Lecture 5: Arrays, Strings, Command Line Arguments

COMP26020 Part 1 (C) Lecture Notes

Pierre Olivier

These notes summarise the important points mentioned in the lectures. They are supposed to be a help for revising and not a way to avoid attending the live lectures and watching the videos. In other words, live lectures and videos may include examinable content that is not present in these notes.

The slides for this lecture are available here:
https://olivierpierre.github.io/comp26020-lectures/05-c-arrays-strings-cmdline.

Videos and recordings of live sessions can be found on the video portal: https://video.manchester.ac.uk/lectures.

Here we consider arrays, strings and command line arguments in C.

Arrays

Arrays are declared with a statement starting with the type, followed by the name of the array variable, then between brackets the size of the array, here 4 elements:

```
int array[4];
```

We can write something in an array slot with the affectation symbol, equals, and the index to write between bracket. For example here we write 10 in the first slot and 20 in the second one:

```
array[0] = 10;
array[1] = 20;
```

In C the indexing starts at 0, i.e. the first slot is index 0. We read a slot by indicating its index between brackets, for example here we read the second slot:

```
int x = array[1];
```

Something important to note is that arrays are stored contiguously in memory. Remember on x86-64 the size of `int` is 4 bytes. So in memory, for our array of 2 elements, we have 2*4 = 8 contiguous bytes in memory holding the array's data.

Multi-Dimensional Arrays

Arrays can be of multiple dimensions. To declare such an array we can use several pairs of square brackets. For example to declare a 3 by 2 array:

```
int my_2d_array[3][2];
```

- We also refer to particular slots with pairs of square brackets:

```
my_2d_array[0][0] = 0;
my_2d_array[0][1] = 1;
my_2d_array[1][0] = 2;
my_2d_array[1][1] = 3;
my_2d_array[2][0] = 4;
my_2d_array[2][1] = 5;
```

This is for a 2 dimensional array, but there can be more e.g. `int array[2][2][2]`. Arrays are still laid out contiguously in memory with the last dimension first, so we have `my_2d_array[0][0]`, followed by `my_2d_array[0][1]`, followed by `my_2d_array[1][0]`, and so on.

Arrays: Static Initialisation

Rather than initialising the array after its declaration, we can also initialise it with its declaration. This is called **static** initialisation. With it, it is possible to omit the size of the first dimension, as shown on this example:

```
int array[] = {1, 2, 3, 4, 5};
int array_2d[][2] = { {1, 2}, {3, 4}, {5, 6} };
```

Here we create and initialise a one dimensional array with 5 slots, and a two-dimensional array of size 3 by 2.

Character Arrays: Strings

In C a **string** is simply a character array that is terminated with a special character, `'\0'`, marking the end of the string. This character is used by many functions to know where a given string ends. For example when we call `printf` on a string, it prints on the console all the characters of the array until it encounters the termination character. We can declare a string like an array. It can be done statically, and we use the double quotes to indicate a string:

```
char string1[6] = "hello";
char string2[] = "hello";
```

The compiler will automatically include the termination character for these. However, be careful to account for it when setting the size of the array, even if `"hello"` only has 5 characters, we need a sixth for the anti slash 0. We can also fill an array after its declaration:

```
string3[0] = 'h';
string3[1] = 'e';
string3[2] = 'l';
string3[3] = 'l';
string3[4] = 'o';
string3[5] = '\0';  // Important here
```

When a string is created in such a way, the termination character needs to be added manually.

Command Line Arguments

Command line arguments are strings passed on the command line when we invoke a program, separated with spaces. For example here we execute the program `gcc` with 3 arguments: `test.c`, `-o`, and `program`:

```
$ gcc test.c -o program
```

Command Line Arguments: `argc` and `argv`

The `main` function can be declared with 2 parameters:

- `argc`, an integer indicating the number of command line arguments.
- `argv`, a bi-dimensional character array, in other words an array of strings, that contains the arguments.

Here is an example of a simple program simply printing the number of command line arguments and the first 3 of these arguments:

```
int main(int argc, char **argv) { // 'char ** argv' means 'char argv[][]'
    printf("Number of command line arguments: %d\n", argc);
    // This is a bit dangerous...
    printf("argument 0: %s\n", argv[0]);
    printf("argument 1: %s\n", argv[1]);
    printf("argument 2: %s\n", argv[2]);
    return 0;
}
```

Don't be scared by the `**` before `argv`, it is just another way to declare a 2 dimensional character array. This program is not ideal: what happens if this code is called with less than 3 arguments? We get a memory error The program trying to print command line arguments that do not exist, leaking weird values from its memory. That's quite dangerous, there could be sensitive things like a password in there. We'll remediate this in the next lecture regarding control flow.

String to Integer Conversion

Command line arguments are strings, so if numbers are passed they need to be converted them. This can be done with the `atoi` function. To use it, `stdlib.h` needs to be included:

```c
#include <stdio.h>
#include <stdlib.h> // needed for atoi
/* Sum the two integers passed as command line integers */
int main(int argc, char **argv) {
    int a, b;
    // Once again, dangerous!
    a = atoi(argv[1]);
    b = atoi(argv[2]);
    printf("%d + %d = %d\n", a, b, a+b);
    return 0;
}
```

To convert a string to a `double` there is `atof`.