CSE 333 – SECTION 1

Introduction to and Working with C

Your TAs

- Sunjay Cauligi – Undergraduate Student, CSE
- Meghan Cowan – Undergraduate Student, CSE
- Renshu Gu – Graduate Student, EE
- Joshua Rios – Undergraduate Student, CSE
- Soumya Vasisht – Graduate Student, AA

- Email are posted on the course website
  - But try to use the staff email instead of our individual emails
- Office hours are posted

- Please use the discussion board!

Questions, Comments, Concerns

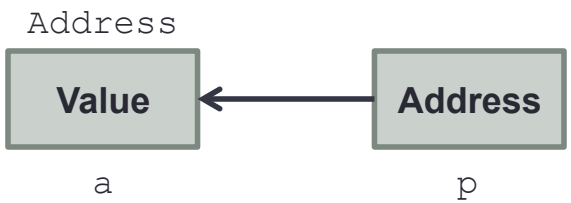
- Do you have any?
- Exercises going ok?
- Lectures make sense?

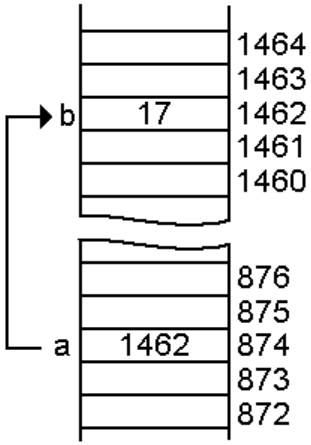
Quick Refresher on C

- General purpose programming language
- Procedural
- Often used in low-level system programming
- Supports use of pointer arithmetic
- Provides facilities for managing memory
- C passes all of its arguments by value
  - Pass-by-reference is simulated by passing the address of a variable

Pointers

- A data type that stores an address
- Used to indirectly refer to values
- Can add to or subtract from the address
  - It's just another number

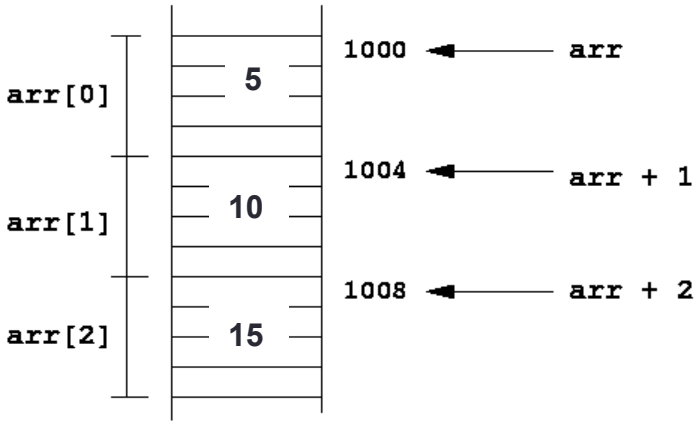




Arrays and pointers

- arr[0] <==> *arr
- arr[2] <==> *(arr + 2)

- How about arr, arr+2, *arr+2 or *arr++?



Output parameters

- C parameters are pass-by-value
- What if you want to modify a passed in parameter?
  - Why would this be useful in the first place?
  - Multiple return values

Output parameters

```
void make4_v1(int i) {
  i = 4;
}

void make4_v2(int *i) {
  int j = 4;
  i = &j;
}

void make4_v3(int *i) {
  *i = 4;
}
```

See also: `[output_params.c]`

Example

[basic_pointer.c]

```
#include <stdio.h>
void f(int *j) {
  (*j)++;
}
int main() {
  int i = 20;
  int *p = &i;
  f(p);
  printf("i = %d\n", i);
  return 0;
}
```

Pointers to pointers

```
 45   55   56   57   58   59   60   61   62   63   64   65   66   67   68   69
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    | 58 |    |    | 63 |    | 55 |    |    | h  | e  | l  | l  | o  | \0 |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
```

```
char *c = "hello";
char **cp = &c;
char ***cpp = &cp;
```

- Why could this be useful?

Function pointers

- We can have pointers to functions as well
- Syntax is a little awkward
  - Example: `int (*ptr_to_int_fn)(int, int)`
  - Makes sense if you think about it hard
- We will be using these in the homework assignments!
- Demo: `[function_pointer.c]`

Debugging with gdb

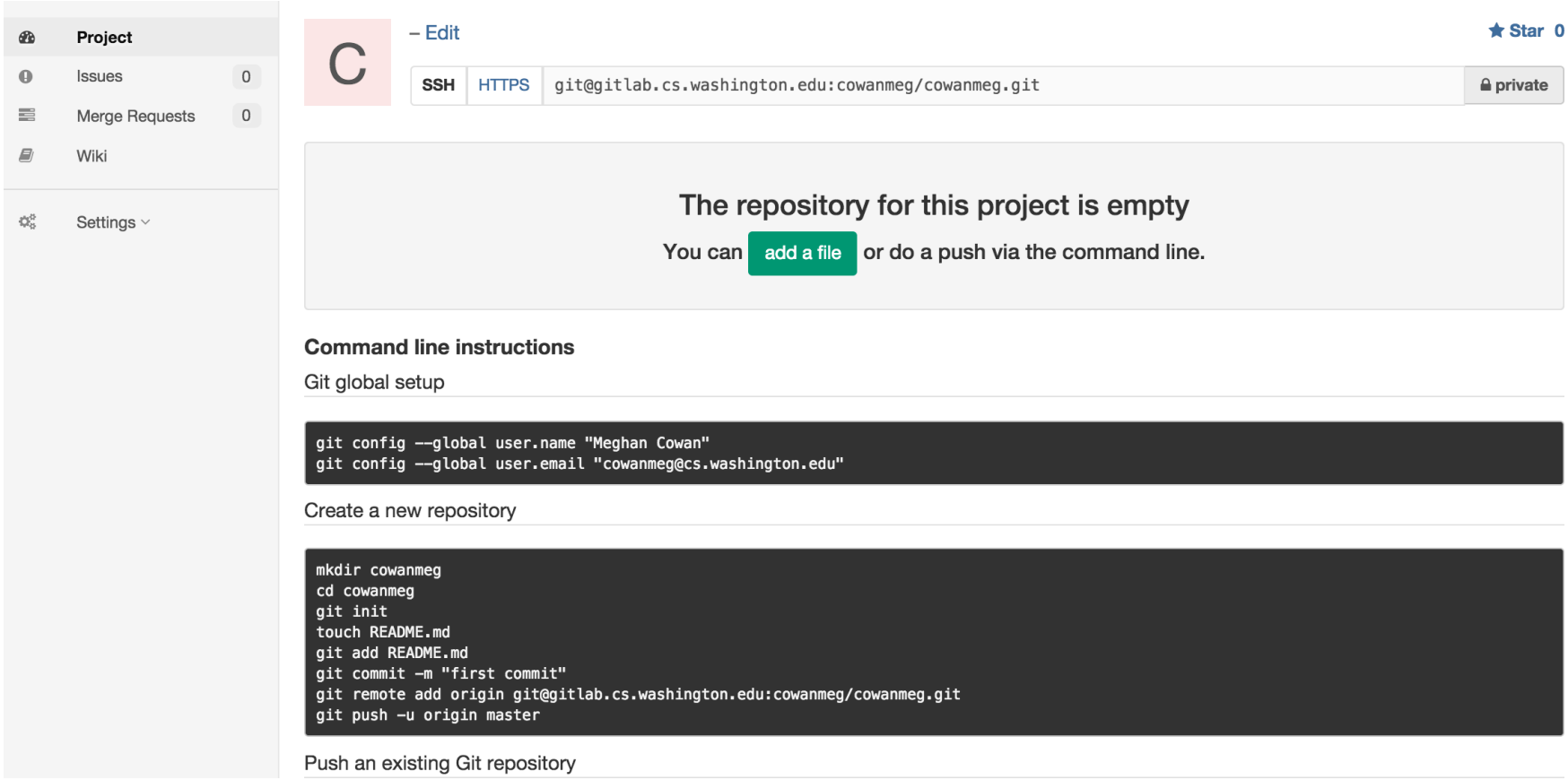
- Just like in CSE 351, gdb is your friend
- Unlike CSE 351, we will be debugging C/C++ code, not assembly
  - Instead of `n(ext)i` and `s(tep)i`, use `n(ext)` and `s(tep)`
- Your first instinct for bug fixing should be gdb, not printf
- If you want something a little more friendly, use `gdb -tui`
  - It's pretty darn helpful!
- Demo: `[buggy.c]`

Looking up documentation

- Don’t go straight to Google / Stack Overflow / etc.
- Use the built-in man pages
  - `man <program/utility/function>`
  - `man -f <name>` or `whatis <name>`
  - `apropos <keyword>`
- Much more documentation is linked on the 333 home page
  - Under “Resources” on the left side of the page

Gitlab Intro - Sign In

- Sign In using your **CSE netID**
- https://gitlab.cs.washington.edu/
- Most of you should have repos created for you (if you don't e-mail us)



SSH Key Generation

- Step 0: Check if you have a 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 1: Generate a new SSH key
  - Run **ssh-keygen -t rsa -C "$your_e-mail"** to generate a new key.
  - Click enter to skip creating or a password or create one (good practice) when prompted.
- Step 2: Copy SSH key
  - run **cat ~/.ssh/id_rsa.pub**
  - Copy the complete key key starting with ssh- and ending with your username and host
- Step 3: Add SSH key to gitlab
  - Navigate to your ssh-keys page (In the top menu bar click on profile then SSH Keys in the side menu)
  - Click the green 'Add SSH Key' button in the right corner.
  - Paste into the Key text box and lave the Title text box blank.

First Commit

- **git clone <repo url from project page>**
  Clones your repo
- **touch README.md**
  Creates a file called README.md
- **git status**
  Prints out the status of the repo.
  Should see 1 new file README.md
- **git add README.md**
  Stages a new file/updated file for commit.
  git status: README.me staged for commit
- **git commit -m "First Commit"**
  Commits all staged files with the comment in quotes.
  git status: Your branch is ahead by 1 commit.
- **git push -u origin master (FIRST COMMIT ONLY) / git push (NORMAL)**
  Publishes the changes to the central repo.
  You should now see these changes in the web interface.

References

- SSH Key generation:
  http://doc.gitlab.com/ce/ssh/README.html
- Basic Git Tutorial:
  http://courses.cs.washington.edu/courses/cse401/15wi/project/git.html