



Lecture 10

Strings

Command Line Arguments



YESTERDAY...

- Revisited pointers
- Debugged some code – FUN!
- Looked at some special functions we can use for characters



TODAY...

- Strings (a love story of arrays and characters)
- Command line arguments

WHERE IS THE CODE?

**LIVE LECTURE CODE CAN BE
FOUND HERE:**

<https://cgi.cse.unsw.edu.au/~cs1511/21T3/live/Week05/>

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?

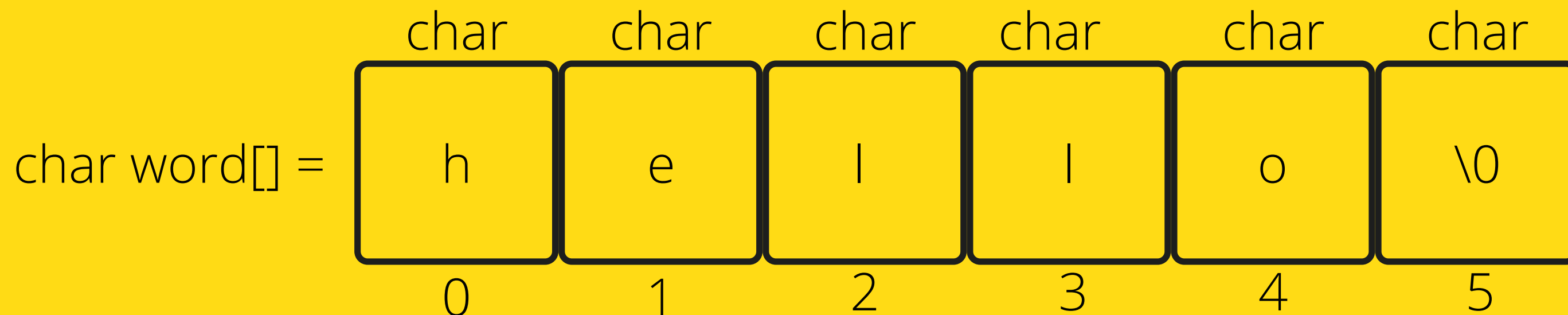
AND WHAT DOES IT LOOK LIKE VISUALLY?

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

```
char word[] = "hello" (the more  
convenient way)
```

THIS IS ALSO THE SAME AS:

```
char word[] = {'h','e','l','l','o','\0'}
```



HOW DO WE READ A STRING?

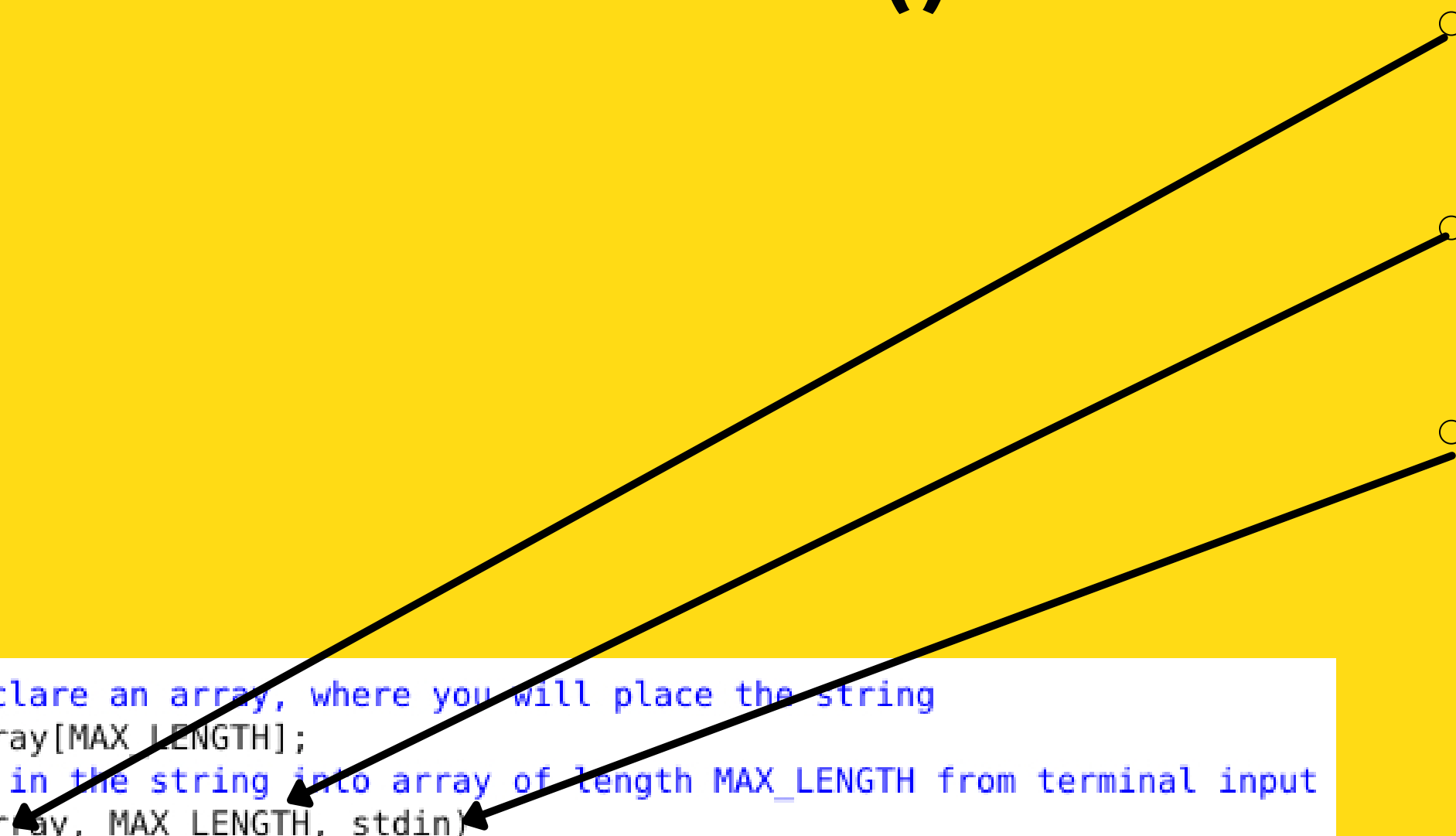
INTRODUCING FGETS()

- 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**
 - This means that the input will always be from terminal

```
//1. Declare an array, where you will place the string
char array[MAX_LENGTH];
// Read in the string into array of length MAX_LENGTH from terminal input
fgets(array, MAX_LENGTH, stdin)
```

A diagram consisting of three black arrows pointing from the code in the box to the function parameters in the list. The first arrow points from 'array' in 'fgets(array, MAX_LENGTH, stdin)' to 'array[]'. The second arrow points from 'MAX_LENGTH' to 'length'. The third arrow points from 'stdin' to 'stream'.

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

USING THE NULL KEYWORD

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

```
1 #include <stdio.h>
2
3 #define MAX_LENGTH 15
4
5 int main (void) {
6
7     //1. Declare an array, where you will place the string
8     char array[MAX_LENGTH];
9
10    printf("Type in a string to echo: ");
11    //2. Read a string into the array until Ctrl+D is pressed,
12    //    which is indicated by NULL keyword
13    while (fgets(array, MAX_LENGTH, stdin) != NULL) {
14        printf ("The string is: \n");
15        printf ("%s", array);
16        printf("Type in a string to echo: ");
17    }
18    return 0;
19 }
```




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

ANOTHER FUNCTION TO WRITE A STRING

FPUTS()

```
1 #include <stdio.h>
2
3 #define MAX_LENGTH 15
4
5 int main (void) {
6
7     //1. Declare an array, where you will place the string
8     char array[MAX_LENGTH];
9
10    printf("Type in a string to echo: ");
11    //2. Read a string into the array until Ctrl+D is pressed,
12    //    which is indicated by NULL keyword
13    while (fgets(array, MAX_LENGTH, stdin) != NULL) {
14        printf ("The string is: \n");
15        printf("%s", array);
16        //You can also use a function called fputs(array[], stream)
17        //to do the same thing as line 15 - print out the string to terminal
18        fputs(array, stdout);
19        printf("Type in a string to echo: ");
20    }
21    return 0;
22 }
```



- 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**
 - This means that the output will always be in terminal

SOME OTHER INTERESTING STRING FUNCTIONS

<STRING.H> STANDARD LIBRARY

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 occurrence of a character
-

CHECK OUT THE REST OF THE FUNCTIONS:

[HTTPS://WWW.TUTORIALSPOINT.COM/C_STANDARD_LIBRARY/
STRING_H.HTM](https://www.tutorialspoint.com/c_standard_library/string_h.htm)

SOME OF THESE STRING FUNCTIONS IN ACTION

<STRING.H> STANDARD LIBRARY

CHECK OUT THE REST OF THE FUNCTIONS:

[HTTPS://WWW.TUTORIALSPOINT.COM/C_STANDARD_LIBRARY/
STRING_H.HTM](https://www.tutorialspoint.com/c_standard_library/string_h.htm)

```
1 #include <stdio.h>
2 #include <string.h>
3
4 #define MAX_LENGTH 15
5
6 int main (void) {
7
8     //Declare an original array
9     char word[MAX_LENGTH];
10
11     //Example using strcpy to copy from one string
12     //to another (destination, source):
13     strcpy(word, "Sasha");
14     printf("%s\n", word);
15
16     //Example using strlen to find string length (returns int not including
17     // '\0':
18     int length = strlen("Sasha");
19     printf("The size of the string Sasha is: %d\n", length);
20
21     //Example using strcmp to compare two strings character by character:
22     //this function will return 0 if strings are equal
23     //other int if not the same
24     int compare_string1 = strcmp("Sasha", "Sashha");
25     printf("The two strings are the same: %d\n", compare_string1);
26
27     compare_string1 = strcmp(word, "Sasha");
28     printf("The two strings are the same: %d\n", compare_string1);
29
30     return 0;
31 }
```

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
```

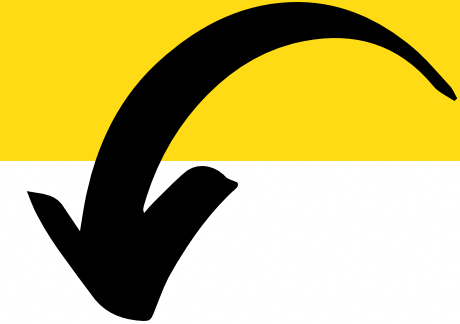
Arguments (input parameters)

TIME TO CHANGE THAT VOID

LET'S GET OUR MAIN FUNCTION TO ACCEPT SOME INPUT PARAMETERS

- `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)

- In order to change your main function to accept command line arguments on first running, you need to change the void input:



```
#include <stdio.h>

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

    return 0;

}
```

AN EXAMPLE OF CODE

**INT MAIN (INT
ARGC, CHAR
*ARGV[])**

```
#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 located in argv[0] and is %s\n", argv[0]);

    //Let's now print out all the command line arguments given
    int i = 1;
    while (i < argc) {
        printf("The command line argument at index %d (argv[%d]) is %s\n",
            i, i, argv[i]);
        i++;
    }
    return 0;
}
```

```
avas605@vx7:~$ gcc argv_demo.c -o argv_demo
avas605@vx7:~$ ./argv_demo The Week of Freedom has Arrived
There are 7 command line arguments in this program
The program name is located in argv[0] and is ./argv_demo
The command line argument at index 1 (argv[1]) is The
The command line argument at index 2 (argv[2]) is Week
The command line argument at index 3 (argv[3]) is of
The command line argument at index 4 (argv[4]) is Freedom
The command line argument at index 5 (argv[5]) is has
The command line argument at index 6 (argv[6]) is Arrived
```

WHAT IF YOU WANT NUMBERS AND NOT STRINGS?

REMEMBER THAT EACH
COMMAND LINE
ARGUMENT IS A STRING

```
avas605@vx7:~$ gcc atoi_demo.c -o atoi_demo
avas605@vx7:~$ ./atoi_demo 1 2 3 4 5
The command line argument at index 1 (argv[1]) is 1
The command line argument at index 2 (argv[2]) is 2
The command line argument at index 3 (argv[3]) is 3
The command line argument at index 4 (argv[4]) is 4
The command line argument at index 5 (argv[5]) is 5
The sum of the command line arguments is 15
```

- 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>

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main (int argc, char *argv[]) {
5
6     //Remember that the command line arguments are strings, so if you need
7     //to do mathematical operations, you will need to convert them to numbers
8     //You can do this with the help of the atoi() function in <stdlib.c>
9
10    //Let's now print out all the command line arguments given and
11    //add them together to give the sum of command line arguments
12
13    int sum = 0;
14    //Start the index at 1, because 0 is the program name
15    int i = 1;
16    while (i < argc) {
17        printf("The command line argument at index %d (argv[%d]) is %d\n",
18              i, i, atoi(argv[i]));
19        sum = sum + atoi(argv[i]);
20        i++;
21    }
22    printf("The sum of the command line arguments is %d\n", sum);
23    return 0;
24 }
```


BREAK TIME (5 MINUTES)

Jax and Juno have fallen in love (via the internet) and Jax wishes to mail her a ring. Unfortunately, they live in the country of Kleptopia where anything sent through the mail will be stolen unless it is enclosed in a padlocked box. Jax and Juno each have plenty of padlocks, but none to which the other has a key. How can Jax get the ring safely into Juno's hands?

PROBLEM TIME

PUTTING IT ALL TOGETHER: COMMAND LINE ARGUMENTS, AND CHARACTERS, AND STRINGS!?



Your task (should you choose to accept it) is to read in two strings from the command line. The first of these arguments will be either the word "numbers" or "words". This will allow you to know what needs to be compared, and the next command line argument will either be a number or a word depending on what you are comparing. Another string will then be prompted from the user once the program starts, and the two strings will be compared for similarity – if the string is a word, then the number of similar letters will be counted. Otherwise, if the string is a number, we will say whether the two numbers match.

FEEDBACK?

**PLEASE LET ME KNOW ANY
FEEDBACK FROM TODAY'S
LECTURE!**

www.menti.com

Code: 4630 6399



WHAT DID WE LEARN TODAY?



STRINGS

string_demo.c

COMMAND LINE ARGUMENTS

argv_demo.c
atoi_demo.c

PUTTING IT ALL TOGETHER

compare.c

ANY QUESTIONS?

**DON'T FORGET YOU CAN
ALWAYS EMAIL US ON
CS1511@CSE.UNSW.EDU.AU
FOR ANY ADMIN QUESTIONS**

**PLEASE ASK IN THE FORUM
FOR CONTENT RELATED
QUESTIONS**

