ITB001 Problem Solving and Programming

Semester 1, 2006

Text Processing Assignment
Part 3: Language Translation

Due: Friday, 5th May 2006
Weight: 20%

Assignment 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 encipher 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.

Overview of Part 3

> *Many Australian university graduates of the SMS and email generation lack the written communication skills to land a job. Australian Association of Graduate Employers president Ben Reeves blamed individuals' reliance on text messaging and emails as their chief means of written communication. Mr. Reeves said the increase in the quality of IT knowledge of candidates was "being counterbalanced by a drop in the written skills of those candidates".*
>
> The Australian Higher Education supplement, 15/2/06

> *Je suis un rock star,*
> *Je avez un residence,*
> *Je habite [lah],*
> *A la South of France,*
> *Voulez vous partir with me,*
> *And come and rester [lah],*
> *With me in France.*
>
> Chorus from Je Suis Un Rock Star by Bill Wyman

In this part of the assignment we will develop a program that will not only help you improve your job prospects, but will also help you appreciate the sensitive lyrics above from the Rolling Stones' guitarist.

Specifically, we will develop a program which allows text messages to be translated from one language to another. This will be done by first defining a dictionary which gives synonyms in the target language for words in the source language. We can then extract source-language words from the text message and, if they are in the dictionary, replace them with an equivalent target-language phrase.

This part of the assignment comprises four distinct tasks.

Task 3a requires you to define the dictionary as a data abstraction that can store and retrieve word-synonym pairs.

Task 3b requires you to write procedures to separate the individual words in the text message and reassemble them again.

Task 3c requires you to complete the translation program with a procedure that replaces words extracted from the text message with their synonyms from the dictionary (if any).

Task 3d requires you to clearly document your solution to Task 3b, which is the most technically challenging task, so that another programmer can understand how you achieved your solution.

Tasks 3a and 3b can be completed independently so if you get temporarily stuck on one you can switch to the other. Task 3c uses your solutions to both Tasks 3a and 3b. Your solutions to Tasks 3a, 3b and 3c will be needed for Part 4 of the overall text processing assignment.

Requirements for Task 3a

The aim of this task is to develop a data abstraction for a dictionary of synonyms, to be used in performing language translation. The abstraction must provide its users with a constructor for creating new dictionaries, operations for updating a given dictionary, and selectors for looking up words and for finding out how many words are defined in a dictionary.

Your task is to implement a data abstraction for dictionaries that provides five separate procedures which can be applied as follows. Symbols in *italics* denote argument expressions.

- `(make-dict)` returns an empty dictionary.
- `(add-to-dict` *word synonym dictionary*`)` returns the given dictionary extended with a new entry linking the given word to its synonym. Both the word and synonym are represented as character strings. If the dictionary already contains an entry for the given word, the old entry should be replaced with the new one.
- `(remove-from-dict` *word dictionary*`)` returns the given dictionary with any entry for the given word removed. If the word did not appear in the dictionary then the dictionary is returned unchanged.
- `(lookup` *word dictionary*`)` returns the synonym associated with the given word in the given dictionary. If, however, there is no entry in the dictionary for the word supplied then the procedure should return string `"???"` to indicate that the word has no synonym in the dictionary.
- `(dict-size` *dictionary*`)` returns the number of distinct words defined in the given dictionary.

Below are some examples you can use to test your solution to Task 3a. (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.) The tests construct two separate dictionaries, one for translating Short Messaging Service (SMS) text to English, and another for translating French to English. (Neither dictionary is very comprehensive—you may choose to extend them with more entries.) Note that these tests are intended to be performed in the order shown. Also note that the (re)definitions used to update the dictionaries do not return values.

1. `(define sms->english (make-dict))` creates the SMS-to-English dictionary

2. `(define french->english (make-dict))` creates the French-to-English dictionary

3. `(lookup "Oui" french->english)` returns string `"???"` because there is nothing in the dictionary yet

4. `(define french->english (add-to-dict "Oui" "Yes" french->english))` uses redefinition to add an entry to the French dictionary

5. `(define french->english (add-to-dict "Non" "No" french->english))`

6. `(lookup "Oui" french->english)` returns string `"Yes"`

7. `(define sms->english (add-to-dict "c" "see" sms->english))` adds an entry to the SMS dictionary

8. `(lookup "u" sms->english)` returns `"???"`

9. `(define sms->english (add-to-dict "r" "are" sms->english))`

10. `(define sms->english (add-to-dict "u" "you" sms->english))`

11. `(lookup "u" sms->english)` returns `"you"`

12. `(define sms->english (add-to-dict "l8r" "later" sms->english))`

13. `(define french->english (add-to-dict "un" "a" french->english))`

14. `(define french->english (add-to-dict "le" "the" french->english))`

15. `(define french->english (add-to-dict "la" "the" french->english))`

16. `(define french->english (add-to-dict "a" "in" french->english))`

17. `(dict-size french->english)` returns `6` which tells us how many words there are in the French dictionary

18. `(define french->english (add-to-dict "avez" "have" french->english))`

19. `(define sms->english (add-to-dict ":-)" "I'm happy" sms->english))`

20. `(dict-size sms->english)` returns `5`

21. `(define sms->english
      (add-to-dict "gr8" "great" sms->english))`

22. `(lookup "l8r" sms->english)` returns `"later"`

23. `(lookup "alligator" sms->english)` returns `"???"`

24. `(define sms->english
      (add-to-dict "lol" "Laughing out loud"
        sms->english))`

25. `(define french->english
      (add-to-dict "Je" "I" french->english))`

26. `(define french->english
      (add-to-dict "suis" "am" french->english))`

27. `(define sms->english
      (add-to-dict "sum1" "someone" sms->english))`

28. `(dict-size sms->english)` returns `8`

29. `(define sms->english
      (remove-from-dict "lol" sms->english))` uses redefinition to remove an entry from the SMS dictionary

30. `(dict-size sms->english)` returns `7` because there is now one less word in the dictionary

31. `(lookup ":-(" sms->english)` returns `"???"`

32. `(define sms->english
      (add-to-dict "rotfl"
        "I'm rolling on the floor laughing"
        sms->english))`

33. `(define sms->english
      (add-to-dict ":-(" "I'm sad!" sms->english))`

34. `(dict-size sms->english)` returns `9`

35. `(define french->english
      (add-to-dict "A" "In" french->english))`

36. `(define french->english
      (add-to-dict "habite" "live in" french->english))`

37. `(define sms->english`
    `(add-to-dict "@" "at" sms->english))`

38. `(define sms->english`
    `(add-to-dict ":-(" "Sob!" sms->english))` replaces a previously-defined word in this dictionary

39. `(dict-size sms->english)` returns `10`

40. `(lookup ":-(" french->english)` returns `"???"` because this is the wrong dictionary in which to find emoticons

41. `(lookup ":-(" sms->english)` returns `"Sob!"` because the original synonym for this emoticon has been replaced

42. `(define sms->english`
    `(add-to-dict "tmrw" "tomorrow" sms->english))`

43. `(dict-size sms->english)` returns `11`

44. `(define sms->english`
    `(add-to-dict "2moro" "tomorrow" sms->english))`

45. `(dict-size sms->english)` returns `12` because there is no reason for two distinct words not to have the same synonym

46. `(lookup "Non" french->english)` returns `"No"`

47. `(lookup "habite" french->english)` returns `"live in"`

48. `(dict-size french->english)` returns `11`

Requirements for Task 3b

Our ultimate goal in Task 3c is to translate words in a text message, using the dictionaries created in Task 3a. To make this possible, however, we first need the ability to extract the individual words from a message. In this task we will define a procedure called `string->tokens` to do this. So that we can reassemble the message once translation is complete, we will also define a complementary procedure `tokens->string`.

Firstly, though, we need to consider carefully what we consider a 'word' to be. Although we would normally say that a word is a contiguous sequence of alphabetic characters, the SMS dictionary shown in the tests above treated combinations of letters and digits as words (e.g., '`l8r`'), as well as emoticons constructed entirely of punctation marks (e.g., '`:-)`'). Therefore, we will divide our text messages into three kinds of 'tokens':

- contiguous sequences of blank spaces (or, more generally, any 'whitespace' characters);
- contiguous sequences of alphanumeric characters (letters and digits); and
- contiguous sequences of punctuation marks (i.e., any characters that are not whitespace, letters or digits).

Your first task is to define a procedure called `string->tokens` which accepts a character string and returns a list of strings comprising the tokens in the original string, as per the above categorisation of tokens, in the order they originally appeared. (This is an example of a 'tokenization' procedure. In computer science such procedures are an important initial step in all language processing programs.)

Your second task is to define a procedure `tokens->string` which accepts a list of strings (tokens) and appends them together to return a single string.

Below are some examples you can use to test your solution to Task 3b. (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.)

- `(string->tokens "See Spot run")` returns
  `(list "See" " " "Spot" " " "run")`
- `(string->tokens " See Spot run ")` returns
  `(list " " "See" " " "Spot" " " "run" " ")`
- `(string->tokens " ")` returns `(list " ")`
- `(string->tokens "")` returns `empty`

- `(string->tokens " The quick brown fox???!!!")` returns
  `(list " " "The" " " "quick" " " "brown" " " "fox" "???!!!")`
  which shows how consecutive punctuation marks are kept together as a single token

- `(string->tokens "This is (very) sneaky! :-)")` returns
  `(list "This" " " "is" " " "(" "very" ")" " " "sneaky" "!" " " ":-)")`
  which shows how the process separates words composed of letters from adjacent punctuation marks

- `(string->tokens "c u l8r, alligator!")` returns
  `(list "c" " " "u" " " "l8r" "," " " "alligator" "!")`
  which illustrates extraction of an alphanumeric token

- `(tokens->string empty)` returns `""`

- `(tokens->string (list "See" " " "Spot" " " "run"))` returns `"See Spot run"`

- `(tokens->string (list "Dr" "." " " "Who" "?"))` returns `"Dr. Who?"`

- ```
  (tokens->string
    (string->tokens "The quick brown fox jumps over the
      lazy dog!"))
  ```
  returns `"The quick brown fox jumps over the lazy dog!"`

- In general, for any text message m, it should be the case that
  `(tokens->string (string->tokens` m`))` returns m
  ```

Requirements for Task 3c

In this task you will use your solutions to Tasks 3a and 3b to complete the translation procedure. Your task is to define a procedure called `translation` which accepts a string representing a text message and a dictionary of synonyms, and returns the text message with any words that appear in the dictionary replaced by their synonyms. Tokens that are not in the dictionary are left unchanged.

Below are some examples you can use to test your solution to Task 3c. (The examples 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.) Note that these tests assume that dictionaries `sms->english` and `french->english` have been defined and populated as per the tests for Part 3a above.

- `(translation "This is written in English." sms->english)`
  returns `"This is written in English."` unchanged because none of the words in the message are in the dictionary
- `(translation "c u l8r, alligator!" sms->english)`
  returns `"see you later, alligator!"`
- `(translation "I'll c u @ QUT 2moro." sms->english)`
  returns `"I'll see you at QUT tomorrow."`
- `(translation "What, r u sure!? :-(" sms->english)`
  returns `"What, are you sure!? Sob!"`
- `(translation "That's gr8, mate! :-)" sms->english)`
  returns `"That's great, mate! I'm happy"`
- `(translation "Je suis un rock star, Je avez un residence, Je habite [lah], A la South of France" french->english)`
  returns `"I am a rock star, I have a residence, I live in [lah], In the South of France"` thus finally revealing the full emotional depth of Bill Wyman's lyrics

Requirements for Task 3d

Your task in this part of the assignment is to produce a document which describes your solution to Task 3b above. (This is not to say that documenting your solutions to Tasks 3a and 3c is unimportant, but Task 3b is the most challenging of the three.) To do this you should follow the step-by-step *Basic Problem Solving Process* explained in the Week 3 ITB001 tutorial. A copy of the process can be downloaded from the Lecture Materials page on the ITB001 OLT site. Use this process to document your solution to Task 3b. You should produce a file named `ITB001-3b.doc`. (Although the file is your answer to Task 3d, it describes how you developed your solution to Task 3b, so we use '`3b`' in the file name.)

You may produce this file using Microsoft Word or as a plain text document using emacs, vi, Notepad or some other text editor. However, the Online Assessment System expects the file to be named as shown above. Therefore, if you are submitting a plain text file, you may need to rename the file before submitting it to OAS, e.g., to change the extension from '`.txt`' to '`.doc`'. Ask the teaching staff if you don't know how to do this. (If you want to submit your document in some other format, such as PostScript or Portable Document Format, this is fine, but again the file has to have the name expected by OAS.)

Criterion Referenced Assessment for Part 3

Part 3 of the ITB001 text processing assignment will be marked on the *correctness*, *clarity* and *conciseness* of your solutions. With respect to the Faculty of Information Technology's Graduate Capabilities, assessment of this part of the assignment will use the following guidelines.

Graduate Capabilities	%	High achievement	Good	Satisfactory	Unsatisfactory
GC1: Knowledge and Skills	60	Program works to specification	Program has a few minor defects	Program has several minor defects	Program has major defects or is largely incomplete
GC2: Critical and Creative Thinking	20	Solutions are concise and elegant	Solutions follow sound practices	Some solutions are clumsy or convoluted	Solutions show a poor understanding of the principles
GC3: Communication	20	– Program code is well modularised and clearly commented – Documentation is clear and precise	– Program code is clearly commented – Documentation is precise but unclear in places	– Program code is commented but unclear in parts – Documentation is complete but hard to understand	– Code is confusing or inadequately commented – Documentation is largely incomplete or is misleading
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 the assignment			

Submission Process for Part 3

This part of the assignment will be submitted as two separate files, a Scheme file containing your solutions to Tasks 3a, 3b and 3c, and a document file containing your answer to Task 3d. Your program code will be tested automatically, so please adhere to the following rules.

- Complete your solutions to Tasks 3a, 3b and 3c using template file `ITB001-3.scm` from the ITB001 OLT page for this part of the assignment.

- If you have any test examples in the program file, please comment them out with semi-colons so that they do not interfere with the testing software.

- Comment out with semi-colons any procedure or code in the program file 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.

- Complete your solution to Task 3d as a separate document file, named `ITB001-3b.doc`.

- Submit both files `ITB001-3.scm` and `ITB001-3b.doc` 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 files. 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.

- Standard FIT late penalties will apply as follows: 1 day late, 10% of the given mark is deducted; 2 days late, 20% of the given mark is deducted; 3–4 days late, 30% of the given mark is deducted; 5–7 days late, 40% of the given mark is deducted; 8–10 days late, 50% of the given mark is deducted; and 11 or more days late, 100% of the given mark is deducted.

- This is an individual assignment. Please read, understand and follow QUT's guidelines regarding plagiarism. 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`).