What did we learn last week?

Assignment 1

● Everything you need to know about CS Paint!

Characters and Strings

● Using letters and words in C

Memory and Pointers

● Memory addresses and how to use them
What are we covering today?

Command Line Arguments
- Adding information to our program when it runs

Professionalism
- Some important skills as a programmer

Pointers continued
- Directly addressing memory
Characters and Strings Recap

Our new variable type: `char`

- Represents a letter
- Is also a number, an ASCII code, and we'll often use `ints` to represent a character
- When used in arrays, they're referred to as strings
- Strings often end before the end of the array they're stored in
- When they do, we store a null terminator `\0` after the last character
Strings in Code

Strings are arrays of type char, but they have a convenient shorthand

```c
// a string is an array of characters
char word1[] = {'h','e','l','l','o'};
// but we also have a convenient shorthand
// that feels more like words
char word2[] = "hello";
```

Both of these strings will be created with 6 elements. The letters h,e,l,l,o and the null terminator \0
Command Line Arguments

Sometimes we want to give information to our program at the moment when we run it

- The "Command Line" is where we type in commands into the terminal
- Arguments are another word for input parameters

```
$ ../program extra information 1 2 3
```

- This extra text we type after the name of our program can be passed into our program as strings
Main functions that accept arguments

**int** main **doesn't have to have** **void** **input parameters!**

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

- **argc** will be an "argument count"
  - This will be an integer of the number of words that were typed in (including the program name)
- **argv** will be "argument values"
  - This will be an array of strings where each string is one of the words
An example of use of arguments

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    int i = 1;
    printf("Well actually %s says there's no such thing as ", argv[0]);
    while (i < argc) {
        fputs(argv[i], stdout);
        printf(" ");
        i++;
    }
    printf("\n");
}
```
Arguments in argv are always strings

But what if we want to use things like numbers?

- We can read the strings in, but we might want to process them

$ ./program extra information 1 2 3

- In this example, how do we read 1 2 3 as numbers?
- We can use a library function to convert the strings to integers!
- \texttt{strtol()} - "string to long integer" is from the stdlib.h
Code for transforming strings to ints

Adding together the command line arguments

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int total = 0;

    int i = 1;
    while (i < argc) {
        total += strtol(argv[i], NULL, 10);
        i++;
    }
    printf("Total is %d.\n", total);
}
```
What does it mean to be a professional engineer?

Four pillars of being a professional:

1. Communication
2. Teamwork
3. Resilience
4. Technical Skills
Communication

Does everyone understand what you're working on?
Communication

Making sure everyone understands what you’re doing

- Problem solving in teams involves shared understanding
- In order to solve human problems, we must understand what people need and how we can help them
- The more we communicate with computers the more risk we have of treating people like machines
- The ability to explain our code is important to keep us on track
- It’s especially important to be able to explain your code to non-programmers
Teamwork

Code is very rarely created alone

Teams that get along are usually more successful than teams of pure skill
Teamwork

Code is very rarely created alone

- Teamwork involves sharing and compromise
- Can you work with other people’s ideas?
- Can you follow someone else’s style and structure?
- Can you adapt your structure so that other people can use it?
- Can you provide support to your teammates?
- Teams made of people who get along are usually more successful than teams made of very skilled individuals!
Resilience

Work is hard. We need to look after ourselves

- You shouldn't have to "survive" your job
- Dealing with "impossible situations"
- Failure is inevitable, what counts is how you recover, not whether you fail
Technical Skills

How’s your programming?

- Yes, this comes last in the list
- It's still important but it can't be your only focus
- Still, we have the majority of our degrees to learn technical programming
More about Resilience and Surviving

You have an assignment due soon

- Success isn’t about getting everything done
- It’s about prioritising your effort so you don’t have to do as much work!

Priorities:

- What gets you the most marks with the least amount of time?
- Code Style?
- Clean, basic functionality?
- There are more marks in the earlier stages than later
- Aim for what you can achieve without burning out
Don’t Panic!

Surviving is about acting rationally in panicky situations

- Take a moment to breathe
- Figure out what your options are
- Break problems down into small bits
- Complete small pieces one at a time
- Aim for whatever gets you enough
Don’t Panic!

Surviving is about acting rationally in panicky situations

- Take a moment to assess where you’re up to
- Figure out what your options are
- Break everything down into small bits
- Complete small pieces one at a time
- Aim for whatever gets you the highest marks
Becoming a Professional

It doesn't have to happen yet . . . and it's always ongoing learning!

- Remember to communicate with colleagues
- Follow as well as lead when you're in a team
- Look after yourself
- And above all . . .
- Care about yourself, the people around you and your work
Learning something new is better than being good at something!

Remember . . . as nice as high marks are, they're not the same as long term fulfilment

"I don't care who you are, where you're from, what you've done . . . as long as you love C." - The Backstreet Boys
Pointers Recap

Pointers are Memory Addresses

- We'll use pointers to remember where variables are.
- The value stored in a pointer is an address in memory.
- `*` is used to declare a pointer.
- After it's created `*` is used to dereference a pointer - find the value of the variable the pointer is "aimed at".

```c
int i = 100;
// create a pointer called ip that points at
// an integer in the location of i
int *ip = &i;
```
Pointers in use

- `&` is used to find the address of a variable
- It can be used to assign an address to a pointer

```c
int i = 100;
// create a pointer called ip that points at
// the location of i
int *ip = &i;
printf("The value of the variable at %p is %d", ip, *ip);
```
Ok let's make a simple program

This program is called The Jumbler

- It will take some numbers as command line arguments
- It will jumble them a little, changing their order
- Then it will print them back out

- We'll make some use of functions and pointers here!
What functions do we want?

Deciding how to split up your functionality

- A function that reads the command line arguments as integers
- A function that swaps two numbers
- A function that swaps several numbers
- A function that prints out our numbers
Converting our Command Line Arguments

We'll read the command line arguments and convert them to ints

- Note that we're ignoring the first element of arguments because we know that it's the name of the program and not one of our numbers

```c
void read_args(int nums[MAX_NUMS], char *arguments[], int argCount) {  
    int i = 0;  
    while (i < MAX_NUMS && i < argCount - 1) {  
        nums[i] = strtol(arguments[i + 1], NULL, 10);  
        i++;  
    }  
}
```
Printing our numbers

This is a trivial function

- The only issue is that we might have to work with an array that isn't full
- So we use numCount to stop us early if necessary

```c
void print_nums(int nums[MAX_NUMS], int numCount) {
    int i = 0;
    while (i < MAX_NUMS && i < numCount) {
        printf("%d ", nums[i]);
        i++;
    }
}
```
Using Pointers to swap variable values

A simple swap function

- This function doesn't even know whether the ints are in arrays or not
- It sees two memory locations containing ints
- and uses a temporary int variable to swap them

```c
void swap_nums(int *num1, int *num2) {
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```
Jumble performs some swaps

This function just loops through and swaps a few numbers

- This is a good candidate for a function that could be changed or written differently and just used by our main without thinking about it

```c
void jumble(int nums[MAX_NUMS], int numCount) {
    int i = 0;
    while (i < MAX_NUMS && i < numCount) {
        int j = i * 2;
        if (j < MAX_NUMS && j < numCount) {
            swap_nums(&nums[i], &nums[j]);
        }
        i++;
    }
}
```
Using all the functions in the main

A nice main makes use of its functions

- It's very easy to read this main!
- It shows its steps using its function names
- There isn't much code to dig through

```c
int main(int argc, char *argv[]) {
    int numbers[MAX_NUMS];
    read_args(numbers, argv, argc);
    jumble(numbers, argc - 1);
    print_nums(numbers, argc - 1);
    return 0;
}
```
What did we learn today?

Command Line Arguments
- Reading input that's typed in with the program command

Professionalism
- Being ready for a career in computing

Pointers in Functions
- Using pointers in a program with functions