CS 240: Programming in C

Lecture 3: File I/O

PURDUE UNIVERSITY

Homework 1

- Due Wednesday, Sept 4 at 9:00 PM
- Submissions after midnight will be 0 pts
- Get it done by Friday!
  - Homework 2 comes out next Monday
- Adhere to the code standard
  - We will be grading for code style
  - Use the linter!

Supported environments

- Purdue CS Linux systems are set up for this course
- If you're going to work at home:
  - You should SSH into a Purdue CS system (e.g., data.cs.purdue.edu)
  - Here's a guide to SSH

Reminder

Work alone.

From last time

- Function **prototypes**
- Need to declare a function before you can use it

file1.c

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

   temp = first_val * second_val;

   return temp;
}
```

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;
}
```

Function prototypes

- What happens when you have a lot of functions?
- What if you want to use the same functions in many different .c files?
- This is where `#include` becomes useful

#include

- “#” indicates a **preprocessor** directive
  - Extra steps before compilation begins
  - We’ll talk more about this later in the semester
- `#include` 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/may5/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/may5/x.h

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

Example of #include

/home/may5/x.c

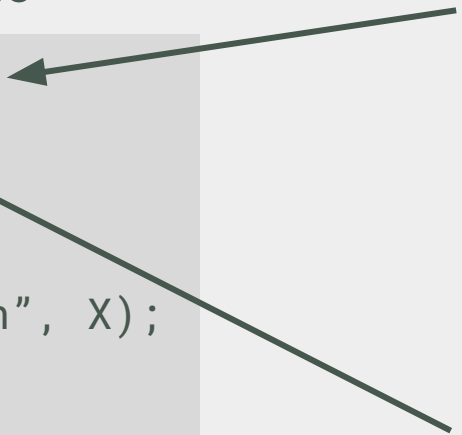
```
#include <stdio.h>
#include “x.h”

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

/usr/include/stdio.h

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

/home/may5/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

Where to find header files?

- Most will be in preconfigured paths (e.g., /usr/include/)
- Use the `-I` flag to specify additional include paths

Where to find header files?

```
$ gcc -I /home/may5/my_lib -o my_prog main.c
```

/home/may5/my_prog/main.c

```
#include <stdio.h>
#include <my_lib.h>
int main() {
    printf("Val %d\n", X);
    return 0;
}
```

/home/may5/my_lib/my_lib.h

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

Boolean variables

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

```
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

Lecture Quiz!

- Assignment in Gradescope (Quizzes)
- Enter your response on a laptop, phone, or tablet
- You will have 5 minutes to complete it

Lecture Quiz #0

Write the GCC command(s) to compile and link these source files into an executable named **my_secret**:

GCC flags

```
-c
-g
-Wall
-Werror
-O
-o file
-ansi
-std=X
-I path
```

/home/may5/main.c

```
#include <stdio.h>
#include “x.h”

int main() {
    int a = 3;
    a = secret_fn(a);
    printf(“%d\n”, a);
    return 0;
}
```

/home/may5/x.h

```
int secret_fn(int a);
```

/home/may5/secret.o

```
01101001101010011001001
1010                    1001
1101   Object code for  0100
       secret_fn()
00011010110101000100110
```

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 the 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 about what they point to - only that they are not NULL.

Always check the return value!

- fopen() could fail for many reasons.
- 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.

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”);
```

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 the 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);
```

Example

```
#include <stdio.h>

int main() {
   fprintf(stdout, “Hello, world!\n”);
   return 0;
}
```

Same example with a file

```
#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;
}
```

With 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

- Start Homework 1!
- Read Chapter 7 of K&R
  - Skip 7.3
  - and/or Chapter 13 in Beej
- Understand the following functions:
  - `foef()`
  - `ferror()`
  - `clearerr()`
  - `fwrite()`
  - `fread()`

Slides

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