Welcome to CS429H!

Week 0

Introductions

Fun game: Try to guess how old each of us are :3

Why pay attention in this class?

- Understand how computers really, actually work
- Write more efficient, performant code
- Make educated choices while programming
- Affects every discipline in computer science, and anything done on a computer
- Required for graduating from UT
- It’s really fun

Class Logistics

“Amateurs talk about tactics, but professionals study logistics.” - idk

Poll

How do you feel about this class?

A. I’m completely prepared and not scared at all!
B. I’m pretty confident I’ll do ok
C. It could go either way
D. Not gonna lie I’m a little worried
E. Help

Logistical stuff

- FERPA (...and why you should care)
  - Student confidentiality
  - Your information is confidential and will not be shared
  - You cannot share class material publicly
- Title IX (...and why you should care)
  - ALL teaching assistants and professors are mandatory reporters for Title IX
  - If we hear anything, we are required to report incidents of sex discrimination, sexual harassment, sexual assault, sexual misconduct, interpersonal violence, and stalking
  - Resources

Stress

- 429H is not an easy class
  - Lots of new materials
  - Unfamiliar programming environments
  - Fast, often relentless pace
- Struggling in this course is normal
  - There will be times you won't know the answer of the solution
  - This is expected—we want we everyone to succeed, but the only way we can help is if you ask for it
- If you find yourself overly overwhelmed or spending more time on this class than you think you should be, please reach out to Dr. Gheith or the TAs
  - We can help out as far as the class goes
  - We can provide other resources where we are not able to help

Mental health resource available at UT

Resources

Confidential

- Counseling and Mental Health Center
- University Health Services
- University Ombuds
- Confidential Advocates
- Victims Advocate Network (VAN)

Non-Confidential

- University Title IX Coordinator
- Student Emergency Services
- Student Conduct and Academic Integrity
- Behavior Concerns Advice Line
- University of Texas Police Department

Logistical stuff

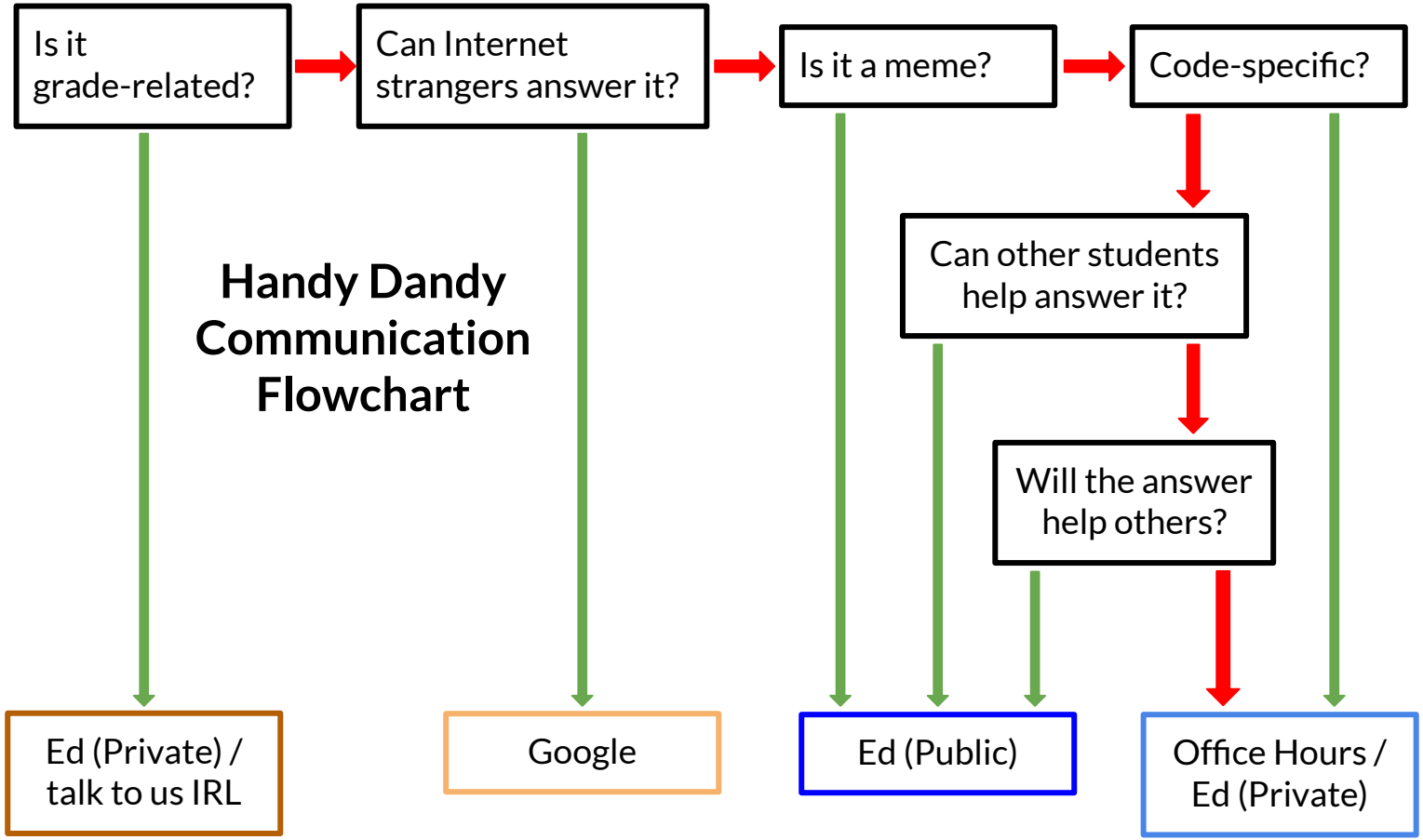
- Canvas
  - Only used for posting grades
  - Grants access to Ed Discussion

Logistical stuff

- Ed Discussion
  - We will be posting announcements and clarifications here
  - Notifications are iffy, please check the site regularly
  - Please use private posts for anything specific to your code or your grades
  - Otherwise please post publicly so others can benefit from your question!
  - Help answer others' questions!
  - Private posts are also a great way to contact us + Dr. Gheith
- **If you do not have access to Canvas/Ed please stay afterwards and talk to us**

Ed Etiquette (“Ediquette”)

- If more people can see your question, you will get an answer much faster
- Before asking a private question, ask yourself:
  - Would this create a beneficial discussion?
  - Would other students learn something from this question?
- Unless your question contains information about your code or something specific to your implementation, there is a good chance it would be better not being private.
- Answer other students’ questions! This will help you to learn more
- If you know part of the answer, put it in a follow-up discussion
- Ed is a great way to ask questions about assignments
- Discord/Skype/Messenger/Slack/IRC/following us home is NOT


Is it grade-related?
Can Internet strangers answer it?
Is it a meme?
Code-specific?
Handy Dandy Communication Flowchart
Can other students help answer it?
Will the answer help others?
Ed (Private) / talk to us IRL
Google
Ed (Public)
Office Hours / Ed (Private)

Quiz

Which one of these would be a good **public** Ed post?

A. “Here’s a meme I made about yesterday’s lecture”
B. “Quiz question submission: (...)”
C. “What is the return type of getchar() in libc?”
D. “qiuz fri???? tomorow comp arch. done for semester??? no more content or quiz correct??”
E. “This is going to affect the local economy. Thanks Joe Biden”

Quiz

Which one of these would be a good **public** Ed post?

Afterthoughts on Pipelining

Logistical stuff

- Office hours
  - **Come to office hours!** (bolded because this is important)
  - We have made a pinned post on Ed Discussion detailing OH times and locations
    - https://edstem.org/us/courses/53774/discussion/4137439
- We can:
  - Answer questions about class content
  - Help debug code
  - Go over past quizzes
  - Give life advice*
- We cannot:
  - Write code for you
  - Give quiz answers
  - Do your taxes

*YMMV.

Logistical stuff

- Office hours (this gets two slides because it's so important)
  - Show up to office hours!
- We want each of you to succeed in this course, but we cannot do that if we never talk to you
- Showing up to office hours does not mean: you don't know what's going on, we're going to think you're dumb, you can't do an assignment yourself, etc.
- Showing up to office hours does mean: you're making us happy, you want to learn more about the class material, etc.

Logistical stuff

- Discussion sections
  - You are here! Please don't leave :(
  - Quiz every other week, starting next week
- General structure:
  - First half quiz or quiz review from previous week
  - Second half content related to what's covered in lecture (maybe even brownies)
  - Impromptu office hours if we end early

Prep Session Recap

- Don’t cheat (this includes ChatGPT)
- Come to office hours
- Ask questions
- Do your work on the lab machines
  - VSCode has a nice remote extension you can use
- Know how to access a lab machine through a terminal and use vim
- If VSCode fails for whatever reason and you can’t come to the GDC, but a terminal still connects, this is not a valid reason for an extension
- Work together by sharing high level ideas!

Assignments/Grading

Poll

How’s your status on P1?

A. What’s P1?
B. I’ve heard of it
C. I’ve cloned the starter code and/or looked through it
D. I’ve started planning/writing code
E. I’m mostly done but might still have bugs
F. P1 any% speedrun

Overview

- One project every week (50% of final grade)
  - Test case due Monday(?)
  - Project implementation + report due Wednesday (?)
  - P1 is out now! Go check your emails!
- One quiz every two weeks (40% of final grade)
  - Roughly 45 minutes
  - Covers content up to the most recent Tuesday's lecture (inclusive)
- Participation (10% of final grade)
  - Quiz questions (2+) (counts for 2/10 points)
  - Scribe notes (4+) (counts for 4/10 points)
  - ???

Projects

- Weekly assignments (generally due on Wednesday)
- Graded on:
  - Correctness
  - Test Case
  - Code Quality
  - Report
- Test Cases are due 2 days before the code and report
- We encourage you to crowdsource a list of invalid test cases beforehand
  - Your classmates will be able to fix their test case before the deadline
  - You will know which test cases you don't have to debug
  - A spreadsheet would be good for this

Correctness

- A subset of student tests plus the official tests will be used for grading correctness
  - Will include expected-case behavior
  - Will include edge-case behavior
  - May include stress tests (be prepared to optimize your code!)
  - Will not include invalid cases (behavior not covered by the spec*, incorrect behavior per the spec)
- You should pretty much know your correctness score by the deadline
- Correctness matrix: https://www.cs.utexas.edu/~gheith/cs429h_s24_p1.html
  - Noah's improved matrix: https://www.cs.utexas.edu/~nklayman/cs429h_s24_p1.html
- ~60-70% of the grade

*spec is defined by the README and conversations in discussion/lecture and on Ed

Generated at 2023/10/26 14:48, avg score: 931

	Test	t0	t1	5	10	16	27	28	33	37	51	2	3	4	6	7	8	9	12	13	15	18	20	21	22	23	24	25	26	29	30	31	32	34
☐	Weight	10	10	6	6	6	6	6	6	6	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Passing	60	60	59	52	57	60	59	60	58	60	42	60	58	59	59	5	57	59	60	57	60	39	40	60	46	4	39	38	46	37	42	60	55
☐	2_e199	1000	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.
☐	3_51b6	1000	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	x	.	.	x	.	x	.	.	.	.	.
☐	4_5cbf	1000	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	x	.	.	.	.	.	x	.	.	.	x	x	.	x	.	.	.	.
☐	5_e5d6	1000	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	.	.	x	.	x	.	.	.	.	.
☐	6_9849	1000	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	.	.	x	x	.	.	.	.	.	.
☐	7_e238*	455	.	.	.	x	.	.	.	.	.	x	.	x	.	.	x	.	.	.	x	.	x	x	.	x	x	x	x	x	.	.	.	.	.
☐	8_285a	1000	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	.
☐	9_5822	1000	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	.	.	x	.	.	x	x	.	.	x
☐	10_51a1	1000	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.
☐	11_77b7	911	.	.	.	x	.	.	.	.	.	x	.	.	.	.	x	.	.	.	.	.	x	x	.	x	x	x	x	.	.	.	.	.	.
☐	12_d0b3	1000	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	x	.	.	x	.	.	.	.	.	.	.	.
☐	13_2c8e*	500	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.
☐	14_0165	1000	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.
☐	15_e851	1000	.	.	.	.	.	.	.	.	.	x	.	.	.	.	x	.	.	.	.	.	x	.	.	x	x	x	x	x	.	x	.	.	.
☐	16_c7a1	1000	.	.	.	.	.	.	.	.	.	x	.	.	.	.	x	x	.	.	.	.	x	x	.	.	x	x	x	x	.	x	.	.	.
☐	17_20c3	823	.	.	.	x	.	.	.	.	x	x	.	x	.	.	x	.	.	.	x	.	x	x	.	x	x	.	x	.	x	.	.	.	.
☐	18_6b81	1000	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	.	.	.	x	.	.	x	.	.	.	.

Test Cases

- **Write good test cases! Your classmates will thank you!**
- What we are looking for:
  - Test cases that are valid
  - Test cases that are very thorough and effectively test for correctness
  - Test cases that find unique edge cases or creative code paths not used by the provided tests
  - Test cases that are **useful to other people! (feedback/comments are good!)**
- What we are not looking for:
  - Invalid tests
  - Obfuscated / hard to understand tests
  - Overly simple tests
  - Tests that will crash the grading server
- Just because your test isn't selected doesn't mean it's bad!
- ~10-20% of the grade

Code Quality

- Make sure to use a consistent code style!
- Unlike Java or Python, no single widely accepted C style
- Feel free to adopt one of many on the internet or to define your own
- Include documentation only where necessary:
  - to motivate confusing function signatures
  - to explain complex mathematical operations (particularly pointer math)
  - to summarize and break up large blocks of code
- Typically, not all of your code should be in one singular file
  - If a file gets really big, consider if it would make sense to split it up into multiple files
- Meaningful git commit messages, don't leak memory
- Main thing we are looking for - does your code make sense, is it readable?
- ~5-15% of the grade

Report

- Just a handful of questions, can be found in the REPORT file
- 2-3 sentence answers are usually enough
  - Please do not write an essay for us
- Feel free to do research, just cite your sources
  - Links are fine, this isn't an English class
- ChatGPT: You can use it to help, but the actual response must be your own
  - Though often the information it gives you is wrong anyways
- Make sure to commit the report
- ~5-15% of the grade

How to turn in a project (Git crash course)

- Step 0: Set up SSH access
  - You should have received an email with instructions on how to submit your public key
  - Follow the steps if you haven't already done so and gotten access to P1
  - This is very important! Every project is released in this way!
  - **If you do not have access yet, stay after and talk to us! Do not wait until Wednesday!**
- Once you do this, you will be able to clone your copy of P1

$ cat ~gheith/public/k429h

```
K=~gheith/dropbox/cs429h_s24/`id -un`.pub

test -f ~/.ssh/id_rsa.pub || ssh-keygen -N "" -t rsa -f ~/.ssh/id_rsa

cp ~/.ssh/id_rsa.pub $K

chmod go+r $K
```

How to turn in a project (Git crash course)

- `git clone git@git.gheith.com:cs429h_s24_p1_<csid>`
  - “Download the project from the server to my machine”
- `git add main.c other.c`
  - “I’d like to mark main.c and other.c as files I want to save”
- `git commit -m “changes i just made”`
  - “Save all the files I’ve added”
- `git push`
  - “Push all the changes I’ve saved to the server”
  - If you don’t push, we can’t grade it

How to turn in a project (Git crash course)

Typical workflow:

1. “git clone” the project
2. Make some changes to the code
3. Test the changes if possible
4. “git add” your changes
5. “git commit” your changes
6. Repeat steps 2-5 as desired (can push many commits together)
7. “git push” your changes to the server
8. Repeat steps 2-8 until you’re done with the project

Projects

- All projects have a deadline of 11:59, and an additional “soft deadline”
  - The soft deadline will be 11:59 or later, never earlier
  - The soft deadline will not be known beforehand
  - Projects submitted after 11:59 but before the soft deadline will not be considered late
  - The matrix will not reflect the soft deadline (but your grades will)
- After the deadline, you can turn a project in late at any time* for 50% of the grade
  - To make sure we see it, submit a regrade request. (See later slides)
- By default we will grade the last commit before the soft deadline
- Soft deadlines are chosen to benefit the most people

*during the semester. Please don’t come find me in 20 years

Quizzes

- On paper - bring a pencil or pen next week!
- Open notes/open book
- Not open internet/open friends/open StackOverflow
- Will cover lecture content and project content up to the Tuesday before

Quiz

Assuming no catastrophic weather events, how many quizzes will we have?

A. 1
B. 2
C. 3
D. 4
E. 5
F. 6
G. 7
H. 8
I. 9
J. a

Participation

Important factors:

- Questions/comments in class/discussion
- Scribe notes
- Ed contributions
- Office hours attendance
- Submitting quiz questions

If you participate in the class you'll do better in the class (source: empirical testing)

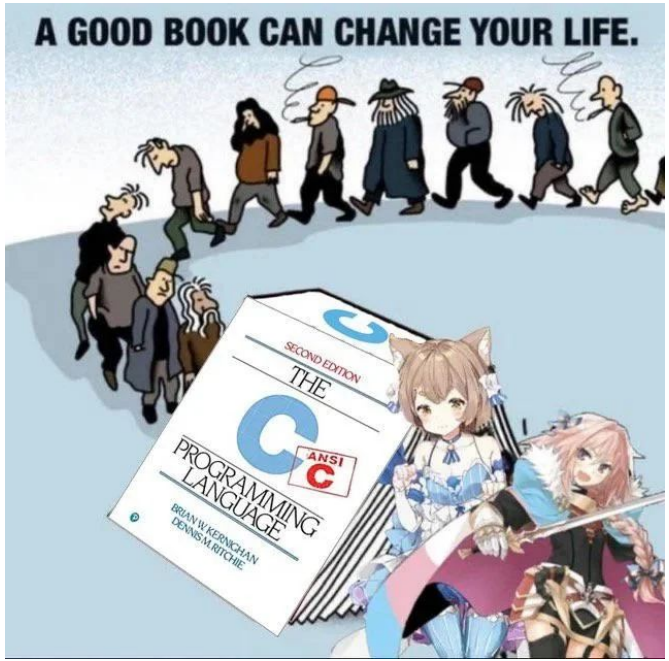
Regrade Requests

- If you think we made a mistake in grading something:
  - Make a private note on Ed
  - Tag it with Regrades - we may miss it otherwise
  - Include your name, eid, which assignment, any other info we might need to figure out what you actually want graded
  - Include a description of why you think something should be regraded (justify yourself)
  - Please do not submit frivolous regrade requests just because you're hoping to get more points
- Don't wait until the last minute!

P1

Poll

How familiar are you with C?



A. Never heard of it
B. I've used it before
C. I could do a 314 assignment in C
D. I'm pretty good at C
E. I'm Dennis Ritchie

What's an interpreter? Something ducks walk on?

- Some languages are **compiled** into machine language
  - A program called the compiler takes the human written code and converts it to machine code, which is then run natively by the architecture
  - C, C++, Rust
  - Example: Your mom wants you to get groceries, so she writes down a shopping list and gives it to you. You look at the list and realize she forgot a semicolon on line 4 so you refuse to go.
- Some languages are **interpreted**
  - A program called the interpreter reads the human written code bit by bit and acts out each part of the code
  - Python, JavaScript
  - Example: Your mom wants you to get groceries, so she sends you to the supermarket. Then she facetimes you and tells you exactly what to do because you don't know what a radish looks like

Quiz

Is the critter language compiled or interpreted?

A. Compiled
B. Interpreted
C. Yes
D. No

C basics (“casics”)

- Primitives
- Pointers
- Arrays
- Strings
- Structs
- Memory Layout/Structure
- Malloc/Free

C Primitives

- Integers
  - Signed vs unsigned
  - Integer widths
- Float / Double
- Char
- Bool
- Void

C Primitives

- Integers
  - Signed vs unsigned
  - Integer widths
- Float / Double
- Char
- Bool
- Void

```
#include <stdint.h>
#include <stdbool.h>
int a;
long b;
int32_t c;
uint64_t d;
float e;
double f;
bool g = true;
bool h = 0;
char i = ‘A’;
char j = 65;

void foo() {return;}
```

C Primitives

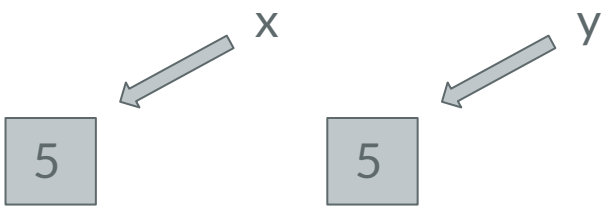
- Integers
  - Signed vs unsigned
  - Integer widths
- Float / Double
- Char
- Bool
- Void

```
#include <stdint.h>
#include <stdbool.h>
int a; // 2 or 4B (usually 4)
long b; // 4 or 8B (usually 8)
int32_t c; // exactly 4B
uint64_t d; // exactly 8B
float e;
double f;
bool g = true;
bool h = 0;
char i = 'A';
char j = 65;

void foo() {return;}
```

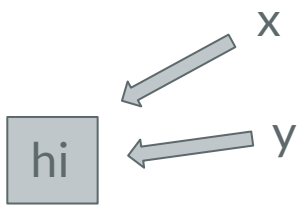
Pointers

- Pass-by-value vs Pass-by-reference
- Java example

```
int x = 5;
int y = x;
```




```
Object x = new Object(“hi”);
Object y = x;
```

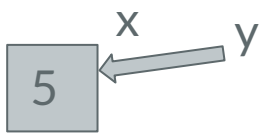


Pointers

- In C, you get full control of memory
- All data types can be a “pointer” version - this is a distinct data type
- Indicated by an asterisk between the type and variable name
- Ampersand converts to pointer reference; asterisk dereferences

```
uint32_t x = 5;
uint32_t y = x;
```

```
uint32_t x = 5;
uint32_t *y = &x;
uint32_t z = *y;
```



Pointers

- Whitespace does not matter in C
- Careful when declaring multiple pointer variables at once

```
uint32_t* a;
uint32_t *b;
uint32_t * c;
uint32_t* d, e; // Is e an int or int ptr?
```

Pointers

- Whitespace does not matter in C
- Careful when declaring multiple pointer variables at once

```
uint32_t* a;
uint32_t *b;
uint32_t * c;
uint32_t* d, e; // Is e an int or int ptr?
// Compiler sees as uint32_t *d; uint32_t e;
```

Pointers

- Internally, pointers are just unsigned integers
  - Size depends on architecture
- Corresponds to the location in memory they point to

```
uint32_t *x = 0x80000000;
*x = 5;
```

0x80000000	0x80000004	0x80000008	0x8000000C	0x80000010
5				

Pointers

- This means you can do math with them
- **C automatically does pointer math relative to the size of the data type**
  - You'll see why later
- To get more granularity, cast to a smaller data type

```
uint32_t *x = 0x80000000;
*x = 5;
*(x+1) = 10;
```

```
char *y = (char*)x;
*(y+8) = 40;
```

0x80000000	0x80000004	0x80000008	0x8000000C	0x80000010
5	10	40		

Pointers

- Pointers are just an address
- void* can point to anything
  - Cannot directly dereference - have to cast first
- Can have pointers to functions

```
void *x;
*(int*) x = 429;
*(char*) x = '\0';
/* Functions that take or return ptrs */
bool my_generic_function(void *ptr){...}
void *my_other_generic_function(int i) {...}
/* Pointer to a function */
int (*my_function_ptr)(int, int);
```

Arrays

- Arrays in C are just pointers to the first element!
- Stored contiguously in memory
- Indexing operator [] is just an alias for dereferencing and pointer math

```
int x[5];
x[0] = 1;
x[1] = 2;
x[3] = 10;
```

```
int *y = x;
*(y+0) = 1;
*(y+1) = 2;
*(y+3) = 10;
```

Arrays

Some more generalization on arrays and pointers

- *(ptr + i) is the same as ptr[i]
- ptr + i is the same as &ptr[i]
- fun fact: ptr[i] is the same as i[ptr]

Since arrays are just sequential data, no way to get array length (store it yourself)

Strings

But what about strings? Just an array of characters that ends with ‘\0’ (zero byte)

The array may be longer than the string, but never shorter than (str_length + 1)

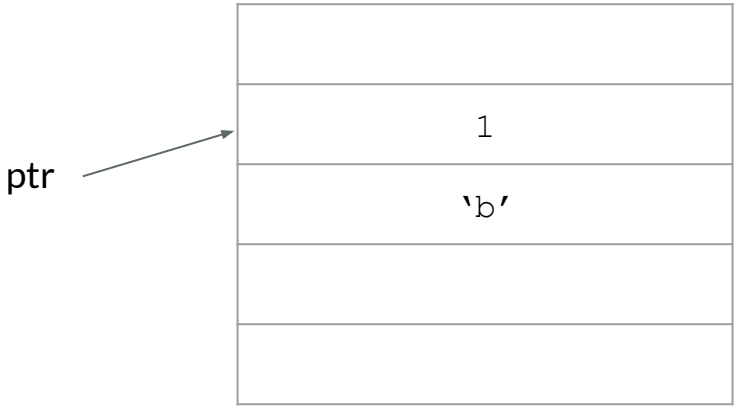
Even more pointers!

```
char *str = “Hello, world!”;
char ch = str[4]; // what’s in ch?
str[0] = 'A'; //segmentation fault (literal strings are read-only)
char new_str[100];
strcpy(new_str, str); //now we have an editable copy of str
new_str[0] = 'A';
```

Structs

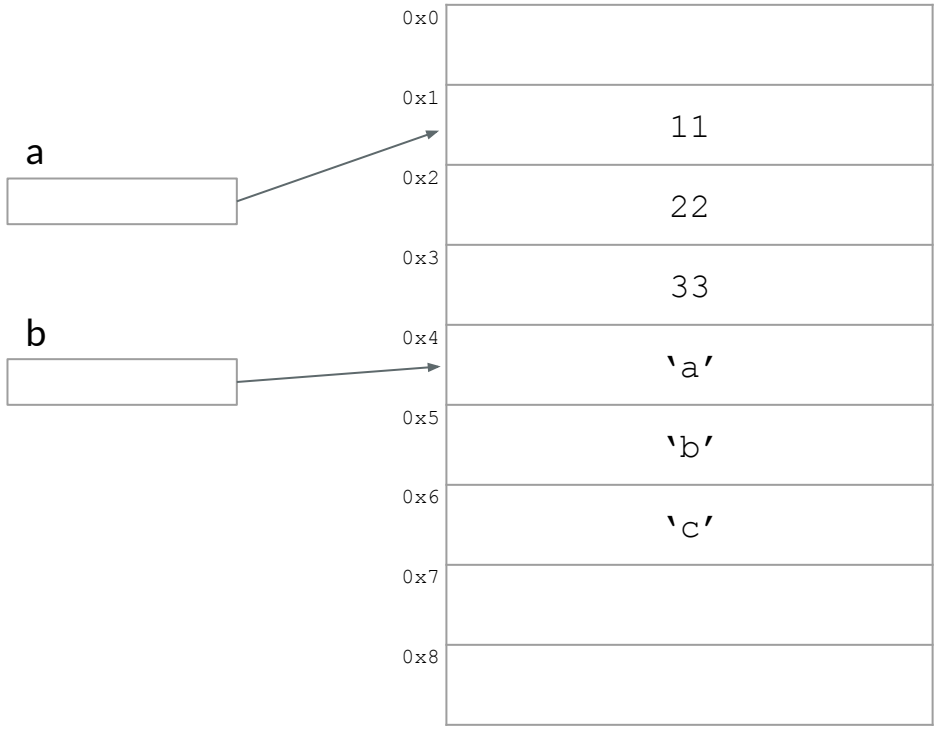
- A wrapper around a group of variables
- Keeps them together in memory

```
struct example {
    uint8_t a;
    char    b;
};
...
struct example * ptr = {1, 'b'};
```



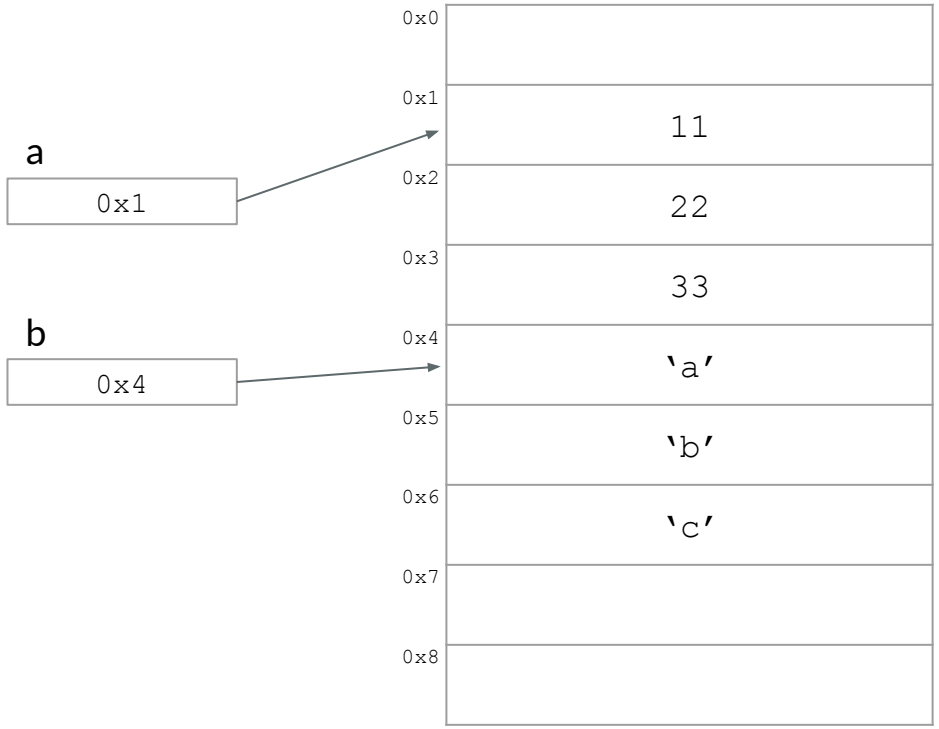
Annotate on the screen!

```
uint8_t a[] = {11, 22, 33};
char    b[] = {'a', 'b', 'c'};

// What goes in a and b?
```



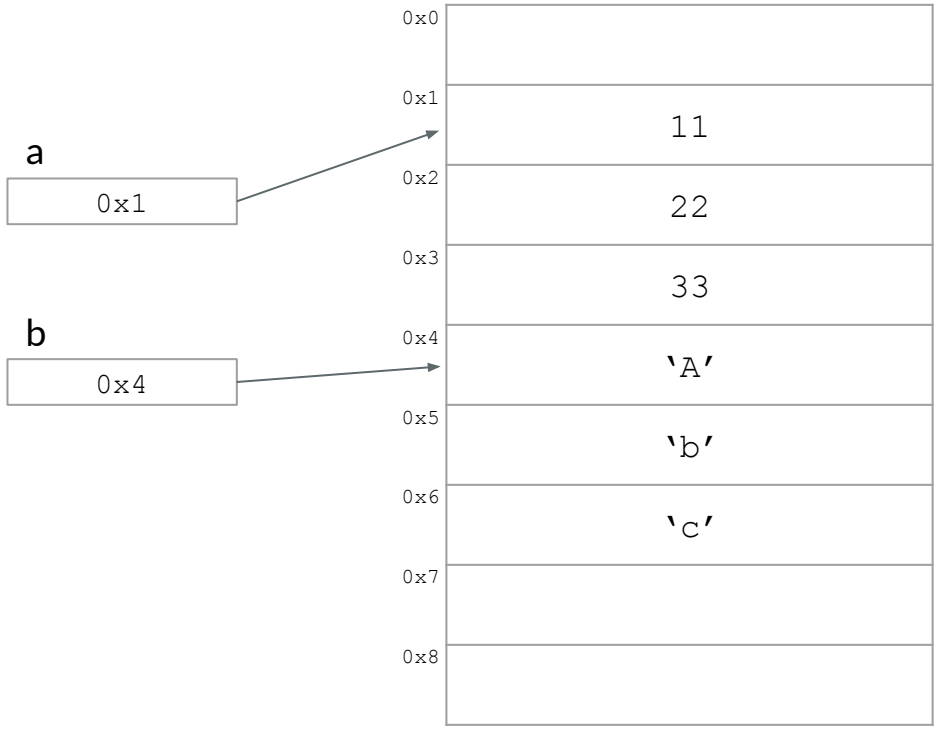
Annotate on the screen!

```
uint8_t a[] = {11, 22, 33};
char    b[] = {'a', 'b', 'c'};

// What happens to memory
// after this line?
a[3] = 65;
```



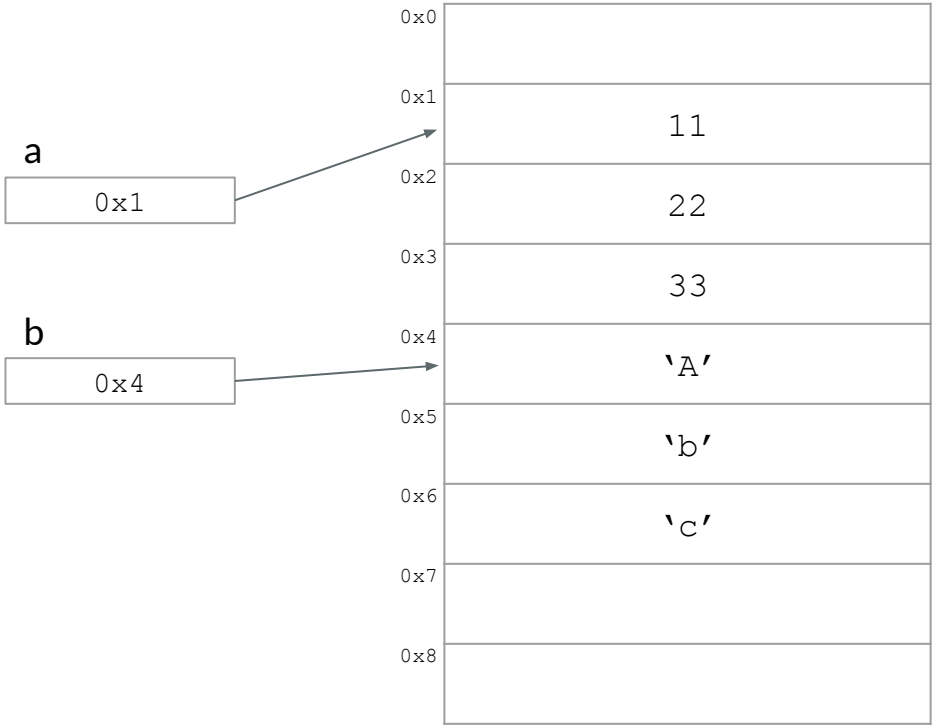
Annotate on the screen!

```
uint8_t a[] = {11, 22, 33};
char    b[] = {'a', 'b', 'c'};
a[3] = 65;

// What is the output?
printf("%c", b[0]);
```



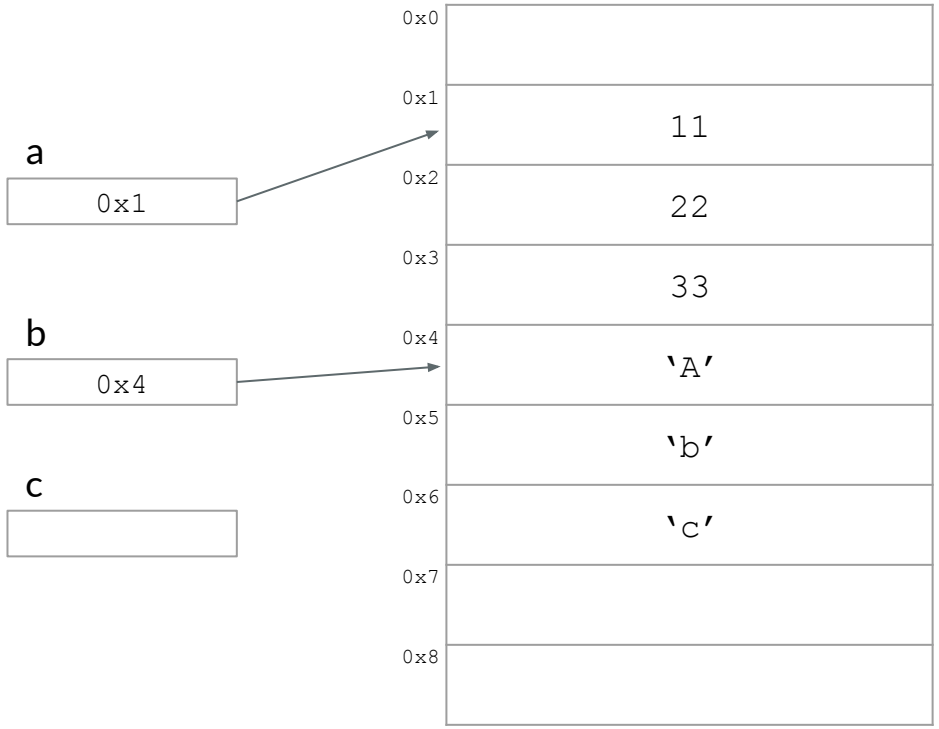
Annotate on the screen!

```
uint8_t a[] = {11, 22, 33};
char    b[] = {'a', 'b', 'c'};
a[3] = 65;

// What is the output?
printf("%d %d", a[1],
(int)&a[1]);
```



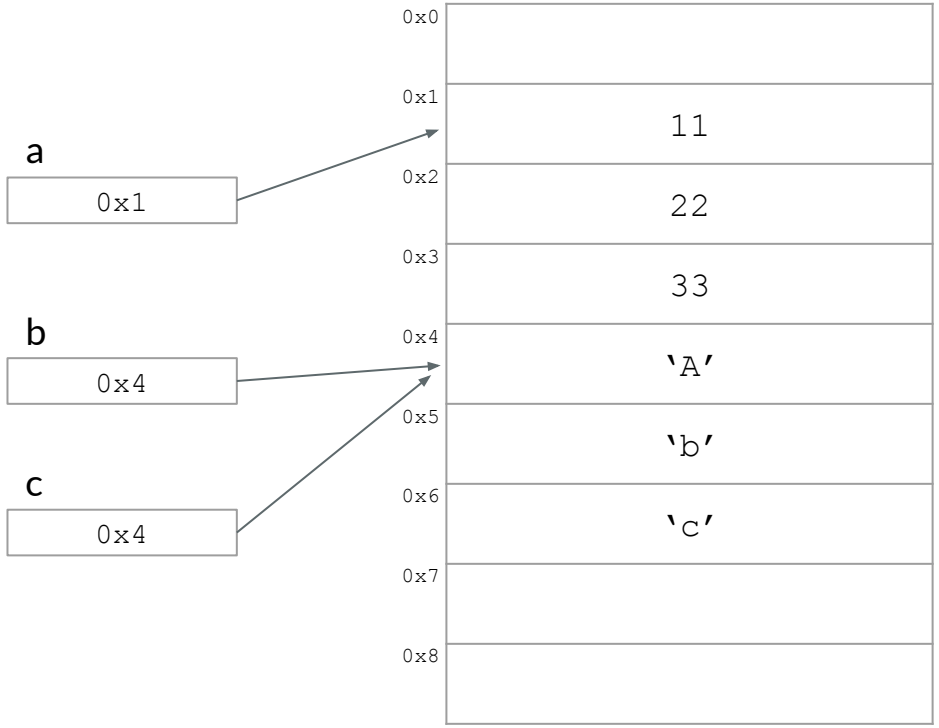
Annotate on the screen!

```
uint8_t a[] = {11, 22, 33};
char    b[] = {'a', 'b', 'c'};
a[3] = 65;

// What is in c and
// where does it point?
char    *c = b;
```



Annotate on the screen!

```
uint8_t a[] = {11, 22, 33};
char    b[] = {'a', 'b', 'c'};
a[3] = 65;
char    *c = b;

...
// What should memory look like
// so that this prints "Ab"
printf("%s\n", c);
```



malloc/free

- Similar to new in Java
- You need malloc when you need to dynamically allocate memory
  - What does it mean to dynamically allocate memory?
- When might you need malloc in p1?
- What is the difference between malloc and calloc?
- What is free, and is it necessary?

```
void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
```

Notes on p1

- The Makefile will compile both C and C++ code - but that means we will end up with 2 main functions!
  - Solution 1: rename main.cxx & slice.cxx to .cpp so they get ignored by the Makefile
  - Solution 2: delete the cxx files to make your code, and run `git clone git@git.gheith.com:cs429h_s24_p1` to get a copy of the starter code
  - Be sure to run `make clean` after making this change!
- The Makefile has a typo! (line 39)
  - `- ${CC} -MMD -MF $B/$*.d -c -o $@ ${C_FLAGS} $*.c`
  - `+ ${CC} -MMD -MF $B/$*.d -c -o $@ ${CC_FLAGS} $*.c`
  - Now you can change CC_FLAGS on line 6 to include debugging symbols
  - `- CC_FLAGS=-Wall -Werror -std=c99`
  - `+ CC_FLAGS=-Wall -Werror -std=c99 -g`
- The grading server will use its own Makefile to compile and run your code
  - We will make sure it works for us - but fixing your copy will help you debug locally

A Programming Languages Aside

- Put some stuff about how they'll be coming up with the fun spec here

Questions?