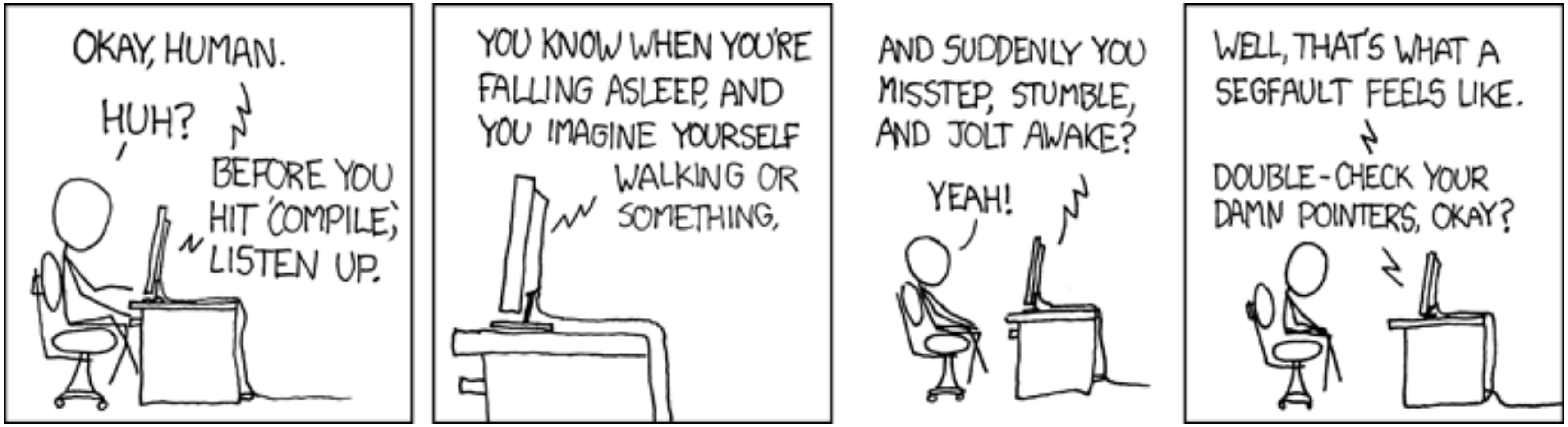
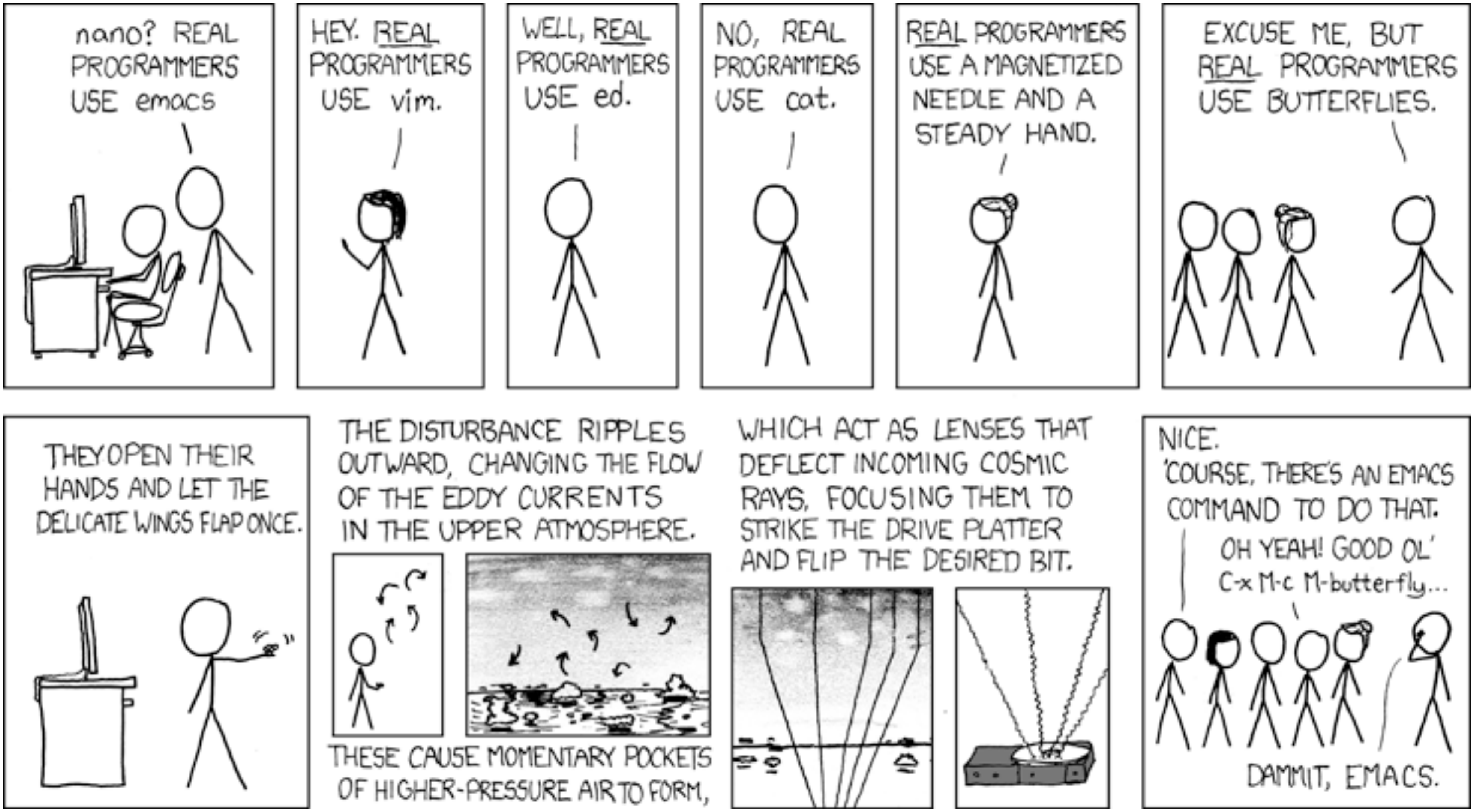
CSC209 Review


OKAY, HUMAN.
HUH?
BEFORE YOU HIT 'COMPILE,' LISTEN UP.
YOU KNOW WHEN YOU'RE FALLING ASLEEP, AND YOU IMAGINE YOURSELF WALKING OR SOMETHING,
AND SUDDENLY YOU MISSTEP, STUMBLE, AND JOLT AWAKE?
YEAH!
WELL, THAT'S WHAT A SEGFAULT FEELS LIKE.
DOUBLE-CHECK YOUR DAMN POINTERS, OKAY?

http://xkcd.com

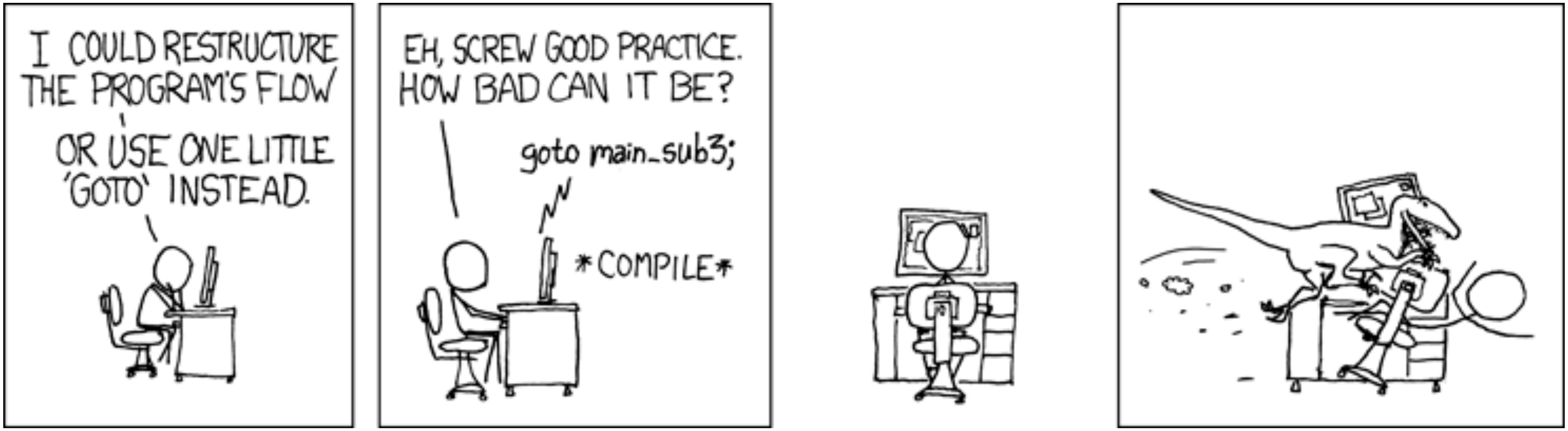

nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

http://imgs.xkcd.com/comics/real_programmers.png

Real programmers set the universal constants at the start such that the universe evolves to contain the disk with the data they want.

“Real programmers”

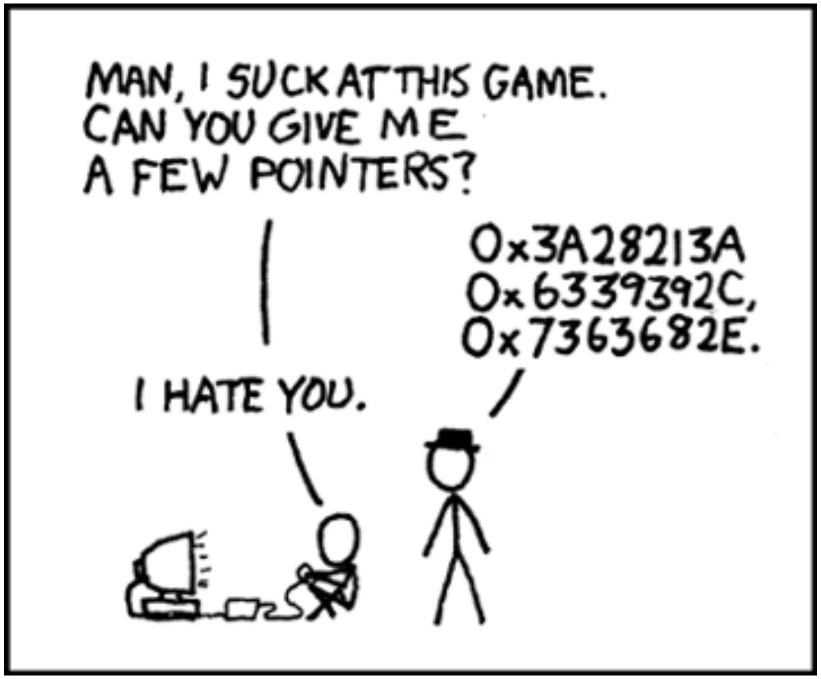
- use a good tool for the job
- write readable code
- check for errors and write readable error messages
- test their code
- collaborate
- use code review
- leave egos behind



I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE

http://imgs.xkcd.com/comics/goto.png

Remainder


MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A 0x6339392C, 0x7363682E.
I HATE YOU.

http://xkcd.com

- Please check schedule on course web site
- Review session
  - Tuesday April 9 1:00-3:00 Room TBA
  - Friday April 10 1-3 BA 1170
- Please submit remark requests promptly (after you get marks back…)
- *You will have a chance to verify posted marks before final marks are submitted. Please do check.*

CSC209: Software tools …

- Unix
  - files and directories
  - permissions
  - utilities/commands
- Shell
  - programming
  - quoting
  - wild cards
  - files

… and systems programming

- C
  - basic syntax
  - functions
  - bits
  - arrays
  - structs
  - strings
  - pointers (!!!)
  - function pointers
  - header files

…and more systems programming

- System calls
- Files
- Processes (fork, exec)
- Inter-process Communication
  - signals
  - pipes
  - sockets
  - select

What can you do?

- Write a shell script to automate some tasks.
  - Run some tests multiple times
- Write a program to run and monitor other programs
  - Kill them if they take too long
- Write a program that splits tasks into multiple processes to take advantage of multiple cores.
- Use a Makefile to build a large system

What else?

- Write a web server!
- Write a shell!

- But more importantly, you can begin to understand what happens when
  - A program “hangs”
  - A program “crashes”
  - Two programs share the same file
  - A process has terminated but is still in the process table.

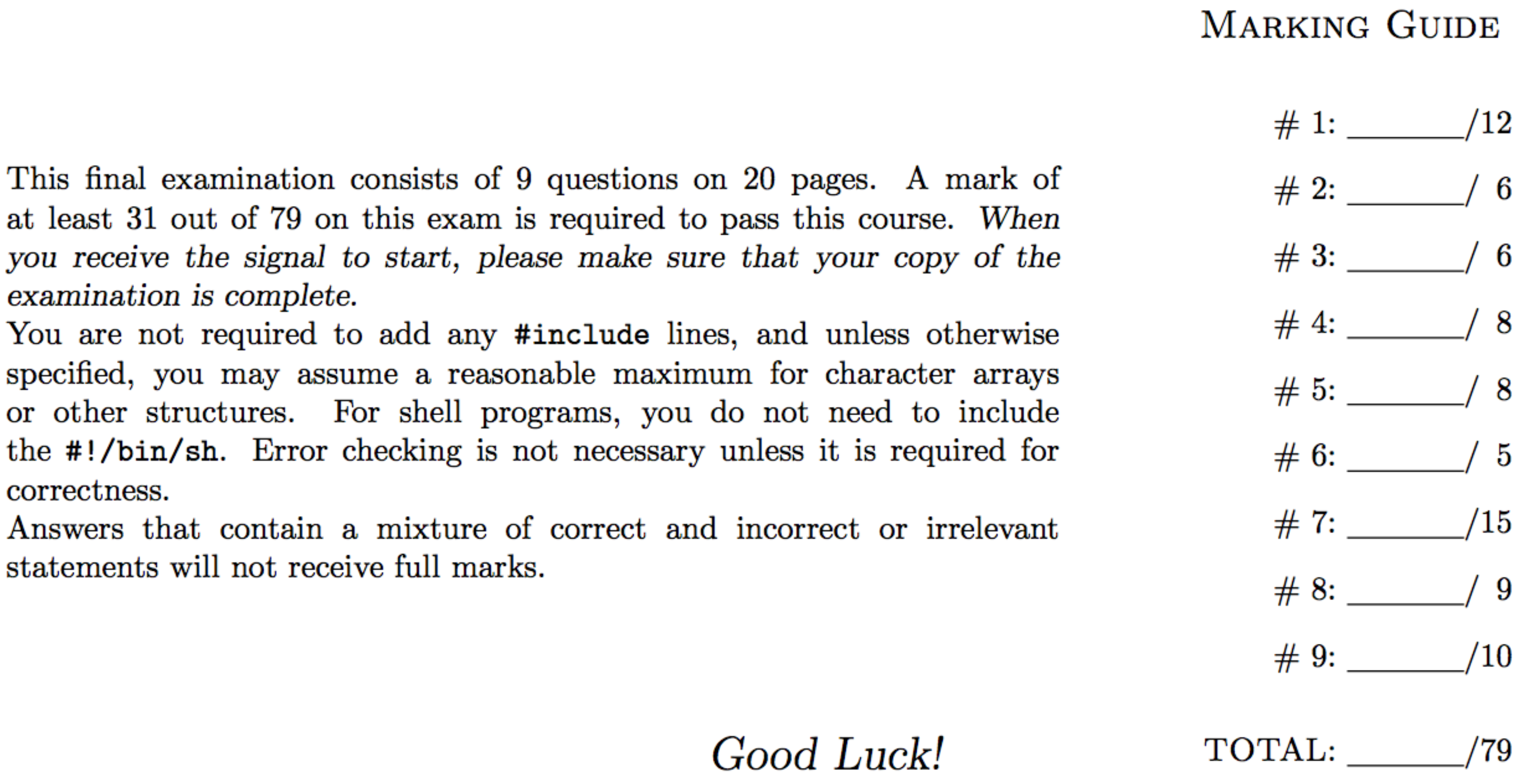
Final Exam

- How to study
  - Look at previous exams for structure.
  - Play with example code provided.
- Covers everything in the course
- Closed book exam except…
  - Bring one 8.5x11 sheet of paper
    - double-sided (no magnifying glasses allowed)
  - The exam also contains an aid sheet with prototypes and shell info.
    - It is already posted on the course web site

Final exam

- Topics
  - Shell
    - redirection, pipes, job control
    - permissions, file system navigation
    - environment variables
  - C
    - strings, pointers, structs, functions, bits, memory
  - Make and compilation
  - System calls
    - File I/O, fork, exec, pipe, signals, sockets

Final Exam Front page



Marking Guide

# 1:	_______/12
# 2:	_______/ 6
# 3:	_______/ 6
# 4:	_______/ 8
# 5:	_______/ 8
# 6:	_______/ 5
# 7:	_______/15
# 8:	_______/ 9
# 9:	_______/10
TOTAL:	_______/79

This final examination consists of 9 questions on 20 pages. A mark of at least 31 out of 79 on this exam is required to pass this course. *When you receive the signal to start, please make sure that your copy of the examination is complete.*

You are not required to add any `#include` lines, and unless otherwise specified, you may assume a reasonable maximum for character arrays or other structures. For shell programs, you do not need to include the `#!/bin/sh`. Error checking is not necessary unless it is required for correctness.

Answers that contain a mixture of correct and incorrect or irrelevant statements will not receive full marks.

Good Luck!

Shell Concepts

- Stdin, stdout, stderr
- I/O redirection
  - prog > outfile 2>&1 – same

redirect STDERR to STDOUT

- Job control
- Pipes

Bourne shell programming

- quoting
  - single quotes inhibit wildcard replacement, variable substitution and command substitution.
  - double quotes inhibit wildcard replacement only
  - back quotes cause command substitution.
- Command substitution
- variables – environment and local
  - `str1=“string”`
  - `str2=“string”`
  - `if test $str1 = $str2; then … fi`

Bourne shell programming

- `test -f filename` – test if a file exists
- Command line arguments
  - $0 = name of script, $1 .. $n = arguments
- `set` assigns positional parameters to a list of words.
- `read` – reads from stdin
- `expr` – math functions

Compiler vs. Interpreter

- Compiler translates whole program to object code.
  - produces the most highly optimized code
- Interpreter translates one line of code at a time.
  - can quickly make changes and try things out
- C – compiled
- Java – compiled to byte code, then interpreted.
- Shell –interpreted.

Software Tools

- Tools save you time and make you a better programmer:
  - editor, language choice, debugger, build system, version control system, regression testing, issue tracking, profiling and monitoring.
- High-level scripting languages make it possible to glue programs together to do all kinds of time-saving tasks.

Programs as Data

- Executables are just files that can be copied, moved, searched and even edited.
- Compilers are just programs that operate on source code and produce executables.
- Programming tools treat program source code as data
- High-level programming languages give us easier ways to operate on programs:
  - automated testing, build systems, version control.

Programming in C

- Memory model
  - pointers are addresses with a type
- Remember that no variables are automatically initialized.
- Arrays
  - contiguous region of memory with fixed size
- Pointers
  - dereference with *
  - get the address of a variable with &

Strings

- Remember the null termination character ('\0')
- Most string functions depend on it.
- Whenever possible use the string functions rather than re-implementing them.
- E.g., use `strncpy` rather than copying each character.
- Be careful to ensure that you don't walk of the end of a character array.

Dynamic memory allocation

- memory allocated using `malloc` should be freed when it is no longer needed (unless you are about to exit)
- keep a pointer to the beginning of the region so that it is possible to free
- memory leak occurs when you no longer have a pointer to a region of dynamically allocated memory

When to use malloc?

- when passing a pointer to a new region of memory back from a function.
- when you don't know until runtime how much space you need.
- This is a poor use of malloc:

```
main(){
   char *str1 = malloc(MAXLEN);
   ...
   free(str1)
   return 0;
}
```

Header files

- Header files contain function prototypes and type definitions.
- Never `#include` a file containing functions and variable declarations file. You will run into trouble.
- Header files are useful when your program is divided into multiple files.
- Use Makefiles to compile programs. Saves typing and takes advantage of separate compilation.

System Calls

- Perform a subroutine call into the Unix kernel
- Interface to the kernel
- main categories
  - file management
  - process management
  - error handling
  - communication
- Error handling
  - system calls usually return -1 (Always check!)
  - errno

Processes

- process state: running, ready, blocked
- `fork()` – creates a duplicate process
- `exec()` – replaces the program being run by a different one.
- file descriptors maintained across fork and exec
- process ids – `getpid()`, `getppid()`

Process Termination

- Orphan process:
  - a process whose parent is the init process because its original parent died
- Zombie process:
  - a process that is “waiting” for its parent to accept its termination status.

```
wait(int *status);

r = waitpid(pid_t pid, int *status,
  int options)
```

- Use macros to check the status:
  - WIFEXITED, WIFSIGNALED, WEXITSTATUS

Inter-process Communication (IPC)

- Data exchange between process:
  - message passing: files, pipes, sockets
- Limitations of files for IPC data exchange
  - slow
  - possibly altered by other processes
- Limitations of pipes:
  - two processes must be running on the same machine
  - two processes must be related
- Sockets overcome these limitations

Streams? File Descriptors?

- Unix has two main mechanisms for managing file access
  - streams: high-level, more abstract (and portable)
    - you deal with a pointer to a FILE structure, which keeps track of info you don’t need to know
    - `fopen(), fprintf(), fread(), fgets()`
  - file descriptors: each file identified by a small integer (on Unix), low-level, used for files, sockets and pipes.
  - Binary versus text I/O

Signals

- Signals are software interrupts, a way to handle asynchronous event.
- Examples: control-C, termination of child, floating point error, broken pipe.
- Normal processes can send signals.
- `kill(pid, SIG)` – sent SIG to pid
- `sigaction()` – install a new signal handler for a signal

Sockets

- Sockets allow communication between machines
- TCP/IP protocol – internet address, ports
- Protocol families: PF_INET, PF_LOCAL
- Server side initialization takes 4 steps
  - `socket()` – initialize protocol
  - `bind()` – initialize addresses
  - `listen()` – initialize kernel structures for pending connections
  - `accept()` – block until a connection is received.

Sockets

- Client initializes socket using `socket(),` and then calls `connect()`
- Need to be wary of host byte orders.
- Communication is done by reading and writing on file descriptors.
- Ports are divided into three categories: well-known, registered, and dynamic (or private).
- Socket types:
  - `SOCK_STREAM = TCP`
  - `SOCK_DGRAM = UDP`

Multiplexing I/O

- `select()` allows a process to block on a set of file descriptors until one or more of them are ready.
- Read calls on a “ready” file descriptor will only block while the data is transferred from kernel to user space.
- Makes it easier for one process to handle multiple sources of input.
- `select()` takes “file descriptor sets” as arguments
- The macros FD_SET, FD_ISSET etc. are used to manipulate the bit set data structure.

File interface

- “Everything is a file”
- We treat all sorts of devices as if they were files, and use the file interface (open, read, write, close) all over the place.
  - files
  - directories
  - pipes
  - sockets
  - kernel info via /proc

Unix Philosophy

- Write programs that do one thing well
- Write programs that work together
- Write programs to handle text streams because that is the universal interface.

All the best with your exams,

and have a good summer!