ZESTAW ZADAŃ TRENINGOWYCH - SERIA 3

📋 ZADANIA Z FOLD_LIST (10 zadań)

1. `first_n : int -> 'a list -> 'a list`

Weź pierwsze n elementów z listy używając `List.fold_right`. Dla n > długość listy zwróć całą listę.

2. `drop_last : 'a list -> 'a list`

Usuń ostatni element z listy używając `List.fold_right`. Dla pustej listy zwróć pustą listę.

3. `index_of : 'a -> 'a list -> int option`

Znajdź indeks pierwszego wystąpienia elementu (zaczynając od 0) używając folda.

4. `split_at : int -> 'a list -> 'a list * 'a list`

Podziel listę na pozycji n używając folda. Zwróć parę (pierwsze n, reszta).

5. `count_occurrences : 'a -> 'a list -> int`

Policz wystąpienia elementu na liście używając folda.

6. `alternate_sum : int list -> int`

Oblicz sumę naprzemienną (1 - 2 + 3 - 4 + ...) używając folda z licznikiem.

7. `group_consecutive : int list -> int list list`

Pogrupuj kolejne jednakowe elementy używając folda. `[1;1;2;2;2;3]` → `[[1;1];[2;2;2];[3]]`

8. `is_sorted : int list -> bool`

Sprawdź czy lista jest posortowana rosnąco używając folda.

9. `merge_sorted : int list -> int list -> int list`

Połącz dwie posortowane listy w jedną posortowaną używając folda (na jednej z list).

10. `unique_elements : int list -> int list`

Zwróć listę unikalnych elementów (w kolejności pierwszego wystąpienia) używając folda.

🌳 ZADANIA Z FOLD_TREE (5 zadań)

Dla typu: `type 'a tree = Leaf | Node of 'a tree * 'a * 'a tree`

1. `tree_size : 'a tree -> int`

Policz łączną liczbę węzłów i liści w drzewie używając folda.

2. `tree_product : int tree -> int`

Oblicz iloczyn wszystkich wartości w drzewie używając folda (1 dla pustego).

3. `tree_any : ('a -> bool) -> 'a tree -> bool`

Sprawdź czy jakikolwiek element spełnia predykat używając folda.

4. `tree_level_sums : int tree -> int list`

Oblicz sumy na każdym poziomie drzewa używając folda. Zwróć listę sum dla każdego poziomu.

5. `tree_replace : 'a -> 'a -> 'a tree -> 'a tree`

Zamień wszystkie wystąpienia pierwszego elementu na drugi używając folda.

🧮 ZADANIA Z INDUKCJI (5 zadań)

DOWODY (3 zadania):

1. Udowodnij że dla każdej listy `xs` i funkcji `f`:

`length (map f xs) = length xs`

2. Udowodnij że dla każdego drzewa `t`:

`tree_size (mirror_tree t) = tree_size t`

3. Udowodnij że dla każdych list `xs` i `ys`:

`length (xs @ ys) = length xs + length ys`

SFORMUŁOWANIE ZASAD (2 zadania):

4. Sformułuj zasadę indukcji dla typu:

```ocaml
type 'a nested_list =
  | Item of 'a
  | List of 'a nested_list list
```

5. Sformułuj zasadę indukcji dla typu:

```ocaml
type binary_op = Add | Sub | Mul | Div
type formula =
  | Const of int
  | BinOp of binary_op * formula * formula
```

🔤 ZADANIA Z TYPOWANIA (10 zadań - podaj typ)

1. `fun f lst -> List.filter (fun x -> not (f x)) lst`
2. `fun x -> match x with | [] -> 0 | [a] -> a | a :: b :: _ -> a + b`
3. `fun f g -> fun x -> fun y -> f (g x) (g y)`
4. `fun opt def -> match opt with | Some x -> x | None -> def`
5. `fun lst -> List.fold_left max (List.hd lst) lst`
6. `fun f lst1 lst2 -> List.map2 f lst1 lst2`
7. `fun pred -> fun lst -> List.partition pred lst`
8. `fun f -> List.fold_right (fun x acc -> f x :: acc)`
9. `fun x y z -> if x > y then z x else z y`
10. `fun lst -> List.fold_left (fun acc x -> x :: acc) []`

🎯 ZADANIA ODWROTNE - TYP → FUNKCJA (10 zadań)

Napisz funkcję o podanym typie:

1. `('a -> 'a) -> 'a -> 'a`
2. `int -> int list -> int list`
3. `('a -> 'b) -> 'a option -> 'b option`
4. `'a list -> 'a list -> 'a list`
5. `('a -> 'a -> int) -> 'a list -> 'a list`
6. `bool -> 'a -> 'a -> 'a`
7. `('a -> bool) -> ('a -> bool) -> ('a -> bool)`

8. `'a -> ('a -> 'b) -> 'b`

9. `('a -> 'b -> 'c) -> ('a * 'b) -> 'c`

10. `int list -> (int -> bool) -> int option`

🌲 ZADANIA Z OPERACJI NA DRZEWACH (5 zadań)

Dla typu: `type 'a tree = Leaf | Node of 'a tree * 'a * 'a tree`

1. `tree_filter : ('a -> bool) -> 'a tree -> 'a tree`

Usuń wszystkie węzły które nie spełniają predykatu (zastąp je liśćmi).

2. `tree_depth_at : 'a -> 'a tree -> int option`

Znajdź głębokość pierwszego wystąpienia elementu (0 dla korzenia). None jeśli nie ma.

3. `tree_count_at_level : int -> 'a tree -> int`

Policz ile węzłów (nie liści) jest na poziomie n (korzeń ma poziom 0).

4. `balance_simple : 'a list -> 'a tree`

Stwórz zbalansowane drzewo z posortowanej listy (środkowy element jako korzeń).

5. `tree_fold_postorder : ('b -> 'a -> 'b -> 'b) -> 'b -> 'a tree -> 'b`

Zaimplementuj fold który przechodzi drzewo w kolejności post-order (lewe, prawe, korzeń).

📝 ZADANIA Z GRAMATYK BEZKONTEKSTOWYCH (2 zadania)

1. Napisz gramatykę bezkontekstową dla języka:

```
L = {w w^R | w ∈ {a,b}*}
```

(słowa które są palindromami o parzystej długości, np. "", "aa", "abba", "baab")

2. Napisz gramatykę dla prostego języka SQL SELECT:

```
Przykłady:
"SELECT x FROM table"
"SELECT x, y FROM table WHERE x > 5"
"SELECT * FROM table WHERE x = y AND z < 10"
```

(pomiń szczegóły składni stringów i liczb)

⚙️ ZADANIA Z KOMPILATORÓW (2 zadania)

1. Kompilator z funkcjami:

```
ocaml
```

```
type expr =
  | Int of int
  | Var of string
  | Add of expr * expr
  | Let of string * expr * expr
  | Fun of string * expr
  | App of expr * expr

type instr =
  | Push of int
  | Load of int      (* załaduj zmienną z pozycji n *)
  | Store of int     (* zapisz na pozycję n *)
  | AddOp
  | MakeClosure of instr list  (* stwórz closure *)
  | Apply            (* aplikuj funkcję *)
```

Napisz `compile : expr -> instr list` dla tego języka.

2. Analiza dead code:

Napisz `remove_dead_code : instr list -> instr list` która usuwa instrukcje Push po których natychmiast następuje Pop (i inne oczywiste martwe instrukcje).

🔧 ZADANIA Z INTERPRETERÓW (2 zadania)

1. Interpreter z rekordami:

Napisz kompletny interpreter dla języka z:

```ocaml
type expr =
  | Int of int
  | Bool of bool
  | Var of string
  | Add of expr * expr
  | Let of string * expr * expr
  | Record of (string * expr) list    (* {x=5; y=true} *)
  | Get of expr * string              (* record.field *)
  | Set of expr * string * expr       (* record with field=value *)

type value =
  | VInt of int
  | VBool of bool
  | VRecord of (string * value) list
```

2. Interpreter z pętlami:

Napisz kompletny interpreter dla języka z:

```ocaml
type expr =
  | Int of int
  | Bool of bool
  | Var of string
  | Add of expr * expr
  | Lt of expr * expr
  | Let of string * expr * expr
  | Assign of string * expr           (* x := 5 *)
  | Seq of expr * expr                (* e1; e2 *)
  | While of expr * expr              (* while cond do body *)
  | Skip                              (* no-op *)

type value =
  | VInt of int
  | VBool of bool
  | VUnit                             (* dla Skip i Assign *)
```

Uwaga: Ten interpreter potrzebuje mutable environment dla Assign!