more gcc options

```
$ gcc digit.c -lm -o digits
```

- `-lm` — link in the `m` library (`libm.so`)
- `-o digits` — call the output file digits intead of default

`gcc` is actually carrying out multiple steps here[1]:

- preprocessing — dealing with things that start with #
- compiling — turn source into executable form
- linking — Get missing functions from libraries (eg `printf()`)
  - eg `stdio.h` tells the compiler that `printf()` exists but not what it does.

[1] not an exhaustive list

Aside: printf `man` page(s)

What about `printf()`?

```
$ man printf
PRINTF(1)          User Commands          PRINTF(1)
...

$ man -k printf
...
```

Man sections (from `man man`):

1 Executable programs or shell commands
2 System calls (functions provided by the kernel)
3 Library calls (functions within program libararies)
...

So we want `man 3 printf`

Grade calculation

See `mark1.c`.

mark1.c

Note:

- `me + 0.85 * fe`
  - Operator precedence
- `if (ex1 > ex2)`
  - Conditions are delimited by parentheses
  - `else` clause is optional
  - Braces around each branch are optional[2] for single statements.
  - Indenting means nothing.

[2] By language rules. Our style guide has views!

See `mark2.c`

mark2.c

Ternary operator:

```
double totalEx = (ex1 > ex2) ? ex1 : ex2;
```

Syntax: *exp1* **?** *exp2* **:** *exp3*

$$\text{Evaluates to} \begin{cases} exp2 & \text{if } exp1 \text{ is true} \\ exp3 & \text{else} \end{cases}$$

If used badly, this can make code hard to read. Use it sparingly.

Compile and run `tern1.c`.
Examine source.
Do not ever do that!

mark3.c

This line has some problems

```
int totalA = (a1 + a2 + a3 + a4) / 2;
```

In C: `int` / `int` $\rightarrow$ `int`[3]

Changing the type of `totalA` to `double` isn't enough.

```
double totalA = (a1 + a2 + a3 + a4) / 2.0;
```

We could also have used:

- (double)2
- (1.0 * 2)

[3] Holds for other operators as well

mark4.c

The previous version's structure is limited for three reasons:

1. It mixes setup with the calculation itself
2. It is not simple to repeat the calculation
3. All values are hardcoded (We'll fix that later)

Create a function to perform the calculation.
See `mark4.c`

Note:
Even though it doesn't take input, you could still use a structure like `mark4.c` for debugging or testing.

Prototypes

See `mark5.c`

(Fixed size) Arrays

Rather than list each value separately we can load them into an array and pass that.

See `mark6.c`

The syntax for C arrays looks similar to other languages (java arrays / python tuples). However:

- The size of the array must be specified when creating it.
- The size is part of the type, so `int a[3]` and `int a[4]` are different types.
- Whole arrays can't be compared
- Arrays can't be reassigned once created
  - `int b[3]`
  - `int c[3]`
  - `c=b` Not allowed

`struct`

Arrays are convenient if:

- All values are of the same type
- Don't care which is which

Structs also group items together but each one has a name.

See `mark7.c`

Note:

- You need the `;` after the closing brace
- The name of the type is `struct marks` not `marks`
- The declaration from line 4 to line 11 *declares a type* not a variable.

Arrays in structs

See `mark8.c`

Note:

- C does not initialise variables by default.
- Unless you have compiler warnings turned up, it won't warn you.

for loop

C uses `for` not a "for each" loop (as in python and some Java loops).

```
for (int i = 0 ; i < 4; ++i) {
    totalA += values.assignments[i];
}
```

1. `int i = 0` — Done once at the beginning of the loop
2. `i < 4` — If this is false, stop the loop
3. `totalA +=...` — Loop body
4. `++i` — Done after the loop body. Now jump to 2

1 2 3 4 **2** 3 4 **2** 3 4 **2** 3 4 **2** (i==4)

for

All three parts of the loop header are optional:

```
for (;remain > 0;)

for (;;) // loop forever

for (A; B; C) {
    BODY
}
```

is equivalent to:

```
A;
while (B) {
    BODY;
    C;
}
```

do while

```
do {
    BODY
} while (TEST);
```

This loop will execute BODY at least once. vs:

```
while (TEST) {
    BODY;
}
```

which might not execute BODY at all (If TEST fails first time).