Binary search trees in ML

Mandatory problem no 3 in INF3110/4110

To be delivered before noon 15th November 2004

In this problem we want to work with *binary search trees* containing *words* (i.e., `strings`). The search trees are defined as

```
datatype bintree = NOLEAF |
                   LEAF of string |
                   NODE of (string*bintree*bintree);
```

The `NOLEAF` constructor is used when a `NODE` has only one child, as in this little tree with only two words:

```
NODE("bravo",
     LEAF("alpha"),
     NOLEAF)
```

Functions

You shall write the following functions that operate on `bintrees`:

`n_in_tree(t)` that computes how many words are stored in the binary search tree `t`.

`is_in_tree(k,t)` returns `true` if the word `k` can be found in the tree `t`.

`max_in_tree(t)` finds the "biggest" word (i.e., the last in alphabetical order based on the standard SML-operator <) in `t`.

`add_to_tree(k,t)` returns a new tree consisting of an old tree `t` with `k` added to it, unless the word already is in the tree. (In other words, there are no duplicates in the tree.)

`create_tree(wl)` builds a binary search tree from the words in the `string` list `wl`. Any duplicates should only occur once.

`flatten_tree(t)` is the inverse operation: it will create a `string` list of the words in the tree `t` in infix (i.e., sorted) order.

`fold_tree(f,t)` will use the function `f` to fold the elements of `t` in infix (i.e., sorted) order. Use left-to-right folding. For example, when using the test data mentioned below,

```
fold_tree((fn (s,t) => s ^ t),
          Trees_tree);
```

will produce

```
val it =
"aagainstallandarearmsasatbosombreastbutbycandayearth'sflowingfoolsgodh#"
: string
```

which is the concatenation of all the words. (The "#" at the end of the line indicates that only the initial part of the string is shown.)

Test data

On the file ~inf3110/oblig-3/testdata.sml can be found the following test data:

```
val Trees =
["i", "think", "that", "i", "shall", "never", "see",
 "a", "poem", "lovely", "as", "a", "tree",
 "a", "tree", "whose", "hungry", "mouth", "is", "prest",
 "against", "the", "earth's", "sweet", "flowing", "breast",
 "a", "tree", "that", "looks", "at", "god", "all", "day",
 "and", "lifts", "her", "leafy", "arms", "to", "pray",
 "a", "tree", "that", "may", "in", "summer", "wear",
 "a", "nest", "of", "robins", "in", "her", "hair",
 "upon", "whose", "bosom", "snow", "has", "lain",
 "who", "intimately", "lives", "with", "rain",
 "poems", "are", "made", "by", "fools", "like", "me",
 "but", "only", "god", "can", "make", "a", "tree."];

val Trees_tree = create_tree Trees;

n_in_tree(Trees_tree);
is_in_tree("tree", Trees_tree);
is_in_tree("trees", Trees_tree);
max_in_tree(Trees_tree);

fold_tree((fn (s,t) => s ^ ", " ^ t), Trees_tree);
```

Delivery

Your answer should be sent to your tutor by e-mail and contain

1. the SML program with enough comments to make it easily readable and
2. a printout of a run on the test data mentioned above.