Numbers and Types

• Numbers in C programs have types.
• Numbers with a decimal point are type `double`, e.g. 3.14159, -34.56, 42.0
• C also lets write numbers in scientific notation:
  \(2.4\times10^5\) \(\Rightarrow\) \(2.4 \times 10^5 \Rightarrow 240000.0\)
  Numbers in scientific notation are also type `double`
• Numbers without decimal point or exponent are type `int`, e.g. 42, 0, -24
Variables

- Variables are used to store a value.
- The value a variable holds may change over its lifetime.
- At any point in time a variable stores one value.
- C variables have a type.
Variable Types

To start with, we will only consider 2 types of variables:

- **int** for integer values, e.g.: 42, -1
- **double** for decimal numbers 3.14159, 2.71828
Declaring, Initialising and Using Variables

// Declare
int answer;

// Initialise
answer = 42;

// Use
printf("%d", answer);

- **Declare** The first time a variable is mentioned, we need to specify its type. This tells C it needs to set aside a chunk of memory (RAM) for the variable.

- **Initialise** Before using a variable we need to assign it a value. Before we do this, the memory location just contains whatever 'garbage' values that happened to be there before.
Variable Names (and other Identifiers)

- Variable names can be made up of letters, digits, and underscores.
- Beware variable names are case sensitive, e.g., `hello` and `hEllo` are different names.
- Beware certain words can’t be used as variable names: `if`, `while`, `return`, `int`, `double`.
- These keywords have special meanings in C programs. You’ll learn what many of them are as we go on.
Variable Names (and other Identifiers)

In this course we must also follow the Style Guide

- They must be valid C identifiers AND
- They must begin with a lower case letter
- They must not use any underscore characters
- Identifier names should be meaningful
- Single letter variables should be avoided unless they are loop counters or numbers from a maths formula
- Where identifier names are composed of several words, the first word should be in lower case and the first letter of each subsequent word should be in upper case
  - eg myFirstVariable
Output using `printf()`

- **No variable**

```c
printf("Hello World\n");
```

- **A single variable** - Use conversion specifier `%d` to print an `int` (integer) value

```c
int answer = 42;
printf("num is %d\n", answer);
```

- **More than one variable**

```c
int j=5;
int k=10;
printf("j is %d and k is %d\n", j,k);
```
• **Printing doubles** - Use conversion specifier `%lf` to print a double (floating point) value

```c
double x;
x = 1.34432;
printf("x is %lf\n", x);
```

In addition, most conversion specifiers have options for finer control, e.g., `%3lf` instructs `printf` to use a precision of three.
Arithmetic Operators

- C supports the usual maths operations: $+ - \ast /$
- **Precedence** is as you would expect from high school, e.g.:
  \[ a + b \ast c + d/e \implies a + (b \ast c) + (d/e) \]
- **Associativity** (grouping) is as you would expect from high school, e.g.:
  \[ a - b - c - d \implies ((a - b) - c) - d \]
- Use brackets if in doubt about order arithmetic will be evaluated.
- Beware division may not do what you expect.
Division in C

- C division does what you expect if either operand is a `double`
  If either operand is a `double` the result is a `double`.
  \[ 2.6 / 2 \Rightarrow 1.3 \]
- C division may not do what you expect if both arguments are `int`.
- The result of dividing 2 integers in C is an `int`.
- The fractional part is discarded (not rounded!).
  \[ 5 / 3 \Rightarrow 1 \text{ (not 2)} \]
- C also has the `%` operator (integers only).
  Computes the modulo (remainder after division)
  \[ 14 \% 3 \Rightarrow 2 \]
Discuss with your neighbour

What are the values of the following expressions?

6 * 7 - 8 * 9/10
2 * 3 * 4 + 5 * 6
5 * 6 / 4
3 / 2
1.0 / 2.0
1 / 2.0
Input using scanf()

scanf uses a format string like printf.

- Use %d to read an int (integer) value

```c
int answer;
printf("Enter the answer: ");
scanf("%d", &answer);
```

- Use %lf to read a double (floating point) value

```c
double e;
printf("Enter e: ");
scanf("%lf", &e);
```
To start with, we will only consider 2 types of variables:

- typically 4 bytes used to store an `int` variable
- 4 bytes $\rightarrow$ 32 bits $\rightarrow$ $2^{32}$ possible values (bit patterns)
- only $2^{32}$ integers can be represented - which ones?
- $-2^{31}$ to $2^{31} - 1$
  i.e. -2,147,483,648 to +2,147,483,647
Real Representation

• commonly 8 bytes used to store a **double** variable
• 8 bytes → 64 bits → $2^{64}$ possible values (bit patterns)
• 64-bits gives huge number of patterns but infinite number of reals
Integer Overflow/Underflow

- storing a value in an int outside the range that can be represented is illegal
- unexpected behaviour from most C implementations e.g. the sum of 2 large positive integers is negative
- may cause programs to halt, or not to terminate
- can creates security holes
- bits used for int can be different on other platforms
- C on tiny embedded CPU in washing machine may use 16 bits
  \(-2^{15} \text{ to } 2^{15} - 1\) i.e. -32,768 to +32,767
- we’ll show later how to handle this, for now assume 32 bit ints
It can be useful to give constants (numbers) a name.

One method is \texttt{#define} statement e.g.
\begin{verbatim}
#define SPEED_OF_LIGHT 299792458.0
#define MIN_PER_HOUR 60
\end{verbatim}

It often makes your program more readable.

It can make your program easier to update particularly if the constant appears in many places.

\texttt{#define} statements go at the top of your program after \texttt{#include} statements.

For good style, \texttt{#define} names should be all capital letters + underscore.
Mathematical functions

- Mathematical functions not part of standard library, essentially because tiny CPUs may not support them.
- A library of mathematical functions is available including:
  - `sqrt()`
  - `sin()`, `cos()`, `tan()`
  - `log()`
  - `exp()`
- Above functions take a **double** as argument (input) and return a **double** (output).
- For example: `double result = sqrt(1.5);`
Mathematical functions

To use the mathematical functions:

• Extra include line needed at top of program:
  ```
  #include <math.h>
  ```

• dcc includes maths library by default
  most compilers need extra option:
  gcc needs `-lm` e.g.:

  ```
  gcc -Werror -Wall -O -o circle circle.c -lm
  ```
Linux Command: cp

- Linux Command `cp`: copies files and directories.
- `cp sourceFile destination`
- If the destination is an existing file, the file is overwritten
- if the destination is an existing directory
  the file is copied into the directory
- To copy a directory use `cp -r sourceDir destination`
Linux Command: `mv`

- **Linux Command** `mv` moves or renames a file.
- `mv source destination`
- If the destination is an existing file, the file is overwritten.
- If the destination is an existing directory, the file is moved into the directory.
Linux Command: rm

- Linux Command `rm` removes a file.
- Usually no undo or recycle bin - be careful & have backups
- `rm filename`
- `rm -r directoryName`
  - This will delete a whole directory.
  - Be extra careful with this command