Header Files

CMSC 104 Section 02
April 22, 2024

Administrative notes

Reminder: quiz on Wednesday

- In class, on Blackboard
- Unless you have permission, you must take the quiz in the classroom
- Will not include today's lecture

Lots of classwork and homework

- Homework 7 due tonight
- Classwork 8 due Wednesday
- Homework 8 released today; due next Monday (April 29)
- Classwork 9 released Wednesday; due Wednesday, May 1

Today’s lecture

A peek behind the scenes at how C really works

- You now know enough about programming to be told the truth

What the heck is a “.h” file?

- What’s in it?
- Why do we need them?

What’s this “preprocessor” nonsense all about?

header files

- Anything that ends in “.h” is a header file
- Header files contain function prototypes for all of the functions found in the specified library.
  - They also contain definitions of constants and data types used in that library.
- By reusing code, especially from the standard C library, you reduce the chance for errors while making the code easier to maintain (instead of re-implementing everything yourself)

Pre-existing and user-defined header files

There are a number of pre-written header files that come standard with C implementations

- On Linux systems they’re usually in the directory /usr/include

(A quick look at the contents of /usr/include on gl.umbc.edu)

```
[arsenaut@linux3 include]$ ls
acc_user.h          elfutils            gmpxx.h             mtd                 pwd.h               term_entry.h
aio.h               endian.h            gnu                 nc_tparm.h          pypy3.9             term.h
aliases.h           envz.h              gnumake.h           ncurses             python2.7           termio.h
alloca.h            err.h               gnu-versions.h      ncurses_dll.h       python3.10          termios.h
a.out.h             errno.h             GraphicsMagick      ncurses.h           python3.11          tgmath.h
argp.h              error.h             grp.h               ncursesw            python3.12          thread_db.h
argz.h              eti.h               gshadow.h           net                 python3.6m          threads.h
ar.h                etip.h              iconv.h             netash              python3.7m          tic.h
arpa                execinfo.h          ieee754.h           netatalk            python3.8           time.h
asm                 fcntl.h             ifaddrs.h           netax25             python3.9           ttyent.h
asm-generic         features.h          ImageMagick-7       netdb.h             rdma                uchar.h
assert.h            features-time64.h   inttypes.h          neteconet           re_comp.h           ucontext.h
atlas               fenv.h              ivl_target.h        netinet             regex.h             ulimit.h
atlas-x86_64-base   ffi.h               julia               netipx              regexp.h            unctrl.h
bits                ffitarget.h         KHR                 netiucv             resolv.h            unicode
boost               ffitarget-x86_64.h  langinfo.h          netpacket           rpc                 unistd.h
btparse.h           ffi-x86_64.h        lapacke_config.h    netrom              sanitizer           unity.h
byteswap.h          finclude            lapacke.h           netrose             sched.h             util.h
c++                 FlexLexer.h         lapacke_mangling.h  nfs                 scsi                utime.h
cblas_f77.h         fmtmsg.h            lapacke_utils.h     nlist.h             search.h            utmp.h
cblas.h             fnmatch.h           lapack.h            nl_types.h          semaphore.h         utmpx.h
cblas_mangling.h    form.h              lastlog.h           nss.h               setjmp.h            values.h
common_ini_test.h   fpu_control.h       libdrm              numpy               sgtty.h             veriuser.h
complex.h           fstab.h             libelf.h            obstack.h           shadow.h            video
cpio.h              fts.h               libgen.h            openssl             shell.h             vpi_user.h
crypt.h             ftw.h               libintl.h           orc                 signal.h            wait.h
ctype.h             fuzzer              libsync.h           panel.h             sound               wchar.h
cursesapp.h         gconv.h             libxml2             paths.h             spawn.h             wctype.h
cursesf.h           gdb                 limits.h            pciaccess.h         stab.h              wordexp.h
curses.h            gelf.h              link.h              pcrecpparg.h        stdc-predef.h       X11
cursesm.h           getopt.h            linux               pcrecpp.h           stdint.h            xcb
cursesp.h           gforth              locale.h            pcre.h              stdio_ext.h         xen
cursesw.h           ghdl                lzma                pcreposix.h         stdio.h             xf86drm.h
cursslk.h           GL                  lzma.h              pcre_scanner.h      stdlib.h            xf86drmMode.h
dirent.h            GLES                malloc.h            pcre_stringpiece.h  string.h            xray
dlfcn.h             GLES2               math.h              _pli_types.h        strings.h           yaml.h
drm                 GLES3               mcheck.h            poll.h              sv_vpi_user.h       zconf.h
duo.h               glob.h              memory.h            printf.h            sys                 zdict.h
duo_private.h       glvnd               menu.h              proc_service.h      syscall.h           zlib.h
dwarf.h             gmp.h               misc                profile             sysexits.h          zstd_errors.h
EGL                 gmp-mparam.h        mntent.h            protocols           syslog.h            zstd.h
eigen3              gmp-mparam-x86_64.h monetary.h          pthread.h           tar.h
elf.h               gmp-x86_64.h        mqueue.h            pty.h               termcap.h
```

Header File	Contains Function Prototypes for:
stdio.h	standard input/output library functions, file input/output
math.h	math functions
ctype.h	functions for testing characters and data types
dirent.h	working with directories
errno.h	error handling
limits.h	sizes of basic types
locale.h	translating programs into different human languages
stddef.h	working with different specific-sized data types
string.h	functions for working with strings
time.h	Getting time and date information
unistd.h	Standard cross-Unix & Unix-like operating systems functions

So what's ***really*** in a .h file?

Function prototypes

Constants you declare

Custom data types you declare - `struct`s, `enum`s, …

But NO function specifications!!!

- Wait a minute - what good, really, are these .h files?
- Where ARE these function specifications?
- And most importantly, how does my code work?

Let's go through that, one thing at a time

Example of a .h file

Math.h (from gl.umbc.edu’s /usr/include/math.h file)

A quick digression - writing your own .h file

You can write your own header file

- Useful if you're writing functions that you will use over and over, in all the rest of your programs
- You just write those functions once and include the headers/link in the code whenever you want to use them.

errMessage.h file (on GitHub Code_samples repo)

Why would you do that?

- Because you can :-)
- Only if you're going to write code you want to reuse
- Helps you develop small, modular code that you can more easily debug????

How do you use that file?

In testing.c file from GitHub Code_samples repo:

```
#include <stdio.h>
#include “errMessage.h”
```

Okay, let’s explain this

#include is a pre-processor command. When you start to compile the C file, testing.c, a “pre-processor” takes a first look at your program. Everywhere a #include statement occurs, the pre-processor goes and gets the .h file and copies its contents into your C program.

- So before your program is compiled,
  - #include <stdio.h> is replaced with all the code in the file /usr/include/stdio.h
  - #include “errMessage.h” is replaced with all the code in the file ../errMessage.h
- Only then is the program compiled

More questions on #include?

Yeah, what's with the <> and the “”?

```
#include <stdio.h>
#include “errMessage.h”
```

That tells the pre-processor where to look for the file in question

- <> tells the pre-processor to look in the standard location - on Linux systems, typically /usr/include
- “ “ tells the pre-processor to look in the current directory; e.g., my cs104 directory or your CW8 directory or…

WHERE'S THE FUNCTION CODE?????

Where do we hide the function code?

We define the function prototypes in a .h file.

We put the corresponding function definitions in another .c file

- E.g., errMessage.c in the GitHub Code_samples repo

So how does the system know where to find that .c file?

- We haven't included that file in the program

By ***linking*** C files together to produce an executable

The simplest linking example

```
gcc testing.c errMessage.c -o testing.o
```

This compiles the testing.c program (which #includes the errMessage.h function prototypes) and then compiles the errMessage.c program, and links the results together in a single executable file called testing.o

Now when you run testing.o you have everything you need.

Summary: what's the point of today's lecture?

Header files (.h) contain function prototypes, type definitions and constant definitions that you will use multiple times

- There are a lot of pre-written .h files, and you can write your own

Header files do NOT include the function definitions. Those are in other .c files which are compiled and linked in to your code to produce a single executable.

This helps:

- Develop small, modular programs which are more likely to be correct and easier to debug
- Reuse code over and over