CS 220: Introduction to Parallel Computing

Beginning C

Lecture 2

Today's Schedule

- More C Background
- Differences: C vs Java/Python
- The C Compiler
- HW0

Today's Schedule

- **More C Background**
- Differences: C vs Java/Python
- The C Compiler
- HW0

Architectural Differences

- C is a bit different than Java or Python
- It is **compiled** to machine code
  - Java runs on a virtual machine (JVM)
  - Python is interpreted
    (translated to machine code on the fly)
- We can achieve better performance with C, but are also given more responsibility
  - Memory management is up to us
    (no automatic garbage collection)

C Advantages

- It is fairly simple: the language does not have a multitude of features
  - Coming from Java, the syntax is familiar
- In cases where we operate close to the hardware, it can be much easier to implement than the equivalent Java/Python/etc.
  - Wide use for systems programming
  - Want to contribute to the Linux kernel? It's written in C (including the drivers)
- Performance

C Disadvantages

- Much less functionality is available in the standard library than other languages
- Memory leaks
- Segmentation faults (invalid memory access)
- No objects
  - If you're used to object-oriented programming in Java or Python, C will make you rethink your program flow

Hello World in C

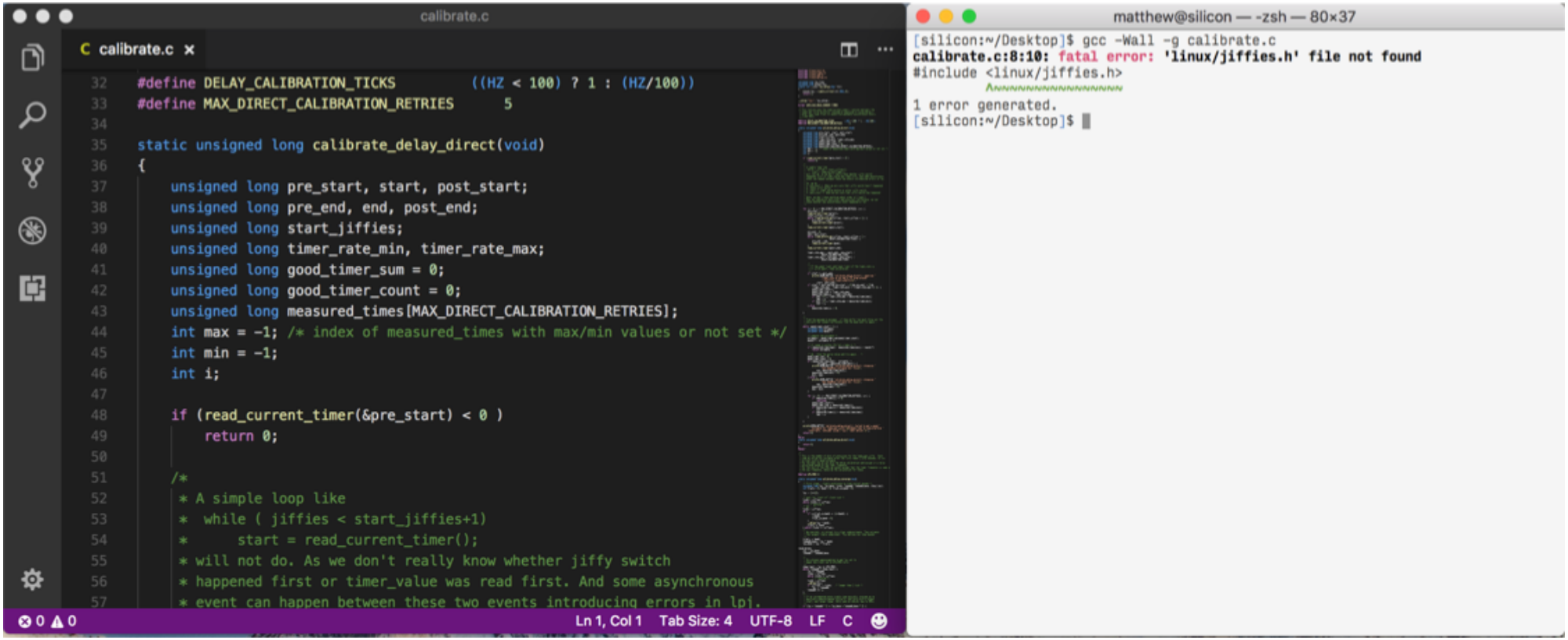
```c
#include <stdio.h>


int main(void)
{
    printf("Hello, World!\n");
    return 0;
}
```

Writing C Programs

- Using an IDE (like Eclipse, IntelliJ, etc) is less common in the C world
- Many C developers prefer to use a text editor and a terminal to write their programs
  - Text editor: edit, save
  - Terminal: compile, run
- There's a tutorial on the course schedule page for setting up your editor and C compiler

Writing C Programs



Testing Your Code

- **Very Important:** compile ***and*** test your code on the department machines before turning it in
  - We can't grade it on your specific laptop
- C compilers can implement the C specification differently
  - The standards committee releases new specifications periodically
  - In fact, in olden times, there were several different, incompatible versions of C

Windows

- One last tip: developing C programs on Windows can be tricky
  - What works on Windows may not work at all on the department Linux machines
- The course website has information for setting up a Linux virtual machine on Windows
- There are also other options available… use them at your own risk!

Today's Schedule

- More C Background
- **Differences: C vs Java/Python**
- The C Compiler
- HW0

A Program in C – Spot the Differences

```c
#include <stdio.h>

void say_hello(int times);

int main(int argc, char *argv[]) {
    say_hello(6);
    return 0;
}

/* Say Something */
void say_hello(int times) {
    int i;
    for (i = 1; i <= times; ++i) {
        printf("Hello world! (#%d)\n", i);
    }
}
```

Output:

```
Hello world! (#1)
Hello world! (#2)
Hello world! (#3)
Hello world! (#4)
Hello world! (#5)
Hello world! (#6)
```

Differences from Java/Python

- Whitespace is mostly ignored
  - Semicolons are **required**
- Comments: /* */ and //
- Including libraries looks a bit different
- No public/private etc. access modifiers
- Forward declarations (prototypes)
- But, there are a lot of similarities…

Similarities

- Arithmetic is mostly the same
- We use `&&`, `||`, and `!=` instead of `and`, `or` and `not`
- If, then, else
- Loops
- Switches

Today's Schedule

- More C Background
- Differences: C vs Java/Python
- **The C Compiler**
- HW0

Compilation

- Something you may not be familiar with is **compiling** your programs
  - Who has used **javac** and **java** from the command line?
  - …Who presses the "Run" button in Eclipse/IntelliJ?
- With C, the compiler is very important
- It takes your C code and transforms it into **machine code** to produce a program **binary**
  - Runs natively on the hardware – no VM/interpreter

Program Binaries

- After you've compiled your program and produced an executable binary, you can run it!
- You can even copy your program to other similar machines and it will run
  - Unlike Java/Python, you don't have to install anything first
- However, note "**similar**" above – the binaries are platform- and architecture-specific

Platform Differences

- Your compiled C program will generally only run on its target architecture and platform
- If you compiled on a Mac, then the binary won't work on Linux
- If you compile on an x86-based processor (Intel, AMD), the binary won't work on ARM (Qualcomm, Apple, Samsung mobile CPUs, Raspberry Pi…)
- Java/Python don't have this limitation!

Phases of C Compilation

1. ***Preprocessing***: perform text substitution, include files, and define macros. The first pass of compilation.
   - Directives begin with a #
2. ***Translation***: preprocessed code is converted to machine language (also known as **object code**)
3. ***Linking***: your code likely uses external routines (for example, printf from stdio.h). In this phase, libraries are added to your code

The C Preprocessor

- We've seen include statements:
  - `#include <stdio.h>`
- Another common use case is constants:
  - `#define PI 3.14159`
  - Note: no equals sign. This is just simple text replacement!
- You can also define **macros** that essentially cut and paste reusable code snippets into your work

Include Paths

- There are two types of includes:
  - `#include <blah>`
  - `#include “blah”`
- When angle brackets are used, the system-wide library paths are searched
- With quotes, you are specifying a **local** path (in the same folder as your code)
- In this class, you’ll only need to worry about the system libraries

Compiling from the Command Line

- `gcc my_code.c`
  `./a.out`
  Produces and runs a binary file called 'a.out'

- You can also turn on error messages:
  `gcc -Wall my_code.c`

- And give your program a name:
  `gcc -Wall my_code.c -o my_prog`

Making Diagnostics Readable

The last command line option to gcc I recommend is -fdiagnostics-color.

```
gcc -fdiagnostics-color -Wall my_code.c -o out.exe
```

```
 my_code.c:9:6: warning: conflicting types for ‘say_hello’
[enabled by default]
 void say_hello(int times) {
      ^
first.c:5:5: note: previous implicit declaration of
‘say_hello’ was here
     say_hello(6);
     ^
```

Today's Schedule

- More C Background
- Differences: C vs Java/Python
- The C Compiler
- **HW0**

Basic Input/Output

- Requires the standard I/O library:
  - `#include <stdio.h>`
- Printing text:
  - `printf("hi there!\n");`
- We can also print out variables with **format strings**

Format Strings (1/2)

- Let's look at a print example:
  - `printf("<format>", var1, var2, … , varN);`
- The variable list is optional:
  - `printf("hello world!\n");`
  - Note that we need to provide the **newline** character
- This style of I/O tells the C compiler **what** and **where** you want to read or write

Format Strings (2/2)

- The C compiler looks through your format string to determine the order to print and in what format:
  - `printf("Hello %s, it is January %d.", "Alice", 24);`
  - `Hello Alice, it is January 24.`
- There are several **format specifiers** available:
  - %d or %i – integer
  - %s – string
  - %f – floating point
  - … And many more

GitHub

- If you haven't already, register for an account on GitHub
- Visit the course website for homework instructions
  - See: Assignments → Homework
- The schedule page also has some information about using git