# IC221: Systems Programming
# 6-Week Practicum

12 Feb 2015

**Instructions:**

- This is a 1 hour practicum.
- The practicum is open notes, open book, and open internet. Previous lab submissions count as notes. You may not, however, communicate with anyone directly, e.g., via instant message, stack overflow, or etc.
- There are four practicum problems. If you complete 1/4, you will receive a 60%, 2/4 you will receive an 80%, 3/4 you will receive a 90%, and 4/4 you will receive 100%.
- All problems are graded pass/fail. No partial credit.
- There is 5 points of extra credit on one of the problems. You must succesfully complete the problem to receive the extra credit.
- To retrieve the source, run `~aviv/bin/ic221-up`, then `cd ~/ic221/practicum/6-week`
- All code submissions will occur via the `~aviv/bin/ic221-submit` with the option `practicum/6-week`. You can submit multiple times, only your final submission will be graded.
- You must also turn in this document at the completion of the examination. Feel free to include any notes or comments in writing within this document.
- You can **test your submission** by running the test script `test.sh`

Name: _____

Alpha: _____

1. This problem refers to the starter code in the `bash` directory. Your task is to complete the program `inrange.sh` which takes the following arguments:

   ```
   ./inrange.sh low high test
   ```

   The output of the program should be `yes` if `test` is greater than or equal to `low` and less than or equal to `high`. Additionally, `inrange.sh` should exit with value 0 if the `test` is in range, and exit with value 1 if the `test` is not in range.

   Below is some sample output:

   ```
   aviv@saddleback: bash $ ./inrange.sh 0 2 0
   yes
   aviv@saddleback: bash $ ./inrange.sh 0 2 1
   yes
   aviv@saddleback: bash $ ./inrange.sh 0 2 2
   yes
   aviv@saddleback: bash $ ./inrange.sh 0 2 3
   no
   aviv@saddleback: bash $ ./inrange.sh 0 2 -1
   no
   aviv@saddleback: bash $ ./inrange.sh 0 2 1 >/dev/null; echo $?
   0
   aviv@saddleback: bash $ ./inrange.sh 0 2 3 >/dev/null; echo $?
   1
   ```

   **YOU SHOULD ONLY EDIT THE SCRIPT WHERE THE `TODO` IS MARKED.**

2. This problem refers to the starter code in the `dynamicmem` directory. Your task is to **fix** the program `cmdlist.c` which has a memory leak. The program implements a very basic link list, storing all the command line arguments, and printing them out in reverse, one per line. When functioning properly, without memory leaks, you should expect the following output:

   ```
   aviv@saddleback: dynamicmem $ valgrind --leak-check=full ./cmdlist go navy beat army
   ==17582== Memcheck, a memory error detector
   ==17582== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
   ==17582== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
   ==17582== Command: ./cmdlist go navy beat army
   ==17582==
   army
   beat
   navy
   go
   ./cmdlist
   ==17582==
   ==17582== HEAP SUMMARY:
   ==17582==     in use at exit: 0 bytes in 0 blocks
   ==17582==   total heap usage: 10 allocs, 10 frees, 108 bytes allocated
   ==17582==
   ==17582== All heap blocks were freed -- no leaks are possible
   ==17582==
   ==17582== For counts of detected and suppressed errors, rerun with: -v
   ==17582== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
   ```

   **YOU MAY ONLY ADD `free()`'S TO THE PROGRAM AND MAY NOT EDIT IT ANY OTHER WAY.**

3. This problem refers to the starter code in the `pointers` directory. Your task is to edit the program `casting.c` such that it can access the `unsigned int`'s stored in the string `buffer`. You should print out each of the integers one per line using the `%u` format. When functioning correctly, your output should be:

```
aviv@saddleback: pointers $ ./casting
10
20
2147483647
4294967295
```

**YOU MAY NOT JUST PRINT THAT OUTPUT BUT INSTEAD USE POINTER CASTING OVER THE STRING.** The test script will check for that.

4. This problem refers to the starter code in the `strings` directory. Your task is to edit the program `commandcheck.c` such that it properly checks the arguments against the `args` string. To do this, you must fill in the string `buf` with all the arguments from `argv[1]` and onward. When functioning correctly, your program will have the following output:

```
aviv@saddleback: strings $ ./commandcheck go navy beat army
yes
aviv@saddleback: strings $ ./commandcheck go navy beat
no
```

**(5pt extra credit)** Complete `commandcheck.c` such that it does not have a buffer overflow or a segmentation fault on large argument input.

**YOU SHOULD ONLY EDIT THE PROGRAM WHERE THE `TODO` IS MARKED.**