CS211 – Operating Systems and Programming

Lab 2: Functions

The goal of this week's lab is to get you started writing functions in C. We will practice using the `if` and `for` expressions by implementing a linear sort and a **Bubble Sort** algorithm.

The basic problem is: permute a given list of n integers $\{a_0, a_1, a_2, \ldots, a_{n-1}\}$ so that $a_0 \leqslant a_1 \leqslant a_2 \leqslant \cdots \leqslant a_{n-1}$. (*Remember*: C indexes its arrays from 0, so the list a of n elements is $a[0], a[1], \ldots, a[n-1]$).

The **linear sort** algorithm is one of the simplest (and slowest) algorithms for doing this.

The basic idea is

- Compare a_0 with each of $a_1, a_2, \ldots, a_{n-1}$, and if you find that a_0 is the larger, swap their values. When this is done, we should have that a_0 is the smallest element in the list.
- Now compare a_1 with each of $a_2, a_3, \ldots, a_{n-1}$, and swap their values if you find that a_1 not the smaller. When this is done, a_1 will contain the second smallest element in the list. (Note: this uses the `Swap()` function from Week 4).
- Repeat the process until we have compared a_{n-2} with a_{n-1}.
- Now the list should be sorted.

Linear Sort

```
FOR i = 0 to n − 1
    FOR j = i + 1 to n
      IF a_j < a_i THEN
        Swap(a_i, a_j)
      FI
    END
END
```

This is implemented in the `LinearSort.c` program, which you can download from http://www.maths.nuigalway.ie/~niall/CS211.

1. Compile and run the `LinearSort.c`. Make sure you understand each line of code, and how the programme works.
2. Adjust the code so that, instead of running on a list of 8 elements between 0 and 30, it prompts the user for
   - The total number of elements in the list. You may assume that this is at most 30.
   - The maximum value that any element can have.
3. Modify the code so that a count is kept of the total number of "swaps" that are made. This should be reported when the program finishes running.

PTO

The **Bubble Sort** algorithm works as follows:

- Compare a_0 with a_1 and swap them if they are not in order. Now compare a_1 and a_2, and swap them if they are not in order. Repeat until we have compared a_{n-2} with a_{n-1}, and swapped them in necessary.
- When this is done, we should have that the largest element is in a_{n-1} (it will have "bubbled" to the top of the list).
- We continue by applying the process to the set $\{a_0, a_1, a_2, \ldots, a_{n-2}\}$. At the end of that the second largest element will be in a_{n-2}.
- Now apply the process to the set $\{a_0, a_1, a_2, \ldots, a_{n-3}\}$, etc.

Bubble Sort

```
FOR i = 0 to n − 1
    FOR j = 0 to (n − i − 1)
      IF a_{j+1} < a_j THEN
        Swap(a_j, a_{j+1})
      FI
    END
END
```

Write a C program that implements this and check if it is more efficient than the linear sort method: i.e., it requires less swapping of values.

. . . . . .

Functions

Write functions with the headers

`int LinearSort(int *a, int n);` and `int BubbleSort(int *a, int n);`

These take as their arguments an integer array, `a`, of length `n`, and sorts their entries in ascending order using, respectively, the Linear Sort and Bubble Sort algorithms. Both return the number of `Swap`s that was preformed.

Assignment

Submit a programme that contains the `LinearSort` and `BubbleSort` functions, along with a `main` function that calls them for *copies* the same list of random integers. The output from both sortings should be displayed, along with the number of swaps that each preformed, and a statement as to which required fewer swaps.

Submit your code to the `Lab 2` section on Blackboard, **no later than 5pm, Thursday 13 Feb**. The code **must** include

(i) comments at the start that include your name, ID number, and email address;

(ii) a short description of what the program does, written in your own words;

(iii) the name and email address of any person you collaborated with on the assignment, and a statement of what each of you contributed.