Middle East Technical University | Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring '2011-2012

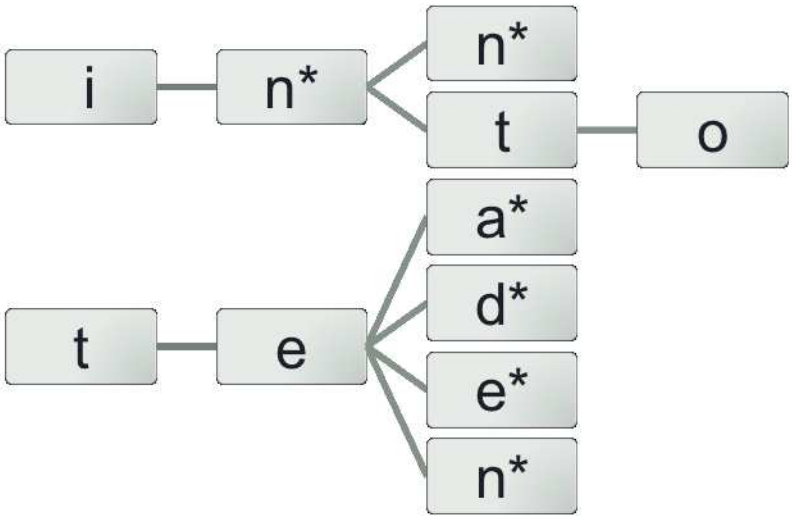
Homework 3

Due date: 22 April 2012, Sunday, 23:55

1 Objectives

In this homework you are going to create a module that implements a Dictionary Abstract Data Type. You can think of it as the data structure that is used by the spell checker in your word processor. When you are editing a document, if the word you typed doesn't exist in the dictionary, it gets underlined, so that you can correct it later. In other words, the Dictionary ADT holds the list of words in a language without their definitions.

Since checking if a word exists in the dictionary, and inserting a new word to the dictionary must be fast, you must implement it using a data structure called Prefix Tree or Trie. Below is a sample dictionary implemented using a Prefix Tree. This dictionary contains the following words: in, inn, into, tea, ted, tee, ten.



During check of a list of symbols in prefix tree lookup, you start traversal from the root and based on the next symbol in the list you follow one of the branches. If no branches match the next symbol, search fails, thus the list is not included in the Dictionary. When there is no symbol left in the list and the node currently visited is final (shown as '*' in the figure) the lookup is successful.

2 Tasks

You are going to write your module in a file called "Dictionary.hs". The following definitions must be at the start of your module.

```
module Dictionary(Dictionary, empty, check, insert, merge, subs, toList, fromList) where \\
data Dictionary a = ...
```

In your module, you are going to implement the following functons:

`empty :: Dictionary a`
: This function will return an empty dictionary.

`check :: Dictionary a -> [a] -> Bool`
: This function will check if a given list (containing chars or other orderable data type) exists in the dictionary. If it exists, check will return true.

`insert :: Dictionary a -> [a] -> Dictionary a`
: This function will insert a given list (containing chars or other orderable data type) to the dictionary in the first argument. Then it will return the new dictionary.

`merge :: Dictionary a -> Dictionary a -> Dictionary a`
: This function will merge two dictionaries and return a new dictionary containing the words in both dictionaries.

`subs :: Dictionary a -> Dictionary a -> Dictionary a`
: This function will subtract the second dictionary from the first dictionary and return a dictionary that contains all the words that don't exist in the second dictionary.

`toList :: Dictionary a -> [[a]]`
: This function will return a list containing all the words in the dictionary.

`fromList :: [[a]] -> Dictionary a`
: This function will create a dictionary containing all the words in the given list, by successively inserting the words in the list to an empty dictionary.

`show`
: You are also going to write a show function for the dictionary class. See the sample I/O part for how you are going to format your string. This part is very important, homeworks without a proper show function will most probably get zero from black box evaluation.

Remark: Note that 'merge' and 'subs' operations are best carried out in prefix tree structures. Converting tree to list then inserting or deleting from first list is not an efficient solution. Your solutions will be tested on large dictionaries and if slow, you might loose points.

3 Sample I/O

```
Hugs> :l Dictionary.hs
Dictionary> insert empty "tee"
␣
␣␣␣␣'t'-->␣␣
␣␣␣␣␣␣␣␣'e'-->␣␣
␣␣␣␣␣␣␣␣␣␣␣␣'e'-->␣*
```

```
Dictionary> insert (insert empty "tee") "tea"
␣
␣␣␣␣'t'-->␣␣
␣␣␣␣␣␣␣␣'e'-->␣␣
␣␣␣␣␣␣␣␣␣␣␣␣'a'-->␣*
␣␣␣␣␣␣␣␣␣␣␣␣'e'-->␣*

Dictionary> merge (insert (insert empty "tee") "into") (insert empty "tea")
␣
␣␣␣␣'i'-->␣␣
␣␣␣␣␣␣␣␣'n'-->␣␣
␣␣␣␣␣␣␣␣␣␣␣␣'t'-->␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣'o'-->␣*
␣␣␣␣␣'t'-->␣␣
␣␣␣␣␣␣␣␣'e'-->␣␣
␣␣␣␣␣␣␣␣␣␣␣␣'a'-->␣*
␣␣␣␣␣␣␣␣␣␣␣␣'e'-->␣*

Dictionary> subs (merge (insert (insert empty "tee") "into") (insert empty "tea")) (insert empt
␣
␣␣␣␣'i'-->␣␣
␣␣␣␣␣␣␣␣'n'-->␣␣
␣␣␣␣␣␣␣␣␣␣␣␣'t'-->␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣'o'-->␣*
␣␣␣␣␣'t'-->␣␣
␣␣␣␣␣␣␣␣'e'-->␣␣
␣␣␣␣␣␣␣␣␣␣␣␣'a'-->␣*

Dictionary> toList (merge (insert (insert empty "tee") "into") (insert empty "tea"))
["into","tea","tee"]
Dictionary> fromList ["inn", "tee", "tea", "into", "ted"]
␣
␣␣␣␣'i'-->␣␣
␣␣␣␣␣␣␣␣'n'-->␣␣
␣␣␣␣␣␣␣␣␣␣␣␣'n'-->␣*
␣␣␣␣␣␣␣␣␣␣␣␣'t'-->␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣'o'-->␣*
␣␣␣␣␣'t'-->␣␣
␣␣␣␣␣␣␣␣'e'-->␣␣
␣␣␣␣␣␣␣␣␣␣␣␣'a'-->␣*
␣␣␣␣␣␣␣␣␣␣␣␣'d'-->␣*
␣␣␣␣␣␣␣␣␣␣␣␣'e'-->␣*

Dictionary> :q
```

4 Regulations

1. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations.

2. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.

3. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications. Your codes will be tested on inek machines, using Hugs. Make sure your code runs correctly on them before submitting.

4. **Restrictions:** You must implement Dictionary using a prefix tree. Although evaluation will be blackbox we will also examine your codes to make sure you implement Dictionary using a prefix tree. You are not allowed to import or use anything from the Haskell library, except Prelude.

5. **Submission:** You must submit your codes via COW. You must submit a file named `Dictionary.hs` that contains the implementation for your Dictionary module.

6. **Late Submission:** See the Course Webpage for Late Submission policy