Imperial College of Science, Technology and Medicine
Department of Computing

M.Sc. C++ Programming – Unassessed Exercise No. 6

Issued: Friday 16 October 2015

Problem Description

An *error-correcting code* encodes a stream of binary digits (bits) for transmission over an unreliable channel such that (some) bit errors can not only be *detected* by the receiver, but can also be *corrected*.

Consider the steps in the transmission of the (very short single-letter) message "A" over an unreliable channel from a sender to a receiver:

1. The sender encodes each character of the message as a sequence of 8 bits. In our case, the ASCII code for 'A' is 65. 65 is 01000001 in binary[1] so the stream of data bits to be sent is 01000001.

2. The sender now inserts check bits (extra error correction information) into the stream of data bits. This is done by considering 4 data bits (call them d_1, d_2, d_3, d_4) at a time, and calculating 3 corresponding check bits (call them c_1, c_2, c_3). The 4 bits of data and 3 check bits are then interleaved and transmitted as a 7-bit code word in the order $c_1, c_2, d_1, c_3, d_2, d_3, d_4$.[2]

   The check bits are calculated as follows:

$$\begin{aligned} c_1 &= \mathsf{parity}(d_1, d_2, d_4) \\ c_2 &= \mathsf{parity}(d_1, d_3, d_4) \\ c_3 &= \mathsf{parity}(d_2, d_3, d_4) \end{aligned}$$

   Here $\mathsf{parity}(a, b, c)$ is 0 if $(a + b + c)$ is even, and 1 otherwise.

   In our example, the first four bits of our data stream are 0100 (so $d_1 = 0$, $d_2 = 1$, $d_3 = 0$, $d_4 = 0$). The check digits are $c_1 = 1$, $c_2 = 0$, $c_3 = 1$ (check them). Consequently the 7-bit code word transmitted is 1001100[3]. The next four bits of our data stream are 0001, with corresponding 7-bit code word 1101001 (check it). Thus the error-corrected data stream transmitted is 1001100 1101001.

3. During transmission from sender to receiver, the error-corrected data stream might undergo some bit corruption. In our example, suppose bit 6 (currently a 0) becomes a 1, so that the received data stream is 1001110 1101001.

4. The receiver considers each 7-bit code word received in turn (call the bits received $b_1, b_2, b_3, b_4, b_5, b_6, b_7$) and computes three parity check values (call them p_1, p_2, p_3) as follows:

$$\begin{aligned} p_1 &= \mathsf{parity}(b_4, b_5, b_6, b_7) \\ p_2 &= \mathsf{parity}(b_2, b_3, b_6, b_7) \\ p_3 &= \mathsf{parity}(b_1, b_3, b_5, b_7) \end{aligned}$$

[1] Recall that binary is the base-two number system used by computers to represent data internally.

[2] The reason for this particular interleaving order will become apparent during the decoding/correction process.

[3] Remember the interleaving order is c_1, c_2, d_1, c_3, d_2, d_3, d_4.

Here $\mathsf{parity}(a, b, c, d)$ is 0 if $(a + b + c + d)$ is even, and 1 otherwise.

If p_1, p_2 and p_3 are all 0, there were no single-bit errors in the transmission of this code word, and the original data bit stream can be recovered as b_3, b_5, b_6, b_7. If one or more of p_1, p_2, p_3 are 1, however, then there has been an error. The position of the bit error is given by the decimal value of the binary number $p_1 p_2 p_3$[4]. This bit can be flipped[5] to correct the error, and the original data bit stream recovered as b_3, b_5, b_6, b_7.

In our example, the first 7-bit code word received is 1001110. The parity bits are $p_1 = 1$, $p_2 = 1$, $p_3 = 0$ (check them). Thus the position of the bit error in binary is 110, which is 6 in decimal. Flipping bit 6 yields corrected code word 1001100, and extracting b_3, b_5, b_6, b_7 gives corrected data bits 0100.

The next 7-bit code word received is 1101001. Now the parity bits are $p_1 = 0$, $p_2 = 0$, $p_3 = 0$ (check them). This indicates that there were no single-bit errors. The original data bit stream can be recovered by extracting b_3, b_5, b_6, b_7 to give data bits 0001.

5. The receiver has thus received the correct binary data stream (0100 0001) even though a bit error was made in transmission. (In fact the error correction is strong enough to correct a single-bit error in every 7-bit code word transmitted).

Pre-supplied helper functions

To get you started, you are initially supplied with two helper functions (with prototypes in **correct.h** and implementations in the file **correct.cpp**, both of which are available from the URL: `http://www.doc.ic.ac.uk/~wjk/C++Intro/correct/`):

1. `void ascii_to_binary(char ch, char *binary)` which takes in a character (`ch`) and returns (in the output parameter `binary`) a string representation of the 8-bit binary number corresponding to the ASCII code of `ch`.

   For example, the code:

   ```
   char binary[9];
   ascii_to_binary('A', binary);
   ```

   results in the string `binary` having the value "`01000001`" (The ASCII code for 'A' is 65 in decimal, which is 01000001 in binary).

2. `char binary_to_ascii(char *binary)` which converts a string representing a binary number (in the input parameter `binary`) to an ASCII character (the return value of the function).

   For example the code:

   ```
   char ch;
   ch = binary_to_ascii("01000001");
   ```

   results in the character `ch` having the value `'A'`.

[4]Now the reason for the chosen interleaving ordering may be more apparent.

[5]i.e. if it is a '0' make it a '1', if it is a '1' make it a '0'

Specific Tasks

1. Write a function `text_to_binary(str,binary)` which converts a string of characters (the input parameter `str`) into a corresponding stream of bits[6] (the output string parameter `binary`). Also write a complementary function `binary_to_text(binary,str)` which converts a stream of bits (the input string parameter `binary`) into a corresponding character string (the output parameter `str`).

   For example the code:

   ```
   char binary[512];
   text_to_binary("Art",binary);
   ```

   should result in the string `binary` having the value "`010000010111001001110100`" (This is because the binary representations of the ASCII codes of 'A', 'r' and 't' are 01000001, 01110010 and 01110100 respectively).

   In a complementary fashion, the code:

   ```
   char text[32];
   binary_to_text("010000010111001001110100", text);
   ```

   should result in the string `text` having the value "`Art`".

   For full credit for this part, at least one of your functions should be recursive. However, partial credit (up to 75%) will be awarded for purely iterative solutions.

2. Write a function `add_error_correction(data, corrected)` which outputs (in the output string parameter `corrected`) an error-corrected version of a data bit stream (the input string parameter `data`).

   For example, the code:

   ```
   char correct[512];
   add_error_correction("0100", correct);
   ```

   should result in the string `correct` having the value "`1001100`". (For an explanation of this, see part 2 of the Problem Description).

   Similarly, the code:

   ```
   char correct[512];
   add_error_correction("01000001", correct);
   ```

   should result in the string `correct` having the value "`10011001101001`".

3. Write a function `decode(received, decoded)` which outputs (in the output string parameter `decoded`) an error-corrected version of a received bit stream (the input string parameter `received`). The function should return the number of errors corrected.

   For example, the code:

   ```
   char decoded[512];
   errors = decode("1001110", decoded);
   ```

   should result in the string `decoded` having the value "`0100`", with the value of `errors` set to `1`. (For an explanation of this, see part 4 of the Problem Description).

[6] You may represent a "stream of bits" as a string consisting of '0' and '1' characters

What to hand in

Place your function implementations in the file **correct.cpp** and corresponding function declarations in the file **correct.h**. Use the file **main.cpp** to test your functions (available from `http://www.doc.ic.ac.uk/~wjk/C++Intro/correct/`). Create a **makefile** which compiles your submission into an executable file called **correct**. If the exercise was assessed you would have to submit three files **correct.cpp**, **correct.h** and a **makefile**. But since it is not assessed, you do not need to hand in anything.

How You Will Be Marked

You will be assigned a mark (for all your programming assignments) according to whether your program works or not, whether your program is clearly set out with adequate blank space, comments and indentation, whether you have used meaningful names for variables and functions, and whether you have used a clear, appropriate and logical design.

Hints

1. Feel free to define any auxiliary functions which would help to make your code more elegant.
2. Question 1 is a lot easier if you make use of the pre-supplied helper functions in your answer.
3. Try to attempt all questions. If you cannot get one of the questions to work, try the next one (all three may be tackled independently).