ITB001 Problem Solving and Programming

Semester 1, 2006

Text Processing Assignment
Part 1: Statistical Analysis

Due: Friday, 24th March 2006

Weight: 5%

Overview

The major assignment in ITB001 involves developing a computer program to perform various text processing tasks. Text processing is an important application of Information Technology and the four parts of this assignment will introduce you to a variety of challenges in the area.

Part 1 is an introductory exercise involving statistical analysis of a given text message to determine its size in bits, bytes and (computer) words, and how much it would cost to transmit it. Completing this part of the assignment will give you practice at writing program code, and will briefly introduce the way text is represented in a computer.

Part 2 involves a more sophisticated form of statistical analysis and encryption of a given text message. The analysis procedure counts the number of distinct English words in the message. The encryption procedure allows you to encrypt your message with a particular key so that it can be read only by someone with a corresponding decryption key. Completing this part of the assignment will give you practice at writing and documenting much larger programs, and will give you insight into the challenges involved in interpreting and manipulating text in a computer program.

Part 3 requires you to develop a program for translating text from one language to another. This is done by creating a dictionary of words and their synonyms in another language, and by then using this to replace those words in the text that are found in the dictionary. Completing this part of the assignment will give you practice at designing and using a data abstraction, and will illustrate the difficulty of automatic language translation.

Part 4 combines the features developed in the preceding parts and introduces an interactive spellchecker. In particular, it requires you to construct a user interface

that makes it easy to apply various operations to text messages. Completing this part will give you practice at reusing previously-developed solutions and at developing interactive programs, and introduces the basic ideas underlying the important 'object-oriented' programming paradigm.

Parts 1 to 3 can all be completed independently. Part 4 makes use of your solutions to the previous three parts. In Parts 2 to 4 of the assignment you need to not only produce a working program, but you must also write a document explaining your solution.

Motivation and Background to Part 1

In this part of the assignment our primary goal is to discover how much it would cost to transmit a given SMS or e-mail message, represented here by a character string. To do this we will assume that we are charged proportionally to the size of the message and therefore must first determine how much space the message occupies in our computer's memory. In particular, we assume that we are charged for the whole number of 'words' sent.

However, our use of the term 'word' here refers not to how many English words the text message contains, but to how many 'words' of computer memory the character string occupies. Briefly, all data in a computer is represented using binary digits, or 'bits'. A contiguous sequence of bits forming an accessible sub-field in a computer's memory is known as a 'byte'. Today most computers use 8-bit bytes, although this is merely a convention. Importantly for this part of the assignment, each character in a character string occupies one byte. A sequence of bytes forms a computer 'word', which is the fundamental addressable unit of a computer's memory. For instance, a computer with 8-bit bytes and 32-bit words has 4 bytes per word.

Assignment Tasks for Part 1

To complete this part of the assignment you must develop a Scheme procedure called `text-cost` which tells us how many bits, bytes and words of computer memory are required to store a given text message and how much it would cost to transmit the message.

However, to perform this calculation we need to know how many bits there are per byte, how many bits there are per word, and the cost of sending one word, so these characteristics will be parameters to our procedure. In addition, we will make our procedure robust by having it check the arguments it is given to ensure they are of the right types and have 'reasonable' values.

To get started, download the Scheme template file `ITB001-1.scm` from the ITB001 OLT site. You will edit your solution into this file, using DrScheme, and submit it via the Online Assessment System (OAS) when you have finished.

The necessary features of your `text-cost` procedure are as follows.

1. It must accept the following four arguments:
   (a) A positive number representing the number of bits per byte on the particular computer of interest.
   (b) A positive number representing the number of bits per word on the particular computer of interest.
   (c) A positive number representing the number of cents per word charged to send messages in the SMS or e-mail system of interest.
   (d) A character string representing the text message to be sent.
2. The procedure must return a list of four numbers representing the following values:
   (a) The number of bits required to store the message, assuming that each character in the message occupies one byte.
   (b) The number of bytes required to store the message, assuming that each character in the message occupies one byte.
   (c) The number of whole words required to store the message, assuming that each word contains as many characters as will fit.
   (d) The cost in cents of sending the message, assuming that only whole (not partial) words can be sent.
3. In addition to performing the above calculations, your `text-cost` procedure must check that each of its four arguments is of the correct type. That is, the first three arguments must be positive numbers and the fourth one must be a character string. If this is not the case then the procedure should return the string `"ERROR"`.
4. Finally, your `text-cost` procedure must check that the given arguments are 'reasonable' in two ways:
   (a) The number of bits per byte must be at least eight. This is because a typical computer character set contains 256 characters. To represent this many distinct values requires at least eight binary digits ($2^8 = 256$).
   (b) The number of bits per word must be a whole multiple of the number of bits per byte. This is because a byte is a sub-field of a word. For instance, if bytes consisted of 8 bits, it would be reasonable to have words of 32 bits, because 4 whole bytes would fit into a word. However, it would not make sense to have 30-bit words in this case because only three bytes could fit and the remaining 6 bits in the word would be wasted.

   If either of these conditions is false then your procedure should return `"ERROR"`.

Some Tests for Part 1

Below are some examples you can use to test your solution. (They are the same ones as in the template file.) However, we will use different tests when marking your solution, so you should make sure that your procedure works for other cases as well.

- `(text-cost "8" 16 5 "See Spot run.")` returns `"ERROR"` because `"8"` is a string, not a number
- `(text-cost 0 16 5 "See Spot run.")` returns `"ERROR"` because `0` is not a positive number
- `(text-cost 8 16 -5 "See Spot run.")` returns `"ERROR"` because `-5` is not a positive number
- `(text-cost 8 16 5 7)` returns `"ERROR"` because `7` is a number, not a string
- `(text-cost 6 24 10 "See Spot run.")` returns `"ERROR"` because we can't represent a character in only 6 bits
- `(text-cost 8 30 5 "See Spot run.")` returns `"ERROR"` because the number of bits per word is not a whole multiple of the number of bits per byte
- `(text-cost 8 32 1 "")` returns `(list 0 0 0 0)` bits, bytes, words and cents because a 'null' message consumes no space and costs nothing to send
- `(text-cost 8 32 4 "X")` returns `(list 8 1 1 4)` bits, bytes, words and cents
- `(text-cost 8 32 4 "XYZ")` returns `(list 24 3 1 4)` bits, bytes, words and cents, revealing that it costs the same amount to send three characters as one in this situation
- `(text-cost 8 32 4 "See Spot run.")` returns `(list 104 13 4 16)` bits, bytes, words and cents
- `(text-cost 8 16 5 "The quick brown fox jumps over the lazy dog!")` returns `(list 352 44 22 110)` bits, bytes, words and cents
- `(text-cost 12 60 8 "The quick brown fox jumps over the lazy dog!")` returns `(list 528 44 9 72)` bits, bytes, words and cents[1]
- `(text-cost 11 33 10 "See Spot run.")` returns `(list 143 13 5 50)` bits, bytes, words and cents (even though no computer is likely to ever have such a peculiar architecture!)

[1]Some computing history: The CDC Cyber mainframe computer from the 1970s used twelve bits to represent characters and had 60-bit words. (To be more precise, it actually had 6-bit bytes, but used two bytes to represent each character when the full character set was needed, and it had 64-bit words, but four of these bits were reserved for the operating system and were not accessible by applications programs.)

Criterion Referenced Assessment for Part 1

Overall the ITB001 text processing assignment will be marked on the *correctness*, *clarity* and *conciseness* of your solutions. For this first part, however, only the correctness of your code will be taken into consideration.

With respect to FIT's Graduate Capabilities, assessment of this part of the assignment will use the following guidelines.

Graduate Capabilities	%	High achievement	Good	Unsatisfactory
GC1: Knowledge and Skills	100	Program works to specification	Program has minor defects	Program has major defects
GC4: Lifelong Learning	0	– Your ability to complete technical tasks punctually is assessed indirectly through the application of late penalties – Your ability to use technical documentation effectively is assessed indirectly through your use of the manuals and textbooks needed to successfully complete this part of the assignment		

Submission Process for Part 1

This part of the assignment will be submitted as a single Scheme file. Solutions will be tested automatically, so please adhere to the following rules.

- Complete your solution using the template file from the ITB001 OLT page for the relevant part of the assignment.
- If you have any test examples in the file, please comment them out with semi-colons so that they do not interfere with the testing software.
- Comment out with semi-colons any procedures or code that is syntactically incorrect, i.e., creates an error when you press 'Run' in DrScheme, because this will cause the automatic testing software to reject your whole program.
- Submit your solution via the Online Assessment System (`http://www.fit.qut.edu.au/students/oas/index.jsp`). Once you have logged into OAS using your QUT Access user name and password, one of the assignment items that will be available to you will be for ITB001. You should

choose the action '`Submit`' and submit your solution file. If you make a mistake, such as submitting an empty file or a draft version, you are able to re-submit as many times as you like before the due date and time. You can re-submit by choosing the action '`Re-submit`' and entering the name of the file that you would like to submit. Each re-submission overwrites any previous submissions.

- You must submit your solution before midnight to avoid incurring a late penalty. You should take into account the fact that the network might be slow or temporarily unavailable when you try to submit. Network slowness near the deadline will not be accepted as an excuse for late assignments. To be sure of avoiding a late penalty, submit your solution well before the deadline. (E-mails sent to the teaching staff at 12:01am saying that you are having trouble submitting your file are not appreciated.)

- Standard QUT late penalties will apply. For more information about these penalties please visit `http://www.fit.qut.edu.au/forstudents/current/policy.jsp#assessment`.

- This is an individual assignment. Please read, understand and follow QUT's guidelines regarding plagiarism, which can be accessed via ITB001's OLT site. Scheme programs submitted in this assignment will be subjected to analysis by the MoSS (Measure of Software Similarity) plagiarism detection system (`http://www.cs.berkeley.edu/~aiken/moss.html`).