedoutput using the following command

```
diff actualoutput expectedoutput
```

Example: read/display a time

- suppose a program is supposed to prompt the user to enter the hour and the minutes for a time, check they are valid, then display the results in the format h:mm
- a sample run of the program might look something like

```
please enter the hour (1-12)
10
please enter the minute (0-59)
27
the time is 10:27
```

Creating a test case

- our test case for that particular run would have an input file that just contained the user input, i.e. just the 10 and 27

```
10
27
```

- the expected output file would contain exactly what we expect the program to cout/printf as it runs, i.e.

```
please enter the hour (1-12)
please enter the minute (0-59)
the time is 10:27
```

- the diff command would show the line-by-line differences between the output actually produced and the output expected

Thinking up test cases

- most programs require a great many test cases, e.g. in our time example we would want different cases to cover
  - the smallest valid hour (1) and the largest (12)
  - the smallest valid minute (0) and the largest (59)
  - the hours just “outside” the valid range (0 and 13)
  - the minutes just outside the valid range (-1 and 60)
  - a variety of valid times
  - a variety of invalid times
- for each, our sample output must reflect all prompts, error messages, and other output we expect that case to generate

Automating testing

- using a text editor, we can put the user input data for a test case into a file (i.e. the exact lines of text we would usually type as the program runs)
- suppose we usually have to type in our name and two numbers as the program ran, then the test file content might look like

```
dave
128
6.4
```

- the < and filename can be used to run our program but have it read its input from the file instead of from the keyboard

```
./myprogx < mytestcasefile
```

- this allows us to quickly re-run the program on a test case without manually retyping each time (faster, less chance of error)

Automating multiple test cases

- if we have multiple such test cases to run, we could run them from the command line, e.g.

```
    ./myprogx < testfile1
   ./myprogx < testfile2
etc
```

- or we could put all these commands into yet another file, e.g. called runmyfiles, and then tell the bash command interpretter to run all of them in sequence:

```
    bash runmyfiles
```

- this is much faster, and eliminates the chance of missing any test cases