http://bit.ly/gwsofteng16

CS 2113
Software Engineering

Lecture 1:
Overview and Introduction to C

The George Washington University

Welcome

CS 2113 Software Engineering 1

- Prereq: CS 1112 Algorithms & Data Structures

Professor Tim Wood

- timwood@gwu.edu
- Office Hours: Tues 11-12:30
  - SEH 4580

TA: Bo Mei

- bomei@gwu.edu
- Office Hours: Mondays 10:10-11AM and 1:30-2:10PM

LA: Victoria and Josh

- In class helpers
- Won’t tell you the solutions

Lectures: TOMP201
Wednesday 3:30-6PM

Labs: TOMP405
Monday 9:00-10:10
or
Monday 2:10-3:20

Course Website

http://bit.ly/gwsofteng16

Course Outline

Weeks 1-3: Introductory C programming

- Syntax, memory management, libraries, file IO

Weeks 4-8: Intermediate Java programming

- Quick review, objects, class hierarchies

Weeks 9-14: Advanced Java Topics

- GUIs, concurrency/threading, IO, networking, web

Throughout: Software engineering techniques

- Requirements, Architecture, Design Principles

Course Overview

Course Goals:

- Learn the basics of C programming
- Understand the memory model used in Java and C
- Deduce software requirements from a problem description
- Design complex software architectures
- Get excited about more advanced programming topics

Workload:

- Weekly(ish) small programming exercises	(30%)
- 3 programming projects	(30%)
- Quizzes and labs (come to class!)	(15%)
- Participation (in class and online)	(5%)
- Midterm and Final	(20%)

- *Grade weights subject to slight changes

Course Policies

READ the syllabus!

Late work

- You get **two late passes** - each lets you delay a deadline by **48 hours**. Save them for a rainy day / computer failure!
- No late submissions for **weekly exercises** without a pass
- Programming projects: lose 5% per 8 hours late

Academic Integrity

- Your code and solutions must be your own!
- You **may** meet with other students to discuss your ideas
  - But you **may NOT** share or copy any code
- If you use a tutor, ask the tutor to email me
- Penalties for violating the code include failing this course!
- See syllabus website for more details, or ask me

If you have a disability that may affect you in this course, let me know

Piazza

We will use Piazza for a course Q&A forum

- Great way to get participation points!

Allows you to both ask and answer questions

- Answering questions can help your participation grade!
- Use common sense: answer general questions, but don't post solutions to homeworks

If you have general questions, post to Piazza first

If you have specific questions relating to your code, ask me or the TA

- Office hours if possible, email otherwise

Computer Policy

There is a computer in front of you.

Computers are distracting.

Blue slide = use computer to do an exercise

White slide = **do not use computer for anything**

I reserve the right to revoke computer privileges

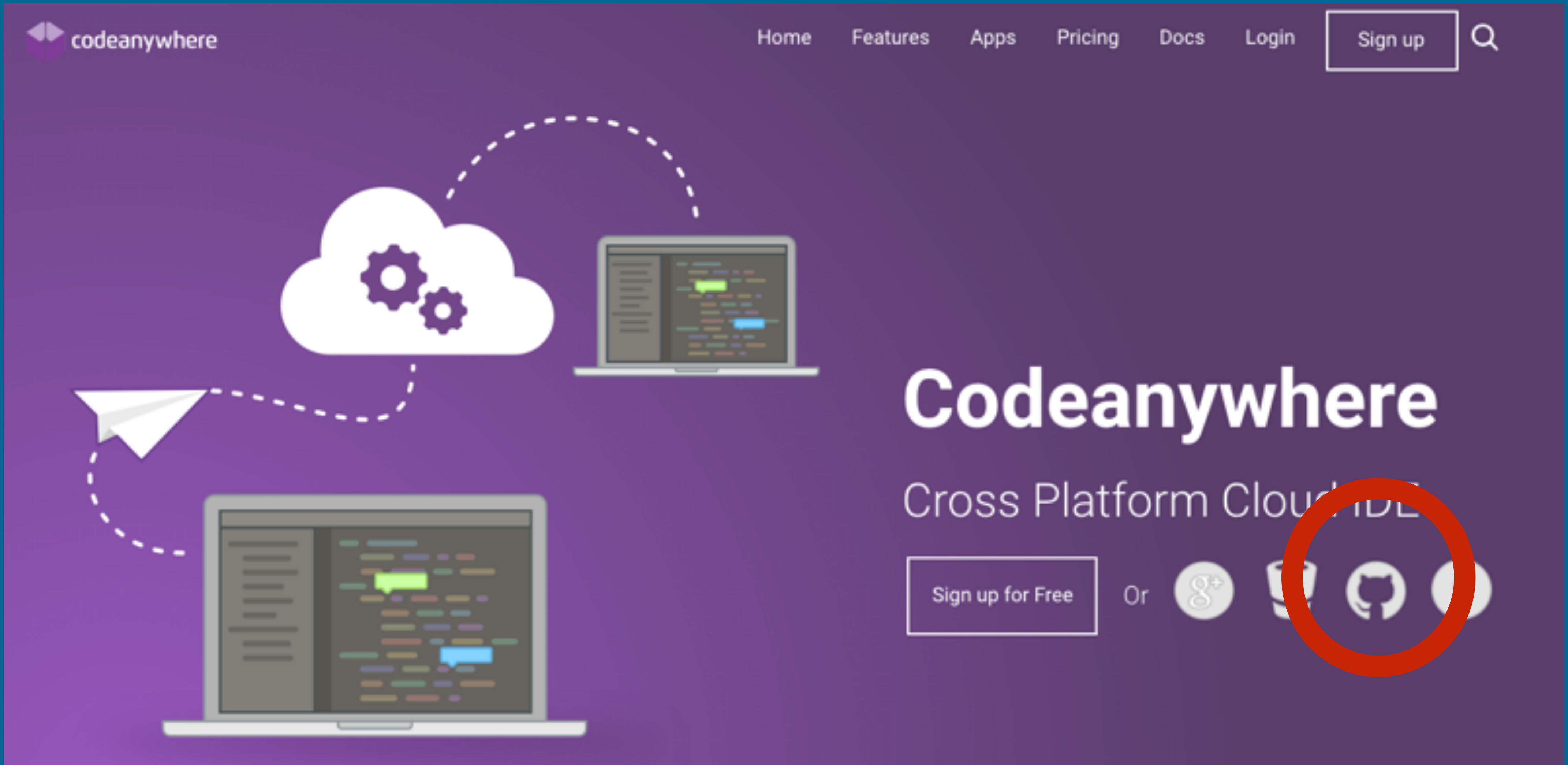
- as if you were a small child.

Development on CA

- We will use the C web-based dev tools

http://codeanywhere.com

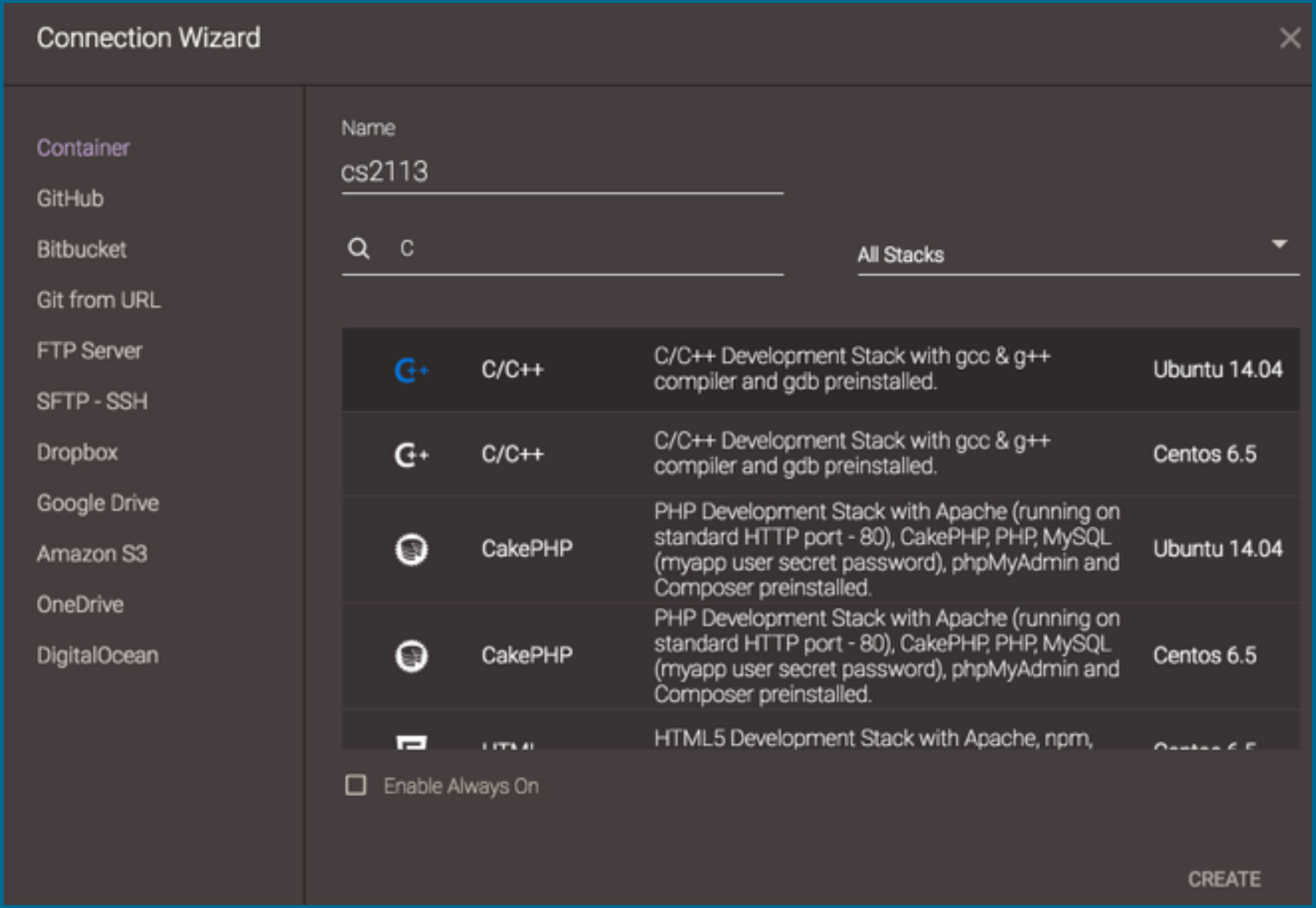
- Web based text editor and command line
  - Mainly designed for web app development (RoR, PHP, etc)



Login with git! (octocat)

CA Setup

- Signup for a free account
- Create a New Workspace
  - Name: CS2113
  - Choose C/C++ Ubuntu 14.04
- Wait...



Name: cs2113

Search: c

All Stacks

C/C++	C/C++ Development Stack with gcc & g++ compiler and gdb preinstalled.	Ubuntu 14.04
C/C++	C/C++ Development Stack with gcc & g++ compiler and gdb preinstalled.	Centos 6.5
CakePHP	PHP Development Stack with Apache (running on standard HTTP port - 80), CakePHP, PHP, MySQL (myapp user secret password), phpMyAdmin and Composer preinstalled.	Ubuntu 14.04
CakePHP	PHP Development Stack with Apache (running on standard HTTP port - 80), CakePHP, PHP, MySQL (myapp user secret password), phpMyAdmin and Composer preinstalled.	Centos 6.5
	HTML5 Development Stack with Apache, npm,	

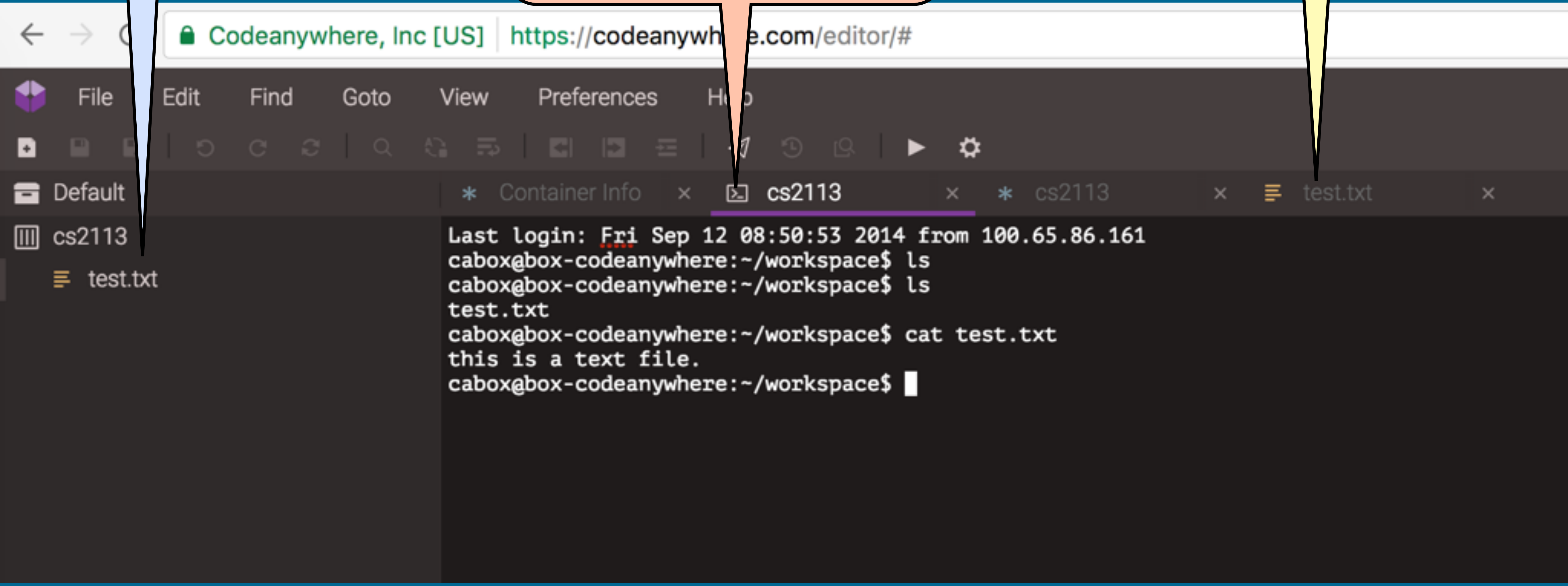
☐ Enable Always On

CodeAnywhere IDE

File List (May need to refresh)

Command Line Terminal

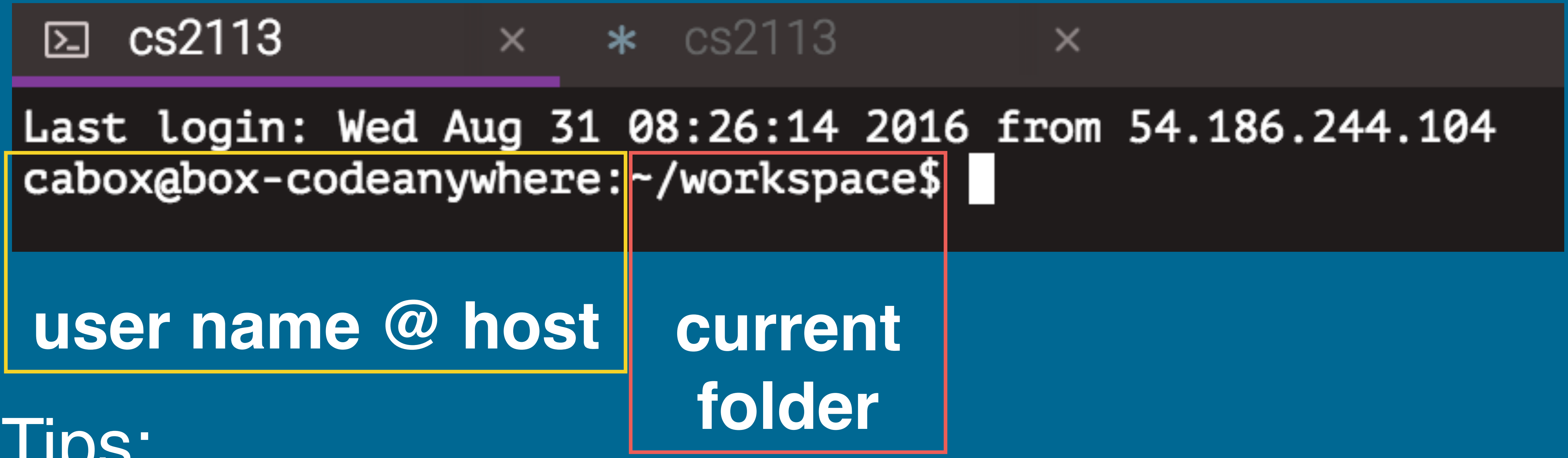
Open editing window



```
Last login: Fri Sep 12 08:50:53 2014 from 100.65.86.161
cabox@box-codeanywhere:~/workspace$ ls
cabox@box-codeanywhere:~/workspace$ ls
test.txt
cabox@box-codeanywhere:~/workspace$ cat test.txt
this is a text file.
cabox@box-codeanywhere:~/workspace$
```

Command Line

- An **SSH Terminal** gives you a **Command Line** for interacting with a remote Linux computer
  - Very intimidating at first... you'll get used to it
  - You **WILL** use this when you get a job (even if you work at Microsoft)



```
Last login: Wed Aug 31 08:26:14 2016 from 54.186.244.104
cabox@box-codeanywhere:~/workspace$ █
```

- Tips:
  - **ctrl-c** will "cancel" / quit most apps / clear your current command
  - the **TAB** key will autocomplete commands/directories

Basic Unix

- You log in to your "~/workspace/" directory
  - Try these commands

```
$ pwd
    // prints current directory path
$ mkdir lec-1
    // makes a directory named lec-1
$ cd lec-1
    // change directory into lec-1
$ pwd
    // ???
$ cd ..
$ pwd
    // ???
$ ls
    // ???
$ ls ..
$ ls ../..
$ ls /
    // ???
```

Basic Unix Commands

`mkdir d`	create directory "d"
`pwd`	print current location
`ls`	list directory contents
`ls -l`	detailed listing
`cd d`	change directory to "d"
`cp a b`	copy file "a" to "b"
`mv a b`	move file "a" to "b"
`rm a`	delete file "a"
`..`	parent directory
`.`	current directory

More tips

Use **up/down arrows** to cycle through past commands

Use **tab** key to auto-complete directory/file names

- Very convenient! Use this a lot!

To copy a full directory:

- `cp -r dir1 dir2`

Wildcard file/directory matching with `*`

- `cp lec-*.pdf docs/`
- `rm *.o`
- **Be careful! There is no undo**

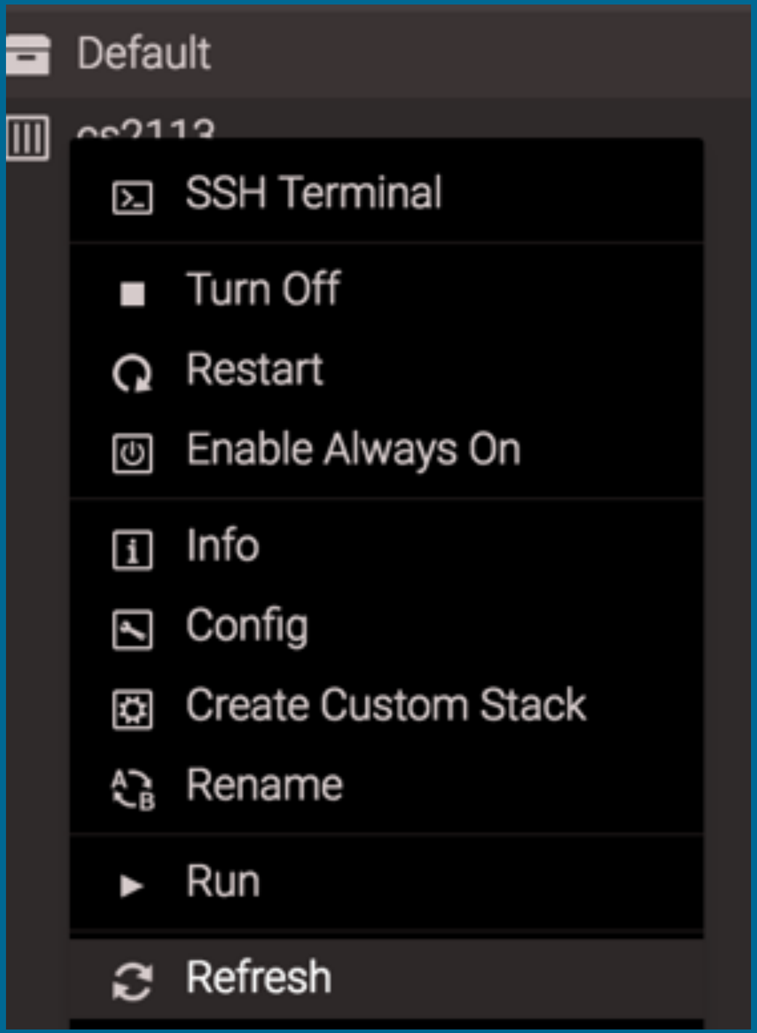
Basic Git

- To **clone** a repository (i.e., get a copy of its files for your local use)

```
git clone https://github.com/cs2113f16/lec-1.git
cd lec-1
```

- Then, refresh your web file view by right clicking on CS2113

- The lec-1/ folder should appear in the file list so you can edit files!



Basic Git

- To **clone** a repository (i.e., get a copy of its files for your local use)

```
git clone https://github.com/cs2113f16/lec-1.git
cd lec-1
```

- Later you will need:

```
git add FILE1 FILE2
git commit -m "Description of work done"
git push origin master
```

on to the real stuff!

(unless you have any questions)

The B.C. Era (Before C)

It was the 1960s:

- Sex, drugs, rock and roll...
- and operating systems!

Multics OS

- One of the first "time shared" OS
- Big and messy

Unics OS

- Project from Bell Labs
- Small, powerful, and portable

Original Unix OS was written in PDP-7 assembly code

- Writing an OS in assembly is hard work!

wikimedia commons

The Birth of C

Authors of Unix needed an easier to use, more portable programming language

Developed the **C** language

- By Dennis Ritchie and Brian Kernighan in 1969-1973
- Wanted a higher level language than assembly

Ported Unix to C in 1973

- Used by pretty much every major OS since

Since then C has evolved and been standardized

- ANSI C89 and ANSI C99

For the full history, see:

- The Development of the C Language, by Dennis Ritchie.

High vs Low Level

Low Level

- Full control over hardware, finely optimized

High Level

- Abstractions over hardware, greater expressiveness

```
.globl func
func:
   movl $5, %eax
   movl $1, %ebx
L1:cmpl $0, %eax
   je L2
   imull %eax, %ebx
   decl %eax
   jmp L1
L2:    ret
```

http://linuxgazette.net/issue94/ramankutty.html

```
int func(int val)
{
    int j=1;
    for (int i=1; i<=val; i++)
        j=j*i;
    return j;
}
```

Computing ??? in Assembly and C

Why Learn C?

Relatively high level

- General purpose and versatile

Relatively low level

- Allows direct access to memory

Well optimized

- C compilers often produce code as efficient as assembly
- No runtime system or VM

Ubiquitous

- Used to write nearly every OS, many software systems
- Popular for open source projects
- C compilers exist for most platforms

Differences from Java

- C has pointers
  - Direct access and control of memory
- Less strict type checking
- No objects or classes
- “Preprocessor language” runs before compilation
- Does not use a virtual machine

- C gives you greater power, but take care...
  - Less checking by compiler, none by run-time system
  - Will need debugger to track down errors

“Hello World” in C

Preprocessor command to load the **stdio** library

“main” function is always run at program start

```
#include <stdio.h>

int main ()
{
   printf ("CRUSH all humans!\n");

   return 0;
}
```

printf writes to screen. **\n** adds a new line

return 0 on success

Compiling with gcc

C source code must be compiled

Many C compilers exist

- We will use **gcc**

gcc syntax:

```
timwood@shell:~/lec-1$ gcc sourcefile.c -o execname
     // compiles "sourcefile.c" into an executable named "execname"

timwood@shell:~/lec-1$ gcc sourcefile.c
     // default executable name is "a.out"

timwood@shell:~/lec-1$ gcc sourcefile.c -std=c99
     // enforces the ANSI C99 standard
```

Your start with C

- Write a program that prints a quote, song lyric, proverb, or personal motto to the screen
  - Name the file: **myquote.c**
- **TYPE IN ALL YOUR CODE (no copy/paste)**
- Compile it with **gcc** and run it
- Some tips:
  - Remember to **#include** the **<stdio.h>** library
  - Create an **int main()** function
  - **printf** to the screen and be sure to end with a **\n**ewline
  - **/*** Put your name in a comment ***/**

(and you can take a break now)

My program

```
/* Prof. Wood’s quote */
#include <stdio.h>

int main ()
{
  printf("### Prof. Wood's Quote Program ###\n\nA quote by Mitch Radcliffe: \n");
  printf("\"A computer lets you make more mistakes faster than any invention in
human history-with the possible exceptions of handguns and tequila.\"\n\n");

  // Note: use \" to print a quotation mark

  return 0;
}
```

```
timwood@hobbes:~/lec-1$ gcc myquote.c -o quote
timwood@hobbes:~/lec-1$ ./quote
### Prof. Wood's Quote Program ###

A quote by Mitch Radcliffe:
"A computer lets you make more mistakes faster than any invention in
human history-with the possible exceptions of handguns and tequila."
timwood@hobbes:~/lec-1$
```

C Syntax

Most lines end with semicolon;

- Exception: control blocks (for, while, functions, etc)
- Semicolon is end of a statement

Comments

```
// a single line comment
/* multiple line
comment */
int /* bad, but valid, comment*/ j;
```

Variables: declare at top of a block

```
int j;    // an integer variable
int j=42; // declare and set a variable
```

Operators

C knows math:

- c = a * b
- sum += i   // short for sum=sum+i
- complex = b*(4 + 5/x) * (22-(4/5+j))
  - but it doesn't know exponents without help from math library functions

C knows logic:

- **0 is false**
- **1 is true** (and so is 2, 3, -104.567, etc)
  - No reserved keywords for true or false!
- **&** for bitwise AND, **&&** for logical AND
- **|** for bitwise OR, **||** for logical OR
- Test equality with == or !=
  - just one = is assignment!

1&1 = ?
1&0 = ?
0&0 = ?

1 | 1 = ?
0 | 1 = ?
1 | 0 = ?

110 & 101 = ?
111 & 000 = ?
110 | 101 = ?
111 | 000 = ?

bitwise returns a set of bits (some 0 some 1)

logical returns 0 or 1

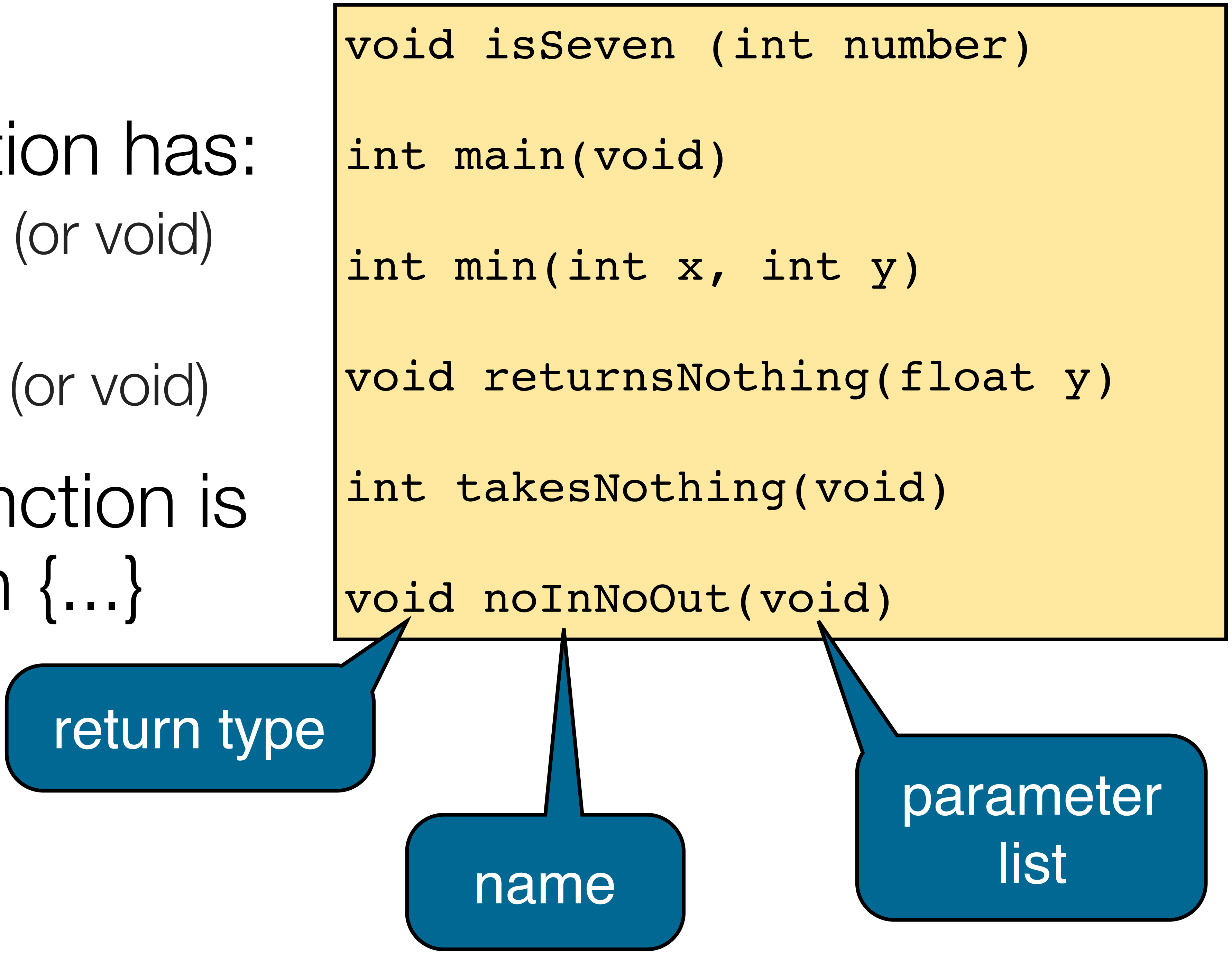
Functions

Let you modularize and reuse code

Every function has:

- Return type (or void)
- Name
- Parameters (or void)

Body of function is enclosed in {...}



```
void isSeven (int number)

int main(void)

int min(int x, int y)

void returnsNothing(float y)

int takesNothing(void)

void noInNoOut(void)
```

Control Flow

- Curly brackets designate control sections
- Parenthesis used for function arguments or math

```
void isSeven (int number)
{
    if (number == 7)
    {
        printf("Yes, %d is seven!\n", number);
        return;
    }
    else
        printf("Nope, not seven.\n");
}
int main(void)
{
    int i;
    for(i=0; i < 12; i++)
    {
        isSeven(i);
    }
    isSeven(2*(3+1)-1);
}
```

Using printf()

Function for printing **formatted** text to screen

- Use symbols to designate where variables should be filled in

```
double PI = 3.14159265358979323846264338;
int n = 2;
printf("My favorite numbers are %d and %lf", n, PI);
```

Can specify formatting preferences:

```
printf("pi = %.2lf\n", PI);
printf("pi = %.9lf\n", PI);
```

Must use the right identifier!

```
printf("pi = %d ????\n", PI);
```

Program output

```
My favorite numbers are
2 and 3.141593
pi = 3.14
pi = 3.141592654
pi = 2147483631 ????
```

C-Reference Sheet

- **Use the reference sheet to write as many of the following programs as you can. Work with your neighbor.**

- Write a program to print out the numbers 1 to 10 and their squares

- Write a function to print the first `n` Fibonacci numbers. Each Fibonacci number is the sum of the two preceding ones. The sequence starts out 0, 1, 1, 2, 3, 5, 8, 13, ...

- Write a program that uses a switch statement to take the day of week in number format (0...6) and print out the day (Mon...Sun). Test it with a for loop.

Where to Declare

Prior to C99, all variables had to be declared either at the top of a function, or globally (outside any function).

Since C99, variables can also be declared at top of blocks

Some compilers will let you declare variables wherever you like

- Best practice is to just declare at top of blocks

```
int main ()
{
    int numDaysInYear = 365;               // Works on all compilers
    printf ("numDaysInYear = %d\n", numDaysInYear);

    long numStarsInUniverse = 100000000000L; // Sometimes not allowed
    printf ("Number of stars = %ld stars\n", numStarsInUniverse);
}
```

Where to Declare

In for loop? Generally, no.

```
int main ()
{
    int i;

    for (i=0; i<10; i++) {        // Allowed.
        printf ("i=%d\n", i);
    }
    for (int j=0; j<10; j++) {
      // Not allowed. j needs to be at the top.
        printf ("j=%d\n", j);
    }
}
```

Some compilers will allow... other's won't

Variable Scope

Scope defines who knows about what variables

- **Global** or **local**?

Var defined inside a control block only exists within the control block

Var defined outside of all functions is global

What happens if defined locally and globally?

```
#include <stdio.h>

int numUsers = 0;

void addUser ()
{
  numUsers++;
}

void remUser ()
{
  numUsers--;
}

void main ()
{
  int peakUsers;
  addUser();
  addUser();
  peakUsers = numUsers;
  remUsers();
  remUsers();
  printf("Peak: %d\n,
    peakUsers);
}
```

Chars

rhymes with cars

Use **char** to hold a **single** letter

- Only uses 1 byte
- Range of a char?

```
char letter = 'a';
char letters = "abc"; // NO!
```

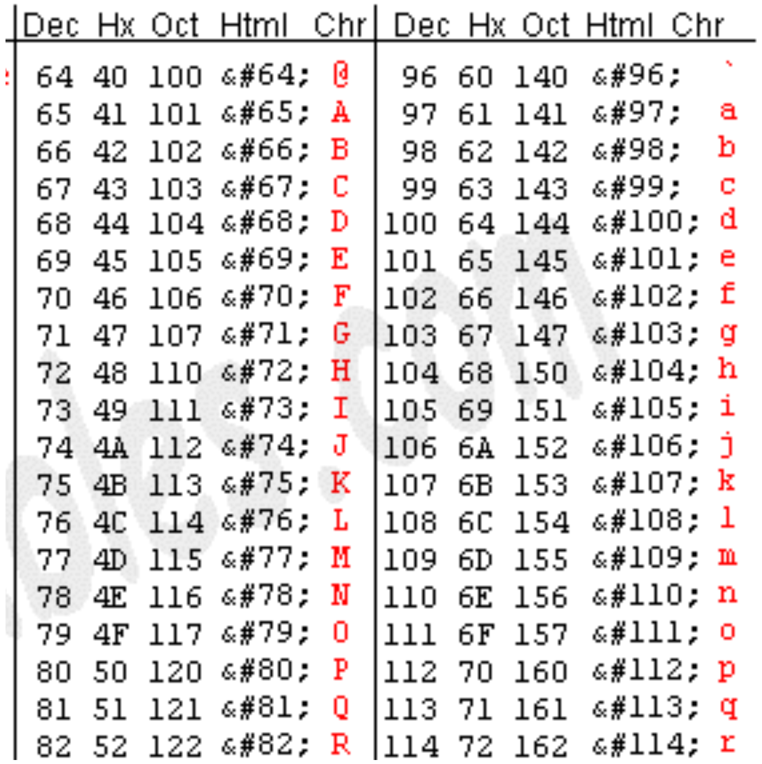
Bits -> numbers -> letters

C supports signed (+/-) and unsigned chars

- Why?

More uses of chars

- Byte-by-byte memory access!
- ...?



Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
64	40	100	&#64;	@	96	60	140	&#96;	`
65	41	101	&#65;	A	97	61	141	&#97;	a
66	42	102	&#66;	B	98	62	142	&#98;	b
67	43	103	&#67;	C	99	63	143	&#99;	c
68	44	104	&#68;	D	100	64	144	&#100;	d
69	45	105	&#69;	E	101	65	145	&#101;	e
70	46	106	&#70;	F	102	66	146	&#102;	f
71	47	107	&#71;	G	103	67	147	&#103;	g
72	48	110	&#72;	H	104	68	150	&#104;	h
73	49	111	&#73;	I	105	69	151	&#105;	i
74	4A	112	&#74;	J	106	6A	152	&#106;	j
75	4B	113	&#75;	K	107	6B	153	&#107;	k
76	4C	114	&#76;	L	108	6C	154	&#108;	l
77	4D	115	&#77;	M	109	6D	155	&#109;	m
78	4E	116	&#78;	N	110	6E	156	&#110;	n
79	4F	117	&#79;	O	111	6F	157	&#111;	o
80	50	120	&#80;	P	112	70	160	&#112;	p
81	51	121	&#81;	Q	113	71	161	&#113;	q
82	52	122	&#82;	R	114	72	162	&#114;	r

ASCII Codes

Arrays

Use arrays to store a “list” of variables

```
int main ()
{
  int profits[52];
  int w;
  int sum = 0;

  for(w=0; w < 52; w++)
  {
    profits[w] = w*10;
    sum += profits[w];
  }

  printf("Profits in third week: %d\n",
         profits[2];
  printf("Total profit: %d\n", sum);

  return 0;
}
```

array size must be a constant

array indexes start at 0!

Buffer Overflows

What happens in Java?

```
// bad Java code
int myArray[12];
myArray[99] = 666;
```

What happens in C?

```
// bad C code
int myArray[12];
myArray[99] = 666;
```

Java tracks the size of an array... C does not

- What is the cost/benefit?

C Array Exercises

- Fill in the missing functions in the **arrays.c** file
- Print out all the numbers

- How do your random numbers compare when you re-run the program? How do they compare to your neighbor?

C Array Exercises

- Fill in the missing functions in the **arrays.c** file
- Print out all the numbers
- How do your random numbers compare when you re-run the program? How do they compare to your neighbor?
- Calculate the sum and average of the numbers array
- If you want more randomness, try adding:

```
srand(time(NULL));
```

Exercise 1

- Available on website
- Using bit operations to store data very efficiently
- Due on Tuesday @ 11:59PM
- There will NOT be lab this Monday (Labor Day)
- Come to office hours for help!
- Or post questions to piazza

Summary

We've discussed:

- The origins of C
- A bit about unix/linux
- C programming syntax
  - variables and data types
  - control flow

Remember the website: **bit.ly/gwsofteng16**

Exercise 1 is due on Tuesday 9/11 at 11:59PM

No lab on Labor day