601.220 Intermediate Programming

Hello world program in C

Outline

- Compiling and running a simple C program

Having technical issues? Feeling stuck?

- Consult posted slides and tutorials for reference
- Don't be afraid to ask for help

Where to ask

- Ask questions on Piazza
- Instructor office hours will be announced soon
- Course assistants (CAs) will set office hours soon

Hello world

```c
// hello_world.c:
#include <stdio.h>

// Print "Hello, world!" followed by newline and exit
int main(void) {
    printf("Hello, world!\n");
    return 0;
}
```

```
$ gcc hello_world.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
Hello, world!
```

Basic C/C++ programming workflow

- Edit file (using emacs or vim) `hello_world.c`
- Compile using GNU C compiler (gcc) to compile, link, and create executable
  - If compile-time errors reported, edit .c file and re-compile
- Run the executable file
  - If run-time errors reported/detected, go back to edit step

Inside the compiler

- Step 1: preprocessor
  - Bring together all the code that belongs together
  - Process the directives that start with #, such as `#include`
    - We'll soon also see `#define`
- Step 2: compiler
  - Turn human-readable *source code* into *object code*
  - Might yield warnings & errors if your code has mistakes that are "visible" to compiler
- Step 3: linker
  - Bring together all the relevant *object code* into a single executable file
  - Might yield warnings & errors if relevant code is missing, there's a naming conflict, etc

Hello world

```c
// hello_world.c:
#include <stdio.h>

// Print "Hello, world!" followed by newline and exit
int main(void) {
    printf("Hello, world!\n");
    return 0;
}
```

- `#include` is a preprocessor directive, similar to Java `import`
- `main` is a function, every program has exactly one `main`
- `int` is its return type
- `main(void)` says that `main` takes no parameters

Hello world

```c
// hello_world.c:
#include <stdio.h>

// Print "Hello, world!" followed by newline and exit
int main(void) {
    printf("Hello, world!\n");
    return 0;
}
```

- `printf` prints a string to "standard out" (terminal)
- `\n` denotes the newline character
- `return 0` means "program completed with no errors"
- Explanatory comment before function is good practice
  - `// Print ...`

Basic C/C++ programming workflow

- To compile hello_world.c and link to give executable file:
  - `gcc -std=c99 -Wall -Wextra -pedantic hello_world.c`
- To run executable file named `a.out`:
  - `./a.out`

Hello world

What if we omit `#include <stdio.h>`?:

```
// hello_world_err.c:

// Print "Hello, world!" followed by newline and exit
int main(void) {
    printf("Hello, world!\n");
    return 0;
}
```

```
$ gcc hello_world_err.c -std=c99 -pedantic -Wall -Wextra
hello_world_err.c: In function ‘main’:
hello_world_err.c:4:5: warning: implicit declaration of function ‘print
    4 |     printf("Hello, world!\n");
      |     ^~~~~~
hello_world_err.c:4:5: warning: incompatible implicit declaration of bu
hello_world_err.c:1:1: note: include ‘<stdio.h>’ or provide a declarati
  +++ |+#include <stdio.h>
```

Multiple copies of code

- Imagine if every time you wanted to snapshot (or "branch" or share) your project, you made a copy

```
$ ls cs220-eg/
hello_world.c  Makefile  README

$ cp -r cs220-eg cs220-eg-2020-09-02

$ ls cs220-eg cs220-eg-2020-09-02
cs220-eg:
hello_world.c  Makefile  README

cs220-eg-2020-09-02:
hello_world.c  Makefile  README
```

Multiple copies of code

- Suppose you "snapshot" or share your code frequently, and at different times on different machines
  - Up to you to remember "meanings" and relationships of copies
  - Lots of copies = lots of space; waste, redundancy
  - Difficult for team members to track where the latest version of each file lives

Git to the rescue

- A *repository* ("repo") stores all versions of all project files, and their entire histories back to the beginning of the project
  - Repos eliminate disadvantages of the "lots of copies" method, while still facilitating snapshotting, branching, sharing
  - Cleverly organized to avoid storing redundant data
  - Repo (master/origin) can be *local* (on your computer) or *remote* (e.g., on bitbucket.org or github.com)