

COMP1511 PROGRAMMING FUNDAMENTALS

# LECTURE 9

Recap Command Line Arguments  
Pointers

# LAST WEEK...

- 2D Arrays
- Strings
- Command Line Arguments (went into overdrive with excitement)

# TODAY...

- The lovely Dr Andrew Taylor will be taking you through the content:
  - Recap command line arguments
  - Start looking at pointers

“

WHERE IS THE CODE?



**Live lecture code can be found here:**

[HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/23T1/LIVE/WEEK05/](https://cgi.cse.unsw.edu.au/~cs1511/23T1/LIVE/WEEK05/)

# STRINGS RECAP

- 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?

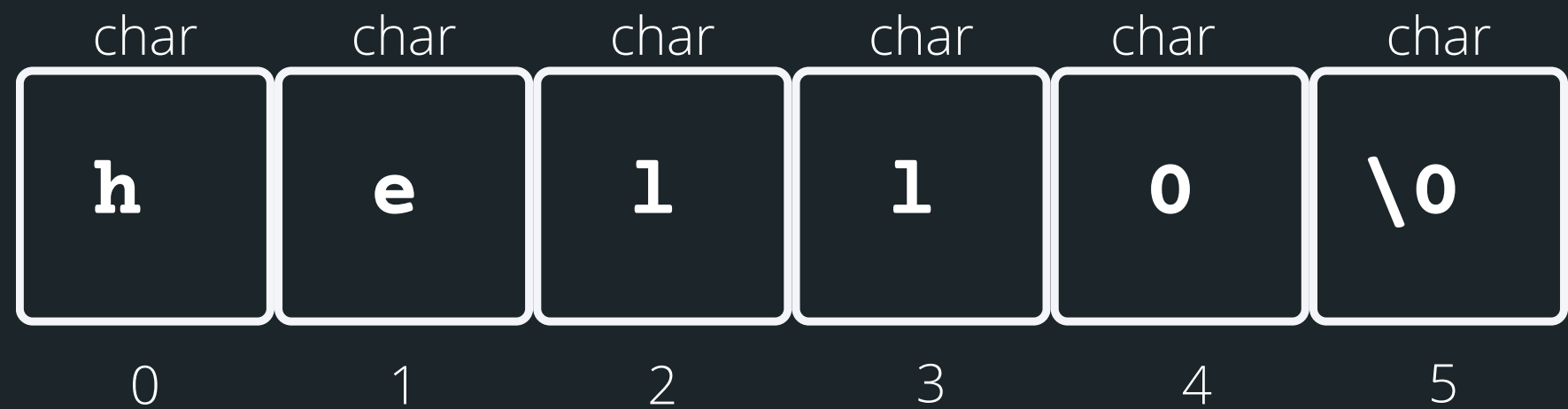
- Because strings are an array of characters, the array type is char.
- To declare and initialise a string, you can use two methods:

//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, stdin)
```

# 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

```
1 #include <stdio.h>
2
3 #define MAX_LENGTH 15
4
5 int main(void) {
6     // Declare an array where you will place the string
7     char array[MAX_LENGTH];
8
9     printf("Type in a string to echo: ");
10    // Read in the string into the array until Ctrl+D is
11    // pressed, which is indicated by the NULL keyword
12    while (fgets(array, MAX_LENGTH, stdin) != NULL) {
13        printf("The string is: \n");
14        printf("%s", array);
15        printf("Type in a string to echo: ");
16    }
17    return 0;
18 }
```





# 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:~$ gcc 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

```
1 #include <stdio.h>
2
3 int main (int argc, char *argv[]) {
4     printf("There are %d command line arguments in this program\n", argc);
5
6     //argv[0] is always the program name
7     printf("The program name is %s (argv[0])\n", argv[0]);
8
9     // What about the other command line arguments? Let's loop through
10    // the array and print them all out!
11    for (int i = 0; i < argc; i++) {
12        printf("The command line argument at index %d"
13              "argv[%d] is %s\n", i, i, argv[i]);
14    }
15
16    return 0;
17 }
```

```
avas605@vx02:~$ gcc 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

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main (int argc, char *argv[]) {
5     // Remember that the command line arguments are all strings, so if you
6     // need to do mathematical operations, you will need to convert them
7     // to numbers
8     // You can do this with a really handy function atoi() in the stdlib.h library!
9
10    // Let's print out all the command line arguments given and then add
11    // them together to give the sum of the command line arguments
12
13    int sum = 0;
14    for (int i = 1; i < argc; i++) {
15        printf("The command line argument at index %d (argv[%d]) is %d\n",
16              i, i, atoi(argv[i]));
17        sum = sum + atoi(argv[i]);
18    }
19    printf("The sum of the arguments is %d\n", sum);
20
21    return 0;
22 }
```

```
avas605@vx02:~$ gcc 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`

# BREAK TIME



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

# POINTERS

- 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?

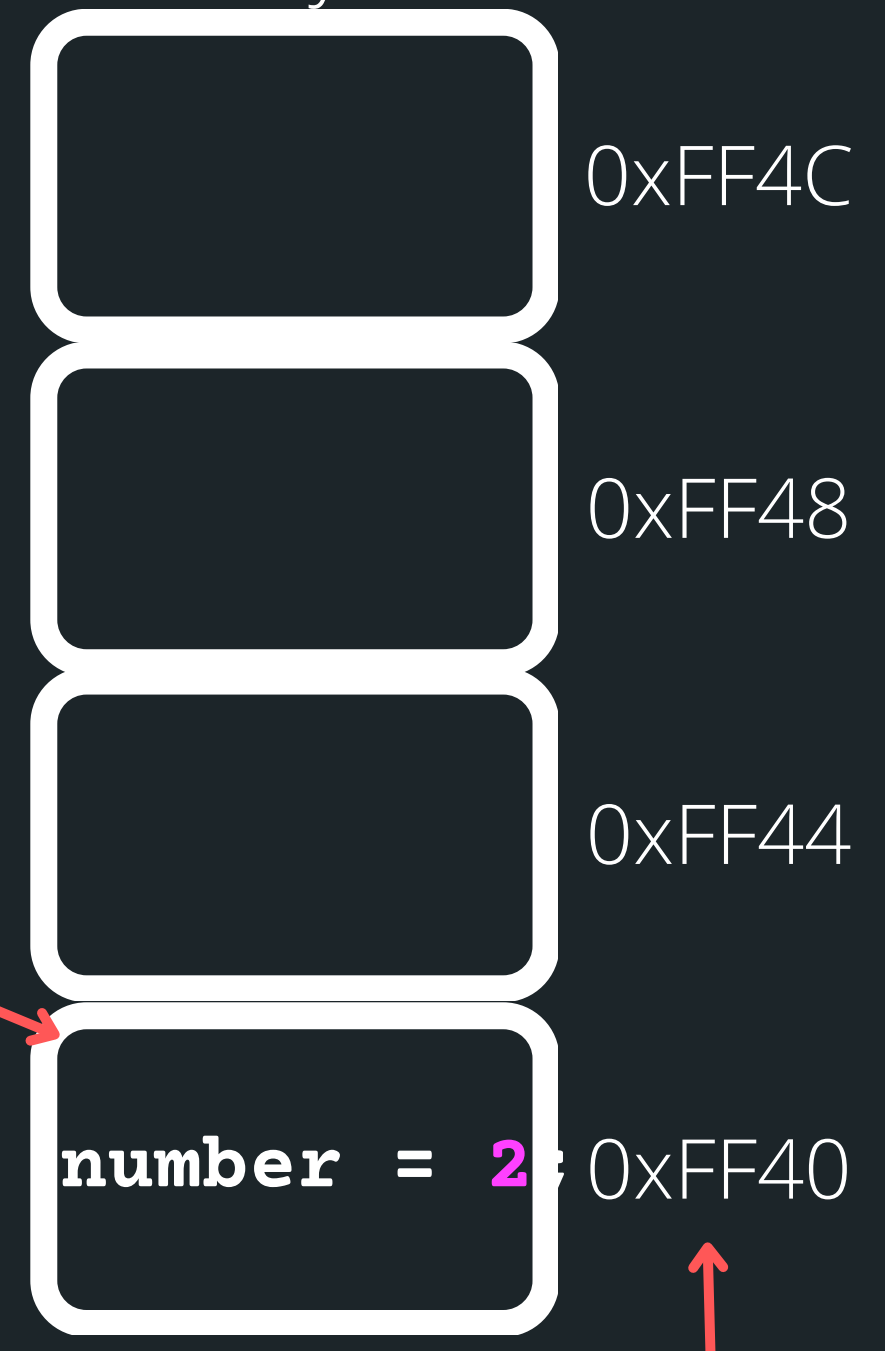
```
// Declare a variable of  
// type int. called number  
// Assign the value 13 to  
// box
```

```
int number = 2;
```

```
// Declare a pointer  
// variable that points to  
// an int and assign the  
// address of number to it
```

```
int *number_ptr = &number;
```

Memory Stack



```
// 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
```

# POINTERS

## 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**

**A SIMPLE POINTERS  
EXAMPLE**

**`pointers_simple.c`**

- A simple pointers example

# 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?



# 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/al86y6y82aex>

# WHAT DID WE LEARN TODAY?

STRINGS  
RECAP

COMMAN LINE  
ARGUMENTS  
RECAP

`compare_numbers.c`  
`compare_strings.c`

POINTERS  
`shufflin.c`



# REACH OUT



## CONTENT RELATED QUESTIONS

Check out the forum



## ADMIN QUESTIONS

[cs1511@unsw.edu.au](mailto:cs1511@unsw.edu.au)