COMP1511 - Programming Fundamentals

Term 1, 2019 - Lecture 7
Stream B

What did we learn last week?

Code Reviews

- Sharing our code to learn more and catch mistakes

Debugging

- The gentle art of finding and removing software errors (bugs)

Looping C

- More looping code as well as ints vs doubles

What are we covering today?

Computers as theoretical tools

- Fundamentals of what a computer is
- How we use memory in C

Arrays

- Using multiple variables at once

What is a computer?

At the most fundamental level . . .

- A processor that executes instructions
- Some memory that holds information

The Turing Machine

Originally a theoretical idea of computation

- 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

Turing Machines

Some images of Turing Machines

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

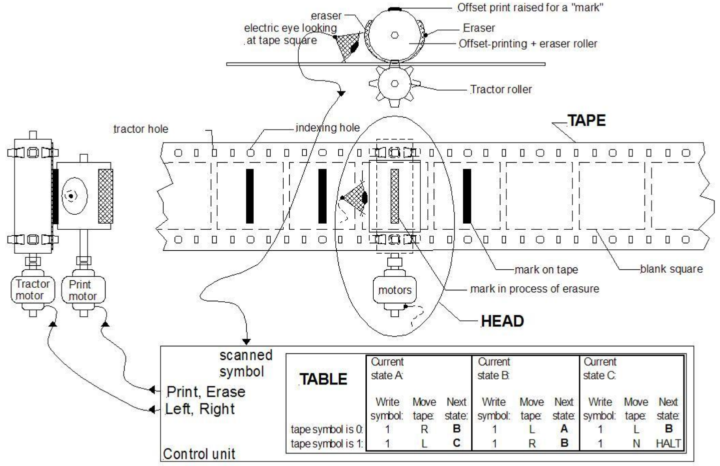
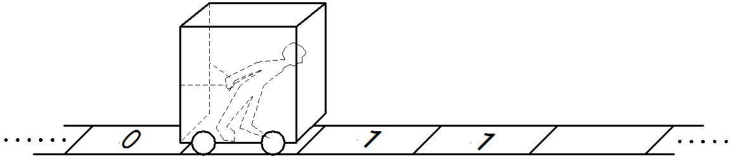


TABLE	Current state A:			Current state B:			Current state C:		
	Write symbol:	Move tape:	Next state:	Write symbol:	Move tape:	Next state:	Write symbol:	Move tape:	Next state:
tape symbol is 0:	1	R	**B**	1	L	**A**	1	L	**B**
tape symbol is 1:	1	L	**C**	1	R	**B**	1	N	HALT

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.



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

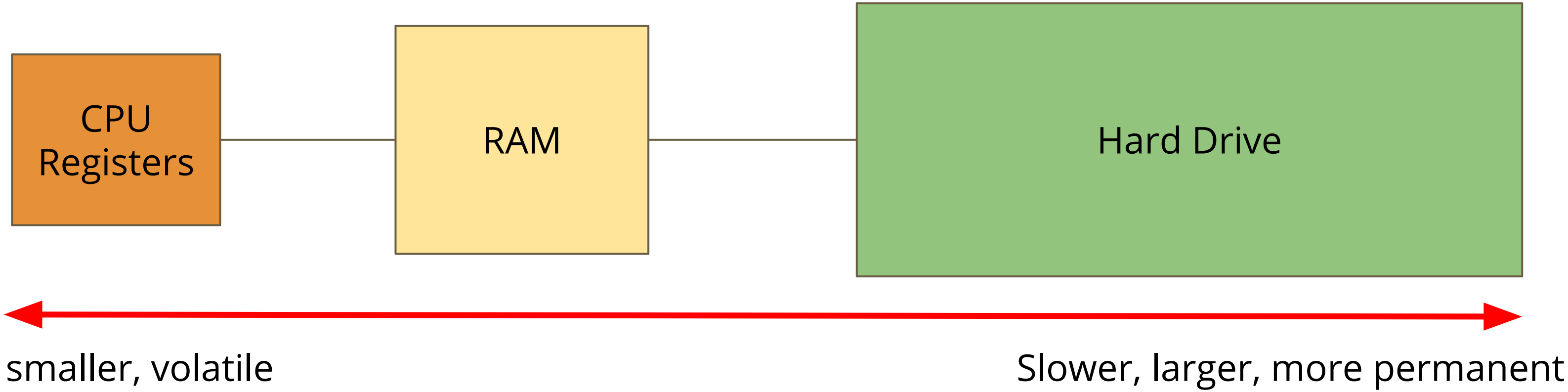
In our C Programming

- State - where are we up to in the code right now
- Instructions - compiled from our lines of code
- Reading/Writing - Variables

Memory

All forms of Data Storage on a computer

- From registers (tiny bits of memory on the CPU) through Random Access Memory (RAM) and to the Hard Disk Drive. All of these are used to remember something



How does C use memory

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

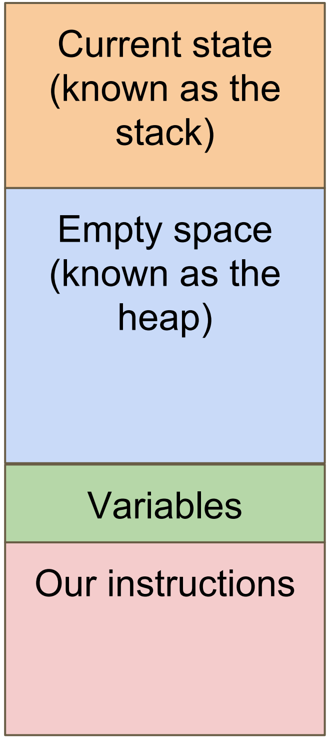
- In **Memory**
- When we run our program, all the instructions are copied into RAM
- Our CPU will work through memory executing our instructions in order
- Our variables are stored in RAM as well
- Reading and writing to variables will change the numbers in RAM

A snapshot of a program in memory

What happens in memory when we run a program?

- Our Operating System gives us a chunk of memory
- Our program copies its instructions there

- Some space is reserved for declared variables
- The **Stack** is used to track the current state
- The stack grows and shrinks as the program runs
- The **Heap** is empty and ready for use
- We can use the heap to store data while the program is running



Current state (known as the stack)

Empty space (known as the heap)

Variables

Our instructions

There’s more . . . later

Computers and programs are highly complex

- This was just an overview
- As you go through your learning, you will unlock more information
- For now, we have enough understanding to continue using C

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 (and why we need arrays) . . .

An Example

Let’s record everyone’s marks at the end of the term

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

```
int main (void) {
    int marksJames1;
    int marksJames2;
    int marksJames3;
    int marksJames4;
    // etc
```

If we want to test all these ints

We'd need a whole bunch of identical if statements

In this situation

- 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 . . .

```
int main (void) {
    int marksJames1;
    int marksJames2;
    int marksJames3;
    int marksJames4;
    // etc

    if (marksJames1 >= 50) {
        // pass
    }
    if (marksJames2 >= 50) {
        // pass
    }
    // etc
```

An Array of Integers

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

We’ll look at how they work after the break

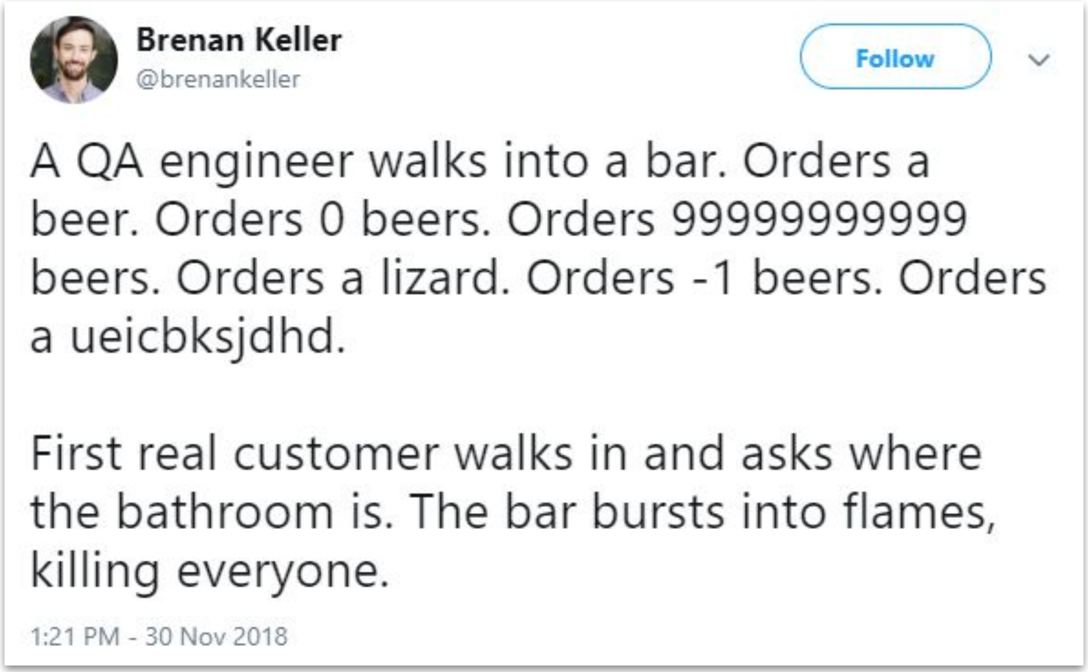
Break Time

Theory Behind Computers

- The idea of a processor and memory
- How C uses memory

Arrays

- We're moving on to collections of variables



A QA engineer walks into a bar. Orders a beer. Orders 0 beers. Orders 99999999999 beers. Orders a lizard. Orders -1 beers. Orders a ueicbksjdhd.

First real customer walks in and asks where the bathroom is. The bar bursts into flames, killing everyone.

1:21 PM - 30 Nov 2018

How to Approach Weekly Tests

The difference between labs and tests

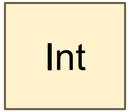
- Some people will try to complete all labs 100%
- This is possible
- Some people will try to get 100% in weekly tests
- This is only just maybe possible

- **0-1 Questions** - Come to help sessions, maybe do some extra reading
- **1-2 Questions** - You are doing fine, keep it up
- **2-3 Questions** - Things have gone very well this week, keep working!

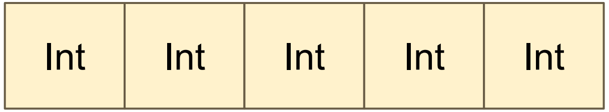
Arrays

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



A single integer
worth of memory



An array that holds 5 integers

Declaring an Array

Similar, but more complex than declaring a variable

```
int main (void) {
    // declare an array
    int arrayOfMarks[10] = {0};
```

- **int** - the type of the variables stored in the array
- **[10]** - the number of elements in the array
- **= {0}** - Initialises the array as all zeroes

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

	0	1	2	3	4	5	6	7	8	9
arrayOfMarks	55	70	44	91	82	64	62	68	32	72

In this example, element 2 of arrayOfMarks is 44 and element 6 is 62

Accessing elements in C

C code for reading and writing to individual elements

```
int main (void) {
    // declare an array, all zeroes
    int arrayOfMarks[10] = {0};

    // make first element 85
    arrayOfMarks[0] = 85;
    // access using a variable
    int accessIndex = 3;
    arrayOfMarks[accessIndex] = 50;
    // copy one element over another
    arrayOfMarks[2] = arrayOfMarks[6];
    // cause an error by trying to access out of bounds
    arrayOfMarks[10] = 99;
```

Reading and Writing

Printf and scanf with arrays

- We can’t printf a whole array
- We also can’t scanf a line of user input text into an array
- We can do it for individual elements though!

The trick then becomes looping to access all individual elements one by one

User input/output with Arrays

Using printf and scanf with Arrays

```
int main (void) {
    // declare an array, all zeroes
    int arrayOfMarks[10] = {0};

    // read from user input into 3rd element
    scanf(“%d”, &arrayOfMarks[2]);
    // output value of 5th element
    printf(“The 5th Element is: %d”, arrayOfMarks[4]);

    // the following code DOES NOT WORK
    scanf(“%d %d %d %d %d %d %d %d %d %d”, &arrayOfMarks);
```

Let’s make a basic program using Arrays

Let’s use an array to store the marks of a class of students

- The program will have an array of five students’ marks
- It will output all the marks to verify that they were correct
- It will then tell us what the average marks were

Break it down

As always, start simple and build up

- We’ll start by creating an array
- Then we’ll access the elements to put values in
- Finally, we’ll loop through, accessing elements by index to output them

Creating the Array in Code

Assigning elements via their index

```
int main (void) {
    // declare the array, size 5
    int arrayOfMarks[5] = {0};

    // enter the marks (we’re doing this manually for now)
    arrayOfMarks[0] = 63;
    arrayOfMarks[1] = 88;
    arrayOfMarks[2] = 43;
    arrayOfMarks[3] = 55;
    arrayOfMarks[4] = 67;
```

Let’s loop through and see those values

Accessing all array elements by looping

```
// continued from last slide
// loop through the array and output the elements
int counter = 0;
while (counter < 5) {
    printf(“%d\n”, arrayOfMarks[counter]);
    counter++;
}
```

Now that we have our array

It will look a bit like this:

	0	1	2	3	4
arrayOfMarks	63	88	43	55	67

Next, we can loop through to find:

- The lowest
- The highest
- And the average

Looping and Arrays

```
// continued from previous slides
// loop through the array and add up the marks
counter = 0;
int total = 0;
while (counter < 5) {
    total += arrayOfMarks[counter];
    counter++;
}
double numElements = 5;
double avgMark = total/numElements;
printf(“Average Mark was: %lf\n”, avgMark);
```

Wait, what was that new syntax?

+= is another shorthand operator

- It’s used for accumulating values in a variable

```
int a = 0;
int b = 0;

// These two lines of code will do the same thing
a += 5;
b = b + 5;

// both a and b are now equal to 5
```

What about input into an array?

This program would be much more useful if we could input marks

- We can run scanf inside a loop to enter values

```
int main (void) {
    // declare the array, size 5
    int arrayOfMarks[5] = {0};

    // enter the marks from user input by looping
    int counter = 0;
    while (counter < 5) {
        scanf(“%d”, &arrayOfMarks[counter]);
    }
```

A Marks Calculator

Now we have a program that totals marks and calculates an average

- It uses an array to store multiple similar values
- We’ve looked at accessing elements of an array
- We’ve also looked at looping through the array for different purposes

Challenges

- Can you find the highest and lowest marks?
- Can you also output which indexes you found the highest and lowest in?

What did we learn today?

Computers in Theory

- A processor and some memory
- Turing machines as theoretical computers
- How C works in memory

Arrays

- How to make and use arrays of integers
- How to loop through arrays