15-213 Recitation 6: C Review

30 Sept 2016

Agenda

- Reminders
- Lessons from Attack Lab
- C Assessment
- Programming Style
- Cache Lab Overview
- Appendix: valgrind
- Appendix: Clang / LLVM

Reminders

- Attack Lab is due **tomorrow!**
- "But if you wait until the last minute, it only takes a minute!" - ***NOT!***
- Cache Lab will be released **tomorrow!**

Image credit: pixabay.com

Lessons from Attack Lab

- **Never**, **ever** use gets
  - use fgets instead if you need that functionality
- Use functions that pass an explicit buffer length if possible
  - strncpy/strncat instead of strcpy/strcat, snprintf instead of sprintf
  - Limit scanf/fscanf input lengths with %123s
- Or use a function that dynamically allocates a large-enough buffer
  - asprintf (GNU library) instead of sprintf
- If none of those is possible, be **very** careful about checking input size
- Stack protections make it harder to exploit a buffer overflow – but not impossible

C Assessment

- Can you **easily** answer all of the problems on the following slides?
  - For each question, take a minute to write down your answer
- If not, please come to the C Bootcamp:
  - Wednesday 7:30-9pm, Location TBD
- You need this for the rest of the course. **If in doubt, come to the C Bootcamp!**

C Question 1

Which of the following lines has a problem?

If it does, how might you solve it?

```
    int main(int argc, char** argv) {
1      int *a = malloc(100 * sizeof(int));
2      for (int i=0; i<100; i++) {
3         if (a[i] == 0) a[i]=i;
4         else a[i]=0;
       }
       ...
5      free(a);
6      return 0;
    }
```

C Question 1

What can malloc return? Can malloc fail?

```
int main(int argc, char** argv) {
   int *a = malloc(100 * sizeof(int));
   for (int i=0; i<100; i++) {
      if (a[i] == 0) a[i]=i;
      else a[i]=0;
   }
   ...
   free(a);
   return 0;
}
```

C Question 1

Allocated memory is not initialized.

What function does this?

```
int main(int argc, char** argv) {
   int *a = malloc(100 * sizeof(int));
   for (int i=0; i<100; i++) {
      if (a[i] == 0) a[i]=i;
      else a[i]=0;
   }
   ...
   free(a);
   return 0;
}
```

C Question 1 (bonus)

Declaring a variable in a for loop requires:

-std=c99 (or later standard)

```
int main(int argc, char** argv) {
   int *a = malloc(100 * sizeof(int));
   for (int i=0; i<100; i++) {
      if (a[i] == 0) a[i]=i;
      else a[i]=0;
   }
   ...
   free(a);
   return 0;
}
```

C Question 1

The code has been revised to address the two problems.

```
int main(int argc, char** argv) {
   int *a = calloc(100 * sizeof(int));
   if (a == NULL) { ...}
   for (int i=0; i<100; i++) {
      if (a[i] == 0) a[i]=i;
      else a[i]=0;
   }
   ...
   free(a);
   return 0;
}
```

C Question 2

- What is the value of A and B? Why?

```
#define IS_GREATER(a, b) a > b

int is_greater(int a, int b) {
   return a > b;
}

int A = IS_GREATER(1, 0) + 1;
int B = is_greater(1, 0) + 1;
```

C Question 2

A uses a macro, which does textual substitution

Following the order of operations: 1 > 0 + 1 => 1 > 1 => 0

```
#define IS_GREATER(a, b) a > b

int is_greater(int a, int b) {
   return a > b;
}

int A = 1 > 0 + 1;
int B = is_greater(1, 0) + 1;
```

C Question 2

B uses a function call and behaves as expected:

B = 1 + 1 => 2

```
#define IS_GREATER(a, b) a > b

int is_greater(int a, int b) {
   return a > b;
}

int A = IS_GREATER(1, 0) + 1;
int B = is_greater(1, 0) + 1;
```

C Question 3

Which of the following lines has a problem?

How would you solve the problem(s)?

```
   int *foo(int *allocate) {
1     int a = 3;
2     allocate = malloc(sizeof(int));
3     if (allocate == NULL) abort();
4     return &a;
   }
```

C Question 3

allocate is a local copy of the pointer

- “*allocate =’’ assigns to the caller’s location
- To allocate for the caller, foo(int **allocate)

```
int *foo(int *allocate) {
   int a = 3;
   allocate = malloc(sizeof(int));
   if (allocate == NULL) abort();
   return &a;
}
```

C Question 3

Where is a? To where does &a point?

```
int *foo(int *allocate) {
   int a = 3;
   allocate = malloc(sizeof(int));
   if (allocate == NULL) abort();
   return &a;
}
```

C Assessment

- Did you know the answers to all of the problems? If not,

COME TO THE C BOOTCAMP

C Programming Style

- Properly document your code
  - Header comments, overall operation of large blocks, any tricky bits
- Write robust code – check error and failure conditions
- Write modular code
  - Use interfaces for data structures, e.g. create/insert/remove/free functions for a linked list
  - No magic numbers – use #define
- Formatting
  - 80 characters per line
  - Consistent braces and whitespace
- No memory or file descriptor leaks

C Programming Exercise

- Learn to use getopt
  - Complete the code to process the commandline
  - Write a simple calculator program

Form pairs

- One student needs a laptop
- Login to a shark machine

```
$ wget http://www.cs.cmu.edu/~213/activities/rec6.tar
$ tar xf rec6.tar
$ cd rec6
$ make
```

man 3 getopt

```
int getopt(int argc, char * const argv[],
           const char *optstring);
```

- If there are no more option characters, getopt() returns -1.
- optstring is a string containing the legitimate option characters.
    - If such a character is followed by a colon, the option requires an argument
        - getopt() places a pointer to the following text in optarg
    - getopt() finds an option character in argv that was not included in optstring, or if it detects a missing option argument, it returns '?'

If You Get Stuck on cachelab

- **Please read the writeup.** ***Please read the writeup.*** ***Please read the writeup.*** ***Please read the writeup!***
- CS:APP Chapter 6
- View lecture notes and course FAQ at http://www.cs.cmu.edu/~213
- Office hours Sunday through Thursday 5:00-9:00pm in WeH 5207
- Post a **private** question on Piazza
- `man malloc`, `man valgrind`, `man gdb`, gdb's `help` command

KEEP

CALM

and

READ

THE

WRITEUP

Appendix: valgrind

- A suite of tools for debugging and profiling memory use, among other things
  - find where memory that wasn't freed was allocated
  - track origin of uninitialized values
  - show heap usage over time
  - detect reads and writes of invalid locations
  - detect illegal and double frees

valgrind: Finding Memory Leaks

- valgrind --leak-resolution=high --leak-check=full --show-reachable=yes --track-fds=yes *./my_prog <args>*
  - your program runs as normal, though much, **much** slower
- read/write errors and uses of uninitialized values are reported as they occur
- un-freed memory is reported on program termination

Clang / LLVM

- Cachelab – Part B Matrix Transpose

- Clang is a gcc-equivalent C compiler
  - Support for code analysis and transformation
- New methods of style checking and trace generation
  - Compiler will check your variable usage and declarations
  - Compiler will also instrument the code to record all memory accesses to a file