MCS 2016-2017
C/C++ Mock Laboratory Examination

Imperial College London

Monday 12 December 2016

- ☞ You must complete and submit a working program by the end of the session.
- ☞ Log into the Lexis exam system using your DoC login as both your login and as your password (**do not use your usual password**).
- ☞ You are required to create a header file **shrink.h**, a corresponding implementation file **shrink.cpp** and a **makefile** according to the specifications overleaf.
- ☞ Use the file **main.cpp** to test your functions. You will find this file in your Lexis home directory (**/exam**). If you are missing this file please alert one of the invigilators.
- ☞ **Save your work regularly.**
- ☞ The system will log you out automatically once the exam has finished. **It is therefore important that you save your work and quit your editor when you are told to stop writing.** No further action needs to be taken to submit your files - the final state of your Lexis home directory (**/exam**) will be your submission.
- ☞ No communication with any other student or with any other computer is permitted.
- ☞ **This question paper consists of 4 pages.**

Problem Description

Dictionary-based compaction is a lossless compression technique that makes use of a table of commonly used words to compress text.

So given the table:

encoding	word
00	iguana
01	aardvark
02	swimming
03	Zambezi
04	river
05	rescue

the sentence:

The iguana and the aardvark were swimming in the Zambezi river.

is compressed as:

The 00 and the 01 were 02 in the 03 04.

To avoid ambiguities when decoding, any occurrences of *00*, *01*, *02* etc. in the text to be compressed are prefixed with a '!' character, thus being encoded as *!00*, *!01*, *!02* etc. Further, any occurrences of the '!' character are replaced by '!!'.

Thus the sentence:

The aardvark and the iguana went swimming on 12/01/10!

is compressed as:

The 00 and the 01 went 02 on !12/!01/!10!!

For the purposes of this exercise, you may assume that there will be at most 100 entries in the table (encoded from 00 to 99).

Specific Tasks

1. Write a function `lookup(word,dictionary)` which returns the encoding of the word in the dictionary (as an integer). If the word is not present in the dictionary then the function should return -1. The first parameter to the function (i.e. word) is a read-only string containing a single English word (or a number). The second parameter (i.e. dictionary) is an array of read-only strings making up the dictionary, terminated by an empty string.

   For example, the code:

   ```
   const char *dictionary[] = { "iguana", "aardvark",
    "swimming", "Zambezi", "river", "rescue" , "" };

   int word = lookup("aardvark", dictionary);
   ```

   should result in the integer `word` having the value 1.

2. Write a function `encode(word, compressed, dictionary)` which produces the compressed version of a given word (or number string). The first parameter to the function (i.e. `word`) is an input string containing the word to be compressed. The second parameter (i.e. `compressed`) is an output parameter which should contain the corresponding dictionary encoding; if the word does not exist in the dictionary, then the compressed word should be the same as the input word. Number strings should be encoded by prefixing them with a "!" (so "`05`" becomes "`!05`").

   For example, the code:

   ```
   char compressed[100];
   encode("Zambezi", compressed, dictionary);
   ```

   should result in the string `compressed` having the value "`03`".

   As another example, the code:

   ```
   char compressed[100];
   encode("elephant", compressed, dictionary);
   ```

   should result in the string `compressed` having the value "`elephant`".

Place your function implementations in the file **shrink.cpp** and corresponding function declarations in the file **shrink.h**. Use the file **main.cpp** to test your functions. Create a **makefile** which compiles your submission into an executable file called **shrink**.

(The two parts carry, respectively, 40% and 60% of the marks)

Hints

1. Feel free to define any auxiliary functions which would help to make your code more elegant.
2. The standard header `<cctype>` contains some library functions that you may find useful. In particular:
   - `int isalpha(char ch)` returns nonzero if `ch` is a letter from `'A'` to `'Z'` or a letter from `'a'` to `'z'`.
   - `int isdigit(char ch)` returns nonzero if `ch` is a digit between `'0'` and `'9'`.
   - `int isalnum(char ch)` returns nonzero if `ch` is a letter from `'A'` to `'Z'`, a letter from `'a'` to `'z'` or a digit between `'0'` and `'9'`.
3. Try to attempt all questions. If you cannot get one of the questions to work, try the next one, e.g. if you have the function prototype for Question 1, you can attempt Question 2.