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 Feb 23
- Midterm (18%): ~~Thursday Nov 10th,~~ 7:30 - 9:30 PM April 6
- Final (22%): ~~Dec 21st, 7:25 PM - 9:25 PM~~ May 10 2:45-4:45

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.* IA-32

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
snares-01 ... snares-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 bash tcsh

→ How do you:
list the contents of a directory? ls Show details? Hidden files? ls -l ls -a

get more information about commands? man ls

display what directory you're currently in? pwd

copy a file? cp scp remove a file? rm

move to another directory? cd move "up" a directory? cd ..

make a new directory? mkdir

remove a directory? rmdir

rename a file or directory? mv

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? Deb's like it
and
other editors?

prog1.c

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

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

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

   // Create space to save string of characters
   char * input_string = malloc(128);
// INPUT: prompt user for input
printf("Enter your CS login: ");

// INPUT: read keyboard input into input_string variable
if ( fgets(input_string, 50, stdin) == NULL )
fprintf(stderr, "Error reading user input.\n");

// Replace '\n' with '\0'
int len = strlen(input_string);
if ( input_string[len - 1] == '\n' ) {
input_string[len - 1] = '\0';
}

// OUTPUT: print CS login to terminal
printf("Your login: %s\n", str);

return 0;

}
```

imports

flush buffer

newline char and null char at end of string. ex.) "abc" = a b c \n null

len = 4

a b c \0 \0
len = 3

input _string

COMPILE, RUN, DEBUG, SUBMIT

2. Compile -- build executable from C source

```
$gcc prog1.c
```

OR

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

rename output

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

```
$a.out
```

./a.out
assembler output

→ Why a.out?

OR

```
$prog1
```

./prog1

4. Debug

%s = string
%p = pointer
%c = char

1. Add print stmts: - printf("hello%s\n", varname)
2. Use gdb >gdb prog1
3. write test harness

gdb commands:
b main
b f1
next
step
print

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
  scp csLogin@best-linux.cs.wisc.edu:/p/course/CS354-deppeler/.vimrc
- 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;
}

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;
}
```

unless using CLAs

output

bing 6
BanG 5
Boom 1

Functions

function: like a method in java

caller function: starts a new function

callee function: the function being started

Functions Sharing Data

argument: data passed to a function

parameter: variables that store the data being passed

pass-by-value (passing in): copy of the argument value is made

return-by-value (passing out): copy of the return value

C Logical Control Flow

Sequencing

execution starts in main
flows top to bottom
one statement then, the next
; terminates statements
{ } indicate blocks of code

Selection

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

boolean and string not types in c ? T T F
no boolean true value in c

if - else like java

→ 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");
```

dangling else

use { } to avoid this

switch switch statements same as java

C Logical Control Flow

Repetition

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

same as java

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

++j

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

don't forget ;

Recall Variables

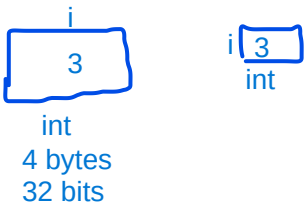
char, int, double

What? A *scalar variable* is a primitive unit of storage

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

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

name, value, type
i 3 int



Aspects of a Variable

identifier: name

value: data stored

type: representation of data

IA32 architecture used for this class
int = 4 bytes = 32 bits

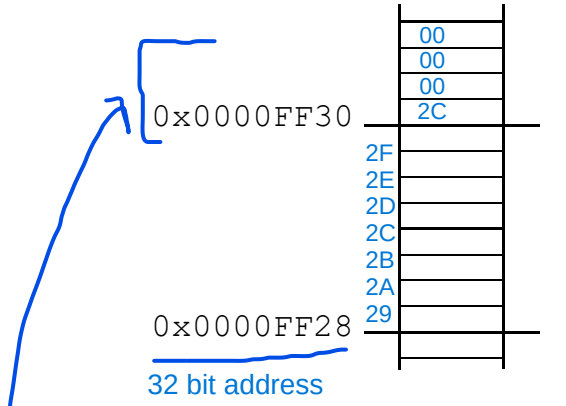
address: starting location of variable's memory

size: number of bytes needed

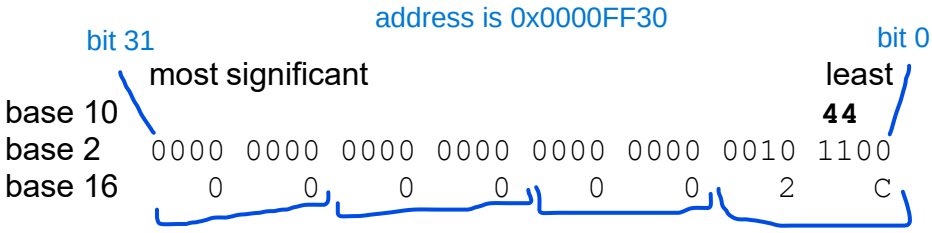
- *A scalar variable used as a source operand*
  e.g., `printf("%i\n", i);`
- *A scalar variable used as a destination operand*
  e.g., `i = 11;`

Linear Memory Diagram

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



32 bit address



byte addressability: each address identifies one byte

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

little endian: least significant byte at lowest address — CS354/IA-32

big endian: most significant byte at lowest address

Meet Pointers

What? A *pointer* variable is

- a scalar var whose value is a memory addresss
- similar to Java references

Why?

- for indirect access to memory
- for indirect access to functions
- they are common in C libraries
- for access to memory mapped hardware

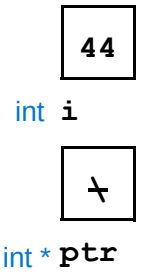
How?

→ Consider the following code:

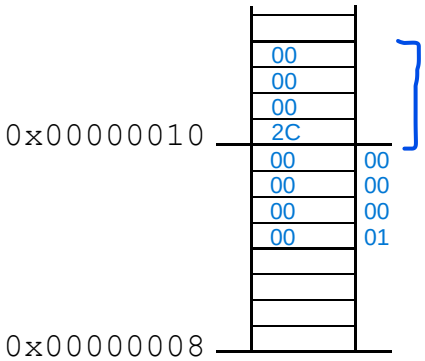
```
void someFunction(){
   int i = 44;

   int *ptr = NULL;
```

ptr = &i

Basic Diag.



Linear Diag.



→ What is `ptr`'s initial value? 0x0 address? 0xC 0x0000000C type? int * size? 4

pointer: does the pointing, contains address of pointee

pointee: what is pointed at

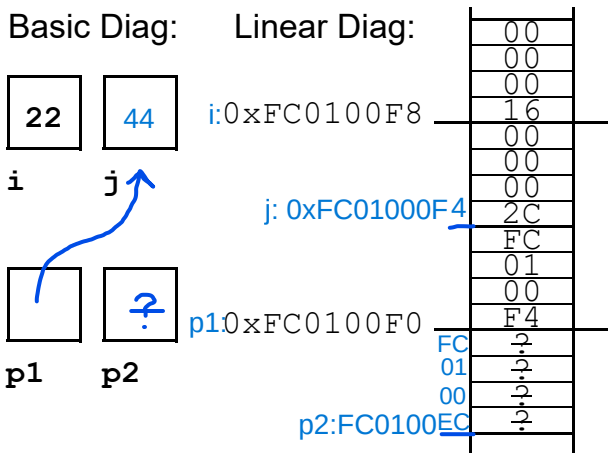
`&` *address of* operator:

`*` *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
```



→ What is `p1`'s value? 0xFC0100F0

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

printf("p1 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 usually .... (except maybe cs354)

→ What is `p2`'s value?

undefined / "uninitialized"

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

int *p2 = 0x0; or: int *p2 = NULL; (preferred method)

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

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

SEG FAULT, no permission to read address 0x0

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

```
printf("%i\n", *p2);
```
might be SEG FAULT, might be random value, might be 0

→ 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 pointer, if you wanted to declare 2 pointers, do int *p1, *p2; (just use 2 lines though)

→ What does the code below do?

```
int **q = &p1;
```
pointer to a pointer to an int