Programowanie funkcyjne – laboratorium 6
18.11.2009 r.

1. Dana jest sygnatura zbioru uporządkowanego:

```
module type ORD_SET =
  sig
    type 'a compare = 'a -> 'a -> bool
    type 'a set

    val empty : 'a compare -> 'a set
    val insert : 'a set -> 'a -> 'a set
    val from_list : 'a compare -> 'a list -> 'a set
    val to_list : 'a set -> 'a list
  end
```

   Zaimplementuj strukturę `BSTSet : ORD_SET` używając polimorficznych drzew BST z poprzednich zajęć.

2. Napisz program, który

   (a) odczyta zawartość pliku jako listę znaków (`char list`)

   (b) przekształci tę listę na listę słów, ignorując znaki inne niż litery i zamieniając litery na małe

   (c) posortuje słowa używając modułu BSTSet (albo innej implementacji zbiorów uporządkowanych elementów)

   Wskazówki:

   (a) Funkcja zwracająca zawartość pliku jako listę znaków wygląda tak:

```
let read_file name =
  let file = open_in name in
  let rec chars l =
      try chars (input_char file :: l)
      with End_of_file -> List.rev l
  in
      chars [];;
```

   (b) Do zrobienia słów z listy znaków mogą przydać się następujące funkcje:

```
Char.code : char -> int,
(*  zwraca kod ASCII znaku, *)

Char.escaped : char -> string,
(*  'Char.escaped c' zwraca napis zawierający 'c', przy czym specjalne
znaki są traktowane specjalnie, np. Char.escaped '\t'  =  "\\t", *)

Char.lowercase : char -> char,
(*  'Char.lowercase c' zwraca mały odpowiednik litery 'c', *)

fold_left (^) "" : string list -> string
```

   (c) Do porównywania stringów służy funkcja 'String.compare'. Można też użyć przeciążonej operacji `(<=)`.

3. Zaimplementuj moduł słownika o następującej sygnaturze:

```
module type DICT =
  sig
  type ('k, 'v) dict
    (* typ słowników z kluczami typu 'k i wartościami typu 'v *)

  type 'a compare = 'a -> 'a -> bool
    (* typ funkcji porównujących wartości typu 'a *)

  val empty : 'k compare -> ('k,'v) dict
    (* zwraca pusty słownik używający danej funkcji porównującej *)

  val add : ('k,'v) dict -> 'k -> 'v -> ('k, 'v) dict
    (* dodaje parę (klucz, wartość) do słownika *)

  val find : ('k,'v) dict -> 'k -> 'v option
    (* 'find d k' zwraca 'Some v', jeśli 'd' zawiera parę '(k,v)', 'None'
       w przeciwnym przypadku. *)

  exception Undefined

  val update : ('k,'v) dict -> 'k -> ('v -> 'v) -> ('k,'v) dict
    (* 'update d k f' zgłasza wyjątek 'Undefined', jeśli słownik 'd' nie
       zawiera klucza 'k'. Jeśli 'd' zawiera parę '(k,v)', zwracany jest
       słownik powstały przez zastąpienie jej przez '(k, f v)'. *)

  val pairs : ('k,'v) dict -> ('k * 'v) list
    (* 'pairs d' zwraca listę par (klucz, wartość) posortowaną
       według kluczy *)
end
```

4. Używając słownika zaimplementowanego w zadaniu 3 i elementów zadania 2 napisz program, który

   (a) odczyta zawartość pliku w postaci listy znaków

   (b) policzy, ile razy każdy znak występuje w pliku

   (c) zwróci listę par (znak, krotność) posortowaną malejąco względem krotności

   Wskazówki:

   (a) Do liczenia krotności użyj słownika z parami (znak, liczba).

   (b) Do posortowania po krotnościach możesz użyć słownika z parami (liczba, znak).