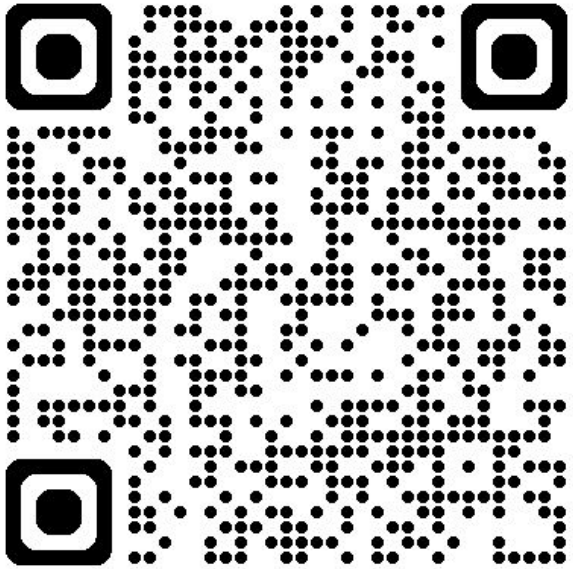
COMP1511/1911 Programming Fundamentals

Week 5 Lecture 1

Command Line Arguments
Lecture Program 2D Arrays of structs

Link to Week 5 Live Lecture Code

https://cgi.cse.unsw.edu.au/~cs1511/25T3/code/week_5/



Last Week

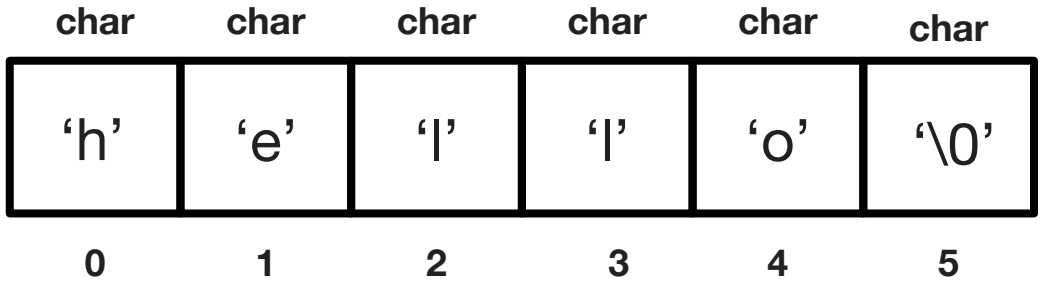
- 2D Arrays
- Strings
- We did not get up to arrays of strings or command line arguments

Today's Lecture

- Recap strings
- Array of strings
- Command line args
- Revision: A bigger 2D array of structs with enums program!
  - mud_and_bones.c
  - Putting together concepts needed in assn1
  - Style tips for assn 1

Strings recap: What are they?

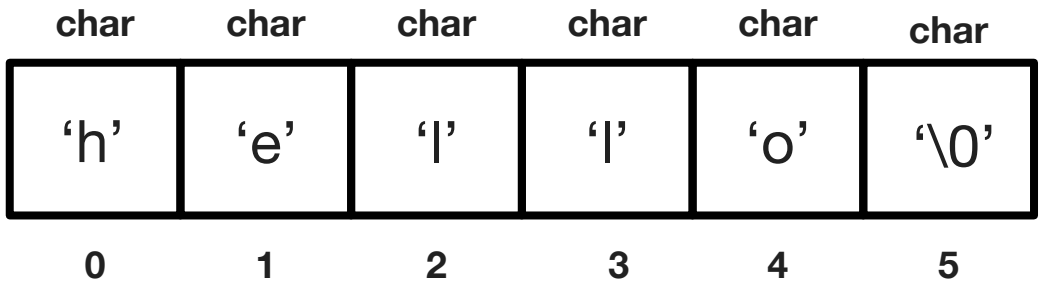
- Strings are a collection of characters
- In C a string is
  - an array of **`char`**
  - that ends with a special character `'\0'` (null terminator)



Printing Strings

```c
char word[] = "hello";
int i = 0;
while (word[i] != '\0') {
    printf("%c", word[i]);
    i++;
}
```

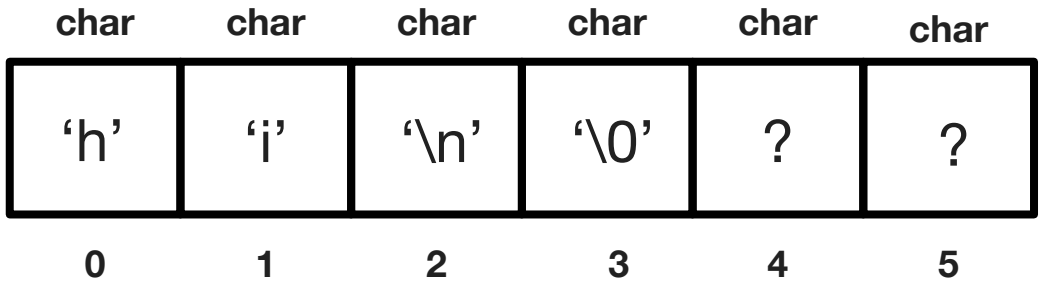
```c
// the easy way
// using printf with %s
char word[] = "hello";
printf("%s", word);
```



Strings: How do we read them in?

```
char array[MAX_LENGTH];
// Read in the string into array of length MAX_LENGTH
// from standard input - which by default is the terminal
fgets(array, MAX_LENGTH, stdin);
```

Assume **MAX_LENGTH** is 6 and the user types in **hi** then presses **enter** we would get an array like:



string.h library functions

Some other useful functions for strings:

`strlen()`	gives us the length of the string excluding the '\0'
`strcpy()`	copy the contents of one string to another
`strcmp()`	compare two strings
`strcat()`	append one string to the end of another (concatenate)
`strchr()`	find the first occurance of a character in a string

Find more here: https://www.tutorialspoint.com/c_standard_library/string_h.htm

String Functions: strcpy strlen

```c
// Declare an array to store a string
char puppy[MAX_LENGTH] = "Boots";

// Copy the string "Finn" into the word array
// be careful the array is big enough so you do not overflow
// the array
strcpy(puppy, "Finn");
printf("%s\n", puppy);
// Find string length. It does NOT include '\0' in the length
int len = strlen(puppy);
printf("%s has length %d\n", puppy, len);
```

String Functions: strcmp

```c
// Declare an array to store a string
char name[] = "Oscar";

// Use strcmp to compare 2 strings
// It will return 0 if the strings are equal
// A negative number if the first string < second string
// A positive number if the first string > second string
if (strcmp("Oscar", name) == 0) {
    printf("Hello Oscar!\n");
} else {
    printf("You are not Oscar!\n");
}
```

String Functions: fgets and strcmp

```c
// Declare an array to store a string
char name[MAX_LENGTH];
printf("Type in a name: ");

// Read in a string
fgets(name, MAX_LENGTH, stdin);

// Use strcmp to compare 2 strings
if (strcmp("Oscar", name) == 0)
    printf("Hello Oscar!\n");
} else {
    printf("You are not Oscar!\n");
}
```

What issue would we get here?

String Functions: Use with care

What is wrong with this?

```c
// Declare an array to store a string
char favourite_food[] = "kfc";

// Actually pizza is now my favourite food
strcpy(favourite_food, "pizza");
printf("%s\n", favourite_food);
```

Try running string_functions_danger.c

String Functions Demo

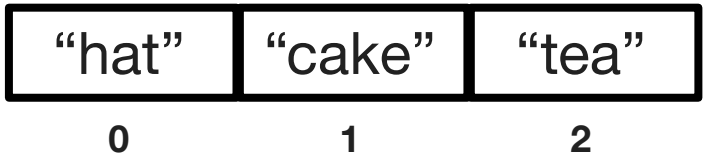
string_functions.c

string_functions_danger.c

Array of Strings

```
// This array can store 3 strings.
// Each string has max size 5, including '\0'
char words[3][5] = {"hat", "cake", "tea"};
```

- You can have an array of strings!
- You can also think of it as a 2D array of characters

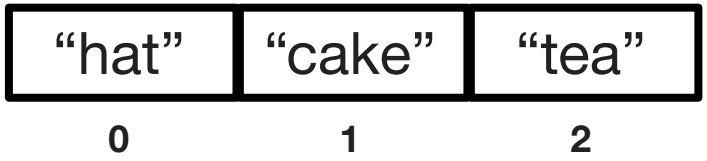


	col 0	col 1	col 2	col 3	col 4
row 0	‘h’	‘a’	‘t’	‘\0’	
row 1	‘c’	‘a’	‘k’	‘e’	‘\0’
row 2	‘t’	‘e’	‘a’	‘\0’	

Array of Strings

```c
char words[3][5] = {"hat", "cake", "tea"};
// Using 1 index gives us a row/string
// This would print "cake"
printf("%s\n", words[1]);
```

- You can have an array of strings!
- You can also think of it as a 2D array of characters

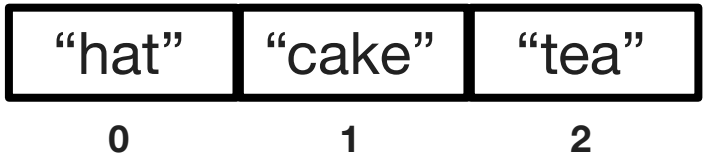


	col 0	col 1	col 2	col 3	col 4
row 0	'h'	'a'	't'	'\0'	
row 1	'c'	'a'	'k'	'e'	'\0'
row 2	't'	'e'	'a'	'\0'	

Array of Strings

```
char words[3][5] = {"hat", "cake", "tea"};
// Using 2 indexes gives us a character
// This would print the 'e' from "tea"
printf("%c\n",words[2][1]);
```

- You can have an array of strings!
- You can also think of it as a 2D array of characters



	col 0	col 1	col 2	col 3	col 4
row 0	'h'	'a'	't'	'\0'	
row 1	'c'	'a'	'k'	'e'	'\0'
row 2	't'	'e'	'a'	'\0'	

Code Demo: Array of Strings

array_of_strings.c

- initialise data
- print out data

What are Command Line Arguments?

Command Line Arguments

- So far, we have only given input to our program after we have started running that program (using scanf() or fgets())
- Our main function prototype has always been

  `int main(void);`
- Command line arguments allow us to give inputs to our program at the time that we start running it! E.g.

```
$ dcc prog.c -o prog
$ ./prog argument1 argument2 argument3 argument4
$ ./prog 123 hello
```

Command Line Arguments

- To use command line arguments you need to change your main function prototype to

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

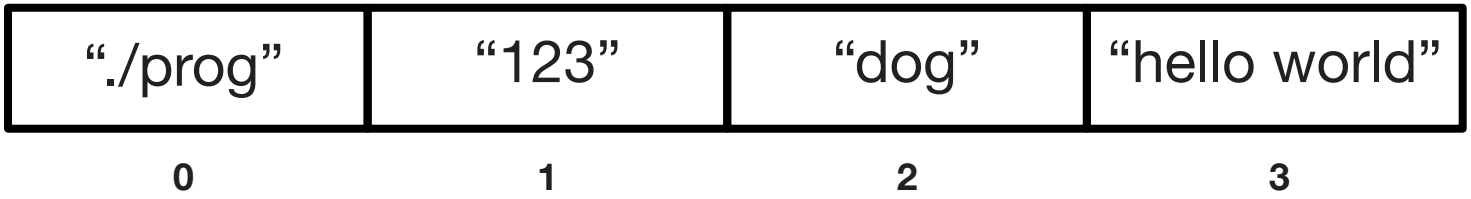
- `argc`
  - a counter for how many command line arguments you have (including the program name)
- `char *argv[]`
  - an array of the different command line arguments
  - each command line argument is a string (an array of char)

Command Line Arguments

- If we ran our program as follows:

```
$ ./prog 123 dog "hello world"
```

- argc would be equal to 4
- argv would be an array of strings we can visualise as follows:



Command Line Arguments

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

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

    // print out all arguments in the argv array
    for (int i = 0; i < argc; i++) {
        printf("Argument at index %d is %s\n", i, argv[i]);
    }
    return 0;
}
```

Command Line Arguments

```
$ dcc -o command_line_args command_line_args.c
$ ./command_line_args 123 dog "Hello World" COMP1511
This program has 5 command line arguments
This program name is ./command_line_args
Argument at index 0 is ./command_line_args
Argument at index 1 is 123
Argument at index 2 is dog
Argument at index 3 is Hello World
Argument at index 4 is COMP1511
```

Converting Strings to Integers: atoi

- You may want to use your command line arguments to perform calculations, but they are strings!
- There is a function that converts strings to integers:
  - `atoi()` in the standard library: <stdlib.h>
  - E.g. `int x = atoi("952")`
    - Would give us a value of 952 stored in x

Converting Strings to Integers: atoi

```c
int main(int argc, char *argv[]) {
    int sum = 0;
    for (int i = 1; i < argc; i++) {
        sum = sum + atoi(argv[i]);
    }
    printf("%d is the sum of all command line args\n", sum);
    return 0;
}
```

Command Line Arguments

- command_line_args.c
- atoi_demo.c

Array Lecture Program
All the key concepts in Practice

Mud and Bones

We have the following “game”.

- A dog (the player) is moving around on a map
- The locations on the map contain either grass or mud.
- They may also contain a bone.
- The dog can move around the map to collect bones
- If he steps in mud he will spread that mud to the next location he goes to.

Mud and Bones

The game ends when:

The dog finds all the bones!

The dog steps in mud 3 times and has to go home for a bath

The player presses Ctrl^D!

Mud and Bones

Important types and constants given to you for this code

```
#define MAP_ROWS 8
#define MAP_COLUMNS 8
enum ground_type {
    GRASS,
    MUD
};
```

```
enum item_type {
    EMPTY,
    BONE
};
struct location {
    enum item_type item;
    enum ground_type ground;
};
```

The Map: 8x8 2D array of struct location

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

	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7
Row 0								
Row 1								
Row 2								
Row 3								
Row 4								
Row 5								
Row 6								
Row 7								

The Map: 8x8 2D array of struct location

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

	Col 0	Col 1	Col 2
Row 0	EMPTY GRASS	BONE MUD	EMPTY GRASS
Row 1	EMPTY MUD	BONE GRASS	EMPTY MUD
Row 2	EMPTY GRASS	EMPTY GRASS	BONE GRASS
Row 3	EMPTY GRASS	EMPTY GRASS	EMPTY GRASS

If we zoom into a section of the map, we can see each one is a struct location with an item type and a ground type

The Map: 8x8 2D array of struct location

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

	Col 0	Col 1	Col 2
Row 0	EMPTY GRASS	BONE MUD	EMPTY GRASS
Row 1	EMPTY GRASS	BONE GRASS	EMPTY GRASS
Row 2	EMPTY GRASS	EMPTY GRASS	EMPTY GRASS

In this example

map[0][1].item
has the value BONE

map[0][1].ground has
the value MUD

Mud and Bones

Provided Function Prototypes

```
void initialise_map(struct location map[MAP_ROWS][MAP_COLUMNS]);
void print_map(
    struct location map[MAP_ROWS][MAP_COLUMNS],
    int dog_row,
    int dog_col,
    int bone_count,
    int mud_count
);
```

Mud and Bones Starter Code

- Creates a map variable
- Calls initialise on the map
- Calls print_board, passing in INVALID_INDEX for dog_row and dog_col and 0 for bone_count (0) and mud_count(0)

Mud and Bones Stage 1

- Initialise dog starting position
  - scan in co-ordinates from the user and set the dog's starting position.
  - If illegal, set to (0, 0)
  - Update the board and mud and bone counts accordingly
    - Increment the bone count and remove bones from the map once found
    - Update the changes in ground_type based on the dog's movement through mud.
  - Print the board!

Mud and Bones Stage 2

In a loop that ends with Ctrl-D (no winning yet)

- Allow the user to enter ‘w’ ‘a’ ‘s’ ‘d’ to move the dog around the map.
- Increment the bone count and remove bones from the map once found.
- Update the mud count if the dog steps in mud. (Don’t worry about spreading mud to other locations yet).
- Print the map after each move
- Note: we are assuming the user types in valid input. If they don’t then the game may crash!!!

Mud and Bones Stage 3

- Implement mud spreading
- Implement
  - winning the game by finding all the bones
  - Losing the game by stepping in mud 3 times

Assignment 1 Style Tips

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

- Functions
- #defines constants for magic numbers including ‘w’ etc
- Comments
- line length

Get feedback from

- style checker
- checking the style guide
- asking your tutor or a help session tutor to give feedback

Feedback Please!

Your feedback is valuable!

If you have any feedback from today's lecture, please follow the link below or use the QR Code.

Please remember to keep your feedback constructive, so I can action it and improve your learning experience.

https://forms.office.com/r/QD15dbp6KD

What did we learn today?

- String recap
  - string_functions.c string_functions_danger.c full_name.c
- Arrays of strings
  - arrays_of_strings.c
- Command Line Arguments
  - command_line_args.c atoi_demo.c
- 2D array of structs with enums coding example
  - mud_and_bones.c

Reach Out

Content Related Questions:

Forum

Admin related Questions email:

cs1511@unsw.edu.au