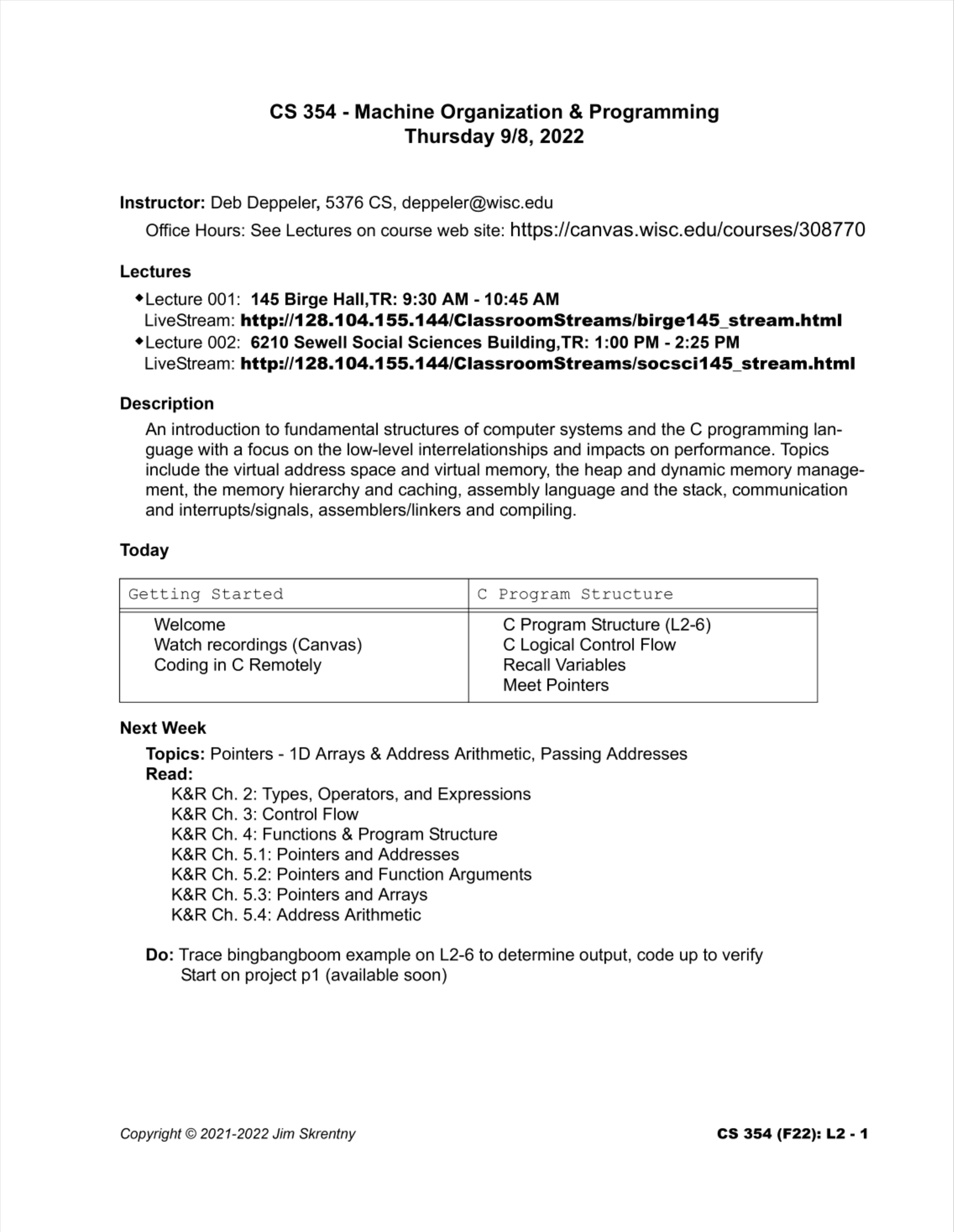
W1

Thursday, October 6, 2022 6:03 PM

W1



CS 354 - Machine Organization & Programming
Thursday 9/8, 2022

Instructor: Deb Deppeler, 5376 CS, deppeler@wisc.edu

Office Hours: See Lectures on course web site: https://canvas.wisc.edu/courses/308770

Lectures

- Lecture 001: **145 Birge Hall,TR: 9:30 AM - 10:45 AM**
  LiveStream: **http://128.104.155.144/ClassroomStreams/birge145_stream.html**
- Lecture 002: **6210 Sewell Social Sciences Building,TR: 1:00 PM - 2:25 PM**
  LiveStream: **http://128.104.155.144/ClassroomStreams/socsci145_stream.html**

Description

An introduction to fundamental structures of computer systems and the C programming language with a focus on the low-level interrelationships and impacts on performance. Topics include the virtual address space and virtual memory, the heap and dynamic memory management, the memory hierarchy and caching, assembly language and the stack, communication and interrupts/signals, assemblers/linkers and compiling.

Today

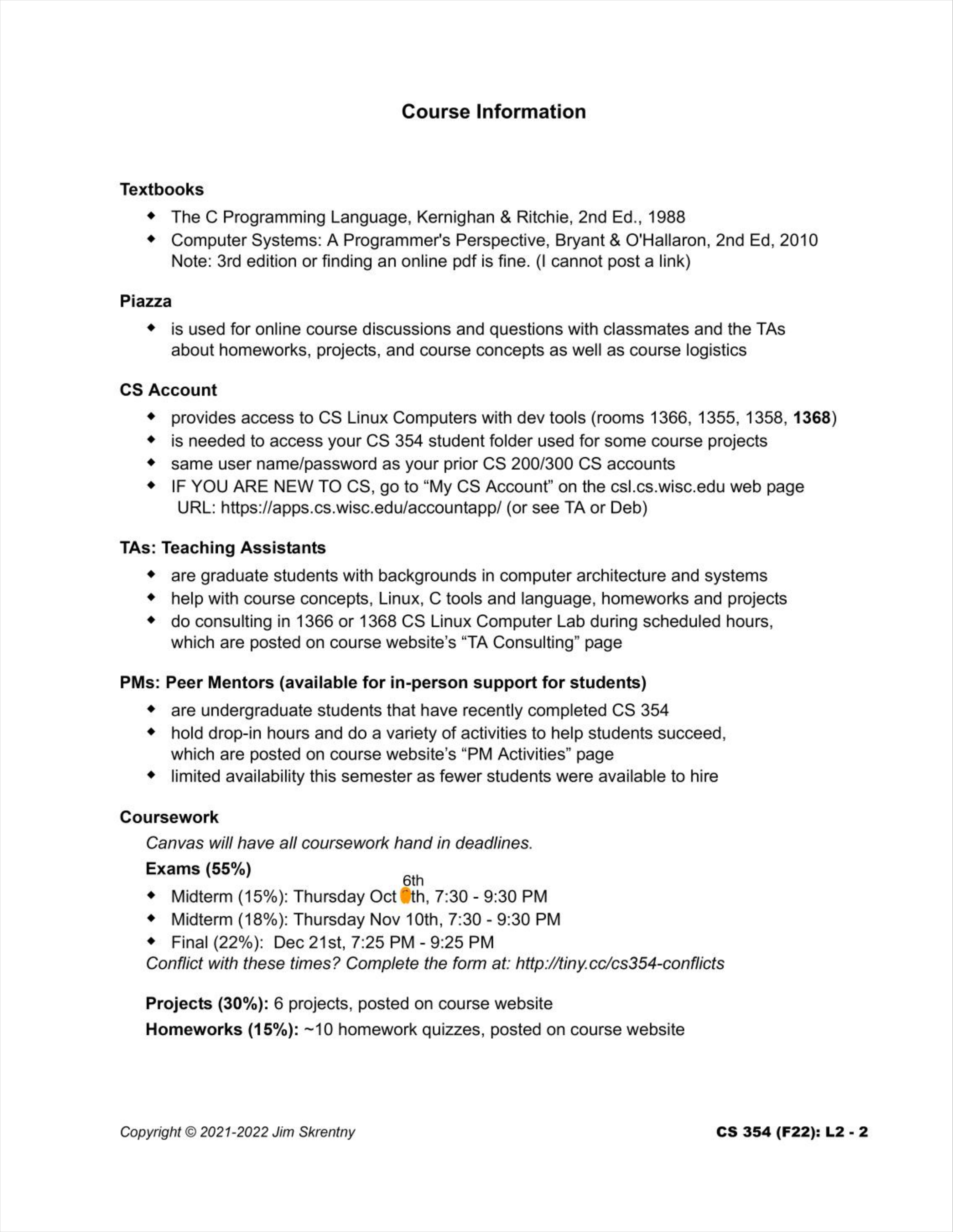
Getting Started	C Program Structure
Welcome Watch recordings (Canvas) Coding in C Remotely	C Program Structure (L2-6) C Logical Control Flow Recall Variables Meet Pointers

Next Week

Topics: Pointers - 1D Arrays & Address Arithmetic, Passing Addresses
Read:

K&R Ch. 2: Types, Operators, and Expressions
K&R Ch. 3: Control Flow
K&R Ch. 4: Functions & Program Structure
K&R Ch. 5.1: Pointers and Addresses
K&R Ch. 5.2: Pointers and Function Arguments
K&R Ch. 5.3: Pointers and Arrays
K&R Ch. 5.4: Address Arithmetic

Do: Trace bingbangboom example on L2-6 to determine output, code up to verify
Start on project p1 (available soon)



Course Information

Textbooks

- The C Programming Language, Kernighan & Ritchie, 2nd Ed., 1988
- Computer Systems: A Programmer's Perspective, Bryant & O'Hallaron, 2nd Ed, 2010
  Note: 3rd edition or finding an online pdf is fine. (I cannot post a link)

Piazza

- is used for online course discussions and questions with classmates and the TAs about homeworks, projects, and course concepts as well as course logistics

CS Account

- provides access to CS Linux Computers with dev tools (rooms 1366, 1355, 1358, **1368**)
- is needed to access your CS 354 student folder used for some course projects
- same user name/password as your prior CS 200/300 CS accounts
- IF YOU ARE NEW TO CS, go to “My CS Account” on the csl.cs.wisc.edu web page
  URL: https://apps.cs.wisc.edu/accountapp/ (or see TA or Deb)

TAs: Teaching Assistants

- are graduate students with backgrounds in computer architecture and systems
- help with course concepts, Linux, C tools and language, homeworks and projects
- do consulting in 1366 or 1368 CS Linux Computer Lab during scheduled hours, which are posted on course website’s “TA Consulting” page

PMs: Peer Mentors (available for in-person support for students)

- are undergraduate students that have recently completed CS 354
- hold drop-in hours and do a variety of activities to help students succeed, which are posted on course website’s “PM Activities” page
- limited availability this semester as fewer students were available to hire

Coursework

Canvas will have all coursework hand in deadlines.

Exams (55%)

6th

- Midterm (15%): Thursday Oct th, 7:30 - 9:30 PM
- Midterm (18%): Thursday Nov 10th, 7:30 - 9:30 PM
- Final (22%): Dec 21st, 7:25 PM - 9:25 PM

Conflict with these times? Complete the form at: http://tiny.cc/cs354-conflicts

Projects (30%): 6 projects, posted on course website

Homeworks (15%): ~10 homework quizzes, posted on course website

Coding in C Remotely - Get Connected to CS

❋ *Use the CS Linux lab computers for CS 354 programming.*

Access CS Linux Computers

Windows: get ssh program like MobaXterm and configure to connect to CS machines
Macs: open terminal and enter ssh <cs_account>@<machine>

machine names:
best-linux.cs.wisc.edu (Macs might cause issues with security certificates)
emperor-01.cs.wisc.edu through emperor-07.cs.wisc.edu
rockhopper-01.cs.wisc.edu through rockhopper-09.cs.wisc.edu
royal-01.cs.wisc.edu through royal-30.cs.wisc.edu
snares-01.cs.wisc.edu through snares-10.cs.wisc.edu

Learn some Linux Commands

command shell

→ How do you:
list the contents of a directory? Show details?

display what directory you're currently in?

copy a file?

remove a file?

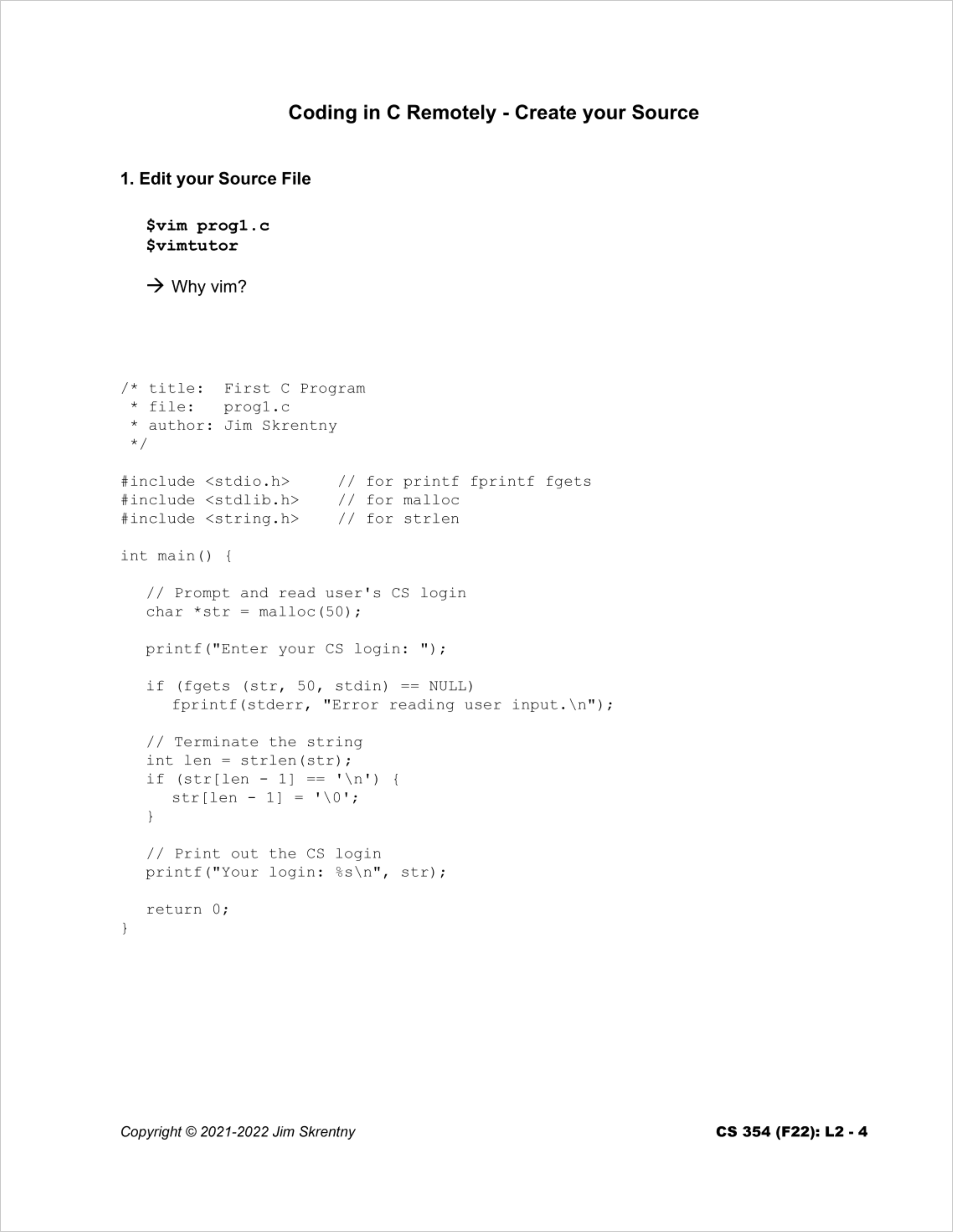
move to another directory? Up a directory?

make a new directory?

rename a file or directory?

remove a directory?

get more information about commands?



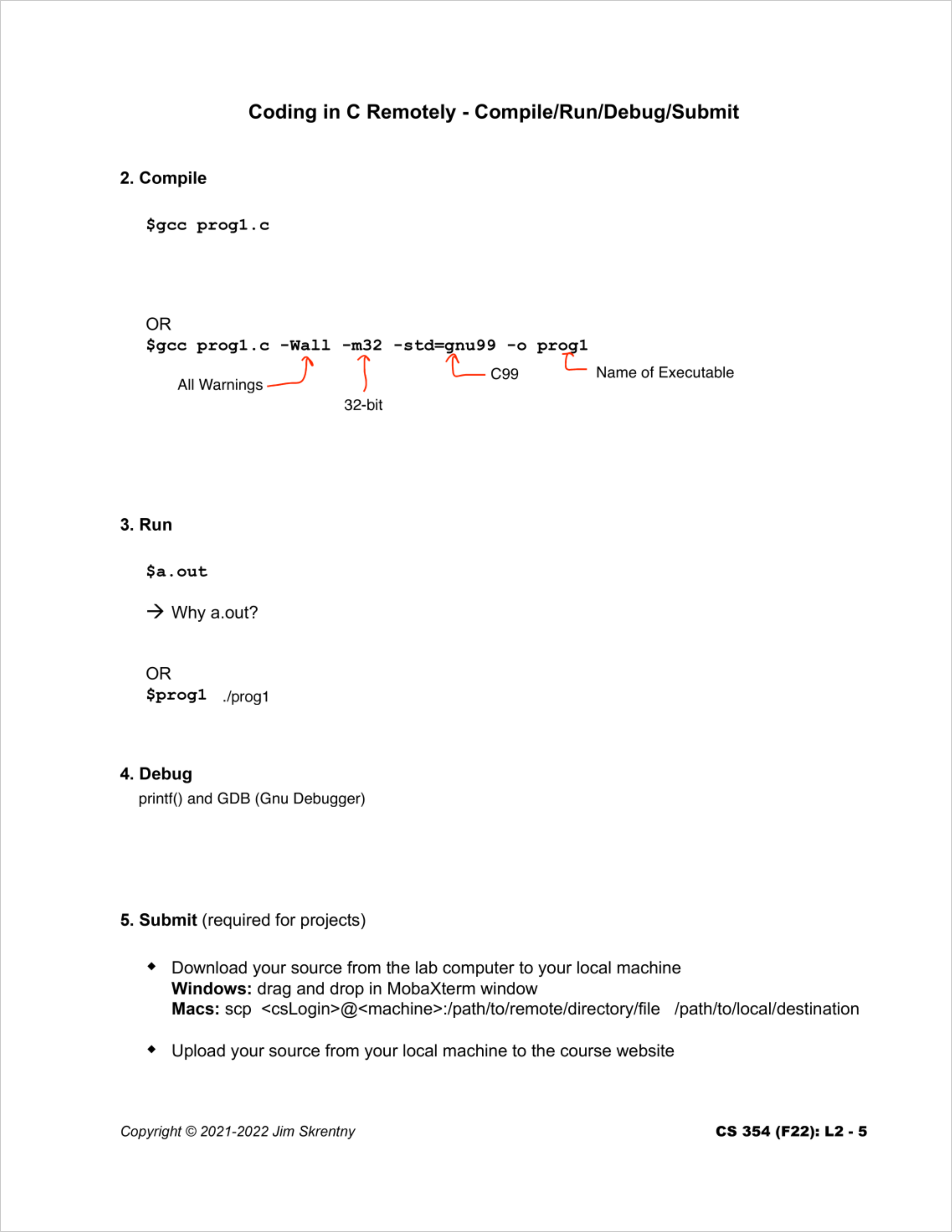
Coding in C Remotely - Create your Source

1. Edit your Source File

```
$vim prog1.c
$vimtutor
```

→ Why vim?

```
/* title:  First C Program
 * file:   prog1.c
 * author: Jim Skrentny
 */

#include <stdio.h>     // for printf fprintf fgets
#include <stdlib.h>    // for malloc
#include <string.h>    // for strlen

int main() {

   // Prompt and read user's CS login
   char *str = malloc(50);

   printf("Enter your CS login: ");

   if (fgets (str, 50, stdin) == NULL)
      fprintf(stderr, "Error reading user input.\n");

   // Terminate the string
   int len = strlen(str);
   if (str[len - 1] == '\n') {
      str[len - 1] = '\0';
   }

   // Print out the CS login
   printf("Your login: %s\n", str);

   return 0;
}
```



Coding in C Remotely - Compile/Run/Debug/Submit

2. Compile

```
$gcc prog1.c
```

OR

```
$gcc prog1.c -Wall -m32 -std=gnu99 -o prog1
```

All Warnings

32-bit

C99

Name of Executable

3. Run

```
$a.out
```

→ Why a.out?

OR

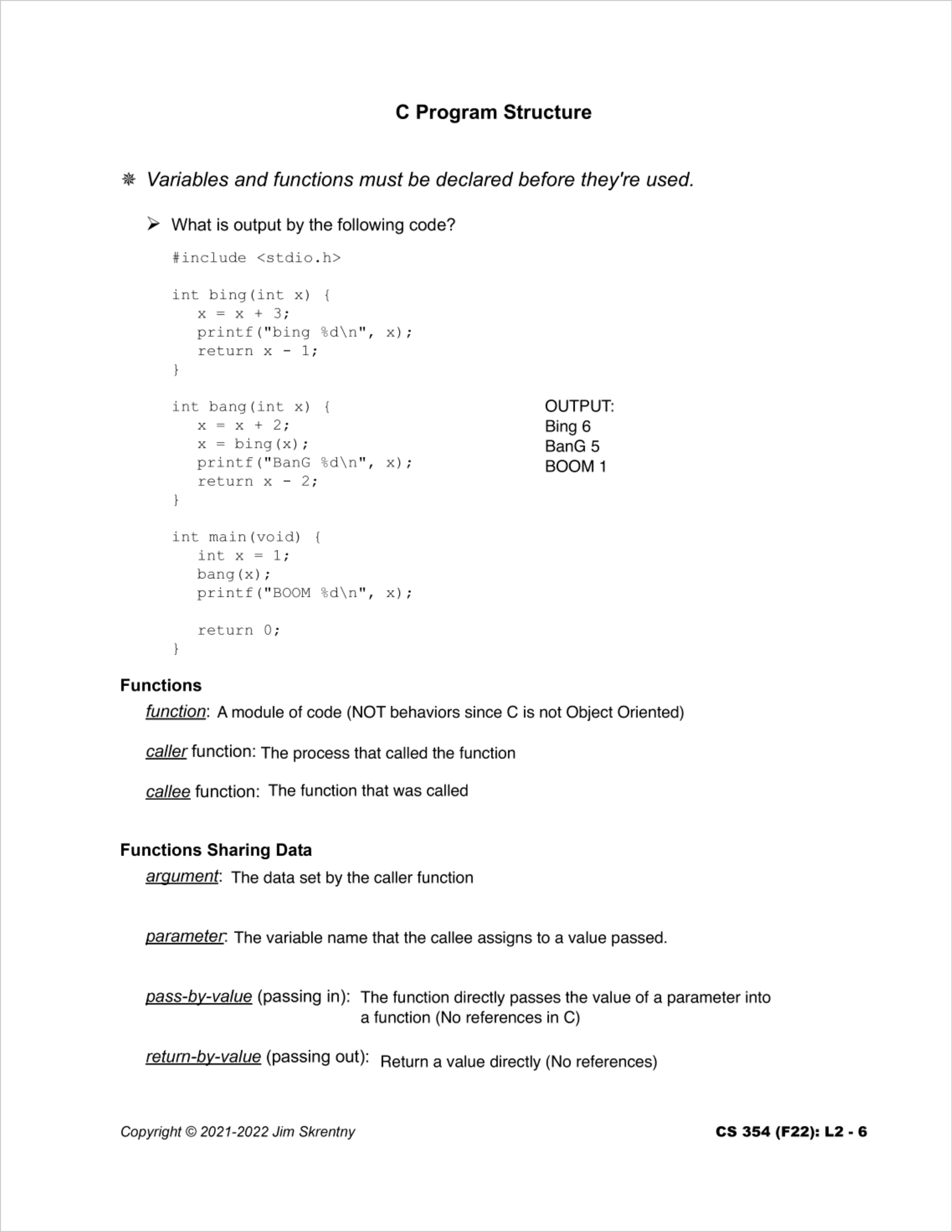
```
$prog1  ./prog1
```

4. Debug

printf() and GDB (Gnu Debugger)

5. Submit (required for projects)

- Download your source from the lab computer to your local machine
  Windows: drag and drop in MobaXterm window
  Macs: scp <csLogin>@<machine>:/path/to/remote/directory/file /path/to/local/destination
- Upload your source from your local machine to the course website



C Program Structure

- *Variables and functions must be declared before they're used.*

  - What is output by the following code?

```
#include <stdio.h>

int bing(int x) {
   x = x + 3;
   printf("bing %d\n", x);
   return x - 1;
}

int bang(int x) {
   x = x + 2;
   x = bing(x);
   printf("BanG %d\n", x);
   return x - 2;
}

int main(void) {
   int x = 1;
   bang(x);
   printf("BOOM %d\n", x);

   return 0;
}
```

OUTPUT:
Bing 6
BanG 5
BOOM 1

Functions

function: A module of code (NOT behaviors since C is not Object Oriented)

caller function: The process that called the function

callee function: The function that was called

Functions Sharing Data

argument: The data set by the caller function

parameter: The variable name that the callee assigns to a value passed.

pass-by-value (passing in): The function directly passes the value of a parameter into a function (No references in C)

return-by-value (passing out): Return a value directly (No references)

C Logical Control Flow

Sequencing

Execution starts in main.

Flows top to bottom (defining the order of functions does matter)

One statement then next

Selection

	Not Real	T	T	F
→ Which value(s) means true?	`true`	42	-17	0
	Use a Macro			

Anything other than zero is true.

if - else

Like Java but with fewer guard rails.

→ What is output by this code when `money` is 11, -11, 0?

```
if (money = 0)       printf("you're broke\n");
else if (money < 0)  printf("you're in debt\n");
else                 printf("you've got money\n");
```

It will always result in "you've got money" as that first line with ALWAYS set money to zero, so it just goes to the else block.

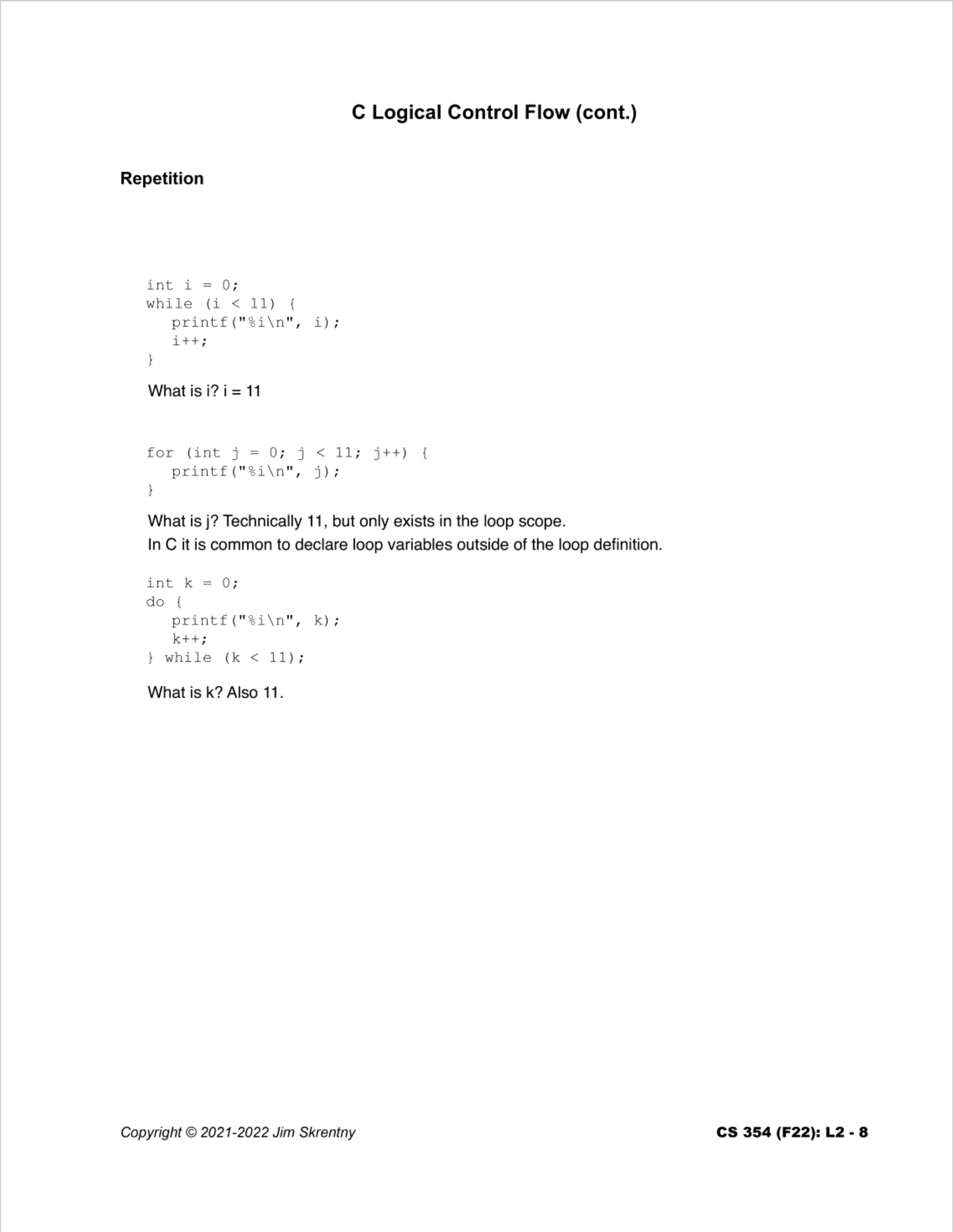
→ What is output by this code when the date is 10/31?

month = 10
day = 31

```
if (   10 ->  month)
   if (   31 ->   day)
      printf("Happy Halloween!\n");
else
   printf("It's not October.\n");
```

It prints out Happy Halloween…but even dates that are invalid will work as long as there are no zeroes in the date as anything other than zero will evaluate to true.

switch like Java, but no strings!



C Logical Control Flow (cont.)

Repetition

```
int i = 0;
while (i < 11) {
   printf("%i\n", i);
   i++;
}
```

What is i? i = 11

```
for (int j = 0; j < 11; j++) {
   printf("%i\n", j);
}
```

What is j? Technically 11, but only exists in the loop scope.
In C it is common to declare loop variables outside of the loop definition.

```
int k = 0;
do {
   printf("%i\n", k);
   k++;
} while (k < 11);
```

What is k? Also 11.

Recall Variables

What? A *scalar variable* is a primitive unit of storage. int, char, short, long, etc.

→ Draw a basic memory diagram for the variable in the following code:

```
void someFunction(){
   int i = 44;
```

Aspects of a Variable

identifier: name

value: data stored in variable's memory location

type: int, double, short, long, float -> representation of memory

address: starting location of the variable's memory (int*, char*…)

size: number of bytes for a variable. (use sizeof(type))

❋ *A scalar variable used as a source operand*

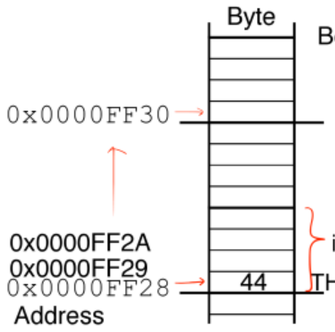
This is source, as i is being read. e.g., `printf("%i\n", i);`

❋ *A scalar variable used as a destination operand*

This is a destination, as i is being assigned.e.g., `i = 11;`

Linear Memory Diagram

A linear memory diagram is



Be mindful of the memory diagram's order!

	most significant						least
base 10							44
base 2	0000	0000	0000	0000	0000	0000	0010 1100
base 16	0	0	0	0	0	0	2 C
	most sig byte						least sig byte

THIS IS NOT THE NUMBER 44, IT IS THE HEX VAL 0x44 because its in a byte!

byte addressability: each byte has its own address.

endianess: byte ordering of variable's bytes when size > 1 byte

little endian: (CS 354 IA 32) Least significant (rightmost) byte is in the lowest address.

big endian: Most significant byte (leftmost) is in the lowest address.

Meet Pointers

What? A *pointer* variable is

◆

◆

Why?

◆

◆

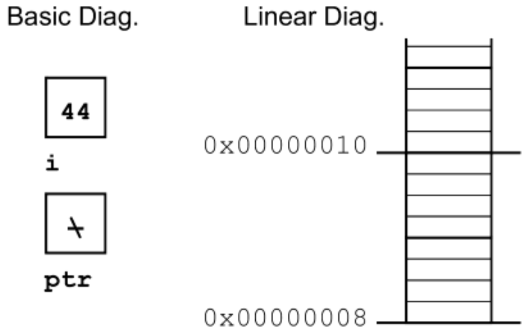
◆

◆

How?

→ Consider the following code:

```
void someFunction(){
   int i = 44;

   int *ptr = NULL;
```



→ What is `ptr`'s initial value? address? type? size?

pointer:

pointee:

`&` *address of* operator:

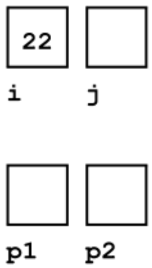
`*` *dereferencing* operator:

Practice Pointers

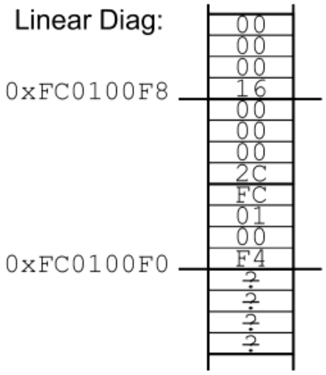
→ Complete the following diagrams and code so that they all correspond to each other:

```
void someFunction(){
   int i =
   int j = 44;
   int *p1 = &
   int *p2;  //at addr 0xFC0100EC
```

Basic Diag:



Linear Diag:



→ What is `p1`'s value?

→ Write the code to display `p1`'s pointee's value.

→ Write the code to display `p1`'s value.

→ Is it useful to know a pointer's exact value?

→ What is `p2`'s value?

→ Write the code to initialize `p2` so that it points to nothing.

→ What happens if the code below executes when `p2` is NULL?

```
printf("%i\n", *p2);
```

→ What happens if the code below executes when `p2` is uninitialized?

```
printf("%i\n", *p2);
```

→ Write the code to make `p2` point to `i`.

→ How many pointer variables are declared in the code below?

```
void someFunction(){
   int* p1, p2;
```

→ What does the code below do?

```
int **q = &p1;
```