COMP1511 PROGRAMMING FUNDAMENTALS

LECTURE 8

Recap 2D arrays and Strings

ON MONDAY...

LAST LECTURE...

- Went back to reinforce 1D arrays
- Looked at 2D arrays (which make up a grid and allow us to do some pretty cool stuff)

THIS LECTURE...

TODAY

- Revisiting scanf() and EOF
- Recap of 2D arrays
- Strings!
- Command line arguments if there is time

WHERE IS THE CODE?

Live lecture code can be found here:

HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/23T1/LIVE/WEEK04/

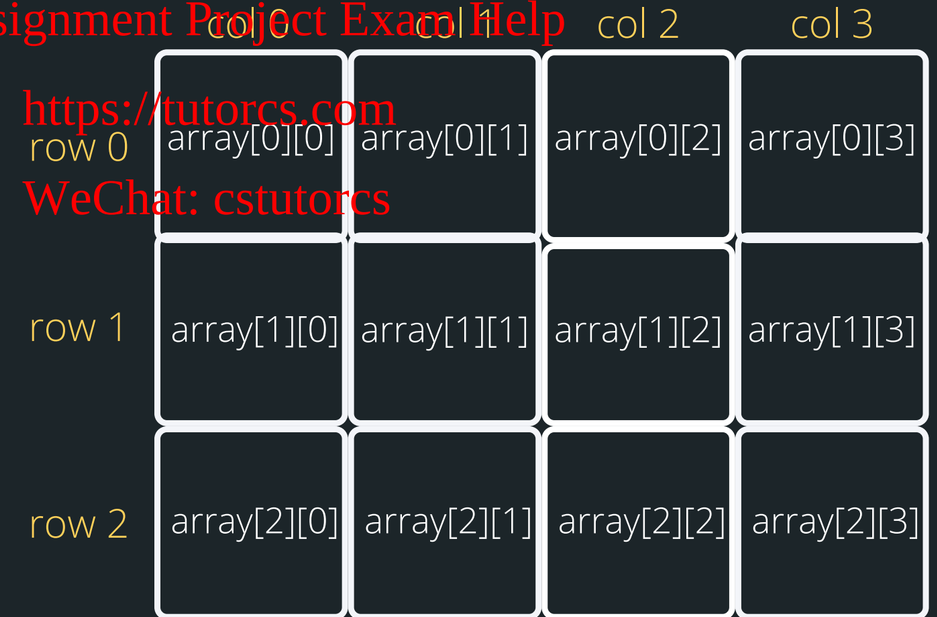
ARRAY OF ARRAYS

A RECAP

For example, let's say we declare an array of arrays:

```
int array[3][4];
```

Visually it looks like this and showing how to access each of the grid elements:



PROBLEM TIME

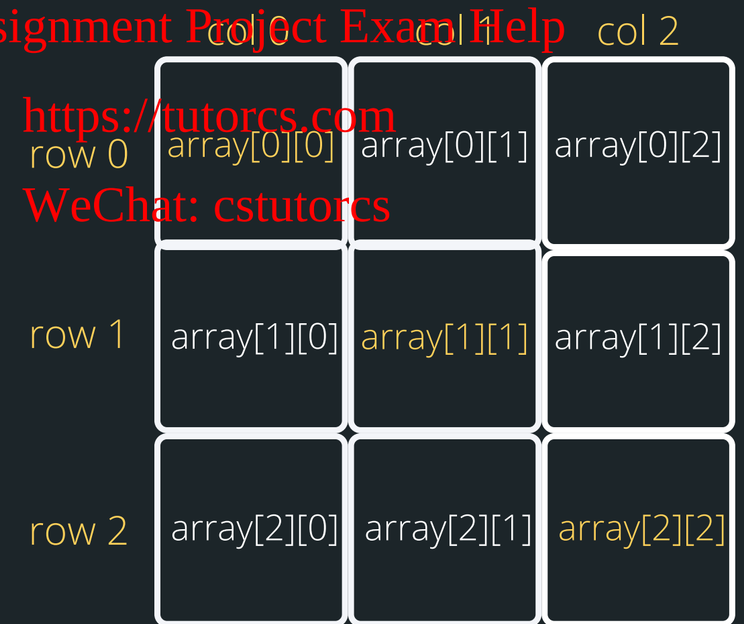
Going back to the question we finished with on Monday, let's go back and move things out into functions...

`2D_arrays.c`

PROBLEM TIME

Write a program in C to find the sum of the right diagonals of a 2D array of numbers. (Assume 2D array will always be square)

diagonals.c

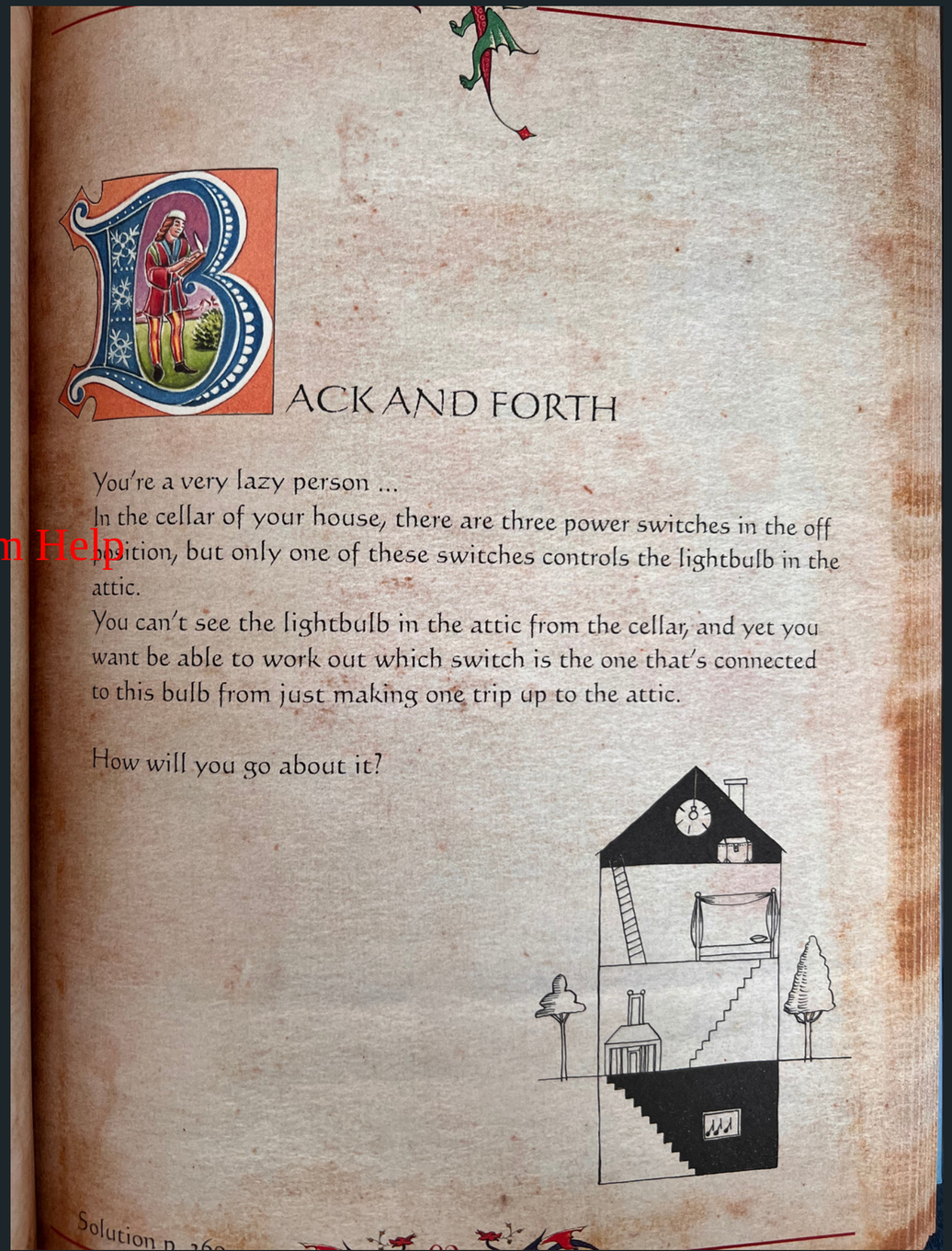


PROBLEM TIME

Now a bit harder, what about the left diagonals?

diagonals.c

BREAK TIME



BACK AND FORTH

You're a very lazy person ...
In the cellar of your house, there are three power switches in the off position, but only one of these switches controls the lightbulb in the attic.
You can't see the lightbulb in the attic from the cellar, and yet you want be able to work out which switch is the one that's connected to this bulb from just making one trip up to the attic.

How will you go about it?

Solution p. 360

STRINGS

WHAT ARE THEY?

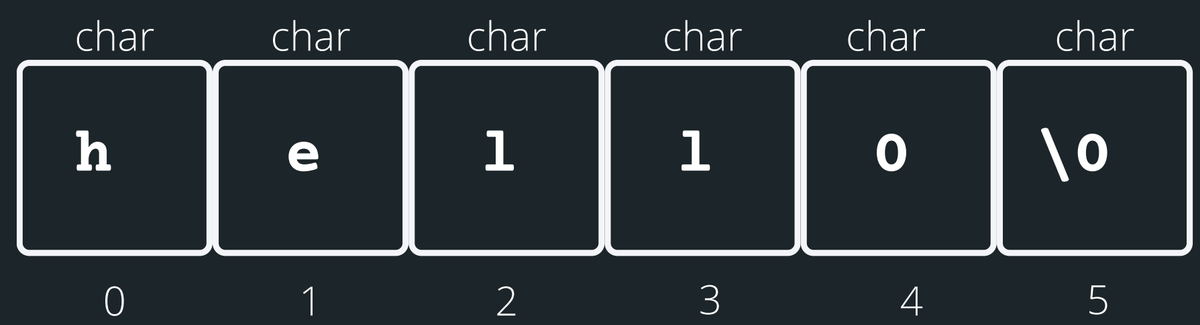
- Strings are a collection of characters that are joined together
  - an array of characters!
- There is one very special thing about strings in C - it is an array of characters that finishes with a
  - This symbol is called a null terminating character
- It is always located at the end of an array, therefore an array has to always be able to accomodate this character
- It is not displayed as part of the string
- It is a placeholder to indicate that this array of characters is a string
- It is very useful to know when our string has come to an end, when we loop through the array of characters

HOW DO WE DECLARE A STRING?

WHAT DOES IT LOOK LIKE VISUALLY?

- Because strings are an array of characters, the array type is char.
- To declare and initialise a string, you can use two methods:

```
//the more convenient way
char word[] = "hello";
//this is the same as'\0':
char word[] = {'h','e','l','l','o','\0'};
```



HELPFUL LIBRARY FUNCTIONS FOR STRINGS

FGETS()

There is a useful function for reading strings:

`fgets(array[], length, stream)`

The function needs three inputs:

- array[] - the array that the string will be stored into
- length - the number of characters that will be read in
- stream - this is where this string is coming from - you don't have to worry about this one, in your case, it will always be stdin (the input will always be from terminal)


```
// Declare an array where you will place the string that you read from somewhere
char array[MAX_LENGTH];
// Read in the string into array of length MAX_LENGTH from terminal input
fgets(array, MAX_LENGTH, sdin)
```

HOW DO I KEEP READING STUFF IN OVER AND OVER AGAIN?

Using the **`NULL`** keyword, you can continuously get string input from terminal until Ctrl+D is pressed

- fgets() stops reading when either length-1 characters are read, newline character is read or an end of file is reached, whichever comes first


```c
#include <stdio.h>

#define MAX_LENGTH 15

int main(void) {
    // Declare an array where you will place the string
    char array[MAX_LENGTH];

    printf("Type in a string to echo: ");
    // Read in the string into the array until Ctrl+D is
    // pressed, which is indicated by the NULL keyword
    while (fgets(array, MAX_LENGTH, stdin) != NULL) {
        printf("The string is: \n");
        printf("%s", array);
        printf("Type in a string to echo: ");
    }
    return 0;
}
```

HELPFUL LIBRARY FUNCTIONS FOR STRINGS

FPUTS()

Another useful function to output strings:

`fputs(array[], stream)`

The function needs two inputs:

- array[] - the array that the string is be stored in
- stream - this is where this string will be output to, you don't have to worry about this one, in your case, it will always be stdout (the output will always be in terminal)


```
// Declare an array where you will place the
string that you read from somewhere
char array[MAX_LENGTH];
// Read in the string into array of length
MAX_LENGTH from terminal input
fgets(array, MAX_LENGTH, sdin)
//Output the array now
fputs(array, stdout)
```

SOME OTHER INTERESTING STRING FUNCTIONS

<STRING.H> STANDARD LIBRARY

CHECK OUT THE REST OF THE FUNCTIONS: HTTPS://WWW.TUTORIALSPOINT.COM/C_STANDARD_LIBRARY/STRING_H.HTM

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
- **`strcat()`** attach one string to the end of another (concatenate)
- **`strcmp()`** compare two strings
- **`strchr()`** find the first or last occurance of a character

USING SOME OF THESE FUNCTIONS

STRINGS

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

#define MAX_LENGTH 15

int main(void) {
    // Declare an array
    char word_array[MAX_LENGTH];

    // Example using strcpy to copy from one string
    // to another (destination, source)
    strcpy(word_array, "Jax");
    printf("%s\n", word_array);

    // Example using strlen to find string length
    // returns the int length NOT including '\0'
    int length = strlen("Sasha");\n
    printf("The size of string 'Sasha' is %d chars\n", length);

    // Example using strcmp to compare two strings character
    // by character - function will return:
    // 0 = two strings are equal
    // other int if not the same

    int compare_string = strcmp("Jax", "Juno");
    printf("The two strings are the same: %d\n", compare_string);

    compare_string = strcmp(word_array, "Jax");
    printf("The two strings are the same: %d\n", compare_string);
    return 0;
};
```

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
```

CODE TIME :)

- Read in two numbers from the command line arguments and state whether the two numbers are the same or not

`compare_numbers.c`

- Let's make it a bit more interesting, read in two strings from the command line arguments and compare the strings to say whether they are the same or not!

`compare_strings.c`

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://www.menti.com/alafjm9rxpmy

WHAT DID WE LEARN TODAY?

2D ARRAY RECAP

2D_array.c

diagonals.c

STRINGS

echo.c

string.c

COMMAND LINE ARGUMENTS

argv_demo.c

atoi_demo.c

compare_numbers.c

compare_strings.c

REACH OUT

CONTENT RELATED QUESTIONS

Check out the forum

ADMIN QUESTIONS

cs1511@unsw.edu.au