Practical C Programming

Advanced Preprocessor

- # - quotes a string
- ## - concatenates things
- #pragma
  - http://gcc.gnu.org/onlinedocs/cpp/Pragmas.html
- #warn
- #error

Defined Constants

Macro	Meaning
`__FILE__`	The currently compiled file
`__LINE__`	The current line number
`__DATE__`	The current date
`__TIME__`	The current time
`__STDC__`	Defined if compiler supports ANSI C
…	Many other compiler-specific flags

Multi-file Development

- Want to break up a program into multiple files
  - Easier to maintain
  - Multiple authors
  - Quicker compilation
  - Modularity

Header Files

- Should usually only contain declarations
  - Functions
  - #defined macros
  - Variables… **RARELY**
    - Usually a BAD idea, use **extern** instead
- Paired with an implementation file

File Scope

- "Global Variables" are actually limited to the file
- extern maybe be used to import variables from other files

File A

```
int x;
```

File B

```
extern int x;
```

Will refer to the same memory location

Example

a.c

```
int x = 0;

int f(int y)
{
        return x+y;
}
```

b.c

```
#include <stdio.h>

extern int x;
int f(int);

int main()
{
        x = 5;
        printf("%d", f(0));

        return 0;
}
```

Including a Header File Once

```
#ifndef _MYHEADER_H_
#define _MYHEADER_H_

…Definitions of header to only be included once



#endif
```

Makefiles

- Express what files depend upon others
- If any are modified, build smallest set required

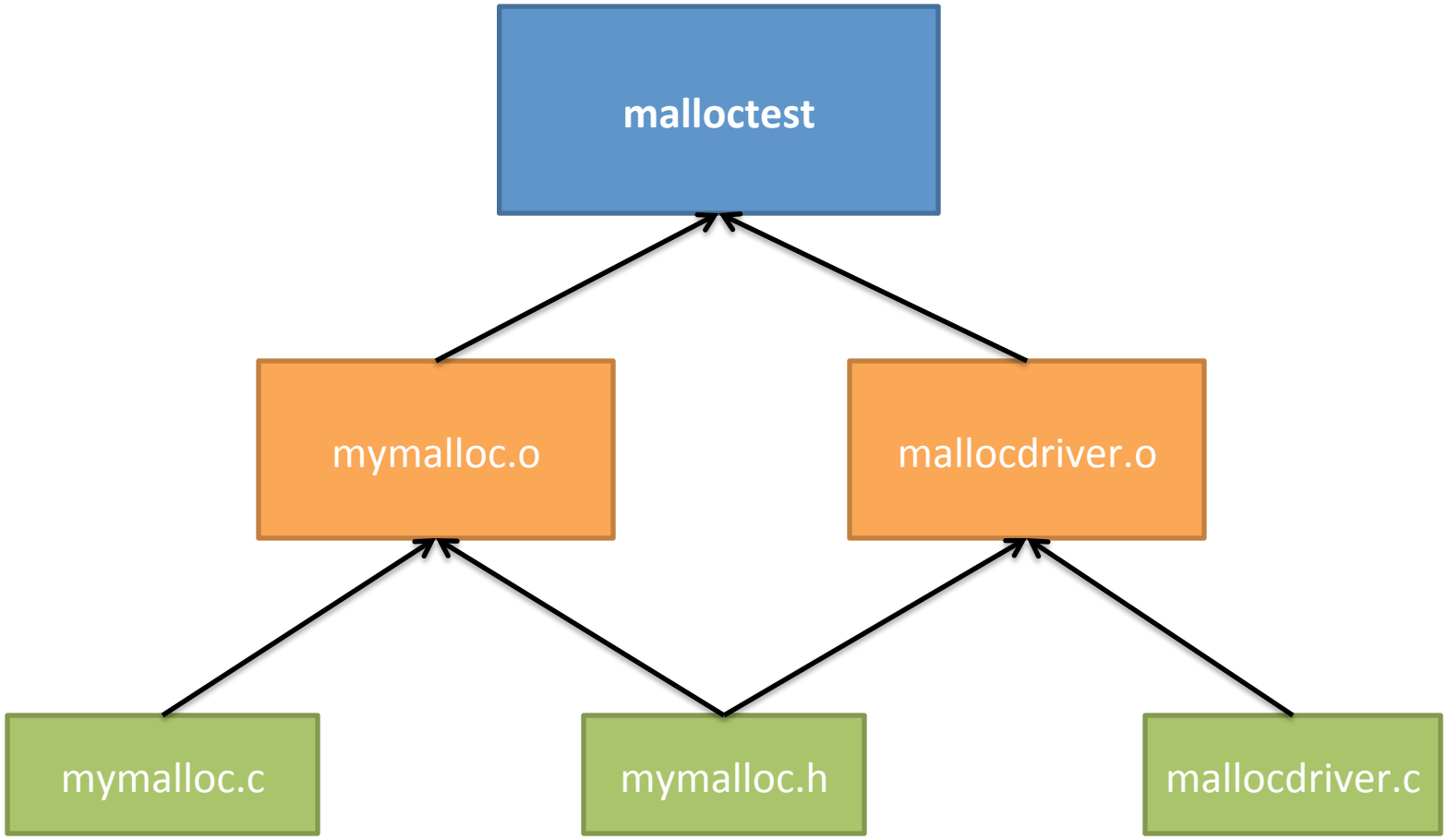
Makefile

```
malloctest: mymalloc.o mallocdriver.o
   gcc –o malloctest mymalloc.o mallocdriver.o

mymalloc.o: mymalloc.c mymalloc.h
   gcc –c mymalloc.c

mallocdriver.o: mymalloc.h mallocdriver.c
   gcc –c mallocdriver.c
```

Dependency Graph



Working on Large Projects

- Two main issues that must be addressed
  - Sharing work
  - Tracking work

- Large projects include multiple authors
  - Working on same files at same time
  - Changes have to be combined into a single global version
- Large projects span large time windows
  - Code evolves constantly
  - Multiple versions are released
  - Code history allows walking through the timeline

Combining Modifications

- Multiple programmers collaborating on a single codebase
  - Need to keep up to date with everyone else's changes
  - Need to make sure that modifications don't conflict

- This requires a single master version of the code
  - When 1 programmer adds code to the project they must apply (commit) it to the master version
    - Becomes visible to all other developers
  - The master version is accessible to everyone

Code History

- Code history tracks the changes to a codebase overtime
  - Every change is included in a single commit
  - Commits represent a delta from the previous state of the code
    - A diff or a patch against the previous version
    - Allows access to the entire codebase at any point in the development timeline
- Uses:
  - Reverse a bunch of changes if they were a bad idea
  - Locate and identify sources of bugs

Code Repositories / Version Control

- Any competent programming team uses some type of version control/repo
  - Automates the tracking and merging of changes from many developers
- Code is stored in a central repository that all developers have access to
  - Repository stores entire commit history of the project from the very beginning
  - Allows developers to easily check out the latest version that includes all of the changes that have been made
  - As developers make changes, they can then commit their modifications back to the repository
- Key Point: Only the deltas are committed
  - Combining modifications is easier, reduces space requirements

Merging changes

- Two developers work independently and modify a code base
  - They both need to commit their changes
  - But both of them modified the same original version (without the other's changes)
  - Version control systems were designed for this scenario
- Two possible scenarios:
  - Each developer modified different parts of the code base
    - In this case the changes are automatically merged
  - Each developer modified the same parts of the code base (e.g. changed the same line of code)
    - This creates a conflict, which a human must resolve before a merge

Releases and Tags

- Most large projects release specific versions
  - Contain a set of new features or bug fixes all at once
  - Ensures that everyone is using the same identical version of the code
- Version control allows creating releases
  - A specific commit (a point in the timeline) is chosen
  - This commit is marked in some way (tagged)
  - The tag will never move, a given release will always point to the same version of the code
- Releases often need to be updated with bug fixes
  - But the codebase has changed since the release was made
  - The answer is to create a parallel history inside a branch, that is separate from the master version

Version Control Systems

- Traditional centralized systems
  - Master version + history stored on a server
    - CVS: Concurrent Versions System
    - Subversion (SVN)
- Distributed systems
  - No single master version, instead developers replicate the entire repository locally
  - Merges are explicit operations done by the user
    - Git (developed for the Linux kernel)
      - Probably the most popular
    - Mercurial

git commands

- git init
  - Creates a repo inside a directory
- git clone <path or URL>
  - Creates a local copy of an existing repo
- git add <filename>
  - Add a file to local repo
- git commit
  - Commit local changes to local repo
- git push
  - Push local commits to a remote repo
- git pull
  - Apply commits from a remote repo to the current repo