Introductions, C++ start

Computer Systems Programming, Spring 2024

Instructor: Travis McGaha

TAs:

CV Kunjeti
Felix Sun
Heyi Liu
Kevin Bernat
Lang Qin
Sean Chuang
Serena Chen
Yuna Shao

pollev.com/tqm

- How are you?

Lecture Outline

- **Introduction & Logistics**
  - **Course Overview**
  - **Assignments & Exams**
  - **Policies**
- C++ Intro
  - Helloworld, iostream, strings
  - vector
  - references
  - map & unordered_map (start)

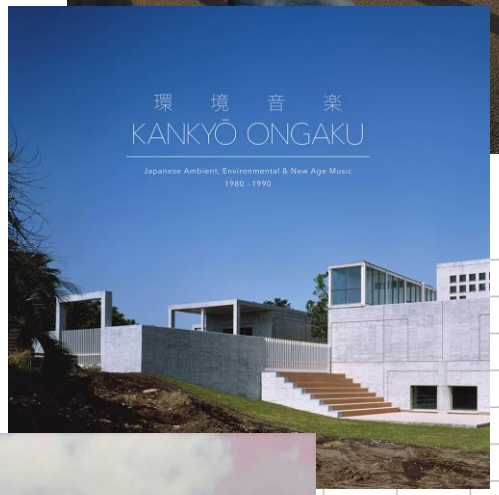
Instructor: Travis McGaha

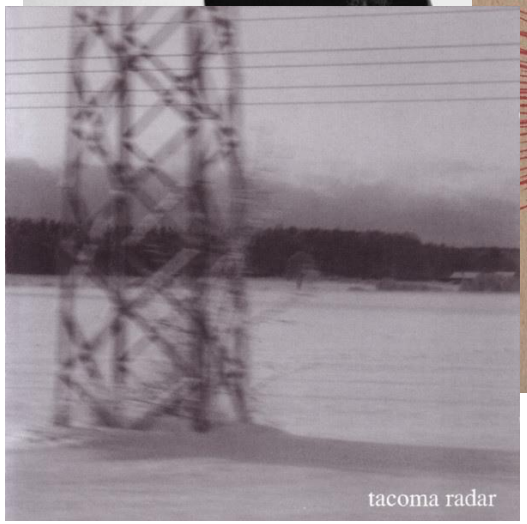
- UPenn CIS faculty member since… August 2021
  - Currently my sixth semester at UPenn
  - Third Semester with CIT 5950… and I am still trying new stuff
  - Lots of the same content, but in a different order, new assignments and more of a focus on C++

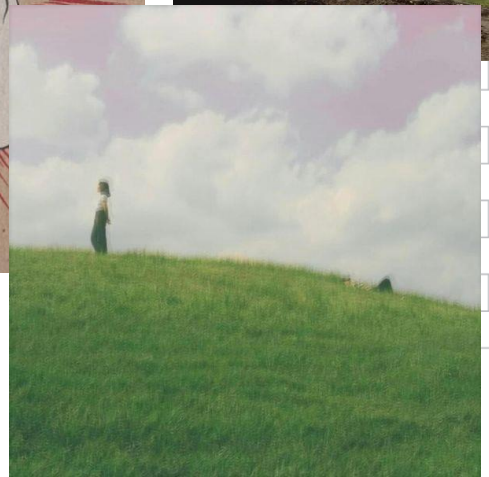
- Education: University of Washington, Seattle
  - Masters in Computer Science in March 2021
  - Bachelors in Computer Engineering in June 2019
  - Instructed a course that covers very similar material

Instructor: Travis McGaha

- I like most music







Instructor: Travis McGaha

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

Instructor: Travis McGaha

- I like video games

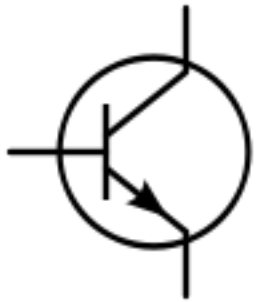
Instructor: Travis McGaha

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

Instructor: Travis McGaha

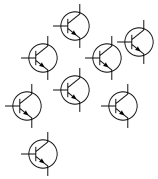
- I care a lot about your actual learning and that you have a good experience with the course
- I am a human being and I know that you are one too. If you are facing difficulties, please let me know and we can try and work something out.
- More on my personal website: https://www.cis.upenn.edu/~tqmcgaha/

Course Overview

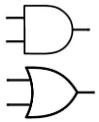


Course Overview

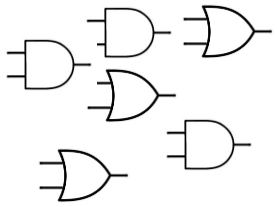
Course Overview



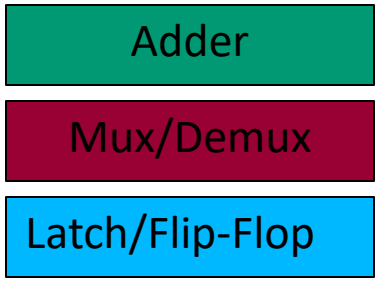
Course Overview



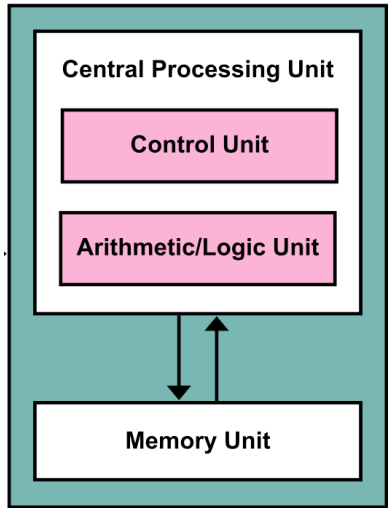
Course Overview



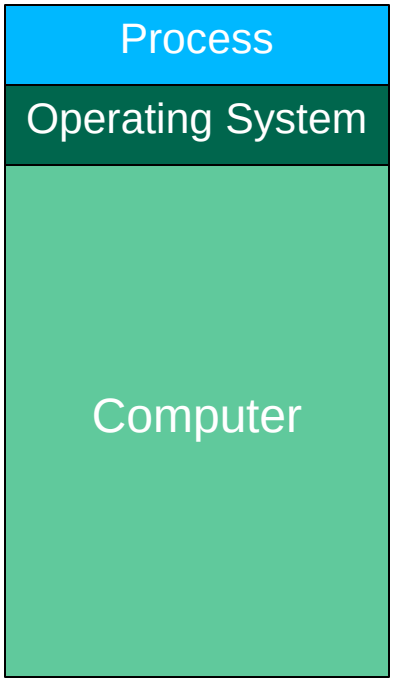
Course Overview



Course Overview



Course Overview



Course Overview

Wittgenstein's Ladder

- "My propositions serve as elucidations in the following way: anyone who understands me eventually recognizes them as nonsensical, when he has used them—as steps—to climb beyond them. (He must, so to speak, throw away the ladder after he has climbed up it.)

  He must transcend these propositions, and then he will see the world aright."
  - Ludwig Wittgenstein (Tractatus Logico-Philosophicus)

"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)

pollev.com/tqm

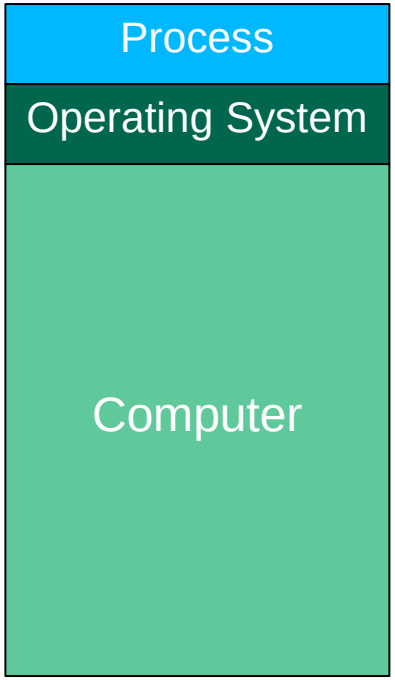
- What color is the sky?

pollev.com/tqm

- What color is grass?

We lied to you (but in a good way)

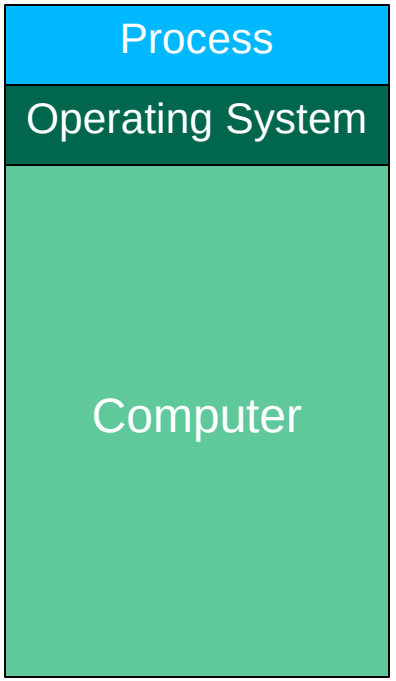
- Is the LC4 model for a computer true? **Eh……. no**
- Is it a useful model? **Yes**



We lied to you (but in a good way)

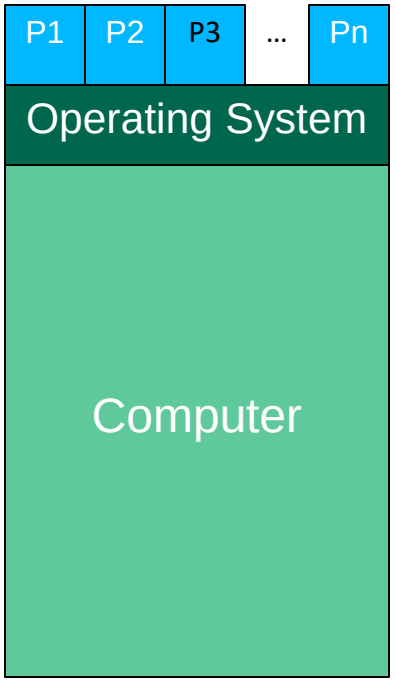
- Is memory one giant array of bytes? **Eh……. no**
- Is this a useful model? **Yes**

Course Overview



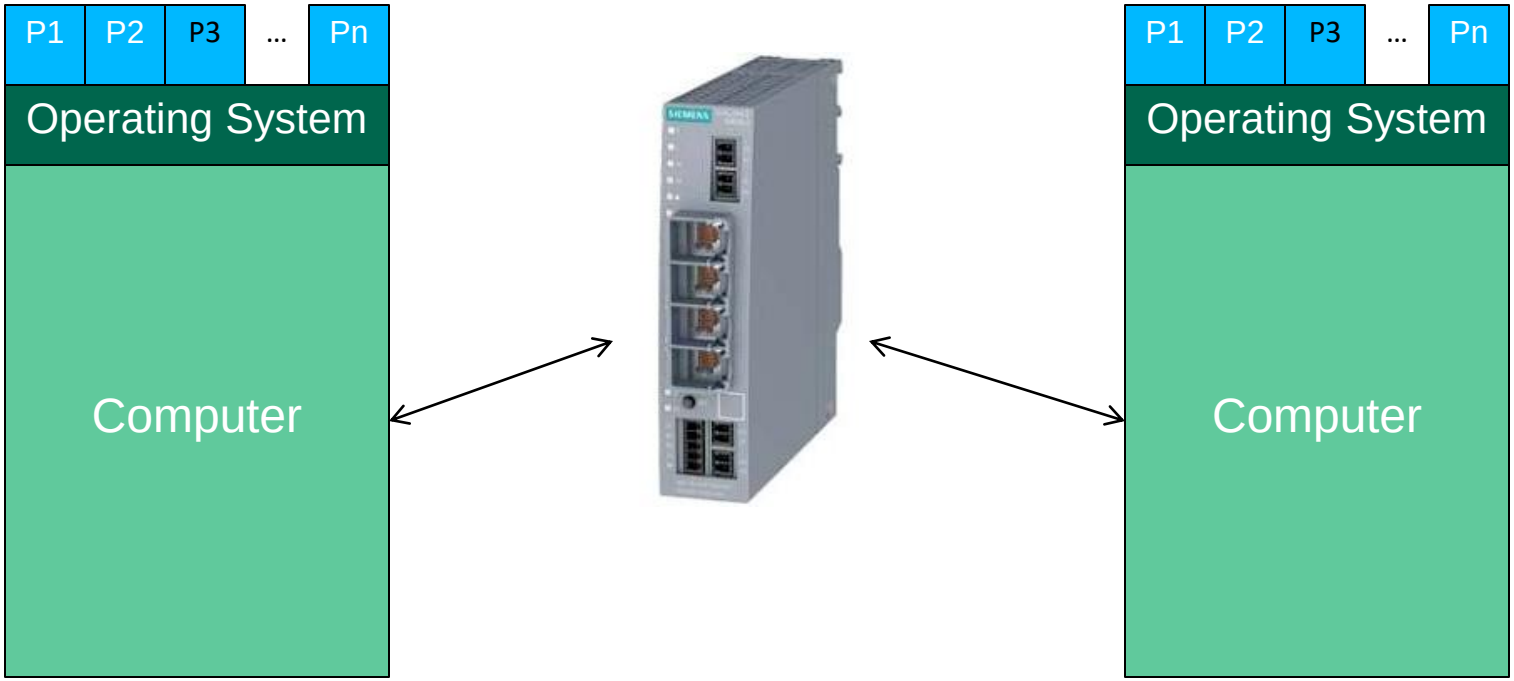
OS does A LOT more than just printing, reading input, video display, and timer

Course Overview



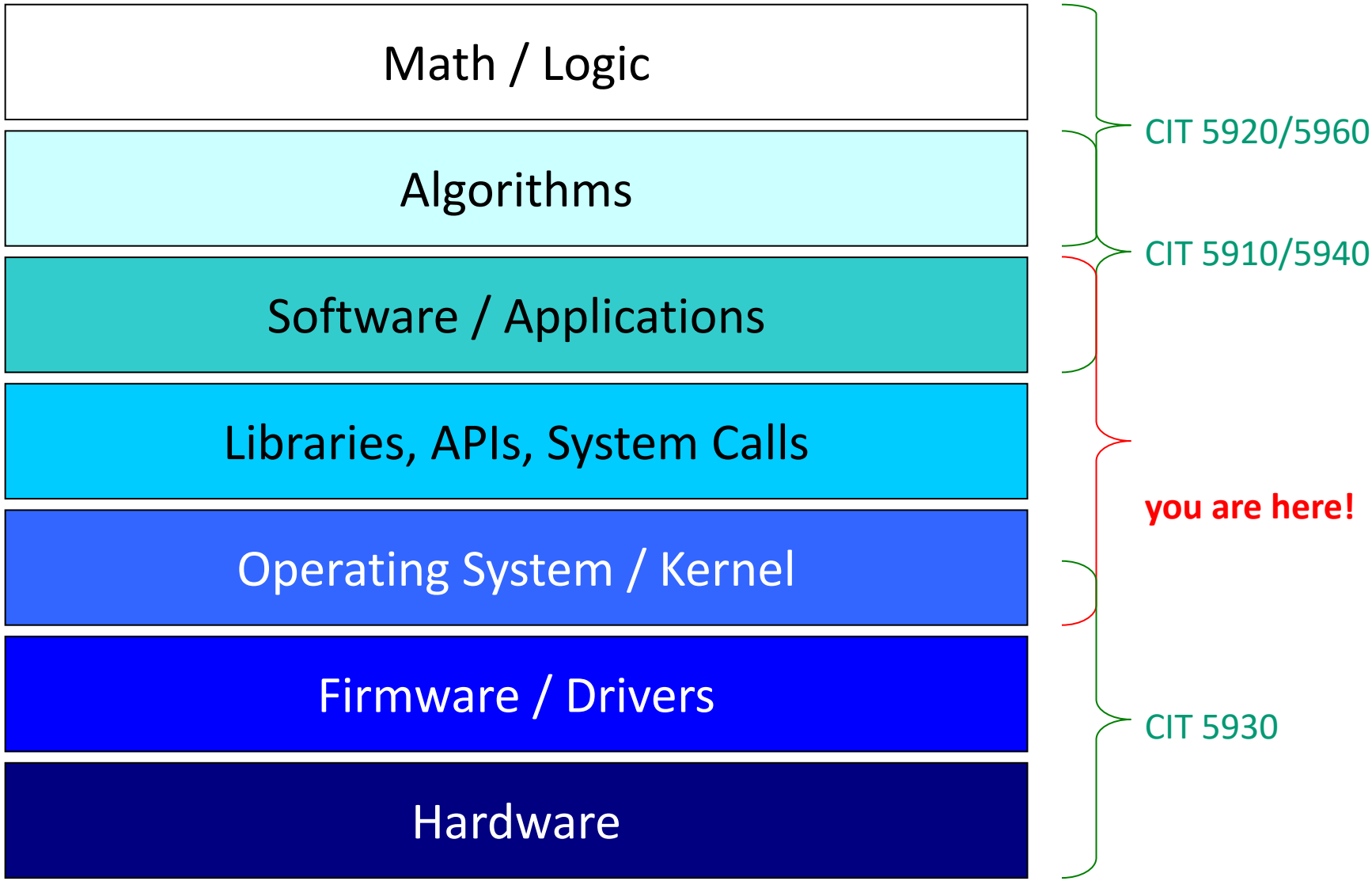
THERE IS A LOT GOING ON TO SUPPORT THIS

Course Overview



Course Overview

Course Overview



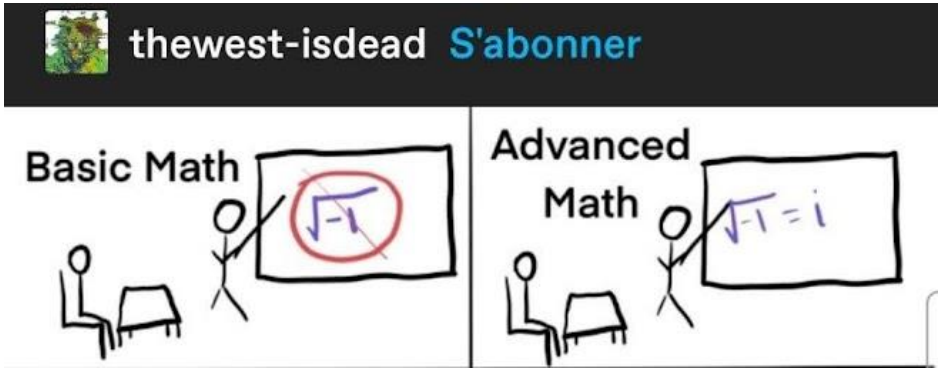
I'm going to lie to you (but in a good way)

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

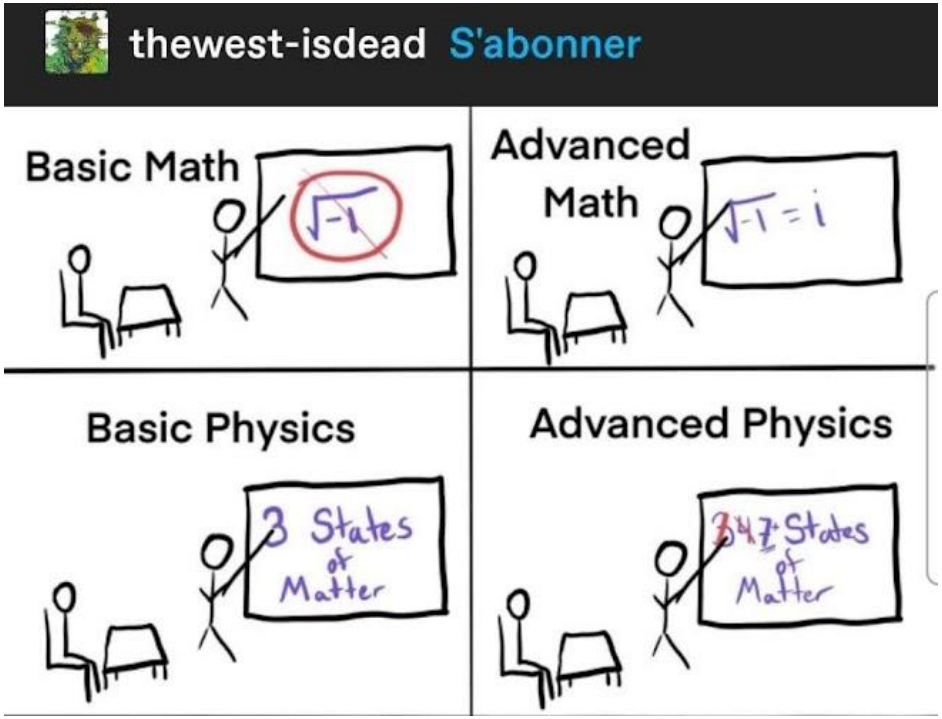
- "If it were necessary for us to understand how every component of our daily lives works in order to function - we simply would not."
  - AnRel (UNHINGED: A Guide to Revolution for Nerds & Skeptics)

- 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.

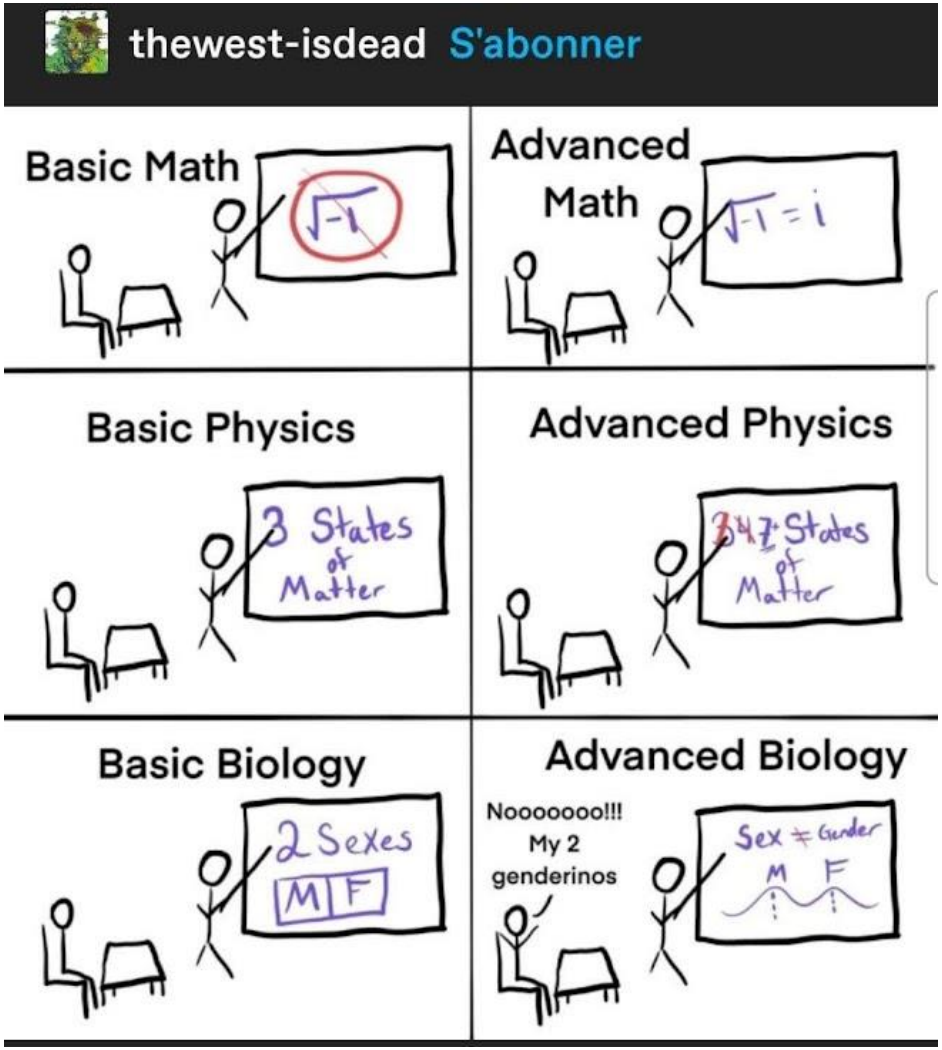
This goes beyond this course



This goes beyond this course



This goes beyond this course



This goes beyond this course

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 cover some of these lightly with the beginning of the semester ☺

CIT 5950 Learning Objectives

- To leave the class with a better understanding of:
  - 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
  - C++

- 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 (5)
  - Due every ~2 weeks
  - Applications of course content
- Check-in “Quizzes” (~10)
  - Unlimited attempt low-stake quizzes on canvas 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:
  - Homeworks (55%)
  - Final Project (15%)
  - Exams (25%)
    - Midterm 10%
    - Final 15%
  - Check-in Quizzes (5%)

- 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
  - Late days given on request (Request usually granted)
  - No cap on the number of late days per assignment
  - More than 3 on an assignment requires approval from Travis
  - HW0 & HW1 need to be finished before we release the final project though
  - 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, zero for class participation (3%). 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. 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 PennOS

Course Infrastructure

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

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
- 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++ Intro**
  - **Helloworld, iostream, strings**
  - vector
  - references
  - map & unordered_map (start)

Hello World in C++

helloworld.cpp

```
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS

using namespace std;

int main() {
  cout << "Hello, World!" << endl;
  return EXIT_SUCCESS;
}
```

- Looks simple enough…
  - Compilation command if you want to compile this yourself:

```
g++-12 -Wall -g -std=c++23 -o helloworld helloworld.cpp
```

  - Let's walk through the program step-by-step to highlight some differences

Hello World in C++

helloworld.cpp

```
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS

using namespace std;

int main() {
  cout << "Hello, World!" << endl;
  return EXIT_SUCCESS;
}
```

- `iostream` is part of the ***C++*** standard library
  - Note: you don't write "`.h`" when you include C++ standard library headers
    - But you *do* for local headers (*e.g.* `#include "Deque.hpp"`)
  - `iostream` declares stream *object* instances
    - *e.g.* `cin`, `cout`, `cerr`

Hello World in C++

helloworld.cpp

```
#include <iostream>   // for cout, endl
#include <cstdlib>    // for EXIT_SUCCESS

using namespace std;

int main() {
  cout << "Hello, World!" << endl;
  return EXIT_SUCCESS;
}
```

- `cstdlib` is the ***C*** standard library's `stdlib.h`
  - Nearly all C standard library functions are available to you
    - For C header `math.h`, you should `#include <cmath>`
  - We include it here for `EXIT_SUCCESS`

Hello World in C++

helloworld.cpp

```
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS

using namespace std;

int main() {
  cout << "Hello, World!" << endl;
  return EXIT_SUCCESS;
}
```

- `using namespace std;`
  - It is there because I said so (can't use it in header files tho)
  - We include it here so that I can say `cout` instead of `std::cout`

```
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS

int main() {
  std::cout << "Hello, World!" << std:: endl;
  return EXIT_SUCCESS;
}
```

Hello World in C++

helloworld.cpp

```
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS

using namespace std;

int main() {
  cout << "Hello, World!" << endl;
  return EXIT_SUCCESS;
}
```

- "`cout`" is an object instance declared by `iostream`, C++'s name for `stdout`
  - `std::cout` is an object of class `ostream`
    - http://www.cplusplus.com/reference/ostream/ostream/
  - Used to format and write output to the console
  - We use << to send data to cout to get printed

Hello World in C++

helloworld.cpp

```
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS

using namespace std;

int main() {
  cout << "Hello, World!" << endl;
  return EXIT_SUCCESS;
}
```

- `endl` is a pointer to a "manipulator" function
  - This manipulator function writes newline (`'\n'`) to the `ostream` it is invoked on and then flushes the `ostream`'s buffer
  - This *enforces* that something is printed to the console at this point

Let's do a slightly more complex program

Greeting C++

greeting.cpp

```
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS
#include <string>      // for string

using namespace std;

int main() {
  string expected {"Travis"};
  // ...
}
```

- `string` is part of the ***C++*** standard library
  - We still have to #include it
    - No more char* !
  - When we initialize any variable in C++, we use the {}

Greeting C++

greeting.cpp

```
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS
#include <string>      // for string

using namespace std;

int main() {
  string expected {"Travis"};
  cout << "Who are you?" << endl;
  // ...
}
```

Greeting C++

greeting.cpp

```
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS
#include <string>      // for string

using namespace std;

int main() {
  string expected {"Travis"};
  cout << "Who are you?" << endl;

  string input {};
  // ...
}
```

- Declares an empty string ("")
  - You ~~should~~ **must** always initialize a variable with {} even if empty

Greeting C++

greeting.cpp

```
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS
#include <string>      // for string

using namespace std;

int main() {
  string expected {"Travis"};
  cout << "Who are you?" << endl;

  string input {};
  cin >> input;

  // ...
}
```

- Reads from stdin (terminal input) into "`input`"
  - This works for reading in numbers too

Greeting C++

greeting.cpp

```
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS
#include <string>      // for string

using namespace std;

int main() {
  string expected {"Travis"};
  cout << "Who are you?" << endl;

  string input {};
  cin >> input;

  // ...
}
```

- Reads from stdin (terminal input) into "`input`"
  - This works for reading in numbers too

Greeting C++

greeting.cpp

```
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS
#include <string>      // for string

using namespace std;

int main() {
  string expected {"Travis"};
  cout << "Who are you?" << endl;

  string input {};
  cin >> input;

  if (input == expected) {

  }
}
```

- Can use == to compare strings

Greeting C++

greeting.cpp

```
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS
#include <string>      // for string

using namespace std;

int main() {
  string expected {"Travis"};
  cout << "Who are you?" << endl;

  string input {};
  cin >> input;

  if (input == expected) {
    cout << "Hello " << input << "!" << endl;
  }
}
```

- Can chain << repeatedly
  - Would also work for printing a lot of types
    - (including integer and floating point types)

Greeting C++

- Print to `cerr` when there is an error.
- Also known as stderr
- This case is debatably an error.

greeting.cpp

```
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS
#include <string>      // for string

using namespace std;

int main() {
  string expected {"Travis"};
  cout << "Who are you?" << endl;

  string input {};
  cin >> input;

  if (input == expected) {
    cout << "Hello " << input << "!" << endl;
  } else {
    cerr << "Who the hell are you????" << endl;
  }
  return EXIT_SUCCESS;
}
```

pollev.com/tqm

- What does this code print?

```
#include <iostream>
#include <cstdlib>

using namespace std;

void modify_int(int x) {
  x = 5;
}

int main() {
  int num = 3;
  modify_int(num);
  cout << num << endl;
  return EXIT_SUCCESS;
}
```

C and C++

helloworld3.cc

```
#include <cstdio>      // for printf
#include <cstdlib>     // for EXIT_SUCCESS

int main(int argc, char* argv[]) {
  printf("Hello from C!\n");
  return EXIT_SUCCESS;
}
```

- C is (roughly) a subset of C++
  - You can still use **`printf`** – but **bad style** in ordinary C++ code
    - E.g. Use `std::cerr` instead of `fprintf(stderr, …)`
  - Can mix C and C++ idioms if needed to work with existing code, but avoid mixing if you can
    - **Use C++**

C++ Documentation

- As said, there is a LOT to C++
  - There are a lot of functions, objects, features, etc
  - We will NOT have time to talk about them all

- We highly recommend you make use of a C++ reference
  - cplusplus.com
    - Most find this one easier to read
    - Probably has the information you need
  - cppreference.com
    - Much more detailed (in Travis' opinion)
    - Is in various languages (scroll to the bottom)

Lecture outline

- Introduction & Logistics
  - Course Overview
  - Assignments & Exams
  - Policies
- **C++ Intro**
  - Helloworld, iostream, strings
  - **vector**
  - references
  - map & unordered_map (start)

STL Containers ☺

- A container is an object that stores (in memory) a collection of other objects (elements)
  - Implemented as class templates, so hugely flexible
  - More info in C++ Primer §9.2, 11.2

- Several different classes of container
  - Sequence containers (vector, deque, list, ...)
  - Associative containers (set, map, multiset, multimap, bitset, ...)
  - Differ in algorithmic cost and supported operations

vector

C++ equivalent of ArrayList

- A generic, dynamically resizable array
  - https://cplusplus.com/reference/vector/vector/
  - Elements are store in contiguous memory locations
    - Can index into it like an array
    - Random access is O(1) time
  - Adding/removing from the end is cheap (amortized constant time)
  - Inserting/deleting from the middle or start is expensive (linear time)

- Most common member function: **`push_back()`**
  - Adds an element to the end of the vector

Vector example

```
#include <iostream>
#include <vector>

using namespace std;

int main(int argc, char* argv[]) {
  vector<int> vec {6, 5, 4};
  vec.push_back(3);
  vec.push_back(2);
  vec.push_back(1);

  cout << "vec.at(0)" << endl << vec.at(0) << endl;
  cout << "vec.at(1)" << endl << vec.at(1) << endl;

  // iterates through all elements
  for (size_t i = 0U; i < vec.size(); ++i) {
     cout << vec.at(i) << endl;
  }

  return EXIT_SUCCESS;
}
```

Most containers are in a module of the same name

Constructs a vector with three initial elements

Add three integers to the vector

Print all the values in the array

Vector iterator

```
int main(int argc, char* argv[]) {
  vector<int> vec {6, 5, 4};

  vector<int>::iterator it = vec.begin();
  it = vec.insert(it, 3);
  ++it;
  it = vec.insert(it, 1);
  it = vec.end();
  it = vec.insert(it, 2);


  cout << "Iterating:" << endl;
  for (it = vec.begin(); it < vec.end(); ++it) {
     cout << *it << endl;
  }

  return EXIT_SUCCESS;
}
```

Can get an iterator to the beginning of the vector

Insert 3 // {3, 6, 5, 4}

Advances iterator to index 1

Inserts 1 to index 1 {3, 1, 6, 5, 4}

Sets iterator to the end

Same as push_back(2);

Accesses the current element of the iterator

range for loop

- Syntactic sugar similar to Java's foreach

```
for (declaration : expression) {
  statements
}
```

  - *declaration* defines the loop variable
  - *expression* is an object representing a sequence
    - Strings, and most STL containers work with this

```
string str{"hello"};
// prints out each character
for (char c : str) {
  cout << c << endl;
}
```

range for loop vector example

- If you need to iterate over every element in a sequence, you should use a range for loop.
  - Why? It is harder to mess it up that way

```
int main(int argc, char* argv[]) {
  vector<int> vec {6, 5, 4};
  vec.push_back(3);
  vec.push_back(2);
  vec.push_back(1);

  // iterates through all elements
  for (int element : vec) {
     cout << element << endl;
  }

  return EXIT_SUCCESS;
}
```

Other vector functions

- pop_back()
  - Removes the last element of the vector
- empty()
  - Returns true if the vector is empty
- clear()
  - Removes all elements currently in the vector
- erase(iterator position)
  - Erases from the element at the specified position
- A bunch more:
  - https://www.cplusplus.com/reference/vector/vector/

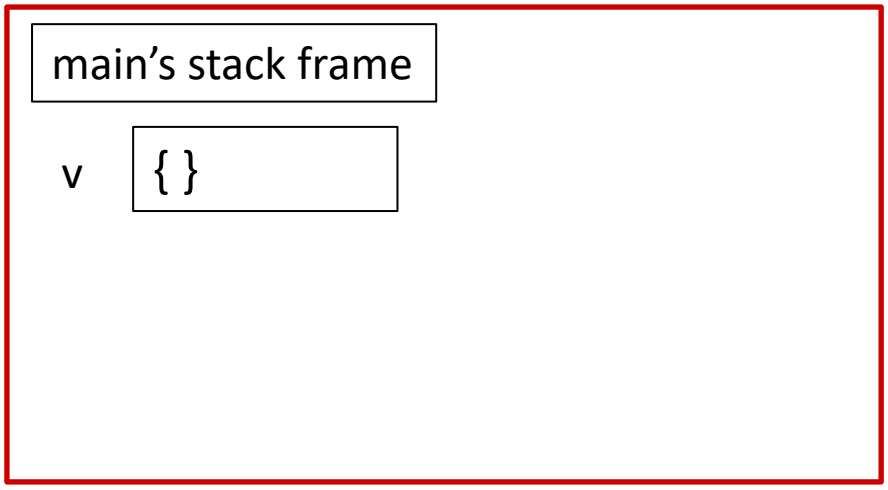
Poll Everywhere

pollev.com/tqm

- What is the final value of v by the end of the main() function?

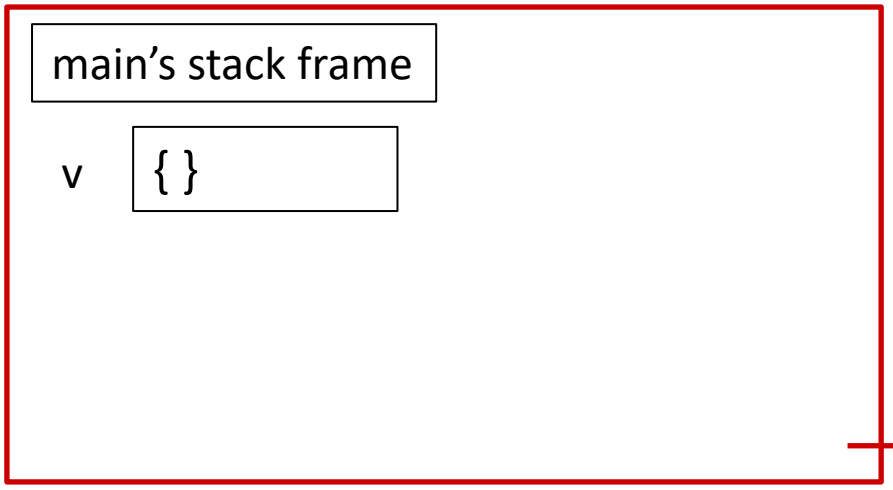
```
#include <vector>
#include <iostream>

using namespace std;

void populate_vec(vector<int> v) {
  v.push_back(5950);
}

int main() {
  vector<int> v {};
  populate_vec(v);
  cout << v.size() << endl;
  for (size_t i = 0U; i < v.size(); ++i) {
     cout << v.at(i) << endl;
  }
  return EXIT_SUCCESS;
}
```

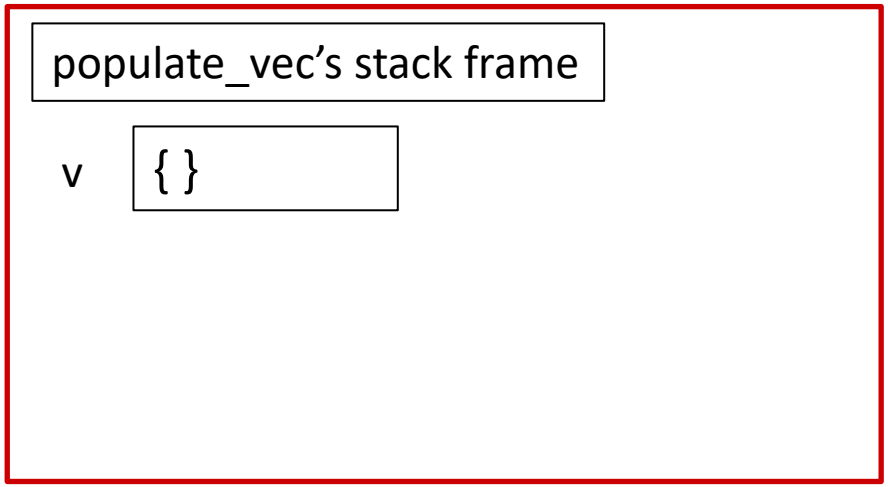
Visualization




```
#include <vector>
#include <iostream>

using namespace std;

void populate_vec(vector<int> v) {
  v.push_back(5950);
}

int main() {
  vector<int> v {};
  populate_vec(v);
  cout << v.size() << endl;
  // loop removed for space
  return EXIT_SUCCESS;
}
```

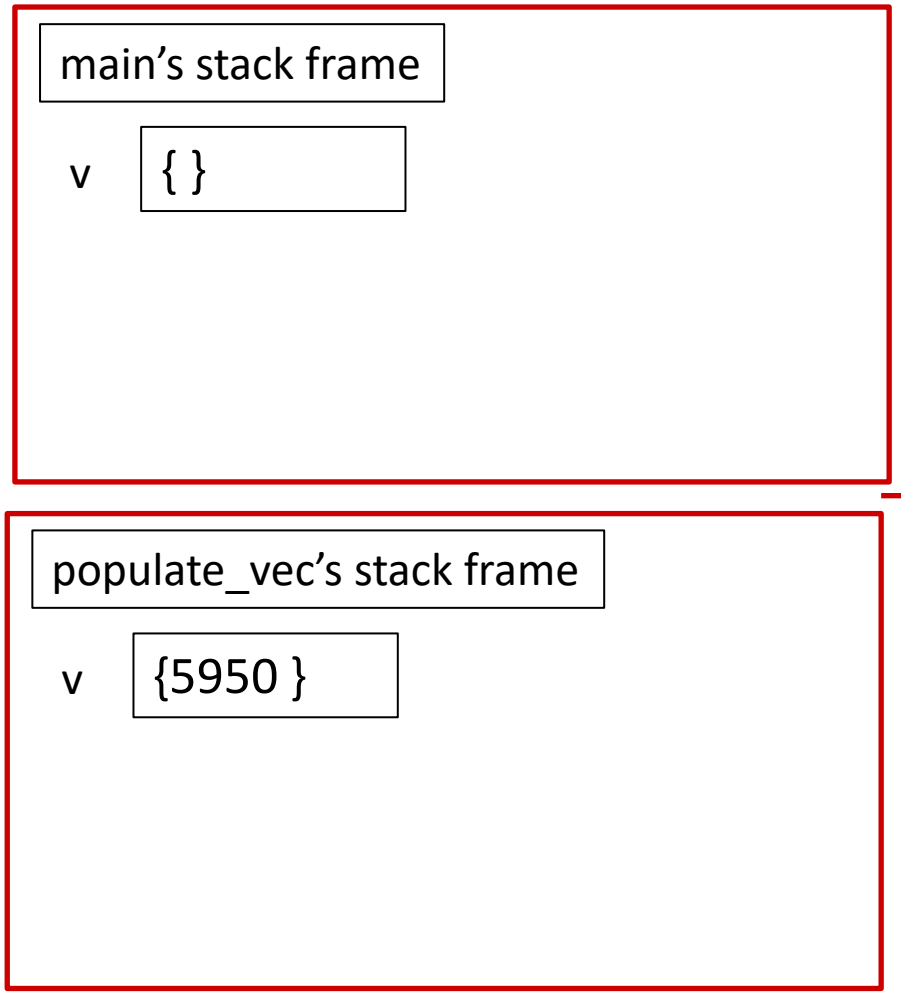
Visualization





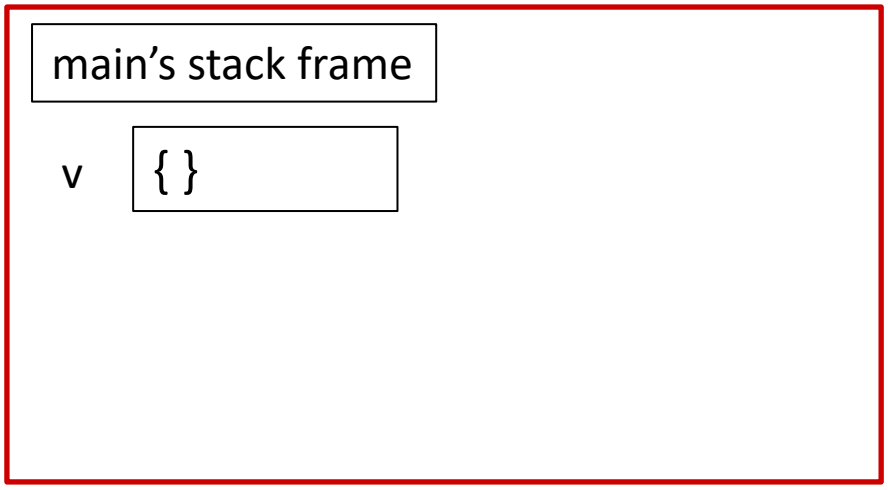

```
#include <vector>
#include <iostream>

using namespace std;

void populate_vec(vector<int> v) {
  v.push_back(5950);
}

int main() {
  vector<int> v {};
  populate_vec(v);
  cout << v.size() << endl;
  // loop removed for space
  return EXIT_SUCCESS;
}
```

Visualization




```
#include <vector>
#include <iostream>

using namespace std;

void populate_vec(vector<int> v) {
  v.push_back(5950);
}

int main() {
  vector<int> v {};
  populate_vec(v);
  cout << v.size() << endl;
  // loop removed for space
  return EXIT_SUCCESS;
}
```

Visualization




```
#include <vector>
#include <iostream>

using namespace std;

void populate_vec(vector<int> v) {
  v.push_back(5950);
}

int main() {
  vector<int> v {};
  populate_vec(v);
  cout << v.size() << endl;
  // loop removed for space
  return EXIT_SUCCESS;
}
```

Lecture outline

- Introduction & Logistics
  - Course Overview
  - Assignments & Exams
  - Policies
- **C++ Intro**
  - Helloworld, iostream, strings
  - vector
  - **references**
  - map & unordered_map (start)

References

Note: Arrow points to *next* instruction.

- A **reference** is an alias for another variable
  - *Alias*: another name that is bound to the aliased variable
    - Mutating a reference ***is*** mutating the aliased variable
  - Introduced in C++ as part of the language

```
int main(int argc, char** argv) {
  int x = 5, y = 10;
  int& z = x;

  z += 1;
  x += 1;

  z  = y;
  z += 1;

  return EXIT_SUCCESS;
}
```

When we use '&' in a type declaration, it is a reference.

x	5

y	10

References

Note: Arrow points to *next* instruction.

- A **reference** is an alias for another variable
  - *Alias*: another name that is bound to the aliased variable
    - Mutating a reference ***is*** mutating the aliased variable
  - Introduced in C++ as part of the language

```
int main(int argc, char** argv) {
  int x = 5, y = 10;
  int& z = x;   // binds the name "z" to x

  z += 1;
  x += 1;

  z  = y;
  z += 1;

  return EXIT_SUCCESS;
}
```


x, z	5

y	10

References

Note: Arrow points to *next* instruction.

- A **reference** is an alias for another variable
  - *Alias*: another name that is bound to the aliased variable
    - Mutating a reference ***is*** mutating the aliased variable
  - Introduced in C++ as part of the language

```
int main(int argc, char** argv) {
  int x = 5, y = 10;
  int& z = x;   // binds the name "z" to x

  z += 1;   // sets z (and x) to 6
  x += 1;

  z  = y;
  z += 1;

  return EXIT_SUCCESS;
}
```

x, z	6

y	10

References

Note: Arrow points to *next* instruction.

- A **reference** is an alias for another variable
  - *Alias*: another name that is bound to the aliased variable
    - Mutating a reference ***is*** mutating the aliased variable
  - Introduced in C++ as part of the language

```
int main(int argc, char** argv) {
  int x = 5, y = 10;
  int& z = x;   // binds the name "z" to x

  z += 1;   // sets z (and x) to 6
  x += 1;   // sets x (and z) to 7

  z  = y;   // Normal assignment
  z += 1;

  return EXIT_SUCCESS;
}
```

x, z	7

y	10

References

Note: Arrow points to *next* instruction.

- A **reference** is an alias for another variable
  - *Alias*: another name that is bound to the aliased variable
    - Mutating a reference ***is*** mutating the aliased variable
  - Introduced in C++ as part of the language

```
int main(int argc, char** argv) {
  int x = 5, y = 10;
  int& z = x;   // binds the name "z" to x

  z += 1;   // sets z (and x) to 6
  x += 1;   // sets x (and z) to 7

  z  = y;   // sets z (and x) to the value of y
  z += 1;

  return EXIT_SUCCESS;
}
```

x, z	10

y	10

References

Note: Arrow points to *next* instruction.

- A **reference** is an alias for another variable
  - *Alias*: another name that is bound to the aliased variable
    - Mutating a reference ***is*** mutating the aliased variable
  - Introduced in C++ as part of the language

```
int main(int argc, char** argv) {
  int x = 5, y = 10;
  int& z = x;   // binds the name "z" to x

  z += 1;   // sets z (and x) to 6
  x += 1;   // sets x (and z) to 7

  z  = y;   // sets z (and x) to the value of y
  z += 1;   // sets z (and x) to 11

  return EXIT_SUCCESS;
}
```

x, z	11

y	10

Pass-By-Reference

Note: Arrow points to *next* instruction.

- C++ allows you to use real pass-by-*reference*
  - Client passes in an argument with normal syntax
    - Function uses reference parameters with normal syntax
    - Modifying a reference parameter modifies the caller's argument!

```
void swap(int& x, int& y) {
  int tmp = x;
  x = y;
  y = tmp;
}

int main(int argc, char** argv) {
  int a = 5, b = 10;

  swap(a, b);
  cout << "a: " << a << "; b: " << b << endl;
  return EXIT_SUCCESS;
}
```

Parameters are attached
To variables provided by caller

(main) **a**	5

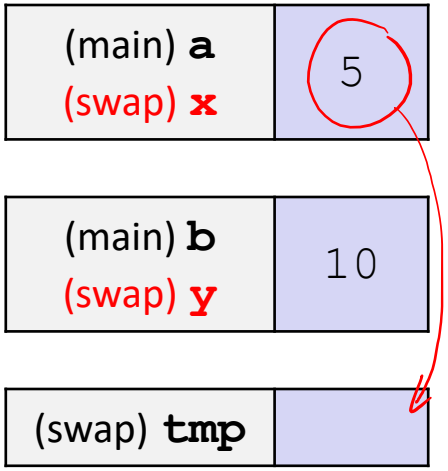
(main) **b**	10

Pass-By-Reference

Note: Arrow points to *next* instruction.

- C++ allows you to use real pass-by-*reference*
  - Client passes in an argument with normal syntax
    - Function uses reference parameters with normal syntax
    - Modifying a reference parameter modifies the caller's argument!

```
void swap(int& x, int& y) {
  int tmp = x;
  x = y;
  y = tmp;
}

int main(int argc, char** argv) {
  int a = 5, b = 10;

  swap(a, b);
  cout << "a: " << a << "; b: " << b << endl;
  return EXIT_SUCCESS;
}
```

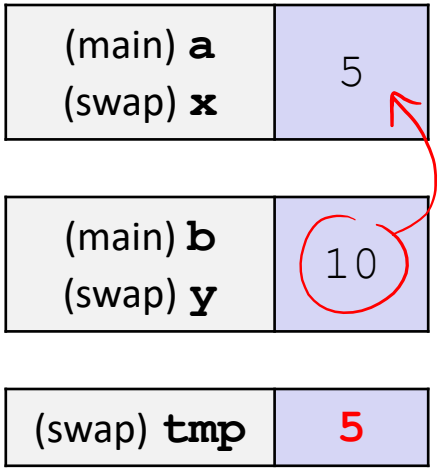


Pass-By-Reference

Note: Arrow points to *next* instruction.

- C++ allows you to use real pass-by-*reference*
  - Client passes in an argument with normal syntax
    - Function uses reference parameters with normal syntax
    - Modifying a reference parameter modifies the caller's argument!

```
void swap(int& x, int& y) {
  int tmp = x;
  x = y;
  y = tmp;
}

int main(int argc, char** argv) {
  int a = 5, b = 10;

  swap(a, b);
  cout << "a: " << a << "; b: " << b << endl;
  return EXIT_SUCCESS;
}
```

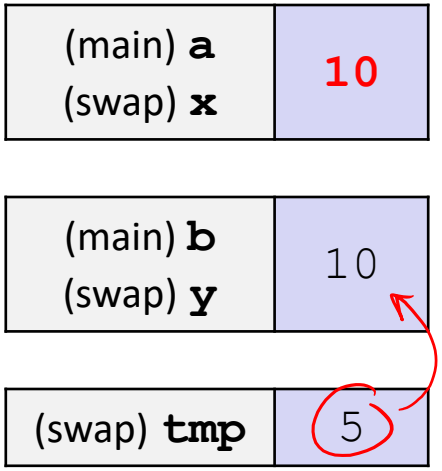


Pass-By-Reference

Note: Arrow points to *next* instruction.

- C++ allows you to use real pass-by-*reference*
  - Client passes in an argument with normal syntax
    - Function uses reference parameters with normal syntax
    - Modifying a reference parameter modifies the caller's argument!

```
void swap(int& x, int& y) {
  int tmp = x;
  x = y;
  y = tmp;
}

int main(int argc, char** argv) {
  int a = 5, b = 10;

  swap(a, b);
  cout << "a: " << a << "; b: " << b << endl;
  return EXIT_SUCCESS;
}
```



Pass-By-Reference

Note: Arrow points to *next* instruction.

- C++ allows you to use real pass-by-*reference*
  - Client passes in an argument with normal syntax
    - Function uses reference parameters with normal syntax
    - Modifying a reference parameter modifies the caller's argument!

```
void swap(int& x, int& y) {
  int tmp = x;
  x = y;
  y = tmp;
}

int main(int argc, char** argv) {
  int a = 5, b = 10;

  swap(a, b);
  cout << "a: " << a << "; b: " << b << endl;
  return EXIT_SUCCESS;
}
```

(main) **a** (swap) **x**	10

(main) **b** (swap) **y**	**5**

(swap) **tmp**	5

Pass-By-Reference

Note: Arrow points to *next* instruction.

- C++ allows you to use real pass-by-*reference*
  - Client passes in an argument with normal syntax
    - Function uses reference parameters with normal syntax
    - Modifying a reference parameter modifies the caller's argument!

```
void swap(int& x, int& y) {
  int tmp = x;
  x = y;
  y = tmp;
}

int main(int argc, char** argv) {
  int a = 5, b = 10;

  swap(a, b);
  cout << "a: " << a << "; b: " << b << endl;
  return EXIT_SUCCESS;
}
```

(main) **a**	10

(main) **b**	5

Pass-By-Reference

- C++ allows you to use real pass-by-*reference*
  - Client passes in an argument with normal syntax
    - Function uses reference parameters with normal syntax
    - Modifying a reference parameter modifies the caller's argument!

- Can use on objects as well!

- Now v is actually changed"

```
void populate_vec(vector<int>& v) {
  v.push_back(5950);
}

int main() {
  vector<int> v {};
  populate_vec(v);
  cout << v.size() << endl;
  // loop removed for space
  return EXIT_SUCCESS;
}
```

- **DID NOT GET PAST HERE IN LECTURE. THE REST WILL BE COVERED NEXT LECTURE**

Poll Everywhere

pollev.com/tqm

- What will happen when we run this?

A. Output "(3,3,3)"

B. Output "(3,3,2)"

C. Compiler error about arguments to foo (in main)

D. Compiler error about body of foo

E. We're lost…

```
void foo(int& x, int& y, int z) {
  z = y;
  x += 2;
  y = x;
}

int main(int argc, char* argv[]) {
  int a = 1;
  int b = 2;
  int& c = a;

  foo(a, b, c);
  cout << "(" << a << ", " << b
       << ", " << c << ")" << endl;

  return EXIT_SUCCESS;
}
```

Poll Everywhere

pollev.com/tqm

- What will happen when we run this?

Note: Arrow points to *next* instruction.

A. Output "(3,3,3)"

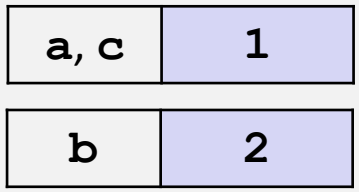
B. Output "(3,3,2)"

C. Compiler error about arguments to foo (in main)

D. Compiler error about body of foo

E. We're lost…

```
void foo(int& x, int& y, int z) {
  z = y;
  x += 2;
  y = x;
}

int main(int argc, char* argv[]) {
  int a = 1;
  int b = 2;
  int& c = a;

  foo(a, b, c);
  cout << "(" << a << ", " << b
       << ", " << c << ")" << endl;

  return EXIT_SUCCESS;
}
```



Poll Everywhere

pollev.com/tqm

- What will happen when we run this?

Note: Arrow points to *next* instruction.

A. Output "(3,3,3)"

B. Output "(3,3,2)"

C. Compiler error about arguments to foo (in main)

D. Compiler error about body of foo

E. We're lost…

```
void foo(int& x, int& y, int z) {
→  z = y;
   x += 2;
   y = x;
}

int main(int argc, char* argv[]) {
   int a = 1;
   int b = 2;
   int& c = a;

   foo(a, b, c);
   cout << "(" << a << ", " << b
        << ", " << c << ")" << endl;

   return EXIT_SUCCESS;
}
```

z	1

(main) a, c (foo) x	1

(main) b (foo) y	2

Poll Everywhere

pollev.com/tqm

- What will happen when we run this?

Note: Arrow points to *next* instruction.

A. Output "(3,3,3)"

B. Output "(3,3,2)"

C. Compiler error about arguments to foo (in main)

D. Compiler error about body of foo

E. We're lost…

```
void foo(int& x, int& y, int z) {
  z = y;
→ x += 2;
  y = x;
}

int main(int argc, char* argv[]) {
  int a = 1;
  int b = 2;
  int& c = a;

  foo(a, b, c);
  cout << "(" << a << ", " << b
       << ", " << c << ")" << endl;

  return EXIT_SUCCESS;
}
```

z	2

(main) a, c (foo) x	1

(main) b (foo) y	2

Poll Everywhere

pollev.com/tqm

- What will happen when we run this?

Note: Arrow points to *next* instruction.

A. Output "(3,3,3)"

B. Output "(3,3,2)"

C. Compiler error about arguments to foo (in main)

D. Compiler error about body of foo

E. We're lost…

```
void foo(int& x, int& y, int z) {
  z = y;
  x += 2;
→ y = x;
}

int main(int argc, char* argv[]) {
  int a = 1;
  int b = 2;
  int& c = a;

  foo(a, b, c);
  cout << "(" << a << ", " << b
       << ", " << c << ")" << endl;

  return EXIT_SUCCESS;
}
```

z	2

(main) a, c (foo) x	3

(main) b (foo) y	2

Poll Everywhere

pollev.com/tqm

- What will happen when we run this?

Note: Arrow points to *next* instruction.

A. Output "(3,3,3)"

B. Output "(3,3,2)"

C. Compiler error about arguments to foo (in main)

D. Compiler error about body of foo

E. We're lost…

```
void foo(int& x, int& y, int z) {
  z = y;
  x += 2;
  y = x;
}

int main(int argc, char* argv[]) {
  int a = 1;
  int b = 2;
  int& c = a;

  foo(a, b, c);
  cout << "(" << a << ", " << b
       << ", " << c << ")" << endl;

  return EXIT_SUCCESS;
}
```

z	2

(main) a, c (foo) x	3

(main) b (foo) y	2

Poll Everywhere

pollev.com/tqm

- What will happen when we run this?

Note: Arrow points to *next* instruction.

A. Output "(3,3,3)"

B. Output "(3,3,2)"

C. Compiler error about arguments to foo (in main)

D. Compiler error about body of foo

E. We're lost…

```
void foo(int& x, int& y, int z) {
  z = y;
  x += 2;
  y = x;
}

int main(int argc, char* argv[]) {
  int a = 1;
  int b = 2;
  int& c = a;

  foo(a, b, c);
  cout << "(" << a << ", " << b
       << ", " << c << ")" << endl;

  return EXIT_SUCCESS;
}
```

z	2

(main) a, c (foo) x	3

(main) b (foo) y	3

Poll Everywhere

pollev.com/tqm

- What will happen when we run this?

Note: Arrow points to *next* instruction.

A. Output "(3,3,3)"

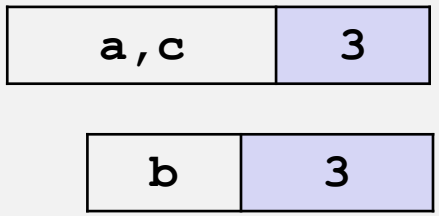
B. Output "(3,3,2)"

C. Compiler error about arguments to foo (in main)

D. Compiler error about body of foo

E. We're lost…

```
void foo(int& x, int* y, int z) {
  z = *y;
  x += 2;
  y = &x;
}

int main(int argc, char** argv) {
  int a = 1;
  int b = 2;
  int& c = a;

  foo(a, &b, c);
→ std::cout << "(" << a << ", " << b
    << ", " << c << ")" << std::endl;

  return EXIT_SUCCESS;
}
```



Lecture Outline

- Introduction & Logistics
  - Course Overview
  - Assignments & Exams
  - Policies
- **C++ Intro**
  - Helloworld, iostream, strings
  - vector
  - references
  - **map & unordered_map (start)**

STL map

- One of C++'s *associative* containers: a key/value table, implemented as a search tree
  - http://www.cplusplus.com/reference/map/
  - General form: `map<key_type, value_type> name;`
  - Keys must be *unique*
    - **multimap** allows duplicate keys
  - Efficient lookup (O(log n)) and insertion (O(log n))
    - Access `value` via **operator[]** (example: **map_name[key]**)
      - if key doesn't exist in map, it is added to the map with a "default" value
  - Elements are type `pair<key_type, value_type>` and are stored in *sorted* order (key is field **first**, value is field **second**)
    - Key type must support less-than operator (<)

map Example

#include <map>

map_example.cpp

```
int main(int argc, char** argv) {
  map<int,string> table{};
  map<int,string>::iterator it{};

  table.insert(pair<int,string>(2, "hello"));
  table[4] = "NGNM";
  table[6] = "mutual aid";

  cout << "table[6]:" << table[6] << endl;

  it = table.find(4);

  if (it != table.end()) {
    cout << "4 exists as a key in the map" << endl;
  }

  cout << "iterating:" << endl;
  for (pair<int, string> p : table) {
    cout << "[" << p.first << "," << p.second << "]" << endl;
  }
  return 0;
}
```

Map elements

Equivalent behavior

Returns iterator. (end if not found)
can also use map.contains() to see if a key exists

Access the key and value stored in the pair

STL `set`

- One of C++'s *associative* containers: a container of unique values, implemented as a search tree
  - http://www.cplusplus.com/reference/set/
  - General form: `set<element_type> name;`
  - elements must be *unique*
    - **`multiset`** allows duplicate elements
  - Efficient lookup (O(log n)) and insertion (O(log n))
  - Inserting an element that already exists does nothing
  - Can use `count(element)` to see if the element exists
  - Elements are stored in *sorted* order
    - element type must support less-than operator (<)

`set` Example

set_example.cpp

```
#include <set>

int main(int argc, char** argv) {
  set<string> names {};

  names.insert("bjarne");
  names.insert("ken");
  names.insert("dennis");
  names.insert("travis");
  names.insert("bjarne");

  bool exists = names.contains("bjarne");
  cout << "Is bjarne in the set?: " << exists << endl;

  numbers.erase("travis");

  for (string name : names) {
    cout << name << endl;
  }
  return EXIT_SUCCESS;
}
```

Doesn't insert duplicate elements

prints "true"

Removes the element "travis"

Prints every name in the set

Unordered Containers (C++11)

- `unordered_map`, `unordered_set`
  - And related classes `unordered_multimap`, `unordered_multiset`
  - Average case for key access is O(1), so generally preferred
    - But range iterators can be less efficient than ordered `map`/`set`
  - See *C++ Primer*, online references for details