Memory, Data, & Addressing II
CSE 351 Winter 2024

Instructor:
Justin Hsia

Teaching Assistants:
Adithi Raghavan
Aman Mohammed
Connie Chen
Eyoel Gebre
Jiawei Huang
Malak Zaki
Naama Amiel
Nathan Khuat
Nikolas McNamee
Pedro Amarante
Will Robertson

http://xkcd.com/138/
Relevant Course Information

- Lab 0 due today @ 11:59 pm
  - *You will revisit the concepts from this program in future labs!*

- HW2 due Wednesday, HW3 due Friday
  - Autograded, unlimited tries, no late submissions

- Lab 1a released today, due next Monday (1/15)
  - Pointers in C (requires course material through bit shifting in Lesson 5)
  - Last submission graded, can optionally work with a partner
    - One student submits, then add their partner to the submission
  - Short answer “synthesis questions” for after the lab
Late Days

❖ You are given 5 late day tokens for the whole quarter
  ▪ Tokens can only apply to Labs
  ▪ Can earn up to 2 more via answering lecture polling questions

❖ Count lateness in days (even if just by a second)
  ▪ Special: weekends count as one day
  ▪ No submissions accepted more than two days late

❖ Late penalty is 10% deduction of your score per day
  ▪ Only late labs are eligible for penalties
  ▪ Penalties applied at end of quarter to maximize your grade

❖ Use at own risk – don’t want to fall too far behind
  ▪ Intended to allow for unexpected circumstances
Memory & Data II
Lesson Summary (1/2)

❖ Pointers are data objects that hold addresses
  ▪ Type of pointer determines size of thing being pointed at, which could be another pointer
  ▪ & = “address of” operator
  ▪ * = “value at address” or “dereference” operator
  ▪ NULL is a constant for a pointer to “nothing”

❖ Can visualize using box-and-arrow diagrams:
Lesson Summary (2/2)

❖ Arrays are adjacent locations in memory storing the same type of data
  ▪ Strings are null-terminated arrays of characters (ASCII)

❖ Pointer arithmetic scales by size of target type
  ▪ Convenient when accessing array-like structures in memory:  \( a[i] \leftarrow *(a + i) \)
  ▪ Be careful when using – particularly when casting variables

\[
\begin{array}{c|c|c|c}
\text{str} & 0x33 & 0x35 & 0x31 & 0x00 \\
\end{array}
\]

&str \rightarrow 0x7ff...7bf8;
Lesson Q&A

❖ Learning Objectives:
  ▪ Define pointers and their significance in computer memory organization.
  ▪ Declare, initialize, and manipulate pointers in C using address-of, dereference, and arithmetic operators.
  ▪ Handle I/O operations with C strings, accounting for the null character.

❖ What lingering questions do you have from the lesson?
  ▪ Chat with your neighbors about the lesson for a few minutes to come up with questions
Polling Questions (1/2)

- `int x = 351;`  
  `char* p = &x;`  
  `int ar[3];`

- How much space does the variable `p` take up?
  - A. 1 byte
  - B. 2 bytes
  - C. 4 bytes
  - D. 8 bytes

- Which of the following expressions evaluate to an address?
  - A. `x + 10`
  - B. `p + 10`
  - C. `&x + 10`
  - D. `*(&p)`
  - E. `ar[1]`
  - F. `&ar[2]`
Polling Questions (2/2)

❖ The variable values after Line 3 executes are shown on the right. What are they after Line 5?

```c
void main() {
   int a[] = {0x5,0x10};
   int* p = a;
   p = p + 1;
   *p = *p + 1;
}
```

<table>
<thead>
<tr>
<th></th>
<th>Data (hex)</th>
<th>Address (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[0]</td>
<td>5</td>
<td>0x100</td>
</tr>
<tr>
<td>a[1]</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

p  a[0]  a[1]
(A) 0x101  0x5  0x11
(B) 0x104  0x5  0x11
(C) 0x101  0x6  0x10
(D) 0x104  0x6  0x10
Homework Setup

❖ How much memory (in bytes) is allocated for the following?

- `short s;`
- `short* p;`
- `short ar[351];`
- "short"
Memory & Data II — Context
Examining Data Representations

❖ Code to print byte representation of data
  ▪ Treat any data type as a byte array by casting its address to char*
  ▪ C has unchecked casts  !! DANGER !!

```
void show_bytes(char* start, int len) {
    int i;
    for (i = 0; i < len; i++)
        printf("%p\t0x%.2hhX\n", start+i, *(start+i));
    printf("\n");
}
```

❖ printf legend:
  ▪ Special characters: \t = Tab, \n = newline
  ▪ Format specifiers:  %p = pointer,
                         %.2hhX = 1 byte (hh) in hex (X), padding to 2 digits (.2)
Examining Data Representations

❖ Code to print byte representation of data
  ▪ Treat any data type as a byte array by casting its address to char*
  ▪ C has unchecked casts  !! DANGER !!

```c
void show_bytes(char* start, int len) {
    int i;
    for (i = 0; i < len; i++)
        printf("%p\t0x%.2hhX\n", start+i, *(start+i));
    printf("\n");
}

void show_int(int x) {
    show_bytes( (char*) &x, sizeof(int));
}
```
show_bytes Execution Example

```c
int x = 123456; // 0x00 01 E2 40
printf("int x = %d;\n", x);
show_int(x);    // show_bytes((char*) &x, sizeof(int));
```

❖ Result (Linux x86-64):

- **Note:** The addresses will change on each run (try it!), but fall in same general range

```c
int x = 123456;
0x7fffb245549c  0x40
0x7fffb245549d  0xE2
0x7fffb245549e  0x01
0x7fffb245549f  0x00
```
Java References

❖ In Java, everything that is not a primitive data type is an object
  ▪ An object variable is actually a “reference” – a restricted pointer

```
class Record { ... }
Record x = new Record();
```

❖ Reference restrictions:
  ▪ No pointer arithmetic, just reassignment
    ▪ Reassignment must adhere to rules set by typing system (e.g., inheritance)
  ▪ References can only be “dereferenced” in ways that match class definition
    ▪ e.g., calling a method, accessing a field in object

❖ All higher-level languages use pointers/addresses under the hood, but likely abstracted away from the programmer
Discussion Question

❖ Discuss the following question(s) in groups of 3-4 students

- I will call on a few groups afterwards so please be prepared to share out
- Be respectful of others’ opinions and experiences

❖ Brainstorm some reasons why you think the designers of C (released in 1972) gave its programmers access to “raw” pointers.

- What might these reasons say about the implicit values embedded in C?
Group Work Time

- During this time, you are encouraged to work on the following:
  1) If desired, continue your discussion
  2) Work on the homework problems
  3) Work on the lab (if applicable)

- Resources:
  - You can revisit the lesson material
  - Work together in groups and help each other out
  - Course staff will circle around to provide support