CS 240: Programming in C

Lecture 2: Compiling, Object Files, Linking, and Execution

PURDUE UNIVERSITY

Announcements

- Homework 1 will be released on Monday

Why C?

- It is still widely used *and growing*
    - TIOBE August 2024 index lists it as #3
    - Python is #1, C++ is #2
- Programming Language of the Year 2019
- Used in a huge number of embedded and IoT devices
- Ubiquitous for systems programming
- Small
- Fast

C vs. Java

- You already know how to write some C code
  - Java was designed using C/C++ style syntax
- But, there are *many* differences
  - See Lecture 1 slides

C

- Created by Dennis Ritchie 1969-1973 at Bell Labs
- Early operating systems typically implemented entirely in assembly
  - Not portable
- Desire to make UNIX portable
- With C, only about 5% is written in assembly
  - Much easier to port to different architectures

C

- The Linux kernel is written in C
  - So is most of Windows’ and Mac’s kernels
- Many libraries and programs are also in C
  - Especially if they need to be fast
  - GCC, GDB, Valgrind, OpenSSL, MySQL, Doxygen, GLFW, SDL, FFMPEG, libVLC, libmpv, curl, and many more…
- Embedded systems
- Firmware
- Drivers

Why C?

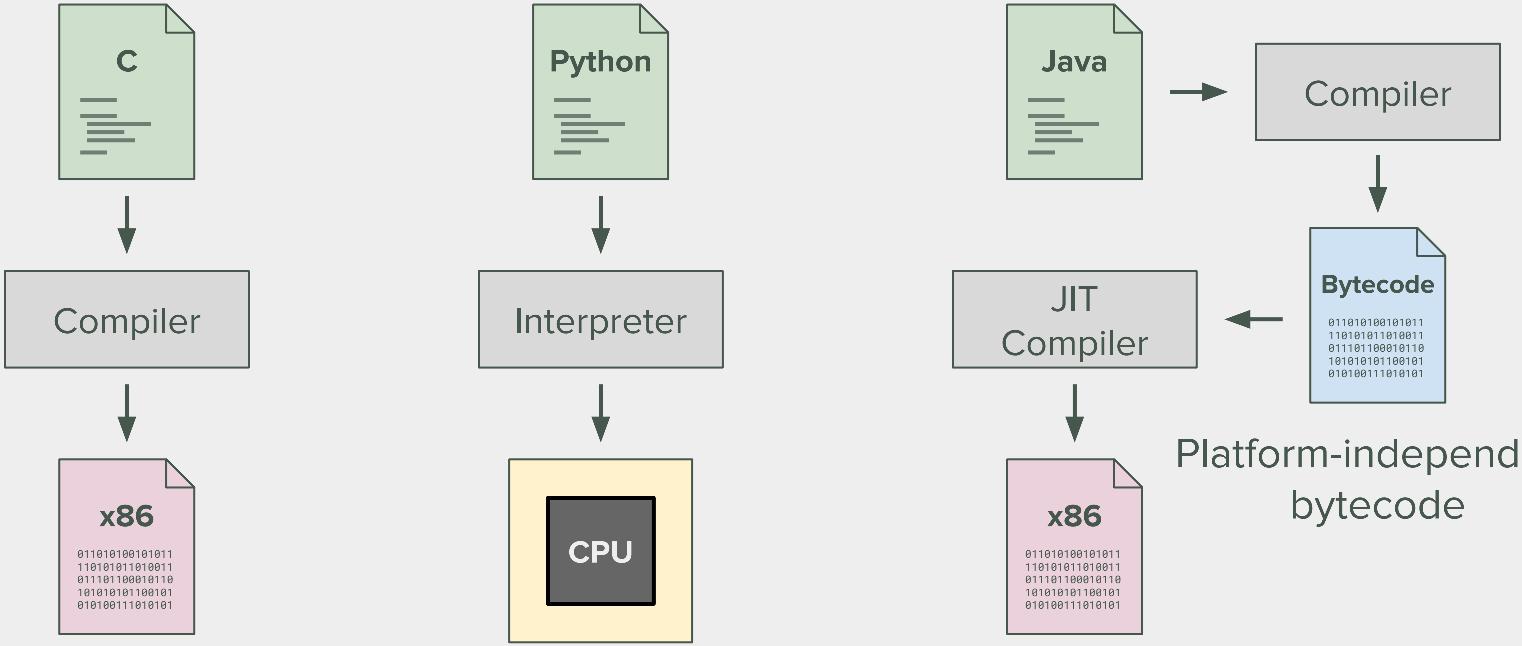
- It’s fast
- It’s powerful
- It’s simple
    - Easy to do low-level things
    - No abstractions to worry about

The C Standard

Continues to evolve

- K&R C (C78)
- ANSI C and ISO C (C89 and C90)
- C99
- C11 (C1X)
- C17
- C23

Compilation



```
011010100101011
110101011010011
011101100010110
101010101100101
010100111010101
```

Machine language

Executes directly on CPU

Machine language

Platform-independent bytecode

Compilation

- Many C compilers exist
    - GCC, Clang, MSVC
- In this course, we will be using **gcc** exclusively

gcc

- GNU Compiler Collection
- Standard compiler for most UNIX-like operating systems
- First released March 22, 1987
- Many different front ends: C, C++, Objective-C, Objective-C++, Fortran, ~~Java~~, Ada, Go

Common gcc flags

`-c`	Compile file into object code
`-g`	Include debug symbols
`-Wall`	Enable ALL warnings
`-Werror`	Turn warnings into errors
`-O`	Optimize the output file
`-o file`	Output to ‘file’
`-ansi`	Adhere to the ANSI standard
`-std=X`	Adhere to some standard X (e.g., C17)

Examples of flags

- Compile file.c into file.o, make it debuggable, and enable all warnings

```
$ gcc -g -Wall -c file.c
```

- Compile X.c into Y.o, no debugging, C99 standard

```
$ gcc -c X.c -std=c99 -o Y.o
```

- Compile and optimize

```
$ gcc -O -c file.c
```

What is an ‘object’ file?

- An object file is like an incomplete executable
  - It is the compiled form of a C module
  - It contains binary code
  - It contains a symbol table
  - Usually has a **.o** or **.obj** filename extension
- To create an executable from multiple object files, we need to **link** them together
- **One** object must contain `main()`
- gcc knows how to link objects too!

Examples of linking

- Compile two C files and link them together

```
$ gcc -Wall -Werror -g -c file1.c
$ gcc -Wall -Werror -g -c file2.c
$ gcc -o my_progr file1.o file2.o
```

- Could do the same thing in one step, without generating object files:

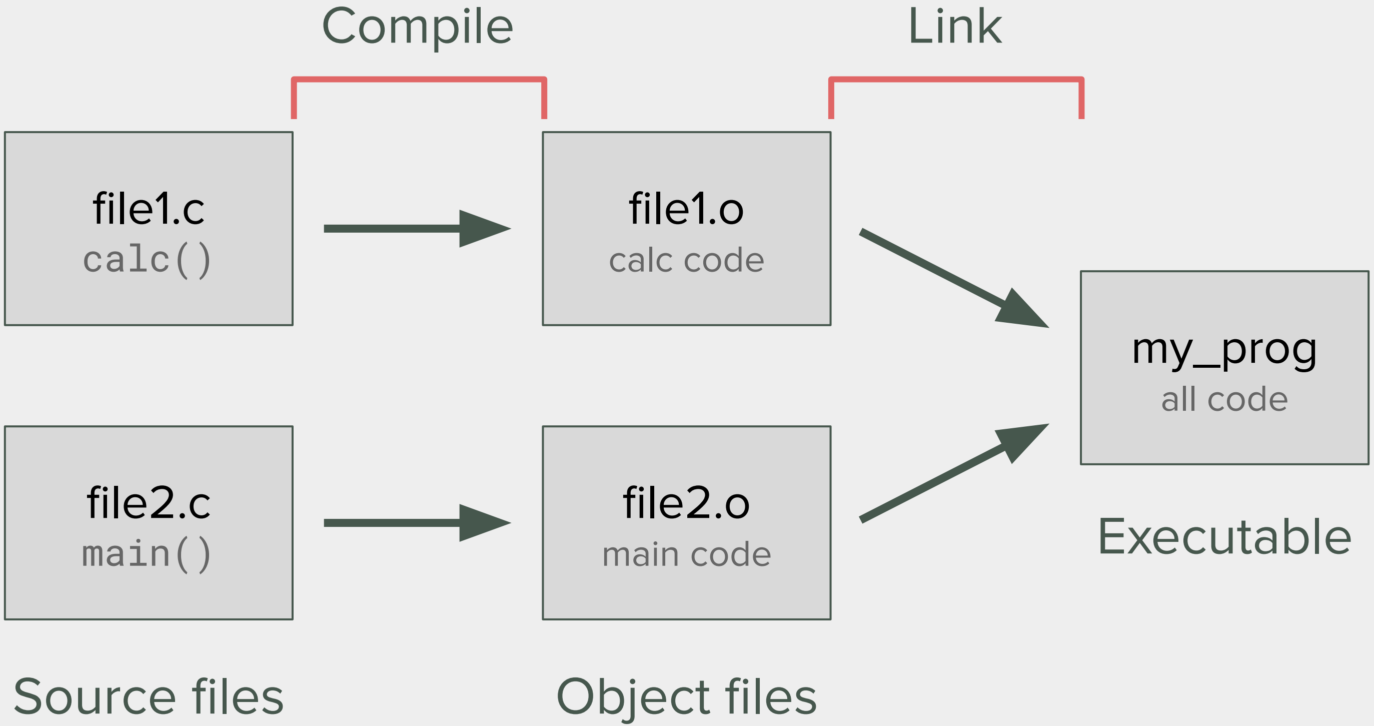
```
$ gcc -Wall -Werror -g -o my_prog file1.c file2.c
```

- Why would we want any of these object files?

Why object files?

- It takes a long time to compile “big” applications if they consist of lots of C files.
  - It’s better to do **incremental compilation** of the application
- You can give parts of programs to people without letting them see the source code
  - That’s the way your homework will be

Illustration of compile + link


Compile

Link

file1.c
`calc()`

file1.o
calc code

my_prog
all code

file2.c
`main()`

file2.o
main code

Executable

Source files

Object files

Execution

- If there were no errors compiling or linking your program, you can invoke it by typing its name:

```
$ gcc -Wall -Werror -c hello.c
$ gcc -o hello hello.o
$ ./hello
Hello, world!
```

Common errors

- When putting functions into separate modules, they need to have **prototypes** (forward declarations for functions)
  - Prevents type mismatches
  - A little extra bookkeeping for the programmer to make sure of types

file1.c

```c
float calc(float first_val, float second_val) {
   float temp = 0.0;

   temp = first_val * second_val;

   return temp;
}
```

file2.c

```
#include <stdio.h>

int main() {
    float result;
    result = calc(11.10, 3);

    printf(“My salary is $%f\n”, result);

    return 0;
}
```

file2.c with prototype

```
#include <stdio.h>

float calc(float first, float sec);

int main() {
    float result;
    result = calc(11.10, 3);

    printf(“My salary is $%f\n”, result);

    return 0;
}
```

Takehome Quiz!

- Assignment in Gradescope (Quizzes)
  - If you're not in Gradescope, **email me**
- Due 24 hours after lecture ends
- Quizzes **must** be hand written
  - No credit otherwise
- Use the template on the course webpage
- We'd really prefer you scan your quiz, if at all possible
  - But if you can't, a picture is okay

Takehome Quiz #0

1. Give one interesting fact about C
   - which was **not discussed** during lecture
2. What is your major? If you’re a CS major, what track are you in / decided on?
3. What did you do over the summer?

For Next Lecture

- Keep practicing
- Read Chapter 2 of K&R
  - Skip section 2.7
- Read Beej’s up through Chapter 3.3
  - Optional, but recommended
- Homework 1 will be released Monday!

Slides

- Slides are heavily based on Prof. Turkstra's material from previous semesters.