CSC 161

Fall 2024

Exam #2 Review

Announcements

- *"Hi, I'm your new instructor!"*
  - Peter-Michael Osera, *i.e.*, *PM*
  - osera@cs.grinnell.edu
- Logistics forthcoming!
  - Syllabus, grading, *etc.*, unchanged
  - Minor schedule changes
- Exam #2, W 11/8
  - Covers weeks 6–9
  - Mentor session: Tuesday

Exam #2 Learning Outcomes

1. In the context of low-level, systems programming
   - Identify the various memory safety issues that commonly affect programs
   - Reason about the data representation and behavior of existing programs
   - Author effectful programs using fundamental imperative operations, *e.g.*, sequenced statements, conditions, and loops
   - Author small, memory-safe programs that interact with the world
2. Identify the three primary memory safety issues that arise in C programs:
   - Access errors
   - Memory leaks
   - Uninitialized values

Exam #2 Learning Outcomes (Cont.)

1. Reason about the data representation and behavior of existing programs:
   - Reason about a C program using a stack-and-heap model of computation
2. Author small, memory-safe programs that interact with the world
   - Author a C program involving static types, the pre-processor, mutable variables, functions, loops, and structs
   - Author C programs in a const-correct style
   - Author a C program involving pointers, arrays, and strings
   - Author a C program involving IO and dynamically allocated memory
   - Employ memory safety strategies to architect a memory-safe C program

From Outcomes to Questions

1. Manipulate **bit-level representations** of numbers (week 6)
2. Trace through code involving **stack and heap manipulation** (week 9)
3. Identify **memory safety issues** in existing code (week 9)
4. Write a program that processes **strings** and **arrays** (week 7–8)
5. Write a program using **structs** (week 8)
6. Write a program using **dynamically allocated memory** (week 9)

Bit-level representation

Concepts

- What is:
  - Sign magnitude representation?
  - One's complement?
  - Two's complement?
- What are the components of an IEEE floating point number?

Practice

1. Convert the following decimal (base-10) number to an 8-bit binary number: **163**.
2. Convert the following two's complement binary number to a decimal number: **10011101**.

(Additional practice: convert numbers between base-10 and base-2; use an online calculator to check your work.)

Stack-and-Heap Manipulation

Concepts

- What is the stack and heap?
- When do we allocate values on the stack versus the heap?
- How are pointer values stored in a C program?

Practice

Give stack-and-heap diagrams for each of the points indicated in the program on the following page.

(Additional practice: trace through code snippets on `pythontutor.com`.)

```c
void f(int *x, int y) {
  // POINT A
  int* z = (int*)
    malloc(sizeof(int));
  *z = *x + y;
  *x = *z * 2;
  y = 0;
  // POINT B
}
```

```c
int main() {
  int q = 4;
  int r = 7;
  f(&q, r);
  // POINT C
}
```

Memory Safety Issues

Concepts

- What are the three primary memory safety issues that occur in C programs?

Practice

Identify the memory error(s) in the code to the right. For each memory error, identify the (a) offending line number, (b) the type of error, and (c) a fix for the memory error.

(Additional practice: trace through bad code snippets on `pythontutor.com`.)

```c
int* f(int *arr) {
  int result [] =
    { 0, 0, 0, 0, 0 };
  for (int i = 0; i < 5; i++) {
    result[i] = arr[i] * 2;
  }
  return result;
}
```

```c
int main() {
  int *arr = (int*)
    malloc(sizeof(int) * 5);
  for (int i = 0; i < 5; i++) {
    arr[i] = i;
  }
  int *result = f(arr);
  printf("%d\n", result[4]);
  free(arr);
}
```

Strings and Arrays

Concepts

- How do you declare a string/array (local) variable? Literal?
- How do you access an element of a string/array?
- How do you iterate over a string/array?
- What standard library functions are available for `char`s? Strings?

Practice

Write a function `void string_to_upper(char *str)` that takes a mutable C-string `str` as input and mutates `str` so that all of its alphabetical letters are uppercase. (*Hint*: functions from `ctype.h` will be useful here.)

Structs

Concepts

- How do you model data using a `struct`?
- How do you use (initialize/access) a `struct` value?

Practice

Consider writing a game where a player is represented by a ball with:

- A *position*, represented as integral coordinates in 2D space, `x` and `y`.
- A *color*, represented as a non-NULL C-string. Whenever the player moves to the origin `(0, 0)`, their color becomes "`green`". Otherwise, their color is "`red`".

Initially, the player begins at the origin.

1. Create a `struct player` declaration with the appropriate fields.
2. Write a function `struct player make_player(void)` that creates and returns a new `player` value.
3. Write a function `void move (struct player *p, int dx, int dy)` that mutates `player p` by moving them by distances `dx` and `dy` in the `x` and `y` directions, respectively.

Dynamically Allocated Memory

Concepts

- How do I allocate memory on the heap?
- How do I free memory on the heap when I am done with it?

Practice

Write a function `int* replicate(int n, int k)` that takes two numbers n and k as input and returns a heap-allocated array of size n. Each element of the outputted array is initialized with value k.