ZESTAW ZADAŃ TRENINGOWYCH

📋 ZADANIA Z FOLD_LIST (10 zadań)

1. `sum_list : int list -> int`

Zsumuj wszystkie elementy listy używając `List.fold_left`.

2. `product_list : int list -> int`

Oblicz iloczyn wszystkich elementów listy używając `List.fold_right`.

3. `length_list : 'a list -> int`

Oblicz długość listy używając folda.

4. `reverse_list : 'a list -> 'a list`

Odwróć listę używając `List.fold_left`.

5. `max_list : int list -> int option`

Znajdź maksymalny element (None dla pustej listy) używając folda.

6. `concat_strings : string list -> string`

Połącz wszystkie stringi w jeden używając `List.fold_left`.

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

Zostaw tylko liczby dodatnie używając folda.

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

Podwój każdy element listy używając folda.

9. `count_true : bool list -> int`

Policz ile jest wartości `true` na liście używając folda.

10. `all_even : int list -> bool`

Sprawdź czy wszystkie liczby są parzyste używając folda.

🌳 ZADANIA Z FOLD_TREE (5 zadań)

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

1. `sum_tree : int tree -> int`

Zsumuj wszystkie wartości w drzewie używając `fold_tree`.

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

Policz wszystkie węzły (nie liście) w drzewie.

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

Oblicz wysokość drzewa (Leaf ma wysokość 0).

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

Zbierz wszystkie wartości z drzewa w jedną listę (in-order).

5. `all_positive : int tree -> bool`

Sprawdź czy wszystkie wartości w drzewie są dodatnie.

🧮 ZADANIA Z INDUKCJI (5 zadań)

DOWODY (3 zadania):

1. Udowodnij że dla każdej listy `xs`:

`length (reverse xs) = length xs`

2. Udowodnij że dla każdej listy `xs` i `ys`:

`reverse (xs @ ys) = reverse ys @ reverse xs`

3. Udowodnij że dla każdego drzewa `t`:

`count_leaves (mirror t) = count_leaves t`

SFORMUŁOWANIE ZASAD (2 zadania):

4. Sformułuj zasadę indukcji dla typu:

```ocaml
type expr =
  | Var of string
  | Add of expr * expr
  | Mult of expr * expr
```

5. Sformułuj zasadę indukcji dla typu:

```ocaml
type 'a rlist =
  | Empty
  | Single of 'a
  | Concat of 'a rlist * 'a rlist
```

🔤 ZADANIA Z TYPOWANIA (10 zadań - podaj typ)

1. `fun x -> x + 1`
2. `fun f x -> f (f x)`
3. `fun x y -> if x > 0 then y else y + 1`
4. `fun f g x -> f (g x)`
5. `fun x -> (x, x)`
6. `fun (x, y) -> x + y`
7. `fun f lst -> List.map f lst`
8. `fun x y z -> x (y z)`
9. `fun pred lst -> List.filter pred lst`
10. `fun f acc lst -> List.fold_left f acc lst`

🎯 ZADANIA ODWROTNE - TYP → FUNKCJA (10 zadań)

Napisz funkcję o podanym typie:

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

8. `'a option -> 'a -> 'a`

9. `('a -> 'a -> bool) -> 'a list -> 'a list`

10. `int -> (int -> int) -> int`

🌲 ZADANIA Z OPERACJI NA DRZEWACH (5 zadań)

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

1. `flatten_tree : 'a tree -> 'a list`

Spłaszcz drzewo do listy (pre-order traversal).

2. `map_tree : ('a -> 'b) -> 'a tree -> 'b tree`

Zastosuj funkcję do każdej wartości w drzewie.

3. `find_path : 'a -> 'a tree -> bool list option`

Znajdź ścieżkę do elementu (true = prawo, false = lewo), None jeśli nie ma.

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

Zwróć listę poziomów drzewa (każdy poziom to osobna lista).

5. `prune : int -> 'a tree -> 'a tree`

Usuń wszystkie węzły głębsze niż `n` (zastąp je liśćmi).

📝 ZADANIA Z GRAMATYK BEZKONTEKSTOWYCH (2 zadania)

1. Napisz gramatykę bezkontekstową dla języka:

```
L = {a^n b^m c^n | n,m ≥ 0}
```

(tyle samo a i c, dowolnie dużo b w środku)

2. Napisz gramatykę dla poprawnych wyrażeń z nawiasami:

```
Przykłady: "", "()", "(())", "()()", "((()))"
```

Gramatyka ma być jednoznaczna.

⚙️ ZADANIA Z KOMPILATORÓW (2 zadania)

1. Maszyna stosowa dla wyrażeń:

```ocaml
type expr = Int of int | Add of expr * expr | Mult of expr * expr
type instr = Push of int | AddOp | MultOp
```

Napisz funkcję `compile : expr -> instr list` która kompiluje wyrażenie do instrukcji maszyny stosowej.

2. Obliczanie maksymalnej wysokości stosu:

Dla programu z zadania 1, napisz funkcję `max_stack_height : instr list -> int` która oblicza maksymalną wysokość stosu podczas wykonania programu (startując z pustym stosem).

🔧 ZADANIA Z INTERPRETERÓW (2 zadania)

1. Rozszerzenie interpretera:

Dodaj do swojego interpretera obsługę:

```ocaml
| Div of expr * expr  (* dzielenie całkowite *)
| Mod of expr * expr  (* reszta z dzielenia *)
```

Pamiętaj o obsłudze dzielenia przez zero!

2. Let z wieloma zmiennymi:

Rozszerz interpreter o konstrukcję:

```ocaml
| LetMulti of (ident * expr) list * expr
```

Przykład: `LetMulti([("x", Int 5); ("y", Int 3)], Add(Var "x", Var "y"))` Wszystkie wyrażenia po prawej stronie mają być ewaluowane w oryginalnym środowisku!