ZESTAW ZADAŃ TRENINGOWYCH - SERIA 2

📋 ZADANIA Z FOLD_LIST (10 zadań)

1. `min_list : int list -> int option`

Znajdź minimalny element (None dla pustej listy) używając `List.fold_left`.

2. `last_element : 'a list -> 'a option`

Znajdź ostatni element listy używając `List.fold_left`.

3. `take_while_positive : int list -> int list`

Weź elementy z początku listy dopóki są dodatnie używając `List.fold_right`.

4. `partition_even_odd : int list -> int list * int list`

Podziel listę na parzyste i nieparzyste używając folda (zwróć (parzyste, nieparzyste)).

5. `flatten_lists : 'a list list -> 'a list`

Spłaszcz listę list używając `List.fold_right`.

6. `any_true : bool list -> bool`

Sprawdź czy przynajmniej jedna wartość to `true` używając folda.

7. `string_length_sum : string list -> int`

Zsumuj długości wszystkich stringów używając folda.

8. `remove_duplicates : int list -> int list`

Usuń duplikaty z listy (zostaw pierwsze wystąpienie) używając folda.

9. `zip_with_indices : 'a list -> (int * 'a) list`

Dodaj indeksy do elementów (zaczynając od 0) używając folda.

10. `group_by_sign : int list -> int list * int list * int list`

Podziel na (ujemne, zero, dodatnie) używając folda.

🌳 ZADANIA Z FOLD_TREE (5 zadań)

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

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

Policz wszystkie liście w drzewie używając `fold_tree`.

2. `depth_tree : 'a tree -> int`

Oblicz głębokość drzewa (maksymalną odległość od korzenia do liścia).

3. `tree_to_list_preorder : 'a tree -> 'a list`

Zbierz wartości w kolejności pre-order (korzeń, lewe, prawe).

4. `find_in_tree : 'a -> 'a tree -> bool`

Sprawdź czy element występuje w drzewie używając folda.

5. `tree_max : int tree -> int option`

Znajdź maksymalną wartość w drzewie (None dla samych liści).

🧮 ZADANIA Z INDUKCJI (5 zadań)

DOWODY (3 zadania):

1. Udowodnij że dla każdej listy `xs` i elementu `x`:

`reverse (x :: xs) = reverse xs @ [x]`

2. Udowodnij że dla każdego drzewa `t` i funkcji `f`:

`height (map_tree f t) = height t`

3. Udowodnij że dla każdych list `xs`, `ys`, `zs`:

`(xs @ ys) @ zs = xs @ (ys @ zs)` (łączność konkatenacji)

SFORMUŁOWANIE ZASAD (2 zadania):

4. Sformułuj zasadę indukcji dla typu:

```ocaml
type 'a option_tree =
  | Empty
  | Node of 'a option_tree * 'a option * 'a option_tree
```

5. Sformułuj zasadę indukcji dla typu:

```ocaml
type instruction =
  | Push of int
  | Pop
  | Add
  | Dup
type program = instruction list
```

🔤 ZADANIA Z TYPOWANIA (10 zadań - podaj typ)

1. `fun f x y -> f y x`
2. `fun lst -> List.length lst + 1`
3. `fun x -> fun y -> fun z -> (x, y, z)`
4. `fun (f, g) x -> (f x, g x)`
5. `fun pred lst -> List.exists pred lst`
6. `fun x -> x :: x :: []`
7. `fun f lst -> List.fold_right f lst []`
8. `fun opt -> match opt with | Some x -> x | None -> 0`
9. `fun f g h x -> f (g x) (h x)`
10. `fun lst1 lst2 -> List.map2 (+) lst1 lst2`

🎯 ZADANIA ODWROTNE - TYP → FUNKCJA (10 zadań)

Napisz funkcję o podanym typie:

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

7. `('a -> bool) -> ('a -> bool) -> 'a -> bool`

8. `'a list -> 'a list -> bool`

9. `int -> 'a -> 'a list`

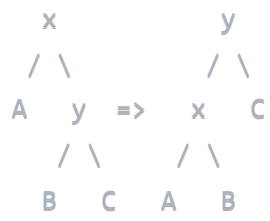
10. `('a option -> 'b) -> 'a list -> 'b list`

🌲 ZADANIA Z OPERACJI NA DRZEWACH (5 zadań)

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

1. `rotate_left : 'a tree -> 'a tree`

Wykonaj rotację w lewo dla drzewa BST:



2. `tree_paths : 'a tree -> 'a list list`

Zwróć wszystkie ścieżki od korzenia do liści (każda ścieżka to lista wartości).

3. `tree_zip : 'a tree -> 'b tree -> ('a * 'b) tree option`

Połącz dwa drzewa tej samej struktury w jedno. None jeśli struktury się różnią.

4. `substitute : 'a -> 'a -> 'a tree -> 'a tree`

Zastąp wszystkie wystąpienia pierwszego elementu drugim w drzewie.

5. `tree_fold_with_path : ('a -> bool list -> 'b -> 'b) -> 'b -> 'a tree -> 'b`

Fold który przekazuje też ścieżkę do elementu (true=prawo, false=lewo).

📝 ZADANIA Z GRAMATYK BEZKONTEKSTOWYCH (2 zadania)

1. Napisz gramatykę bezkontekstową dla języka:

$$L = \{a^i b^j c^k \mid i = j \text{ lub } j = k,\ i,j,k \geq 0\}$$

(tyle samo a i b, LUB tyle samo b i c)

2. Napisz gramatykę dla wyrażeń arytmetycznych z operatorami +, *, nawiasami i liczbami:

- `*` ma wyższy priorytet niż `+`
- Oba operatory są lewostronnie łączne
- Przykłady: `2+3*4`, `(2+3)*4`, `1+2+3*4*5`

⚙️ ZADANIA Z KOMPILATORÓW (2 zadania)

1. Kompilator z warunkami:

```ocaml
type expr =
  | Int of int
  | Bool of bool
  | Add of expr * expr
  | If of expr * expr * expr

type instr =
  | Push of int
  | PushBool of bool
  | AddOp
  | JumpIfFalse of int
  | Jump of int
```

Napisz `compile : expr -> instr list` dla wyrażeń z if-then-else.

2. Optymalizacja stosu:

Dla programu z zadania poprzedniego, napisz funkcję `optimize : instr list -> instr list` która usuwa niepotrzebne instrukcje Push+Pop występujące obok siebie.

🔧 ZADANIA Z INTERPRETERÓW (2 zadania)

1. Interpreter z listami:

Dodaj do interpretera obsługę:

```ocaml
| Nil                          (* pusta lista *)
| Cons of expr * expr          (* dodanie elementu *)
| Head of expr                 (* pierwszy element *)
| Tail of expr                 (* reszta listy *)
| IsEmpty of expr              (* czy pusta *)
```

Pamiętaj o dodaniu `VList of value list` do typu `value`!

2. Interpreter z wyjątkami:

Dodaj obsługę wyjątków:

```ocaml
| Throw of expr                (* rzuć wyjątek *)
| TryCatch of expr * ident * expr (* try e1 catch x -> e2 *)
```

Wskazówka: Możesz użyć wyjątków OCaml lub dodać `VException of value` i propagować je przez eval.