C++ course – Exercises Set 2

Wouter Verkerke (Jan 2023)

Exercise 2.1 – Sorting numbers and strings

The goal of this exercise is to learn how to use pointers and references with functions

You will write a program that sorts an array of 10 random integers using the bubble sort algorithm:

```
for (i=0; i<n-1; i++) {
  for (j=0; j<n-1-i; j++) {
    // if A[j+1] is greater than A[j], swap A[j] and A[j+1]
  }
}
```

Part 1 – Sorting arrays of integers

- Write a small main program that allocates an array of 10 integers and fill them with random values.
    - NB: *You can manually code a series of 10 'random' numbers in the initializer of the array. You do not need to develop code that calls a random number generator and fills the array.*
- Encapsulate the above bubble sort algorithm in a separate `sort()` function that takes the array of integers, and the length of that array as input arguments.
- Code the missing part of the above bubble sort algorithm, as explained in the comment in red.

  Do not include this code inside the `sort()` function, instead code it in a separate `order()` function, that you call from within `sort()`.

  Your `order()` function should take two function arguments, each of the type 'reference to integer'
    - NB: the `order()` function you're asked to develop here has a different functionality than the `swap()` routine described in the course material of Module 2 (which swaps the input arguments unconditionally)

- Add code to your main program that prints out the array after sorting to verify that all works correctly.

- Make a copy of your source code, which has a different name. In this copy, re-implement, the `order()` function using pointers as function arguments, rather than references. Adjust the code inside the `sort()` function accordingly.

  Do you like the pointer or the reference version better?

Part 2 – Sorting arrays of strings

- Make another copy of the program. Change the code in it such so that it allocates an array of then `const char*` elements. Initialize the pointers with 10 text literals (e.g. `"blah"`). Why do you need the elements of the array to be `const char*` rather than `char*` here?

- Adjust the function arguments of the `sort()` and `order()` functions so that they can work with arrays of `const char*`, instead of arrays of `int`.

  (*) Think about what the C++ data type of the arguments of `order()` function should be to make it completely analogous to the integer version? The easiest way to figure it out is to think of '`const char*`' as a fundamental type just link `int` and to proceed with a straightforward substitution of one with the other.

- Adjust the code of the `sort()` function. A particular point of attention is how to do the equivalent of 'a>b' for values of type `const char*`. What can't you just compare the pointer values to compare the strings?

  Instead, you can use the `strcmp(a,b)` function declared in `<string.h>` from the C++ standard library. This function returns an integer value greater or smaller than zero, depending on the lexical order of the two character arrays `a` and `b`. What can't you just compare the pointer values to compare the strings?

- As a last step, reimplement the `order()` function to be able to order the two elements of type `const char*`. Do you still need the reference in the function argument declaration? (Hint: think again about the point made in (*)).

Exercise 2.2 – Function overloading

The goal of this exercise is to understand the basics of function overloading.

- Implement the following overloaded `min()` functions

```
int min(int a,int b) ;
double min(double a,double b) ;
int min(int arr[], int len) ; // returns minimum of array
```

- Write a small program that tests your three implementations with matching examples.

- Now try call `min()` passing a `double` and an `int` as argument.
  Explain why this does not work.

- What possible solution can be implemented to make the above call work?
  Implement this solution.