CS111ACE Section 1

Welcome (back) to C++!

Attendance Form -> https://tinyurl.com/CS111A-pollen
PollEV (For anonymous Q+A) -> pollev.com/tripmaster419
Game Plan

- Icebreaker
- Welcome + Syllabus
- C/C++ Refresh
Game Plan

- Icebreaker:
  - Name
  - Pronouns (if you’d like!)
  - Hometown
  - Name of the person to the left of you
  - Favorite ___
Who am I?

- 2nd year Coterm in CS (Systems)
- CS undergrad (Computer Engineering)
- Section Leader, Head TA (CS106B), Coordinator (CS198)
- I LOVE teaching!
Syllabus Time!
Classroom Norms

- We’re all here to learn! Please be respectful of each other.
- There are no bad questions! This material is difficult, and I’m always happy to clarify / rewind
- Please refrain from using jargon (defined here as overly-technical language that is unrelated to what we’re learning) in class.
  - Don’t be that person
Lightning Review -> CS107

Before we start…

One of the most important things to remember in all of computer architecture…

One Bit is a single 0 or 1
One Byte is 8 bits (i.e. a series of eight 0 or 1’s)
Lightning Review -> CS107

The memory used by our programs reside in an array-like structure called main memory or program memory.

We’ll learn more about these in a few weeks…
Lightning Review -> CS107

The memory used by our programs reside in an array-like structure called **main memory** or **program memory**.

<table>
<thead>
<tr>
<th>Index</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX</td>
<td>MAX</td>
</tr>
<tr>
<td>MAX-1</td>
<td>MAX-1</td>
</tr>
<tr>
<td>MAX-2</td>
<td>MAX-2</td>
</tr>
<tr>
<td></td>
<td>(many more indices)</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

* You can think of memory kind of like this! *
The memory used by our programs reside in an array-like structure called **main memory** or **program memory**.

<table>
<thead>
<tr>
<th>Index</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX</td>
<td>MAX</td>
</tr>
<tr>
<td>MAX-1</td>
<td>MAX-1</td>
</tr>
<tr>
<td>MAX-2</td>
<td>MAX-2</td>
</tr>
<tr>
<td>(many more indices)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

* You can think of memory kind of like this! *

Variables take up slots of this array. We reference them at their **lowest address**.
Lightning Review -> CS107

The memory used by our programs reside in an array-like structure called **main memory** or **program memory**.

You can think of memory kind of like this!

<table>
<thead>
<tr>
<th>Index</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX</td>
<td>MAX</td>
</tr>
<tr>
<td>MAX-1</td>
<td>MAX-1</td>
</tr>
<tr>
<td>MAX-2</td>
<td>MAX-2</td>
</tr>
<tr>
<td>(many more indices)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Variables take up slots of this array. We reference them at their **lowest address**.

In the diagram on the left, what does a 4-byte integer look like at address 0x2?
The memory used by our programs reside in an array-like structure called **main memory** or **program memory**.

* You can think of memory kind of like this!

<table>
<thead>
<tr>
<th>Index</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>index MAX</td>
<td>byte MAX</td>
</tr>
<tr>
<td>index MAX-1</td>
<td>byte MAX-1</td>
</tr>
<tr>
<td>index MAX-2</td>
<td>byte MAX-2</td>
</tr>
<tr>
<td>(many more indices)</td>
<td>(many more indices)</td>
</tr>
<tr>
<td>index 5</td>
<td>byte 5</td>
</tr>
<tr>
<td>index 4</td>
<td>byte 4</td>
</tr>
<tr>
<td>index 3</td>
<td>byte 3</td>
</tr>
<tr>
<td>index 2</td>
<td>byte 2</td>
</tr>
<tr>
<td>index 1</td>
<td>byte 1</td>
</tr>
<tr>
<td>index 0</td>
<td>byte 0</td>
</tr>
</tbody>
</table>

Variables take up slots of this array. We reference them at their **lowest address**.

In the diagram on the left, what does a 4-byte integer look like at address 0x2?

Stanford University
Lightning Review -> CS107

The memory used by our programs reside in an array-like structure called **main memory** or **program memory**.

You can think of memory kind of like this!

<table>
<thead>
<tr>
<th>Index</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX</td>
<td>MAX</td>
</tr>
<tr>
<td>MAX-1</td>
<td>MAX-1</td>
</tr>
<tr>
<td>MAX-2</td>
<td>MAX-2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Variables take up slots of this array. We reference them at their lowest address.

Food for thought: we *could* either pass around our 4 byte integer value, or *its reference in memory*. Would it ever make more sense to pass the reference instead?
Lightning Review -> CS107

The memory used by our programs reside in an array-like structure called main memory or program memory.

* Because the “array” of main memory is so big, we very frequently need to keep track of “indices” for our data as variables.
Lightning Review -> CS107

The memory used by our programs reside in an array-like structure called main memory or program memory.

* Because the “array” of main memory is so big, we very frequently need to keep track of “indices” for our data as variables
* Another problem is that we need to remember how many bytes to read after our initial byte (i.e. a char takes up 1 byte but an int takes up 4 bytes!)
Lightning Review -> CS107

These “indices” into main memory are so common, that we created a whole type for them: **pointers**
These “indices” into main memory are so common, that we created a whole type for them: **pointers**

- A pointer is just a memory address. It might look like an ugly **hexadecimal** number (0xffffc3a0), but you could also read that in **decimal** as a *really* big index into a really big array (4294951840)
These “indices” into main memory are so common, that we created a whole type for them: **pointers**

- A pointer is just a memory address. It might look like an ugly hexadecimal number (0xffffc3a0), but you could also read that in decimal as a *really* big index into a really big array (4294951840)
  - The reason why we use pointers is that they’re **much more expressive** than using integer indices…
These “indices” into main memory are so common, that we created a whole type for them: **pointers**

- A pointer is just a memory address. It might look like an ugly **hexadecimal** number (0xffffffffa0), but you could also read that in **decimal** as a **really** big index into a really big array (4294951840)
  - The reason why we use pointers is that they’re **much more expressive** than using integer indices…
    - int **x** must be dereferenced **twice** to get data
Lightning Review -> CS107

These “indices” into main memory are so common, that we created a whole type for them: **pointers**

- A pointer is just a memory address. It might look like an ugly **hexadecimal** number (0xffffc3a0), but you could also read that in **decimal** as a really big index into a really big array (4294951840)
  - The reason why we use pointers is that they’re **much more expressive** than using integer indices…
    - int **x** must be dereferenced **twice** to get data
    - int **x** refers to 4 **bytes**.
  - Neither of these things would be expressible with normal indices (unless you had crazy variable names :3)
These “indices” into main memory are so common, that we created a whole type for them: **pointers**

- A pointer is just a memory address. It might look like an ugly **hexadecimal** number (0xffffffffa0), but you could also read that in **decimal** as a **really** big index into a really big array (4294951840)
  - The reason why we use pointers is that they’re **much more expressive** than using integer indices…
    - int **x** must be dereferenced **twice** to get data
    - int **x** refers to 4 bytes.
  - Neither of these things would be expressible with normal indices (unless you had crazy variable names :3)
- Final thoughts: although pointers can get tricky, I find that drawing things out and visualizing them as **numbers** can simplify them!
Any questions?

- Particularly those of the “will I have to remember X in CS111?”
These “indices” into main memory are so common, that we created a whole type for them: **pointers**

- A pointer is just a memory address. It might look like an ugly **hexadecimal** number (0xffffc3a0), but you could also read that in **decimal** as a **really** big index into a really big array (4294951840)
  - The reason why we use pointers is that they’re **much more expressive** than using integer indices…
    - int **x** must be dereferenced **twice** to get data
    - int **x** refers to 4 **bytes**.
- Neither of these things would be expressible with normal indices (unless you had crazy variable names :3)
- Final thoughts: although pointers can get tricky, I find that drawing things out and visualizing them as **numbers** can simplify them!
Example Exercises

- Let’s say I’m given a void * pointer called myPointer, and I’m told that it’s pointing to a float. How would I extract the float from the pointer?
Example Exercises

- Let’s say I’m given a void * pointer called `myPointer`, and I’m told that it’s pointing to a `float`. How would I extract the `float` from the pointer?
  - `float f = *(float *)myPointer;`
Example Exercises

● Let’s say I’m given a void * pointer called `myPointer`, and I’m told that it’s pointing to a `float`. How would I extract the `float` from the pointer?
  ○ `float f = *(float *)myPointer;`

● When would I want to use a `double` pointer? By this I mean a pointer to a pointer (something like `int **myInt;`)
Example Exercises

- Let’s say I’m given a void * pointer called `myPointer`, and I’m told that it’s pointing to a `float`. How would I extract the `float` from the pointer?
  - `float f = *(float *)myPointer;`

- When would I want to use a `double` pointer? By this I mean a pointer to a pointer (something like `int **myInt;`)
  - This question is a little misleading, but you’ll want to use double pointers when you want to modify what the inner pointer is pointing to. You can think about it like passing a pointer by reference in c++. 
Example Exercises

- Let’s say I’m given a void * pointer called myPointer, and I’m told that it’s pointing to a float. How would I extract the float from the pointer?
  - float f = *(float *)myPointer;

- When would I want to use a double pointer? By this I mean a pointer to a pointer (something like int **myInt;
  - This question is a little misleading, but you’ll want to use double pointers when you want to modify what the inner pointer is pointing to. You can think about it like passing a pointer by reference in c++.

```c
void addToFront (node **front, node *toAdd) {
}
```

Remember that dereference on the RHS means “read the value”, but on the LHS it means “overwrite the value”
Breather Slide...
Your First Task

● Go to github and clone the repo for our first section.
  ○ I’ll show you how to do this!
  ○ Cloning info can be found at
    https://github.com/tmaster628/Win23_CS111A_S1

● Work in groups of 2-3 and code up the 5 functions in main.cpp
  ○ You will probably need to google things! This is an important skill in programming
Helpful / Not Helpful Links

- How to Debug Memory Code (Not Helpful)
- Thorough but pretty clear introduction to STL (Helpful)
- My Github Page for repos (Somewhat Helpful)