CS 220: Introduction to Parallel Computing

Functions and Pointers

Lecture 3

Today's Agenda

- Submitting Assignments with GitHub
- Functions and Pointers
- Wrapping up HW0 (if needed)

Today's Agenda

- **Submitting Assignments with GitHub**
- Functions and Pointers
- Wrapping up HW0 (if needed)

git

- In 220, we'll be using **git** to manage our code
- git (and source control in general) is used extensively in industry, science, and more
- GitHub is a service that wraps a nice web interface and hosting platform around git
- For some tips, see the course schedule page

Homework/Project Submission

- To submit your work, you'll upload it to GitHub
- There are several ways to accomplish this, including:
  1. The web interface
  2. The desktop client
  3. Command line
- As long as your changes show up on the site, that's all you need to do to submit!
  - We'll collect the git **repositories** after the deadline

Demo: git workflow

Today's Agenda

- Submitting Assignments with GitHub
- **Functions and Pointers**
- Wrapping up HW0 (if needed)

C Functions

- Functions are defined in C like this:

```
<return type> <function name>(<argument list>) {
     ...
}
```

- If the function does not return a value, the return type is void
- If there are no arguments, then the argument list is void (not required)

Calling a Function

What happens when we call a function?

1. System makes a note of the return address
2. Storage is set up for “formal args”
3. “Actual” args are copied into formal args
4. Control branches to first statement of function
5. Execute function!
6. Copy return value into memory
7. Jump back to the return address

Formal vs. Actual Arguments

- "Formal" arguments are specified in your function:

```
void location(int x, int y);
```

- "Actual" arguments are the actual (raw) values passed into the function:

```
location(2, 4);
```

Passing by Value

- In C, everything is passed **by value**
- This means that when you call a function, like:

```
location(2, 4);
```

- **Copies** will be made of 2 and 4 and passed to the location function
- Changing these values inside the function doesn't have an impact elsewhere
  - They are internal to the function

Passing by Reference (1/2)

- Sometimes we actually do want to change the value of a variable when it's passed into a function:

```
int a = 3;
int b = 8;
printf("%d, %d\n", a, b);
swap(a, b);
printf("%d, %d\n", a, b);
```

- Prints:

```
3, 8
8, 3
```

Passing by Reference (2/2)

- We need to pass by **reference**
- In C, we accomplish this by passing in the **memory address** of the variable:
  - The address is passed by value
  - We can use the address to find the variable in memory and change it
- If you have ever heard of **pointers** in C, this is what they're used for!

New Syntax

- & - the 'address of' operator. When a function takes a pointer as an argument, you need to give it an address, not the value of the variable
- int * x_p; - defining a pointer. Note that this doesn't create an integer, it creates a **pointer to an integer**.
- Finally when accessing a variable, *x_p is the **dereference** operator – it follows the address and looks up the actual value being pointed to
  - Unfortunately, this looks the same as defining it!

Demo: Passing by Reference

Today's Agenda

- Submitting Assignments with GitHub
- Functions and Pointers
- **Wrapping up HW0 (if needed)**

HW0

- Use this time to check your HW0 submission
  - It's due Monday, just in case you ran into problems with your setup
    - In that case, stop by Monday office hours or use the time now to debug
- If you're already done with HW0, then head out and have an awesome weekend!
  - But double check that it's actually up on GitHub first ☺