

CS 222: Intro

Chris Kauffman

Week 1-1

Goals

- ▶ Motivate study of C, esp. for engineers
- ▶ Give an idea of C as a language
- ▶ Overview of course mechanics

Who I expect You are

EE and ECE Majors

Required down the line for

- ▶ ECE 445 - Computer Organization
- ▶ ECE 447 - Single-Chip Microcomputers
- ▶ ECE 465 - Computer Networking Protocols

BENG Majors

- ▶ Concentration in Biomedical Signals and Systems
- ▶ Concentration in Bioengineering Healthcare Informatics
- ▶ Concentration in Bioengineering Pre-health

CS Majors who are confused, crazy, or desperate

Am I right?

Motivation

You just have to know C. Why? Because for all practical purposes, every computer in the world you'll ever use is a von Neumann machine, and C is a lightweight, expressive syntax for the von Neumann machine's capabilities.

–Steve Yegge, *Tour de Babel*

Von Neumann Machine

Traditionally 3 parts

Processing arithmetic and logic

- ▶ Wires/gates that accomplish fundamental ops
- ▶ +, -, *, AND, OR, etc.
- ▶ Modern CPUs use **registers**: special fast memory

Control where and what to do next

- ▶ Special register indicating next instruction
- ▶ Typically memory address of instruction

Memory for remembering stuff

- ▶ Giant array of bits/bytes
- ▶ Accessible **by number**

Input/Output for the humans

- ▶ Not in the original, but pretty essential
- ▶ Special memory locations

Insights to Von Neumann

Really no difference between "programs" and "data" on the computer

- ▶ They're all electrons at some point
- ▶ Change data, change program
- ▶ Will be important to understand memory allocation, dreaded pointers

Low Languages

- ▶ Binary Opcodes
 - ▶ 1's and 0's, represent the digital signal of the machine
 - ▶ Each code corresponds to an instruction, possibly also a memory location
- ▶ Assembly
 - ▶ Tiny bit more readable than binary
 - ▶ Directly translated to binary using a program
 - ▶ Specific to each CPU - close to the machine

An Example

	HEXADECIMAL	ASSEMBLY
0:	55 = 0101 0101	push %ebp
1:	89 e5	mov %esp,%ebp
3:	83 e4 f0	and \$0xffffffff0,%esp
6:	83 ec 10	sub \$0x10,%esp
9:	c7 04 24 00 00 00 00	movl \$0x0,(%esp)
10:	e8 fc ff ff ff	call 11 <main+0x11>
15:	c9 = 1100 1001	leave
16:	c3 = 1100 0011	ret

Looks like fun, right? (CS 262, ECE 445)

Another Example: MIPS

MIPS from CS 367 (?)

500:	add	s1, s2, s3	1000:	6
501:	sub	s1, s2, s3	1001:	0
502:	lw	s1, s2	1002:	0
503:	add	s3, s1, s2	1003:	2
504:	sw	s1, s3	1004:	0
505:	add	s2, s1, s2	1005:	0
506:	j	508	1006:	10
507:	bne	s1, s2, 502	1007:	10
508:	beq	s2, s3, 507	1008:	0

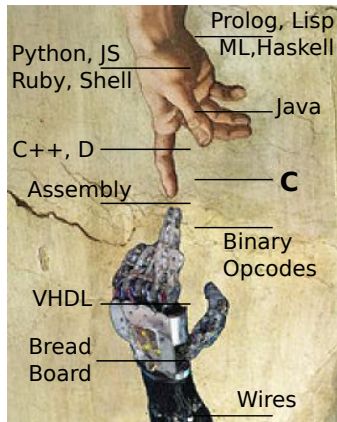
...

Initial state

- ▶ s1 = 0
- ▶ s2 = 1000
- ▶ s3 = 4
- ▶ Control = 500

Why C is around

Pure Abstraction



- ▶ Save you from assembly
- ▶ Abstract from a specific CPU so programs can be portable
- ▶ Provide programmer convenience
 - ▶ More Readable
 - ▶ Software design easier
- ▶ Stay close to the machine

Bare Metal

Img Source: <http://bpmredux.wordpress.com/>

Thoughts

In C, more than any other language I've ever used, it pays to tie your thinking to the machine, not the abstractions.

In C you can

- ▶ Get the memory address of any variable or function
- ▶ Add, store, and mangle those addresses
- ▶ Get memory by entering a function
- ▶ Release memory by finishing a function
- ▶ Explicitly request (`malloc`) and clean up (`free`) memory
- ▶ Ask for size of types in memory units (`sizeof`) during allocation
- ▶ Ignore types of variable (integer, real, character, etc)
- ▶ Use arrays and addresses interchangeably

You must do many of the above to get work done in C. Knowledge of how the underlying target machine works, at least generally, is essential.

What I'd Use C For

I wouldn't use C unless I am . . .

- ▶ Forced to by my boss (most frequent reason)
- ▶ Coding a computationally intensive part
- ▶ In need of *very* fine control of memory
- ▶ In need of lots of bit operations
- ▶ Running my app *everywhere*
 - ▶ From main frame to smart watch
- ▶ Initially creating another PL (Python, Matlab, Java)
- ▶ Writing an operating system
- ▶ Am feeling *very* masochistic

Another's Thoughts

Both early Unix and C compilers had simple structures, are easy to port, require few machine resources to run, and provide about 50%–80% of what you want from an operating system and programming language.

–Richard Gabriel, [The Rise of “Worse is Better”](#)

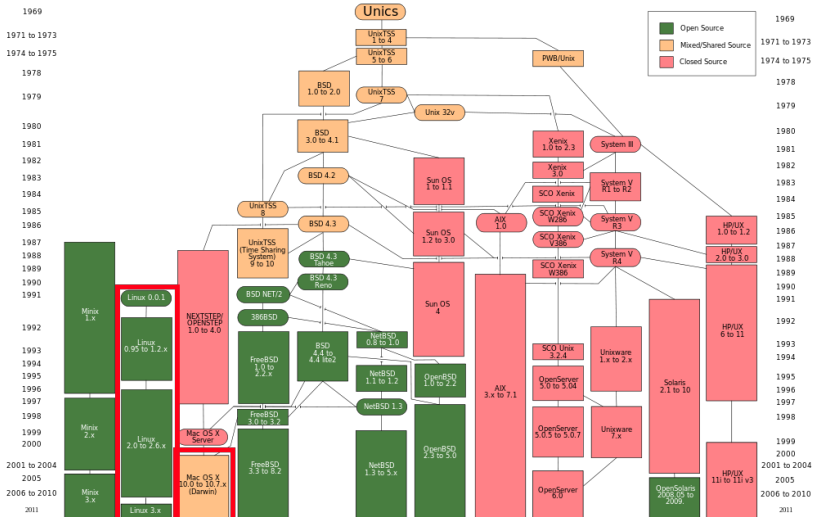
Another Motivation

You also have to know C because it's the language that Unix is written in, and happens also to be the language that Windows and virtually all other operating systems are written in, because they're OSes for von Neumann machines, so what else would you use? Anything significantly different from C is going to be too far removed from the actual capabilities of the hardware to perform well enough, at least for an OS - at least in the last century, which is when they were all written.

-Steve Yegge, [Tour de Babel](#)

An Operating System is...

The Many Unices



Unix

Unix means to me

- ▶ Certain set of interfaces to the machine
- ▶ Access many parts of the machine via C language and libraries
- ▶ Access many parts of machine via a **command shell**

I assume

Many of you will have to do a little bit-wrangling in your time

- ▶ Low-level manipulation of memory
- ▶ Interface a device of some type

C on Unix commonly used

Preconditions

You should have programming experience.

- ▶ CS 112 or equivalent
- ▶ Vaguely remember what things like source code, functions, loops, maybe arrays/lists might be

Also you must be able to

- ▶ Read and write (English)
- ▶ Do arithmetic
- ▶ Browse the web
- ▶ Install programs
- ▶ Think, slowly and patiently

Outcome Expectations

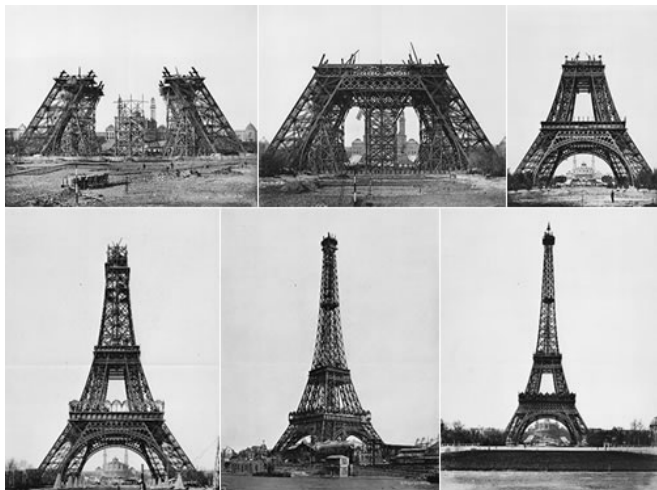
After this class



After some years in the wild



Real Software Projects. . .



Take lots of cooperation, like any other engineering effort

Daily Course Mechanics

Meet Tuesday/Thursday Evening

4:30-5:45 Lecture/discussion part 1

5:45-6:00 Break (eat something, please!)

6:00-7:10 Lecture/discussion part 2

Syllabus

Tons of info on

- ▶ Grading
- ▶ Assignment submission
- ▶ Policies (late work, etc.)
- ▶ Schedule

Programming Assignments

- ▶ 7 of them during the semester
 - ▶ Must do all, no drops
- ▶ 42% of your grade
- ▶ 2-3 Problems covering latest material
- ▶ Submit to blackboard, 11:59 p.m. Tuesdays
 - ▶ Intention: work over the weekend
 - ▶ Bring questions to Tuesday class
 - ▶ Exams are Thursdays
 - ▶ Objections now

Exams

2 midterm exams during semester

- ▶ 1 hour 15 min lecture
- ▶ 15 minute review
- ▶ 15 minute break
- ▶ Take exam \approx 1 hour

1 final exam

- ▶ Last session (a Tuesday)
- ▶ Comprehensive
- ▶ Must get 50% or better to pass the class

All exams **open resource**

Cheating

Don't cheat

- ▶ Easy to catch
- ▶ Pain for you, Pain for me (makes me ornery)
- ▶ If you don't get caught, you'll still suck at programming
- ▶ What is cheating in **this** programming class?

Participation

- ▶ Each session, front 2 rows are hot seats
- ▶ Chance to earn participation extra credit
- ▶ Just try: answer questions, give feedback
- ▶ Up to 3% overall bonus
- ▶ Don't want/need participation, sit elsewhere

Slides and Demo Code

- ▶ Will try to make slides available before class
- ▶ Slides always available sometime after class
- ▶ Slides are not much good without accompanying conversation
- ▶ Demo code is **often relevant to HW** problems

Textbook

Zyante's Programming C

- ▶ Available here: <https://zybooks.zyante.com/#/zybook/GMUCS222Summer2015/>
- ▶ Cost around \$48
- ▶ **Required** exercises: worth 3% of grade
- ▶ Registration code: GMUCS222Summer2015

Your Teaching Team

Chris Kauffman Instructor

- ▶ Mail: kauffman@cs.gmu.edu
- ▶ Office: Engineering 5341
- ▶ Hours: Plan 3-4pm Tue/Thu
- ▶ Other times preferable?

Phe Hung Le Grader

- ▶ **NOT** a teaching assistant
- ▶ Grading questions only, no general questions
- ▶ ple13@masonlive.gmu.edu

BREAKTIME

Back in 15 minutes

Hot seats: Come up front if you want some love

Goals

- ▶ Setting up your environment
- ▶ A few tools
- ▶ Overview of some elements in C
- ▶ May not finish today

Ahead of Time Translation \equiv Compilation

Machine doesn't speak C

- ▶ What language does it speak?

Need a go-between

Setup Goal 1: Get a Compiler

GNU Compiler Collection: gcc

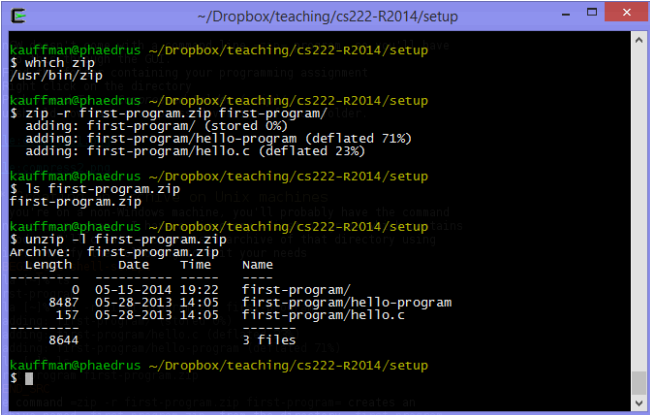
- ▶ Includes C compiler
- ▶ Free
- ▶ Widely used

Other options?

http://en.wikipedia.org/wiki/C_compiler#C_compilers

- ▶ Not used in our class
- ▶ Who would write a compiler?

Setup Goal 2: Get a Terminal



```
~/Dropbox/teaching/cs222-R2014/setup
kauffman@phaedrus ~/Dropbox/teaching/cs222-R2014/setup
$ which zip
/usr/bin/zip

kauffman@phaedrus ~/Dropbox/teaching/cs222-R2014/setup
$ zip -r first-program.zip first-program/
adding: first-program/ (stored 0%)
adding: first-program/hello-program (deflated 71%)
adding: first-program/hello.c (deflated 23%)

kauffman@phaedrus ~/Dropbox/teaching/cs222-R2014/setup
$ ls first-program.zip
first-program.zip

kauffman@phaedrus ~/Dropbox/teaching/cs222-R2014/setup
$ unzip -l first-program.zip
Archive: first-program.zip
Length      Date       Time      Name
-----
0           05-15-2014 19:22    first-program/
8487       05-28-2013 14:05    first-program/hello-program
157        05-28-2013 14:05    first-program/hello.c
-----
8644                               3 files
```

- ▶ A lovely, uncomplicated way to talk to your machine
- ▶ Type a command, watch what happens
- ▶ REPL (sort of)

Getting A Unix Environment

See Homepage->Course Content->Setting up your Environment

Windows Set up with **Cygwin** (unixy environment)

Mac OS X Terminal.app and getting gcc

Linux I'll assume you know what you're doing

Alternatives

Mingw partial unix for windows

Virtualbox Run a full OS in a single window

SSH Log into school machines, work only on the terminal

Windows cmd If you are a masocist.. still need a compiler

In Class Terminal

I'll use a virtual machine

- ▶ Windows on the laptop
- ▶ Linux in the VM
- ▶ Terminals in linux

Sometimes use Cygwin terminals

A Profound Decision: Text Editor

Programs are text files

- ▶ In our case with `.c` at the end

We'll be editing text files

- ▶ You need to pick a text editor

Setup Goal 3: Get a Text Editor

Many of them, list a few good ones in *Setting up your Environment*

jEdit Available on all platforms, built by programmers for programmers, intuitive, colorful, reasonable power, and free.

NotePad++ Windows only, free software, many students swear by it.

TextPad Windows only, free for eval, pay to keep, solid editor for programming and what I used when forced to use only Windows tools

TextMate Mac only, nonfree, award-winning.

Tools that Grow

Jedit



Emacs



Tools for the Art

Jedit



Emacs



In Class Editing

I'll use emacs, sometimes cygwin

- ▶ I'm an emacs person
- ▶ Use what makes you productive

Environment Setup

[Set up your environment](#) - click link to see instructions

Once finished, should *at least* be able to compile and run

```
lila [example-code]% gcc hello.c
lila [example-code]% ./a.out
Hello World
lila [example-code]%
```

On Windows

```
lila [example-code]% gcc hello.c
lila [example-code]% ./a.exe
Hello World
lila [example-code]%
```

Hello World

hello.c is a file containing the following program

```
#include <stdio.h>
int main() {
    printf("Hello World\n");
}
```

In w01-1-code.zip

A note on IDEs:

From `setup.html`:

*We'll be focusing on learning the steps to produce programs: editing, compiling, debugging, running on the terminal. Integrated Development Environments (IDEs) like Microsoft Visual Studio, Apple's XCode, and Eclipse will be a major hindrance to this effort as they hide many of those details. Those tools have their place for professional software engineers who can dive into the details if the need arises, but since we're learning this stuff the first time, we need to be concerned with such details. To that end, **no help will be given diagnosing problems in an IDE**. Stick to a text editor and the gcc command line interface.*

Tour of C

- ▶ Look at first textbook program
- ▶ See some features of C
- ▶ Revisit them in more detail over next few weeks

Every Programming Language

Look for the following

- ▶ Comments
- ▶ Statements/Expressions
- ▶ Variable Types
- ▶ Assignment
- ▶ Basic Input/Output
- ▶ Function Declarations
- ▶ Conditionals (if-else)
- ▶ Iteration (loops)
- ▶ Aggregate data (arrays, structs, objects, etc)
- ▶ Library System

First Textbook Program: salary.c

```
/* From Zyante Programming C Ch 2.15 w/ modifications
   Calculate age in days based on input, assumes 365 day years

   compile:      gcc age.c
   run on mac:   a.out
   run on win:   a.exe
*/

#include <stdio.h>

int main(void) {
    int userAgeYears = 0;
    printf("Enter your age in years: \n");
    scanf("%d", &userAgeYears);

    // Declare anywhere
    int userAgeDays = userAgeYears * 365;
    printf("You are %d days old.\n", userAgeDays);
    return 0;
}
```


Comments

- ▶ Removed by compiler
- ▶ There to help you remember what you were thinking last year
- ▶ There to help the poor bastard who has to work on your terrible code when you move on to greener pastures
- ▶ Two allowed syntaxes

```
/* Comment */  
// Comment
```

```
/* Traditional C comment which can span multiple lines, and  
be as verbose as you like. Everything between the  
symbols is ignored */
```

```
// A newer comment syntax introduced by C++, only comments  
// to the end of the line so make sure that the symbols  
// appear at the beginning of each line
```

Commenting/Uncommenting

All text editors worth their salt can

- ▶ Comment a region of program
- ▶ Uncomment a region of a program

This is **incredibly useful** during program development.

- ▶ Try something and find it doesn't work
- ▶ Comment it out and try something else
- ▶ Still have the first way available

Learn how your editor does it

JEdit Install TextTools Plugin, Toggle Range Comment

Emacs `comment-region` `uncomment-region`

Variable Types

Integer like 0, 1, 15, -32, 32456

- ▶ **Important:** int as in `int x = 32;`
- ▶ Less : long, short, long int, long long

Real like 0.0, 0.5, -42.378, 5.34e-13

- ▶ **Important:** double as in `double x = 1.234;`
- ▶ Less often but sometimes: float

Character like c, Hello World, Wouldn't\nIt\nBe\nNice?

- ▶ **Important:** char as in `char x = 'c';`
- ▶ **Important:** char * as in
 - ▶ `char *x = "Hello World";`
- ▶ **Important:** char x[] as in
 - ▶ `char x[] = "Hello World";`
- ▶ Observation: *Character data sucks in C*

Boolean like true and false

- ▶ ???

Nothing huh?

- ▶ **Important:** void

Declare then Use

Must *declare* variables before using **and** give them a *type*

Right

```
int main(){
    int x = 4;
}
```

```
int main(){
    int x;
    x = 4;
}
```

Wrong

```
int main(){
    x = 4;
}
```

```
int main(){
    x = 4;
    int x;
}
```

Historical Note

Old C

```
int main(){
    int x, y;
    double d;

    x = 4;
    y = x + 2;
    d = 12.34;
}
```

New C (C99/C11)

```
int main(){
    int x;
    x = 4;
    int y = x + 2;

    double d;
    d = 12.34;
}
```

A Lesson

All languages change

- ▶ New words enter English (e.g. *truthiness*, *selfie*)

New ideas enter PLs

- ▶ C is changing
- ▶ Very slowly compared to other PLs

Gotchya: not every compiler translates C to machine language the same way

- ▶ May not support the latest lingo (C11)
- ▶ Use our environment so that you are compatible

Statements/Expressions - Do Something

Assignment is very common use 'equals sign'

```
x = 5;
```

End with a semicolon: ;

- ▶ Most frequent error is forgetting ;

Arithmetic

```
int x, int y = 5;
```

```
x = y * 2 + 1;
```

```
x = (y * 2) + 1;
```

```
x = y * (2 - 1);
```

```
x = x * x + y - 1;
```

```
x = y / 2;
```

```
x = y % 2;
```

```
/* ??? */
```

Real Arithmetic

```
double x, double y = 5.0;
x = y * 2 + 1;
x = (y * 2) + 1;
x = y * (2 - 1);
x = x * x + y - 1;
x = y / 2;
x = y % 2; // !!!
```


Input/Output

Beginning C

Terminal printf and scanf

Files (later) fprintf and fscanf with fopen and fclose

Later Binary I/O with fwrite and fread

printf

Simple String messages

```
printf("Hello world\n");  
printf("Line 1\nLine 2\nLine 3\n");
```

Formatted Output

Substitute variable values into format string at certain locations

%d	integer	%lf	double
%c	character	%s	string

```
printf("An integer %d\n",123);  
printf("A real %f\n",    0.456);  
printf("A string %s\n",  "sweet");  
// Multiple outputs in single statement  
printf("An integer %d    A real %f    A string %s    \n",  
       123,           0.456,           "sweet");
```

scanf

- ▶ For input, especially from terminal
- ▶ Format string specifies kind of input

```
/* Demonstrate some scanf functions, relevant for HW1 */
#include <stdio.h>
int main(){
    printf("Input an integer and a real\n");
    int myint;
    scanf("%d", &myint);          /* & ??? */

    double mydoub;
    scanf("%lf", &mydoub);       /* %lf ??? */

    printf("i: %d    d: %lf\n", myint, mydoub);

    printf("Again!\n");
    scanf("%d %lf", &myint, &mydoub);
    printf("i: %d    d: %lf\n", myint, mydoub);
}
```

Compilation and Preprocessing

gcc performs a bunch of steps

- ▶ Parse, syntax check, optimize, generate assembly, assemble, link...
- ▶ One step is especially tied to C: preprocessing

Preprocessor

- ▶ A partner language to C
- ▶ Change program text before compilation
- ▶ Add files, Substitute text
- ▶ Use directives: #include and #define mostly
- ▶ Makes early changes to the program (pre in preprocessor)

Before and After

Before

```
#include <stdio.h>
#define SOME_NUMBER 42
#define SOME_STRING "Good Stuff"
#define SOME_CODE (x = 2*x)

int main(){
    printf("string: %s\n",
        SOME_STRING);
    int x = 1 + SOME_NUMBER;
    SOME_CODE;
    printf("number: %d\n",x);
}
```

After

```
... stuff from stdio.h ...
...
...
int main(){
    printf("string: %s\n",
        "Good Stuff");
    int x = 1 + 42;
    (x = 2*x);
    printf("number: %d\n",x);
}
```

Typical Preprocessor Use

- ▶ Constant declaration
 - ▶ Convention: CONSTANT_IN_ALLCAPS
 - ▶ `#define PI 3.14159`
 - ▶ `#define KMS_PER_MILE 1.609`
- ▶ Including other files
 - ▶ Headers (`xxx.h`)
 - ▶ `#include <stdio.h>` - bring in `printf`

Notice: no semicolons for preprocessor statements

In First Program

- ▶ Comments
- ▶ Statements/Expressions
- ▶ Variable Types
- ▶ Assignment
- ▶ Basic Input/Output
- ▶ Function Declarations
- ▶ Conditionals (if-else)
- ▶ Iteration (loops)
- ▶ Aggregate data (arrays, structs, objects, etc)
- ▶ Library System

Wrap-up

Hot Seats sign paper with name and card-count

For next meeting

- ▶ Zyante reading, Ch 1-2 (see [schedule](#))
- ▶ Setup your environment (see [setup](#))
- ▶ HW 1 up and due next week

Let's Solve a Problem (or at least start)

- ▶ Each GTA is assigned some lab sections for CS211
- ▶ Each lab section has a day/time and a student count

Your program (pseudocode / python/ whatever)

1. Read the data file provided
2. Compute the Head Count for each GTA
3. Print the Discrepancy: difference between biggest and smallest head count

Input Data File

#	Day	Time	Room	Count	GTA
201	R	12:30	4457	19	Raj
202	R	01:30	4457	15	Raj
203	R	02:30	5358	6	Adam
204	R	03:30	4457	21	Raj
205	R	11:30	5358	11	Ruoxi

Head Counts

Raj	55
Adam	6
Li	11
Discrepancy:	49

Full Data

Input

201	R	12:30	4457	19	Raj
202	R	01:30	4457	15	Raj
203	R	02:30	5358	6	Adam
204	R	03:30	4457	21	Raj
205	R	11:30	5358	11	Li
206	R	12:30	5358	22	Li
207	R	01:30	5358	15	Li
208	R	02:30	5358	12	Adam
2H1	R	03:30	5358	17	Adam
210	R	01:30	1505	25	Adam
212	R	02:30	1505	24	Le
213	R	03:30	1505	24	Le
214	F	08:30	4457	3	Kacem
215	F	09:30	4457	10	Kacem
216	F	10:30	4457	12	Kacem
217	F	11:30	4457	24	Kacem

Head Counts

Raj	55
Adam	60
Li	48
Le	48
Kacem	39

Discrepancy: 24