****

**Bubble Sort Algorithm**

By

Peter Kinsella

This Report is submitted in partial fulfilment of the requirements of the Honours Degree in Electrical and Electronic Engineering (DT021) of the Dublin Institute of Technology

November 10th, 2018

Supervisor: Dr Ray Lynch

Contents

[Background 2](#_Toc529793738)

[Functioning Code 3](#_Toc529793739)

[Function of Code 4](#_Toc529793740)

[Conclusion 5](#_Toc529793741)

Figures

[Figure 1 C++ Implementation 3](#_Toc529793241)

[Figure 2 Assembly Implementation 3](#_Toc529793242)

[Figure 3 Unsorted and sorted array of characters 3](#_Toc529793243)

[Figure 4 Visual representation of Bubble sort 4](#_Toc529793244)

# Background

In this lab a C++ program was linked with an Assembly language module to sort an array of characters using their ASCII code values. By linking the C++ code with the Assembly module, the ability to externally run a sort function was now available.

When a function is called in C++ the corresponding assembly file will be created and filled once the link is created. With regular C++ linking the name of the function within the assembly file, in this case “sort”, will be followed by a series of characters and letters. To clear this up, the use of a function called, extern “C”, will give the assembly file the normal C decoration. This makes the function easier to match in the separate module.

Now, creating a separate .asm file for sort will allow a person to code the workings of the function. This is similar to having a separate class within a C++ program but instead you are calling an external assembly file rather than an integrated class. This allows the user to be able to configure with the stack orientation and program the code in a deeper level.

All variables used in the C++ code are stored on the stack and take up 12 bytes of space. Normally it is only 4 but when generating a debug version it adds 8. There is also 192 bytes are allocated in the stack frame to support the edit and continue feature. So this 192 bytes plus however many variables are used add up to craete what is known as the stack frame. Here the esi register is used as the stack pointer to cycle through the different location in the stack frame. When the assembly subroutine is finished, the calling function cleans the variables off the stack to clear up more space for later use.

By linking assembly and C++, it is possible to manipulate the position of this pointer to allow the user to perform functions in assembly on a deeper level. There are also other registers where values from variable can be stored so that they may be used late for commands such as add, cmp, mov and jge.

# Functioning Code

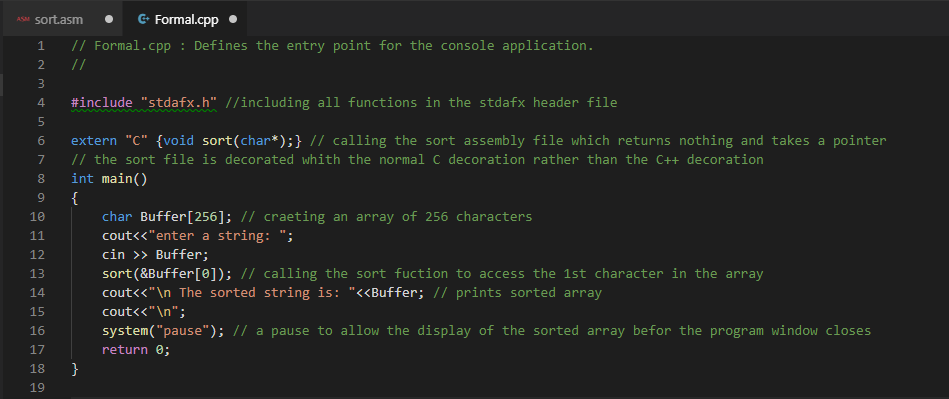


Figure C++ Implementation

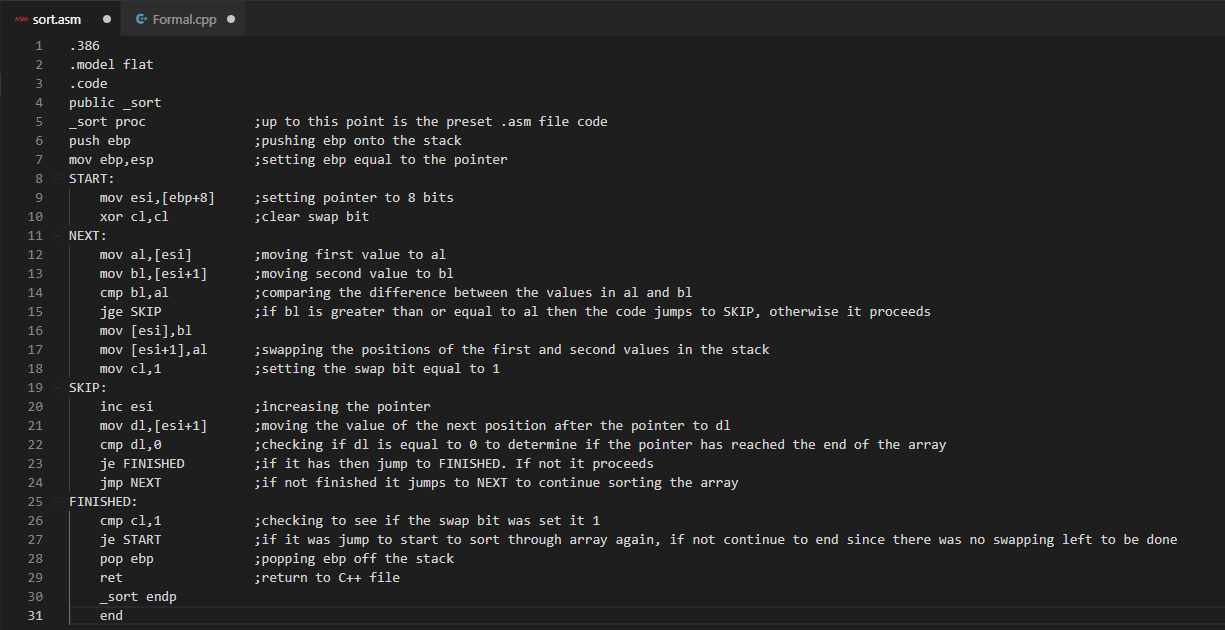


Figure Assembly Implementation



Figure Unsorted and sorted array of characters

# Function of Code

The code above is the implementation of a sorting algorithm which is designed to sort an array of 256 characters by their ASCII code values. Within the C++ file Formal.cpp, the external assembly file sort.asm is called to sort the array.

When the function is called the array of characters is passed to the function and the pointer of the internal stack is set to point at the first value in the array. From this point, the value that the pointer is pointing at is stored in the al register and the value that is one ahead of the pointer is stored in the bl register. The two values are compared and if the value in the bl register is greater than or equal to that of the value in the al register then there is no change in values positions and the code jumps to the SKIP section of the code, thus bypassing the code used to swap the values. If not, then the values in the two registers change position in the stack and the swap register, cl, is set to 1.

After either of these two option occur the pointer is increased in value by one to point at the next position in the stack. The value just after the pointer is stored in the dl register. If this value is 0 that means that the pointer has reached the last value in the stack. If it is not 0 then it will return to the NEXT section as seen in the sort.asm code and proceed with comparing the next two values. If the value in the dl register is equal to 0 the code jumps the FINISHED section.

From here the code checks to see if the swap register was set to 1. If so then the code jumps back to the Start section where the pointer is reset and the swap register is set back to 0.

Once the program can go through a full cycle through the array and not have the swap register set to 1, only then has the array been sorted fully and return to the C++ program file. There the new sorted array is printed to the screen where the outcome of the sort function can be viewed. A visual representation of the sorting process can be seen in figure 4.

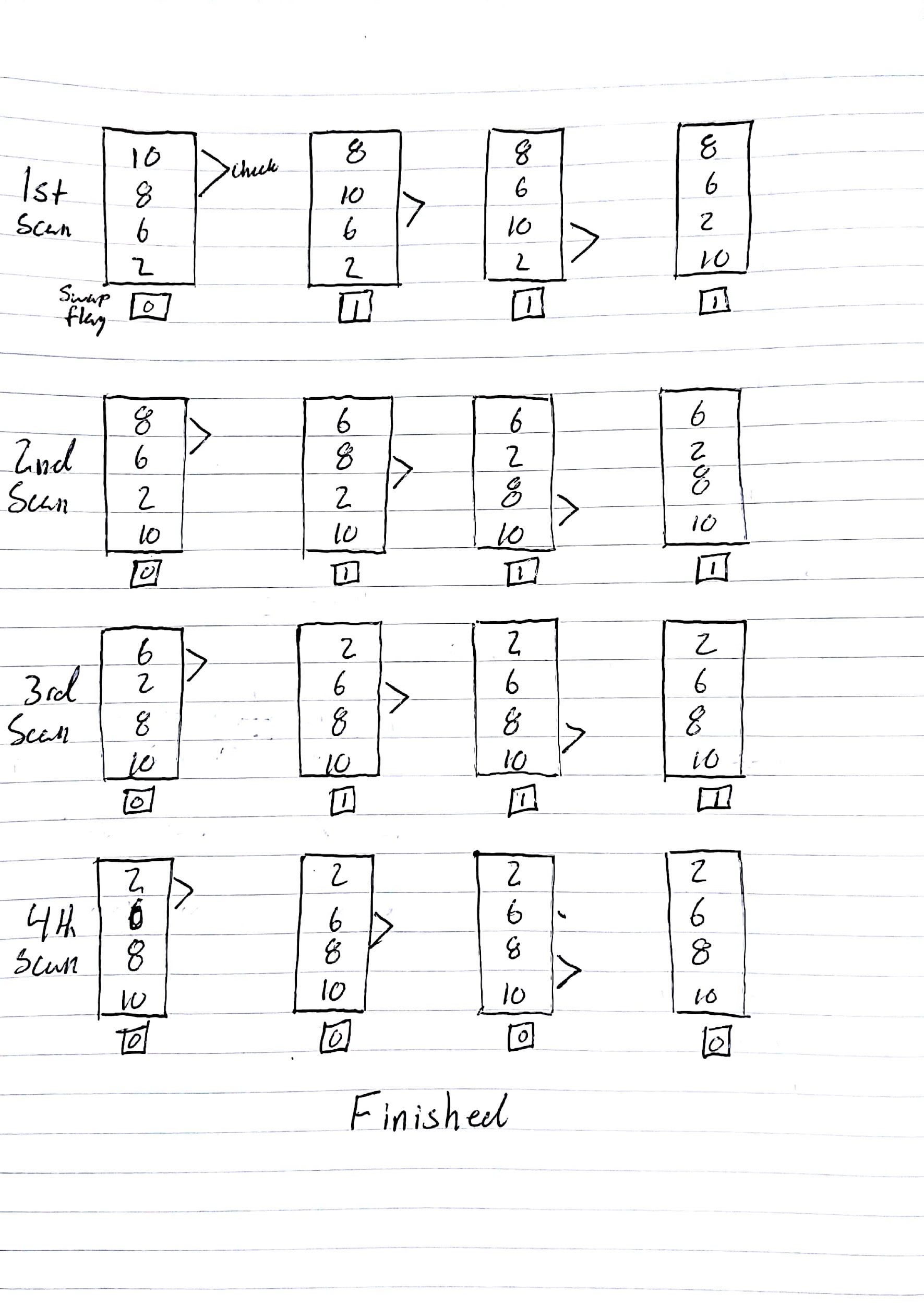


Figure Visual representation of Bubble sort

# Conclusion

In conclusion it can be drawn that with the implementation of the provided assembly code, one can sort up to 256 characters correctly based on their ASCII code values. With more time, it would have been nice to have the code sort the characters with case sensitivity. This means that A and a would be placed together in the final sort. Though, with the current implementation the code works as described.