COMP1511/1911 Programming Fundamentals

Week 3
Lecture 2

Do You Need Help?

- Help sessions exist!
  - https://discourse01.cse.unsw.edu.au/26T2/COMP1511/t/help-sessions-starting-next-week/42
- Just drop in via timetable!
  - No booking required!!

Do You Need Help?

- Revision sessions coming soon!
  - https://www.tickettailor.com/events/comp1511unsw/2265522
- Running in
  - Week 4
  - Week 6
  - Week 8
  - Week 11
- Great if you've fallen behind, or just want to consolidate what you've learnt so far!
  - A bit more structured than help sessions
  - Goes through fun new revision content!

Previously On...

- Functions
  - Breaking our problem into smaller chunks
  - Repeating code without repeating code
- Style
  - Looking cool while in school
- For loops
  - Video coming by end of week

Today

- Starting to look at arrays
- Functions with arrays

Where is the Code?

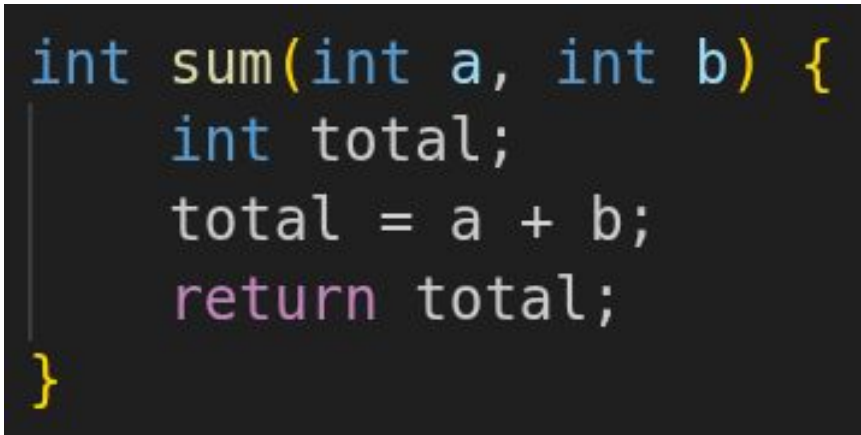
- Lecture code is updated every hour on the hour
- Live lecture code can be found here
  - https://cgi.cse.unsw.edu.au/~cs1511/26T2/

Functions Recap

- A function is a block of statements that performs a specific task
  - Improves readability of code
  - Improves reusability of code
  - Debugging is easier
    - Can narrow bugs down to specific functions
  - Reduces size of code
    - We can reuse code whenever we want!

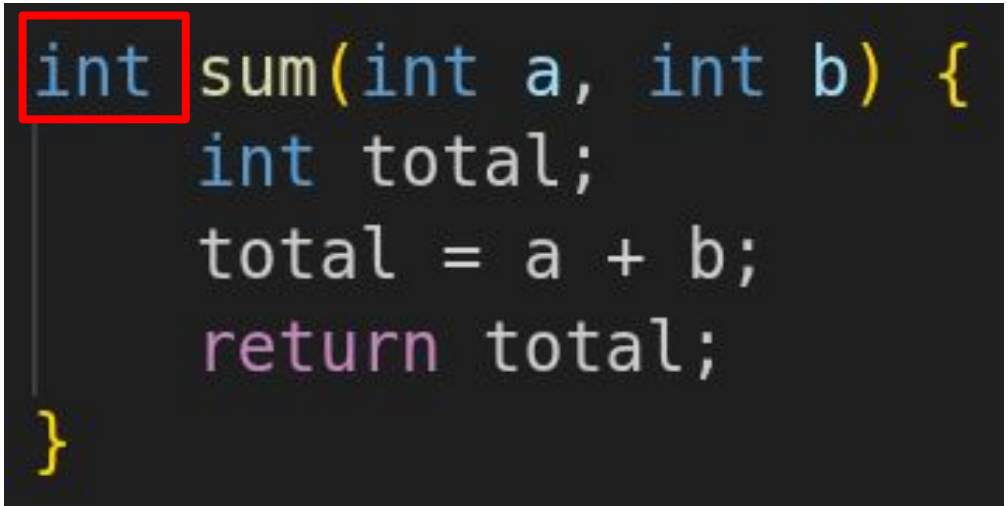
Function Recaps

- Two kinds of functions
  - Pre-defined standard library functions
    - built-in functions
    - printf and scanf inside stdio.h
  - User defined with syntax
    - Return type
    - Function name
    - Arguments
    - Block of code



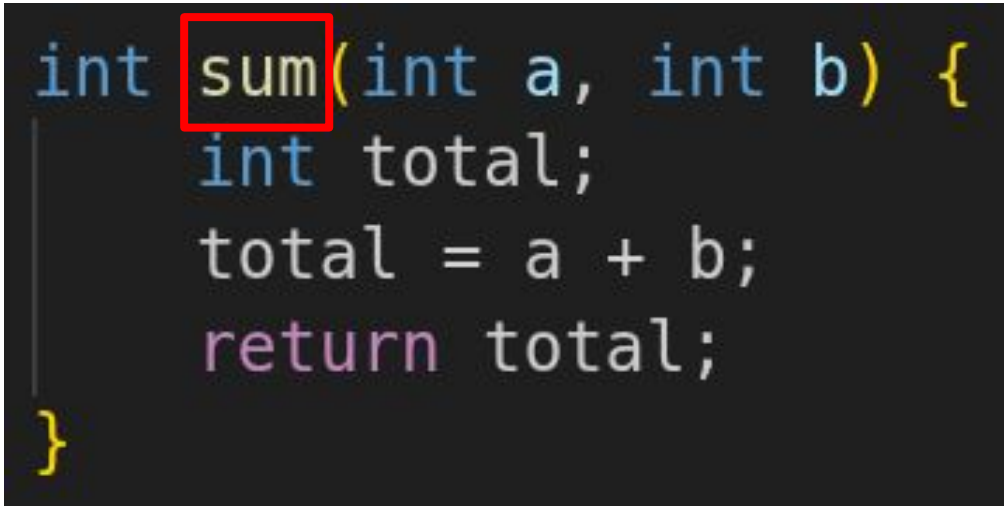
Recap Functions

- Return type
  - Tells us what type the function will return



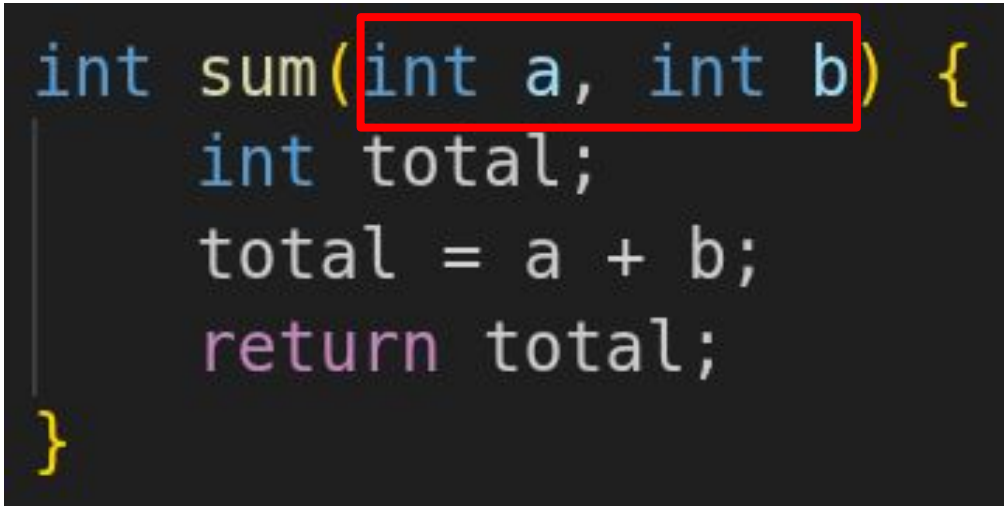
Recap Functions

- Function name
  - What we type when we want to call our function
    - int value = sum(12, 13);



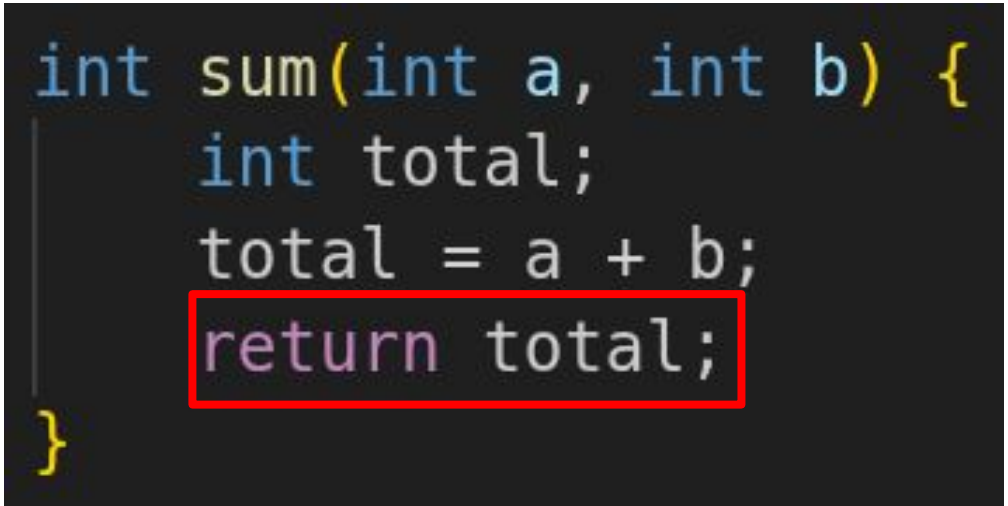
Recap Functions

- Inputs/arguments
  - Values our functions take
    - Have specified types



Recap Functions

- Return statement
  - How we return things from our function, to our code



Recap Function Prototypes

- We must define functions before we call them in main
  - Lets C know the function is defined and on its way!

```c
#include <stdio.h>

int sum(int a, int b);
int multiply(int a, int b);

int main(void) {
    int step_one;
    step_one = sum(12, 13);
    int step_two;
    step_two = multiply(step_one, step_one);
    printf("%d\n", step_two);
    return 0;
}

int sum(int a, int b) {
    int total;
    total = a + b;
    return total;
}

int multiply(int a, int b) {
    int total;
    total = a * b;
    return total;
}
```

Data Structures

- Different problems requires different optimum solutions
- In this course, you learn two pretty cool data structures
  - Arrays (NOW)
  - Linked lists (after flex week)
  - If you want more, Henry teaches COMP2521 once a year!
- Choosing the right data structure depends on understanding what we are trying to achieved
  - Some structures lend themselves better to certain kinds of problems

Arrays

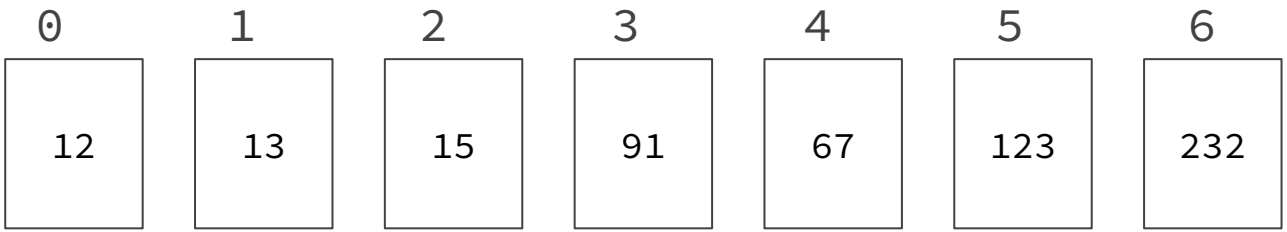
- Very important data type
- A collection of variables of the same type
  - Collection of ints
  - Collection of chars
  - Collection of doubles
  - Think about how this is different from a struct
- We want to be able to deal with this collection as a whole entity
  - Access any variable in the collection easily
  - Change any variable in the collection easily

What Problems Do Arrays Solve?

- Tracking daily Diet Coke consumption
- Tracking daily temperatures for the year
- Tracking how many seasons of Survivor I’ve seen
- Note how all of these collections are made up of the same kind of variable

Array (Visually)

- Grouping a data type as a collection
  - In this example, a collection of integers
- We can access them as a group (collection)
- We can loop through and access each individual element
- This array holds 7 elements
  - You can access the elements by referring to their index



Why Do We Need Arrays?

- Let’s track my Diet Coke consumption without arrays!
  - Any instance of more than 3 cans a day is too many

Why Do We Need Arrays?

- Let’s track my Diet Coke consumption without arrays!
  - Any instance of more than 3 cans a day is too many

```c
#include <stdio.h>

int main(void) {
    int mon = 0;
    int tues = 4;
    int wed = 3;
    int thurs = 1;
    int fri = 17;
    int sat = 4;
    int sun = 9;
    if (mon > 3) {
        printf("Too much diet coke!\n");
    }
    if (tues > 3) {
        printf("Too much diet coke!\n");
    }
    if (weds > 3) {
        printf("Too much diet coke!\n");
    }
```

Why Do We Need Arrays?

- What if I’m tracking this over a month?
  - 30 variables?
- Over a year?
  - 365 variables???
- Over my lifetime??
  - I don’t want to know how many variables...or how much Diet Coke...

Let's Use an Array!

- This is the declaration of an array!

```
int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
```

Let's Use an Array!

- Type of data stored in the array
  - All data in the array MUST be the same type
    - Different to other programming languages

Let's Use an Array!

- Name of the array
  - Follows same rules as naming any other variable

```
int diet coke[7] = {3, 13, 6, 1, 9, 67, 112};
```

Let's Use an Array!

- Number of items in the array
  - We'll cover arrays of unknown sizes later on

```
int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
```

Let's Use an Array!

- The array itself
  - Use curly brackets to initialise it
    - Comma separate the items
    - {} is an array of all 0s

```
int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
```

Let's Use an Array!

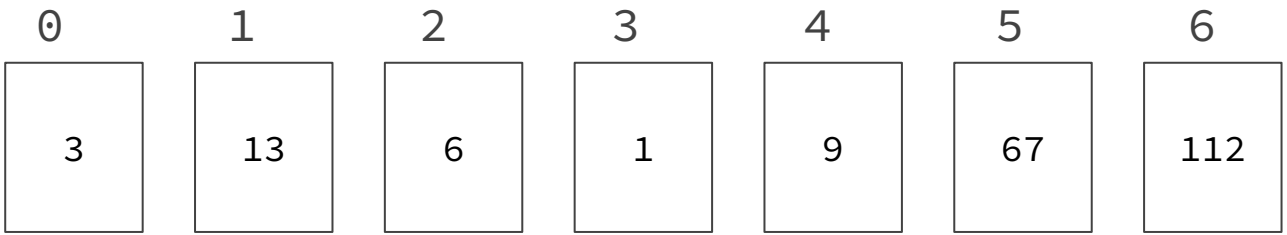
- So let's say we have this array
  - What does that look like visually?

```
int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
```

Array's Visually

- So let's say we have this array
  - What does that look like visually?

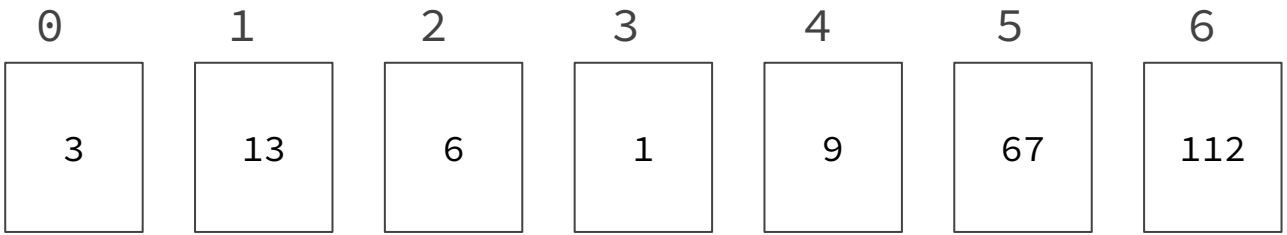
```
int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
```



Array's Visually

- This array holds 7 integers
  - Note that indexing starts at 0

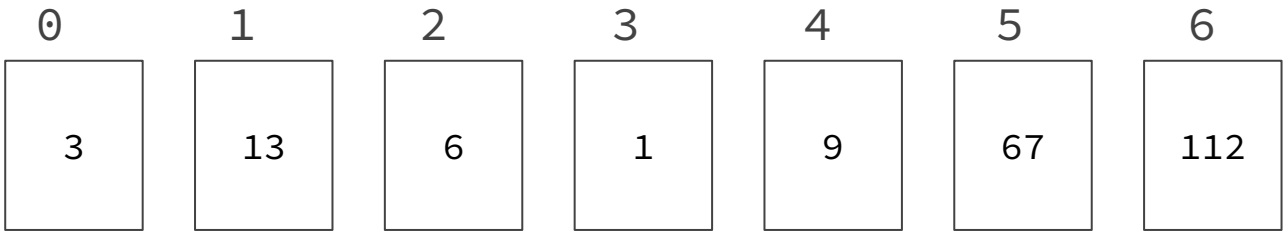
```
int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
```



Array's Visually

- If I wanted to access the second element of the array, I can
  - `diet_coke[1]`

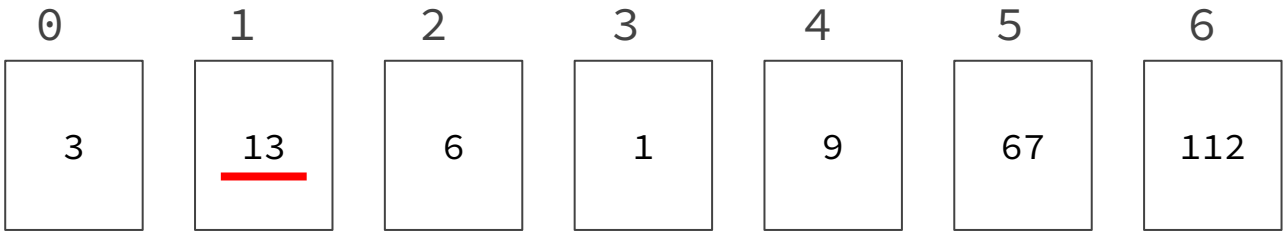
```
int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
```



Array's Visually

- If I wanted to access the second element of the array, I can
  - `diet_coke[1]`

```
int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
```



Using Arrays

- You cannot printf a whole array
  - But you can print individual elements
    - Think about how you'd print all elements in an array
- You cannot scanf a whole array
  - But you can scanf individual elements
    - Think about how you'd scan in an entire array

Using Arrays

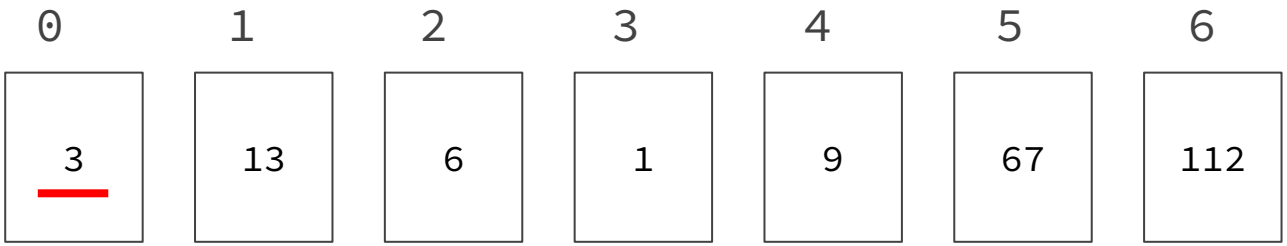
- Let's examine this code

```c
#include <stdio.h>

int main(void) {
    int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
    int i = 0;
    while (i < 7) {
        printf("%d\n", diet_coke[i]);
        i++;
    }
    return 0;
}
```

Using Arrays

- Start at index 0
  - Print whatever is stored at index 0
    - Move to the next index

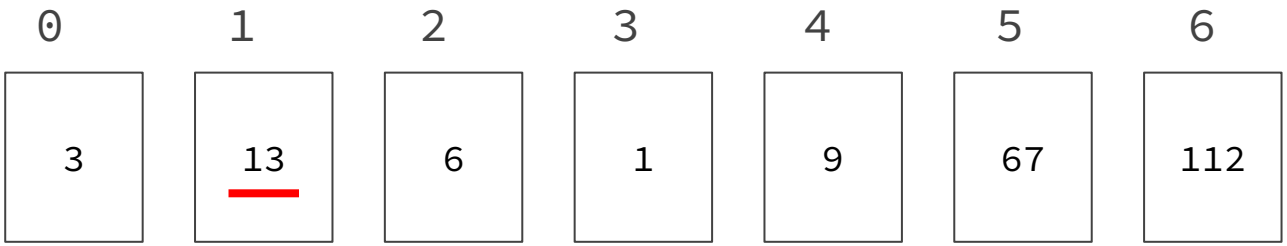
```c
#include <stdio.h>

int main(void) {
    int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
    int i = 0;
    while (i < 7) {
        printf("%d\n", diet_coke[i]);
        i++;
    }
    return 0;
}
```



Using Arrays

- Now at index 1
  - Print whatever is stored at index 1
    - Move to the next index

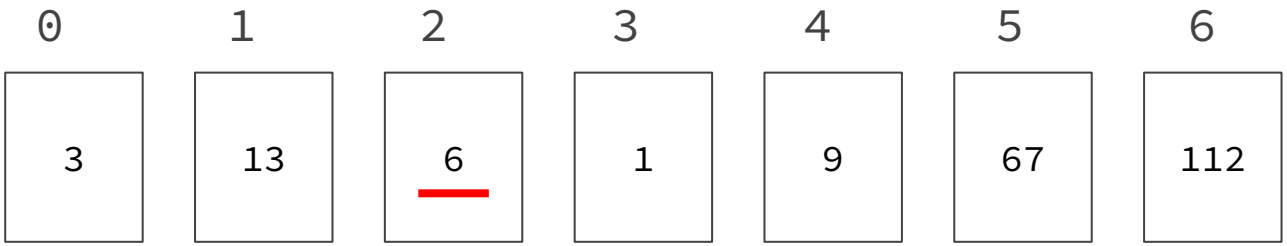
```c
#include <stdio.h>

int main(void) {
    int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
    int i = 0;
    while (i < 7) {
        printf("%d\n", diet_coke[i]);
        i++;
    }
    return 0;
}
```



Using Arrays

- Now at index 2
  - Print whatever is stored at index 2
    - Move to the next index

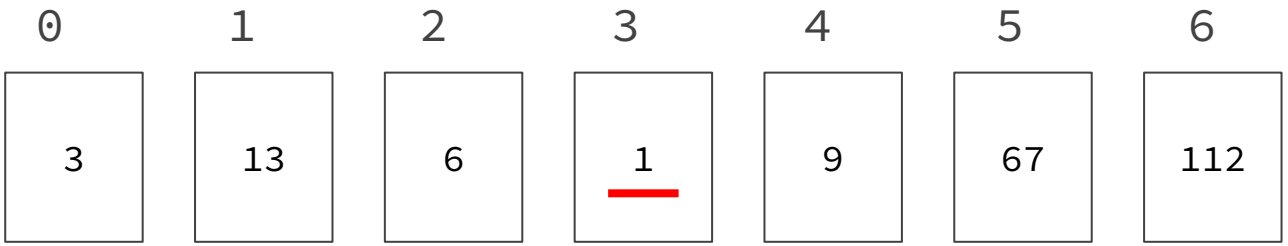
```c
#include <stdio.h>

int main(void) {
    int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
    int i = 0;
    while (i < 7) {
        printf("%d\n", diet_coke[i]);
        i++;
    }
    return 0;
}
```



Using Arrays

- Now at index 3
  - Print whatever is stored at index 3
    - Move to the next index

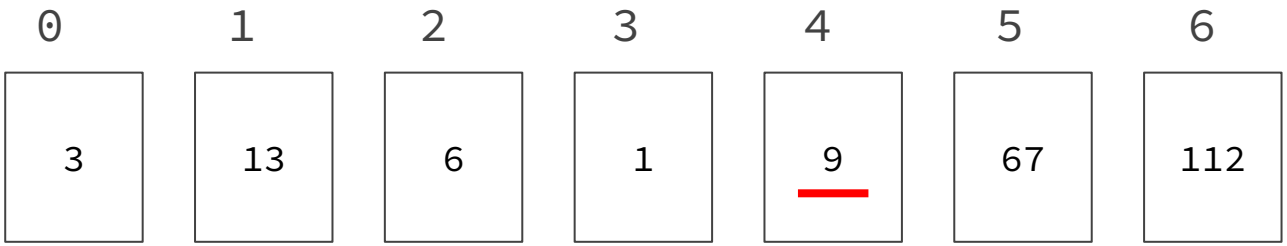
```c
#include <stdio.h>

int main(void) {
    int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
    int i = 0;
    while (i < 7) {
        printf("%d\n", diet_coke[i]);
        i++;
    }
    return 0;
}
```



Using Arrays

- Now at index 4
  - Print whatever is stored at index 4
    - Move to the next index

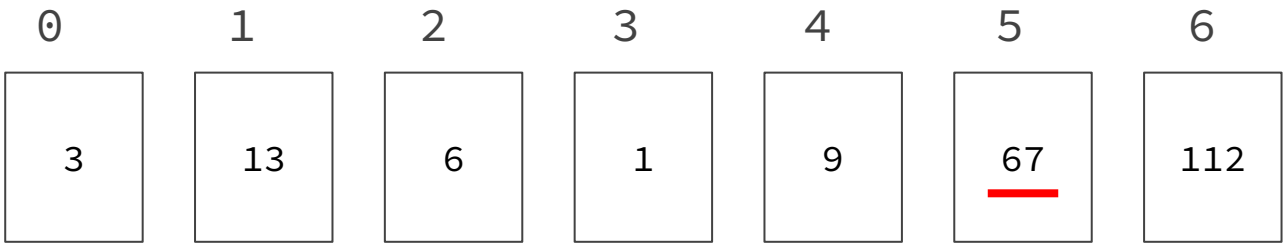
```c
#include <stdio.h>

int main(void) {
    int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
    int i = 0;
    while (i < 7) {
        printf("%d\n", diet_coke[i]);
        i++;
    }
    return 0;
}
```



Using Arrays

- Now at index 5
  - Print whatever is stored at index 5
    - Move to the next index

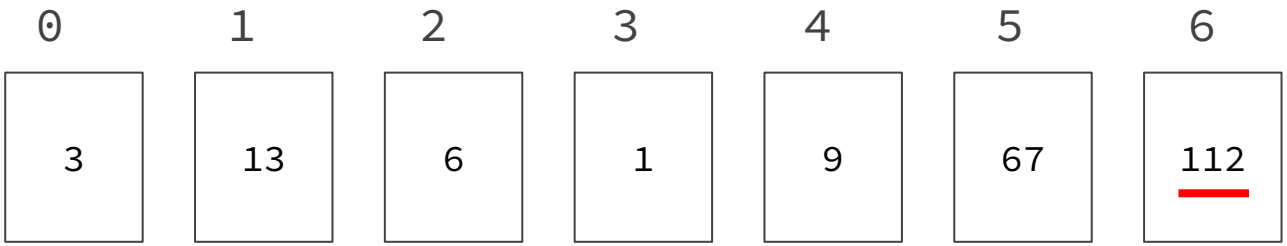
```c
#include <stdio.h>

int main(void) {
    int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
    int i = 0;
    while (i < 7) {
        printf("%d\n", diet_coke[i]);
        i++;
    }
    return 0;
}
```



Using Arrays

- Now at index 6
  - Print whatever is stored at index 6
    - Move to the next index

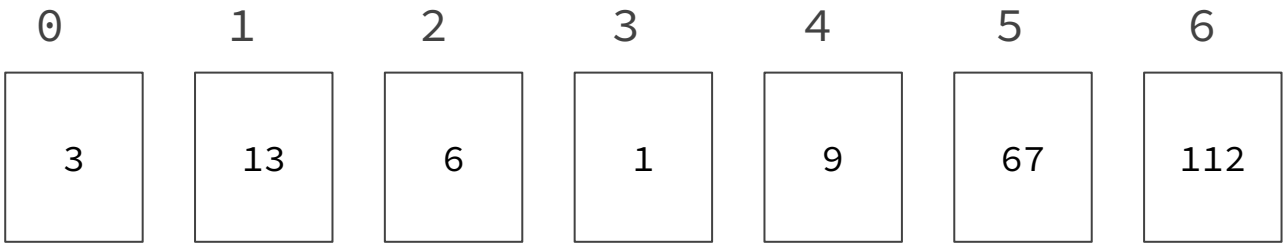
```c
#include <stdio.h>

int main(void) {
    int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
    int i = 0;
    while (i < 7) {
        printf("%d\n", diet_coke[i]);
        i++;
    }
    return 0;
}
```



Using Arrays

- There is no index 7
  - We are done printing this array
  - Yippee!!

```c
#include <stdio.h>

int main(void) {
    int diet_coke[7] = {3, 13, 6, 1, 9, 67, 112};
    int i = 0;
    while (i < 7) {
        printf("%d\n", diet_coke[i]);
        i++;
    }
    return 0;
}
```



Some Important Array Notes

- Arrays are contiguous in memory
  - This means, physically, array elements are stored next to each other in the computer's memory
    - This is not important right now, but will be important in a few weeks
- A string, or a word, is just a special array of characters!

Strings

- Arrays of chars
  - We put them in double quotes

Strings

- You can also define them one character at a time!
  - What's that '\0' doing there?

```c
#include <stdio.h>

int main(void) {
    char good_tv[] = {'S', 'u', 'r', 'v', 'i', 'v', 'o', 'r', '\0'};
    printf("%s\n", good_tv);
    return 0;
}
```

Break Time!

...probably

Using Arrays Problem

- There are 11 castaways at the merge of the hit TV show Survivor. You want to read all of their scores in the latest challenge, and find the winning index! Print the winner index, and the score associated with it. For fun, let’s also print the average score.

For Loops?

- If I have time, now I talk about for loops!

For Loops

- So far, you have learnt looping with while loops
- Some have asked about looping with for loops
  - They are very similar!
- My rule of thumb
  - If I know how many times I want to do something, use a for loop
  - If I’m uncertain, use a while loop
- while loops can do anything for loops can
  - for loops cannot do anything while loops can

For Loops

- Example
  - for loop to iterate over an array that I know the size of
  - for loop if I know the loop should be executed n times
  - while loop to get user input
  - while loop when the increment value is not standard (e.g not 1)
- They’re very similar, and accomplish the same goal, so don’t stress too hard about it!

For Loop Structure

- What does this for loop do?
  - How many times does it do it?

```
for (int i = 0; i < 10; i++) {
    printf("Iteration %d complete!\n", i);
}
```

For Loop Structure

- Increment
  - Executed at the end of each iteration

```
for (int i = 0; i < 10; i++) {
    printf("Iteration %d complete!\n", i);
}
```

While vs For

- These two loops accomplish the exact same task

```c
for (int i = 0; i < 10; i++) {
    printf("Iteration %d complete!\n", i);
}

int i = 0;
while (i < 10) {
    printf("Iteration %d complete!\n", i);
    i++;
}
```

What We Learnt

- Functions
  - How to break our code into smaller blocks
- Style
  - How to make our code readable
- For loops
  - An alternative to while loops

Reach Out

- Check the course forum for questions!
  - Post there too!
- Admin questions
  - cs1511@unsw.edu.au

Thank You

Questions?