15-213 Recitation: Data Lab

Sep 13, 2021

Agenda

- Introduction
- Course Details
- Office Hours
- Data Lab
  - Getting started
  - Running your code
  - ANSI C
  - Reminders
- Floating Point

Introduction

- Welcome to 15-213/14-513/15-513!
- Recitations are for…
  - Reviewing lectures
  - Discussing homework problems
  - Interactively exploring concepts
  - Previewing future lecture material

- Please, **please** ask questions!

Course Details

- How do I get help?
  - Course website: http://cs.cmu.edu/~213
  - Office hours
  - Piazza
  - *Definitely* consult the course textbook
  - **Carefully read the assignment writeups!**
- All labs are submitted on Autolab.
- All labs should be worked on using the **shark machines.**

Office Hours

- **6PM-10PM** on Zoom and in-person (Sun-Fri)
- **11:30AM - 1:30PM** (Wed, Fri)
- Queue link: https://cmqueue.xyz/ **NEW LINK**
- Please locate the TA in the specified location: https://piazza.com/class/kr9vqwncw253c4?cid=284

OH Etiquette

- Office hours are for getting ideas on how to debug or better approach your homework- conceptual OH coming soon as well so look out for that!
- Please try to narrow down your problem area as much as possible to help TAs help you!
- **Write a description!** If you don't have a description, you may be frozen/removed from the queue.
- We will close the queue early so everyone can be helped by around 9:30pm so please keep this in mind!
- Please find the TAs at the carrels. TAs should not need to find you

Data Lab: Getting Started

- Download the handout from autolab
  - `scp <path to datalab.tar> <my course directory>`
  - `ssh <andrewid>@shark.ics.cs.cmu.edu`
  - `cd to the datalab.tar file`
  - `tar -xf datalab.tar`
- Upload `bits.c` file to Autolab for submission

Data Lab: Running your code

- `dlc`: a modified C compiler that interprets *ANSI C* **only**
- `btest`: runs your solutions on random values
- `bddcheck`: exhaustively tests your solutions
  - Checks all values, formally verifying the solution
- `driver.pl`: Runs both dlc and bddcheck
  - Exactly matches Autolab's grading script
  - You will likely only need to submit once
- For more information, **read the writeup**
  - Available under autolab as "**View writeup**"
  - **Read the writeup please!**

Data Lab: What is ANSI C?

Within two braces, all *declarations* must go before any *expressions*.

This is *not* ANSI C.

```
unsigned int foo(unsigned int x)
{
    x = x * 2;
    int y = 5;

    if (x > 5) {
        x = x * 3;
        int z = 4;
        x = x * z;
    }

    return x * y;
}
```

Data Lab: What is ANSI C?

This is ANSI C.

```
unsigned int foo(unsigned int x)
{
    int y = 5;
    x = x * 2;

    if (x > 5) {
        int z = 4;
        x = x * 3;
        x = x * z;
    }

    return x * y;
}
```

This is *not* ANSI C.

```
unsigned int foo(unsigned int x)
{
    x = x * 2;
    int y = 5;

    if (x > 5) {
        x = x * 3;
        int z = 4;
        x = x * z;
    }

    return x * y;
}
```

Data Lab: Reminders

- Casting between **int** and **long** is ok, in either direction
- Be aware of operations and their types!
  - ! returns an **int** *even if the argument is a long*
- Good idea to append “L” suffix to every integer constant
  - `(1L << 63)` is not the same as `1 << 63`
  - `(!x << 63)` is not the same as `((long)!x) << 63`

Form Groups of 3 - 4

- Series of exercises
  - Operators
  - Puzzle

Questions?

- Remember, data lab is due this Thursday!
  - You really should have started already!
- Read the lab writeup!

Looking Ahead… Bomblab!!

What is Bomb Lab?

- An exercise in reading x86-64 assembly code.
- A chance to practice using GDB (a debugger).
- Why?
  - x86 assembly is low level machine code. Useful for understanding security exploits or tuning performance.
  - GDB can save you days of work in future labs **(Malloc)** and can be helpful long after you finish this class.

Downloading Your Bomb

- Here are some highlights of the write-up:
  - Bombs can only run on the shark machines. They fail if you run them locally or on another CMU server.
  - Each bomb is unique - if you download a second bomb, bad things can happen! Stick to only one bomb.
  - Bombs have six phases which get progressively harder.
  - Make sure to read the writeup for more tips and common mistakes you might make.

Detonating Your Bomb

- Blowing up your bomb automatically notifies Autolab
  - ***Dr. Evil*** deducts 0.5 points each time the bomb explodes.
  - It's very easy to prevent explosions using **break points** in GDB. More information on that soon.
- Inputting the correct string moves you to the next phase.
- Don't tamper with the bomb. Skipping or jumping between phases detonates the bomb.
- You have to solve the phases in order they are given. Finishing a phase also notifies Autolab automatically.

Bomb Hints

- ***Dr. Evil*** may be evil, but he isn't cruel. You may assume that functions do what their name implies
  - i.e. phase_1() is most likely the first phase. printf() is just printf(). If there is an explode_bomb() function, it would probably help to set a breakpoint there!
- Use the man pages for library functions.
  - Although you can examine the assembly for snprintf(), we assure you that it's easier to use the man pages ($ man snprintf) than to decipher assembly code for system calls.
- Most cryptic function calls you'll see (e.g. callq … <_exit@plt>) are also calls to C library functions.
  - You can safely ignore the @plt as that refers to dynamic linking.

S21 Bomblab Slides

https://docs.google.com/presentation/d/1c9IVmK69sVndzX5_rZYaL-jj5uk_d3GI/edit?usp=sharing&ouid=105056271954280155624&rtpof=true&sd=true