Introductions, C Refresher

Computer Systems Programming, Spring 2025

Instructor: Travis McGaha

pollev.com/tqm

- How are you?

Administrivia

- First Assignment (HW00 simple_string)
  - Releases Tomorrow
  - "Due" Friday next week 01/24
  - Extended to be due Wednesday the 28th (course selection period ends)
  - Mostly a C refresher
- Check-in 00
  - Releases tomorrow
  - Short unlimited attempt quiz
  - Extended to be due Wednesday the 28th (course selection period ends)
- Pre semester Survey
  - Anonymous
  - Due Wednesday the 28th

Lecture Outline

- **Introduction & Logistics**
  - **Course Overview**
  - **Assignments & Exams**
  - **Policies**
- C "Refresher"
  - Context in this course
  - memory
  - Pointers
  - Arrays
  - Structs
  - The heap
  - const

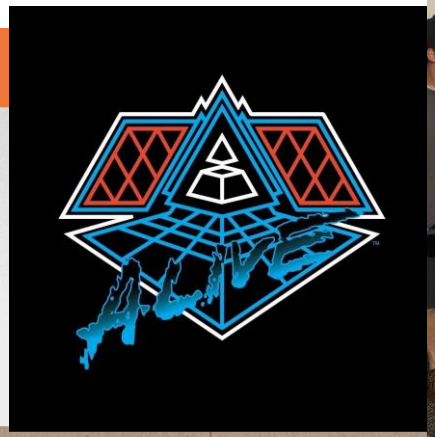
Instructor: Travis McGaha

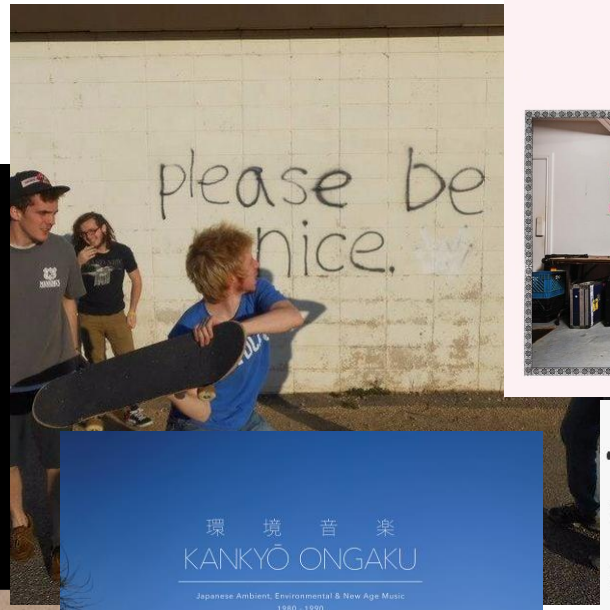
- UPenn CIS faculty member since August 2021
  - Currently my Eighth semester at UPenn
  - Fourth Semester with CIT 5950… and I am still trying new stuff
  - Lots of the same content, but in a different order, new assignments and more of a focus on C++

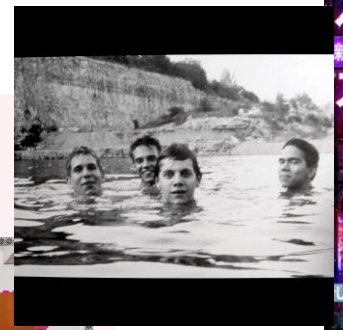
- Education: University of Washington, Seattle
  - Masters in Computer Science in March 2021
  - Bachelors in Computer Engineering in June 2019
  - Instructed a course that covers very similar material

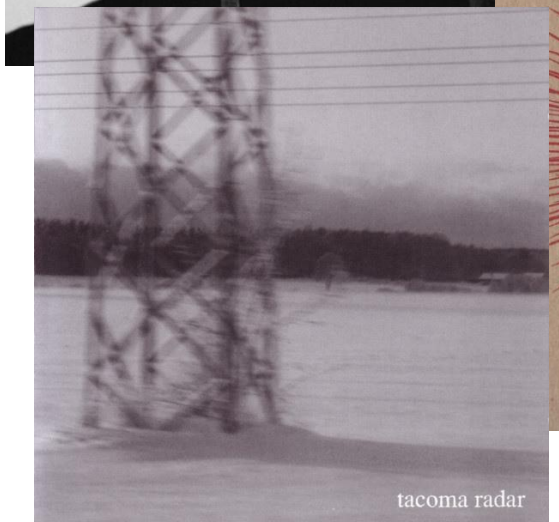
Instructor: Travis McGaha

- I like most music

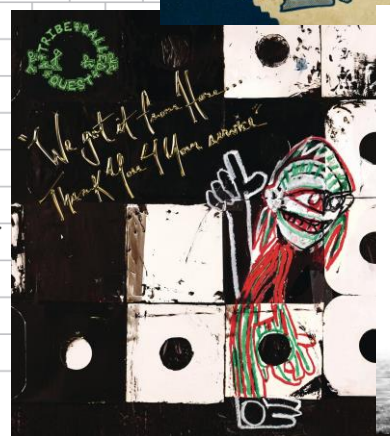








PINK FLOYD THE WALL

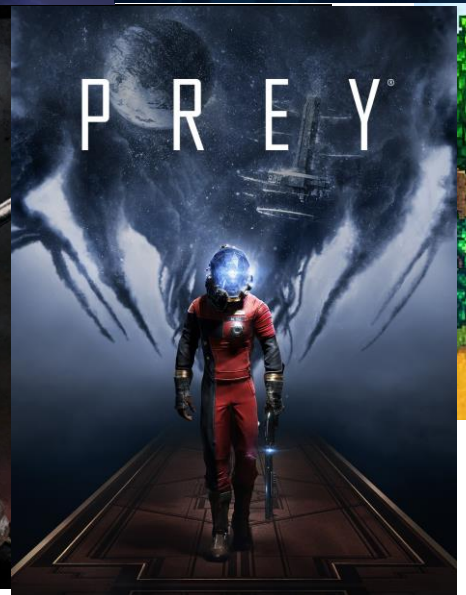


Instructor: Travis McGaha

- I like animals and going outside (especially birds, cats and mountains)

Instructor: Travis McGaha

- I like video games



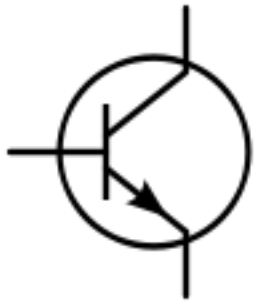
Instructor: Travis McGaha

- I have a general dislike of food
  (Breakfast is pretty good tho)

Instructor: Travis McGaha

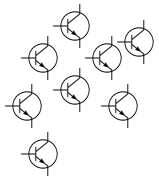
- I care a lot about your actual learning and that you have a good experience with the course
- I am a human being and I know that you are one too. If you are facing difficulties, please let me know and we can try and work something out.
- More on my personal website: https://www.cis.upenn.edu/~tqmcgaha/

Course Overview

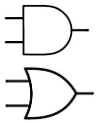


Course Overview

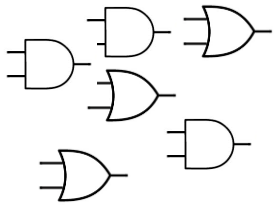
Course Overview



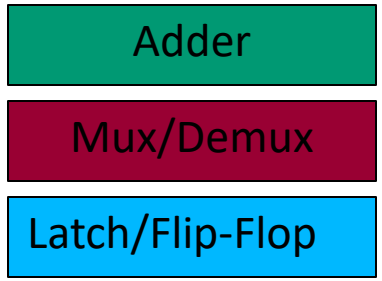
Course Overview



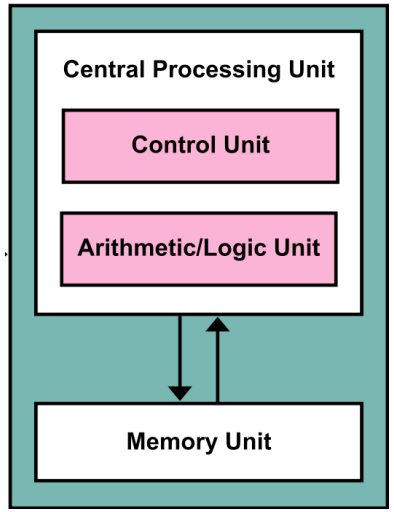
Course Overview



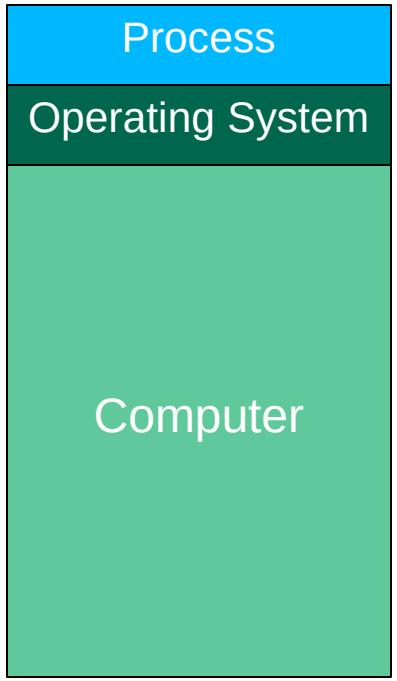
Course Overview



Course Overview



Course Overview



Course Overview

“Lies-to-children”

- "The necessarily simplified stories we tell children and students as a foundation for understanding so that eventually they can discover that they are not, in fact, true."
  - Andrew Sawyer (Narrativium and Lies-to-Children: 'Palatable Instruction in 'The Science of Discworld' ‘)

“Lies-to-children”

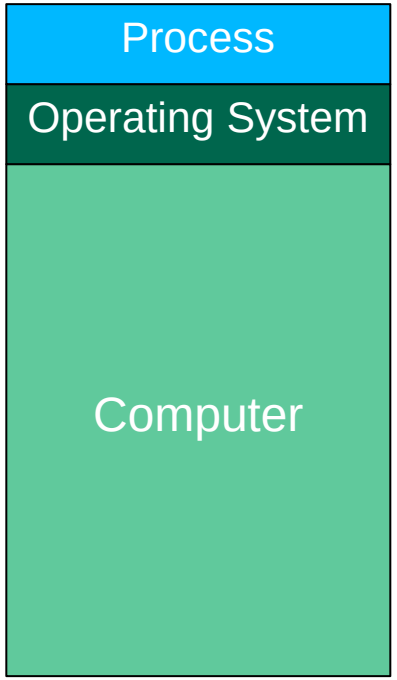
- "A lie-to-children is a statement that is false, but which nevertheless leads the child's mind towards a more accurate explanation, one that the child will only be able to appreciate if it has been primed with the lie"
  - Terry Pratchett, Ian Stewart & Jack Cohen (The Science of Discworld)

Question

- What color is the sky?

We lied to you (but in a good way)

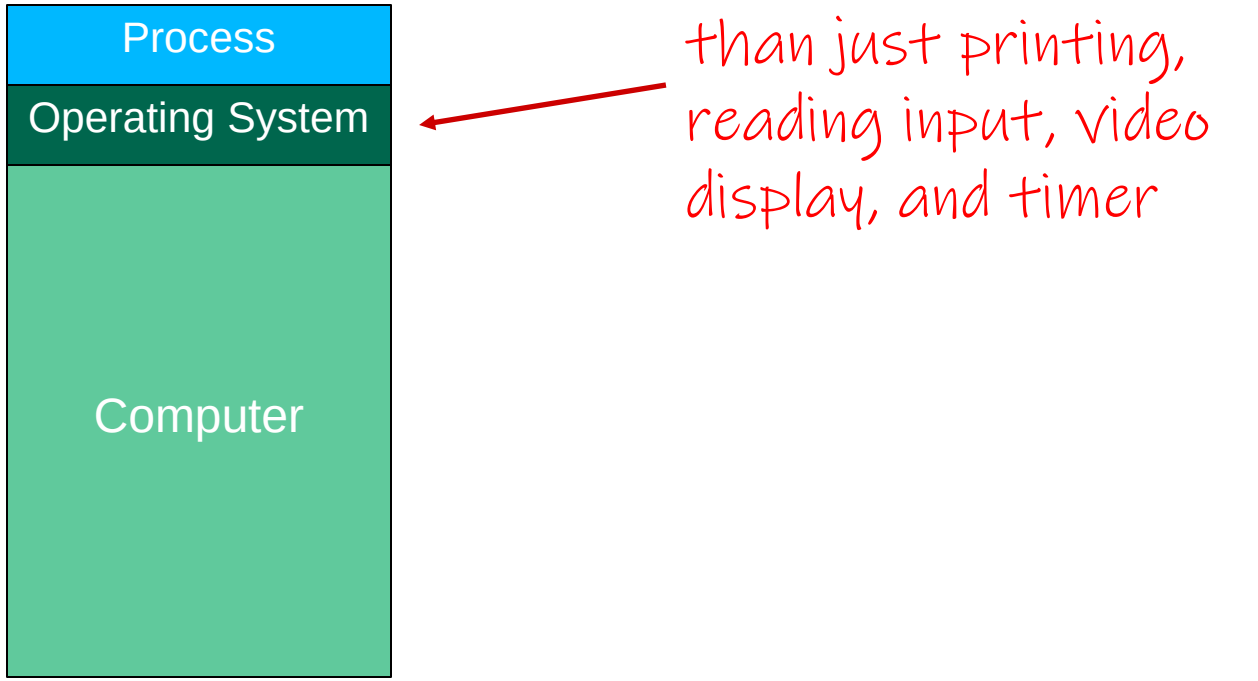
- Is the LC4 model for a computer true? **Eh……. no**
- Is it a useful model? **Yes**



We lied to you (but in a good way)

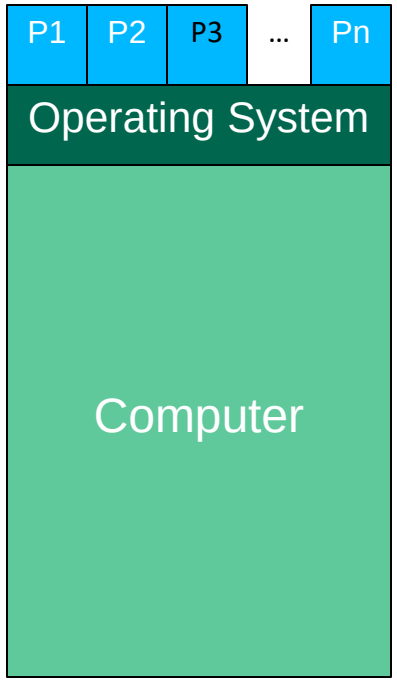
- Is memory one giant array of bytes? **Eh……. no**
- Is this a useful model? **Yes**

Course Overview



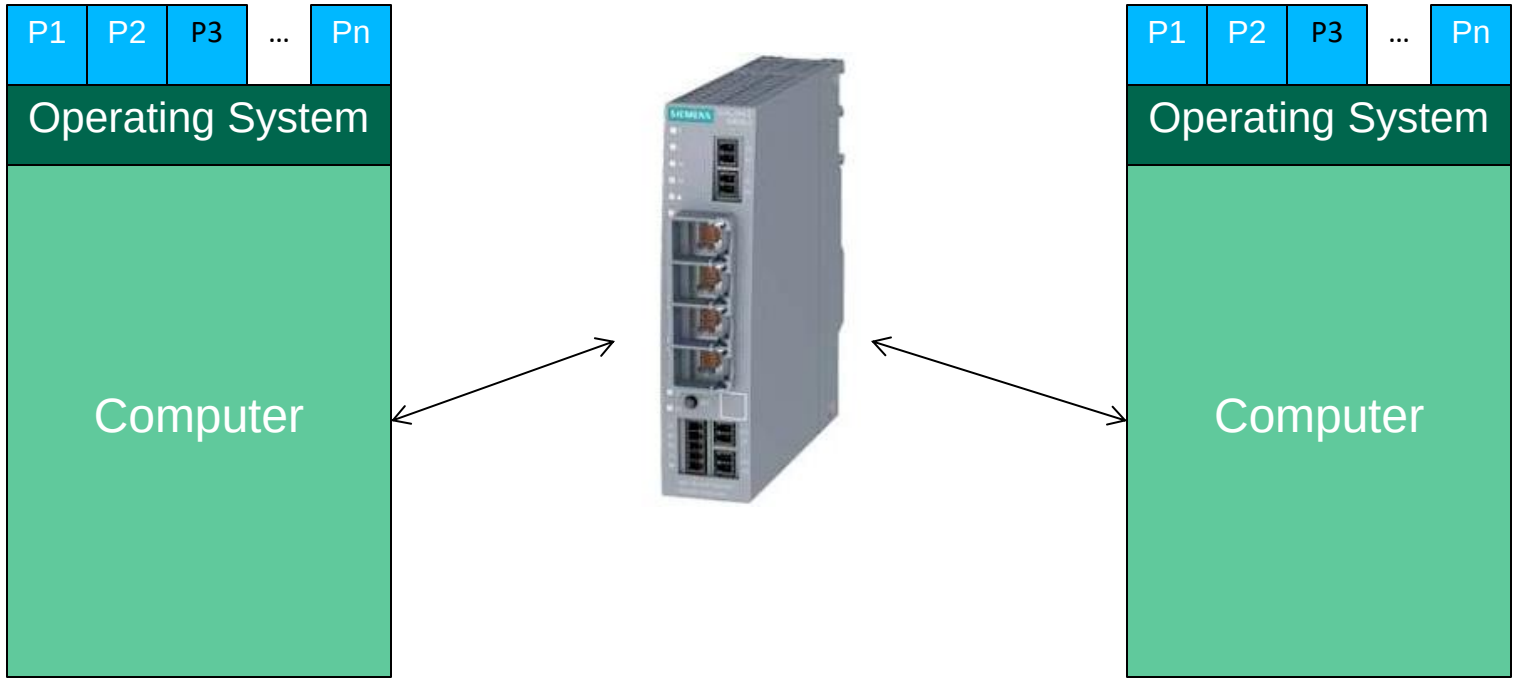
OS does A LOT more than just printing, reading input, video display, and timer

Course Overview



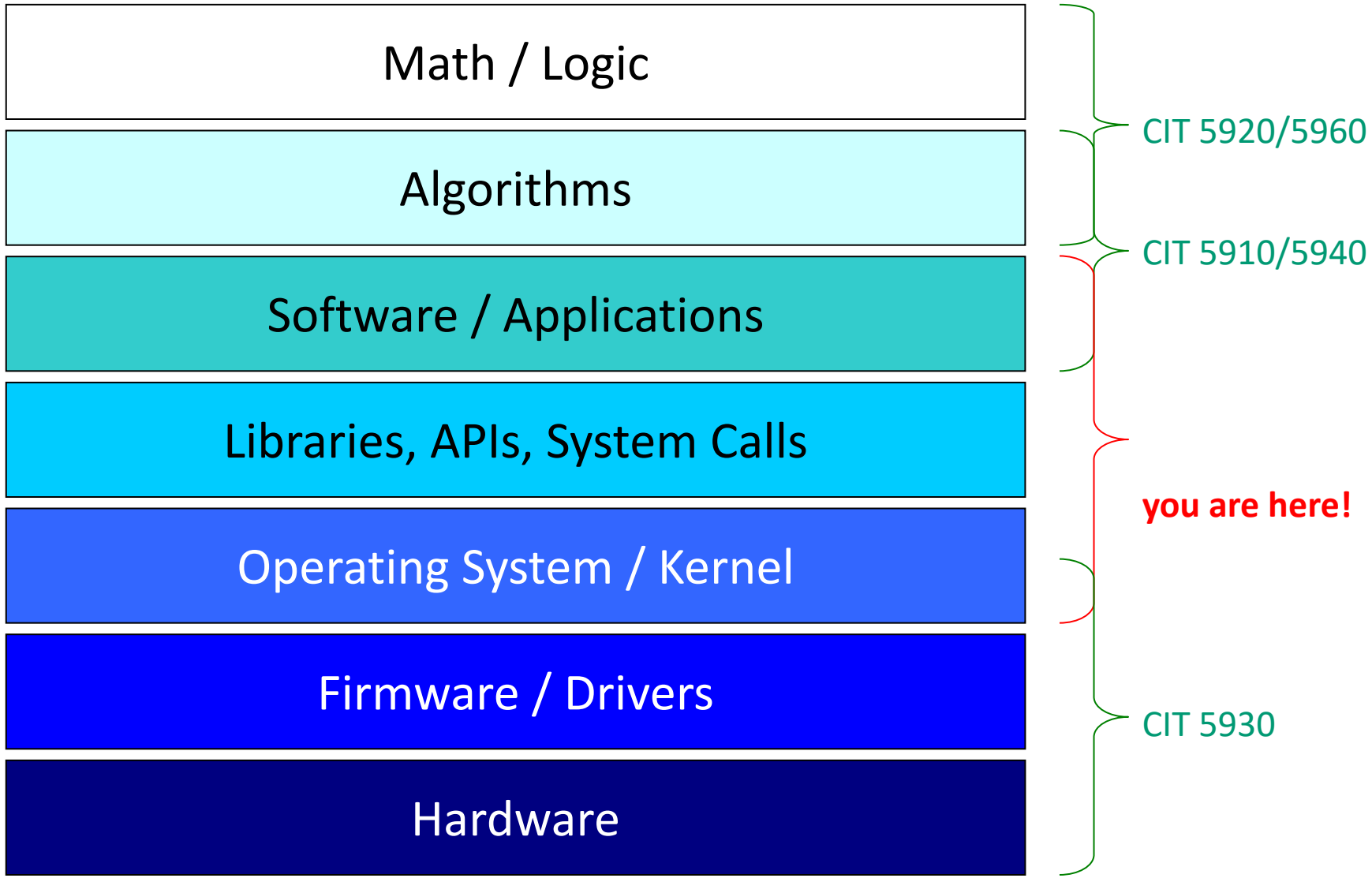
THERE IS A LOT GOING ON TO SUPPORT THIS

Course Overview



Course Overview

Course Overview



I'm going to lie to you (but in a good way)

- "All models are wrong, but some are useful."
  - Same source as below.

- "If it were necessary for us to understand how every component of our daily lives works in order to function - we simply would not."
  - AnRel (UNHINGED: A Guide to Revolution for Nerds & Skeptics)

- This course will reveal more details, but there is still a ton I am leaving out. Even what I say that is accurate, will likely change in the future.

Prerequisites

- Course Prerequisites:
  - CIT 5930

- What you should be familiar with already:
  - **C** programming experience
    - Familiarity with basic data structures
  - C Memory Model
  - Computer Architecture Model
  - Basic UNIX command line skills

- Will still review some of these with the beginning of the semester ☺

CIT 5950 Learning Objectives

- To leave the class with a better understanding of:
  - C++
  - How a lot of software level structures (e.g. vector and string) work
  - How software "interfaces" with the Operating System
  - How a computer runs/manages multiple programs
  - Various system resources and how to apply those to code
    - Threads, networking, file I/O

- Topics list/schedule can be found on course website
  - Note: This is tentative

Disclaimer

- A lot of the course is tentative
  - Travis has taught this before but is CHANGING A LOT this time

- This is a digest, **READ THE SYLLABUS**
  - https://www.seas.upenn.edu/~cit5950/current/documents/syllabus
  - Note: Syllabus is still being updated

Course Components pt. 1

- Lectures (~26)
  - Introduces concepts, slides & recordings available on canvas
  - In lecture polling.
- Sections (12)
  - Reiterates lecture content, lecture clarifications, assignment & exam preparation
- Programming Projects (~10)
  - Due every ~1 week
  - Applications of course content
  - Usually have everything you need for an assignment when it is released
- Check-in “Quizzes” (~12)
  - Unlimited attempt low-stake quizzes on Ed to make sure you are caught up with the material
  - Lowest two are dropped

Course Components pt. 2

- Final Project (1)
  - Due at the end of the semester
  - Can be done solo or in partners (tentatively)
  - Further Details TBD
- Exams (2)
  - Two in-person exams, two pages of notes allowed
  - Details TBD
- Textbook (0)
  - No Textbook, but using a C++ reference would probably be useful
  - https://cplusplus.com/
  - https://en.cppreference.com/w/

Course Grading (Tentative)

- Breakdown:
  - Homework assignments (56%)
  - Final Project (11%)
  - Exams (25%)
    - Midterm 10%
    - Final 15%
  - Check-in Quizzes (5%)
  - Course-wide participation

- Final Grade Calculations:
  - I would LOVE to give everyone an A+ if it is earned
  - Final grade cut-offs will be decided privately at the end of the Semester

Course Policies

- HW Late Policy
  - Checkins are due before Monday's lecture and cannot be turned in late
  - HW's cannot be turned in late, but they can be reopened
    - When you submit a check-in you can also say you want to re-open ONE homework assignment. That homework assignment will be re-opened till the next check-in is due.
    - You can re-open the same assignment multiple times
    - The final project can't be re-opened
  - End of the semester is the end
    (unless there is particularly special circumstances)

- Midterm Clobber Policy
  - Final is cumulative
  - If you do better on the "midterm section" of the final, your midterm grade can be overwritten.

Collaboration Policy Violation

- You will be caught:
  - Careful grading of all written homeworks by teaching staff
  - Measure of Software Similarity (MOSS): http://theory.stanford.edu/~aiken/moss/
  - Successfully used in several classes at Penn

- Zero on the assignment, (5%) deduction on final grade. F grade if caught twice.
  - First-time offenders will be reported to Office of Student Conduct with no exceptions. Possible suspension from school
  - Your friend from last semester who gave the code will have their grade retrospectively downgraded.

Collaboration Policy Violation

- Generative AI
  - I am skeptical of its usefulness for your learning and for your success in the course
  - Some articles on the topic:
    - https://www.aisnakeoil.com/p/chatgpt-is-a-bullshit-generator-but
    - https://www.aisnakeoil.com/p/gpt-4-and-professional-benchmarks
  - Not banned, but not recommended. 95% of the time I see students use it, they are using it wrong. Use your best judgement.
- You will not help your overall grade and happiness:
  - Quizzed individually during project demo, exams on project in finals
  - If you can't explain your code in OH, we can turn you away.
    - This is different than being confused on a bug or with C, this is ok
  - Personal lifelong satisfaction from completing the course

Course Infrastructure

- Course website
  - Schedule, syllabus, assignment specifications, materials …
- Docker
  - Coding environment for hw's, code is submitted to GradeScope
- GradeScope
  - Used for exam grades & HW submissions
- Poll Everywhere
  - Used for lecture polls
- Ed
  - Course discussion board and for check-in quizzes
- Canvas
  - Grades, lecture recordings & surveys

Course-wide participation

- 3% of the final grade

- Threshold to get full credit is VERY low
- Almost everyone gets full credit

- Get points from
  - Participating in poll everywhere
  - Attending recitation
  - Participating in Ed
  - Filling out the surveys for the course.

Getting Help

- Ed
  - Announcements will be made through here
  - Ask and answer questions
  - Sign up if you haven't already!

- Office Hours:
  - Can be found on calendar on front page of canvas page
  - Starts next week (hopefully)

- 1-on-1's:
  - Can schedule 1-on-1's with Travis
  - Should attend OH and use Ed when possible, but this is an option for when OH and Ed can't meet your needs

We Care

- We are still figuring things out, but we do care about you and your experience with the course
  - There is a pre-semester survey available on canvas now. Please fill this out honestly and we will do our best to incorporate people's answers
  - Please reach out to course staff if something comes up and you need help
- PLEASE DO NOT CHEAT OR VIOLATE ACADEMIC INTEGRITY
  - We know that things can be tough, but please reach out if you feel tempted. We want to help
  - Read more on academic integrity in the syllabus

pollev.com/tqm

- Any questions, comments or concerns so far?

Lecture Outline

- Introduction & Logistics
  - Course Overview
  - Assignments & Exams
  - Policies
- **C "Refresher"**
  - **Context in this course**
  - **memory**
  - **Pointers**
  - Arrays
  - Structs
  - The heap
  - const

Context of C in this course

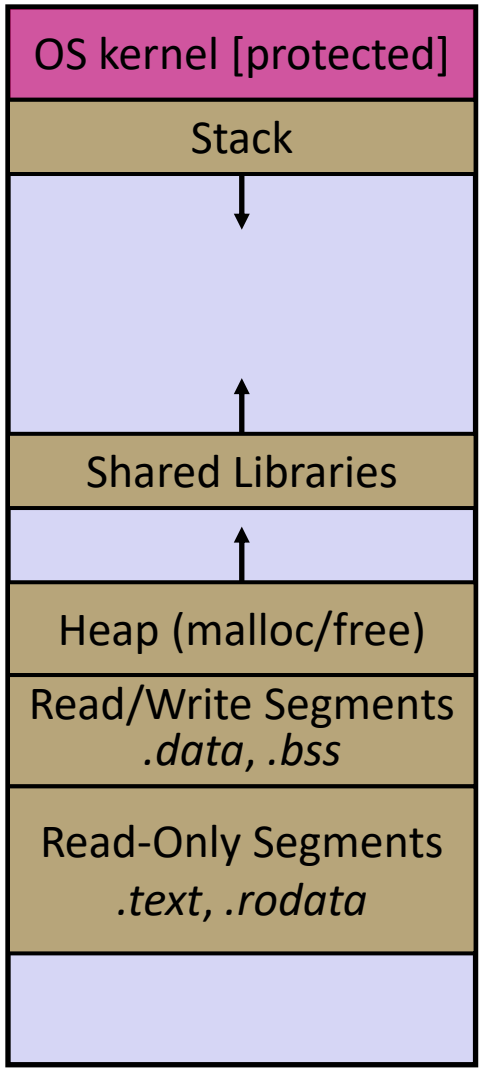
- You will be writing C++ in this course, not C
  - Most C is legal C++
  - For the first few assignments you will write C++ code that also mostly works as C code

- C++ is not C
  - C is the foundation for C++, but C++ is very different
  - We will refresh ourselves on this C foundation but quickly move on to C++

- Recitation tomorrow:
  - More C refresher (If you need it)
  - Not recorded, but slides will be posted

Memory

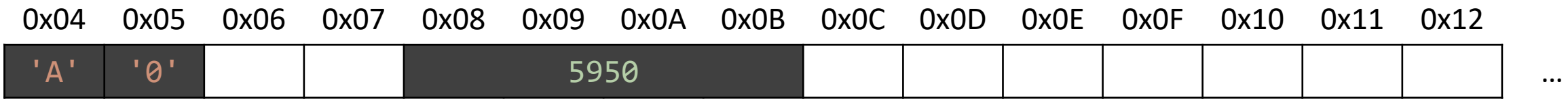
- Where all data, code, etc are stored for a program
- Broken up into several segments:
  - The stack
  - The heap
  - The kernel
  - Etc.
- Each “unit” of memory has an address



Memory as an array of bytes

- Everything in memory is made of bits and bytes
  - Bits: a single 1 or 0
  - Byte: 8 bits
- Memory is a giant array of bytes where everything* is stored
  - Each byte has its own address (“index”)
- Some types take up one byte, others more

```
int main() {
  char c = 'A';
  char other = '0';
  int x = 5950;
}
```



Pointers

POINTERS ARE EXTREMELY IMPORTANT IN C & C++

- Variables that store addresses
  - It stores the address to somewhere in memory
  - Must specify a type so the data at that address can be interpreted

- Generic definition: `type* name;` equivalent `type *name;`
  - Example: `int *ptr;`
    - Declares a variable that can contain an address
    - Trying to access that data at that address will treat the data there as an int

Memory is Huge

- Modern computers are called “64-bit”
  - Addresses are 64-bits (8-bytes)
  - There are 2^{64} possible memory locations, each location is 1-byte
  - 2^{64} is **18,446,744,073,709,551,616.**
  - Pointers must be 64-bits (8-bytes) to be able to hold any address on the computer.

Pointer Operators

- *Dereference* a pointer using the unary * operator
  - Access the memory referred to by a pointer
  - Can be used to read or write the memory at the address
  - Example:

```
int *ptr = ...; // Assume initialized
int a = *ptr; // read the value
*ptr = a + 2; // write the value
```

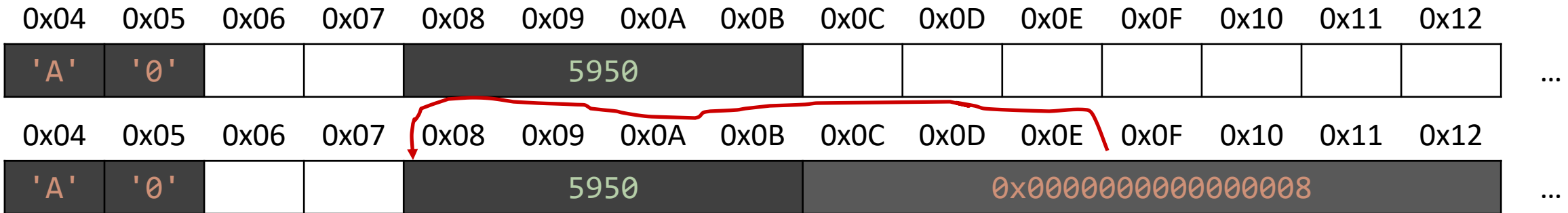
- Get the address of a variable with &
  - `&foo` gets the address of foo in memory
  - Example:

```
int a = 5950;
int *ptr = &a;
*ptr = 2; // 'a' now holds 2
```

Memory as an array of bytes

- Everything in memory is made of bits and bytes
  - Bits: a single 1 or 0
  - Byte: 8 bits
- Memory is a giant array of bytes where everything* is stored
  - Each byte has its own address (“index”)
- Some types take up one byte, others more

```
int main() {
  char c = 'A';
  char other = '0';
  int x = 5950;
  int* ptr = &x;
}
```



Pointer Example

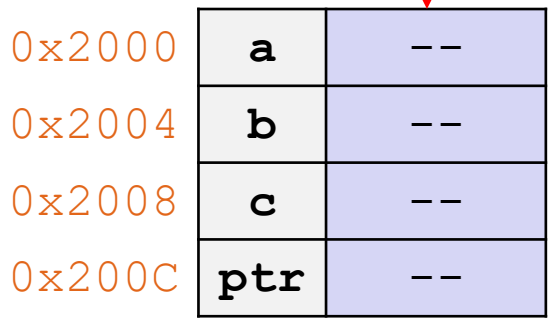
```
int main(int argc, char** argv) {
  int a, b, c;
  int* ptr;   // ptr is a pointer to an int

  a = 5;
  b = 3;
  ptr = &a;

  *ptr = 7;
  c = a + b;

  return 0;
}
```

Initial values are garbage



Pointer Example

```
int main(int argc, char** argv) {
  int a, b, c;
  int* ptr;   // ptr is a pointer to an int

  a = 5;
  b = 3;
  ptr = &a;

  *ptr = 7;
  c = a + b;

  return 0;
}
```

0x2000	a	5
0x2004	b	3
0x2008	c	--
0x200C	ptr	--

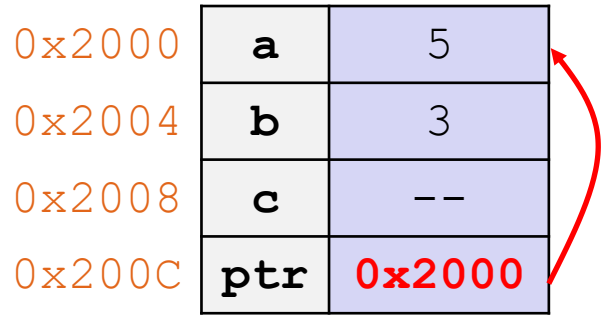
Pointer Example

```
int main(int argc, char** argv) {
  int a, b, c;
  int* ptr;   // ptr is a pointer to an int

  a = 5;
  b = 3;
  ptr = &a;

  *ptr = 7;
  c = a + b;

  return 0;
}
```



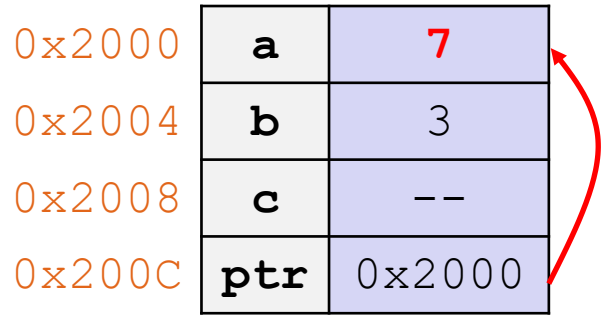
Pointer Example

```
int main(int argc, char** argv) {
  int a, b, c;
  int* ptr;   // ptr is a pointer to an int

  a = 5;
  b = 3;
  ptr = &a;

  *ptr = 7;
  c = a + b;

  return 0;
}
```



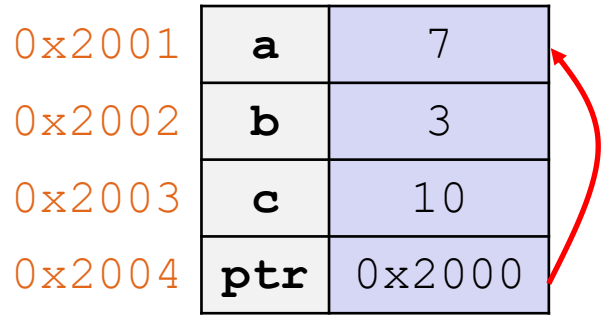
Pointer Example

```
int main(int argc, char** argv) {
  int a, b, c;
  int* ptr;   // ptr is a pointer to an int

  a = 5;
  b = 3;
  ptr = &a;

  *ptr = 7;
  c = a + b;

  return 0;
}
```



Poll Everywhere

pollev.com/tqm

- What does this print?
  - You can assume this compiles and the print syntax is correct.
  - Try drawing with boxes and arrows!

```
int main() {
  int curr = 6;
  int arc = 12;

  int* ptr = &curr;
  *ptr = 2;
  arc = 3;

  int* other = ptr;
  ptr = &arc;
  *ptr = *other
  *ptr += 3;

  // print curr and arc
  cout << curr << endl;
  cout << arc << endl;
}
```

Aside: NULL

- `NULL` is a memory location that is guaranteed to be invalid
  - In C on Linux, `NULL` is `0x0` and an attempt to dereference `NULL` *causes a segmentation fault*

Useful as an indicator of an uninitialized (or currently unused) pointer or allocation error

- It's better to cause a segfault than to allow the corruption of memory!

```
int main(int argc, char** argv) {
  int* p = NULL;
  *p = 1;  // causes a segmentation fault
  return EXIT_SUCCESS;
}
```

Lecture Outline

- Introduction & Logistics
  - Course Overview
  - Assignments & Exams
  - Policies
- **C "Refresher"**
  - Context in this course
  - memory
  - Pointers
  - **Arrays**
  - **Structs**
  - **The heap**
  - **const**

Arrays in C

- Definition: `type` `type name[size]`
  - Allocates `size*sizeof(type)` bytes of *contiguous* memory
  - Normal usage is a compile-time constant for `size`
    (*e.g.* `int scores[175];`)
  - Initially, array values are "garbage"

- Size of an array
  - **Not stored anywhere** – array does not know its own size!
  - The programmer will have to store the length in another variable or hard-code it in
  - No bounds checking!

Using Arrays

Optional when initializing

- Initialization: `type name[size] = {val0,…,valN};`
  - `{}` initialization can *only* be used at time of definition
  - If no size supplied, infers from length of array initializer

- Array name used as identifier for "collection of data"
  - `name[index]` specifies an element of the array and can be used as an assignment target or as a value in an expression
  - Array name (by itself) produces the address of the start of the array
    - Cannot be assigned to / changed

```
int primes[6] = {2, 3, 5, 6, 11, 13};
primes[3] = 7;
primes[100] = 0;  // memory smash!
```

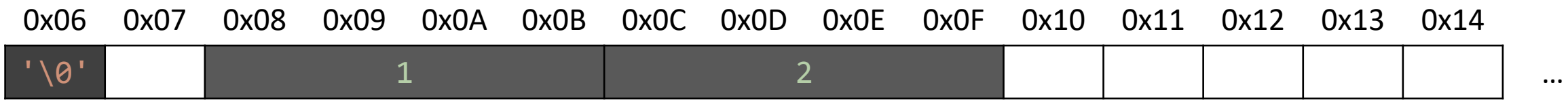
No IndexOutOfBounds
Hope for segfault

Arrays in C

- Here is a memory diagram example:

```
int main() {
  char c = '\0';

  int arr[2] = {1, 2};
}
```



Pointers as C arrays

- Pointers can be set to an array

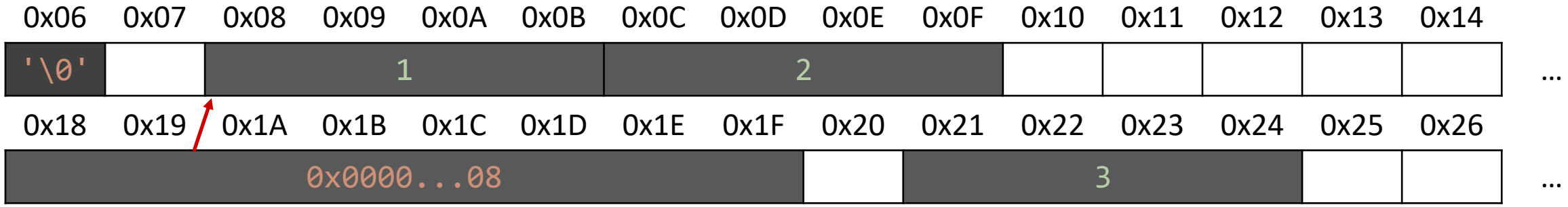
- Pointers can always be indexed into like an array
  - Pointers don't always have to point to the beginning of an array!

```
int main() {
  char c = '\0';

  int arr[2] = {1, 2};

  int* ptr = arr;

  int x = ptr[1] + 1;
}
```



Poll Everywhere

pollev.com/tqm

- What is the final value of core after this code is run? Where is ptr pointing to after this code is run?
  - Hint: Draw it out!

```
void foo() {
  int core[3] = {5940, 5930, 5960};

  core[1] += 20;

  int* ptr = &(core[1]);

  ptr[0] -= 900;

  ptr[1] = 5000;

  core[2] += 20;

  // STOP HERE
}
```

Strings in C

- Strings in C are just arrays of characters with a special character at the end to mark the end of the string: `'\0'`
  - Called the "null terminator" character

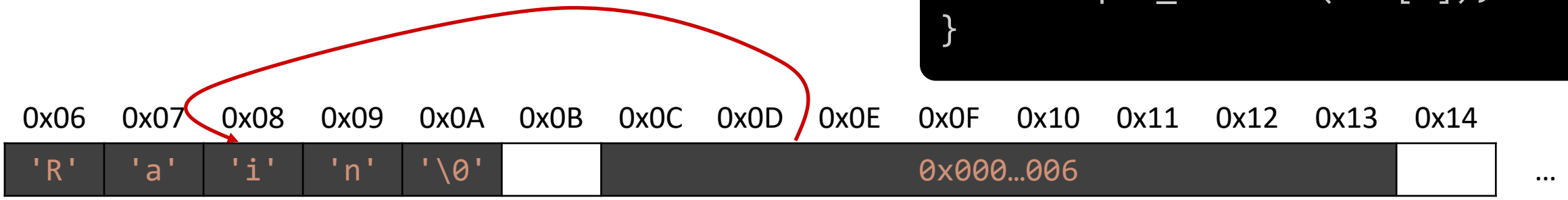
- C-strings are often referred to with a **`char[]`** or a **`char*`**

- Example:
  - `print(str)  // Rain`
  - `print(ptr_str)  // in`

```
int main() {
  char c = '\0';

  char str[5] = "Rain";

  char* ptr_str = &(str[2]);
}
```



Poll Everywhere

pollev.com/tqm

- Finish this code:
  - This function takes in a string and returns the length of the string.
  - Do not call any other function
  - size_t is just an unsigned integer type
  - Remember to index into the pointer like an array!
  - What marks the end of a string?
  - You don't have to use a while loop, but I think it makes the most sense.

```
size_t strlen(char* str) {
  size_t length = 0;

  while (          ) {




  }

  return length;
}
```

The Heap

- For most program memory we care about, things are stored either in the **heap** or **stack**

- In C we allocated with `malloc()` and deallocated with `free()`

- In C++ we will use `new` and `delete`.
  - New still gives us a pointer to the heap
  - We must deallocate the pointer with delete when we are done with the pointer.

```
int main() {
  int* x = new int;

  *x = 3;

  // prints *x which is 3
  cout << *x << endl;

  delete x;
}
```

The Heap

- In C++ we will use `new` and `delete`.
  - New still gives us a pointer to the heap
  - Can use new to allocate an array!
  - Will need this to allocate an array of characters (so a C-style string) in the first homework assignment.
  - We deallocate arrays with `delete[]`

```
int main() {

  int* arr = new int[2];

  arr[0] = 5930;
  arr[1] = 5950;

  delete[] arr;
}
```

- Will talk more about what the heap is and why it is important next lecture. This should be enough for HW00 though.

Structured Data

- A `struct` is a C and C++ datatype that contains a set of fields
  - Similar to a Java class, but with no methods or constructors
  - Useful for defining new structured types of data
  - Acts similarly to primitive variables
- Generic declaration in C++:

```
struct Point {
  float x;
  float y;
};

Point pt;
Point origin = {0.0f, 0.0f};
pt = origin; // pt now contains 0.0f, 0.0f
```

Default values are still garbage!

<- Initializer List

Can be assigned into, used as parameters, etc.

Structured Data: copied not referenced

- A `struct` is a C and C++ datatype that contains a set of fields
  - Similar to a Java class, but with no methods or constructors
  - Useful for defining new structured types of data
  - Acts similarly to primitive variables
    - When we assign a

```
Point pt;
Point origin = {0.0f, 0.0f};
pt = origin; // pt now contains 0.0f, 0.0f

origin.first = 1.0f;

print(origin.first);
print(pt.first);
```

Accessing struct Fields

- Use "." to refer to a field in a struct
- Use "->" to refer to a field from a struct pointer
  - Dereferences pointer first, then accesses field

```
struct Point {
  float x, y;
};

int main(int argc, char** argv) {
  Point p1 = {0.0, 0.0};
  Point* p1_ptr = &p1;

  p1.x = 1.0;
  p1_ptr->y = 2.0;  // equivalent to (*p1_ptr).y = 2.0;
  return 0;
}
```

Const

- `const` is a keyword in C and C++ that means that a variable cannot be modified. It is "constant"

- If a struct is `const` in C or C++, then its members are also `const`.

```
int main() {
  const int x = 3;
  int y = 5;

  x += 1;  // ILLEGAL

  y += 1;

  const Point p = {0.0, 0.0};

  p.first = 1.0; // ILLEGAL
}
```

That's all for now!

- If we got through all this, you should have everything you need for the first homework assignment from this lecture and recitation
- We are going a little fast because I expect you have already seen all or most of this in CIT 5930
- When we get to new material it won't be as fast
- Releasing tomorrow:
  - HW00
  - Pre-semester Survey
  - Check-in00