CS2100 - L05 - Arrays

Week 3

5.1 - Arrays

- Declaration and Initialization
- Accessing Individual Items
- Arrays as Function Parameters

5.2 - Characters

- Declaration and Character Constants
- Conversion to Other Data Types

5.3 - Strings (special Character Arrays)

- Behaviour
- String Library

5.1 - Arrays

Syntax

SYNTAX

Declaration:

```
datatype identifier[ size ];
```

Declaration with initialization:

```
datatype identifier[ size ] = { init_list };
```

- Example:

```
int myArray[3];
```

- Memory Snapshot:

myArray	?????	4026
	?????	4027
	?????	4028

```
int myArray[3] = { 1, 2, 3 };
```

myArray	1	4026
	2	4027
	3	4028

Initialization

- Initialization list can be shorter than array size
  - The rest of the items will simply get a zero value

```
int myArray[3] = { 1 };
```

myArray	1
	0
	0

- Use `{0}` to initialize all array items to zero

Pointers and arrays

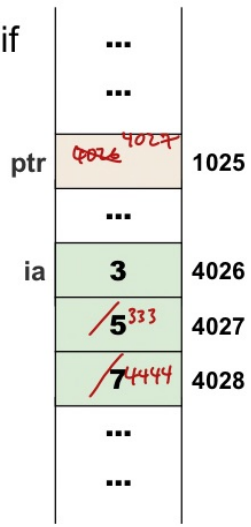
- Array name by itself is the same as **the address of the 0th array element**
- Normal pointer can work just like an array if you set it up properly

```
int ia[3] = {3, 5, 7};
int *ptr;

ptr = ia;
ptr[1] = 333;


ptr = &(ia[1]);
ptr[1] = 4444;
```

Don't Panic! Just apply what you have learnt



Arrays as function parameters

pass-by address / pointer

- Array can be passed into function as parameter
  - but **not as a return data type**

```
void printLessThan( int a[], int size, int criteria )
```

Array parameter. Note that the size is not required here.

General rule: Function working on an array should have a parameter to indicate the number of elements in the array

5.2 - Characters

- Recall from **data representation**:
  - Character is encoded as a small integer
  - In C, each character variable occupies **1 byte**

- **Character data type** stores:
  - Printable Characters:
    - **Letters:** 'A', 'B', … 'Z', 'a', 'b', …., 'z'
    - **Digits:** '0', '1', '2', …., '9'
    - **Symbols:** '~', '!', '*', '$', '(', ' ' …
  - Unprintable Characters:
    - **Control sequence:** '\n', '\t', ….

- Character can be manipulated like a number:

```c
char c = 'a';

c = c + 3; // c is now 'd'

c = c - 'a' + 'A'; //What do you think c is now?
```

5.3 – Strings (Special Character Arrays)

- C provides **string** for this purpose
  - Use **character array** to store multiple characters
  - Add a **special character** (**'\0'**) to indicate it is a string
    - known as **string terminator**
    - '\0' has the ASCII value of 0
- C string is basically a **special case of character array**
  - A source of common confusion ☹

- String constants are surrounded by double quote (")
  - "Hello!"
  - "Hello There!"

'\0' implicit

```
char a[6] = { 'H', 'e', 'l', 'l', 'o', '!' };
```

a

H	e	l	l	o	!

```
char b[7] = "Hello!";
```

string

b

H	e	l	l	o	!	\0

Careful! Remember to count the terminator when declaring the size of a string

Benefits

- String can be read / written as a whole using `printf()` and `scanf()`
  - Use "`%s`" as placeholder
  - Make sure you are using a valid string!

- There are a good range of predefined string functions:
  - Need to include the library `<string.h>`
  - A few commonly used functions are given next for your reference

String library - <string.h>

- strlen(): length of string
- strcpy(): copy string
- strcat(): concatenate strings
- strcmp(): compare two strings