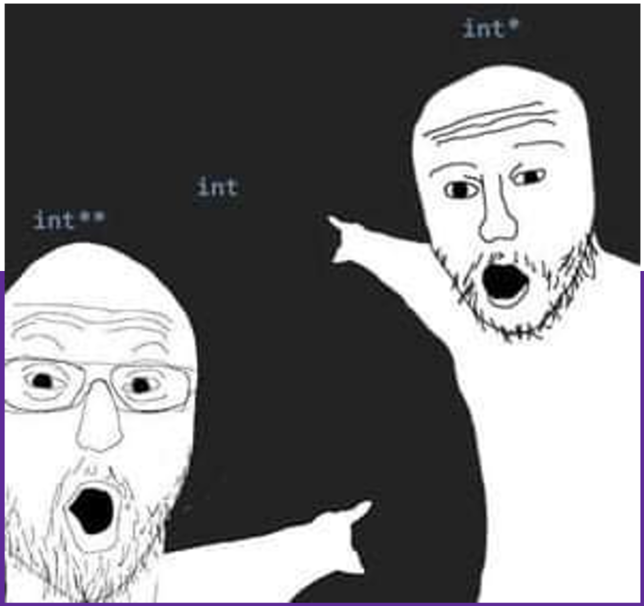
CSE 333 22au
Section 1

C, Pointers, and Gitlab



UNIVERSITY *of* WASHINGTON

Logistics

- Exercise 0:
  - Due **Friday @ 10:00am (9/30)**
- Homework 0:
  - Due **Monday @ 11:00pm (10/3)**
  - Meant to acquaint you to your repo and project logistics

Icebreaker!

Please turn to the people next to you and share:

- Name and Year
- What are you excited about in 333?
- Favorite type of peanut butter (Creamy, Chunky, no preference/allergic)

Setting Up

gcc 11

- CSE Lab machines and the attu cluster have been updated to use `gcc 11`.
- As such we'll be using `gcc 11` this quarter
- To verify that you're using `gcc 11` run:
  - `gcc -v` or
  - `gcc --version`

- If you use the CSE Linux home VM, you need to use the new 22au version even if you have an older one installed.

git/Gitlab Reference

Please take out your devices and follow along ☺️

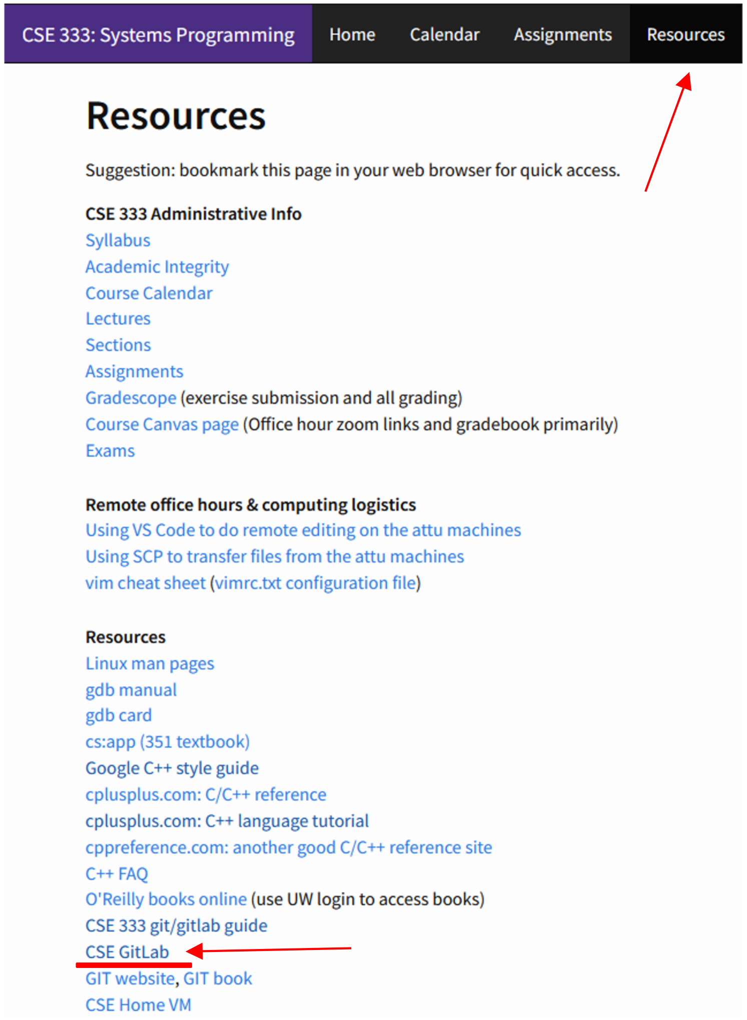
We have a page detailing the process of setting up Gitlab and git!

https://courses.cs.washington.edu/courses/cse333/22au/resources/git_tutorial.html

We'll be following this document during our demo.

Accessing Gitlab

- Sign-in using your **CSE NetID** @ https://gitlab.cs.washington.edu/
- There should be a repo created for you titled: `cse333-22au-<netid>`
- Please let us know if you don't have one!



Resources

Suggestion: bookmark this page in your web browser for quick access.

CSE 333 Administrative Info
Syllabus
Academic Integrity
Course Calendar
Lectures
Sections
Assignments
Gradescope (exercise submission and all grading)
Course Canvas page (Office hour zoom links and gradebook primarily)
Exams

Remote office hours & computing logistics
Using VS Code to do remote editing on the attu machines
Using SCP to transfer files from the attu machines
vim cheat sheet (vimrc.txt configuration file)

Resources
Linux man pages
gdb manual
gdb card
cs:app (351 textbook)
Google C++ style guide
cplusplus.com: C/C++ reference
cplusplus.com: C++ language tutorial
cppreference.com: another good C/C++ reference site
C++ FAQ
O'Reilly books online (use UW login to access books)
CSE 333 git/gitlab guide
CSE GitLab
GIT website, GIT book
CSE Home VM

SSH Key Generation

Step 1a) See if you have an existing SSH key

- Run `cat ~/.ssh/id_rsa.pub`
- If you see a long string starting with `ssh-rsa` or `ssh-dsa` go to Step 2

Step 1b) Generate a new SSH key

- If you don’t have an existing SSH key, you’ll need to create one
- Run `ssh-keygen -t rsa -C "<netid>@cs.washington.edu"` to generate a new key
- Hit enter to skip creating a password
    - git docs suggest creating a password, but it’s overkill for CSE333

Adding your SSH key to Gitlab

Step 2) Copy your SSH key

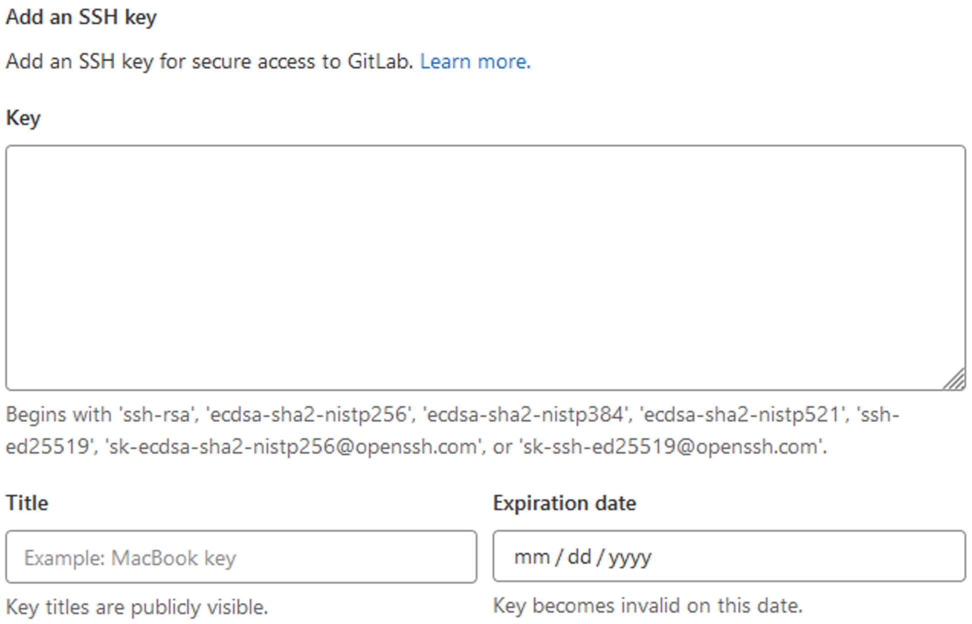
- Run `cat ~/.ssh/id_rsa.pub`
- Copy the complete key starting with `ssh-` and ending with your username and host (i.e. `<netid>@cs.washington.edu`)

Step 3) Add your SSH key to Gitlab

Adding your SSH key to Gitlab

Step 3) Add your SSH key to Gitlab

- Navigate to your ssh-keys page (click on your avatar in the upper-right, then “Preferences,” then “SSH Keys” in the left-side menu)
- Paste into the “Key” text box and give a “Title” to identify what machine the key is for
- Click the green “Add key” button below “Title”



Add an SSH key

Add an SSH key for secure access to GitLab. Learn more.

Key

Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'.

Title

Example: MacBook key

Key titles are publicly visible.

Expiration date

mm / dd / yyyy

Key becomes invalid on this date.

Setting up git

- The `git` command looks for a file named `.gitconfig` in your home directory. Some commands like `commit` and `push` expect certain options to be set and will produce verbose messages if not.
- If you have not already configured `git`, enter the following commands (once) in a terminal window to set these values:

```
git config --global user.name “<your name>”

git config --global user.email <your netid>@cs.washington.edu

git config --global push.default simple
```

First Commit

1. **`git clone <repo url from project page>`**
   a. Clones your repo
2. **`touch README.md`**
   a. Creates an empty file called `README.md`
3. **`git status`**
   a. Prints out the status of the repo: you should see 1 new file `README.md`
4. **`git add README.md`** **(or: `git stage README.md`)**
   a. Stages a new file/updated file for commit.
   ```
   git status: README.md staged for commit
   ```
5. **`git commit -m "First Commit"`**
   a. Commits all staged files with the provided comment/message.
   ```
   git status: Your branch is ahead by 1 commit.
   ```
6. **`git push`**
   a. Publishes the changes to the central repo.
   You should now see these changes in the web interface (may need to refresh).
7. Might need **`git push -u origin master`** on first commit (only), but would be unusual for this to happen

Git Repo Usage

Try to use the command line interface (not Gitlab's web interface)

Only push files used to build your code to the repo
- No executables, object files, etc.
- Don't always use `git add .` to add all your local files

Commit and push when an individual chunk of work is tested and done
- Don't push after every edit
- Don't only push once when everything is done

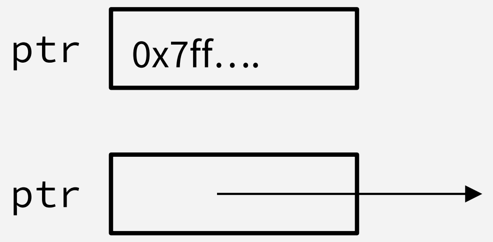
Pointer Review

Pointer Background

- Primitive data type
- Meant to store an address of a value/type (like keeping track of a location in memory)
- Often denoted with an arrow in memory diagrams

```
type* name;
```

```
int32_t* ptr;
```



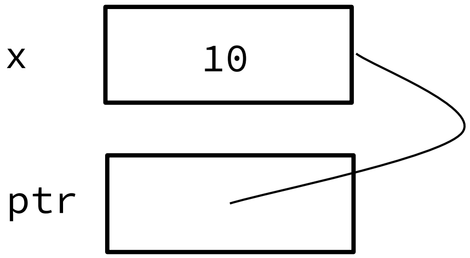
Pointer Syntax and Semantics

- How to get a variable's address (location in memory)?
  - Using the & operator
  - Getting the "address of"

- How to get the associated value of an address?
  - Using the * operator
  - Dereferencing memory

```
int32_t x;
int32_t* ptr;

ptr = &x;
x = 5;
*ptr = 10;
```

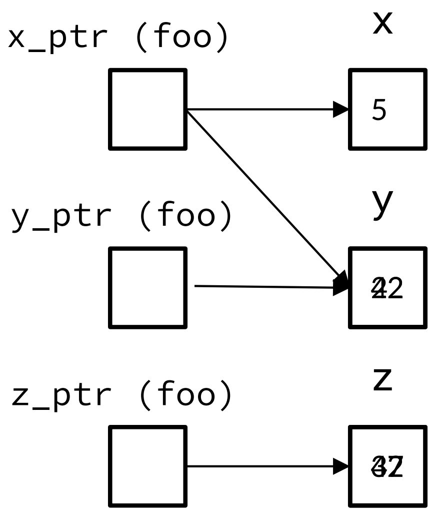


Exercise 1a

Draw a memory diagram like the one above for the following code and determine what the output will be.

```
void foo(int32_t* x_ptr, int32_t* y_ptr, int32_t* z_ptr) {
  x_ptr = y_ptr;
  *x_ptr = *z_ptr;
  *z_ptr = 37;
}

int main(int argc, char* argv[]) {
  int32_t x = 5, y = 22, z = 42;
  foo(&x, &y, &z);
  printf("%d, %d, %d\n", x, y, z);
  return EXIT_SUCCESS;
}
```



So, the code will output 5, 42, 37.

Function Pointers

Function Pointers

- Pointers can store addresses of functions
  - Functions are just instructions in read-only memory, their names are pointers to this memory.
- Used when performing operations for a function to use
  - Like a comparator for a sorter to use in Java
  - Reduces redundancy

```
int one()   { return 1; }
int two()   { return 2; }
int three() { return 3; }

int get(int (*func_name)()) {
  return func_name();
}

int main(int argc, char* argv[]) {
  int res1 = get(one);
  int res2 = get(two);
  int res3 = get(three);
  printf("%d, %d, %d\n", res1, res2, res3);
  return EXIT_SUCCESS;
}
```

Output Parameters

Output Parameters

- Idea: Not necessarily returning values through the **return** statement (`%rax` register)
  - Rather it is changing a location in memory to be another value
  - Manipulating the stack

- Output Parameters is an C idiom in order to emulate “returning values” through parameters
  - Call the function with a parameter that takes in a pointer, or an “address of” a variable
  - This will give a location in memory to change inside of the called function
  - The function will dereference that location and change it to give you a “returned” value

- This is particularly helpful for returning **multiple values**

Output Parameter Example

- Which of the following act as returning a value back to main?

  `quotient` and `remainder`

- What gets printed?

  `4, 2`

```c
void division(int32_t num, int32_t den,
                 int32_t* quotient,
                 int32_t* remainder) {
  *quotient = num / den;
  *remainder = num % den;
}


int main(int argc, char* argv[]) {
  int32_t num = 22, den = 5, quot, rem;
  division(num, den, &quot, &rem);
  printf("%d, %d\n", quot, rem);
  return EXIT_SUCCESS;
}
```

C-Strings

C-Strings

```c
char str_name[size];
```

- A string in C is declared as an **array of characters** that is terminated by a null character `'\0'`.

- When allocating space for a string, remember to add an extra element for the null character.

Initialization Examples

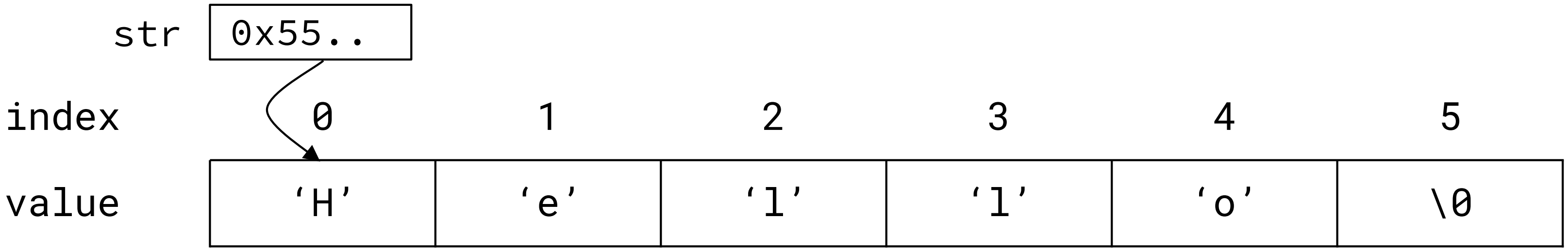
```c
char str[6] = {'H','e','l','l','o','\0'};  // list initialization
char str[6] = "Hello";        // string literal initialization
```

index	0	1	2	3	4	5
value	'H'	'e'	'l'	'l'	'o'	'\0'

- Both initialize the array *in the declaration scope* (*e.g.*, on the Stack if a local var), though the latter can be thought of copying the contents from the string literal.
  - The size 6 is **optional**, as it can be inferred from the initialization.

String Literal Example

```
char* str = "Hello";
```



str 0x55..

index	0	1	2	3	4	5
value	‘H’	‘e’	‘l’	‘l’	‘o’	\0

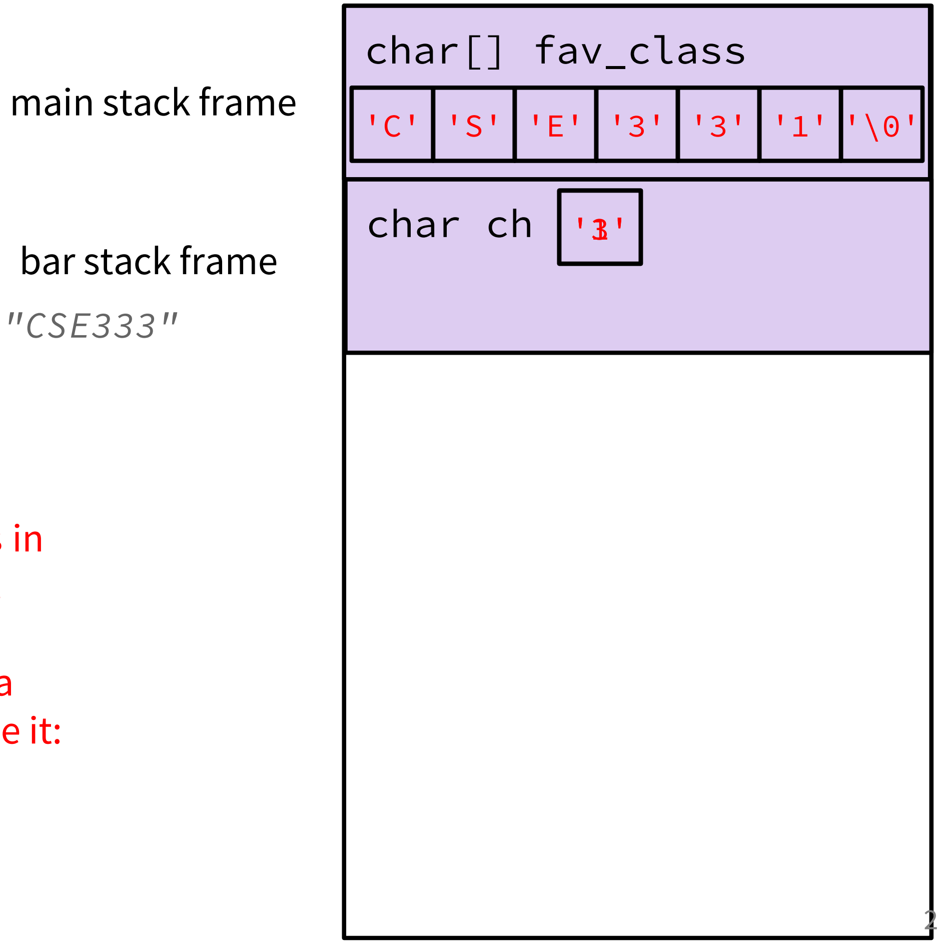
- By default, using a string literal will allocate and initialize the character array in *read-only* memory and the expression will return the *address of the array*, which can be stored in a pointer.

Exercise 1b

The following code has a bug. What's the problem, and how would you fix it?

```
  void bar(char ch) {
  ➡ch = '3';
➡}

  int main(int argc, char* argv[]) {
  ➡char fav_class[] = "CSE331";
  ➡bar(fav_class[5]);
  ➡printf("%s\n", fav_class);  // should print "CSE333"
    return EXIT_SUCCESS;
  }
```



Modifying the argument ch in bar will not affect fav_class in main() because arguments in C are always passed by value.

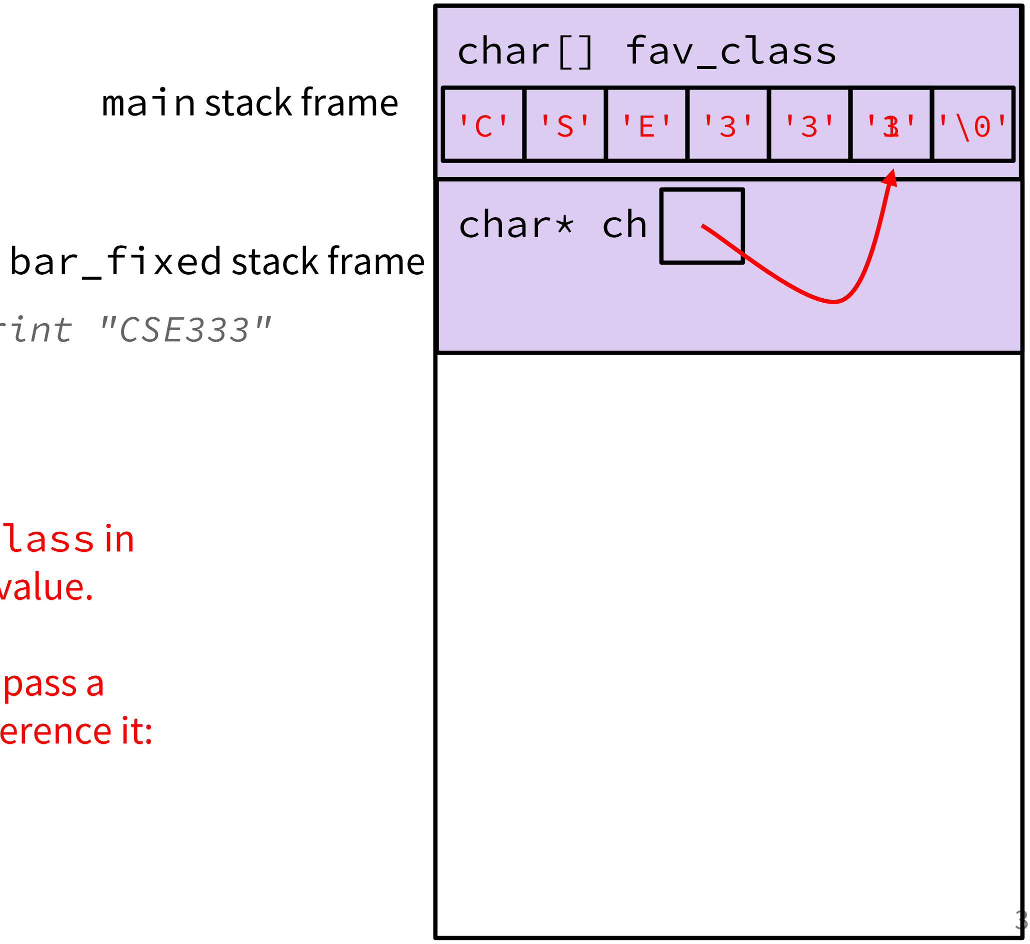
In order to modify fav_class in main(), we need to pass a pointer to a character (char*) into bar and then dereference it:

```
void bar_fixed(char* ch) {
  *ch = '3';
}
```

The following code has a bug. What's the problem, and how would you fix it?

```
void bar_fixed(char* ch) {
  *ch = '3';
}

int main(int argc, char* argv[]) {
  char fav_class[] = "CSE331";
  bar(&fav_class[5]);
  printf("%s\n", fav_class);  // should print "CSE333"
  return EXIT_SUCCESS;
}
```



Modifying the argument ch in bar will not affect fav_class in main() because arguments in C are always passed by value.

In order to modify fav_class in main(), we need to pass a pointer to a character (char*) into bar and then dereference it:

```
void bar_fixed(char* ch) {
  *ch = '3';
}
```