PURDUE UNIVERSITY®

CS 240: Programming in C

Lecture 3: File I/O

Prof. Jeff Turkstra

Announcements

- Ed Discussion Reminders
  - Monitoring
  - Etiquette
  - Posting code
- Quiz reminder – 24 hours after release
  - Unless in lecture

Prof. Turkstra Office Hours

- Monday 3:00pm – 4:00pm
- Thursday 2:00pm – 3:00pm
- Friday 1:30pm – 2:30pm
- …always welcome to email for an appointment

Supported environments

- The Purdue CS Linux systems are set up properly for this course
- If you're going to work at home:
  - We strongly encourage you to SSH into a Purdue CS system
    - data.cs.purdue.edu, borgNN.cs.purdue.edu, etc
  - Alternatively, you can install gcc on your computer
    - (Maybe compile it yourself. It's written in C.)
- We will use gcc on a Purdue CS Linux system to grade your homework

Remote access

- Secure Shell (SSH, OpenSSH, PuTTY, etc)
  - Almost all modern systems include SSH as part of their command line tools
  - Alternatively, https://www.putty.org/
- Graphical session:
  - Run X on your computer
  - X remote display protocol (XDMCP)
  - Run X applications over SSH
- See the FAQ page for more info

HW1

- Available on the course website as of Monday
- Due Sunday, January 26
  - ...with three grace days, actual deadline is Wednesday, January 29 9pm
- Remember, you should try to write the solution out on paper first
- Get it done by Sunday!
  - Homework 2 comes out Monday
- Linter

Reminder

Work alone.

Reading

- Read Chapter 7 in K&R
  - …and/or Chapter 13 in Beej's

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
```

  - But then it doesn't produce any .o files

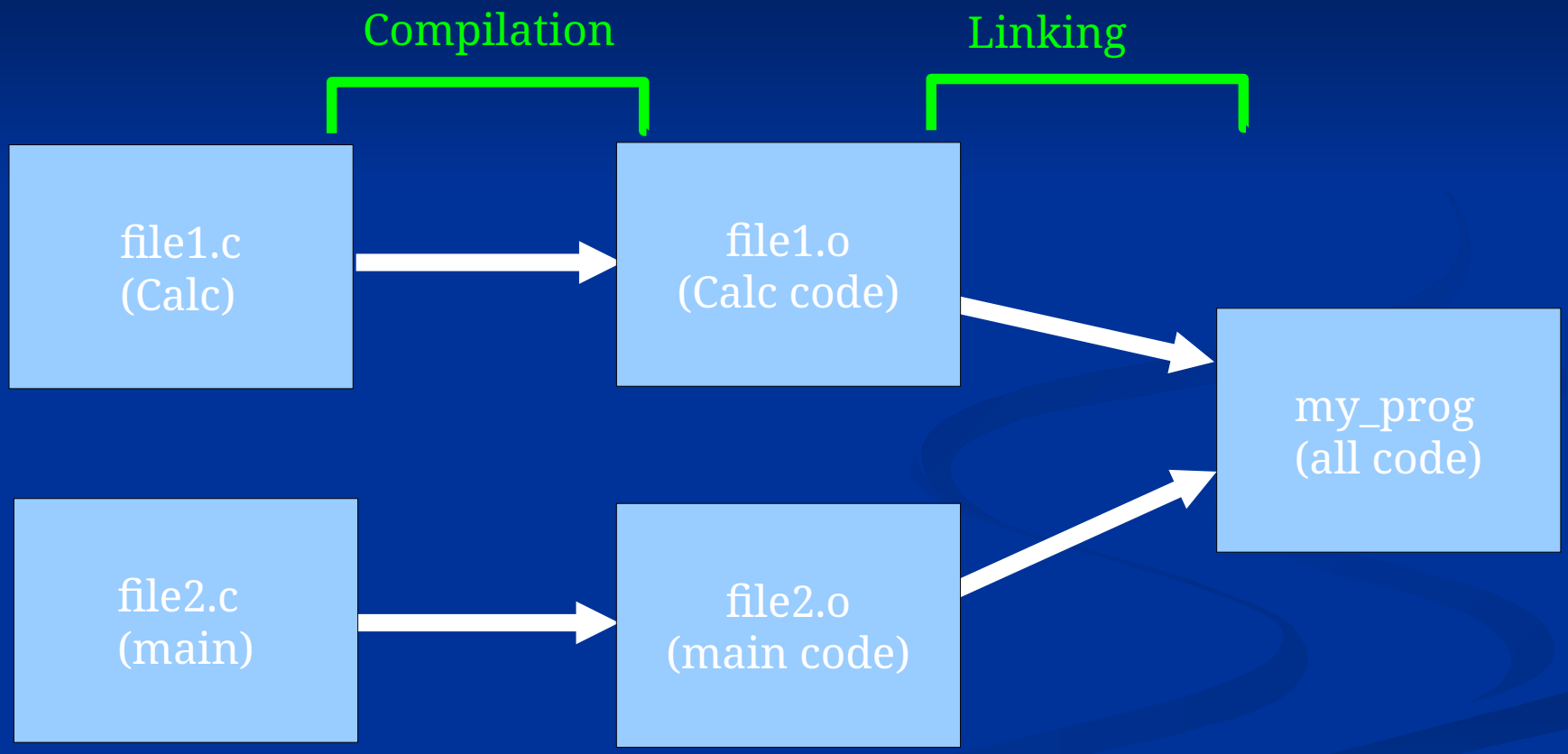
- I know what you're thinking…

  "Why do we want all 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



Executing

- 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!
```

Notes from last week

- Don’t try to look at what’s inside an object file or executable
- When I say “we’ll get to that later” that’s not meant to scare you
- If you see anything you don’t understand, do ask about it

#include

- #include always pulls a header file into another file
  - #include “file.h”
    Pull in file.h from the present directory
  - #include <file.h>
    Pull in /usr/include/file.h

Example of #include

/home/jeff/x.c

```
#include <stdio.h>
#include "x.h"

int main()
{
  printf ("Val %d\n", X);
  return 0;
}
```

/usr/include/stdio.h

```
/*
 * scary things
 * in this file...
 */
typedef FILE ...
```

/home/jeff/x.h

```
#define X   ( 3456 )
```

Final result of #include

```
/*
 * scary things
 * in this file...
 */
typedef FILE ...

#define X  ( 3456 )

int main()
{
  printf ("Val %d\n", X);
  return 0;
}
```

- All of the things that previously resided in separate files were pulled together into one stream
- This gets fed to the compiler

Boolean variables

- We can use boolean variables of the type “bool” and assign values of “true” or “false” to them. E.g.:

```
bool my_function(bool var) {
  bool x = false;
  x = x && var;
  if (x == true) return false;
  else return true;
}
```

- You need to #include <stdbool.h> to do this

Purdue trivia

- "Harvey Washington Wiley was the first professor of chemistry, the first state chemist, the first ROTC instructor, the first baseball coach, and the 'father of the U.S. pure food and drug law.' Yet, the Purdue board of trustees once censored him for riding a bicycle - considered unseemly conduct by a faculty member."

  - A Century and Beyond,
    by Robert W. Topping

File I/O essentials

- Read Chapter 7!
- In a program, we refer to files with FILE pointers
- A file must be opened before writing to or reading from it
  - EXCEPT: stdin, stdout, stderr
  - printf() uses stdout
  - scanf() uses stdin

fopen()

```
FILE *fopen(char *file_name,
            char *mode);
```

file_name: name of file

mode: will we read it, write it, or both?

Modes for fopen()

“r” - open the file in question only for reading. The file must already exist.

”w” - creates a file for writing. If the file existed already, it will be overwritten.

”a” - appends to the end of a file. If the file did not exist in advance, it will be created.

fopen() return values

- If successful, fopen() opens the file and returns a FILE pointer to represent the file
- If it is not successful, fopen() returns a NULL pointer (a zero pointer)
- These FILE pointers are said to be opaque. We don't care what they point to – only that they are not NULL

Always check the return value!

In this class, failure to check the return value of fopen() will result in a poor grade.

fclose()

```
int fclose(FILE *file_pointer);
```

- Every successfully opened file must be closed when we are finished with it
- When the file is closed its internal data is flushed
- fclose() does not set the FILE pointer to NULL
  - You should do that after calling fclose()

fclose() return values

- On success, fclose() returns a 0
- On failure, fclose() returns EOF
- Why might fclose() fail?
- You do not have to check the value returned from fclose() in this class

Takehome Quiz 1

1. Write a C function named “boiler” that, when called, will print the string “Boiler Up!” Its return type should be void. It should not accept any arguments.

2. What is the gcc command used to compile purdue.c into an executable named “boilers”, with warnings treated as errors using the ANSI C standard?

3. How is class so far? Anything I can do to wake you up?

fprintf()

- fprintf() works just like printf() except that it takes an extra FILE pointer argument
- The following are equivalent...

```
printf("Hello, world.\n");
fprintf(stdout, "Hello, world.\n");
```

- Another example...

```
fprintf(stdout, "The number is %d.\n",
        12345);
```

Reading data in C

- C is a little different than Java when it comes to reading data
- Think printf() in reverse…

```
scanf("%s %d", buffer, &int_var);
```

- Returns number of successful conversions

fscanf()

- Works just like scanf() except that it takes an extra FILE pointer argument
- The following are equivalent…

```
scanf("%s", buffer);
fscanf(stdin, "%s", buffer);
```

- Another example…

```
fscanf(file_p, "%d", &var);
fscanf(file_p, "num: %d, name: %s",
        &var, buffer);
```

Examples

```
#include <stdio.h>

int main() {
   fprintf(stdout, “Hello, world\n”);

   return 0;
}
```

Same example with a file

No error checking...

```
#include <stdio.h>

int main() {
   FILE *file_ptr = 0;
   file_ptr = fopen("xyz", "w");

   fprintf(file_ptr, "Hello, world!\n");

   fclose(file_ptr);

   return 0;
}
```

Same example with a file
Proper error checking

```
#include <stdio.h>

int main() {
  FILE *file_ptr = 0;

  file_ptr = fopen(“xyz”, “w”);
  if (file_ptr == NULL) {
    fprintf(stderr, “Can’t open.\n”);
    return 1;
  }

  fprintf(file_ptr, “Hello, world!\n”);
  fclose(file_ptr);
  file_ptr = NULL;

  return 0;
}
```

For Next Lecture

- Look at HW1
- Read Chapter 7 of K&R
  - ...and/or Chapter 13 in Beej's
- Understand the following functions:
  - feof()
  - ferror()
  - clearerr()
  - fwrite()
  - fread()

Boiler Up!