COMP1511 PROGRAMMING FUNDAMENTALS

LECTURE 9

Recap Command Line Arguments
Lecture Program: 2D Arrays
Multi File Projects (if time, but I am really cooking here)

LAST WEEK…

- 2D Arrays
- Strings
- Command Line Arguments

TODAY...

- Hope you have had a great weekend and have gotten started on your assignment
- A bigger 2D array problem (like the assignment!) - I am desperate for churros!
- Multi-file projects (in preparation if time)

WHERE IS THE CODE?

Live lecture code can be found here:

HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/25T1/CODE/WEEK_5/

COMMAND LINE ARGUMENTS

WHAT ARE THEY?

- So far, we have only given input to our program after we have started running that program (using scanf())
- This means our `int main(void) {}` function has always been void as input
- Command line arguments allow us to give inputs to our program at the time that we start running it! So for example:

```
avas605@vx5:~$ dcc test6.c -o test6
avas605@vx5:~$ ./test6 argument2 argument3 argument4
```

TIME TO CHANGE THAT VOID

LET'S GET OUR MAIN FUNCTION TO ACCEPT SOME INPUT PARAMETERS

- In order to change your main function to accept command line arguments on first running, you need to change the void input:

```
int main(int argc, char *argv[]) {}
```

- int argc = is a counter for how many command line arguments you have (including the program name)
- char *argv[] = is an array of the different command line arguments (separated by a spaces). Each command line argument is a string (an array of char)

AN EXAMPLE

```c
#include <stdio.h>

int main (int argc, char *argv[]) {
    printf("There are %d command line arguments in this program\n", argc);

    //argv[0] is always the program name
    printf("The program name is %s (argv[0])\n", argv[0]);

    // What about the other command line arguments? Let's loop through
    // the array and print them all out!
    for (int i = 0; i < argc; i++) {
        printf("The command line argument at index %d"
                    "argv[%d] is %s\n", i, i, argv[i]);
    }

    return 0;
}
```

```
avas605@vx02:~$ dcc argv_demo.c -o argv_demo
avas605@vx02:~$ ./argv_demo We are almost half way through this term!
There are 9 command line arguments in this program
The program name is ./argv_demo (argv[0])
The command line argument at index 0argv[0] is ./argv_demo
The command line argument at index 1argv[1] is We
The command line argument at index 2argv[2] is are
The command line argument at index 3argv[3] is almost
The command line argument at index 4argv[4] is half
The command line argument at index 5argv[5] is way
The command line argument at index 6argv[6] is through
The command line argument at index 7argv[7] is this
The command line argument at index 8argv[8] is term!
```

WHAT IF YOU WANT NUMBERS AND NOT STRINGS?

REMEMBER THAT EACH COMMAND LINE ARGUMENT IS A STRING

- You want numbers, if you want to use your command line arguments to perform calculations
- There is a useful function that converts your strings to numbers:

  `atoi()` in the standard library: <stdlib.h>

WHAT IF YOU WANT NUMBERS AND NOT STRINGS?

REMEMBER THAT EACH COMMAND LINE ARGUMENT IS A STRING

```c
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {
    // Remember that the command line arguments are all strings, so if you
    // need to do mathematical operations, you will need to convert them
    // to numbers
    // You can do this with a really handy function atoi() in the stdlib.h library!

    // Let's print out all the command line arguments given and then add
    // them together to give the sum of the command line arguments

    int sum = 0;
    for (int i = 1; i < argc; i++) {
        printf("The command line argument at index %d (argv[%d]) is %d\n",
                i, i, atoi(argv[i]));
        sum = sum + atoi(argv[i]);
    }
    printf("The sum of the arguments is %d\n", sum);

    return 0;
}
```

```
avas605@vx02:~$ dcc atoi_demo.c -o atoi_demo
avas605@vx02:~$ ./atoi_demo 3 4 5 6 7
The command line argument at index 1 (argv[1]) is 3
The command line argument at index 2 (argv[2]) is 4
The command line argument at index 3 (argv[3]) is 5
The command line argument at index 4 (argv[4]) is 6
The command line argument at index 5 (argv[5]) is 7
The sum of the arguments is 25
```

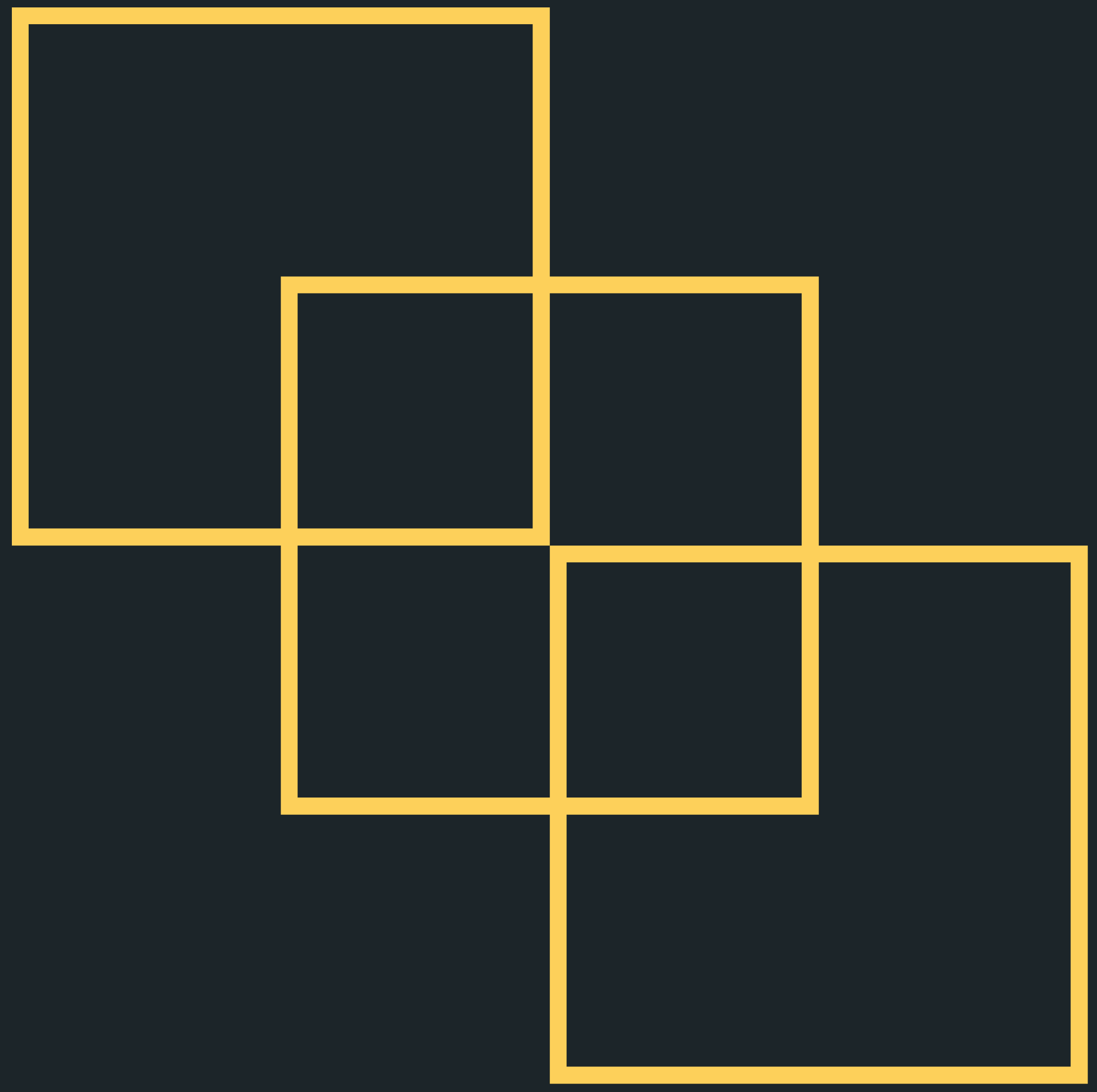
WHAT IF YOU WANT NUMBERS AND NOT STRINGS?

REMEMBER THAT EACH COMMAND LINE ARGUMENT IS A STRING

```c
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {
    // Remember that the command line arguments are all strings, so if you
    // need to do mathematical operations, you will need to convert them
    // to numbers
    // You can do this with a really handy function atoi() in the stdlib.h library!

    // Let's print out all the command line arguments given and then add
    // them together to give the sum of the command line arguments

    int sum = 0;
    for (int i = 1; i < argc; i++) {
        printf("The command line argument at index %d (argv[%d]) is %d\n",
                i, i, atoi(argv[i]));
        sum = sum + atoi(argv[i]);
    }
    printf("The sum of the arguments is %d\n", sum);

    return 0;
}
```

```
avas605@vx02:~$ dcc atoi_demo.c -o atoi_demo
avas605@vx02:~$ ./atoi_demo 3 4 5 6 7
The command line argument at index 1 (argv[1]) is 3
The command line argument at index 2 (argv[2]) is 4
The command line argument at index 3 (argv[3]) is 5
The command line argument at index 4 (argv[4]) is 6
The command line argument at index 5 (argv[5]) is 7
The sum of the arguments is 25
```

BREAK TIME



Can you reproduce this figure using just one line, without lifting the pen and without going back over an already drawn line?

PROBLEM TIME

SIMILAR TO YOUR ASSN 1

Having come back from my time in the States, I was busy consistently trying to find churros. So why not make it an array program that we can use to bring a few concepts together?

You are a player navigating a 2D grid-based map in search of churros while avoiding walls. The game tracks your position, and you can move in four directions. The objective is to collect all churros on the board by reaching their locations.

PROBLEM TIME

SIMILAR TO YOUR ASSN 1

1) You will get some initial user input to set up the map of the place: with churro locations, wall locations and initial player location
2) You will update the map with these details
3) You will keep getting user input using 'wasd' keys to move up/down/left/right - you will keep going until you find a churro! Once you find a churro, you will collect this churro, your task is to collect all the churros on the board
5) You will allow yourself a break and to give up finding churros (after all, you still need to get through your assignment without a sugar crash!) by pressing CTRL+D

PROBLEM TIME

SIMILAR TO YOUR ASSN 1

You are going to get some starter code:

1) initialise_map function
2) print_map function
3) print_location function (print_tile....)

```
void initialise_map(struct location map[MAP_ROWS][MAP_COLUMNS]);
void print_map(struct location map[MAP_ROWS][MAP_COLUMNS]);
void print_location(struct location location, int place_print);
```

PROBLEM TIME

SIMILAR TO YOUR ASSN 1

Your enums in this problem:

```
enum tile {
    EMPTY,
    WALL,
    CHURRO,
    TRAP,
    TELEPORT,
    PLAYER
};

enum player_action {
    MOVE_UP,
    MOVE_DOWN,
    MOVE_LEFT,
    MOVE_RIGHT,
    NOTHING,
    COLLECT
};
```

PROBLEM TIME

SIMILAR TO YOUR ASSN 1

And then a struct location, made up of an entity and place_type at each location:

```
struct location {
    enum tile tile;
    enum player_action player_action;
};
```

PROBLEM TIME

SIMILAR TO YOUR ASSN 1

So that means, your map is an array of structs, with an entity and a location at each grid point:

```
struct location map[MAP_ROWS][MAP_COLUMNS];
```

[0][0]	[0][1]	[0][2]	[0][3]	[0][4]	[0][5]	[0][6]	[0][7]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]	[1][5]	[1][6]	[1][7]
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]	[2][5]	[2][6]	[2][7]
[3][0]	[3][1]	[3][2]	[3][3]	[3][4]	[3][5]	[3][6]	[3][7]
[4][0]	[4][1]	[4][2]	[4][3]	[4][4]	[4][5]	[4][6]	[4][7]
[5][0]	[5][1]	[5][2]	[5][3]	[5][4]	[5][5]	[5][6]	[5][7]
[6][0]	[6][1]	[6][2]	[6][3]	[6][4]	[6][5]	[6][6]	[6][7]
[7][0]	[7][1]	[7][2]	[7][3]	[7][4]	[7][5]	[7][6]	[7][7]

map[4][0].tile
map[4][0].player_action

PROBLEM TIME

SIMILAR TO YOUR ASSN 1

So, each one (for example the cell at row 4 and col 0 initialised with empty entity and clean space):

```
struct location map[MAP_ROWS][MAP_COLUMNS];
```

```
struct location {
    enum tile tile;
    enum player_action player_action;
};
```

```
enum tile {
    EMPTY,
    WALL,
    CHURRO,
    TRAP,
    TELEPORT,
    PLAYER
};
```

```
enum player_action {
    MOVE_UP,
    MOVE_DOWN,
    MOVE_LEFT,
    MOVE_RIGHT,
    NOTHING,
    COLLECT
};
```

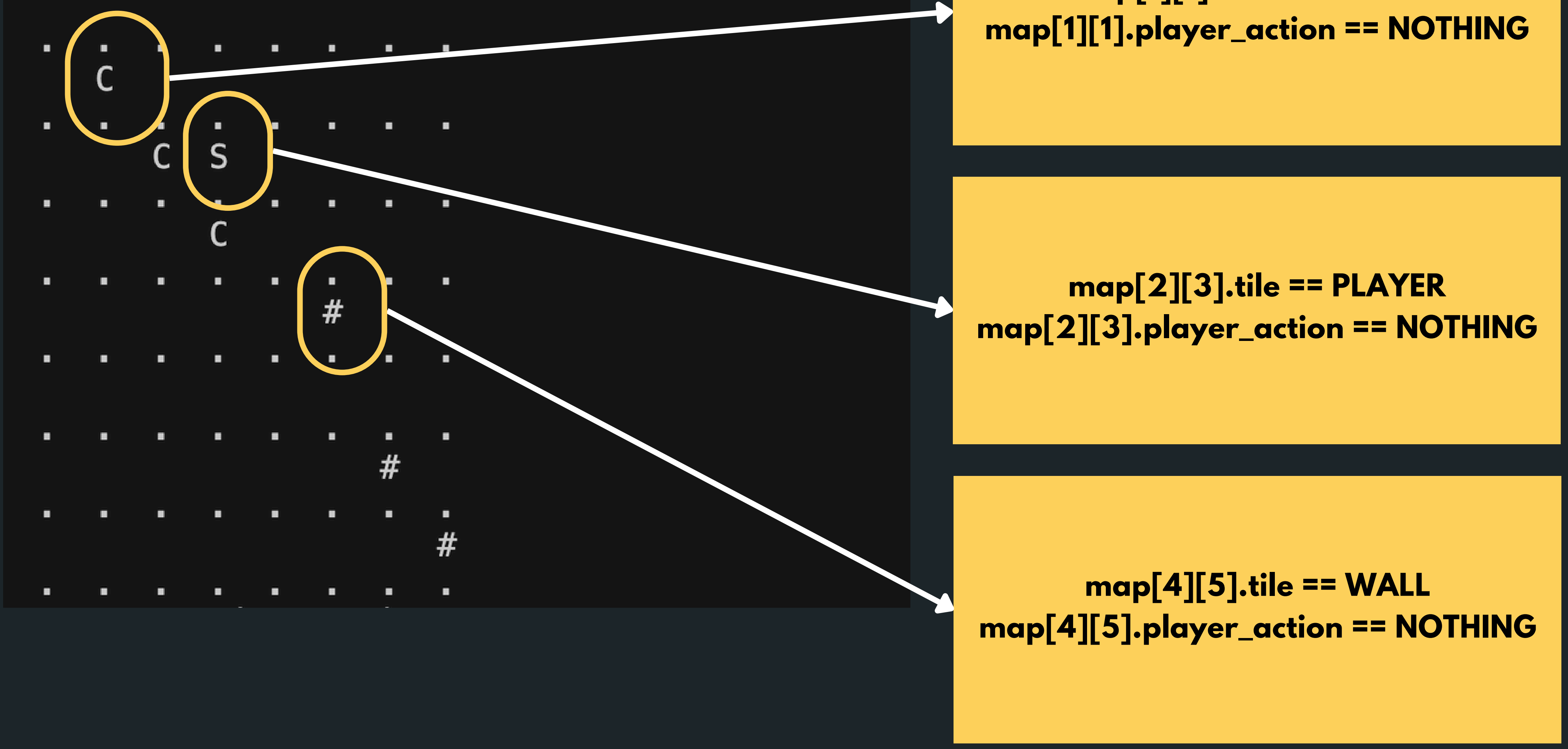
map[4][0].tile == EMPTY
map[4][0].player_action == NOTHING

PROBLEM TIME

SIMILAR TO YOUR ASSN 1

So it looks something like this once initialised:

```
Where are you? 2 3
How many churros would you like to place? 3
Where would you like to place the churro? 1 1
Where would you like to place the churro? 2 2
Where would you like to place the churro? 3 3
How many walls would you like to place? 3
Where would you like to place the wall? 4 5
Where would you like to place the wall? 6 6
Where would you like to place the wall? 7 7
```



ASSN1

STYLE TIPS

Follow the style guide, but some simple things to watch out for:

- Functions
- #defines for magic numbers
- comments
- line length

MULTI-FILE PROJECTS

WHAT ARE THEY?

- Big programs are often spread out over multiple files. There are a number of benefits to this:
  - Improves readability (reduces length of program)
  - You can separate code by subject (modularity)
  - Modules can be written and tested separately
- So far we have already been using the multi-file capability. Every time we #include, we are actually borrowing code from other files
- We have been only including C standard libraries

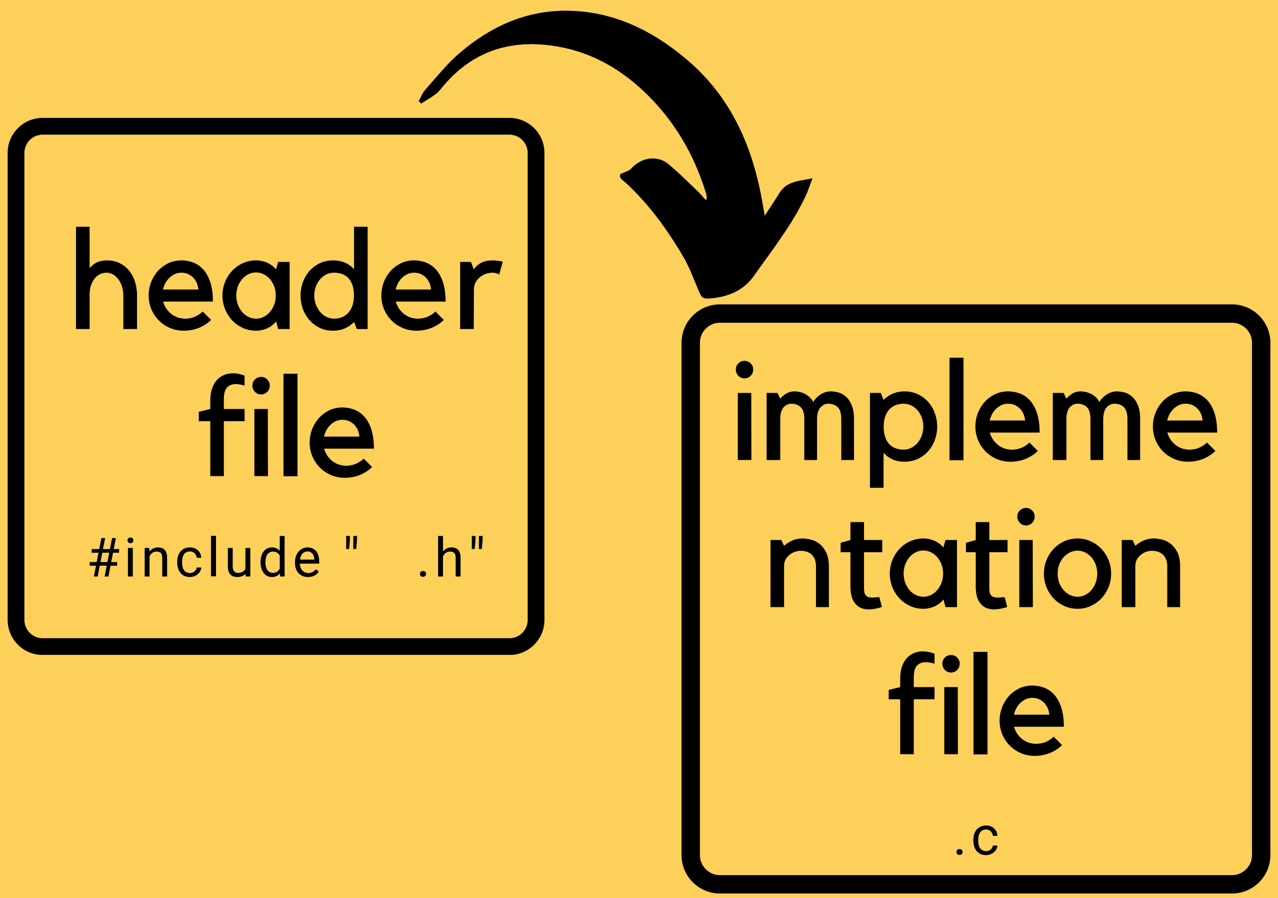
MULTI-FILE PROJECTS

WHAT ARE THEY?

- You can also #include your own! (FUN!)
- This allows us to join projects together
- It also allows multiple people to work together on projects out in the real world
- We will also often produce code that we can then use again in other projects (that is all that the C standard libraries are - functions that are useful in multiple instances)

MULTI-FILE PROJECTS

.H AND .C



- In a multi file project we might have:
  - (multiple) header file - this is the .h file that you have been using from standard libraries already
  - (multiple) implementation file - this is a .c file, it implements what is in the header file.
- Each header file that you write, will have its own implementation file
- a main.c file - this is the entry to our program, we try and have as little code here as possible

MULTI-FILE PROJECTS

.H
HEADER FILE

header file

#include "   .h"

- Typically contains:
  - function prototypes for the functions that will be implemented in the implementation file
  - comments that describe how the functions will be used
  - #defines
  - the file basically SHOWS the programmer all they need to know to use the code
  - NO RUNNING CODE
  - This is like a definition file

MULTI-FILE PROJECTS

.C IMPLEMENTATION

implementation file

.c

This is where you implement the functions that you have defined in your header file

MULTI-FILE PROJECTS

MAIN.C

This is where you call functions from that may exist in other modules.

MULTI-FILE PROJECTS

AN EXAMPLE

- We will have three files:
  - header file - maths.h
  - implementation file - maths.c
    - #include ”maths.h”
  - main file - main.c
    - #include ”maths.h”

MULTI-FILE PROJECTS

AN EXAMPLE HEADER FILE

```c
// This is the header file for the maths module
// example. The header file will contain:
// - any #defines
// - function prototypes and any comments

#define PI 3.14

// Function prototype for a function that
// calculate the square of a number:
int square(int number);

// Function prototype that calculates the sum of
// of two numbers
int sum(int number_one, int number_two);
```

MULTI-FILE PROJECTS

AN EXAMPLE IMPLEMENTATION FILE (NOTE TO INCLUDE THE HEADER THAT WE DEFINED!

```c
// This is the implementation file of maths.h
// We defined two functions in the header file (.h)
// and this is where we actually implement them

// Include your header file in the implementation file
// by using the below syntax:

#include "maths.h"

int square(int number) {
    return number * number;
}

int sum(int number_one, int number_two) {
    return number_one + number_two;
}
```

MULTI-FILE PROJECTS

AN EXAMPLE OF MAIN THAT DRIVES OUR PROGRAM

```c
// This is the main file in our program.
// This is where we drive the program from
// and where we make calls to our modules. We
// need to include the header file for each
// module that we want to use functions from.

#include <stdio.h>
// Include our header file also
#include "maths.h"

int main(void) {
    int number_one = 13;
    int number_two = 10;

    printf("The square of the number %d is %d\n",
                            number_one, square(number_one));
    printf("The sum of %d and %d is %d\n",
                number_one, number_two, sum(number_one, number_two));
    return 0;
}
```

MULTI-FILE PROJECTS

COMPILING

To compile a multi file, you basically list any .c files you have in your project (in the case of our example, we have a maths.c and a main.c file):

```
avas605@vx3:~/maths_module$ dcc maths.c main.c -o maths
avas605@vx3:~/maths_module$ ./maths
The square of the number 13 is 169
The sum of 13 and 10 is 23
avas605@vx3:~/maths_module$
```

The program will always enter in main.c, so there should only be one main.c when compiling

Feedback please!

I value your feedback and use to pace the lectures and improve your overall learning experience. If you have any feedback from today's lecture, please follow the link below. Please remember to keep your feedback constructive, so I can action it and improve the learning experience.

https://forms.office.com/r/Z8WRbaF5ug

WHAT DID WE LEARN TODAY?

LECTURE PROGRAM

churros.c

MULTI FILE PROJECTS

maths.c

maths.h

main.c

REACH OUT

CONTENT RELATED QUESTIONS

Check out the forum

ADMIN QUESTIONS

cs1511@unsw.edu.au