CS2113: Lab 2

Debugging

Modified by Bo Mei

Finding Bugs

- A debugger lets you
  - Run your code line by line
  - Examine variables as code runs
  - Trace back runtime errors
- Step 1: compile with -g flag

```
gcc -g file.c
```

- Step 2: open program with debugger

```
gdb a.out
```

- Step 3: add breakpoints and run

```
break 37
break printList
run
```

stop on line 37

stop at function

gdb Reference

- **`break [line number]/[function name]`**- set breakpoint
  - Compiler will pause program at breakpoint so you can inspect
- **`delete`** – delete all breakpoints
- **`clear [line number]/[function name]`**- delete a specific breakpoint
- **`bt`** - print backtrace
  - Lists all functions on the stack that led to current execution point
- **`c`** - continue until next breakpoint
- **`n`** - next line, stepping over function calls
- **`s`** - step a line, stepping into function calls
- **`p x`** - print out the value of variable "x"
- **`l`** - print out 10 lines of source code for context
- **`r`** – start / restarts execution of the program
- **`kill`** - stop program execution
- **`q`** - quit gdb debugger
- **`help [command]`** - display useful information

next vs step

- Use **n** to go to the next line, passing over functions
  - The function is run, but the debugger doesn't pause
- Use **s** to go step to the next line, pausing on the next line inside the function

```
i = 10;
j = longfunc(i);
printf("j=%d", j);
k = 45;
```

s will step inside longfunc() and pause

n will run through longfunc(), then pause

break vs watch

- Use **break** to stop at a particular line or function

```
break 36
break uniform.c:45
break func
```

- Use **watch** to stop when the condition of a variable changes
    - You can only set a watchpoint on a variable in the current function

```
watch x
watch y == 125
watch *z > 45
```

Try gdb

- git clone https://github.com/cs2113f16/lab-gdb.git
- cd lab-gdb
- ./install-gdb

HaikuGen2000

- #include <time.h>
  - For srand(time(NULL))
- rand() – Randomly return an integer from 0 to a relatively large number (pre-defined by the system)
  - Rand() % 20: randomly generate an int from 0 to 19
- The number of bytes that are reserved using malloc() is hard to trace afterwards.

```
char *q = malloc(20);
int s = sizeof(q); // can't get 20
```

  - **Solution**: Record the number of bytes at the time when using malloc()

HaikuGen2000

- Pay attention to the number of elements in a char array.
    - When to use strlen(line), strlen(line) + 1 or strlen(line) – 1
- Makefile
    - Use unexpanded tab. Copy the tab from Github: cs2113f16/lec-4/Makefile