COMP 302: Programming Languages and Paradigms

Winter 2015: Assignment 1

This assignment is due on Thursday, January 29, 2015 at midnight. It must be submitted using MyCourses**.** The cutoff time is automated. Assignments submitted within the next hour will be accepted but marked late. The system will not accept any attempts to submit after that time.

Problem 1: We can represents dates as a triple of type int*int*int. Valid dates would be of the form (2015, 1, 16). Of course, not all triples are valid. In this problem you are asked to implement functions that operate on dates. Your functions have to work correctly for valid dates, not necessarily for arbitrary triples.

Problem 1.1 (10 marks)
Write a function last_day that takes a triple and evaluates to the last day of the month indicated by the second value of the triple. It should check that the month is a value between 1 and 12 and raise an exception otherwise. Make sure to take leap years into account.

Problem 1.2 (5 marks)
Write a function valid_date that takes a triple and evaluates to true if the triple represents a valid date and false otherwise.

Problem 1.3 (5 marks)
Write a function next_day that takes a triple and returns a triple representing the next day's date.

Problem 1.4 (5 marks)
Write a function precedes that takes two dates and evaluates to true if the first argument comes before the second and false otherwise.

Problem 1.5 (10 marks)
Write a function that takes a list of dates and returns the earliest date in the list. You should write a helper function that determines whether a date precedes another date.

Problem 2: Given two lists ["alice", "bob", "carol", "dave"] and [2, 3, 11, 7, 11], it is useful to construct a list of pairs (["alice", 2), ("bob", 3), ("carol", 11), ("dave, 7)"]

Problem 2.1: (5 marks)
Write a function

```
zip : int list * string list -> (int * string) list
```

that does this. If the arguments have different lengths, the length of the result should be the length of the smaller list.

Problem 2.2 (5 marks)
Write a function

```
unzip : (int*string) list -> : int list * string list
```

that does the opposite.

Problem 2.3: (15 marks)
Prove that

```
For every lst : (int * string) list, zip(unzip lst) == lst
```

Use the method of structural induction on lists for your proof. In the proof state when you are use the fact that expressions are valuable and explain how you know the expressions are valuable. You can assume that zip and unzip are total.

Problem 2.4 (5 marks)
Show that it is not true that `unzip (zip (lst1, lst2) == (lst1, lst2)`

Problem 3: The prefix-sum of a list of integers, lst, is a list psum where the ith element is the sum of the first i elements of lst. For example

```
prefixSum [] == []
prefixSum [2, 3, 11, 7] == [2, 5, 16, 23]
```

Problem 3.1 (5 marks)
Implement the function prefixSum. You must use the incr function defined in class to add a value to each element of a list. The implementation is simple but inefficient since it takes time $O(n^2)$.

Problem 3.2 (5 marks)
Suppose PS(n) is the amount of work done by this prefixSum function on a list of length n. Give a recurrence relation for this function and show that is is $O(n^2)$. You may assume that incr takes time kn, for some constant k, to evaluate to a value.

Problem 3.3 (10 marks)
Write a function prefixSumFast that computes the prefix sum in linear time. It should use a helper function that uses an additional argument to compute the prefix sum.

Problem 3.4 (5 marks)
Give a recurrence for PSF(n), the amount of work done by prefixSumFast on a list of length n. Show that it is an O(n) algorithm.