Language-specific debugging

- Most languages include some features/support for debugging, good idea to know what they are
- Special variables with key information
- Special functions/libraries that can be used
- Special options that can be configured
- We'll look at some features in C/C++ and in bash

C/C++ special #defined values

- Special #defined variables are available (e.g. to include when displaying error/debugging messages)

  `__FILE__` gives current filename, `__LINE__` for line number, and `__TIME__`, `__DATE__` for current time and date

- (note those are double-underscores, not single)
- These can be embedded directly in code, e.g.

```
std::cout << “On line “ << __LINE__ << “, in file “;
std::cout << __FILE__ << std::endl;
```

C/C++ asserts

- #include the <cassert> library (<assert.h> for C)
- The function assert(X) assumes it is being passed a boolean value, and immediately terminates the program if the value is false (displaying the assert line that caused termination)
- This is used as a fail-safe for spots where the developer is sure the condition must be true, and wants to abort processing if not, e.g.

```
assert(myGreatPtr != NULL);
```

Appropriate use of asserts

- Generally we don’t want to rely on asserts in the final/production version of a product (no user wants to see a crash and cryptic source code message)
- They’re primarily used during development as a double-check that something isn’t broken
- We can turn off asserts during g++ compilation (so we don’t have to edit our final code to remove them and risk breaking the code) using flag -DNDEBUG

Bash debugging options

- The -xv flags tell bash to echo each command just before it runs, so we can see which instructions cause an issue
- #! /bin/bash -xv
- It can also be turned on in the middle of a script using
- set -xv
- (and then turned off again later using set +xv)
- The -u flag can also be turned on/off to give us warnings when we use an unbound (undeclared/initialized) variable

Traps in bash

- We can also use trap in bash, to execute a specific command when some event takes place
- e.g. we can use trap to print the value of some variable of interest whenever the script exits (or crashes)
- trap echo “when script stopped, x was ${x}” EXIT

Build our own bash assert

- A C-like assert could be added if we wanted, say to test a condition and exit with a specific status if it was false

```
function assert () {
   if ! [ $cond ] ; then
      exit $2
   fi
}
```

- Pass condition as a string, then exit status as int 0-255

```
assert “$x -lt $y” 1
```