

CS 354 - Machine Organization & Programming

Tuesday January 24 and Thursday January 26, 2023

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

Office Hours: See **Lectures** link on Canvas course

Lectures

♦ **Lecture 001 TR 9:30-10:45 AM 2650 Humanities**

Livestream link: http://128.104.155.144/ClassroomStreams/humanities2650_stream.html

♦ **Lecture 002 TR 2:30-3:45 PM 2340 Humanities Building**

Livestream link: http://128.104.155.144/ClassroomStreams/humanities2340_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 Course Info Getting Started in Linux EDIT COMPILE, RUN, DEBUG, SUBMIT	C Program Structure (L2-6) C Logical Control Flow, seq,sel,rep Recall Variables Meet Pointers

Next Week

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

Scan for review and try examples:

K&R Ch. 2: Types, Operators, and Expressions

K&R Ch. 3: Control Flow

K&R Ch. 4: Functions & Program Structure

Read and take notes before next week:

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%)

- ♦ Midterm (15%): Thursday Oct 6th, 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

Getting Started in Linux

✳ *Use the CSL Linux computers for all CS 354 programming.*

1. Log in or connect remotely to any CSL Linux Workstation (computers)

a. open your computer's **terminal** application

b. enter **ssh cslogin@machine.network**

cslogin: your username for CSL workstations. <https://apps.cs.wisc.edu/accountapp/>

machine: a physical or virtual machine on the CSL network

emperor-01 ... emperor-07

rockhopper-01 ... rockhopper-09

royal-01 ... royal-30

snare-01 ... snare-10

vm-instunix-01 ... vm-instunix-99

network: the CSL's network is **cs.wisc.edu**

c. ssh **cslogin@best-linux.cs.wisc.edu** (runs script to find least busy workstation)

Try some Linux Commands at the shell prompt

command shell

→ How do you:

list the contents of a directory?

Show details? Hidden files?

get more information about commands?

display what directory you're currently in?

copy a file?

remove a file?

move to another directory?

move "up" a directory?

make a new directory?

remove a directory?

rename a file or directory?

EDIT -- Create your C source code file

1. Create new or open existing file in a text-only editor

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

→ Why vim?
and
other editors?

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

Comments

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Imports; common libraries
usually incl.

```
int main(int argc, char *argv[])
```

```
    // Create space to save string of characters
```

```
    char * input_string = malloc(128); // creates a pointer pointing to a char
```

```
    // INPUT: prompt user for input
```

```
    printf("Enter your CS login: ");
```

Print

```
    // INPUT: read keyboard input into input_string variable
```

```
    if ( fgets(input_string, 50, stdin) == NULL )
```

```
    fprintf(stderr, "Error reading user input.\n");
```

Scanner-like function

```
    // Replace '\n' with '\0'
```

```
    int len = strlen(input_string);
```

Standard Input

```
    if ( input_string[len - 1] == '\n' ) {
```

```
        input_string[len - 1] = '\0';
```

```
    }
```

```
    // OUTPUT: print CS login to terminal
```

```
    printf("Your login: %s\n", str);
```

Output

```
    return 0;
```

```
        format; %s is string, %d is int iirc
```

```
}
```

COMPILE, RUN, DEBUG, SUBMIT

2. Compile -- build executable from C source

`$gcc prog1.c`

language standard

OR

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

output+name

filename

show all warnings

type of machine (32-bit in this case)

3. Run -- run executable (program) from command line

`$a.out`

→ Why a.out?

OR

`$prog1`

`./prog1`

4. Debug

1. Add print stmts: `printf("line20"); printf("x = %i\n", x)`

2. Use gdb `gdb prog1` start, b main, step, next, continue

5. Submit work to Canvas assignment (required for projects)

- ◆ Secure copy from lab computer to your local machine
`scp csLogin@best-linux.cs.wisc.edu:/path/to/remote/directory/srcfile local/destination`
- ◆ Refresh Canvas assignment page and upload files from your local machine

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;  
}
```

have to define functions before using them

```
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: like a method in Java

caller function: Function that calls another function

callee function: Function that is called

Functions Sharing Data

argument: Data passed to a function

parameter: Variable that stores information passed

pass-by-value (passing in): Copy of the argument value is passed in

return-by-value (passing out): Copy of the argument that is passed out

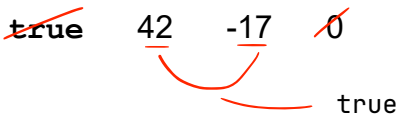
C Logical Control Flow

Sequencing

execution – starts in main, flows from top to bottom

Selection

→ Which value(s) means true? ~~true~~ 42 -17 ~~0~~ true

A diagram showing the truthiness of various values. The word 'true' is crossed out with a red line. The numbers '42' and '-17' are underlined with red lines. The number '0' is crossed out with a red line. A red bracket connects the underlined '42' and '-17' to the word 'true' on the right.

if - else

→ 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");
```

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

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

switch

C Logical Control Flow

Repetition

like Java

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

`i++`; uses value and then increments
`++i`; increments and then uses value

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

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


Recall Variables

What? A scalar variable is primitive unit of storage

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

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

name: i
value: 44
type: int

Aspects of a Variable

identifier: name

value: data stored

type: representation of data

address: Starting location of a variable's memory

size: # of bytes needed

✱ A scalar variable used as a source operand

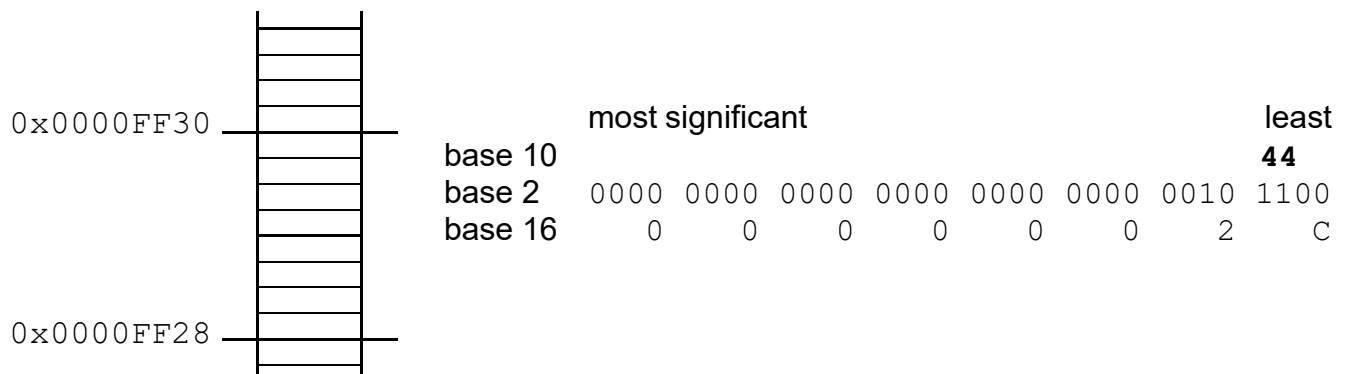
e.g., `printf("%i\n", i);`

✱ A scalar variable used as a destination operand

e.g., `i = 11;`

Linear Memory Diagram

A linear memory diagram is a view of memory as a sequence of bytes



byte addressability: each address identifies 1 byte, 8 bits/byte

endianess: byte ordering for variables that require more than 1 byte

little endian: least sig. byte in lowest address

big endian: most sig. byte in lowest address

Meet Pointers

What? A pointer variable is

- ◆ a scalar var whose value is a mem address
- ◆ similar to Java references

Why?

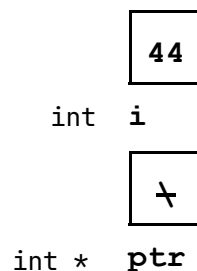
- ◆ for indirect address to memory
- ◆
- ◆
- ◆

How?

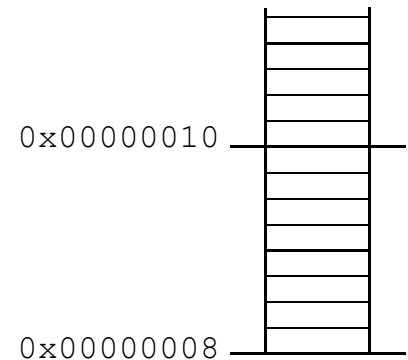
→ Consider the following code:

```
void someFunction() {  
    int i = 44;  
  
    int *ptr = NULL;  
}
```

Basic Diag.



Linear Diag.



→ What is ptr's initial value?

0x0: NULL

address?

0x00000000

type?

int *

size?

4 bytes

pointer: does the pointing, contains address

pointee: what's pointed at

& address of operator:

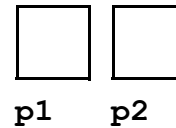
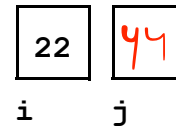
* dereferencing operator:

Practice Pointers

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

```
void someFunction() {
    int i = 22;    //0x16
    int j = 44;    //0x2C
    int *p1 = &j;
    int *p2;    //at addr 0xFC0100EC
```

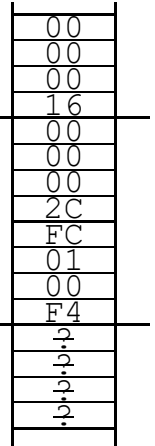
Basic Diag:



Linear Diag:

0xFC0100F8

0xFC0100F0



- What is p1's value?

0xFC0100F4

- Write the code to display p1's pointee's value.

```
printf(p1's pointee = %i\n", *p1)
```

- Write the code to display p1's value.

```
printf("p1 = %p\n", p1)
```

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

not particularly

- What is p2's value?

not initialized

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

```
int *p2 = NULL
```

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

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

segmentation fault

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

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

segfault or random value or zero

- Write the code to make p2 point to i.

```
p2 = &i
```

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

```
void someFunction() {
    int* p1, p2;          1
```

- What does the code below do?

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

pointer to a pointer