C (CONT.)

MULTIPLE PROGRAM FILES

- What if you have a single function you find you reuse often in different pieces of code
    - Copy and paste function - error prone, end up having to fix multiple copies
    - Solution: Create a separate file with that function

HEADER FILES

- Things like `string.h`, `stdio.h`, `stdlib.h` are system header files
  - Use `#include <header_filename.h>`
- There are also custom header files (your own header files)
  - Use `#include "header_filename.h"`

HEADER FILE CONTENT

- Header files contain function prototypes
- Example: `sum.h`

```
#ifndef SUM_H
#define SUM_H

int sum(int, int);

#endif
```

WHAT'S THE OTHER STUFF

- Need to make sure we don't have issues if the header file gets included multiple times
- Use an "include guard"
- Uses preprocessor directives to prevent additional attempts of defining function prototype on 2nd, 3rd, etc. include
- Nests prototype definition in an if
    - Checks to see if macro is set before entering if
    - Sets macro in if -- any future won't enter if

WHAT ABOUT THE FUNCTION DEFINITION?

- Create another file
- Example: `sum.c`

```c
#include "sum.h"

int sum(int a, int b) {
    return a+b;
}
```

USING IT IN ANOTHER FILE

```c
#include <stdio.h>
#include "sum.h"

int main(void) {
    int a = 5;
    int b = 6;
    printf("a = %d, b = %d, sum = %d\n", a, b, sum(a,b));
    return 0;
}
```

USING IT IN ANOTHER FILE (CONT.)

- Told it about the header file with the include
- How does it know where the actual function definition is?
- Need to add c file when compiling
- Example: `gcc program.c sum.c -o run`

COMPILING (CONT.)

- What happens if `sum.c` was actually 10000+ lines long and all we wanted to do was change the format in the print statement in `program.c`
- Can be useful to compile files separately
  - Compile each file to its own object file
  - Create executable by linking object files

COMPILING (CONT.)

Example: compiling files separately

```
gcc -c sum.c
gcc -c program.c
gcc program.o sum.o -o run
```

MORE ON INCLUDES

- There are default search locations for includes
- `#include "header_fname.h"` first looks in current directory (then other predefined ones)
- What if header file isn't in current directory?
    - Can use path in the include statement
    - Can specify additional directories to search
        - `-I` option
        - Example `gcc -Iproj/headers` (or wherever your header files live

STRUCTS

- The closest you'll get to OO in C
- Allows you to group together multiple pieces of data

```
typedef struct Point {
    double x;
    double y;
} Point;
```

STRUCTS (CONT.)

- Previous just defines the struct
- Need to actually declere one and set values
- Declare like any other variable except now with type Point
- Access member with .
- Have a pointer to struct, access member with ->

```
Point p1;
p1.x = 5;
p1.y = 10;
```

STRUCTS (CONT.)

- Can contain regular data types, pointers, arrays, other structs
- Like other types, can allocate on stack or heap (dyanmic with malloc)
- Can create pointers to structs