• 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 (except quantum computers!)
• C variables have a type

We’ll only use 2 types of variable for the next few weeks:

• **int** for integer values, e.g.: 42, -1
• **double** for decimal numbers 3.14159, 2.71828
• typically 4 bytes used to store an **int** variable
• 4 bytes → 32 bits → $2^{32}$ 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
• Why are limits assymetric?
• zero needs a pattern (all zeros)
• can print bit values see:
  https://cgi.cse.unsw.edu.au/~cs1511/code/C_basics/print_bits_of_int.c
• More later and in COMP1521
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 create security holes
- bits used for `int` can be different on other platforms
- C on tiny embedded CPU in washing machine may use 16 bits $-2^{15}$ to $2^{15} - 1$ i.e. -32,768 to +32767
- we’ll show later how to handle this, for now assume 32 bit `ints`
- also arbitrary precision libraries available for C manipulate integers of any size (memory permitting)
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
- reals in (absolute) range $10^{-308}$ to $10^{308}$ can be approximated
- approximation errors can accumulate
- More later and in COMP1521
Variables

- **Declare** The first time a variable is mentioned, we need to specify its type.
- **Initialise** Before using a variable we need to assign it a value.

```c
// Declare
int answer;

// Initialise
answer = 42;

// Use
printf("%d", num);
```
• Variable names can be made up of letters, digits, and underscores.
• Use a lower case letter to start your variable names.
• Beware variable names are case sensitive, e.g., `hello` and `hEllo` are different names.
• Beware certain words can’t be used as variable names: e.g., `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.
Output using printf()

- No variables:
  ```c
  printf("Hello World\n");
  ```

- A single variable:
  ```c
  int num = 5;
  printf("num is %d\n", num);
  ```

- More than one variable:
  ```c
  int j = 5;
  int k = 17;
  printf("j is %d and k is %d\n", j, k);
  ```
Using values in printf()

- Use `%d` to print an `int` (integer) value

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

- Use `%lf` or `%g` to print a `double` (floating point) value

```c
double pi;
pi = 3.14159265359;
printf("pi is %lf\n", pi);
```
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);
  ```

- Use only */%d* and */%lf* format strings with scanf
- Read only 1 value at a time with scanf
- scanf can be used in other ways - don’t do it
- We’ll show you better ways to do other input
Numbers in 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:
2.4e5 \(\Rightarrow\) 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

Numbers in programs are often called constants
(unlike variables they don’t change)
• It can be useful to give constants (numbers) a name.
• It often makes your program more readable.
• It can make your program easier to update particularly if the constant appears in many places
• One method is `#define` statement e.g.
  `#define SPEED_OF_LIGHT 299792458.0`
• `#define` statements go at the top of your program after `#include` statements
• `#define` names should be all capital letters + underscore
• 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`.
  \[ \frac{2.6}{2} \Rightarrow 1.3 \text{ (not 2!)} \]
- C division may not do what you expect if both arguments are integers.
- The result of dividing 2 integers in C is an integer.
- The fractional part is discarded (not rounded!).
  \[ \frac{5}{3} \Rightarrow 1 \text{ (not 2!)} \]
- C also has the `%` operator (integers only). Computes the modulo (remainder after division).
  \[ 14 \% 3 \Rightarrow 2 \]
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(), log(), exp()
  Above functions take a **double** as argument and return a **double**
• Functions covered fully later in course
• Extra include line needed at top of program:
  ```
  #include <math.h>
  ```
  (explained later in course)
• dcc includes maths library by default
  most compilers need extra option:
  gcc needs **-lm** e.g.:
  ```
  gcc -o heron heron.c -lm
  ```
• printf & scanf are functions
• scanf returns a value returns number of items read
• Use this value to determine if scanf successfully read number.
• scanf could fail e.g. if the user enters letters
• OK for now to assume scanf succeeds
• Good programmers always check
• 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` 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