Functional Programming Assignment

Edwin Blake, Matthew Segers, Matthew Welham

1. Introduction

An important aspect of modern medicine and forensics is the ability to compare and analyse DNA and RNA sequences. This analysis can be used in crime scene investigation, can be used to determine (with good accuracy) biological relationships and it can also be used to help narrow down and determine specific viral infections (which helps save lives!).

In this assignment, you will write code that is able to:

1. perform sequence analysis and
2. output all of the optimal sequences for two given sequences.

Strings will be used as "analogies" for DNA/RNA sequences. This analysis is done with the aim of trying to find the best alignment between two sequences (strings) with the ability to insert **spaces**[1] into one or the other at specific places in order to create a better match. Each "column" of characters is then evaluated according to some sort of scoring method (there is no agreed upon perfect scoring scheme).

For example, let us say that matching two characters is worth 2 points, mismatching two characters is worth -1 points, and matching a space and a character is worth -2 points, and you are not allowed to match two spaces (could cause infinite loops), then the following:

To match "LOL" and "POLE" the following would be an optimal match.

```
LOL_
POLE
```

You may be thinking, "matching a space is worse than mismatching" — yes, this is true. However, the two strings are of a different length, thus at some point a whitespace will have to be inserted into one or the other to allow them to match up. If you add it up:

1 mismatch	-1
2 matches (2+2)	4
1 space match	-2
Total alignment score is	1

1.1 Rules

For the purposes of this practical, you will use the following scoring system (as outlined above):

Match Type	Points
Match	2
Mismatch	-1
Space-match	-2

1.2 Side Note

Scheme denotes characters differently. You may need to take this into account, depending on how you code your solution (this is not necessarily useful information). Here is an example table of how scheme denotes characters:

[1] **Please Note**: In this brief, we have used the "_" (underscore) character to denote whitespace. This makes for better readability. Please do the same in your solution. Again, this is just for readability — it is much easier to match up two characters in a column when reading, as opposed to a character and whitespace. It is also easier for tutors to mark if there is a consistent style.

Char	Java	Scheme
Character H	'H'	`#\H`
Character Z	'Z'	`#\Z`
Character f	'f'	`#\f`

Strings are not guaranteed to be the same length. In addition you may find it easier to deal with lists rather than strings internally in your functions. There are existing Scheme functions that will convert between these (see Section 4 below). Please ensure however that all your functions accept strings as arguments and return strings as results where indicated.

2. What to Hand In

Please hand in your solution as an archive named with your **name** and **student number**.

Your Code

Your code has to able to run on one of the two interpreters: Gambit or Racket. You may load the file "simply.scm" on Vula to help you with this practical if you wish.

All your code must comply with acceptable Scheme style rules (if in doubt refer to your slides). Failure to choose appropriate names, create useful comments, etc. can lead to a loss of up to 10% per question. It is vital that you ensure that your code works. Code that does not work can get at most 40% and most likely will get much less.

Evidence of Unit Testing

In addition to your code, or perhaps as comments inside the code, you need to give evidence of thorough testing of the code. In other words for every function there must be test cases and resultant outputs. If you claim that a function is tail recursive, then supply a short trace output that shows this.

Only if you chose to have a separate file of these test results, transcripts and traces then submit that as plain text, pdf or word file (with headings for the various parts).

2.1 README

Please pay attention when writing your README. There are always a number of different ways of running and interpreting code. You cannot assume that he way you have chosen to complete this assignment is obvious to tutors. Please include all information that is relevant to the successful execution of your solution. This includes, but is not necessarily limited to

- operating system
- flavour of Scheme (GAMBIT or RACKET)
- version of interpreter

2.2 You May Not Copy or Share Any Code

Please note that while you may discuss ideas with others you may not copy or share any code.

3. Questions

The total for all the questions is a bit over 100 but you will be marked out of 100, so there is a built-in bonus to doing everything. You will not be given more than 100% however.

3.1 Question One [10]

To begin with, you will write a function that scores the alignment of two chars according to the scoring scheme specified above. So this function will take two characters and return an integer.

Example

Inputs		Output
"h"	"h"	2
"h"	"k"	-1
"h"	"_"	-2

3.2 Question Two [15]

The following function scores the alignment of two strings. In other words, given two strings as input, it compares each "column" of characters and accumulates a score for that alignment, based on the scoring scheme outlined above.

```
(define (alignment-score-not-tail string-one string-two)
  (if (and (not (= (string-length string-one) 0))
           (not (= (string-length string-two) 0)))
      (+ (char-scorer (string-ref string-one 0)
                      (string-ref string-two 0))
         (alignment-score-not-tail (substring string-one 1)
                                   (substring string-two 1)))
      0))
```

Example

Inputs		Output
"Hello"	"_ _low "	-4

Your job is to re-implement the above function **using tail recursion**. This function takes two strings and returns an integer.

3.3 Question Three [25]

Next, you should code a function that determines what the best possible match score is, given two strings. For this you will want to make use of the **max** function, and using **let** (which allows you to store values) may make your code more readable. **This is the part of your code that performs the sequence analysis!**

This function takes two strings and returns an integer.

Example

Input 1	H	E	L	L	O	_	
Input 2	B	_	L	_	O	W	
Calculation	-1	-2	2	-2	2	-2	= -3

Note: the final line is for illustrative purposes, and not part of the input. The example given is one of many optimal sequences.

Example output: -3

Hint: This will be a recursive function. Notice that at each step, it is possible for the program to have one of three choices. It can *either* try to match the characters *or* insert a space in place of one *or* the other and try to match that.

Thus there is a recursive relationship on "dummy`-score`" that looks something like this:

```
(max
  (+ (score-char head1 head2)    ;; Match characters
     (dummy-score tail1 tail2))  ;; and score the rest
  (+ (score-char head1 "_" )      ;; Now add a space to string 2
     (dummy-score tail1 string2)) ;; score the rest
  (+ (score-char "_" head2)       ;; Add a space to string 1
     (dummy-score string1 tail2)) ;; and score the rest
 ))
```

Note: The point of the max is to choose the best one of the three choices.

3.4 Question Four [15]

You will now need to write a function that takes two chars (x and y, say) and a **list of pairs of strings** (e.g. '((“one” “two”) (“three” “four) (“five” “six”)) as input, and that returns modified list of pairs of strings: for each string pair in the list, you will need to add x to the front of the first string and y to the front of the second string.

Example input:

```
"a" "b" '(("one" "two") ("three" "four") ("five" "six"))
```

Example output:

```
'(("aone" "btwo") ("athree" "bfour") ("afive" "bsix"))
```

Example input:

```
"l" "o" '(("lo" "w_") ("lo" "_w"))
```

Example output:

```
(("llo" "ow_") ("llo" "o_w"))
```

3.5 Question Five [20]

The next step is to write a function that takes **both** a scoring function **and** a list of pairs of strings as input, and returns a modified **list of pairs of strings**: the returned list should contain all the *optimal* string pairs from the input, scored according to the input function.

That is, the input scoring function should be the one like that from Question two which takes two strings and evaluates them to return a score (integer). The string pairs that evaluate to the maximum score across the list should then be returned by this new function.

Example input:

```
'( ("hello" "b_low") ("hello_" "b_l_ow") ("hello" "_blow") ("hello"
"blow") ("h_e_llo" “bl_o__w”) )
```

Example output:

```
( ("hello" "b_low") ("hello_" "b_l_ow") ("hello" “_blow") )
```

Explanation of example

In the above example, the function takes a **list of pairs of strings** as shown in the example input. It **also** takes in a **function** (yes, the function takes a function as a parameter). It uses this function that it takes in as a means to **evaluate** the **list of pairs of strings**. It then returns a list of pairs of strings containing all of the pairs of strings that had the highest match-score based on the function it was given to evaluate them with. In

other words, `(("hello" "b_low") ("hello_" "b_l_ow") ("hello" "_blow"))` all had the same score of -3, but `("h_e_llo" "bl_o__w"))` has a score of -12, thus it is dropped from the list.

3.6 Question Six [30]

Finally, we put it all together. You now need to write a function that takes two strings as input and returns a list of all optimal pairs of strings. The general structure of this function will likely resemble that of Question Three's, and you will make use of the functions from Questions Two, Four and Five. Have fun! (**This is the part of your code that actually generates the optimal sequences!**).

Recall:

1. In Question Two (3.2) you wrote the function to score two strings, it returned an integer score.
2. In Question Three (3.3) you explored the structure that recursively goes through all options for checking the two strings, just as will be doing here.
3. In Question Four (3.4) we wrote a function that will add two different characters to the head of pairs of strings in a list. These two characters are the options mentioned in Question 3, namely, two characters, or character and space or space and character.
4. Question Five finally filtered out the non-optimal pairs.

What you have to do now is go through all possible versions of the pair of strings with space(s) inserted in every possible position. This is a very, very, inefficient algorithm!

Hint: to debug the system, consider what a scoring function that always returns zero will give as output here.

Example input:

```
"hello" "blow"
```

Example output:

```
(("hello" "b_low")
 ("hello_" "b_l_ow")
 ("hello_" "b__low")
 ("hello" "_blow")
 ("hello_" "_bl_ow")
 ("hello_" "_b_low")
 ("hello_" "__blow"))
```

4. General Hints

1. It might be useful to know that we made use of lambdas, and the built in scheme functions: map, filter, append and apply.
2. You will probably want to visit this **http://docs.racket-lang.org/reference/strings.html**
3. In particular, you may (and quite possible may not) find the following useful:

```
string-length        list->string        pair?
string-ref           substring           equal?
string-append        make-string
string->list         filter
```

5. General Notes

The problem you are solving is closely related to the maximum common subsequence problem (**http://en.wikipedia.org/wiki/Longest_common_subsequence_problem**). You may notice that, once you

complete the assignment, the solution is not very fast. This is because of the depth to which the solution recurses, and because it re-calculates a lot of things unnecessarily (look at the code and think about it). The way to solve this is to use Dynamic Programming - however, this is beyond the scope of this assignment. You will cover dynamic programming next semester. However, if you are interested you can look here:

http://en.wikipedia.org/wiki/Dynamic_programming

http://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm