Introduction to C

Session 2

March 25

Dr Mark Doughty, School of Engineering and Physical Sciences,
University of Lincoln.

First, some definitions…

- A **variable** is a named value or expression in memory
- The **datatype** of a variable in memory determines its:
  - Size (how much memory it uses)
  - What operations can be done with it
- **Operators** determine how variables can be changed:
  - Unary operators (they only need one value): `-, ++`
  - Binary operators (they need two values): `+, -, *, /, %`
  - Ternary operators (they need three values): `?`

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

First, some definitions…

- An **expression** combines variables and operators – anything which evaluates to a value, eg:
  - `a+b`
  - `1234`
  - `A>=B`

- A **statement**, is a line of code which ‘does something’
  - Assigns an evaluated expression to a variable
  - Combines variables, functions, operators
  - `a = b + c;`
  - `x = printf(y);`

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

First, some definitions…

- **lvalue** (or **l-value**) is anything which can have an identifiable location (a memory address)
  - An l-value can appear on both sides of an assignment operator (=)
  - `a = b;` (a and b are l-values)

- **rvalue** (or **r-value**) is something which doesn't have an identifiable location (i.e. have an address)
  - An r-value can only appear on the right hand side of an assignment operator
  - `a = b + 1;  //ok, 'b+1' is an r-value`
  - `b + 1 = a;  //not ok, will not compile. 'b+1' cannot go on left`

- Easy way to remember is **l**-values can appear on **left** of assignment, **r**-values can only appear on the **right**

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Variable names

- ...can contain letters and digits
- ...should **only** begin with a letter
- ...keywords cannot be used (‘`if`’, ‘`where`’, etc)
- ...are case sensitive (`a` and `A` are two different variable names)

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

c datatypes

- Numbers (`int, float, double`)
- Character (`char`)
- Boolean (`bool`) (need to use the <stdbool.h> header)
- User defined (`struct, union`)

- There are variations of these datatypes which can be used depending on the precision and range required (signed/unsigned, short, long)
  - Have a look here: https://en.wikipedia.org/wiki/C_data_types

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

The sizeof(..) operator

- Returns the size of the memory **in <u>bytes</u>** that a datatype or variable occupies

```
int x;
double y;
char z;
x = 10;
y = 2.12345;
printf("%d", sizeof(z));
```

Outputs the amount of memory used by (in this case) a `char`

Amounts will depend on your compiler, and c standard doesn't specify a size:
(so these may be different depending on compiler flags/platform)

`int` at least 16 bits (4 bytes)
`float` at least 16 bits (4 bytes)
`double` at least 32 bits (8 bytes)
`char` at least 4 bits (1 byte)

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Declaring and using variables

- Can do this in different ways, eg:

```
int x;
int y;
printf("%d", x);
```

Declares two `int`s, outputs the default value for one (in this case, default = 0)

```
int x, y;
printf("%d", x);
```

Declares like types together.

```
int x = 10;
int y;
printf("%d", x);
```

Declare and initialise together.

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Arithmetic Operators

(The values operated on are called ‘operands’)

+	addition	p = q + 8; p = q + r;
-	subtraction	p = q - 8; p = q - r;
*	multiplication	p = q * 8; p = q * r;

/	division	float p = 5/3; // p = 1 (integer division)
		float p = 5.0/3; // p = 1.66667 (float division)
		int p = 5.0/3; // p = 1 (float truncated to an int)

%	Mod	p = 5%3; // p=2 (5/3 is 1 remainder 2)

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Relational Operators

- Compare two operands to produce a Boolean result (true or false)
- In C (like many other programming languages), 1 can represent ‘true’ and 0 ‘false’.

>	‘greater than’	5>6 //evaluates as ‘false’ or 0
<	‘less than’	5<6 //evaluates as ‘true’ or 1
>=	‘greater than or equal to’	4>=4 //evaluates as ‘true’ or 1
<=	‘less than or equal to’	4<=3 //evaluates as ‘false’ or 0

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Relational Operators

- Testing equality between two variables is one of the most common tests:

==	‘is equal to’	4==5; //evaluates as ‘false’ or 0
!=	‘is not equal to’	4!=5; //evaluates as ‘true’ or 1

Note!

- The ‘=‘ operator is the assignment operator. It does not test for equality
- Do not use float values with ‘==‘

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Logical Operators

- Use one or more operands with these binary logic operators to combine Boolean values:

```
&&      AND  // (9==9)&&(6/2==3) evaluates to ‘true’ (true && true)
             // (9==9)&&(6/3==3) evaluates to ‘false’ (true&&false)

||      OR   //(9==9)&&(6/3==3) evaluates to ‘true’ (true||false)

!       NOT  // !(9==9) evaluates to ‘false’ (!true)
```


Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Logical Operators

- Be careful! These logical operators will terminate if an early condition indicates it doesn't need to go on, eg:

```
(9==5)&&(getchar()=='q')
```

The 'getchar' function is not run – the first part is 'false', so the AND will be false.

```
(9==9)||(getchar()=='q')
```

The 'getchar' function is not run – the first part is 'true', so the OR will be true.

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Bitwise Operators

<< left shift Shifts the bits to the left by the number on the right

```
int x = 6;
printf("%d", x<<1);
```

(6) (12)

0110 << 1 = 1100

>> right shift Shifts bits to the right by the number on the right

```
int x = 6;
printf("%d", x>>1);
```

(6) (3)

0110 >> 1 = 0011

Good to remember: Shifting 1 bit position left multiplies the number by 2. Shifting 1 bit position to the right, divides the number by two. (Easier to let compiler/CPU decide now – a multiply may be implemented as shifts for efficiency)

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Bitwise Operators

& Bitwise AND

```
int x = 6;
int y = 4;
printf("%d", x&y);
```

6&4 = 0110 & 0100 = 0100 = **4**

Each bit is compared. The resulting bit is 1 if both are 1's

| Bitwise OR

```
int x = 6;
int y = 1;
printf("%d", x|y);
```

6|1 = 0110 | 0001 = 0111 = **7**

Each bit is compared. The resulting bit is 1 if either are 1's

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Shortcut operators

- Increment/decrement

`++q;` is a shortcut for `q = q + 1;`

`--q;` is a shortcut for `q = q – 1;`

Be careful here!

++q; and q++; are both valid.

```
int x = 6;
printf("%d", ++x);
```

Prints ‘7’

```
int x = 6;
printf("%d", x++);
```

Prints ‘6’
…but, x is now 7!

```
int x = 6;
printf("%d ", x++);
printf("%d", x);
```

Prints ‘6 7’

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Shortcut Operators

- Shortcut assignments:

```
w = w + 3;          w+=3;
d = d – 8;          d-=8;
z = z * 12;         z*=12;
f = f/2;            f/=2;
c = c % 4;          c%=4;
```


Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Operator Precedence
(Order of Evaluation)

- Check https://en.cppreference.com/w/c/language/operator_precedence for a full list

- Operations with a higher precedence are done first.

Precedence	Operator	Description	Associativity
1	++ -- () [] . -> (*type*){*list*}	Suffix/postfix increment and decrement Function call Array subscripting Structure and union member access Structure and union member access through pointer Compound literal(C99)	Left-to-right
2	++ -- + - ! ~ (*type*) * & sizeof _Alignof	Prefix increment and decrement[note 1] Unary plus and minus Logical NOT and bitwise NOT Cast Indirection (dereference) Address-of Size-of[note 2] Alignment requirement(C11)	Right-to-left
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <= > >=	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional[note 3]	Right-to-left
14[note 4]	= += -= *= /= %= <<= >>= &= ^= |=	Simple assignment Assignment by sum and difference Assignment by product, quotient, and remainder Assignment by bitwise left shift and right shift Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

In conclusion …

- In this session, we have covered:
  - C datatypes
  - l-values, r-values
  - C operators
    - Arithmentc
    - Relational
    - Bitwise
    - Shortcut

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.