CS 105-03: Introduction to Programming: Scheme
Final Exam

Friday December 12, 1997

1 Warming Up

(*a*) [4 points] Write a function `(copies n wd)` that takes two arguments: a number and word and returns a sentence which contains n copies of the word wd. For example `(copies 3 'yeah)` ⇒ `(yeah yeah yeah)`.

(*b*) [4 points] Write a function `zip` that takes two lists and zips them, i.e. contsructs one list with elements from the two lists alternating. If one of the lists runs out of elements before the other, just append the remaining elements. Example: `(zip '(1 2 3 4) '(5 6 7 8 9 10 11))` ⇒ `(1 5 2 6 3 7 4 8 9 10 11)`.

2 Abstract Data Types

(*a*) [5 points] Explain what an abstract data type (ADT) is and explain the function of constructors and selectors in ADTs. Illustrate your explanation with at least one of the examples we have seen in class.

(*b*) [5 points] What are the advantages of using ADTs (think in terms of users and implementation).

3 Caesar ciphers

We want to write functions that encode words and sentences using the Caesar cipher. In the Caesar cipher a word is encoded by rotating the letters in the alphabet a given number of times. For example if we rotate by one letter, then CAESAR becomes DBFTBS. If we rotate the word ZOO by four letters it becomes DSS (note that rotating Z by one letter gets us back to A).

You may assume that there is a function `rot` which takes a letter and rotates it by one letter, i.e. `(rot 'A)` ⇒ `B`, `(rot 'B)` ⇒ `C`, etc, finally `(rot 'Z)` ⇒ `A`.

(*a*) [5 points] Write a function `caesar-word` which takes a word and a number and encodes the word by rotating each letter in the word by the given number, e.g. `(caesar-word 'CAESAR 1)` ⇒ `DBFTBS` and `(caesar-word 'ZOO 4)` ⇒ `DSS`.

(*b*) [5 points] Write a function `caesar` which takes a sentence and a number and encodes each word according to the Caesar cipher using the function `caesar-word`. For example: `(caesar '(CAESAR IS HUNGRY) 1)` ⇒ `(DBFTBS JT IVOHSZ)`. (**Hint**: use `every` and `lambda`). (**Note:** you can solve part b even without having solved part a just assuming `caesar-word` works as described above).

(Please turn over)

(*c*) [2 points] Write a function `caesar-decode` that takes an encoded sentence and the number it was encoded with and returns the original sentence, e.g. `(caesar-decode '(DBFTBS JT IVOHSZ) 1)` $\Rightarrow$ `(CAESAR IS HUNGRY)`. (**Hint**: there is a solution in two lines.)

4 Sorting lists

You will implement the sorting algorithm Quicksort. The algorithm works in two steps. First it splits the list which is to be sorted into two lists and sorts these two lists recursively. In the second step it puts the results back together. The difference to mergesort lies in the way the list is split and is put back together. In part a you will implement an easy version of the algorithm. For extra credit you can solve part b which does the splitting of lists in a more sophisticated way.

(*a*) [10 points] The quicksort algorithm sorts a list by taking the first element of the list and splitting the list into two lists: those elements which are at most the first element, and those elements which are larger. For example (3 4 2 1 6) would be split into (3 2 1) and (4 6). Then both lists are sorted recursively, yielding (in the example) (1 2 3) and (4 6), and put back together: (1 2 3 4 6). Make sure your base case(s) work(s).

(*b*) [**Extra credit**: 5 points] As above, but instead of choosing the first element of the list as the pivot element (the element according to which elements are put into the first or second list), choose a random element in the list. Remember that `(random n)` will give you a random number between 0 (inclusive) and n (exclusive).

5 Vectors

Remember that for dealing with vectors you have the following constructors and selectors:

- `(make-vector n)` : function that returns a vector of length n.
- `(vector-length v)` : function that returns the length of vector v.
- `(vector-ref v i)` : function that returns the (i+1)-st element of vector v (for example `(vector-ref v 0)` is the first element of the vector v).

(*a*) [5 points] Write a function `max-vector` that computes the maximum value in a vector, e.g. `(max-vector #(2 -5 7 4))` $\Rightarrow$ `7` (remember you are dealing with vectors here, not with lists, i.e. things like map etc. won't work).

(*b*) [5 points] Write a function `vector->list` that converts a vector into a list containing the same elements, e.g. `(vector->list #(2 5 4 3))` $\Rightarrow$ `(2 5 4 3)`.