

Imperial College of Science, Technology and Medicine  
Department of Computing

M.Sc. C++ Programming – Unassessed Exercise No. 1

**Issued:** Friday 14 October 2016

## Objective

The exercise is to write a two argument, integer-valued function `substring_position(...)` which searches to see if its first (string) argument is a substring of its second (string) argument. If it finds an instance of the substring, then it returns the start position of the *first* instance of the substring in the string, otherwise it returns -1. For example:

CALL	SHOULD RETURN
<code>substring_position("this", "this is a simple exercise")</code>	0
<code>substring_position("is", "this is a simple exercise")</code>	2
<code>substring_position("is a", "this is a simple exercise")</code>	5
<code>substring_position("is an", "this is a simple exercise")</code>	-1
<code>substring_position("exercise", "this is a simple exercise")</code>	17
<code>substring_position("simple exercise", "this is a simple")</code>	-1
<code>substring_position("", "this is a simple exercise")</code>	0
<code>substring_position("", "")</code>	0

The function declaration should be placed in a header file called `substring.h` (along with any other function declarations you might decide to use in your answer), and the function implementation should be placed in an implementation file `substring.cpp` (along with any other function implementations). The header and implementation files should be tested using the test program in the file `main.cpp`, which can be found on the web at the URL <http://www.doc.ic.ac.uk/~wjk/C++Intro/substring/main.cpp>.

## Hints And Suggestions

Try to complete the exercise without using any of the standard string library functions (in particular, the use of `strstr` is strongly discouraged!). You may find it useful to decompose the task as follows:

1. Write a Boolean valued function `is_prefix(...)` which has two string arguments (and possibly other parameters as well). `is_prefix(...)` returns `True` if its first string argument (or perhaps some part of its first string argument specified by other parameters) is a prefix of its second string argument (or perhaps some part of the second string argument specified by other parameters).

You might decide to write a recursive definition for `is_prefix(...)`, since, for example, to check that “indent” is a prefix of “indentation” it is sufficient to check that both strings start with the same letter, and that “ndent” is a prefix of “ndentation”.

2. Test the function `is_prefix(...)`. When satisfied that it is correct, use it in the (recursive or iterative) definition of `substring_position(...)`.

See if you can:

- Define `substring_position(...)` and `is_prefix(...)` using pointer arithmetic instead of array subscripting for running through the strings, and
- define your function `substring_position(...)` recursively, and
- define the function `is_prefix(...)` recursively using exactly two parameters.

However, it is recommended that you first answer the question in whatever way you find most natural before spending too much effort on these finer points.

## A challenge for fun

See if you can provide a one-line iterative function `substring_position2(...)` which uses the string library function `strstr` to perform the same task as `substring_position(...)`. Your function definition body (inside the curly braces) should contain only one semicolon, should not make use of any preprocessor symbols (i.e. no `#define`!) or global variables, and should not invoke any function other than `strstr`. Can you equal or better the current record of just 29 characters for the function body (not counting the braces)?

Imperial College of Science, Technology and Medicine  
Department of Computing

M.Sc. C++ Programming – Unassessed Exercise No. 2

**Issued:** Friday 14 October 2016

## Objective

You are required to implement C/C++ functions for encoding and manipulating surnames according to the way they sound, by using the *Soundex* coding system.

## The Soundex coding system

The *Soundex* coding system represents surnames according to the way they sound rather than the way they are spelt. Surnames that sound the same but are spelt differently (e.g. PAYNE and PAINE) have the same Soundex code.

All Soundex codes take the form of an upper case letter followed by three digits (e.g. B652, E255 or M263). The rules for mapping a surname onto its corresponding Soundex code are as follows:

1. The letter at the start of the Soundex code is always the first letter of the surname.
2. The digits of the Soundex code are assigned using the remaining letters of the surname according to the mapping:

b, f, p, v	→	1	l	→	4
c, g, j, k, q, s, x, z	→	2	m, n	→	5
d, t	→	3	r	→	6

All occurrences of the letters a, e, h, i, o, u, w and y are ignored. If there are not enough digits to make up a four character code, zeroes are added to the end. Additional letters are disregarded.

For example, **Washington** is coded **W252** (W, 2 for the S, 5 for the N, 2 for the G, remaining letters disregarded). **Lee** is coded **L000** (L, 000 added).

3. If the *surname* has two or more *adjacent* letters that have the same number in the Soundex coding, they should be treated as one letter.

For example, **Jackson** is coded as **J250** (J, 2 for the C, K ignored, S ignored, 5 for the N, 0 added).

Note that this rule does **not** imply that it is impossible for adjacent repeated digits to occur in a Soundex coding.

## Tasks

Your specific tasks are to:

1. Write a function `encode(surname, soundex)` which produces the Soundex encoding corresponding to a given surname. The first parameter to the function (i.e. `surname`) is an input string containing the surname that should be encoded. The second parameter (i.e. `soundex`) is an output string into which the Soundex encoding for the string should be placed.

For example, the code:

```
char soundex[5];  
encode("Jackson", soundex)
```

should result in the string `soundex` having the value "J250"

2. Write a **recursive** function `compare(one, two)` which compares two Soundex codes (where `one` and `two` are strings representing the Soundex codes being compared). If the Soundex codes are the same, the function should return 1, else it should return 0.

For example, `compare("S250", "S255")` should return 0 while `compare("W252", "W252")` should return 1.

3. Write a function `count(surname, sentence)` which returns the number of words in a given sentence that have the same Soundex encoding as the given surname.

For example, the function call

```
count("Leeson", "Linnings, Leasonne, Lesson and Lemon.")
```

should return the value 2.

Place your function implementations in the file `soundex.cpp` and corresponding function declarations in the file `soundex.h`. Use the file `main.cpp` to test your functions. `main.cpp` can be found on the web at <http://www.doc.ic.ac.uk/~wjk/C++Intro/soundex/main.cpp>.

## What to hand in

If this was an assessed exercise, you would be required to submit the files `soundex.h`, `soundex.cpp` and a `makefile` using a web submission page. But since this exercise is not assessed you do not actually need to hand anything in.

## 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. 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 character between `'A'` and `'Z'` or between `'a'` and `'z'`.
  - `char toupper(char ch)` returns the upper case equivalent of character `ch`.
3. There are at least three different ways of implementing your answer to Question 1. What are they? Which way is the most elegant?
4. Your answer to Question 3 (i.e. your implementation of the `count` function) should not modify the second parameter (i.e. `sentence`) directly. However, it is acceptable if your code makes and then modifies a *copy* of this parameter.

Imperial College of Science, Technology and Medicine  
Department of Computing

M.Sc. C++ Programming – Unassessed Exercise No. 3

**Issued:** Friday 14 October 2016

## Objective

A **palindrome** is a word or phrase which contains the same sequence of letters when read forwards or backwards.

Characters other than letters (including spaces) are ignored and the case of letters (i.e. whether they are upper case i.e. A-Z or lower case i.e. a-z) is irrelevant. An empty string and a string containing a single letter are both palindromes (by definition).

For example,

- The words “rotor” and “Radar” are palindromes.
- The sentence “A man, a plan, a canal, Panama!” is a palindrome, and so is “Mr. Owl ate my metal worm.”
- The strings “a. . .” and “. . . !” are palindromes.

An **anagram** is a word or phrase formed by reordering the *letters* of another word or phrase. Again, characters other than letters (including spaces) are ignored and the case of letters is irrelevant.

For example:

- “stain” is an anagram of “satin”
- “I am a weakish speller!” is an anagram of “William Shakespeare”
- “Here come dots...” is an anagram of “The Morse Code”.

The purpose of this exercise is to write a set of functions which can be used to identify palindromes and anagrams.

## Tasks

Your specific tasks are to:

1. Write a function `reverse(str1, str2)` which copies `str1` into `str2` backwards. Here `str1` is an input parameter containing the string to be reversed, and `str2` is an output parameter containing the reversed string. So the code:

```
char reversed[9];
reverse("lairepmi", reversed);
```

should result in the string `reversed` having the value “`imperial`”. You may assume that the memory for `string2` has been allocated and is sufficient to hold the output string (including sentinel character).

2. Write a **recursive** function `compare(one, two)` which compares two strings **ignoring punctuation and letter case** (where `one` and `two` are the two strings being compared). If the strings are the same, the function should return 1, else it should return 0.

For example, `compare("This and that", "this, and THAT!")` should return 1 while `compare("these are not the SAME", "these are the SAME")` should return 0.

3. Write a function `palindrome(sentence)` which returns 1 if the given sentence is a palindrome, and 0 otherwise. You may not make any assumptions about the size of the input sentence.

For example, the function call `palindrome("Madam, I'm Adam")` should return 1 while the function call `palindrome("Madam, I'm not Adam")` should return 0.

4. Write a function `anagram(str1, str2)` which returns 1 if string `str1` is an anagram of string `str2`, and 0 otherwise.

For example, the function call `anagram("William Shakespeare", "I am a weakish speller!")` should return 1, while the call `anagram("William Shakespeare", "I am a good speller!")` should return 0.

Place your function implementations in the file **words.cpp** and corresponding function declarations in the file **words.h**. Use the file **main.cpp** to test your functions. The file **main.cpp** can be found on the web at the URL <http://www.doc.ic.ac.uk/~wjk/C++Intro/palindrome/main.cpp>.

*(The four parts carry, respectively, 20%, 30%, 20% and 30% of the marks)*

## What To Hand In

If this was an assessed exercise, you would be required to submit three files: `words.h`, `words.cpp` and a `makefile`. But since it is unassessed, you do not need to hand in anything.

## How You Will Be Marked

If this was assessed, you would be assigned a mark according to:

- whether your program works or not,
- whether your program is clearly set out with adequate blank space and indentation,
- whether your program is adequately commented,
- whether you have used meaningful names for variables and functions, and
- whether you have used a clear, appropriate and logical design.

## Hints

1. Try to attempt all questions. Note that Questions 1 and 2 can be attempted independently, and if you have function prototypes for Questions 1 and 2, you can write Question 3. If you have the function prototype for Question 2, you can write Question 4.
2. The standard header `<cctype>` contains some library functions that you may find useful when answering **Question 2**. In particular:
  - `int isalpha(char ch)` returns nonzero if `ch` is a character between `'A'` and `'Z'` or between `'a'` and `'z'`.
  - `char toupper(char ch)` returns the upper case equivalent of character `ch`.
3. Please note that your solution to **Question 2** should be **recursive**, i.e. the function should call itself. If you cannot manage a recursive solution, up to 50% of the marks will be awarded for an iterative (non-recursive) solution.
4. Feel free to define any auxiliary functions which would help to make your code more elegant. In particular, when answering **Question 4**, you might find it useful to create an auxiliary case-insensitive string sorting function.



Imperial College of Science, Technology and Medicine

Department of Computing

## M.Sc. C++ Programming – Unassessed Exercise No. 4

Issued: Friday 14 October 2016

### Problem Description

The *Braille* system<sup>1</sup> is used by blind people to read and write.

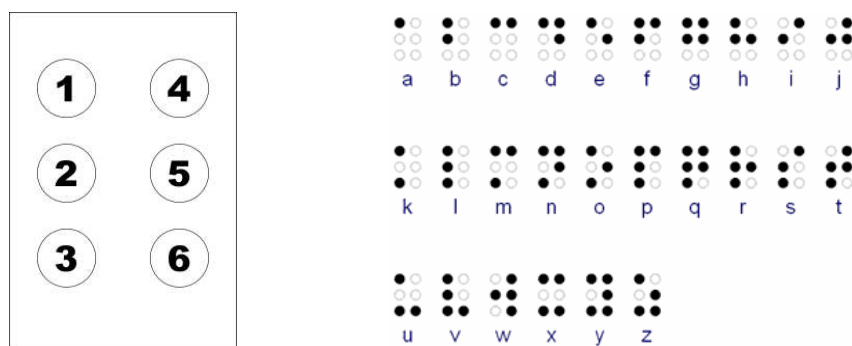

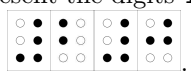









Figure 1: A Braille cell (left) and Braille letter encodings (right).

Braille text consists of a sequence of *cells*, each of which contains six dot positions arranged in two columns of three. The dot positions are numbered 1,2,3 downwards from the top of the lefthand column and 4,5,6 from the top of the righthand column (see left of Figure 1).

The presence or absence of raised dots on the dot positions gives the coding for a symbol. For example, letters are encoded as shown on the right in Figure 1. By default, letters are lower case.

Capital letters are indicated by a preceding each letter by the *capital sign* .

Numbers are represented by preceding each digit in the number by the *number sign*<sup>2</sup> . The digits themselves are encoded by reusing the letters a through i to represent the digits 1 through 9 respectively, and using j to represent the digit 0. So we encode 20 as .

Punctuation is encoded as follows. A period/full stop ('.') is , a comma (',') is , a semicolon (;) is , a hyphen/dash ('-') is , an exclamation mark ('!') is  and a question mark ('?') is . Opening and closing brackets ('(' and ')') are both encoded as .

<sup>1</sup>Devised by Louis Braille in 1821, the Braille system is an adaption of an early tactile military communications system devised by Charles Barbier called *night writing*. Night writing encodes symbols as columns of raised dots and was proposed as a way to allow Napoleon's soldiers to communicate silently and without light at night.

<sup>2</sup>A simplification for this exercise; in real Braille the number sign stays in effect until the next space.

## Specific Tasks

1. Write a function `encode_character(ch,braille)` which produces a string (the second parameter `braille`) that represents the Braille encoding of a single input character (the first parameter `ch`). In the string encoding, a Braille cell should be represented as 6 consecutive characters, one for each dot position; raised dots should be encoded as '0' and unraised dots as '.' For example, the character 'n' should be encoded as the string "0.000.". Any character for which the encoding has not been given (including the space character) should be encoded as ".....". The function should return the length of the Braille-encoded string.

For example, the code:

```
char braille[20];
int size;
size = encode_character('t', braille);
```

should result in the string `braille` having the value ".0000." and `size` having the value 6.

As another example, the code:

```
char braille[20];
int size;
size = encode_character('Z', braille);
```

should result in `braille` having the value ".....0|0.0.00" (i.e. the capital sign followed by the letter z)<sup>3</sup> and `size` having the value 12.

As a final example, the code:

```
char braille[20];
int size;
size = encode_character('5', braille);
```

should result in `braille` having the value "..0000|0...0." (i.e. the number sign followed by the encoding for the letter e)<sup>3</sup> and `size` having the value 12.

2. Write a **recursive** function `encode(plaintext,braille)` which produces the Braille encoding of a plaintext input string. The first parameter to the function (i.e. `plaintext`) is an input string containing the string to be encoded. The second parameter (i.e. `braille`) is an output parameter which should contain the corresponding Braille-encoded string.

For example, the code:

```
char braille[100];
encode("Hello!", braille);
```

should result in the string `braille` having the value:

".....0|00...0.|0...0.|000...|000...|0.0.0.|.00.0.".

As in Question 1, the dashed vertical bars are included here only to aid understanding; they should **not** be included in the output string.

3. Write a function `print_braille(plaintext,output)` which takes a plaintext string and writes the corresponding sequence of 3x2 Braille cells to an output stream. The function should also display the corresponding plaintext character under each Braille cell (where

---

<sup>3</sup>Note the dashed vertical bars in the output string are included here only as an aid to understanding; they should not be included in the output string.

applicable). The first parameter (i.e. **braille**) is the input plaintext string and the second parameter is the output stream (e.g. **cout** or a file output stream).

For example, the code:

```
print_braille("Hello!", cout);
```

should result in the following output written to **cout** (i.e. displayed on the screen):

```
.. 0. 0. 0. 0. 0. ..
.. 00 .0 0. 0. .0 00
.0 .. .. 0. 0. 0. 0.
  H e l l o !
```

## What To Hand In

Place your function implementations in the file **braille.cpp** and corresponding function declarations in the file **braille.h**. Use the file **main.cpp**, available at:

<http://www.doc.ic.ac.uk/~wjk/C++Intro/braille/main.cpp>

to test your functions. Create a **makefile** which compiles your submission into an executable file called **braille**. If this was an assessed exercise, you would be required to submit the three files: **braille.h**, **braille.cpp** and **makefile** electronically using the CATE system.

## 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. Try to attempt all questions. If you cannot get one of the questions to work, try another.
3. 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 isupper(char ch)` returns nonzero if `ch` is a letter from 'A' to 'Z'.
  - `char tolower(char ch)` returns the lower case character corresponding to `ch`.
  - `int isdigit(char ch)` returns nonzero if `ch` is a digit between '0' and '9'.
  - `int ispunct(char ch)` returns nonzero if `ch` is a punctuation character.
4. Please note that your solution to **Question 2** should be **recursive**, i.e. the function should call itself. If you cannot implement a recursive solution, up to 75% of the marks will be awarded for an iterative (non-recursive) solution.

Imperial College of Science, Technology and Medicine  
Department of Computing

M.Sc. C++ Programming – Unassessed Exercise No. 5

**Issued:** Friday 14 October 2016

## Problem Description

*Pig Latin*<sup>1</sup> is an elegant but archaic pseudo-language that was first spoken in the Middle Ages. Invented by commoners so that they could sound as fancy as their lords, it is based on manipulating the letters of English words so that they sound like Latin words in plural feminine form (i.e. they end in an “ay” sound).

It is a little known fact that the Magna Carta was first written in Pig Latin and only later translated into lesser tongues. Today, Pig Latin is used mostly for amusement although it also has more serious applications such as obfuscating song titles to evade copyright restrictions on music web sites.

Any English word may be changed into its Pig Latin equivalent as follows:

1. If the word begins with a vowel, add “way” to it. For example, Pig Latin for **apple** is **appleway**.
2. If the word begins with a letter that is not a vowel, find the first occurrence of a vowel, move all the characters before the vowel to the end of the word, and add “ay”. For example, **grape** becomes **apegray** and **strong** becomes **ongstray**.
3. If the word contains no vowels just add “ay” to it.
4. For the purposes of this exercise the vowels are **a, e, i, o, u** and **y**; but **y** is only considered a vowel if it is not the first or last letter of the word. So **yeti** becomes **etiyay**, **my** becomes **myay** and **crying** becomes **yingcray**.
5. If the word begins with a character that is a digit, leave the word as is. For example, **300** remains **300**.
6. If the word begins with an initial upper case (capital) letter then so should the corresponding Pig Latin word. For example, **Banana** becomes **Ananabay**. You do not need to handle other capitalization patterns (e.g. all uppercase words).

---

<sup>1</sup>an important language to learn in case you ever find yourself on a Roman farm.

## Specific Tasks

1. Write a function `findFirstVowel(word)` which returns the position of the first “vowel” in the given word (see the Pig Latin rules on the previous page for the precise definition of “vowel”). If the word does not contain a vowel 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.

For example, the code:

```
int vowel;  
vowel = findFirstVowel("passionfruit");
```

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

2. Write a function `translateWord(english, piglatin)` which produces a Pig Latin translation for a given English word. The first parameter to the function (i.e. `english`) is an input string containing the English word. The second parameter (i.e. `piglatin`) is an output parameter which should contain the corresponding Pig Latin translation.

For example, the code:

```
char translated[100];  
translateWord("Banana",translated);
```

should result in the string `translated` having the value `Ananabay`.

3. Write a function `translateStream(inputStream, outputStream)` which takes words from an input stream and writes a corresponding Pig Latin translation to an output stream. The first parameter (i.e. `inputStream`) is the input stream (e.g. `cin` or a file input stream) and the second parameter is the output stream. You may assume that both the input and output streams have been initialised/opened appropriately before the function is called, and that no word in the file is longer than 64 characters.

**For full credit, your solution (to question 3) should be recursive.** However, partial credit (up to 75%) will be awarded for a working iterative solution.

For example, given an input file called `fruit.txt` that contains the text:

```
Time flies like an arrow,  
but fruit flies like a banana!  
(Groucho Marx 1890-1977)
```

then the code:

```
ifstream input;  
input.open("fruit.txt");  
translateStream(input, cout);  
input.close();
```

should result in the following output written to `cout` (i.e. displayed on the screen):

```
Imetay iesflay ikelay anway arrowway,  
utbay uitfray iesflay ikelay away ananabay!  
(Ouchogray Arxmay 1890-1977)
```

## What to hand in

Place your function implementations in the file **piglatin.cpp** and corresponding function declarations in the file **piglatin.h**. Use the files **main.cpp** and **fruit.txt** (available from the URL: <http://www.doc.ic.ac.uk/~wjk/C++Intro/piglatin/>) to test your functions. Create a **makefile** which compiles your submission into an executable file called **piglatin**.

## How You Will Be Marked

If this was assessed, you would be assigned a mark 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. For example, in Question 1 you might find it useful to define a helper function `bool isVowel(char ch, int position, int length)` that returns true if the letter `ch` at position `position` in a string with `length` characters is a “vowel” according to the Pig Latin rules.
2. The standard header `<cctype>` contains some library functions that you may find useful. In particular:
  - `int isupper(char ch)` returns nonzero if `ch` is an uppercase letter from 'A' to 'Z'.
  - `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'.
  - `char toupper(char ch)` returns the upper case equivalent of character `ch`.
  - `char tolower(char ch)` returns the lower case equivalent of character `ch`.

Imperial College of Science, Technology and Medicine  
Department of Computing

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

**Issued:** Friday 14 October 2016

## 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<sup>1</sup> 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  $c_1, c_2, d_1, c_3, d_2, d_3, d_4$ .<sup>2</sup>

The check bits are calculated as follows:

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

Here  $\text{parity}(a, b, c)$  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<sup>3</sup>. 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:

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

---

<sup>1</sup>Recall that binary is the base-two number system used by computers to represent data internally.

<sup>2</sup>The reason for this particular interleaving order will become apparent during the decoding/correction process.

<sup>3</sup>Remember the interleaving order is  $c_1, c_2, d_1, c_3, d_2, d_3, d_4$ .

Here  $\text{parity}(a, b, c, d)$  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  $p_1p_2p_3$ <sup>4</sup>. This bit can be flipped<sup>5</sup> 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'.

---

<sup>4</sup>Now the reason for the chosen interleaving ordering may be more apparent.

<sup>5</sup>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<sup>6</sup> (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).

---

<sup>6</sup>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).