CSE 303
Concepts and Tools for Software Development

Magdalena Balazinska

Winter 2010

Lecture 27 – Final Exam Revision

Today

- Final review session!
  - The final is not yet written, but you know roughly what it will cover

- We'll discuss your questions about practice problems
  - Did you bring some?
  - I also have a few

Final exam reminder

- Monday, March 15th @ 8:30-10:20, MGH 241
- Content: Lecture 7 and following
  - There will be questions about tools: svn, debugger, makefiles
- CLOSED book and closed notes
- EXCEPT for **two** 8.5''x11'' pages
  - 7pt font or higher or written manually
  - Both sides

What did we do in 303?

- Using Linux, bash scripting (lectures 1-6)
  - Not on the final
- C (lectures 7-9, 12, 13, 15)
- C++ (lectures 17-20)
- Tools – svn, make, debugger, … (lectures 10, 11, 15, 16)
- Engineering issues – specs, testing, readability, concurrency (lectures 22, 23, 25)

C = Pointers

- … sort of
- 3 regions: stack, heap, static data
  - Don't mix them
- **Never trust any pointer**
  - Memory leaks, dangling pointers, and more
- Arrays = pointers
  - Arrays have no bounds checking or built-in length information
  - Avoid buffer overflow – never trust any user-provided array or buffer

Bad pointer = crash! (we hope)

- Examine `bad-pointer.c` (from sample final 2)
- Why does it crash?

`free()` misused

Consider this binary tree struct:

```
struct Tree {
  int val;
  struct Tree *left, *right;
};
```

What's wrong with this?

```
void free_tree_1(struct Tree *t) {
  if (t == NULL) return;
  free(t);
}
```

This example also in `free-tree.c`

`free()` misused, take 2

What about this?

```
void free_tree_2(struct Tree *t) {
  if (t == NULL) return;
  free(t);
  free_tree_2(t->left);
  free_tree_2(t->right);
}
```

Reminder:

```
struct Tree {
  int val;
  struct Tree *left, *right;
};
```

`free()` misused, take 3

How about now?

```
void free_tree_3(struct Tree *t) {
  if (t == NULL) return;
  free_tree_3(t->left);
  free_tree_3(t->right);
  free(t);
}
```

Reminder:

```
struct Tree {
  int val;
  struct Tree *left, *right;
};
```

C++ objects – like Java, but harder

- Creating objects on the stack vs. the heap
  - Stack: object-valued variables, copying object state with copy ctor
  - Heap: object pointer variables, only the pointer is copied

- Virtual functions
  - Needed to invoke the method implementation appropriate to the *runtime* class of the object
  - Destructors must be virtual (unless you can guarantee that no one ever holds a parent class pointer)

Stack vs. heap

- Examine `stack-vs-heap.cc`

- Why does this run `Parent`'s method?

```
Child ch;
Parent pa = ch;
pa.print();
```

- How about this?

```
Parent *ppa = &ch;
ppa->print();
```

Virtual vs. non-virtual

- Examine `cows.h`, `cows-main.cc`
- When we call `fred`'s methods, which version runs – `Bovine`'s, or `Cow`'s?
- What about when we call the methods through `clarabell`?

Debugging example

- Consider this C code that transforms a variable `x`:

```
/* x is the program input */
int x2 = f1(x);
int x3 = f2(x2);
int x4 = f3(x3);
int x5 = f4(x4);
```

- The output, `x5`, is not what you expect
- How would you use `gdb` to find the function that produces the wrong output?

Debugging example – your answers

- How would you use `gdb` to find the function that produces the wrong output?
  - .

Makefiles

- Why use makefiles?
  - .
- Why not write a bash script?
  - .

Makefile example

Consider this makefile (in `sample-makefile`):

```
all: my_program

my_program: main.o alicelib.o boblib.o

  gcc -g main.o alicelib.o boblib.o -o my_program

%.o: %.c %.h

  gcc -g -c $< -o $@
```

We've just run `make`.

- What commands run if we delete `main.o` and rerun `make`?
- What files can we delete so that only one command is executed when rerunning `make`?

Version control

- How can you share files in a team project?

- What makes version control a good way to share?

- What shouldn’t you put in version control?

Writing a specification

A vague specification:

```
/** Compute number of times word appears in a file. */
int count_occurrences (char *filename, char *word);
```

How can we improve on this spec?

- .

Writing a specification, continued

Reminder:

```
/** Compute number of times word appears in a file. */
int count_occurrences (char *filename, char *word);
```

- More ideas?

Testing example

Using the previous example:

```
/** Compute number of times word appears in a file. */
int count_occurrences (char *filename, char *word);
```

Give an example equivalence class of test cases.

- Example: cases with `word == NULL`
- .

Testing example, continued

Reminder:

```
/** Compute number of times word appears in a file. */
int count_occurrences (char *filename, char *word);
```

- More ideas?

A bit of concurrency

Example code:

```
int total = 0; // global variable
void inc_total (int incr) {
  int new_total = total + incr;
  total = new_total;
}
```

- Multiple threads call `inc_total`, read `total` directly
- The final value of `total` keeps changing; it even varies between executions with the same input
- Why is this happening?

A bit of concurrency, continued

Reminder:

```
int total = 0; // global variable
void inc_total (int incr) {
  int new_total = total + incr;
  total = new_total;
}
```

- How can we fix the problem?