Programowanie funkcyjne – laboratorium 5
4.11.2009 r.

1. Zdefiniujmy polimorficzny typ drzew binarnych.

   ```
   type 'a tree = Leaf | Node of 'a * 'a tree * 'a tree ;;
   ```

   Przypomnij sobie definicje funkcji `size`, `depth`, `full` z zajęć o drzewach. Sprawdź, że ocaml akceptuje te definicje bez żadnych zmian. Zastanów się, jakie powinny być typy tych funkcji i sprawdź, czy takie typy proponuje interpreter.

2. Napisz funkcję `fold_tree`, działającą podobnie jak `fold` działa na listach.

3. Przy pomocy funkcji `fold_tree` zaimplementuj jeszcze raz:

   - `elem_list` tworzącą posortowaną listę elementów drzewa BST
   - `mirror` tworzącą lustrzane odbicie drzewa
   - `size` obliczającą wielkość drzewa
   - `contains` sprawdzającą czy drzewo (niekoniecznie BST) zawiera dany element

4. Napisz przy pomocy `fold_tree` funkcję sprawdzającą, czy drzewo jest pełne (tzn. czy wszystkie liście są jednakowo odległe od korzenia).

5. Zdefiniuj funkcję `tree_to_string` przekształcającą drzewo na napis. Na przykład

   ```
   tree_to_string (Node (1, Leaf, Node (2, Leaf, Leaf))) = "<1<2>>"
   ```

   Użyj `fold_tree` i `string_of_int`.

6. Wypisz drzewo ładniej. Zamiast `<<<1>2<4>>3<<1>2<4>>>` wypisz na przykład:

   ```
     3
   .....
   2   2
  ... ...
  1 4 1 4
   ```

   Wskazówka: napisz funkcję przekształcającą drzewo na listę napisów, po jednym dla każdej linii rysunku. Do wypisania pojedynczej linii użyj `print_string`. Do wypisania listy – `fold`.

7. Zdefiniujmy polimorficzne drzewa BST. Oprócz drzewa potrzebna jest nam funkcja porównująca elementy:

   ```
   type 'a bst_tree = ('a -> 'a -> bool) * 'a tree;;
   ```

   Napisz nowe wersje funkcji

   - `bst_insert` wstawiającej element do drzewa,
   - `bst_insert_list` wstawiającej listę elementów do drzewa (użyj funkcji `fold`),
   - `elem_list` zwracającej listę wierzchołków drzewa BST w porządku infiksowym,
   - `bst_sort` używającej drzew BST do posortowania listy.

8. Przypomnij sobie rozwiązanie jednego z poprzednich zadań, gdzie trzeba było zdefiniować typ danych reprezentujący drzewa dowolnego (skończonego) stopnia. Zdefiniuj dla takich drzew odpowiedniki procedur `map`, `filter` i `fold`.

   W procedurze `filter`, jeżeli odrzucamy jakiś wierzchołek, to odrzucamy również wszystkich jego potomków. Odpowiednik procedury `fold` powinien być sparametryzowany dwiema funkcjami: Jedna powinna działać "w poziomie", kumulując wyniki policzone dla poddrzew zakorzenionych w synach danego węzła. Druga powinna działać "w pionie", okreslając wynik dla poddrzewa zakorzenionego w danym węźle, na podstawie wartości przechowywanej w tym węźle oraz wyniku skumulowanego dla poddrzew zakorzenionych w jego synach.

9. Implementacja słownika. Słownik to typ danych, który przechowuje pary `(unikalny klucz, wartość)`. Dostęp do wartości odbywa się za pomocą kluczy. Zaimplementuj słownik za pomocą drzew BST. Reprezentacja powinna być parametryzowana typami kluczy i wartości. Zaimplementuj funkcje:

   - `empty` tworzącą pusty słownik.
   - `add` dodającą parę `(klucz, wartość)` do słownika.
   - `find` zwracającą element znajdujący się pod kluczem `k`. Funkcja ma zwracać element typu `'v option`:

     ```
     type 'v option = Some of 'v | None;;
     ```

     Jeśli słownik zawiera parę `(k,v)`, zwracana jest wartość `Some (v)`. Jeśli słownik nie zawiera pary `(k,v)`, zwracana jest wartość `None`.
   - `contains` zwracającą `true` jeśli słownik zawiera podany klucz, `false` w przeciwnym przypadku.
   - `update` zmieniającą wartość przechowywaną pod kluczem `k`.
   - `pairs` zwracającą listę par należących do słownika, uporządkowaną w rosnącej kolejności kluczy.