

Week 1 Lecture 2

Variables and Constants

So far

- Welcome and introductions
- Getting started

Today

- Memory
 - Data (variables + constants)
 - Expressions

A brief recap

Out first program

```
#include <stdio.h>

int main(void) {
    printf("Hello, world\n");
    return 0;
}
```

`#include <stdio.h>` -> bring in the standard IO library

`int main(void) {` -> start the main function

`printf("...");` -> print text (string) to the terminal

`Return 0;` -> return out of the main function

How do computers store data?

How do computers store data?

- Computers are electrical
- Electricity is either flowing, or not
- We store electrical charge (or lack thereof) in a large number of on-off switches
- We call these switches "bits" (the smallest possible unit)
- 0 or 1

How do computers store data?

- Alone, a single bit can't do much...
- What if we group them together?

Activity - Spell your name

Letter	Binary Sequence	Letter	Binary Sequence
A	00000	B	00001
C	00010	D	00011
E	00100	F	00101
G	00110	H	00111
I	01000	J	01001
K	01010	L	01011
M	01100	N	01101
O	01110	P	01111
Q	10000	R	10001
S	10010	T	10011
U	10100	V	10101
W	10110	X	10111
Y	11000	Z	11001

01001 00000 01010 00100

By agreeing on what a sequence of 0 s and 1 s means, we can store and retrieve data!

**Where can we put all these bees
bits?**

Different types of memory

- RAM (memory)
 - this is mostly what we care about in 1511
- HDD/SDD (persistant data)
 - Tapes?

**Different types of
memory too**

stack

Heap

**global/
statics**

code

Whiteboard

Show off the grid

Q: How do we use memory in our programs?

A: Variables

- A label for a piece of memory
- "variable" because the value in memory can change
- A certain number of bits required to store that data type
- Stores a specific type of data

To make a variable, you need:

- It's type
- It's name

Some data types

- `int` -> an integer, a whole number (1, -5, 100)
- `char` -> a single character ('a', 'V', ' ')
- `double` -> a floating point number (3.14159)

Each type has different memory requirements

- `int` -> 32 bits in C, 4 bytes
- `char` -> 8 bits, 1 byte
- `double` -> 64 bits, 8bytes

And therefore limits...

- `int` -> `-2,147,483,648` to `2,147,483,647`
- `char` -> `-128` to `127` chars
- `double` -> `-2,147,483,648` to `+2,147,483,647`

Variable names

- Surprisingly important...
- Should describe what it's storing
- You can pick whatever you want (mostly)
- in C, always use lowercase letters
 - `name` is different to `nAme`
- separate words by underscores

`first_name`

Naming variables is an art...
COMP1531 goes into more detail

int

- A whole number, with no fractions or decimals
- Most commonly uses 32 bits (which is also 4 bytes)
- This gives us exactly 2 different possible values
- Exact ranges from -2147483648 to 2147483647

char

- A char type is used to store a single character
- chars have to be wrapped in single quotes, like: 'a'
- Each char is associated with an integer
- We can convert chars to ints, and back
- 'a' and 'A' are different characters!
- chars are just ints under the hood...

Show ascii command in bash

double

- A double-sized floating point number
- A decimal value - "floating point" means the point can be anywhere in the number
- Eg: 10.567 or 105.67 (the points are in different places in the same digits)
- It's called "double" because it's usually 64 bits, hence the double size of our integers (or 8 bytes)

Variables syntax

To declare a variable, you use:

```
<type> <name>;
```

- `int age;`
- `char first_initial;`
- `double pi`

Demo

```
#include <stdio.h>

int main(void) {
    // declare an int.
    int my_age;

    // assign a value to the int.
    my_age = 25;

    // whoops, I wish... let's update
    my_age = 28;

    return 0;
}
```

**OK we can store some data... so
what?**

Printing variables using `printf`

- We can print variables to our terminal!
- We describe the format of how we want text printed, then the actual values.
- To print out a variable value, we use format specifiers with `printf`

- The format specifier (%) indicates WHERE a value will output within the format string.

```
int my_age = 13;  
printf("I am %d years!",  
my_age);
```

After the comma, you put the name of the variable you want to write

**The format specifier must match
the data types passed**

- `%c` for chars
- `%d` for ints “decimal integer”
- `%lf` for “long floating point number” (a double)
- `printf` needs to know what type it should expect in what order, because...

You can have multiple variables:

```
int diameter = 5;  
double pi = 3.141;  
printf("The diameter is %d, pi is  
%lf", diameter, pi);
```

Demo

Break - lecture feedback

COMP1511 Lecture Feedback



writing 🤝 reading

`scanf`

- Scan Formatted
- Reads input from the user in the same format as printf
- Format specifiers `%d`, `%lf`, `%c` are used in the same way
- The `&` symbol tells scanf where to store the data (more details later in term)

```
#include <stdio.h>

int input;
printf("Please enter your age:
");
scanf("%d", &input);
```

Demo

A bit more on `scanf`

```
scanf("%d", &my_int);
```

```
scanf("%c", &my_char);
```

- scanning an int ignores whitespace
- scanning a char does not ignore whitespace
- We can ignore leading whitespace with chars:

- ```
scanf(" %c", &character);
```

## Constants





- A value that will never change
- More efficient to store a constant (less memory)
- Different syntax
- We use UPPERCASE to signify it's a constant

```
#define <NAME> <value>
```

```
#define PI 3.1415
```

## Using variables in expressions

A lot of arithmetic operations will look very familiar in C

- adding 
- subtracting 
- multiplying 
- dividing 
- These will happen in their normal mathematical order
- We can also use brackets to force precedence

```
int age = 28;
int current_year = 2023;
int year_born = current_year -
age;

printf("You were born in %d",
year_born);
```

`chars` are just `ints` playing dress-up

```
char letter = 'b';
letter = letter + 1;
printf("%c\n", letter);
```

^^ Will print 'c'

## **Don't forget your limits**

If we add two large ints together, we might go over the maximum value, which will actually roll around to the minimum value and possibly end up negative

- (Check out Ariane 5 explosion), a simple error like this caused a rather large problem:  
<https://www.bbc.com/future/article/20150505-the-numbers-that-lead-to-disaster>)

- Boeing 787 had to be rebooted every 248 days ( $2^{31}$  -hundredths of a second)
- <https://www.engadget.com/2015-05-01-boeing-787-dreamliner-software-bug.html>
- In a less destructive example, the video Gangnam Style on YouTube maxed out the views counter :  
<https://www.bbc.com/news/world-asia-30288542>

## **Doubles :(**

- No such thing as infinite precision
- We can't precisely encode a simple number like  $\frac{1}{3}$
- If we divide 1.0 by 3.0, we'll get an approximation of  $\frac{1}{3}$
- The effect of approximation can compound the more you use them



## **Remember that C thinks in data types**

- If either numbers in the division are doubles, the result will be a double
- If both numbers are ints, the result will be an int, for example,  $3/2$  will not return 1.5, because ints are only whole numbers
- ints will always drop whatever fraction exists, they won't round nicely, so  $5/3$  will result in 1
- There's ways around all of this...