COMP1511 PROGRAMMING FUNDAMENTALS

LECTURE 6

I ❤ ARRAYS

LAST LECTURE...

ON TUESDAY...

- Talked about good style/bad style
- Functions - what/how/why?

THIS LECTURE...

TODAY...

- Looking back at some functions
- Starting to look at arrays

WHERE IS THE CODE?

Live lecture code can be found here:

HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/22T3/LIVE/WEEK03/

FUNCTIONS RECAP

WHAT?

- A function is a block of statements that performs a specific task

FUNCTIONS RECAP

WHY?

- Improve readability of the code
- Improve reusability of the code
- Debugging is easier (you can narrow down which function is causing issues)
- Reduces size of code (you can reuse the functions as needed, wherever needed)

FUNCTIONS RECAP

HOW?

- Predefined standard library functions (built-in)
  - printf(), scanf() inside stdio.h
- User defined function with syntax:

```
return_type function_name (arguments (type name)) {
    BLOCK OF CODE (Set of instructions for the
    function)
}
```

- return_type - can be any data type such as int, double, char, etc (CAN'T BE ARRAY)
- function_name - whatever your heart desires, should be descriptive
- arguments - what are the inputs into the function
- Block of Code - set of instructions exercuted when call is made to the function

RECAP FUNCTIONS

return type: What type does this function return?

name of function: What will I name my function?

input/ arguments: What am I giving my function?

A function, which adds two numbers together and returns the result

```
int add (int number_one, int number_two) {
    int sum;
    sum = number_one + number_two;
    return sum;
}
```

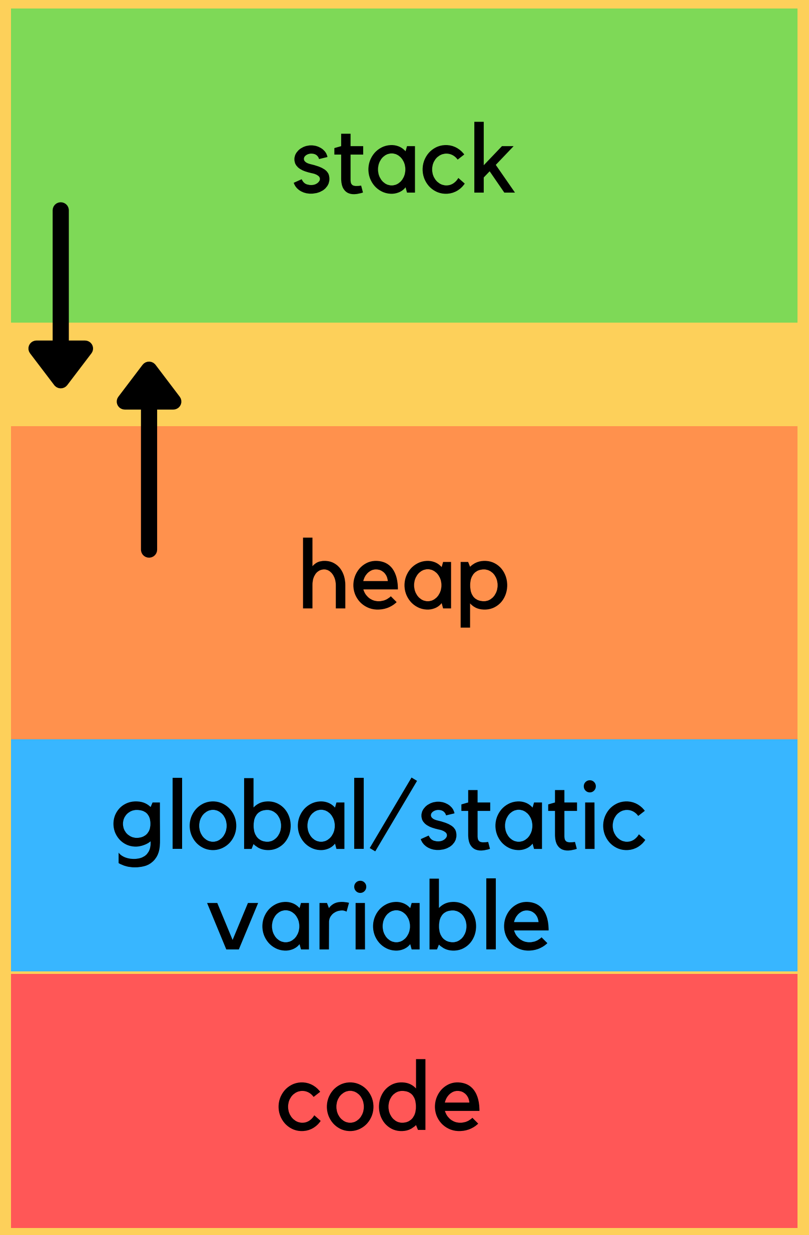
To finish I return an int (sum), which is what I said I would return when I wrote my function

RECAP FUNCTIONS

PROTOTYPE

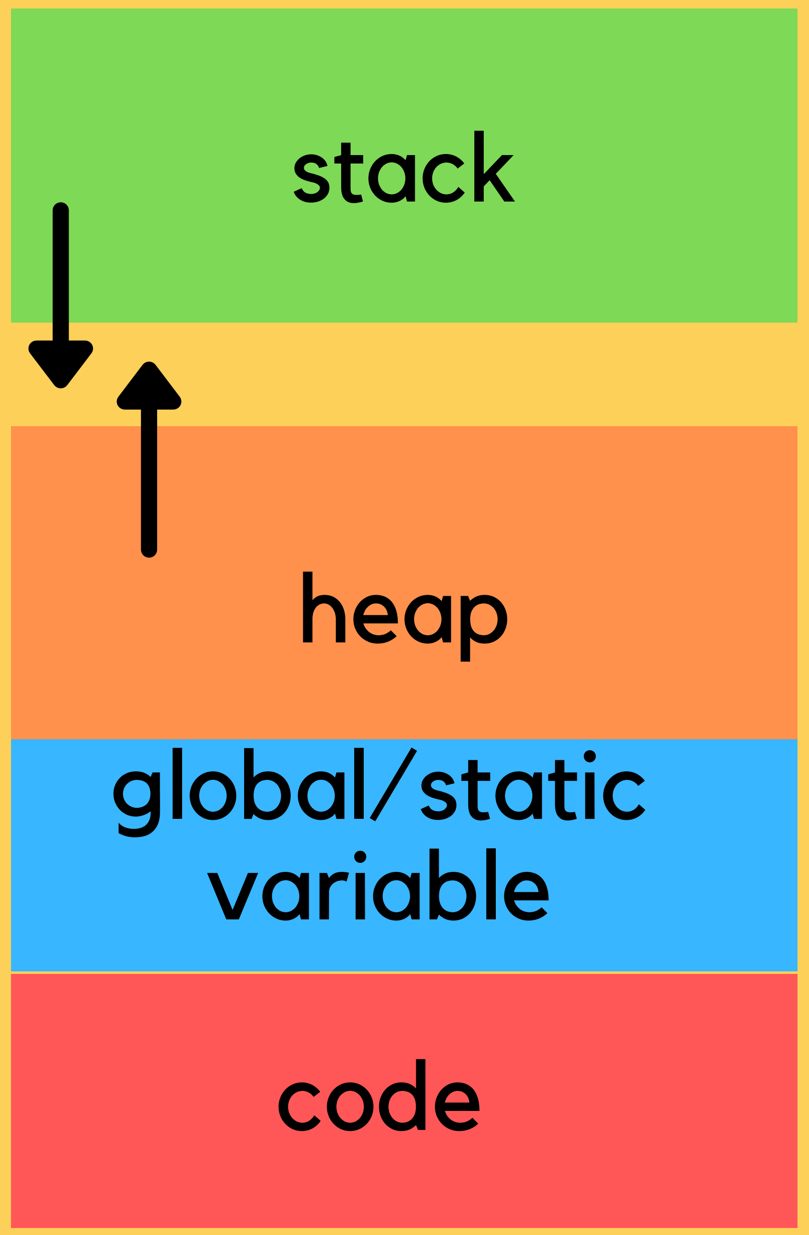
- You must have a prototype above your main to let everyone know the function is defined and is coming!

REVISITING MEMORY



- Our C file is stored on the hard drive
- Our Compiler compiles the code into another file that the computer can read
- When we execute code, the CPU will actually process the instructions and perform basic arithmetic, but the RAM will keep track of all the data needed in those instructions and operations, such as our variables.
- Reading and writing to variables will change the numbers in RAM
- Memory is divided into the stack and the heap
- The stack is an ordered stack and the heap is a random free for all - insert something where you can find space for it.

REVISITING MEMORY



- Stack memory is where relevant information about your program goes:
  - which functions are called,
  - what variables you created,
- Once your block of code finishes running {}, the function calls and variables will be removed from the stack (it's alive!)
- It means at compile time we can allocate stack memory space (not at run time)
- The stack is controlled by the program NOT BY THE developer
- The heap is controlled by the developer (more on this in a few weeks) and can be changed at run time

MEMORY IS IMPORTANT

WITHOUT MEMORY, WE CAN'T REALLY RUN ANYTHING

- Think of your own memory and what it allows you to do.
- Computer memory is important to consider when you are writing your code (we don’t focus on this in 1511, but you will in later courses)
- The more you waste memory, the slower your program gets… you will learn all about this in later computing courses! In 1511 we don’t mind the wastage :)

HOW DO WE EFFICIENTLY SOLVE PROBLEMS?

DIFFERENT PROBLEMS HAVE DIFFERENT OPTIMUM SOLUTIONS

- In this course we will learn about two pretty cool data structures:
  - Arrays (NOW!)
  - Linked Lists (after flexibility week)
- There are of course other data structures that you will learn about in further computing courses
- Choosing the right structure to house our data depends on what the problem is and what you are trying to achieve. Some structures lend themselves better to certain types of problems.

SO WITHOUT FURTHER ADO

THE ARRAY

- A PRETTY IMPORTANT DATA TYPE!
- A collection of variables all of the same type
  - Think about how this is very different to a struct
- We want to be able to deal with this collection as a whole entity, where we can:
  - Access any variable in this collection easily
  - Change any variable in this collection easily

SO WHAT KINDS OF PROBLEMS DO ARRAYS SOLVE?

- Let’s say I want to record the daily ice cream consumption for a week
- What about the daily temperatures for a year?
- The amount of time daily that I spend walking my dogs?

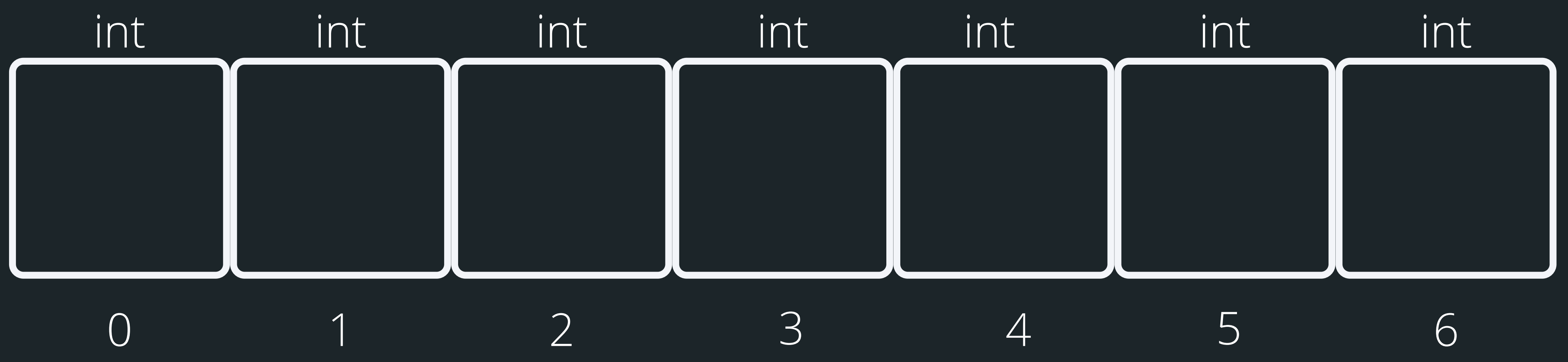
Can you think of other examples?

NOTICE THAT EACH OF THESE COLLECTIONS HAS THE SAME TYPE OF VARIABLE I AM RECORDING

ARRAY (VISUALLY)

NOTE: ALL ELEMENTS OF AN ARRAY MUST BE OF THE SAME DATA TYPE (HOMOGENOUS)

- If we group our data type as a collection, for example a collection of integers:
- We can access them as a group(collection)
- We can loop through and access each individual element of that collection



this array holds 7 integers

You can access elements of an array by referring to their index

WHY DO WE NEED AN ARRAY?

LET'S LOOK AT AN EXAMPLE PROBLEM

- Let’s say I am tracking my ice cream consumption over a week (without arrays)

```
int mon = 2;
int tues = 3;
int wedn = 3;
int thur = 5;
int fri = 7;
int sat = 1;
int sun = 3;
// Any day with 3 or more scoops is too
much!
if (mon >= 3){
    printf("Too much ice cream\n");
}
if (tue >= 3) {.......
```

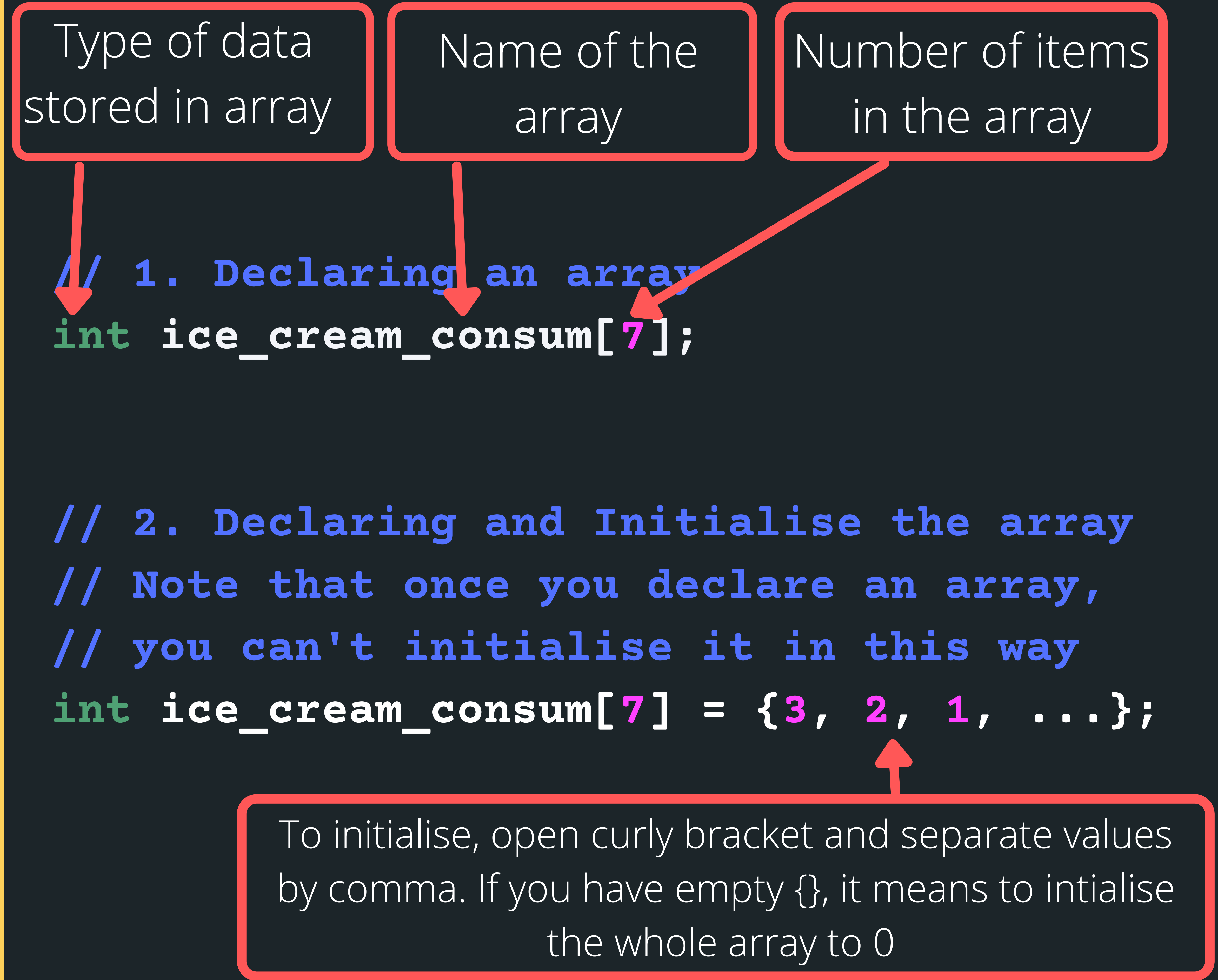
WHY DO WE NEED AN ARRAY?

LET'S LOOK AT AN EXAMPLE PROBLEM

- What if I am tracking this over the month or over a year?
  - Will I need 30 variables/365 variables?

THIS IS A GREAT PLACE TO USE AN ARRAY...

HOW DO WE DECLARE AN ARRAY



Type of data stored in array

Name of the array

Number of items in the array

```c
// 1. Declaring an array
int ice_cream_consum[7];


// 2. Declaring and Initialise the array
// Note that once you declare an array,
// you can't initialise it in this way
int ice_cream_consum[7] = {3, 2, 1, ...};
```

To initialise, open curly bracket and separate values by comma. If you have empty {}, it means to intialise the whole array to 0

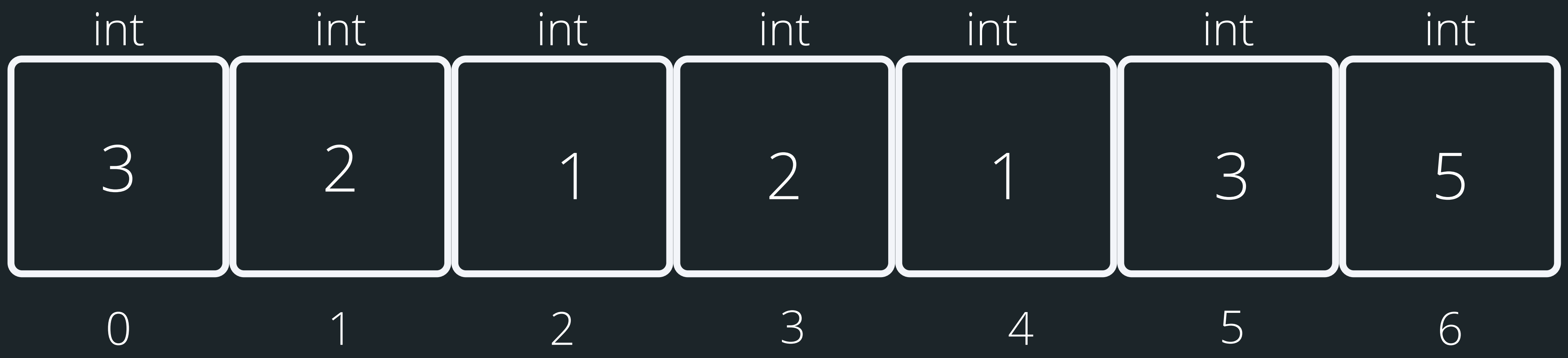
ARRAY (VISUALLY)

DECLARING AND INITIALISING AN ARRAY

- So let's say we have this declared and initialised:

```
int ice_cream_consum[7] = {3, 2, 1, 2, 1, 3, 5};
```

- This is what it looks like visually:



this array holds 7 integers

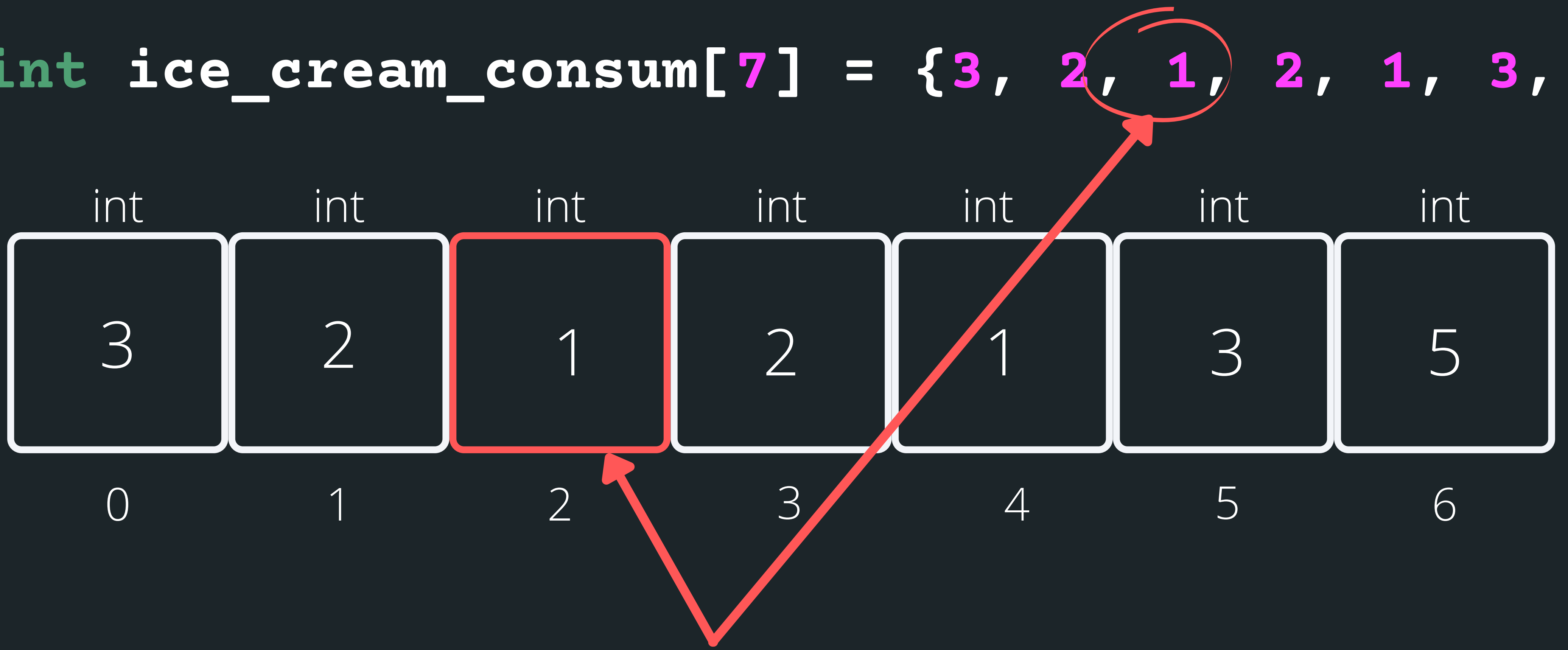
Note that indexing starts at 0

ARRAY (VISUALLY)

ACCESSING ARRAY ELEMENTS

- You can access any element of the array by referencing its index
- Note, that indexes start from 0
- Trying to access an index that does not exist, will result in an error

```
int ice_cream_consum[7] = {3, 2, 1, 2, 1, 3, 5};
```



If I wanted the third element of the array

The index would be 2, so to access it:

ice_cream_consum[2]

USING ARRAYS

CLOSER LOOK

- You can’t printf() a whole array, but you can print individual elements (consider how you could go through the array to print out every element…)
- You can’t scanf() a whole array, i.e. a line of user input test into an array, but you can can scanf() individual elements (think how to do every element in an array…)

USING ARRAYS

CLOSER LOOK

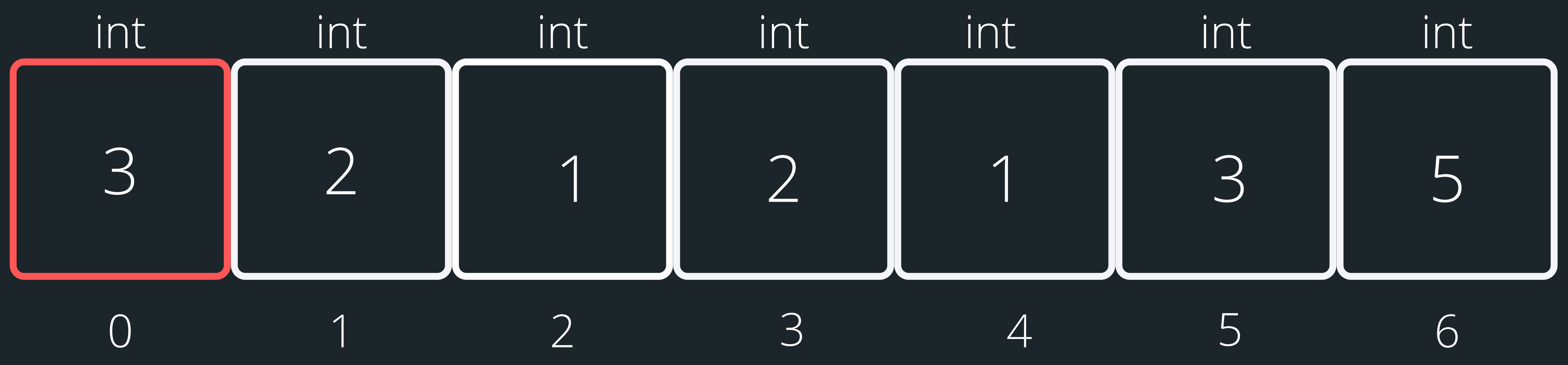
```c
int ice_cream_consum[7] = {3, 2, 1, 2, 1, 3, 5};

int i = 0;
while (i < 7){
    printf("%d ", ice cream_consum[i]);
    i++;
}
```

Start at index 0 (first entry into while loop)

ice_cream_consum[0]

print what is inside index 0



USING ARRAYS

CLOSER LOOK

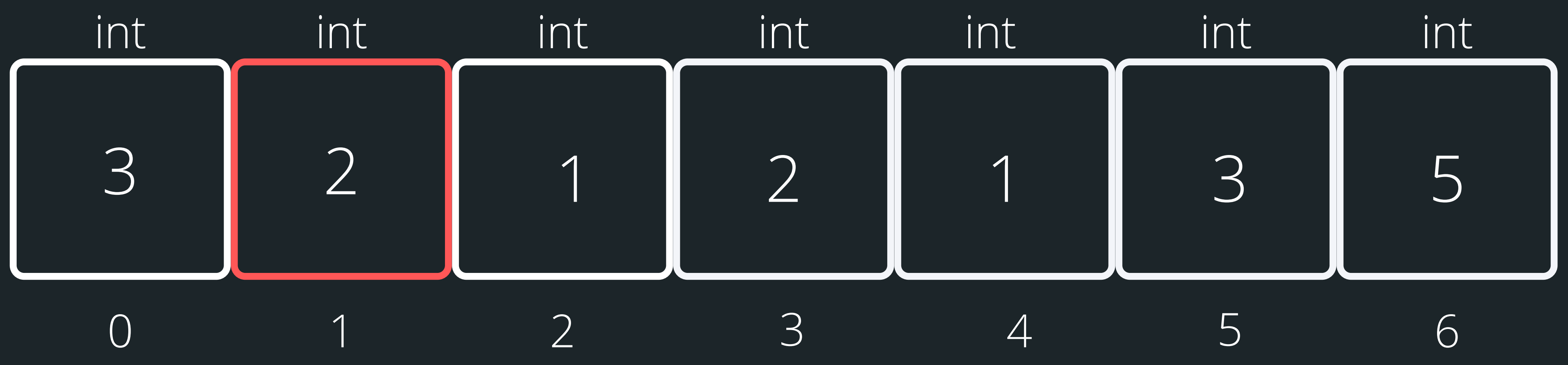
```
int ice_cream_consum[7] = {3, 2, 1, 2, 1, 3, 5};

int i = 0;
while (i < 7){
    printf("%d ", ice cream_consum[i]);
    i++;
}
```

increase index by 1

ice_cream_consum[1]

print what is inside index 1



USING ARRAYS

CLOSER LOOK

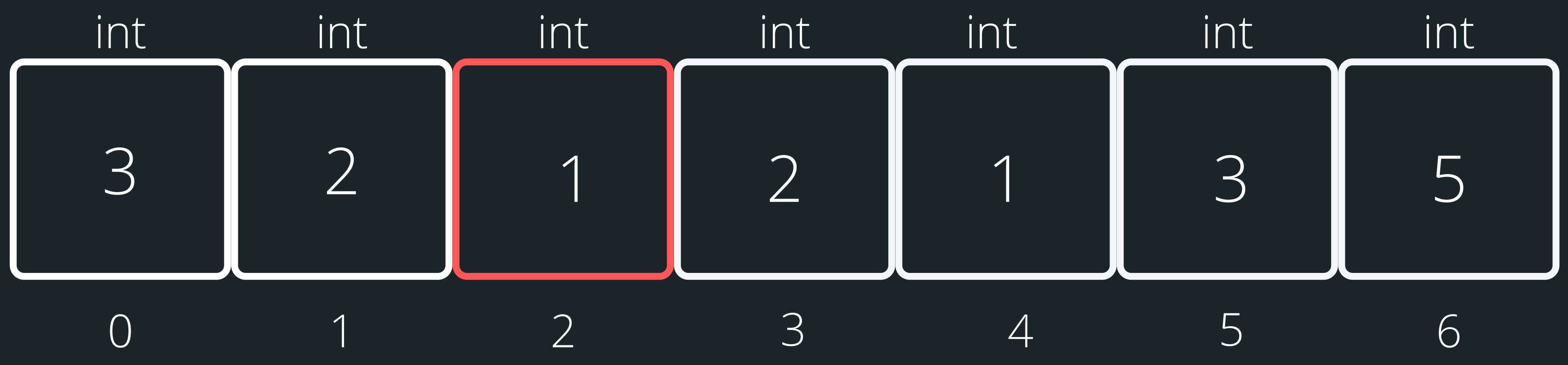
```
int ice_cream_consum[7] = {3, 2, 1, 2, 1, 3, 5};

int i = 0;
while (i < 7){
    printf("%d ", ice cream_consum[i]);
    i++;
}
```

increase index by 1

ice_cream_consum[2]

print what is inside index 2



USING ARRAYS

CLOSER LOOK

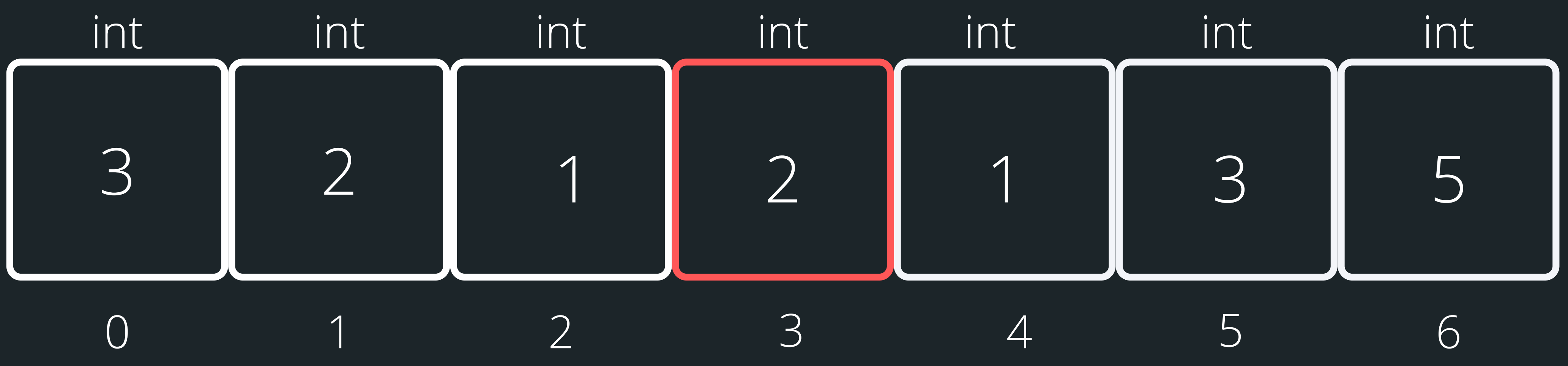
```
int ice_cream_consum[7] = {3, 2, 1, 2, 1, 3, 5};

int i = 0;
while (i < 7){
    printf("%d ", ice cream_consum[i]);
    i++;
}
```

increase index by 1

ice_cream_consum[3]

print what is inside index 3



USING ARRAYS

CLOSER LOOK

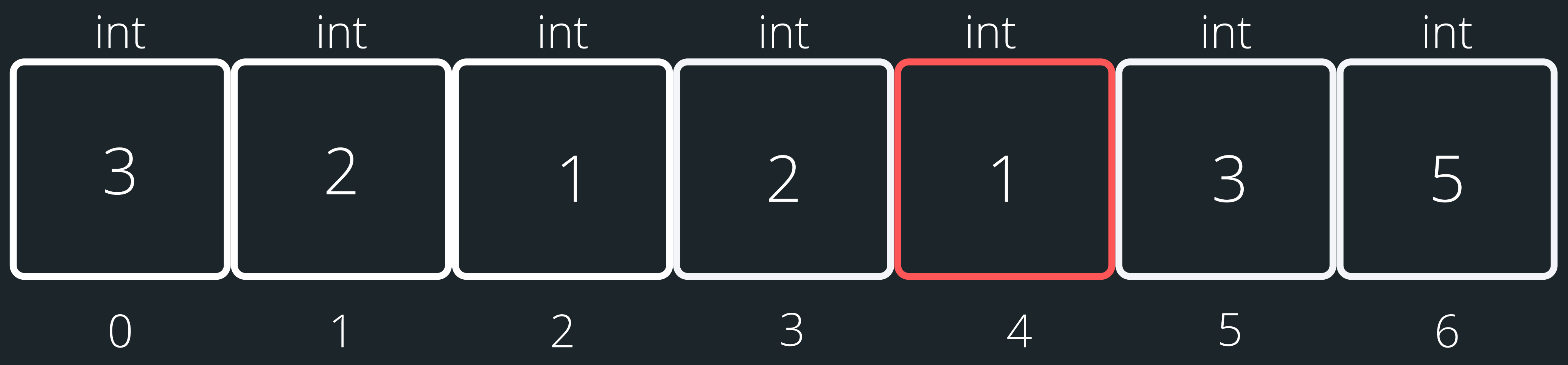
```
int ice_cream_consum[7] = {3, 2, 1, 2, 1, 3, 5};

int i = 0;
while (i < 7){
    printf("%d ", ice cream_consum[i]);
    i++;
}
```

increase index by 1

ice_cream_consum[4]

print what is inside index 4



USING ARRAYS

CLOSER LOOK

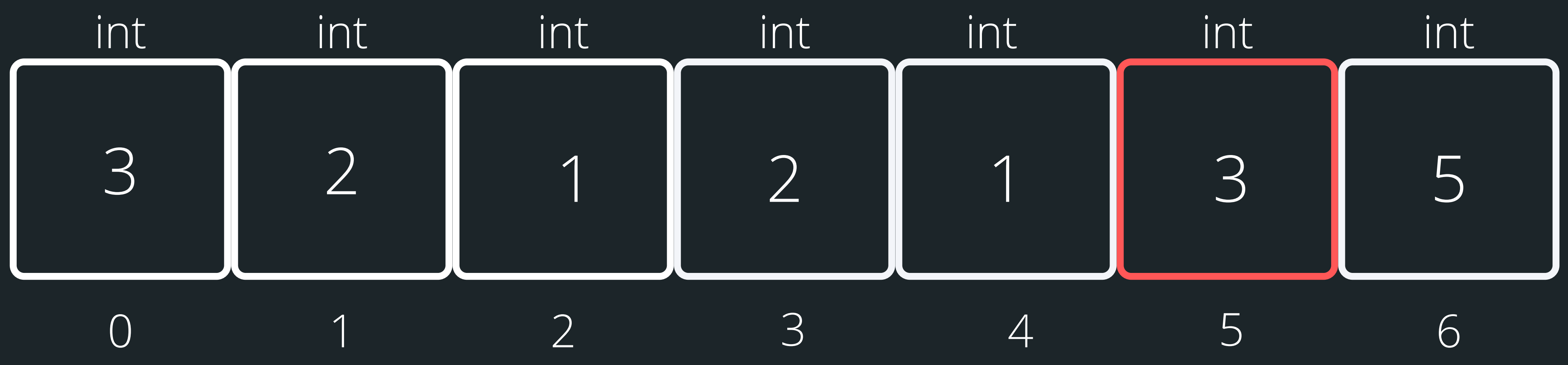
```c
int ice_cream_consum[7] = {3, 2, 1, 2, 1, 3, 5};

int i = 0;
while (i < 7){
    printf("%d ", ice cream_consum[i]);
    i++;
}
```

increase index by 1

ice_cream_consum[5]

print what is inside index 5



USING ARRAYS

CLOSER LOOK

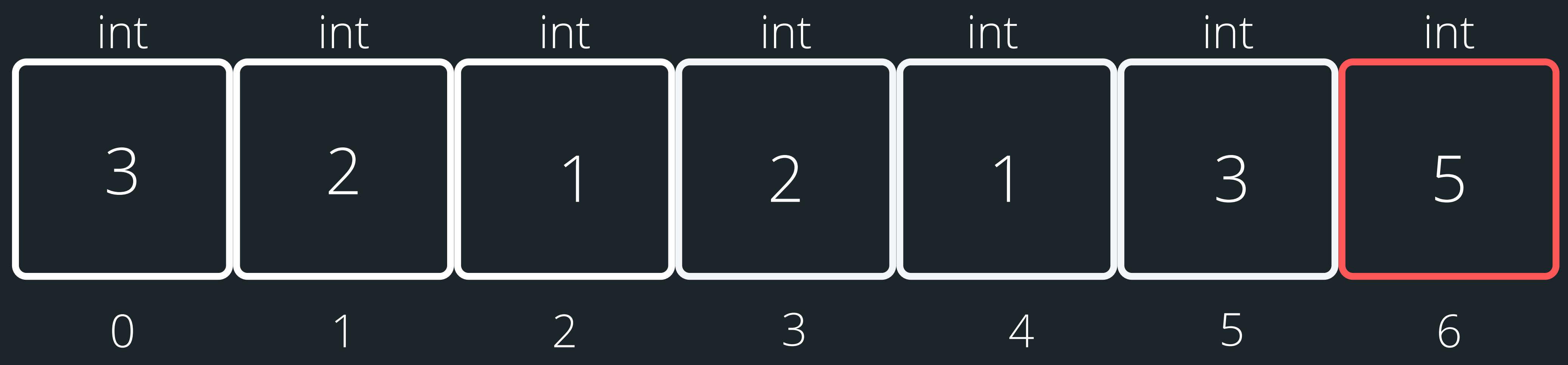
```c
int ice_cream_consum[7] = {3, 2, 1, 2, 1, 3, 5};

int i = 0;
while (i < 7){
    printf("%d ", ice cream_consum[i]);
    i++;
}
```

increase index by 1

ice_cream_consum[6]

print what is inside index 6



PROBLEM SOLVING TIME

HOORAY!

- I meet my friend for ice cream every day for a week (I don't drink coffee)… We want to be able to track how many ice creams in total we all consumed in a week, and also who ate the most ice cream in that week!

`ice_cream.c`

BREAK TIME

TIME TO STRETCH

You have two eggs in a 100-story building. You want to find out what floor the egg will break on, using the least number of drops.

STRINGS

WHAT ARE THEY?

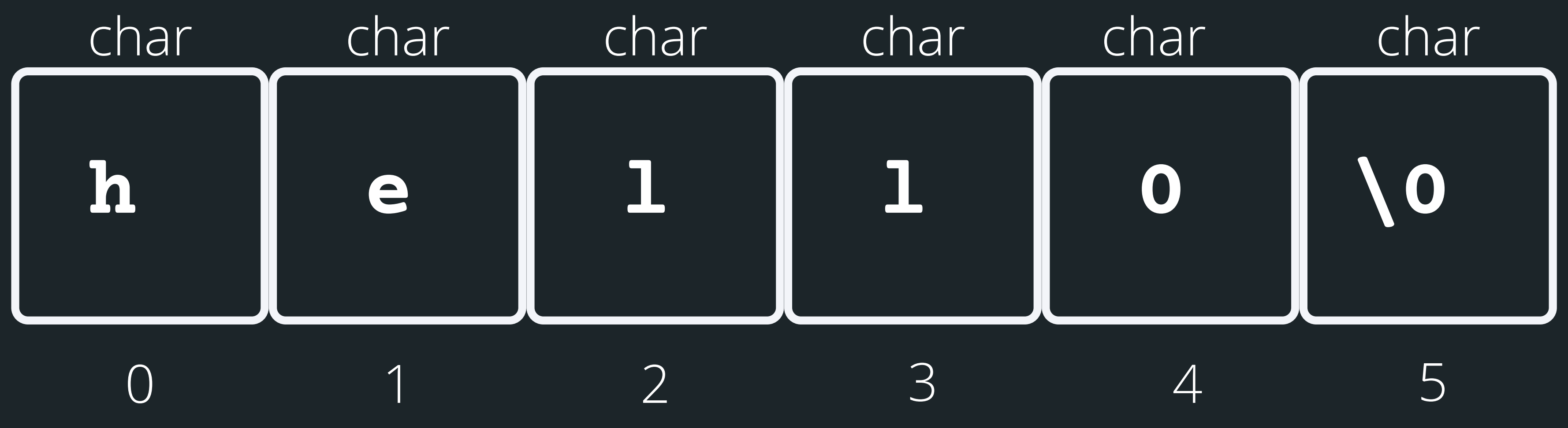
- Strings are a collection of characters that are joined together
  - an array of characters!
- There is one very special thing about strings in C - it is an array of characters that finishes with a
  - This symbol is called a null terminating character
- It is always located at the end of an array,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

SOME OTHER INTERESTING STRING FUNCTIONS

<STRING.H> STANDARD LIBRARY

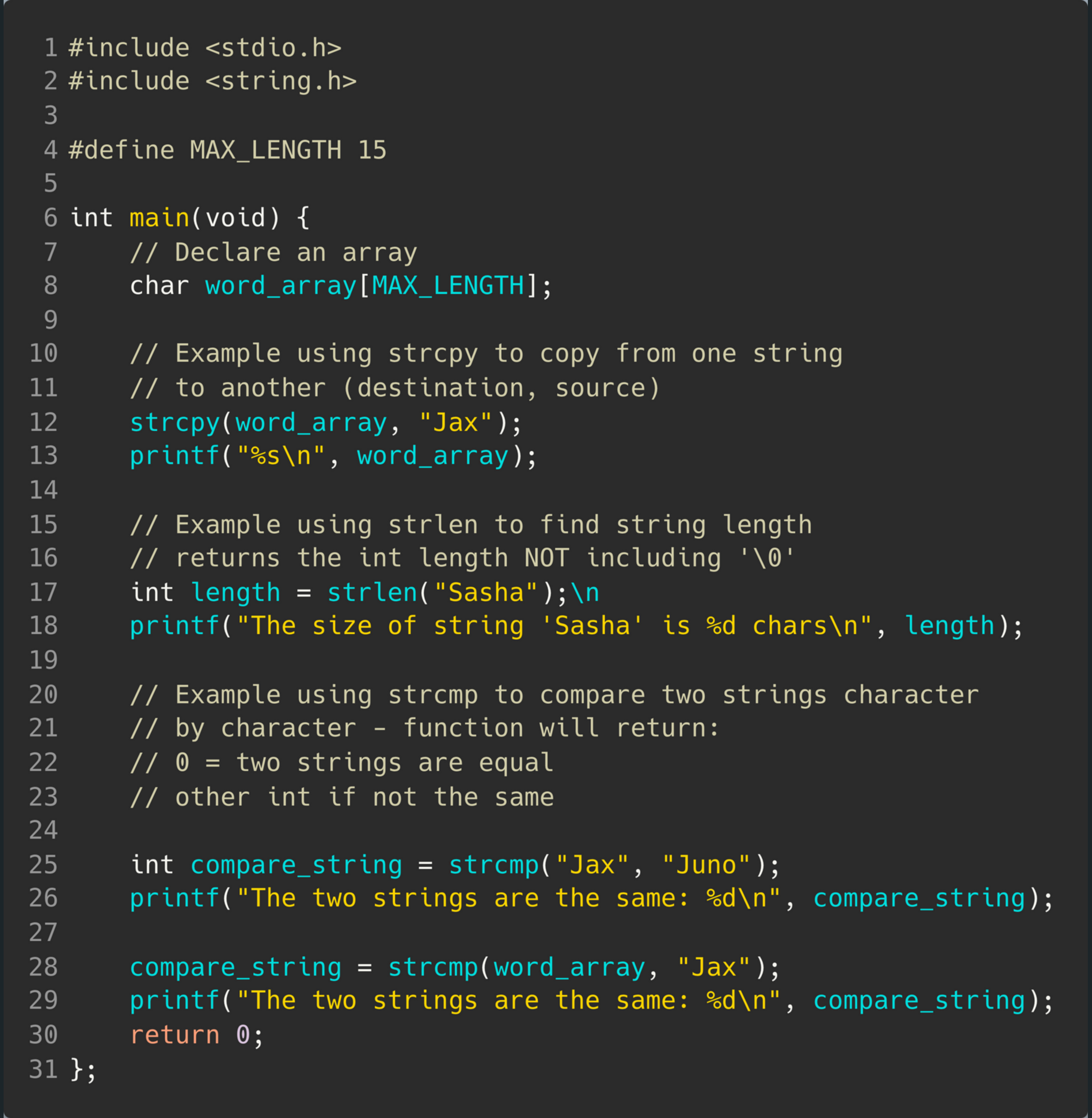
CHECK OUT THE REST OF THE FUNCTIONS: HTTPS://WWW.TUTORIALSPOINT.COM/C_STANDARD_LIBRARY/STRING_H.HTM

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;
};
```

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/albc87vpdxy2

WHAT DID WE LEARN TODAY?

FUNCTIONS RECAP

functions_recap.c

EXPLORING ARRAYS

numbers.c

ice_cream.c

STRINGS (ARRAYS OF CHAR)

strings.c

REACH OUT

CONTENT RELATED QUESTIONS

Check out the forum

ADMIN QUESTIONS

cs1511@cse.unsw.edu.au