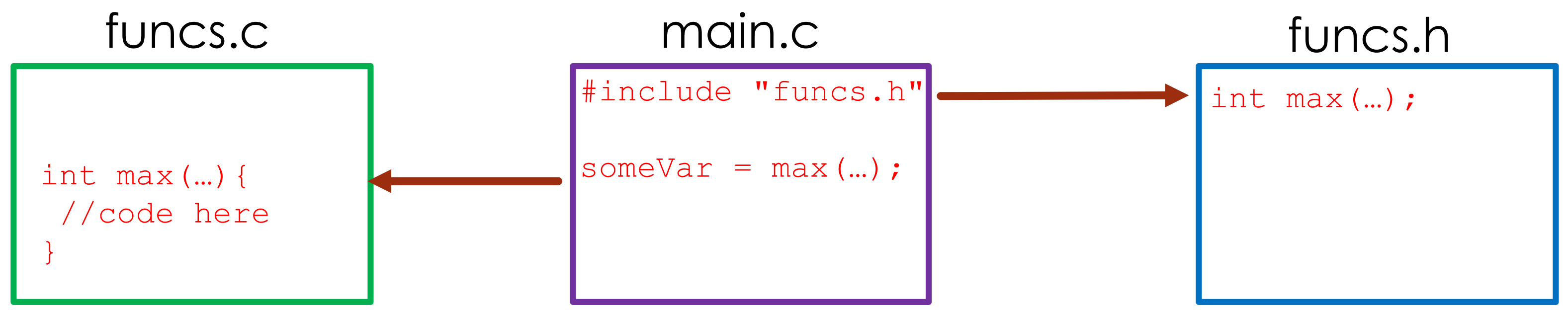
Lab 2

Concepts & Helpful Tips

Concepts Introduced in Lab 2

Task 1 - Multi Source File model

- In this lab, we are starting to split our code into many files:



funcs.c:

```
int max(…){
 //code here
}
```

main.c:

```
#include "funcs.h"

someVar = max(…);
```

funcs.h:

```
int max(…);
```

.c -> C code

This file contains the definition (i.e., body) of functions.

.c -> C code

This file contains the main function and possibly others functions (optional). It may call functions defined in other files.

.h -> header

This file contains function headers (also called function declarations, or function prototypes).

What is a function?

General Helpful Tips about Lab 2

```
gcc -Wall funcs.c main.c -o "executable_name"
```

- Note that the order of the parameters to the command `gcc` does not matter, so this command here: `gcc -Wall -o "executable_name" funcs.c main.c` will do the same as the command above

- Even though we may not be told to do so in our labs, *giving a descriptive name to each of our executable files* (as opposed to using `a.out` over and over) is a good habit to adopt – and we are expected to do this throughout this course!
  - We do this by using the "**-o**" option and using a descriptive and unique name for our executable
  - For example:

```
gcc -Wall funcs.c main.c -o fcn
gcc -Wall t1.c -o t1
```

Helpful Tips about Lab 2

Task 5 - `contains`

- You want to look up C library `string.h`
  - Looking up info about C and its functions and libraries online is not considered Academic Dishonesty
    - It is a **very good habit** for software developers to acquire
  - Copying C code found online and submitting it as ours, as part of our tasks, is considered Academic Dishonesty
    - It is ***not*** a very good habit for software developers to acquire
  - Some of the functions found in `string.h` (like `strstr( )`) can be quite useful in this lab (for example, in Task 5 called `contains` ☺).

Helpful Tips about Lab 2

Task 6

- You may want to construct a function that determines whether a character is a *word character* or not, i.e., a character that can be part of a word (as described in this task)
- Then you can call this function, passing to it the character you have read from the keyboard
  - This function would return true (1) if the character is a *word character*
  - This function would return false (0) if the character is **not** a *word character*
- The idea is that you may need to determine whether a character is a *word character* or not in more than one task in this Lab 2
  - So creating a function which can be called when needed is a very efficient way of solving this problem

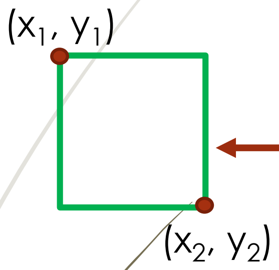
Helpful Tips about Lab 2

<u>Task 8</u>

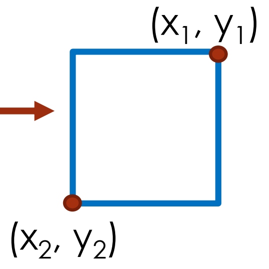
- You cannot implement the task called `Letter frequency` by writing 26 **`if`** statements (1 for each letter) ☹
- <u>Tip:</u>
  - Each letter has a numerical ASCII value
  - Can this numerical value be used at all? You will have to tweak the numerical ASCII value of each letter a little bit
  - Aside from using ASCII, there are other possible ways to solve this problem

Helpful Tips about Lab 2

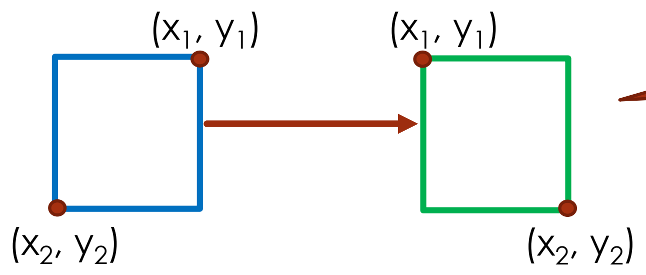
<u>Task 10</u>



- Keep in mind that there can be two ways of describing a rectangle using its top and bottom corners:
  1. Top left corner + bottom right corner
  2. Top right corner + bottom left corner



- To make your code more straightforward and manageable, first, have your code transform the second way of describing a rectangle into the first way:



Then, you only have one type of input to consider!

- Functions declared in `math.h` may be helpful in solving this task
- So, have a look at the content of `math.h` ☺