What is a computer?

At the most fundamental level . . .

- a processor that executes instructions
  - often called Central Processing Unit (CPU)
  - typically a few square mm of silicon
- memory that holds information
  - often called Random-Access Memory (RAM)
  - usually contents loss if power lost/computer turned off
The Turing Machine

A theoretical model of computation

- Developed by Alan Turing in the 1930s
- There is a tape that can be infinitely long
- We have a “head” that can read or write to this tape
- We can move the head along to any part of the tape
- There’s a “state” in which the machine remembers its current status
- There’s a set of instructions that say what to do in each state
- Equivalent in power to all languages we use today
  - for every possible C program, there is an equivalent Turing machine program
A fanciful mechanical Turing machine’s TAPE and HEAD. The TABLE instructions might be on another "read only" tape, or perhaps on punch-cards. Usually a "finite state machine" is the model for the TABLE.
Turing Machines

- A tape and a read/write head
- Some idea of control of the head

Images from Wikipedia
The Processor

We also call them Central Processing Units (CPUs)

- Maintains a “state”
- Works based on a current set of instructions
- Can read and write from/to memory

In our C Programming

- State - where are we up to in the code right now
- Instructions - compiled from our lines of code
- Reading/Writing - Variables
**Examples of data storage used to store information**

- **Registers** (tiny fixed-sized pieces of memory on the CPU) - super-fast, tiny, volatile
- **Cache** (larger pieces of memory on the CPU) - fast, small, volatile
- **Random Access Memory (RAM)** - volatile
- **Hard Disk Drive**. All of these are huge, non-volatile
How does C use data storage?

On the **Hard Drive**
- Our C files (source code) are stored on our Hard Drive
- **dcc** compiles our source into another file, the executable program also stored on our Hard Drive

In **Random Access Memory**
- When we run our program, all the instructions are copied into RAM
- Our CPU will work through memory executing our instructions in order
- Our variables are stored in RAM as well
- Reading and writing to variables will change the numbers in RAM
A snapshot of a program in memory

Our instructions

What happens in memory when we run a program?

- Our Operating System gives us a chunk of memory
- Our program copies its instructions there
- A **stack** in one part of memory is used to track the current state
- The **stack** grows and shrinks as functions is called
- The variables for each function are stored are stored on the **stack**
- more . . . later - **Computers and programs are highly complex**
Arrays

When we need a collection of variables together

Sometimes we need a bunch of variables of the same type

We also might need to process them all

Our current use of ints and doubles might not be able to handle this

Let’s take a look at our current capability . . .
Let’s record everyone’s marks at the end of the term

We could do this as a large collection of integers . . .

def main():
    int marks_student1;
    int marks_student2;
    int marks_student3;
    int marks_student4;
    // etc
If we want to test all these ints

- In this situation:
- We need a whole bunch of nearly identical if statements
- There’s no way to loop through the integers
- Having to rewrite the same code is annoying and hard to read or edit
- So let’s find a better way . . .

```c
int main(void) {
    int marks_student1;
    int marks_student2;
    int marks_student3;
    int marks_student4;
    // etc
    if (marks_student1 >= 50) {
        // pass
    }
    if (marks_student2 >= 50) {
        // pass
    }
    // etc
```
Arrays

If our integers are listed as a collection

We’ll be able to access them as a group

We’ll be able to loop through and access each individual element

What is an array?

A variable is a small amount of memory

An array is a larger amount of memory that contains multiple variables

All of the elements (individual variables) in an array are the same type

Individual elements don’t get names, they are accessed by an integer index
// declare an array
int student_marks[10] = {0};

Similar, but more complex than declaring a variable

int - the type of the variables stored in the array
[10] - the number of elements in the array

Initialisation is much more complex:

= {0} - Initialises the array as all zeroes
= {0,1,2,3,4,5,6,7,8,9} - Initialises the array with these values
Array Elements

- An element is a single variable inside the array
- They are accessed by their index, an int that is like their address
- Indexes start from 0
- Trying to access an index outside of the array will cause errors
- In this example, element 2 of student_marks is 44 and element 6 is 62

<table>
<thead>
<tr>
<th>student_marks</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>55</td>
<td>70</td>
<td>44</td>
<td>91</td>
<td>82</td>
<td>64</td>
<td>62</td>
<td>68</td>
<td>32</td>
<td>72</td>
</tr>
</tbody>
</table>
C code for reading and writing to individual elements

```c
int main(void) {
    // declare an array, all zeroes
    int student_marks[10] = {0};
    // make first element 85
    student_marks[0] = 85;
    // access using a variable
    int access_index = 3;
    student_marks[access_index] = 50;
    // copy one element over another
    student_marks[2] = student_marks[6];
    // cause an error by trying to access out of bounds
    student_marks[10] = 99;
}
```
Printf and scanf with arrays

- We can’t printf a whole array
- We also can’t scanf a whole array
- We can do it for individual array elements though!
- The trick then becomes looping to access all individual elements one by one
Using printf and scanf with Arrays

```c
int main(void) {
    // declare an array, all zeroes
    int student_marks[10] = {0};
    // read from user input into 3rd element
    scanf("%d", &student_marks[2]);
    // output value of 3rd element
    printf("The 3rd Element is: %d\n", student_marks[2]);

    // the following code DOES NOT WORK
    scanf("%d%d%d%d%d%d%d%d%d", &student_marks);
}
```
Let's make a program to track player scores in a game

- We have four players that are playing a game together
- We want to be able to set and display their scores
- We also want to be able to see who’s winning the game
- The game needs to know how many points have been scored in total, so we’ll also want some way of calculating that total
Break down the program

What are the individual elements we need to make?

- First we create an array
- Then we use indexes to access the individual players and enter scores
- We’re going to need while loops to step through the array
- Most of the extra functionality we want will be done by looping through the array
Create the Array and populate it

Setting the elements using indexes (manually for now)

```c
#include <stdio.h>
#define NUM_PLAYERS 4
int main(void) {
    int scores[NUM_PLAYERS] = { 0 }; // assigning values directly to indexes
    scores[0] = 55;
    scores[1] = 84;
    scores[2] = 32;
    scores[3] = 61;
}
```

Source code for players.v0.c
Let’s loop through and see those values

Accessing all array elements by looping

- This is a pretty good candidate for code to put in a function later!

```c
// loop through and display all scores
int i = 0;
while (i < NUM_PLAYERS) {
    printf(
        "Player %d has scored %d points.\n",
        i,
        scores[i]
    );
    i++;
}
```

source code for players.v0.c
Now that we have our array

It will look a bit like this:

Next, we can loop through to find:

The highest

And the total
Finding particular values in an array

If we see all the values, we can easily find the highest

- We’ll loop through all the values in the array
- We’ll save the highest value we’ve seen so far
- Then replace it if we find something higher
- By the time we reach the end, we will have the highest value
Finding the highest score

We could put this in a separate function also!

```c
// finding the highest score
int highest = 0;
int index_highest = -1;
i = 0;
while (i < NUM_PLAYERS) {
    if (scores[i] > highest) {
        highest = scores[i];
        index_highest = i;
    }
    i++;
}
printf("Player %d has the highest score of %d.\n", index_highest, highest);
```
This is even easier than the highest!

We just add all the values to a variable we’re keeping outside the loop

```c
// finding the total
int total = 0;
while (i < NUM_PLAYERS) {
    total += scores[i];
    i++;
}
printf("Total points scored across the players is %d\n", total);
```

(source code for players.v0.c)
+= is another shorthand operator

It’s used for accumulating values in a variable

```c
int a = 0;
int b = 0;
// These two lines of code will do the same thing
// to their respective variables
a += 5;
b = b + 5;
// both a and b are now equal to 5
```
What about input into an array

Remember, we can’t access the whole array, only individual elements

- But we can definitely loop through the array entering values!

```c
// assigning scores using user input
int i = 0;
while (i < NUM_PLAYERS) {
    printf("Please enter Player %d's score: ", i);
    scanf("%d", &scores[i]);
    i++;
}
```

source code for players.v1.c
A Score Tracker

We’ve built our first program using an array

- We’ve accessed elements by index to set their values
- We’ve looped through to access values to output
- We’ve looped through to find highest
- We learnt about accumulating values
- We’ve also looked at reading values into the array
- We’ve seen how we can separate code into a function
// This code is inflexible and it won't handle e.g. 10000 numbers
// a much better approach is to use an array

int x0, x1, x2, x3, x4;
printf("Enter 5 numbers: ");
scanf("%d", &x0);
scanf("%d", &x1);
scanf("%d", &x2);
scanf("%d", &x3);
scanf("%d", &x4);
printf("Numbers reversed are:\n");
printf("%d\n", x4);
printf("%d\n", x3);
printf("%d\n", x2);
printf("%d\n", x1);
printf("%d\n", x0);

source code for reverse5_bad.c
// Flexible code using an array

int x[5]; // array size should be a #define not `5`

printf("Enter 5 numbers: ");

int i = 0;
while (i < 5) {
    scanf("%d", &x[i]);
    i = i + 1;
}

printf("Numbers reversed are:\n");

int j = 4;
while (j >= 0) {
    printf("%d\n", x[j]);
    j = j - 1;
}

source code for reverse5_good.c
Reading 5 Numbers then Printing in Reverse Order - Better

// change this #define to change how many numbers read
#define N_NUMBERS 5

int main(void) {
    int x[N_NUMBERS];
    printf("Enter %d numbers: ", N_NUMBERS);
    int i = 0;
    while (i < N_NUMBERS) {
        scanf("%d", &x[i]);
        i = i + 1;
    }
    printf("Numbers reversed are:\n");
    int j = N_NUMBERS - 1;
    while (j >= 0) {
        printf("%d\n", x[j]);
        j = j - 1;
    }
    return 0;
}
```c
int x[N_NUMBERS];
printf("Enter %d numbers: ", N_NUMBERS);
int i = 0;
int keep_looping = 1;
while (i < N_NUMBERS && keep_looping) {
    // stop loop if scanf doesn't read a number
    if (scanf("%d", &x[i]) != 1) {
        keep_looping = 0;
    } else {
        i = i + 1;
    }
}
printf("Numbers reversed are:\n");
int j = i - 1;
while (j >= 0) {
    printf("%d\n", x[j]);
    j = j - 1;
}
```
An Array is a type of variable

An Array can contain any type of variable

- Arrays can be put inside other arrays!
- We call these multi-dimensional arrays

Think of them as a grid, two or more dimensions
Arrays inside arrays
- can be thought of like a grid
- the outer array contains arrays
- each of those is a row of the grid
- addressed using a pair of integers like coordinates
- all inner arrays are of the same type and size
int main(void) {
    // declare a 2D Array
    int grid[4][4] = {0};
    // assign a value
    grid[1][3] = 3;
    // test a value
    // test a value
    if (grid[2][0] < 1) {
        // print out another value
        printf("The bottom left square is: %d\n", grid[3][0]);
    }
}
I would like to make a simple game called “The Tourist”

- The world is a square grid
- The tourist can move up, down, left or right
- Be able to print out the world, including the location of the tourist
- The tourist likes seeing new things . . .
- Track where they’ve been
- And lose the game if we revisit somewhere we’ve been
Here’s a handy function that we’ll be using to print the map

```c
void print_map(int map[MAP_ROWS][MAP_COLUMNS], int t_row, int t_column) {
    int row = 0;
    while (row < MAP_ROWS) {
        int column = 0;
        while (column < MAP_COLUMNS) {
            if(t_row == row && t_column == column) {
                printf("T ");
            } else {
                printf("%d ", map[row][column]);
            }
            column++;
        }
        row++;
        printf("\n");
    }
}
```
Break the problem down into parts

What do we need to do?

- We need to set up our grid and the tourist’s position
- The tourist needs to move one step at a time
- Each time the tourist visits a location, we set it to 1
- We also check each location to make sure it’s new
The Square Grid World

Variables for the grid and the tourist’s position

```c
#include <stdio.h>

// The dimensions of the map
#define MAP_ROWS 10
#define MAP_COLUMNS 10
#define EXPLORED 1
#define UNEXPLORED 0

// Helper Function: Print out the map as a 2D grid
void print_map(int map[MAP_ROWS][MAP_COLUMNS], int t_row, int t_column);

int main(void) {
    int map[MAP_ROWS][MAP_COLUMNS] = {0};
    // Set up tourist roughly in the middle of the map
    int tourist_row = MAP_ROWS/2;
    int tourist_column = MAP_COLUMNS/2;
```
Controlling the Tourist

Next Steps

- Let’s add movement
- Then track where the Tourist has been, using the map
- After that, we’ll check for places we’ve already been

Looping

- We can loop repeatedly for “turns” to allow the user to input directions
// Looping for multiple "turns"

int keep_looping = 1;

while (keep_looping) {
    // Track where we've been
    map[tourist_row][tourist_column] = EXPLORED;
    printf("Type in 2,4,6 or 8 to move in that num-pad direction: ");
    int input = 0;
    int input_result = scanf("%d", &input);
    if (input_result != 1) {
        keep_looping = 0;
    } else {
        // source code for tourist.c
    }
}
// input was a number
// check which direction to go
if (input == 8) {
    // up
    tourist_row--;}
else if (input == 6) {
    // right
    tourist_column++;
} else if (input == 4) {
    // left
    tourist_column--;}
else if (input == 2) {
    // down
    tourist_row++;
} else {
    // invalid or any other number
    keep_looping = 0;
}
We want the game to end if the tourist revisits a location

- If the location we visit is already 1
- Then we’re going to exit the game
- We can add this check after our movement

```c
// check to see if current location was already explored
if (map[tourist_row][tourist_column] == EXPLORED) {
    printf("I have already been here, HOW BORING!\n");
    keep_looping = 0;
}
```

source code for tourist.c
The Tourist Game

This is now roughly complete

- We can move the tourist
- We can track where we’ve been
- We can display where we’ve been as well as current location
- We can exit if we revisit a location

But how safe is it?

- Try different inputs
- Try moving around a bit
What happens if . . .

Moving around and seeing what works

- Use the controls to move around the map
- Try entering some integers that aren’t the movement

What issues do we find?
Walking off the edge of the map

Our Tourist can walk outside of the bounds of our arrays!

Let’s add some code to check if we’re outside the map and stop that movement

```c
// Check if we've gone off the map
// Clamp to legal map positions
if (tourist_row < 0) {
    // top
    tourist_row = 0;
} else if (tourist_row >= MAP_ROWS) {
    // bottom
    tourist_row = MAP_ROWS - 1;
} else if (tourist_column < 0) {
    // left
    tourist_column = 0;
} else if (tourist_column >= MAP_COLUMNS) {
    // right
    tourist_column = MAP_COLUMNS - 1;
}
```

Source code for tourist.c
What about scoring?

- Could we give the player a score based on the number of places they visited?
- How would we calculate that?
- Also . . .
- Some of this code might be useful in understanding the first assignment