C++ course – Exercises Set 1

Wouter Verkerke (Jan 2023)

Exercise 1.1 – “Hello World”

The goal of this exercise is to verify that computer and compiler setup are functioning correctly.

To verify that your setup runs fine, compile and run the “hello world” example

```
#include <iostream>
using namespace std ;
int main() {
   cout << “Hello World” << endl ;
   return 0 ;
}
```

Exercise 1.2 – Counting chars in a string

The goal of this exercise is to learn basic pointer skills. You will write a C++ program that analyzes the char acter content of a string that is given to you by the user of your program.

The program should display the number of uppercase and lowercase characters, the number of digits and the number of 'other' characters.

Approach

- First write a small program that reads an array of characters from the terminal up to a newline character and prints that same string again to the terminal.

  To do so, first allocate an array of characters, e.g. '`char buffer[256]`'

  Then fill the array with input that is read from the keyboard when you run the program with the following statement: `std::cin.getline(buffer, 256)`

- Add code that loops over the character array and inspects the contents one element at a time, accessed through a pointer.

  The point of this exercise is to practice the use of pointers, therefore you are ***not*** *allowed to use an integer index variable to loop over the character array.*

  Think about the following steps when solving this task:
  - How do you create a pointer and how do you make it point to the first character of the string?
  - How do you make the pointer point to the next character in the string?
  - How do you know that you're at the end of your string?

- Finally add code in your loop that determines if each given character is uppercase, lowercase, a digit or otherwise. Remember that literal values of characters use single quotes, e.g. `'Z'`.

  Follow these steps to solve this task
  - First write a line of code that determines if the inspected character of the array is the uppercase letter A.
  - Now modify this code to determine if the character is *any* upper case letter A-Z. Before you start this find out what the ASCII table looks like (e.g. Google ASCII table) and exploit the ordering of the ASCII characters. *Note that you can use all comparison operators that work on integers (<,>,>=,<=,!) on characters as well.*
  - Add similar code for lowercase characters, digits and 'other' characters and make your program print how many of each are in the string.

Exercise 1.3 – Joining strings

The goal of this exercise is to learn about string manipulation and dynamic memory allocation.

Examine the following program

```
#include <iostream>
using namespace std ;
char* join(const char* a, const char* b) ;
char* joinb(const char* a, const char* b) ;
int main() {

  cout << join("alpha","bet") << endl ;
  cout << joinb("duck","soup") << endl ;
  return 0 ;
}
```

The concrete task for this exercises is to write the missing `join()` and `joinb()` routines that will concatenate character strings. Here, function `joinb()` should insert a blank between the two strings, whereas function `join()` concatenates without a space in between.

Part 1 - Writing the join() function

- First, add an empty body for the function `join()` below the main function in your program code.

- Comment out the line using `joinb()` in the main program for now so you can test your main program and the (for now empty) `join()` code.

- Note that the `join()` function returns a pointer (to a character array). When you return a pointer in a function, it should point to a memory object you have actually allocated. Add code to `join()` that allocates a character array to which the return value pointer can point to.

- Think about whether you should use `new[]` to allocate this memory.

- What length should the allocated string be? Write your code such that exactly the right amount of memory is allocated. Do you need to allocate memory for the terminating NULL character?
  Hint: you can use the Standard Library routine `strlen()`*, declared in* `<string.h>` *to determine the length of a given character array*

- Use the Standard Library routine `strcat(char* s1, const char* s2)` to implement the concatenation part of `join()`. This function appends the

contents of array `s2` to that of `s1`. The function explicitly assumes a well-formed character arrays `s1` and `s2`, i.e. an array where the last element has value 0 to signal the end of the array.

Alternatively, use function `strcat_s(char* s1, int len_s1, const char* s2)`, which is more safe to use as the allocated length of s1 is passed as well to prevent accidental overruns. This function is only available in C++ version 2011, but certain compilers (like Microsoft Visual Studio) effectively prohibit you from use `strcat` – in which case you have no choice but to use this `_s` version)

You can also `strcat()` to append the contents of both `char* a` and `char* b` arrays that were passed as function arguments to your internal buffer.

However, you have to make sure that before the first use your internal buffer character array is an empty well-formed character array.

- Initialize the contents your buffer character array to explicitly form a zero-length well-formed character array, and then use `strcat()` to append the contents of arrays `a`, and `b`.

- Who is responsible for deleting the returned memory? Is there a leak? How would you fix it?

Part 2 - Writing the joinb() function

- Start your `joinb()` function as a copy of `join()`, then modify the concatenation part to insert a space between the two input strings.

- Do any other parts of the code need to be modified?