CS221
C and Systems Programming

Compiled vs Interpreted languages

Compiled	Interpreted
Run faster	Easier to develop code line-by-line
Use less memory	Usually platform independent
Run on more platforms	
C, C++, Java, Rust, Haskell[1], Fortran, Cobol, Ada, Swift[2], Go[3], C#[4]	Java, Python, Ruby, Lisp, R, Matlab, JavaScript, Bash

1. Has nothing to do with me!
2. Used by Apple
3. Used by Google
4. Used by Microsoft

Pick the right tool for your job

- I almost always use Python for text processing
- Java for graphics
- Matlab for numerical computation
- C++ for high-performance computing or "systems" work (e.g. "I really need 8 GB of memory" or "I need 2 programs to talk to each other in real time")

New C syntax

C syntax

Global variables
- Dangerous, since any code can modify them. But sometimes useful

Command line arguments
- `int argc, char** argv`
- `argv[0]` is name of program. `argv[1]` is first cmdline arg

Static
- Just as confusing in C as in Java
- static global variable or method: only can be used in current file
- static method variable: value is not lost between calls to the method (kinda like Java)

Const
- Just like Java's `final`

Show examples of each!

Pros/cons of "const" vs #define

Building new data types - struct

C does not have classes, but we can create `structs`

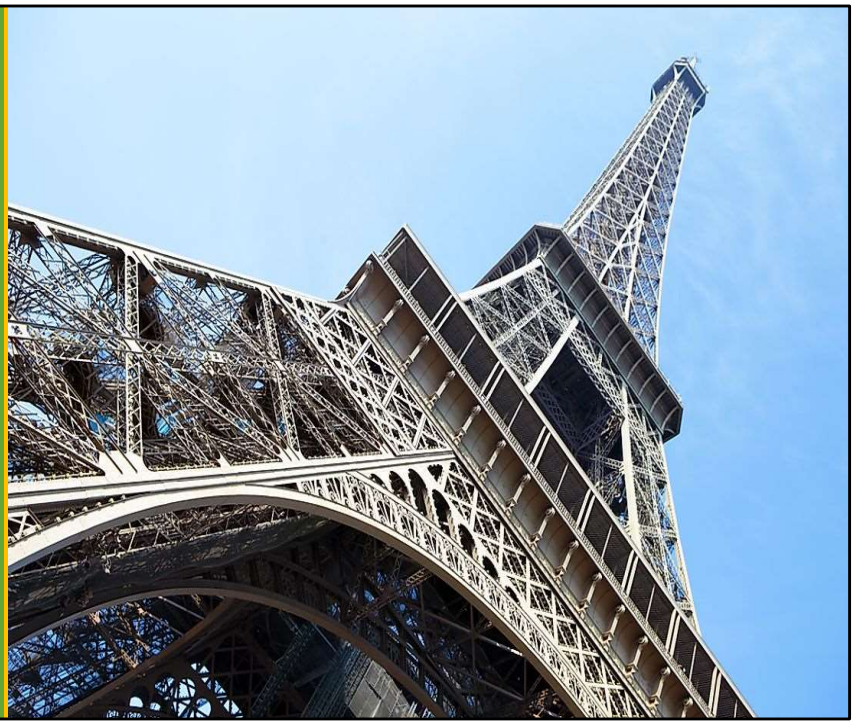
Contains multiple member variables

- But no methods
- And no public/protected/private access control

Use "." to access members, just like in Java

Do example: PlayingCard with value and suit.
Show how we create a variable, using "struct"

C project structure



C project structure

Include files (*.h) and source files (*.c)

- Include files:
  - declare but do not define global variables with `extern`
  - declare but do not define methods
  - declare data structures
- Each source file includes the *.h files it needs

- Can a *.h file include other *.h files?
- How do we prevent problems with duplicated declarations?

Show *.h file protection with #ifdef/#define/#endif

Example file with absVal() and clip() methods.

Should Makefile dependencies include *.h files? OF COURSE!

How does Java do this? (1) Automatically looks at ALL Java files in same directory, (2) import …

git new commands

8

Git basics

git pull	pull remote repo contents to local machine
git add	add ≥1 file to current "version"
git commit	"close" the current version
git push	push all local changes to remote repo

Git versions

Every time we call "git commit" we create a new version

We can list all previous versions, restore an old version to our computer, etc

Git creates safe cloud backups of our work

Look at old versions via "git log", browsing history on GitHub.com

Once a version is committed, it can't be changed. Just make a new version if needed

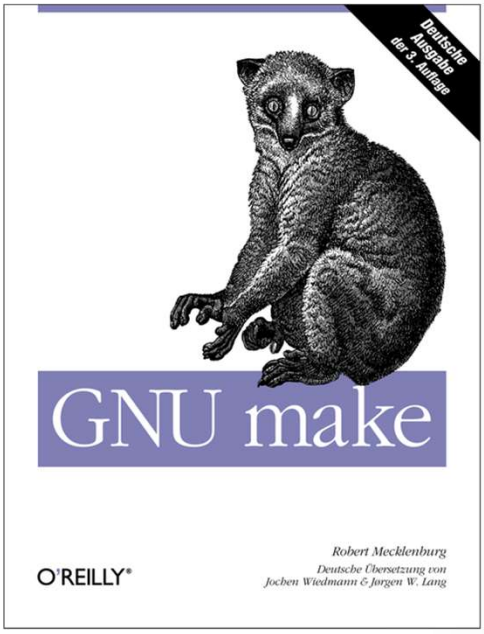
New git stuff

git fetch	Fetch all remote changes to our local database but do not change our local files. Always safe to run
git status	Show info on our local changes, on local vs remote status
git rm	Remove a file from in-progress version
git mv	Rename a file
.gitignore	Specify files on our local computer for git to ignore

Walk thru my marvelous .gitignore

"safe?" git pull may create merge conflicts

make and Makefiles

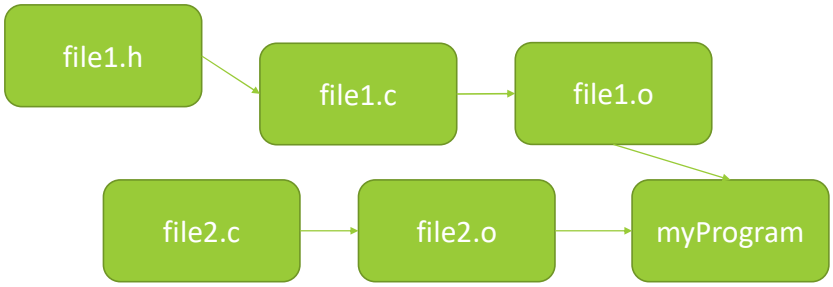


12

make

Tedious to keep typing in gcc command lines with lots of arguments

Waste of time to recompile files that have not changed.



Javac has lots of arguments too!

make

We write a **Makefile** with

- Dependencies and Targets
- Rules to create "downstream" targets from dependencies

Should ***.h** files be included as dependencies?

See example:
TABs before rules
comments
Fake targets: "all" "clean"

Show that rerunning "make" does not do anything, unless we change a dependency
NOTE: Windows always adds ".exe"