CSSE2310/7231 — 1.2

Compiling C and initial features.

Expressions

In programming an expression is a fragment of code which can be "evaluated" to get a value.

Eg:

Literals	`1.2`
	`"a string"`
	`'A'`
Variables	`cost`

Expressions can be combined:

Operators	`a + b`	
	`y+m*x`	// Precedence
Function calls	`get_cost(3)`	
	`f1(nested('A', x + 2))`	
...	`costs[3]`	

Types

In C expressions (and variables) have explicit types.

`int` — signed whole number

Python	C	
	int x;	Explicit declaration
x = 4	x = 4;	
y = 5	int y = 5;	
x = x + y	x = x + y	
x = "cats"		strings aren't **int**s

First program

```c
// Function called "main" returns an int
// and takes two parameters
int main(int argc, char** argv) {
    return 0;
}
```

- C source files should end in `.c` eg `init.c`
- Filenames are case sensitive even if your OS Filesystem isn't

Compile and run

Source is not executable

```
$ ./init.c
bash: ./init.c: Permission denied
$ file init.c
init.c: C source, ASCII text
```

Need to compile source into a program.

```
$ gcc init.c
$ file a.out
a.out: ELF 64-bit LSB pie executable, x86-64, ...
$ ./a.out
$ echo $?
```

- The return value from main is sent outside the program.
- In bash this is available as $?
- Only works until the next command is run

Extra options

You can change what the compiler checks for with optional parameters (not an exhaustive list):

- `-std=c99` or `-std=gnu99` — Which version of the language. You'll *need* at least c99
- `-g` — Add debug information
- `-Wall` — Switch on all "avoidable" warnings.
- `-pedantic` — Warn about more problems
- `-O2` — optimise the build. Don't bother with this especially if you are debugging
- `-Werror` — a single warning stops the build. Don't use this unless you really mean it!

printf

A call to printf starts with a format string containing:

- Escape characters – start with \ (eg '\n', '\t')
- Place holders — start with % (Substitute in an expression)
- Normal characters

```
printf("Text here\n");   // Text + escape character
printf("3+5=%d\n", 8);
printf("3+5=%d\n", (3+5));
```

Place holders must (correctly) describe the type of the expression being substituted.

Output — numbers

Type	Symbol	
int	%d	
unsigned int	%u	
double	%e	Exp notation
		$12.34 \rightarrow 1.234000e+01$
	%f	$12.34 \rightarrow 12.340000$
	%g	Combination
		$12.34 \rightarrow 12.34$
char	%c	'c'→c
		99→c

More C types later.

More gcc options

Count digits in a positive integer:

```
int main(int argc, char** argv) {
    int number=54;
    int result=(int)log10(54)+1;
    printf("%d has %d digits\n", number, result);
    return 0;
}
```

- Note casting syntax: `(int)expression`.

manual pages

```
$ man log10
```

- The first line tells us which page `LOG10` and section 3.
- `SYNOPSIS`
  - `#include` — which header file do we need to include?
  - `Link with` — do we need additional libraries?

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