COMP1511 PROGRAMMING FUNDAMENTALS

LECTURE 9

Lets look at strings and debugging...

LAST WEEK...

- Learnt about 1D arrays
- Looked at 2D arrays (which make up a grid and allow us to do some pretty cool stuff)
- Got introduced to pointers

TODAY...

- Look at strings

WHERE IS THE CODE?

Live lecture code can be found here:

HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/22T2/LIVE/WEEK05/

POINTERS RECAP

- A pointer is another variable that stores a memory address of a variable
- This is very powerful, as it means you can modify things at the source (this also has certain implications for functions which we will look at in a bit)
- To declare a pointer, you specify what type the pointer points to with an asterisk:

```
type_pointing_to *name_of_ variable;
```

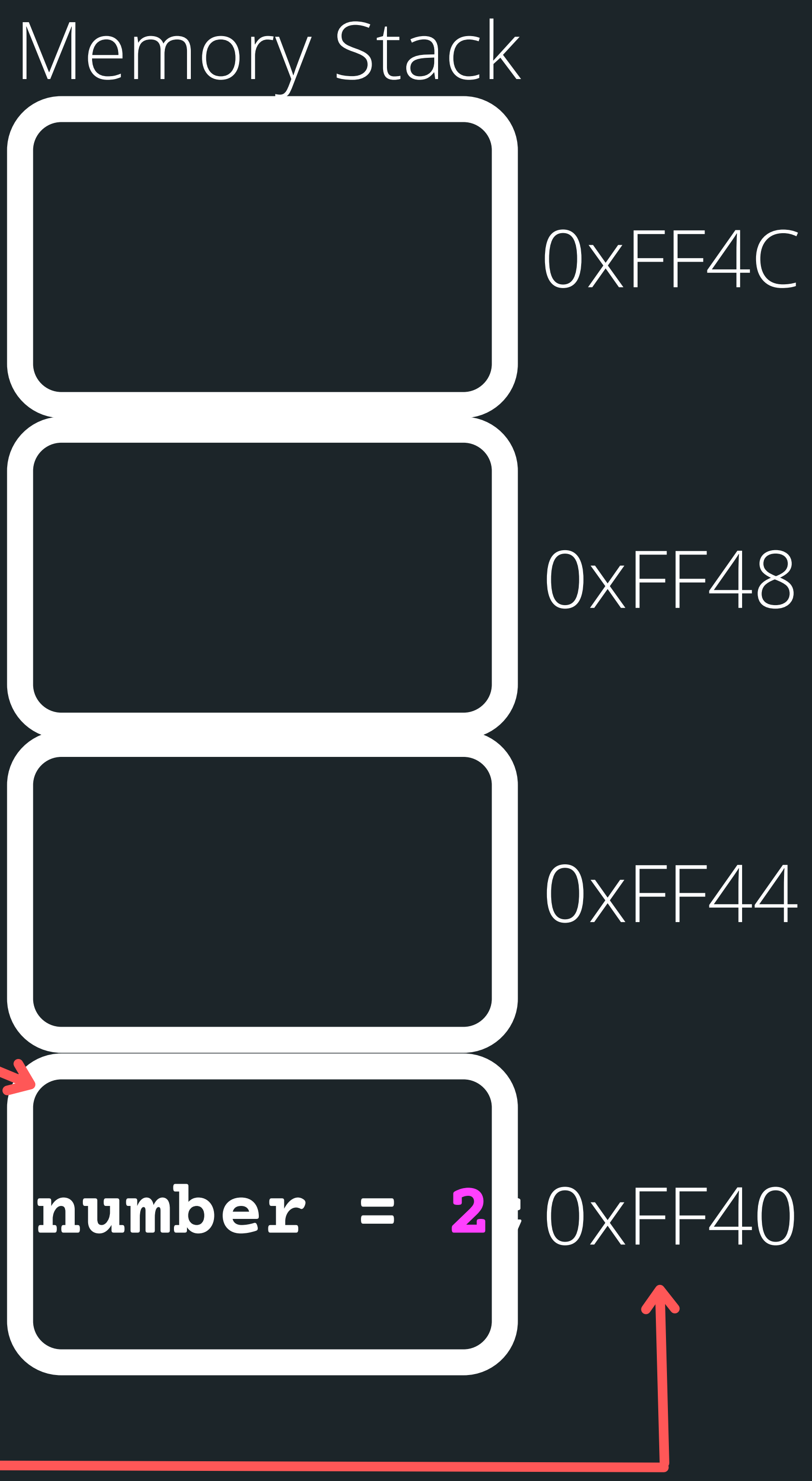
  - For example, if your pointer points to an int:

    ```
    int *pointer;
    ```

VISUALLY WHAT IS HAPPENING?

```
// Declare a variable of
// type int. called number
// Assign the value 13 to
// box
int number = 2;

// Declare a pointer
// variable that points to
// an int and assign the
// address of number to it
int *number_ptr = &number;
```



```
// So now:
number = 13
AND
number_ptr = 0xFF40
```

POINTERS

1) Declare a pointer with a * - this is where you will specify what type the pointer points to. For example, a pointer that stores the address of an int type variable:

```
int *number_ptr;
```

2) Initialise a pointer - assign the address to the variable with &

```
number_ptr = &number;
```

3) Dereference a pointer - using a * , go to the address that this pointer variable is assigned and find what is at that address

```
*number_ptr
```

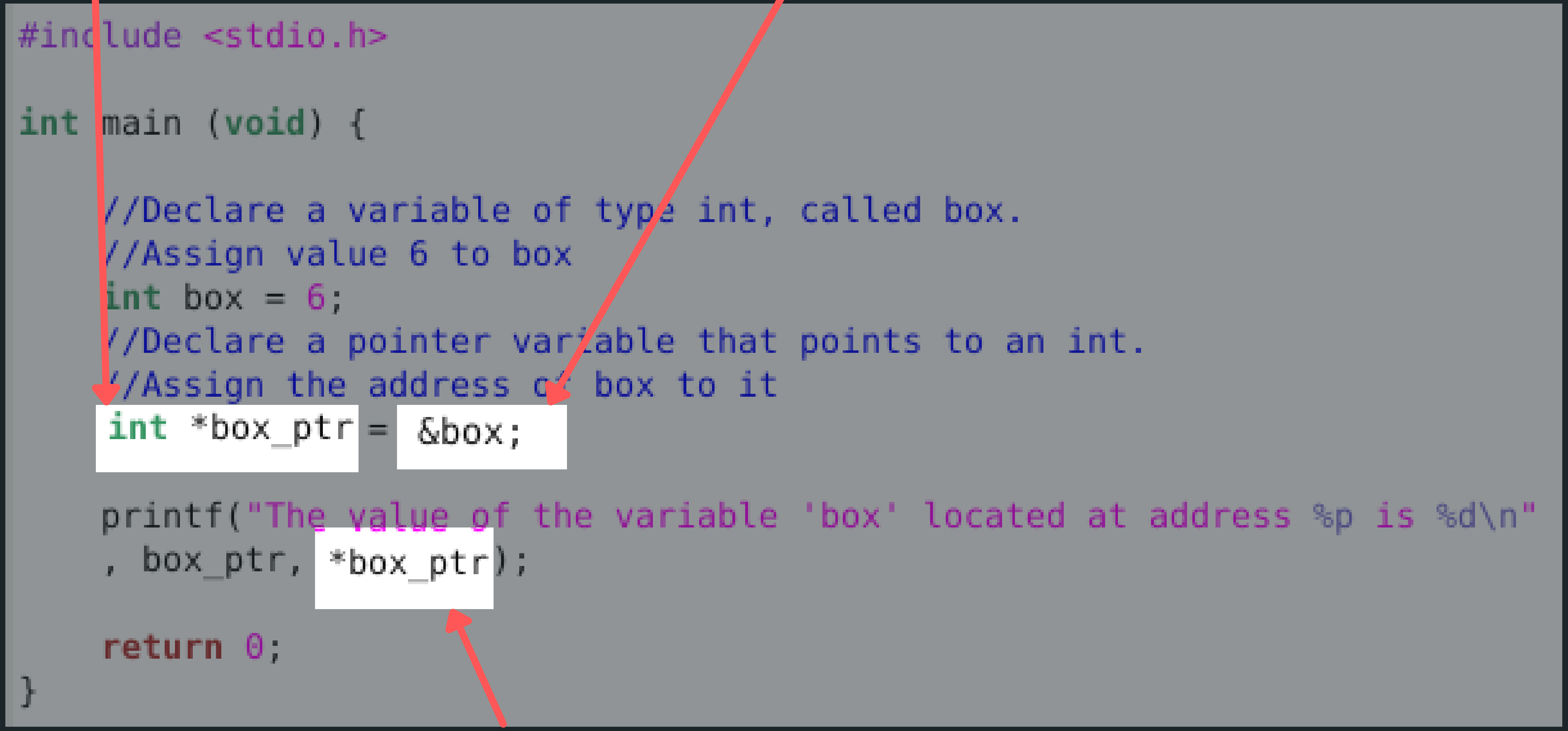
```
Achievement.....d3TecTiv3
```

POINTERS RECAP

THERE ARE THREE PARTS TO A POINTER

1. *Declare a pointer with a * - this is where you will specify what type the pointer points to*

2. Initialise a pointer - assign the address to the variable with &



```
#include <stdio.h>

int main (void) {

    //Declare a variable of type int, called box.
    //Assign value 6 to box
    int box = 6;
    //Declare a pointer variable that points to an int.
    //Assign the address of box to it
    int *box_ptr = &box;

    printf("The value of the variable 'box' located at address %p is %d\n"
    , box_ptr, *box_ptr);

    return 0;
}
```

*3. Dereference a pointer -Using a * , go to the address that this pointer variable is assigned and find what is at that address*

COMMON MISTAKES/ SYNTAX

Let me know in the chat - will this work or not? (yay or nay)

```
int number;
int *number_ptr;

number_ptr = number;

*number_ptr= &number;

number_ptr= &number;

*number_ptr= number;
```

CODE CODE CODE

ARRAYS AND POINTERS AND FUNCTIONS - LET'S BRING IT ALL TOGETHER...

`shufflin.c`

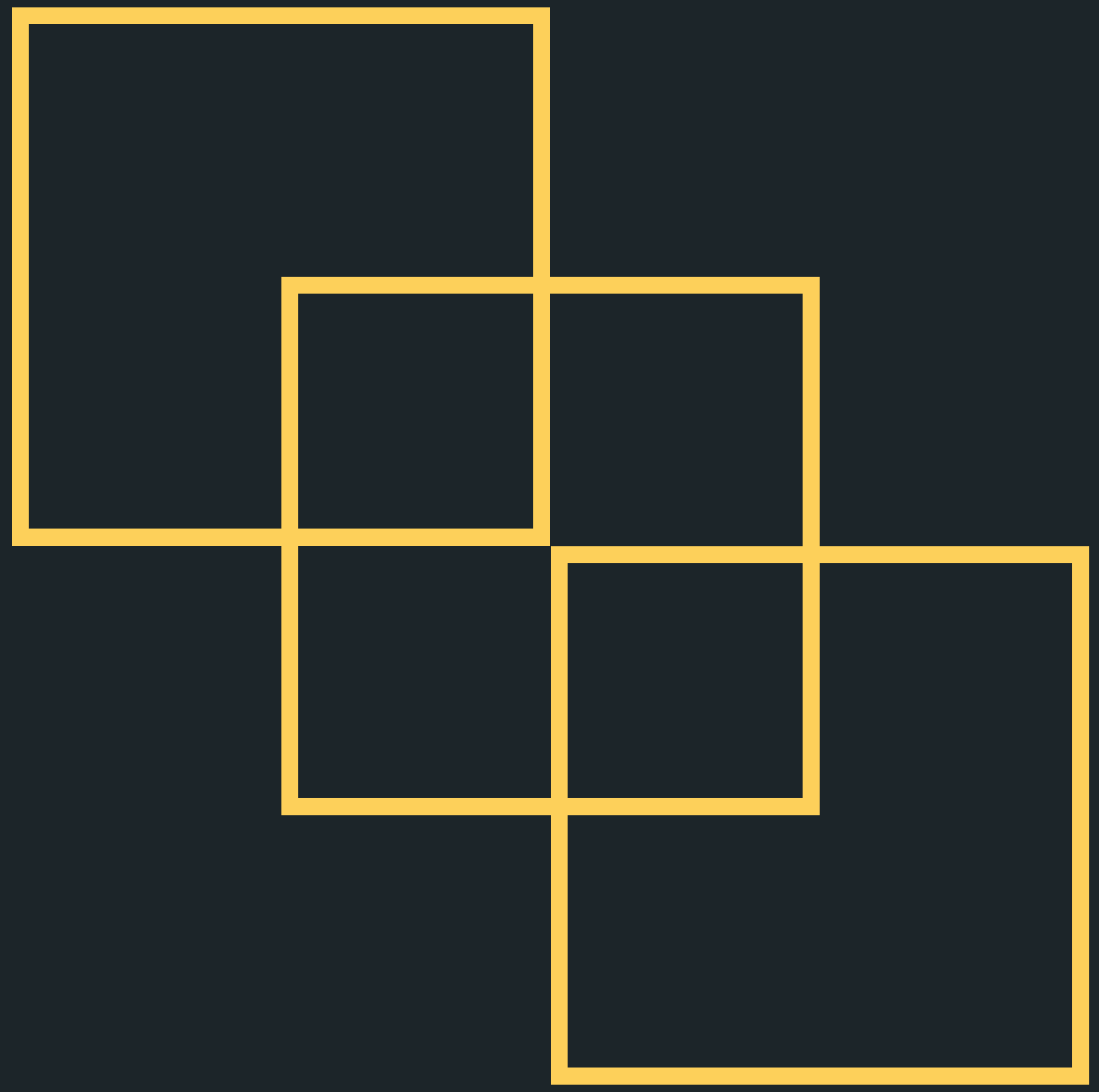
- Let’s see and use some pointers. Now remember that you can only return one thing back to main and you can’t return an array*

- The problem is this:

Read in an array of numbers (user will specify how many numbers they plan to read in). Then the first number and the last number in the array will be swapped, and the modified array printed out again.

- So without using pointers, can you have a swapping function that swaps out two things? How would you return both of those things back to the main?

BREAK TIME



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

C LIBRARIES

GOOD FOR BORROWING A LOT OF FUNCTIONS

GOOD REFERENCE IF YOU ARE INTERESTED IN LEARNING MORE ABOUT EACH LIBRARY:

HTTPS://WWW.TUTORIALSPOINT.COM/C_STANDARD_LIBRARY/INDEX.HTM

- C has a number of standard libraries available to us
- Libraries are usually .h files (header files)
- We can use these libraries whenever we want to borrow some functions by:

```
#include <library_name.h>
```

- So far we have used

  `<stdio.h>` Standard Input/Output Library
- Other useful libraries we may have seen:

  `<stdlib.h>`Standard Library

  `<math.h>` Mathematics Library
- Sometimes we can just borrow functions instead of writing them from scratch, like printf, scanf etc.

HELPFUL LIBRARY FUNCTIONS FOR CHARS

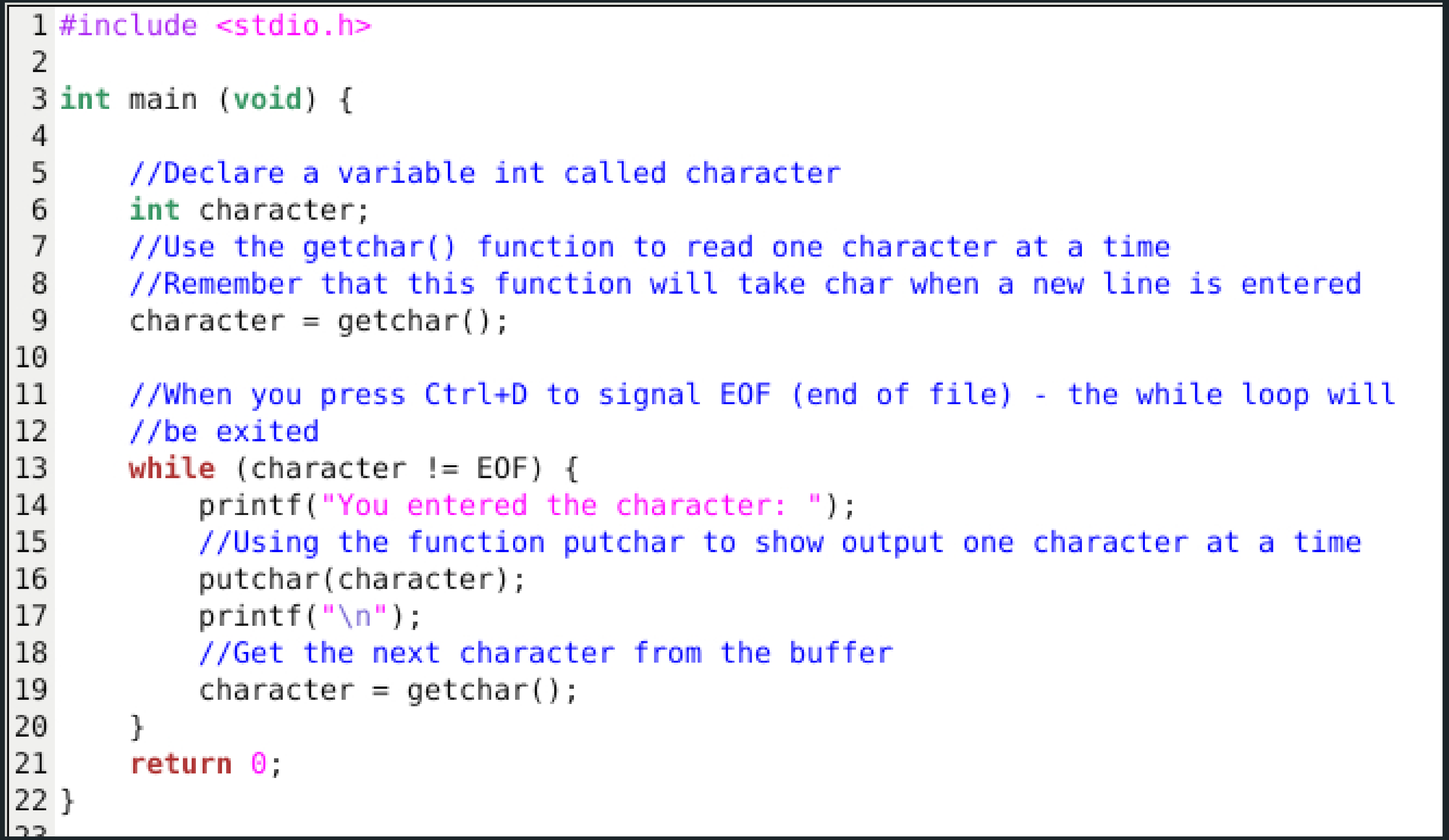
GETCHAR()

- **`getchar()`**is a function that reads a character from input (a single character)
    - Reads one byte of input
    - Usually returns an int (ASCII code of that character that it read)
    - Can return -1 (EOF), which is useful for knowing when to finish input
    - will not get its input until enter is pressed at the end of the line (it keeps filling up a buffer until enter is pressed)

HELPFUL LIBRARY FUNCTIONS FOR CHARS

PUTCHAR()

- **putchar()** is a function that prints out one character to standard output
- Similar to printf("%c", character);



```c
#include <stdio.h>

int main (void) {

    //Declare a variable int called character
    int character;
    //Use the getchar() function to read one character at a time
    //Remember that this function will take char when a new line is entered
    character = getchar();

    //When you press Ctrl+D to signal EOF (end of file) - the while loop will
    //be exited
    while (character != EOF) {
        printf("You entered the character: ");
        //Using the function putchar to show output one character at a time
        putchar(character);
        printf("\n");
        //Get the next character from the buffer
        character = getchar();
    }
    return 0;
}
```

SOME OTHER INTERESTING CHARACTER FUNCTIONS

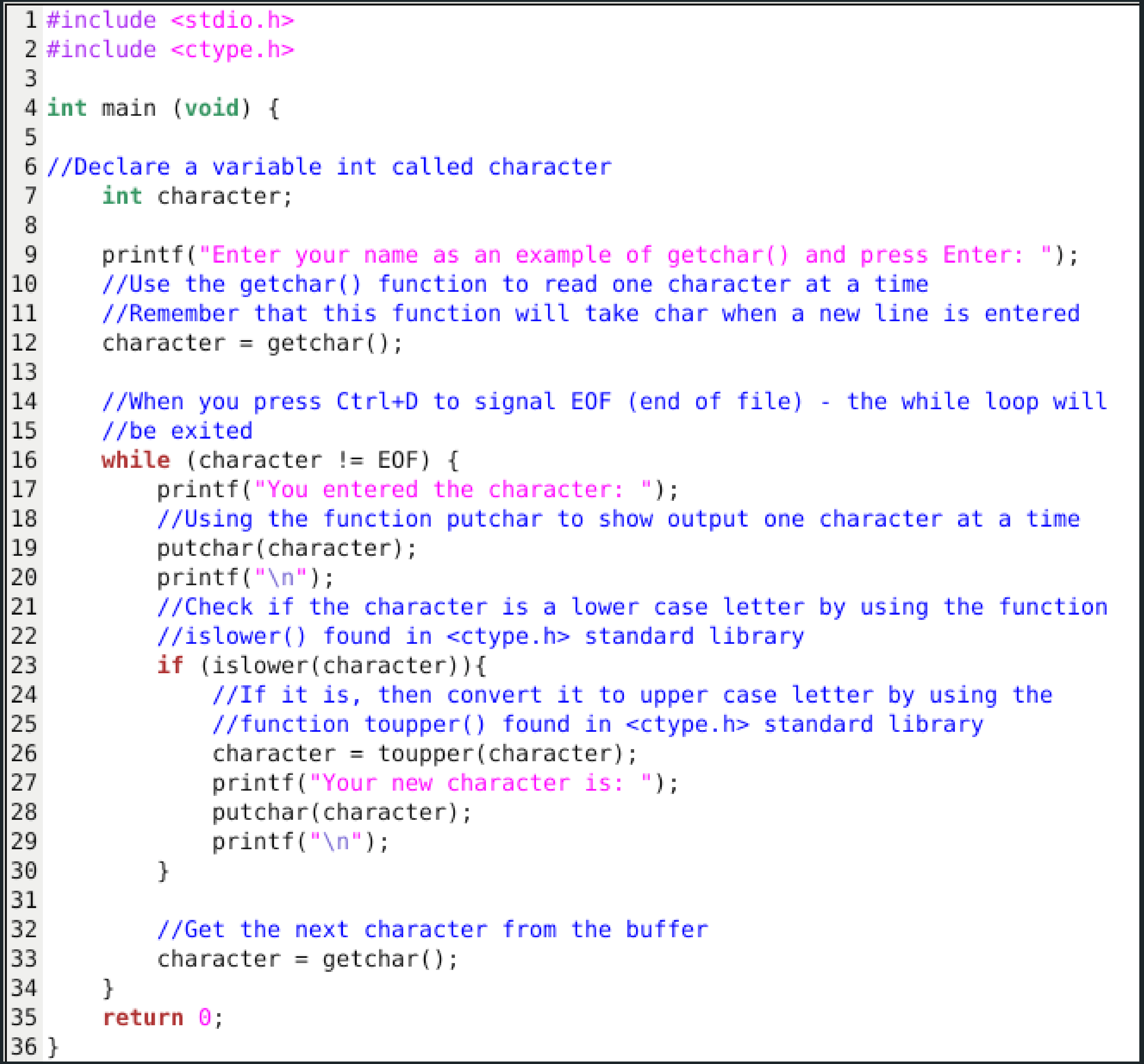
<CTYPE.H> STANDARD LIBRARY

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

Some other useful functions for characters:

- **`isalpha()`** will determine if the character is a letter
- **`isdigit()`** will determine if the character is a number
- **`islower()`** will determine if the character is a lower case letter
- **`isupper()`** will determine if the character is an upper case letter
- **`tolower()`** will convert the character to a lower case letter
- **`toupper()`** will convert the character to an upper case letter

USING SOME OF THESE FUNCTIONS



```
#include <stdio.h>
#include <ctype.h>

int main (void) {

//Declare a variable int called character
    int character;

    printf("Enter your name as an example of getchar() and press Enter: ");
    //Use the getchar() function to read one character at a time
    //Remember that this function will take char when a new line is entered
    character = getchar();

    //When you press Ctrl+D to signal EOF (end of file) - the while loop will
    //be exited
    while (character != EOF) {
        printf("You entered the character: ");
        //Using the function putchar to show output one character at a time
        putchar(character);
        printf("\n");
        //Check if the character is a lower case letter by using the function
        //islower() found in <ctype.h> standard library
        if (islower(character)){
            //If it is, then convert it to upper case letter by using the
            //function toupper() found in <ctype.h> standard library
            character = toupper(character);
            printf("Your new character is: ");
            putchar(character);
            printf("\n");
        }

        //Get the next character from the buffer
        character = getchar();
    }
    return 0;
}
```

STRINGS

WHAT ARE THEY?

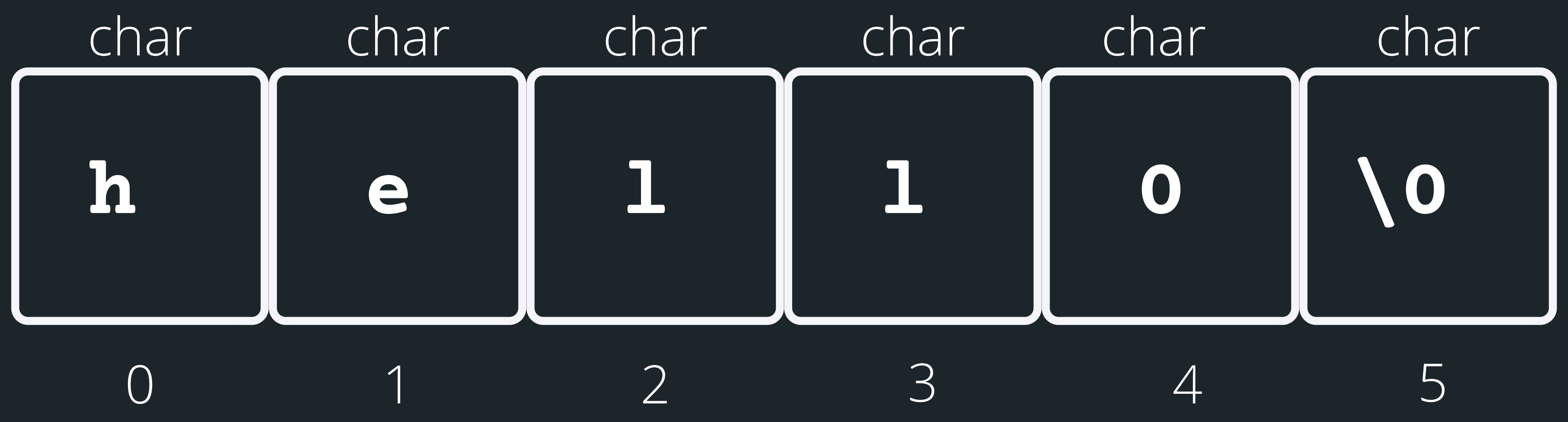
- Strings are a collection of characters that are joined together
  - an array of characters!
- There is one very special thing about strings in C - it is an array of characters that finishes with a `\0`
  - This symbol is called a null terminating character
- It is always located at the end of an array, 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:

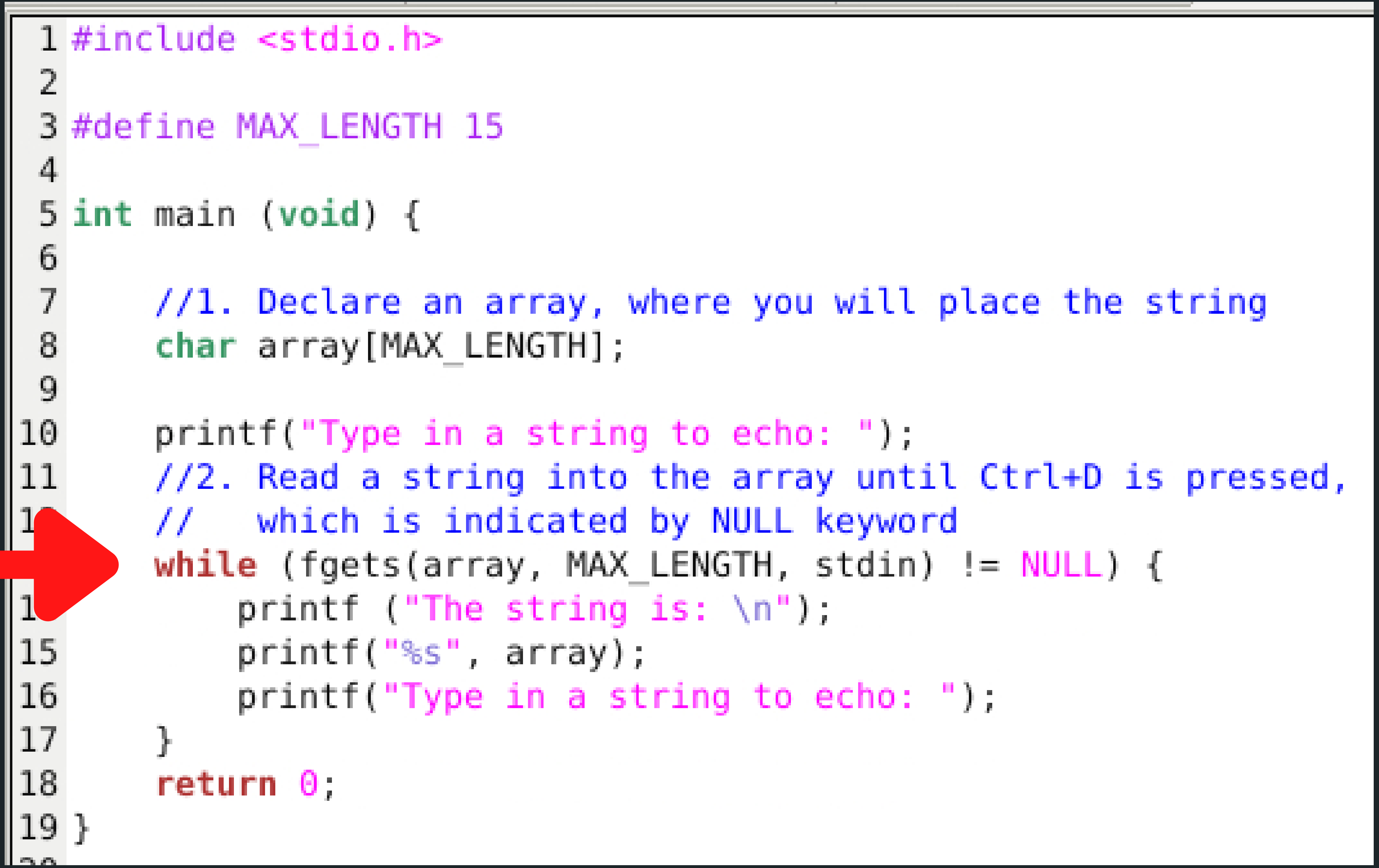
- array[] - the array that the string will be stored into
- length - the number of characters that will be read in
- stream - this is where this string is coming from - you don't have to worry about this one, in your case, it will always be stdin (the input will always be from terminal)

```
// 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) {

    //1. Declare an array, where you will place the string
    char array[MAX_LENGTH];

    printf("Type in a string to echo: ");
    //2. Read a string into the array until Ctrl+D is pressed,
    //   which is indicated by 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

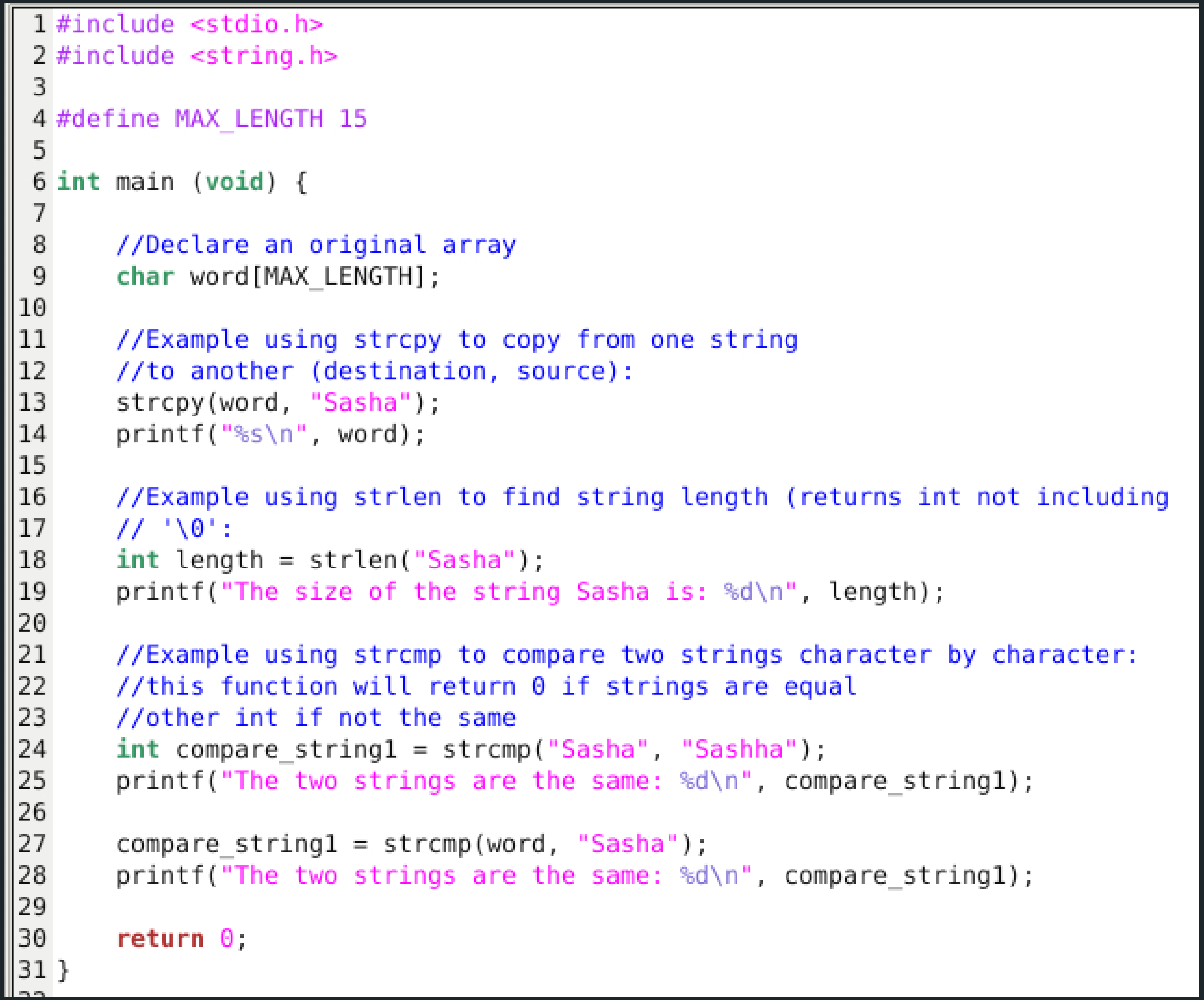
CHECK OUT THE REST OF THE FUNCTIONS: HTTPS://WWW.TUTORIALSPOINT.COM/C_STANDARD_LIBRARY/STRING_H.HTM

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 original array
    char word[MAX_LENGTH];

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

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

    //Example using strcmp to compare two strings character by character:
    //this function will return 0 if strings are equal
    //other int if not the same
    int compare_string1 = strcmp("Sasha", "Sashha");
    printf("The two strings are the same: %d\n", compare_string1);

    compare_string1 = strcmp(word, "Sasha");
    printf("The two strings are the same: %d\n", compare_string1);

    return 0;
}
```

Feedback please!

I value your feedback and use it 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.microsoft.com/r/dKssTn3AU4

WHAT DID WE LEARN TODAY?

POINTERS RECAP

pointers_basic.c

shuffling.c

STRINGS

string_functions.c

REACH OUT

CONTENT RELATED QUESTIONS

Check out the forum

ADMIN QUESTIONS

cs1511@cse.unsw.edu.au