FEEG6002 Advanced Computational Methods 1:

Laboratory-Assignment 5

Contents

Prerequisites: Strings

This lab session focuses on string (and implicitly array) handling, and revises the concepts we have learned last week in more complex examples.

A comment on string handling in C:

Introduction We are now starting to work with pointers, arrays, and associated conventions - for example the convention for strings that they are (i) arrays of chars of which (ii) the last element must be a binary zero (written as '\0' in C).

Revision As a reminder: this binary zero does not count as part of the length of a string but needs a char in memory like all other characters. Functions such as printf and strlen expect the binary zero at the end of the string to know where the string ends. Imagine the string char s[] = "Hello"; -- this has the length 5 because it contains 5 characters, but needs 6 chars in memory because the sixth element is the '\0'. If we print this using printf("My string %s", s);, then the printf function will start at the memory address to which s points [because s is actually a pointer to s[0], i.e. the letter H], and print one character after the other until it finds a binary zero. So it will print Hello and then stop because s[5] (i.e. the sixth element) contains a binary zero. If the binary zero was not there, then printf would continue to print data from memory (until it comes across a zero by chance). At some point the operating system will kill the program because it has started to read in parts of the computer's memory that are not owned by the program. (This is technically known as a segmentation fault, because the computer is reading/writing data in the wrong segment of memory.)

What about testing these codes? The testing of such code is notoriously difficult: if we modify a string and forget to add/move the binary zero to its end, we may not see this immediately when running the code: it is quite possible (and very common after booting) that the computer's memory happens to contain lots of zeros. So we may read a binary zero at the end of a string (and thus the string looks properly terminated and our program works) because the zero happened to be there in memory *by chance*. Running the same program at another time or a different machine may lead to problems if the binary zero is not present to terminate the string.

What do we learn? Being able to run your program without seeing any crashes or problems does *not* necessarily mean that it is bug free. You need to be creative in how you test this, and/or think hard about what exactly is happening. Do discuss this with the demonstrators or your peers to get a better understanding.

1 Training: Strip off spaces from the end of a string

- In a file with name training5.c, write a C function void rstrip(char s[]) that modifies the string s: if at the end of the string s there are one or more spaces, then remove these from the string.

  For example, the string "Hello   " should be modified by rstrip to become "Hello".

  The string "Hello World" should be modified to become "Hello World".

  The name *rstrip* stands for *Right STRIP*, trying to indicate that spaces at the 'right' end of the string should be removed.

  You are invited to use the template training5.c to write and test your function.

  training5.c

```
  #include <stdio.h>

  /* Function void rstrip(char s[])
  modifies the string s: if at the end of the string s there are one
or more spaces,
  then remove these from the string.

  The name rstrip stands for Right STRIP, trying to indicate that
  spaces at the 'right'
  end of the string should be removed.
  */

  void rstrip(char s[]) {
  /* to be implemented */
  }


  int main(void) {
  char test1[] = "Hello World   ";

  printf("Original string reads  : |%s|\n", test1);
  rstrip(test1);
  printf("r-stripped string reads: |%s|\n", test1);

  return 0;
  }
```

- Test your function: The template training5.c contains a main function that calls rstrip and thus provides one possible test. Either before or after you have implemented your first attempt of rstrip, try to think of more possible tests:
  - what are "difficult" cases?
  - for what input might you expect your code to fail?
  - have you considered extreme examples? I.e.:
    - an empty string,
    - a string that does not contain spaces

    - a string consisting of multiple words
    - a long string?

  By trying to anticipate all these cases *before* you submit to the testing system, you practice your skills in systematic testing of code you write.

As usual, submit your work by email to feeg6002@soton.ac.uk with subject line training 5 to receive feedback. The tests will only test the function rstrip. You can thus modify the main function as you like to support your code writing, debugging and testing. However, as usual, your file should compile without errors and warnings when using the recommended switches; this includes the main function.

2 Laboratory: Strip off spaces from the beginning of a string

- In a file with name lab5.c, write a C function void lstrip(char s[]) that modifies the string s: if at the beginning of the string s there are one or more spaces, then remove these from the string.

  For example, the string "   Hello" should be modified by lstrip to become "Hello".

  The string "Hello World" should be modified to become "Hello World".

  The name *lstrip* stands for *Left STRIP*, trying to indicate that spaces at the 'left' side of the string should be removed.

- Test your function: follow the guidelines given above for the training part to anticipate and check all possible use cases, and to test that your code deals with them correctly.

As usual, submit your work by email to feeg6002@soton.ac.uk with subject line lab 5 to receive feedback. The tests will only test the function lstrip. You can thus modify the main function as you like to support your code writing, debugging and testing. However, as usual, your file should compile without errors and warnings when using the recommended switches; this includes the main function.