COMP1511 Programming Fundamentals

Week 4 Lecture 2

Strings, Arrays of Strings
Command Line Arguments

Last Lecture

- Recap Arrays
- Arrays of structs
- 2D Arrays

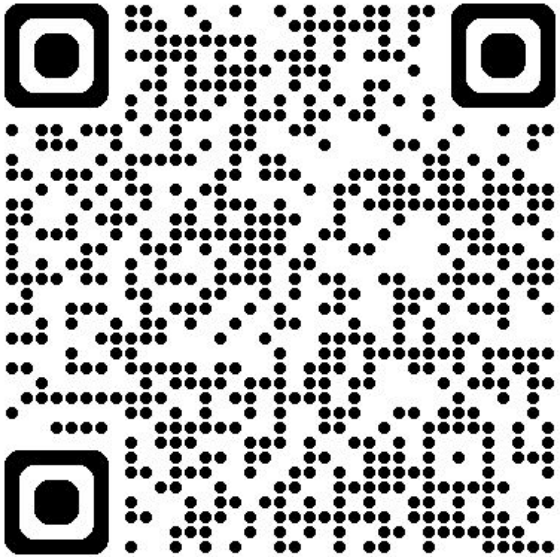
Today's Lecture

- 2D Arrays recap
- Strings
- Arrays of strings
- Command Line Arguments

Note: Anything we don't finish can be covered in the next lecture.

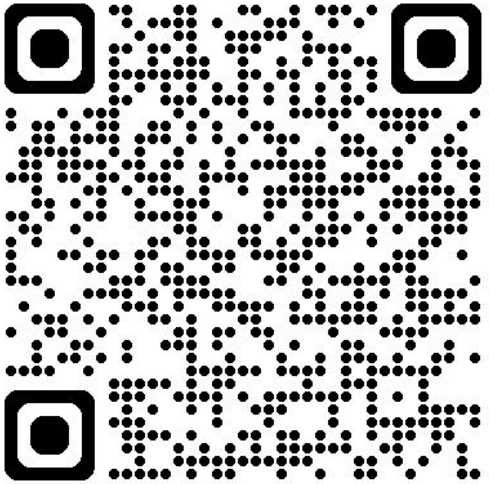
Link to Week 4 Live Lecture Code

https://cgi.cse.unsw.edu.au/~cs1511/25T1/code/week_4/



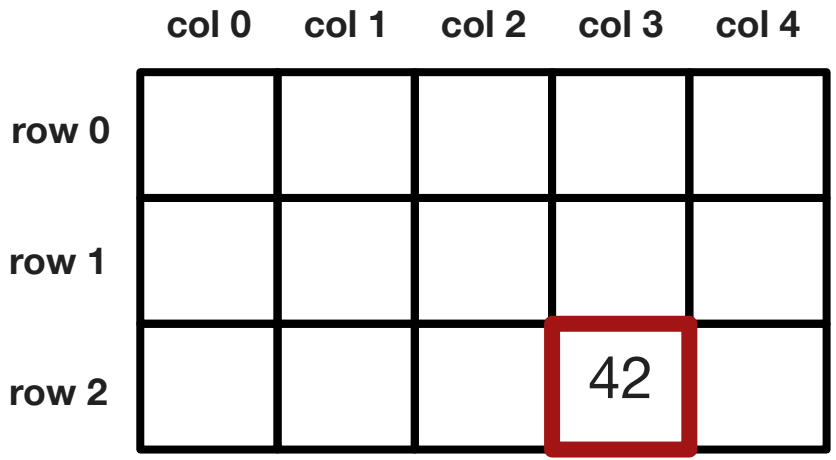
Assignment 1

- Assignment 1 has been released
- Assignment 1 walkthrough video is out!
- https://www.youtube.com/watch?v=meFfz1rR5bg
- Get started soon!



Recap: 2D Arrays: Accessing Indexes

```
// A 2D array with 3 rows and 5 columns of int
int number_grid[3][5];
// To access an element you need to give 2 indexes
number_grid[2][3] = 42;
```

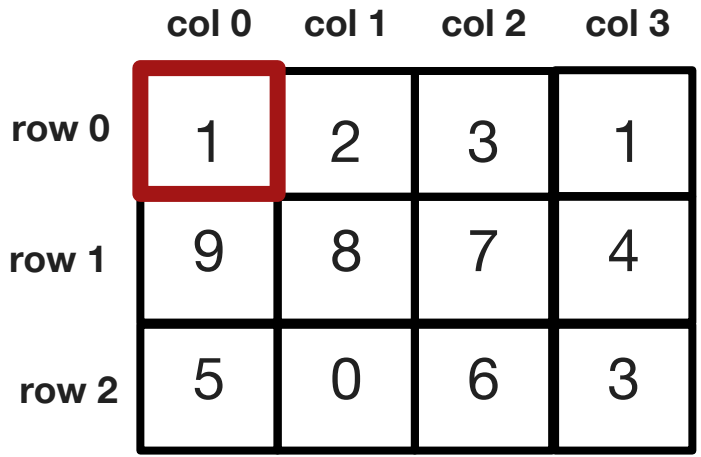


2D Arrays: Traversal

```c
// Assume ROWS is 3 and COLS is 4
int array[ROWS][COLS] = {{1, 2, 3, 1},
                         {9, 8, 7, 4},
                         {5, 0, 6, 3}};

int row = 0;
while (row < ROWS) {
    int col = 0;
    while (col < COLS) {
        printf("%d ", array[row][col]);
        col++;
    }
    printf("\n");
    row++;
}
```

Outer loop: row = **0**
Inner loop: col = 0

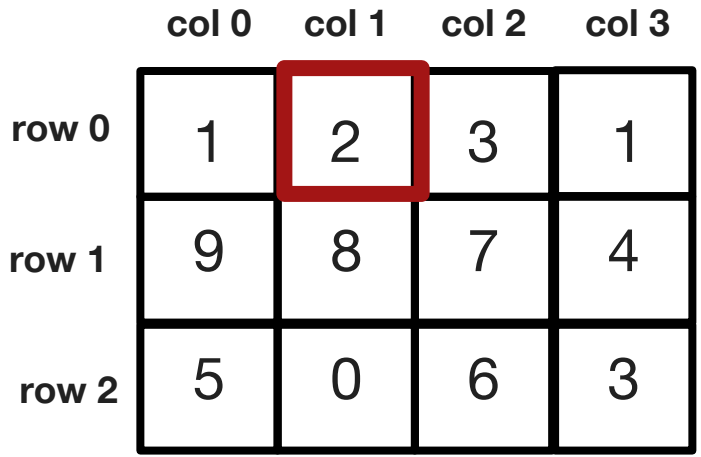


2D Arrays: Traversal

```c
// Assume ROWS is 3 and COLS is 4
int array[ROWS][COLS] = {{1, 2, 3, 1},
                         {9, 8, 7, 4},
                         {5, 0, 6, 3}};

int row = 0;
while (row < ROWS) {
    int col = 0;
    while (col < COLS) {
        printf("%d ", array[row][col]);
        col++;
    }
    printf("\n");
    row++;
}
```

Outer loop: row = **0**
Inner loop: col = 1

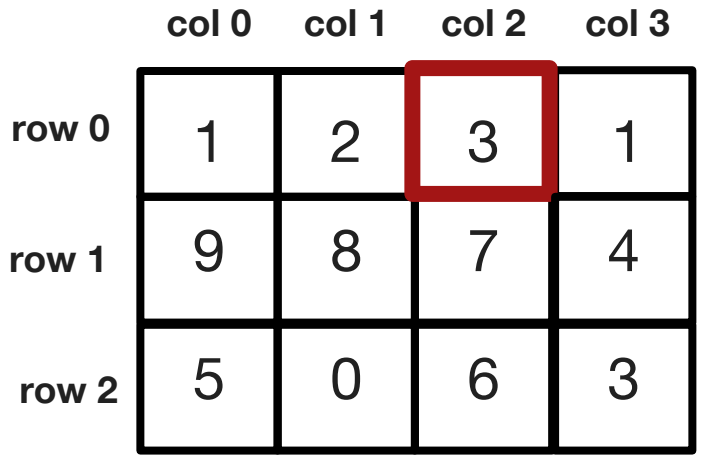


2D Arrays: Traversal

```c
// Assume ROWS is 3 and COLS is 4
int array[ROWS][COLS] = {{1, 2, 3, 1},
                         {9, 8, 7, 4},
                         {5, 0, 6, 3}};

int row = 0;
while (row < ROWS) {
    int col = 0;
    while (col < COLS) {
        printf("%d ", array[row][col]);
        col++;
    }
    printf("\n");
    row++;
}
```

Outer loop: row = **0**
Inner loop: col = 2



2D Arrays: Traversal

```c
// Assume ROWS is 3 and COLS is 4
int array[ROWS][COLS] = {{1, 2, 3, 1},
                         {9, 8, 7, 4},
                         {5, 0, 6, 3}};

int row = 0;
while (row < ROWS) {
    int col = 0;
    while (col < COLS) {
        printf("%d ", array[row][col]);
        col++;
    }
    printf("\n");
    row++;
}
```

Outer loop: row = **0**
Inner loop: col = 3

	col 0	col 1	col 2	col 3
row 0	1	2	3	1
row 1	9	8	7	4
row 2	5	0	6	3

2D Arrays: Traversal

```
// Assume ROWS is 3 and COLS is 4
int array[ROWS][COLS] = {{1, 2, 3, 1},
                         {9, 8, 7, 4},
                         {5, 0, 6, 3}};

int row = 0;
while (row < ROWS) {
    int col = 0;
    while (col < COLS) {
        printf("%d ", array[row][col]);
        col++;
    }
    printf("\n");
    row++;
}
```

Outer loop: row = **1**
Inner loop: col = 0

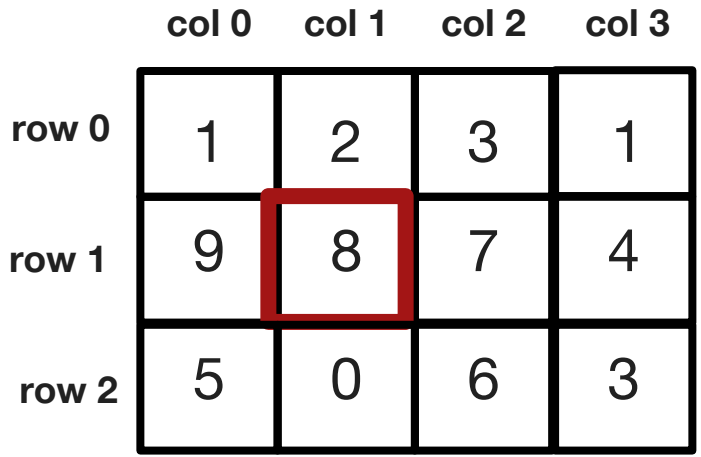
	col 0	col 1	col 2	col 3
row 0	1	2	3	1
row 1	9	8	7	4
row 2	5	0	6	3

2D Arrays: Traversal

```c
// Assume ROWS is 3 and COLS is 4
int array[ROWS][COLS] = {{1, 2, 3, 1},
                         {9, 8, 7, 4},
                         {5, 0, 6, 3}};

int row = 0;
while (row < ROWS) {
    int col = 0;
    while (col < COLS) {
        printf("%d ", array[row][col]);
        col++;
    }
    printf("\n");
    row++;
}
```

Outer loop: row = **1**
Inner loop: col = 1



2D Arrays: Traversal

```
// Assume ROWS is 3 and COLS is 4
int array[ROWS][COLS] = {{1, 2, 3, 1},
                         {9, 8, 7, 4},
                         {5, 0, 6, 3}};

int row = 0;
while (row < ROWS) {
    int col = 0;
    while (col < COLS) {
        printf("%d ", array[row][col]);
        col++;
    }
    printf("\n");
    row++;
}
```

Outer loop: row = **1**
Inner loop: col = 2

	col 0	col 1	col 2	col 3
row 0	1	2	3	1
row 1	9	8	7	4
row 2	5	0	6	3

2D Arrays: Traversal

```c
// Assume ROWS is 3 and COLS is 4
int array[ROWS][COLS] = {{1, 2, 3, 1},
                         {9, 8, 7, 4},
                         {5, 0, 6, 3}};

int row = 0;
while (row < ROWS) {
    int col = 0;
    while (col < COLS) {
        printf("%d ", array[row][col]);
        col++;
    }
    printf("\n");
    row++;
}
```

Outer loop: row = **1**
Inner loop: col = 3

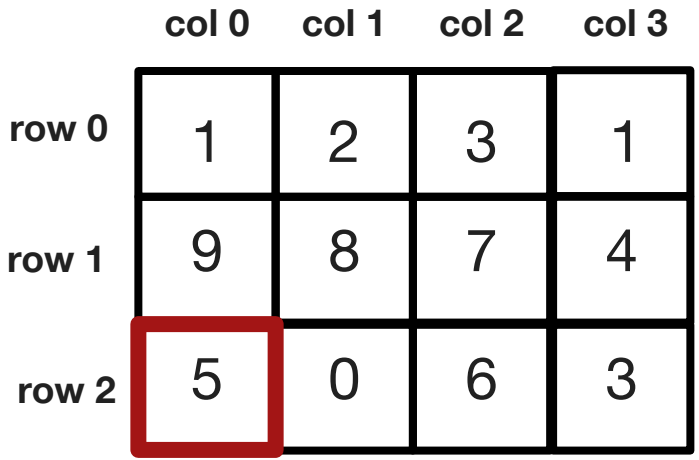
	col 0	col 1	col 2	col 3
row 0	1	2	3	1
row 1	9	8	7	4
row 2	5	0	6	3

2D Arrays: Traversal

```c
// Assume ROWS is 3 and COLS is 4
int array[ROWS][COLS] = {{1, 2, 3, 1},
                         {9, 8, 7, 4},
                         {5, 0, 6, 3}};

int row = 0;
while (row < ROWS) {
    int col = 0;
    while (col < COLS) {
        printf("%d ", array[row][col]);
        col++;
    }
    printf("\n");
    row++;
}
```

Outer loop: row = **2**
Inner loop: col = 0

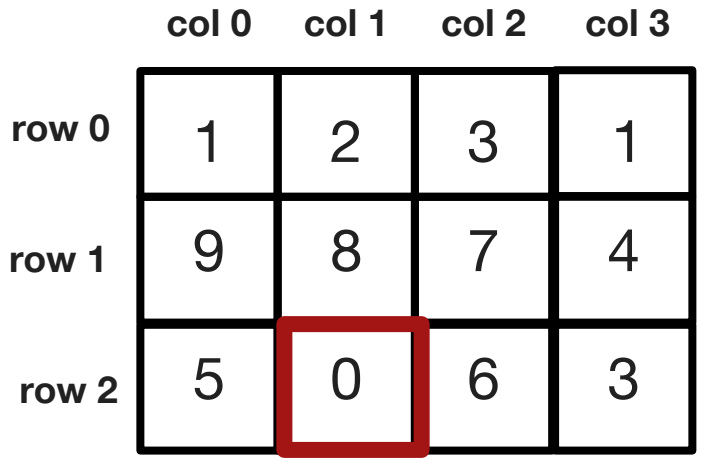


2D Arrays: Traversal

```c
// Assume ROWS is 3 and COLS is 4
int array[ROWS][COLS] = {{1, 2, 3, 1},
                         {9, 8, 7, 4},
                         {5, 0, 6, 3}};

int row = 0;
while (row < ROWS) {
    int col = 0;
    while (col < COLS) {
        printf("%d ", array[row][col]);
        col++;
    }
    printf("\n");
    row++;
}
```

Outer loop: row = **2**
Inner loop: col = 1



2D Arrays: Traversal

```c
// Assume ROWS is 3 and COLS is 4
int array[ROWS][COLS] = {{1, 2, 3, 1},
                         {9, 8, 7, 4},
                         {5, 0, 6, 3}};

int row = 0;
while (row < ROWS) {
    int col = 0;
    while (col < COLS) {
        printf("%d ", array[row][col]);
        col++;
    }
    printf("\n");
    row++;
}
```

Outer loop: row = **2**
Inner loop: col = 2

	col 0	col 1	col 2	col 3
row 0	1	2	3	1
row 1	9	8	7	4
row 2	5	0	6	3

2D Arrays: Traversal

```c
// Assume ROWS is 3 and COLS is 4
int array[ROWS][COLS] = {{1, 2, 3, 1},
                         {9, 8, 7, 4},
                         {5, 0, 6, 3}};

int row = 0;
while (row < ROWS) {
    int col = 0;
    while (col < COLS) {
        printf("%d ", array[row][col]);
        col++;
    }
    printf("\n");
    row++;
}
```

Outer loop: row = **2**
Inner loop: col = 3

	col 0	col 1	col 2	col 3
row 0	1	2	3	1
row 1	9	8	7	4
row 2	5	0	6	3

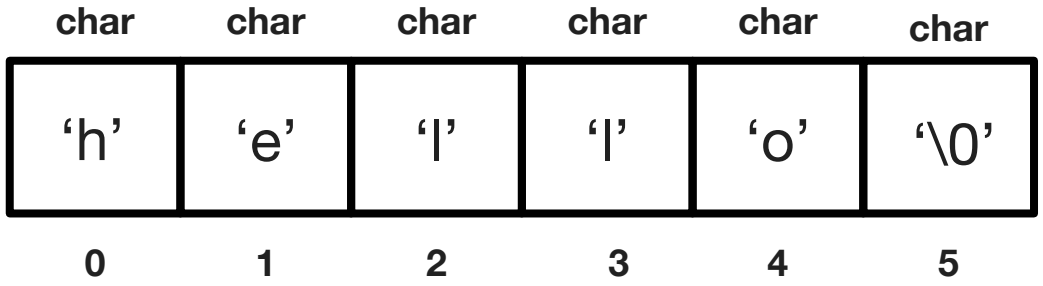
Recap of 2D Arrays Coding

- 2D_array.c
  - copy array
- diagonals.c
  - sum_diagonal_top_right

Strings

Strings: 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)

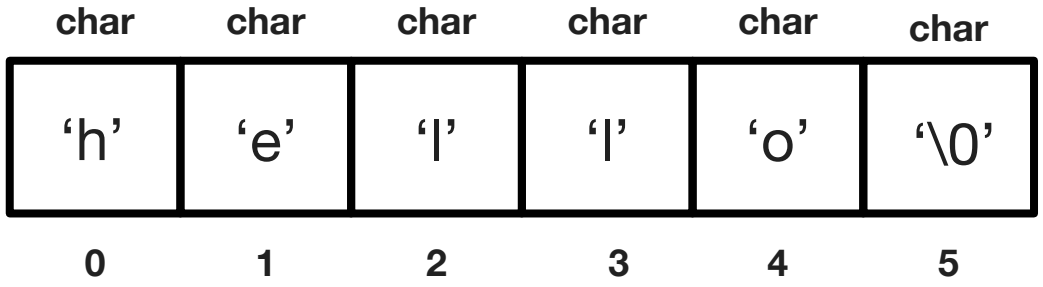


Null Terminator

- The null terminator ‘\0’ must be at the end of every string
  - If it does not have one it is not a string! Just an array of char
- The array must be big enough to store the extra character
- It is not displayed as part of the string
- It is very useful to know when our string has come to an end, when we loop through the array of characters
- Anything in the array after the ‘\0’ is not part of the string

Strings: How do we initialise them?

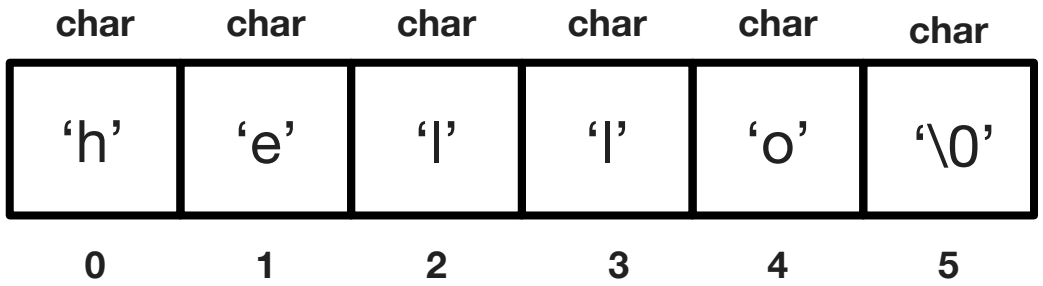
```c
// the painful way
char word[] = {'h','e','l','l','o','\0'};
// the more convenient way which does the same thing
char word[] = "hello";
```



Strings: How do we print them?

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

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



Code Demo

simple_strings.c

- declaring,
- initialising,
- modifying,
- printing strings,
- writing our own printing function

Strings: Can I read them in with scanf %s?

- No. Please don’t. It can read strings that are too long to fit in the array
  - Overwrite other memory - **buffer overflow**
  - Security Vulnerability
  - Hackers can exploit this
  - You will see more about this in COMP1521
- It may not do what you expect/want anyway
  - Stops when it encounters whitespace
- It is forbidden in the style guide. You will lose marks for using it

Strings: How do we read them in?

We fgets them: `fgets(array, size, stream);`

`fgets` needs three inputs:

- array - the array that the string will be stored into
- size - the size of the array
  - fgets will only read in and store a max of `size - 1` characters
- stream - this is where this string is coming from
  - For this course it will always be `stdin` (standard input: by default the input will always be from terminal)

Strings: How do we read them in?

One call to fgets will read in characters until

- size - 1 characters are read in
- a newline character is read in
  - this newline character is stored in the array
- we get to the end of file
  - which is Ctrl+D on a line of its own for terminal input

Note: There is a matching function that prints strings out

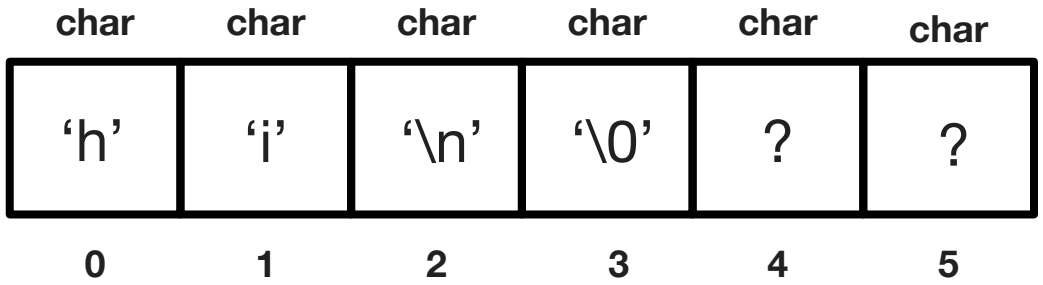
```
fputs(array, stream);
```

- For this course stream will always be `stdout` (terminal)

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:



Strings: How do we read in many of them?

```c
// Declare an array to store your string
char array[MAX_LENGTH];

printf("Type in a string to echo: ");
// Read a string into array again and again
// until Ctrl+D is pressed (indicated by fgets returning NULL)
while (fgets(array, MAX_LENGTH, stdin) != NULL) {
    printf("The string is:\n");
    printf("%s", array);
    printf("Type in a string to echo: ");
}
```

Note: We are only ever storing 1 string at a time with this code

ctype.h library functions

You can use these to make your life easier when working with characters!

`toupper()`	convert a character to uppercase
`tolower()`	convert a character to lowercase
`isupper()`	test whether a character is uppercase
`islower()`	test whether a character is lowercase

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

Code Demo: Reading in Strings

read_strings.c

- Read in a string
- Try reading in a string that is too long
- Repeatedly read in a string
- Convert string to all capitals

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("Edgar", 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?

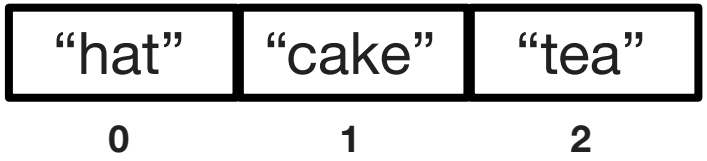
String Functions Demo

string_functions.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

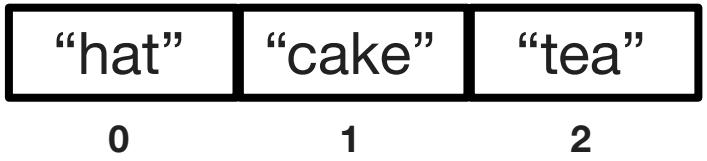


	col 0	col 1	col 2	col 3	col 4
row 0	‘h’	‘a’	‘t’	‘\0’	
row 1	‘c’	‘a’	‘k’	‘e’	‘\0’
row 2	‘t’	‘e’	‘a’	‘\0’	

Array of Strings

```
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

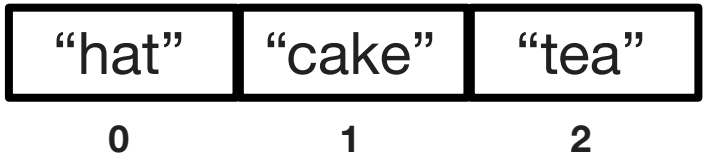


	col 0	col 1	col 2	col 3	col 4
row 0	'h'	'a'	't'	'\0'	
row 1	'c'	'a'	'k'	'e'	'\0'
row 2	't'	'e'	'a'	'\0'	

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'	

Coding 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[])
  ```

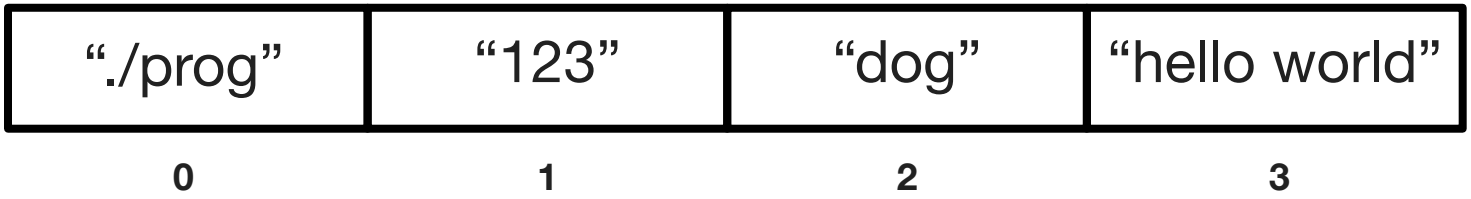
- `argc`
  - a counter for how many command line arguments you have (including the program name)
- `char *argv[]`
  - an array of the different command line arguments
  - each command line argument is a string (an array of char)

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

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/EUq6MitZp4

What did we learn today?

- recap 2D arrays
  - 2D_array.c, diagonals.c
- strings
  - simple_strings.c, read_strings.c, string_functions.c
- arrays of strings
  - array_of_strings.c
- command line arguments
  - command_line_args.c, atoi_demo.c

Enjoy the Rest of the Term!

Sasha will be back next week!

I hope you all learn a lot this term and enjoy your coding!

Have fun!!!!

Bye!!!!!!!!!!!!!!!!!!

Reach Out

Content Related Questions:
Forum

Admin related Questions email:
cs1511@unsw.edu.au