CS 222: The End is Nigh

Chris Kauffman

Week 8-1

Logistics

Reading

- Finish!
- Progress as of Mon posted on BB

Final Exam: Thursday

Homework 6 Up

- Due tonight

Goals

- HW 6 Questions
- Review of Data Structures
- Tools associated with C
- Limits of C and what's beyond
- Final exam review

HW 6: Questions?

Problem 1: I/O on `channel_params`

- `save_channel_params()`: write array to a file (with what function?)
- `load_channel_params()`: load array from a file

Problem 2: List Insertion `int_list_insert()`

- Discuss linked lists today
- Insert integer at an arbitrary index in list
- Requires traversing the list
- Deal with special cases

Problem 3: Digital Clock Display

- Given seconds since beginning of day
- Determine AM/PM + hour + minute
- Manipulate bits so that right LCD bars are shown (shifts, bitwise-or/and, masks)
- This one is AWESOME

Review of Data Structures

Data Structures

- Vector
- Linked List
- Stack: push, pop, top
- Queue: enqueue, dequeue, front

Questions

- What are they made of?
- How would you draw them?
- How do they "work"?
- What operations do they have?

Goals

- A memory checker: `valgrind`
- A debugger: `gdb`
- A build system: `make`
- A shell: `bash`
- Beyond C

Overview

Building software requires

- Appropriate prog lang choice
- Good support tools

Analogy: write a term paper

- Good language: English (the only one I know)
- Spell checker (b/c I suck at speling)
- Citation manager (b/c reference formatting bores me but Mr PC never gets bored)
- Document/Word processor (b/c typewriters are less helpful)

Memory Problems

- In C you get to manage your own memory
- This probably already burned you

```
lila [w08-1-code]% gcc -g badmemory.c
lila [w08-1-code]% a.out
...
Segmentation fault (core dumped)
# WTF? A little help? Anyone? Anyone?
```

Getting Help

- Memory Tools on Windows
  - Discussion here. Synposis → $$$
- Memory Tools on Linux/Mac
  - Valgrind → FREE (available on zeus)

Memory Tools on Linux/Mac

Valgrind[1]: Suite of tools including Memcheck

- Catches most memory errors[2]
  - Use of uninitialized memory
  - Reading/writing memory after it has been free'd
  - Reading/writing off the end of malloc'd blocks
  - Memory leaks
- Source line of problem happened (but not cause)
- Super easy to use
- Slows execution of program *way* down

[1] http://valgrind.org/

[2] http://en.wikipedia.org/wiki/Valgrind

Valgrind in Action

See some common problems in `badmemory.c`

```
# Compile with debugging enabled: -g
lila [w08-1-code]% gcc -g badmemory.c

# run program through valgrind
lila [w08-1-code]% valgrind a.out
==12676== Memcheck, a memory error detector
==12676== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et a
==12676== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright
==12676== Command: a.out
==12676==
Uninitialized memory
==12676== Conditional jump or move depends on uninitialised value(s)
==12676==    at 0x4005C1: main (badmemory.c:7)
==12676==
==12676== Conditional jump or move depends on uninitialised value(s)
==12676==    at 0x4E7D3DC: vfprintf (in /usr/lib/libc-2.21.so)
==12676==    by 0x4E84E38: printf (in /usr/lib/libc-2.21.so)
==12676==    by 0x4005D6: main (badmemory.c:8)
...
```

Debuggers

- There comes a day when `printf` just isn't enough
- On that day you will start compiling with `-g`
- Then you will run `gdb myprog`
- `gdb` is on your Cygwin installation
- Wise ones learn the value of `M-x gdb` (emacs)

Basic Debugger Use

- Start debugger with program you want to debug
- Set *breakpoints*: places to stop
- Run program
- Execution stops at given break point
- Step forward line by line, print variables, examine what might be going wrong

Build Me a System

What’s a traditional, simple build system in Unix again?

How make and Makefile Works

Build up dependencies recursively

- A tree-like structure (actually a DAG)
- Run commands for the lowest level
- Then go up a level
- Then up another ...
- Can recurse to subdirectories to use other `Makefiles` as well
- `Makefile` describes dependencies between source/program files and commands to generate/compile

Makefile Format

```
target : dependecy1 dependency2
        do command 1
        then do command 2
```

`make` is One of Many build systems

Alternatives

- `CMake` which is more cross-platform (Windows/Mac friendly)
- `imake qmake nmake etc` which are variants
- `ant` and `maven` for java projects
- `autoconf` and `automake`, extend make, used for many free software packages
- Every IDE has one (Eclipse, Visual Studio, NetBeans, etc)
  - Most of these are "project based"

The Shell

It would be criminal not to mention that command line shells allow you automate many tasks, formally and ad hoc.

C Code

```
#include <stdio.h>
int main(int argc,
 char **argv){
  int i, n = atoi(argv[1]);
  for(i=0; i<n; i++){
    printf("%d\n",i);
  }
  return;
}
```

Bash code

```
n=10
for i in ‘seq 1 $n‘; do
    echo $i
done
```

Running programs Over and Over

Want to run the program

```
clock_sim 1 > output.1.txt
clock_sim 2 > output.2.txt
clock_sim 3 > output.3.txt
clock_sim 4 > output.4.txt
...
```

Do

```
for i in ‘seq 1 20‘; do
    clock_sim $i > output.$i.txt;
done
```

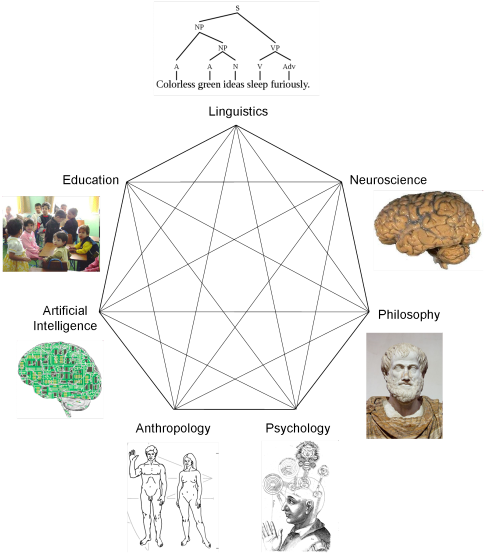
- The shell is another programming language
- Variables, loops, conditions, etc.
- Optimized for working with program runs and files

Big Problems

- Find the minimum area required for a set of logic gates
- Fly an unmanned spacecraft past the planet Pluto
- Build a machine that emits radiation in controlled doses to treat cancer
- Provide a web store which allows buyers to purchase a variety of items from many different vendors

Big problems: can't be solved by cranking out a long `main` in a few hours.

7 +/- 2 [3]



3 http://en.wikipedia.org/wiki/The_Magical_Number_Seven,_Plus_or_Minus_Two

The Thing

The notion of a "thing" is loose. As you read this sentence your cognitive system seamlessly aggregates large numbers of things into progressively larger chunks as that's what brains do.

1. Light of varying wavelengths
2. Shapes on a screen
3. Lines representing letters
4. Collection of letters as words
5. Words strung together
6. Sentences conveying an idea
7. An idea that *may* be on the final exam
    - You can hold about 7 things in short-term memory

Focus

1. Light of varying wavelengths
2. Shapes on a screen
3. Lines representing letters
4. Collection of letters as words
5. Words strung together
6. Sentences conveying an idea
7. An idea that will be on the exam

At any point you can focus your attention at a particular level in this sequence and devote your 7 memory slots to them specifically.

- What colors are on the screen?
- What is the shape of the first letter in bullet 1?
- How do you spell the first word in bullet 4?
- How many words are there in the bullet 5?
- Can you do all this at once?

People Abstract

We automatically abstract things. The abstraction is a single "thing" to keep in memory.

- A word
- A queue
- A language
- A sentence
- A whole program

Programmers shift between abstraction levels

Problem Solving and Abstraction

As you write larger programs, will need to shift between

- Write a function to perform a high-level task
- Write some pseudocode listing the general steps
- Refine the steps to loops, conditionals, comparison, and operations on data structures
- Map simple steps directly into C code and complex steps into function calls
- Implement functions for "complex" steps (empty, add, remove, top, push, etc.)
- Monkey with pointers to get those functions right

Three List Nodes

```
typedef struct int_node_s {
  int data;
  struct int_node_s *next;
} int_node;

typedef struct cust_node{
  int id;
  customer_status status;
  int visit_time;
  int entered_queue_time;
  int arrival;
  struct cust_node *next_cust;
} customer;

typedef struct card_node_s {
  card the_card;
  struct card_node_s *cnp;
} card_node;
```

Similar structure, cognitively different.

- `int_node`'s data is `data`, next pointer is `next`
- `customer` has 5 data fields, next of `next_cust`
- `card_node` has single data field, and next of `cnp`

Library functions for `int_list` will only work with one of them.

Higher Patterns are Hard In C

A small pattern: lists should look the same

- Should treat them the same
- But will need to write a `prepend()`, `insert()`, `remove()`, etc. for each of them
- C doesn't let you do that
  - *Unless* you use `void *` and remember types yourself

```
typedef struct list{
  void *data;
  struct list *next;
} list;
```

Lesson

- We see the pattern but C can't code it
- `int_node`, `cust_node`, `card_node` will occupy 3 slots of <u>your</u> brain memory as they are distinct in C

Patterns

The pattern has a significant human component (minimize human intervention). All software serves human comfort or quality of life; the best patterns explicitly appeal to aesthetics and utility.
–James O. Coplien

Limits

- C has limited abstraction facilities
  - Functions and structs
- C++ has more abstraction facilities (as does Java)
  - Templates/Generics allow parametrized code
  - Classes organize related data and functions
  - Classes can be related

Templates

- Create a pattern that applies to any kind of class.
- Instantiate concrete patterns that work regardless of type
- Overcomes the list problems we had
- See `lists.cpp`

```
list<int> myints;
list<string> mycards;
list<Cart> mycards;
queue<Cart> q1, q2;
stack<Item> s1, s2;
```

Classes

- Group related data and functions
  - push/pop/top with stack
  - add/remove/front with queue
  - empty with both
- Create *objects* which are concrete instances of classes
- Invoke functions associated with those objects
  - *Methods* or
  - *Instance Functions*
  - Jargon for function that works on object

```
queue<string> q;
q.add("Something");
q.add("Something else");
string s = q.front();
q.remove();
bool b = q.empty();

stack<string> s;
s.push("Something");
s.push("Something else");
string t = s.top();
s.pop();
bool b = s.empty();
```

Session Thesis

- Object-oriented design is one way to create abstractions.
  - Model data with classes
  - Classes group data and associated functions
  - Classes can relate to one another
  - Templates/Generics allow more general patterns
- Knowing when OO is appropriate takes practice
- Will you know when it's appropriate if you don't know alternatives?

Opinions

Object-oriented programming is an exceptionally bad idea which could only have originated in California.
– Edsger Dijkstra

C++ is the dumbest language on earth.
. . .
The original brilliant guys and gals here only allowed two languages in Amazon's hallowed source repository: C and Lisp.
– Steve Yegge, Tour de Babel

As someone remarked: There are only two kinds of programming languages: those people always bitch about and those nobody uses.
– Bjarne Stroustrup, Inventor of C++

You learned a popular language: `http://www.langpop.com`

The Ones No One Uses

[Fortran] was in strong contrast to LISP whose purpose was to enable the execution of processes that no one would dream of performing with pen and paper.
– Edsger Dijkstra [4]

You should also know Lisp.

. . .

It's hard, though. It's not sufficient to learn how to write C-like programs in Lisp. That's pointless. C and Lisp stand at opposite ends of the spectrum; they're each great at what the other one sucks at.
– Steve Yegge, Tour de Babel

[4] Keynote address given on 1 March 1999 at the ACM Symposium on Applied Computing at San Antonio, TX

BREAKTIME

Back in 10 minutes

Final Exam Logistics

- Thursday 7/23 4:30-7:10
- 7-8 pages
- Style: Like previous exams
- Content: Comprehensive
- Up to and including today (tools/C++ high level view)

Major Stuff

Topics

- Control Flow
- File I/O
- Creating Aggregate Data
- Pointers and Addresses
- Manual Memory Management
- Handling command line args
- Bit operations
- Fundamental CS data types not in C
  - List, Stack, Queue
- List functions with pointers

Skills

- Basic Compile Errors
- A few C tools
- Evaluating Code
- Knowing Scope of Variables
- Familiarity with HW Codes
- Familiarity with Some of the standard library
  - Memory Allocation
  - File I/O
  - Some string funcs

Review Problems

Will post these and more on the web after class

Evaluations

Extra questions

```
5 4 3 2 1
Yes     No
```

24
Zyante provided enough information and practice for me to do the HWs.

25
My overall experience with Zyante's was

```
5 high
1 low
```

26
I got extra help from people aside from Prof. Kauffman or classmates