Quick intro to debugging

- debugging is an inevitable part of software development
- good coding habits help us reduce the amount and difficulty of debugging
  - top down design
  - following code standards
  - iterative development
- good testing skills help us identify problems early in development
- good debugging skills can simplify the debugging process
- learning to use a debugging tool can also help significantly

reduce the likelihood of problems

- follow code standards as you write the code, rather than cleaning the code up later (otherwise you're debugging the sloppy/hard to read code)
- use iterative development, so bugs are likely to be in (or closely related to) the newly added code
- don't use fancy tricks to shorten your code, it makes it harder to read, maintain, and debug
- split complicated expressions into simpler steps, store intermediate values in variables

debugging tips

- identify how to consistently reproduce the problem, the first step in identifying what's going wrong
- focus on one problem at a time (e.g. the *first* compiler warning/error)
- try explaining the problem to someone else (this helps organize your own thoughts)
- get someone else to look at the code
- take a break (coffee/shower/sleep/whatever) and come back with fresh eyes
- learn to use a debugger (e.g. gdb or ddd on our servers)

Learn to spot common problems

- stack overflow often comes from runaway recursion
- segmentation faults often come from array/pointer issues
- if compiler complains about variables not declared in current scope or about undeclared functions: check for typos in the declaration or where you're using it (and check that you really have declared the item inside the right function)
- if you're getting large/weird values, check for uninitialized variables, or missing error-checking on user input
- look for typos like = instead of == for comparisons

Hypothesize and check

- for harder to spot bugs, try to come up with a few theories as to what is likely to be wrong and think of ways to test/check them
- ways to test/check might involve the input/data combination we've been using to recreate the error, plus examination of variables/values inside the program using extra output statements (e.g. cout the value of a parameter or variable) or using a debugger to follow the changing values

debuggers: gdb on our servers

- (ddd is available as a graphical version of gdb)
- we have to add a special -g option when compiling, so the compiler adds extra information for the debugger

```
g++ myprog.cpp -o myprogx -g -Wall -Wextra
```

- start the debugger, telling it the name of the executable

```
gdb ./myprogx
```

- the debugger spews a bunch of text while initializing, then gives you a text prompt (gdb) and waits for commands

basic gdb commands

- to exit the debugger, enter the command `quit`
- to run the program, enter the command `run`
- if the program crashes, enter the command `backtrace`

(this shows you the sequence of function calls that were active when it crashed, and which lines of the program those calls came from)

setting breakpoints

- before we `run` the code, we can tell the debugger to pause if/when it gets to a specific line or function (called setting a breakpoint)
- to stop in a function use: `break` *`thefuntionname`*
- to stop at a line number use: `break` *`thelinenumber`*
- gdb will run the program normally until it hits a breakpoint, then will pause and wait for you to enter debugger commands

stepping through code

- when paused (e.g. at a breakpoint) we can go through one instructions at a time using either `s` or `step`
- if we want to treat a function call as one step (rather than going into the function and stepping through the inside of it) we can use `next` or `n`
- if we want to resume running normally (until the next breakpoint) we can use `continue` or `c`

Examining variables/parameters

- when paused while inside a function we can get gdb to display the current value of a variable or parameter using
- `p` or `print` and the variable or parameter name, e.g. `p x`

more info

- there are many many more gdb commands and options, and many other debuggers available
- for more on gdb in our systems, see the csci 265 notes
  - csci.viu.ca/~wesselsd/courses/csci265/slides/gdb.pdf
  - www.youtube.com/watch?v=qaUMwRUi6wc