**Яненко Денис**

**Варіант 2**

**Завдання:**

Оголосити необхiднi типи даних i класи типiв для роботи з базою даних. Серед методiв створених класiв має бути хоча б один оператор. Створенi типи даних треба оголосити екземплярами створених та iнших вiдповiдних класiв, зокрема Foldable, Monoid, Functor чи Applicative. Данi, оскiльки базова система введення/виведення детально ще не розглядалась, можна зберiгати у кодi.

2. Публiкацiї. Зберiгаються данi про публiкацiї, якi можуть бути книгою (автор/спiвавтори, назва, мiсто, видавництво, рiк), статтею (автор/спiвавтори, назва статтi, назва журналу, рiк, номер журналу, сторiнки) або тезами доповiдi (автор/спiвавтори, назва доповiдi, назва конференцiї, мiсто, рiк, сторiнки). Визначне функцiї для:

2.1 визначення до якого типу вiдноситься публiкацiя з певною назвою

(книга, стаття або тези);

2.2 пошуку усiх статей (книг, тез) вказаного автора;

2.3 пошуку усiх одноосiбних публiкацiй вказаного автора;

2.4 пошуку усiх видавництв, журналiв, конференцiй;

2.5 статистика по базi за типом публiкацiй;

2.6 iнше.

**Виконання:**

import Data.Monoid

import Data.Semigroup

import Data.List (find, nub, intercalate)

-- Оголошення типу даних для автора

type Author = String

-- Оголошення типу даних для публікації

data Publication =

Book {

authors :: [Author],

title :: String,

city :: String,

publisher :: String,

year :: Int,

pageCount :: Int

}

| Article {

authors :: [Author],

title :: String,

journal :: String,

year :: Int,

issue :: Int,

pages :: (Int, Int) -- (start page, end page)

}

| Thesis {

authors :: [Author],

title :: String,

conference :: String,

city :: String,

year :: Int,

pages :: (Int, Int) -- (start page, end page)

}

deriving (Show, Eq)

-- Власний клас типів для публікацій

class PublicationType publication where

getType :: publication -> String

getSource :: publication -> String

-- Втілення класу типів для книги

instance PublicationType Publication where

getType (Book {}) = "Book"

getType (Article {}) = "Article"

getType (Thesis {}) = "Thesis"

getSource (Book { publisher = pub }) = pub

getSource (Article { journal = j }) = j

getSource (Thesis { conference = c }) = c

-- Оголошення типу даних для списку публікацій

newtype Publications = Publications [Publication] deriving (Show)

-- Втілення класу Monoid для списку публікацій

instance Semigroup Publications where

(Publications xs) <> (Publications ys) = Publications (xs <> ys)

instance Monoid Publications where

mempty = Publications []

-- Функція для визначення типу публікації за назвою

publicationType :: String -> Publications -> Maybe String

publicationType name (Publications publications) =

case find (\p -> title p == name) publications of

Just pub -> Just (getType pub)

Nothing -> Nothing

-- Функція для пошуку усіх публікацій вказаного автора

findPublicationsByAuthor :: Publications -> String -> Publications

findPublicationsByAuthor (Publications pubs) auth = Publications $ filter (\p -> auth `elem` authors p) pubs

-- Функція для пошуку усіх одноосібних публікацій вказаного автора

findSoloPublicationsByAuthor :: Publications -> String -> Publications

findSoloPublicationsByAuthor (Publications pubs) auth = Publications $ filter (\p -> length (authors p) == 1 && auth `elem` authors p) pubs

-- Функція для статистики по базі за типом публікацій

publicationTypeStatistics :: Publications -> [(String, Int)]

publicationTypeStatistics (Publications pubs) =

let getTypeCount typ = length $ filter (\p -> getType p == typ) pubs

bookCount = getTypeCount "Book"

articleCount = getTypeCount "Article"

thesisCount = getTypeCount "Thesis"

in [("Book", bookCount), ("Article", articleCount), ("Thesis", thesisCount)]

-- Функція для отримання унікальних значень видавництв, журналів та конференцій

getAllPublishersJournalsConferences :: Publications -> [(String, [String])]

getAllPublishersJournalsConferences (Publications publications) = [

("Publishers:", nub publishers),

("Journals:", nub journals),

("Conferences:", nub conferences)

]

where

publishers = map getSource $ filter (\p -> getType p == "Book") publications

journals = map getSource $ filter (\p -> getType p == "Article") publications

conferences = map getSource $ filter (\p -> getType p == "Thesis") publications

-- Функція для виведення публікації з форматованим виглядом

printPublication :: Publication -> IO ()

printPublication (Book authors title city publisher year pageCount) =

putStrLn $ "Book: " ++ title ++ " (" ++ show year ++ "), by " ++ formatAuthors authors ++ ", published by " ++ publisher ++ ", " ++ city ++ ", " ++ show pageCount ++ " pages"

printPublication (Article authors title journal year issue (startPage, endPage)) =

putStrLn $ "Article: " ++ title ++ " (" ++ show year ++ "), by " ++ formatAuthors authors ++ ", published in " ++ journal ++ ", Issue " ++ show issue ++ ", pages " ++ show startPage ++ "-" ++ show endPage

printPublication (Thesis authors title conference city year (startPage, endPage)) =

putStrLn $ "Thesis: " ++ title ++ " (" ++ show year ++ "), by " ++ formatAuthors authors ++ ", presented at " ++ conference ++ ", " ++ city ++ ", pages " ++ show startPage ++ "-" ++ show endPage

-- Функція для форматування списку авторів

formatAuthors :: [Author] -> String

formatAuthors authors = intercalate ", " authors

printPublications :: Publications -> IO ()

printPublications (Publications pubs) = mapM\_ printPublication pubs

main :: IO ()

main = do

let publications1 = Publications [

Book ["Author1"] "Title1" "City1" "Publisher1" 2022 300,

Article ["Author2"] "Title2" "Journal1" 2023 1 (10, 20),

Thesis ["Author3"] "Title3" "Conference1" "City2" 2021 (30, 40)

]

publications2 = Publications [

Article ["Author1", "Author4"] "Title4" "Journal2" 2024 2 (15, 25),

Book ["Author5"] "Title5" "City3" "Publisher2" 2023 250,

Thesis ["Author6"] "Title6" "Conference2" "City4" 2022 (50, 60)

]

publications = publications1 <> publications2

publicationName = "Title1"

authorName = "Author1"

putStrLn "All publications:"

printPublications publications

putStrLn $ "\nPublication type for " ++ publicationName ++ ":"

case publicationType publicationName publications of

Just pubType -> putStrLn pubType

Nothing -> putStrLn "Publication not found"

putStrLn $ "\nPublications by " ++ authorName ++ ":"

printPublications $ findPublicationsByAuthor publications authorName

putStrLn $ "\nSolo publications by " ++ authorName ++ ":"

printPublications $ findSoloPublicationsByAuthor publications authorName

putStrLn "\nPublication type statistics:"

print $ publicationTypeStatistics publications

putStrLn "\nUnique publishers, journals, and conferences:"

let categories = getAllPublishersJournalsConferences publications

mapM\_ (\(category, items) -> do

putStrLn category

mapM\_ putStrLn items

putStrLn ""

) categories

**Результат:**

All publications:

Book: Title1 (2022), by Author1, published by Publisher1, City1, 300 pages

Article: Title2 (2023), by Author2, published in Journal1, Issue 1, pages 10-20

Thesis: Title3 (2021), by Author3, presented at Conference1, City2, pages 30-40

Article: Title4 (2024), by Author1, Author4, published in Journal2, Issue 2, pages 15-25

Book: Title5 (2023), by Author5, published by Publisher2, City3, 250 pages

Thesis: Title6 (2022), by Author6, presented at Conference2, City4, pages 50-60

Publication type for Title1:

Book

Publications by Author1:

Book: Title1 (2022), by Author1, published by Publisher1, City1, 300 pages

Article: Title4 (2024), by Author1, Author4, published in Journal2, Issue 2, pages 15-25

Solo publications by Author1:

Book: Title1 (2022), by Author1, published by Publisher1, City1, 300 pages

Publication type statistics:

[("Book",2),("Article",2),("Thesis",2)]

Unique publishers, journals, and conferences:

Publishers:

Publisher1

Publisher2

Journals:

Journal1

Journal2

Conferences:

Conference1

Conference2