CS 241: Systems Programming
Lecture 33. Variadic Functions

Fall 2019

Prof. Stephen Checkoway

Student evals are online

Primary learning goals from course website

- the UNIX command line (in particular the BASH shell)
- a command line editor like Neovim, Emacs, or Nano
- Various command line utilities
- the Git version control system
- C compilers like Clang and GCC
- debuggers like GDB
- linting tools like shellcheck.

More learning goals

More learning goals

- how to write safe shell scripts (specifically BASH-flavored shell scripts);
- how and especially when to program in C;
- what undefined behavior is;
- what memory safety is;
- how to use Github;
- how to set up continuous integration with Travis-CI; and
- how to work with regular expressions.

Projects

Completed project and 2 page report due on Friday!

Presentations are on Monday and Wednesday of next week

Report

A two page (maximum!) write up

- standalone description of your project
- what you accomplished
- what you weren't able to get to
- what you found most challenging
- anything else you think I should know

Demo and presentation

Last week of class (there will be a sign up for the day later in the semester)

Spend 7 minutes showing off and talking about your project

- 5 minutes of talking; 2 minutes of answering questions
- I know public speaking is *awful* (unless you enjoy it), but this is a super low-stakes way to get practice at it in a supportive environment
- Everybody must speak
- (Attendance at both days of presentations is mandatory, I will check with clickers)
- Tell us who you are, what you did, and how you did it (tell us what didn't work if you like)
- Show off some features
- 👏 👏 👏 Get some applause 👏 👏 👏

Asking questions

Each presentation has 2 minutes of question time built in

You must ask questions

Strategies for asking good questions

- During each presentation, think of a question and write it down so you don't forget
- Think about how the project might be extended or design choices made; ask about those
- Ask about some particular functionality
- Don't be a jerk or a show off (not that anyone here would be); ask polite questions

Variable Arguments

Need a way to handle variable length argument lists

- `printf`, `scanf`, etc.
- `execl`
- `open`, `fcntl` — additional parameter when given specific flags

Ideally, have argument checking for fixed parameters

- type checking catches many errors
- allows for compiler optimizations

Variable arguments in C

Two mechanisms (used to be) available:

```
#include <varargs.h>
```

- Old style, not supported — do not use!

```
#include <stdarg.h>
```

- New style — do use!

Types

Somewhere in `stdarg.h` there is

```
typedef /* stuff */ va_list;
```

Need one of these for argument pointer

```
va_list ap;
```

Function prototypes

Use "`...`" in function prototype

```
void varfoo(char const *fmt, ... );
```

Variable arguments must be

- At the end
- Following at least one non-variable argument

Using variable arguments

Three macros used

- `va_start(va_list ap, last)`
- `va_arg(va_list ap, type)`
- `va_end(va_list ap)`

There's a fourth one that's rarely used

- `va_copy(va_list dest, va_list src)`

va_start

Macro used to initialize argument pointer

```
va_start(ap, last);
```

- `ap` — argument pointer
  - initialized to the first argument
- `last` — argument before variable arguments

va_arg

Macro used to access arguments

Returns next parameter in list; advances to the next position

Needs to know type for forward movement and reading

```
double dbl = va_arg(ap, double);
```

va_end

Macro to clean environment up when done

```
va_end(ap);
```

Each `va_start()` and `va_copy()` must be paired with a va_end() in the same function

When implementing a function with a variable number of arguments, how does the programmer know how many arguments there are?

A. Use the `va_number(va_list ap)` macro

B. Format string specifies the number of arguments

C. An explicit "sentinel" value is used at the end of the argument to mark the end

D. The number of additional arguments is passed as a parameter

E. *Some* mechanism must be used to indicate how many there; it varies by function

What happens if the program accesses more arguments than were passed to the function or an argument of the wrong type?

A. This is prevented by the type system (i.e., a compiler error)

B. The default value of 0 is returned

C. A garbage value is returned

D. The program segfaults

E. It's undefined behavior

```
void strange_print(int next, ...) {
  va_list ap;

  va_start(ap, next);
  while (1) {
    switch (next) {
    case 'i': printf("%d", va_arg(ap, int)); break;
    case 'f': printf("%f", va_arg(ap, double)); break;
    case 's': printf("%s", va_arg(ap, char *)); break;
    default: va_end(ap); return;
    }
    next = va_arg(ap, int);
  }
}

strange_print('i', 37, 's', "text", 'f', .25, 0);
```

Open (from musl libc)

Open takes a third parameter (the file system permissions) when creating a file

```
int open(const char *filename, int flags, ...) {
 mode_t mode = 0;
 if ((flags & O_CREAT) || (flags & O_TMPFILE) == O_TMPFILE) {
  va_list ap;
  va_start(ap, flags);
  mode = va_arg(ap, mode_t);
  va_end(ap);
 }
 // ...
}
```

Implementing printf via vfprintf

```
int printf(char const *fmt, ...) {
  va_list ap;
  va_start(ap, fmt);
  int ret = vfprintf(stdout, fmt, ap);
  va_end(ap);
  return ret;
}
```

Implementing `vfprintf` involves reading the format string character by character and deciding what argument to read next based on the character after a `%`

In-class exercise

https://checkoway.net/teaching/cs241/2019-fall/exercises/Lecture-33.html

Grab a laptop and a partner and try to get as much of that done as you can!