Exercise 5 – Pointers

Objective

There are two objectives in this session: firstly to practice using function pointers and to understand when this approach is suitable, and secondly to consolidate your understanding of pointer arithmetic.

Reference Material

This practical session is based on the material in the Pointers in C chapter. The session is located in the following directory:

	Microsoft Windows	*Linux*
Directory:	**c:\qacadv\pointers**	**~/qacadv/pointers**
Solution directory:	**c:\qacadv\pointers\Solution**	**~/qacadv/pointers/Solution**

Overview

Many programs use function pointers to achieve flexible table look-up behaviour. The first two questions in this session use function pointers to implement a simple menu system. Questions 3 and 4 use pointer arithmetic to achieve a simple character filter, where characters typed at the keyboard are translated via a look-up table before being echoed on the screen.

Practical Outline

1. The concept of an array of pointers to functions can be used to rewrite complex `switch` statements where a different function is called for each `switch`.

   On Microsoft Windows open **funcptrs.sln, on Linux** change your current working directory.

   Now study the code in **funcptrs.c**. Eliminate the switch statement by building an array of pointers to functions holding the addresses of `Command0`, `Command1` etc. and selecting a member from that table. A solution is provided in the **Solution** sub-directory, called **funcptrs.c**.

2. **On Microsoft Windows** open **menu.sln, on Linux** stay in the same working directory.

   Open the (empty!) file called **menu.c**. Design and implement a `menu()` function. This will display a list of options: “Cut”, “Copy”, “Paste” and “Select All”.

   With each text item there will be an associated function: **cut_func**, **copy_func**, **paste_func** and **select_all_func**. Function pointers to these will be held in an array.

   Prompt the user to choose one of the functions (give each a number correspomding to its postion in the array, then `scanf` an integer) and call the function corresponding to the choice. Thus, 0 calls cut_func, 1 calls copy_func etc.,

   Try to make your function as generic as possible. Do not restrict yourself to fixed length strings. Also, do not assume a fixed number of entries in the list. A solution is provided in the **Solution** subdirectory called **menu.c**.

Optional Questions

Note: this question uses pointers, but not function pointers.

3. **On Microsoft Windows** open **trans.sln, on Linux** stay in the same working directory.

   Open the empty file called **trans.c**. Write a simple translation program which maps characters typed at the keyboard into characters on standard output. The mapping is to be determined by two command line arguments supplied by the user. Any character found in the first string argument is mapped to the character at the corresponding position in the second string argument. Characters not found in the first parameter are not translated - they are passed unchanged to the output. For now, assume (or insist) that the translation strings are of the same length.

   For example:
   ```
   trans1 abcdef uvwxyz
   the quick brown fox jumped
   thy quiwk vrown fox jumpyd
   ```
   And:
   ```
   trans1 aeiou AEIOU
   the quick brown fox jumped
   thE quIck brOwn fOx jUmpEd
   ```

   A solution is provided in the **Solution** sub-directory called **trans1.c**.

4. Working in the same directory (and solution), consider what should happen if the first argument is longer than the second. In this case a reasonable behaviour is that characters in the excess portion of the first argument are mapped onto the last character of the output string.

   For example, to translate all digits into exclamation marks:

   ```
   trans2 0123456789 !
   ```

   Similarly, to convert all vowels into full stops:

   ```
   trans2 AEIOUaeiou .
   ```

   Improve your program to cope with these cases. A solution is provided in the **Solution** sub-directory called **trans2.c**.

   On Linux be careful with the character used, for example in the C-shell and Bash the exclamation mark is a special shell character, so it must be 'escaped':

   ```
   trans2 0123456789 \!
   ```