ENGG1003 - Friday Week 5

Arrays and Functions: Together at Last!

Does anyone even read the title page?

Also: Maybe Strings & ASCII Codes

Brenton Schulz

University of Newcastle

March 29, 2019

The Story So Far

- Course summary:
  - Flow control
    - `if()`
    - `while()`
    - `for()`
    - `switch()`
  - Variables and data types
  - Functions
  - Arrays
- Today: Arrays and functions together
  - Subtext: Pointers
- Today (maybe): Strings
- Tuesday: File input-output (I/O)

Programming Assignment And Quiz

- The programming assignment will use everything from the previous slide
- The quiz can include everything up to, and including, the Week 5 Tuesday lecture
  - Held in Friday 9-10am lecture
    - 40 mins: 9:10am - 9:50m
  - It will be hand written
    - Yes, *real paper*
  - Mix of:
    - Multiple choice
    - Code reading & analysis
    - Short code writing (1-3 lines)
  - You will **not** be asked to write out a whole program by hand

Arrays and Functions

- On Tuesday:
  - Studied arrays
  - Studied functions
  - Didn't mix them

Arrays and Functions

- On Tuesday:
  - Studied arrays
  - Studied functions
  - Didn't mix them
- There are two ways to pass arrays to functions:

Arrays and Functions

- On Tuesday:
  - Studied arrays
  - Studied functions
  - Didn't mix them
- There are two ways to pass arrays to functions:
  - Pass an array element, eg:

```
1 int function(int x);
2 // ...
3 int array[12];
4 // ...
5 function(array[6]);
```

Arrays and Functions

- On Tuesday:
  - Studied arrays
  - Studied functions
  - Didn't mix them
- There are two ways to pass arrays to functions:
  - Pass an array element, eg:

```
1 int function(int x);
2 // ...
3 int array[12];
4 // ...
5 function(array[6]);
```

  - Give a function a *pointer* to an array
    - Ok, lets break this one down a bit...

Arrays and Functions

- Firstly: why don't we pass a whole array?

Arrays and Functions

- Firstly: why don't we pass a whole array?
  - Arrays can be *huge*
  - Passing a whole array *copies* everything
  - This is a bad idea so C doesn't support it
  - (Advanced) Arguments are put to the *stack*
    - Google stack Vs heap memory allocation for more information. This is beyond ENGG1003.

Arrays and Functions

- Firstly: why don't we pass a whole array?
  - Arrays can be *huge*
  - Passing a whole array *copies* everything
  - This is a bad idea so C doesn't support it
  - (Advanced) Arguments are put to the *stack*
    - Google stack Vs heap memory allocation for more information. This is beyond ENGG1003.
- Instead, C passes a *pointer*
  - This is the *memory address* of the array's start

Arrays and Functions

- Firstly: why don't we pass a whole array?
  - Arrays can be *huge*
  - Passing a whole array *copies* everything
  - This is a bad idea so C doesn't support it
  - (Advanced) Arguments are put to the *stack*
    - Google stack Vs heap memory allocation for more information. This is beyond ENGG1003.
- Instead, C passes a *pointer*
  - This is the *memory address* of the array's start
  - In C, `name` is equivalent to `&name[0]`

Arrays in Memory

- Review: When we declare an array, eg,

```
1 int x[20];
```

  the compiler allocates 20*sizeof(int) = 80 bytes to store it
- The *memory address* of `x[0]` is some seemingly random number, `p`
- `p` is a *byte* address
- Other elements are stored in sequential memory addresses:
  - The address of `x[1]` is `p + 4`
  - The address of `x[i]` is `p + i*4`

Arrays in Memory

- Therefore, to access a given element, `i`, of an array all we need is:
  - A pointer, `p` to the first element
  - Knowledge of the arrays *data type*
    - Specifically, the type's *size*
  - The calculation result of `p + i*size`

Arrays in Memory

- Therefore, to access a given element, `i`, of an array all we need is:
  - A pointer, `p` to the first element
  - Knowledge of the arrays *data type*
    - Specifically, the type's *size*
  - The calculation result of `p + i*size`
- So that's what we do with functions:
  - The function argument is a *pointer* to a *data type*

Arrays in Memory

- Therefore, to access a given element, `i`, of an array all we need is:
  - A pointer, `p` to the first element
  - Knowledge of the arrays *data type*
    - Specifically, the type's *size*
  - The calculation result of `p + i*size`
- So that's what we do with functions:
  - The function argument is a *pointer* to a *data type*
- C syntax:

```
1 return_type function_name(data_type *varName);
```

- Key syntax element: the `*` character

Arrays in Memory

- Therefore, to access a given element, `i`, of an array all we need is:
  - A pointer, `p` to the first element
  - Knowledge of the arrays *data type*
    - Specifically, the type's *size*
  - The calculation result of `p + i*size`
- So that's what we do with functions:
  - The function argument is a *pointer* to a *data type*
- C syntax:

```
1 return_type function_name(data_type *varName);
```

- Key syntax element: the `*` character
- Inside the function use `var[i]` syntax

Key Points

- Because arrays are passed via a pointer the function gets *the actual array*
- Modifying the array in the function modifies the original variable
- You don't *need* a return value
  - In a technically incorrect way: all the array's elements are "returned"

Example

- Write a function which zeros the first N elements of an array of `ints`
  - Function prototype:

Example

- Write a function which zeros the first N elements of an array of `ints`
  - Function prototype:
    - `void zero(int *x, int N);`

Example

- Write a function which zeros the first N elements of an array of `ints`
  - Function prototype:
    - `void zero(int *x, int N);`
  - The value of `N` is needed because C won't tell you how long an array is *within the context of the function*
    - (Advanced) `sizeof(x)` will just be the size of the pointer - 4, or 8 bytes

Example

- Function definition:

```
1 // Zeros first N elements of x
2 void zero(int *x, int N) {
3   int i; // Array index loop counter
4   for(i = 0; i < N; i++)
5     x[i] = 0; // Use array syntax
6   return; // Optional
7 }
```

Other Examples

- Lets write and test these live...
- Write a function which:
  - Returns the sum of an array of length N
  - Returns the maximum value in an array of length N
  - Fills an array with integers between two given numbers `min` and `max`
    - Prototype:

```
void countArray(int *x,
                int min, int max);
```

    - eg: countArray(x, 10, 15) sets:

```
x[] = {10, 11, 12, 13, 14, 15}
```