**Білет20**

**1. Перевірити, чи є st1 підмультимножиною мультимножини st2 (типа bagSubbag "abac" "aaabcc" = True; bagSubbag "abac" "aaacc" = False)**

{-# OPTIONS\_GHC -Wall #-}

module B20 where

bagSubbag :: String -> String -> Bool

bagSubbag (x:xs) w | elem x w = bagSubbag xs (delOne w x)

| otherwise = False

bagSubbag [] \_ = True

delOne :: String -> Char -> String

delOne (x:xs) w | x==w = xs

| otherwise = [x] ++ delOne xs w

delOne [] \_ = ""

**2. Функції згортки і прогонки**

**3. Монади Maybe і Either**

**Білет14**

**1. Написати функцію mergeFiles, зливає всі впорядковані по рядках файли в один впорядкований. Остаточний файл надсилається в стандартний файл виведення stdout. Імена файлів вводяться по запиту програми в одному рядку, відділяючись проміжками. Наприклад: Files> a.txt b.txt c.txt. На запит програми потрібно злити файли з іменами a.txt, b.txt, c.txt.**

{-# OPTIONS\_GHC -Wall #-}

module B14 where

mergeFiles :: IO()

mergeFiles = do putStr "Filenames> "

line <- getLine

let files = split ' ' line

file1 <- readFile (files!!0)

file2 <- readFile (files!!1)

file3 <- readFile (files!!2)

writeFile "stdout.txt" (file1 ++ "\n" ++ file2 ++ "\n" ++ file3)

split :: Char -> String -> [String]

split \_ [] = [""]

split ch (c:cs) | c == ch = "" : rest

| otherwise = (c : head rest) : tail rest

where rest = split ch cs

**2. Конфлікти імен**

**3. Функція guard**

**Білет 17**

**1. Напишіть функцію lequation, котра виводить рядок, що містить кофіцієнти a і b лінійного рівняння a\*x+b = c, розпізнає в рядку коефіцієнти, розв’язує рівняння і виводить відповідь. Кожний коефіцієнт – ціле число або дійсне число у вигляді ddd.dddd (d – десяткова цифра). Числа розділяються проміжками, можуть містити початкові та заключні проміжки. Можливі рядки-відповіді: «х=…», «many», “no”, “error”.**

{-# OPTIONS\_GHC -Wall#-}

module B17 where

import Data.List

import Data.Char

lequation :: IO()

lequation = do putStr "Input numbers> "

x <- getLine

putStrLn (getRes x)

getRes :: String -> String

getRes x = case (getNumbers x) of

Just (a,b,c) -> if a==0 then (if c-b==0 then "many" else "no")

else (show (calc a b c))

\_ -> "error"

calc :: Double -> Double -> Double -> Double

calc a b c = (c-b)/a

getNumbers :: String -> Maybe (Double, Double, Double)

getNumbers xs = let ys = split ' ' xs

ys2 = [y|y<-ys, y/=""]

in if (length ys2 < 3) then Nothing

else case (getNumber (ys2!!0)) of

Just a -> case (getNumber (ys2!!1)) of

Just b -> case (getNumber (ys2!!2)) of

Just c -> Just (a,b,c)

\_ -> Nothing

\_ -> Nothing

\_ -> Nothing

getNumber :: String -> Maybe Double

getNumber xs | elem '.' xs = if (head xs == '.') then getNumber (tail xs)

else if ((xs!!(length xs - 1)) == '.' ) then getNumber [xs!!n|n<-[0..(length xs - 2)]]

else getDouble xs

| otherwise = getInt xs

getDouble :: String -> Maybe Double

getDouble xs = let y = split '.' xs

in if (length y < 2) then Nothing

else case (getInt (y!!0)) of

Just a -> case (getDo (y!!1)) of

Just b -> Just (a+b)

\_ -> Nothing

\_ -> Nothing

getDo :: String -> Maybe Double

getDo xs = let bol = and [isDigit x|x<-xs]

in if bol then Just (getDo2 (xs) 1)

else Nothing

getDo2 :: String -> Integer -> Double

getDo2 (x:xs) n = ((findInt x)/(10^n)) + (getDo2 xs (n+1))

getDo2 [] \_ = 0

getInt :: String -> Maybe Double

getInt xs = let bol = and [isDigit x|x<-xs]

in if bol then Just (getInt2 (reverse xs) 0)

else Nothing

getInt2 :: String -> Integer -> Double

getInt2 (x:xs) n = (findInt x)\*(10^n) + (getInt2 xs (n+1))

getInt2 [] \_ = 0

findInt :: Char -> Double

findInt x = case x of

'1'-> 1.0

'2'-> 2.0

'3'-> 3.0

'4'-> 4.0

'5'-> 5.0

'6' -> 6.0

'7' -> 7.0

'8' -> 8.0

'9'-> 9.0

\_ -> 0.0

split :: Char -> String -> [String]

split \_ [] = [""]

split ch (c:cs) | c == ch = "" : rest

| otherwise = (c : head rest) : tail rest

where rest = split ch cs

**2. Класи типів Eq, Ord i Enum**

**3. Нескінченні списки**

**Білет25**

**1. Напишіть функцію seqDulb, котра просить ввести ім’я файлу, а потім обробляє його рядки і виводить їх в стандартний файл виведення stdout. Обробка файлу: надсилає всі різні рядки (не обов’язково «суміжні»).**

{-# OPTIONS\_GHC -Wall #-}

module B25 where

seqDulb :: IO()

seqDulb = do putStr "Filenames> "

line <- getLine

file <- readFile line

let res = getRes file

writeFile "stdout.txt" res

getRes :: String -> String

getRes xs = unlines (unic (lines xs))

--getRes xs = unlines (reverse (lines xs))

unic :: (Ord a) => [a] -> [a]

unic (x:xs) | elem x xs = unic xs

| otherwise = [x] ++ unic xs

unic [] = []

**2. Функція над списками map, filter.**

**3. Екземпляри класів**

**Білет 7**

**1. Напишіть функцію seqSymb, котра просить ввести ім’я файлу, а потім обробляє його рядки і виводить їх в стандартний файл виведення stdout. Обробка файлу: змінює в кожному рядку максимальну послідовність, складену з К>3 вживань одного і того ж символу, послідовністю (k)s, де s – повторюємий символ**

{-# OPTIONS\_GHC -Wall #-}

module B25 where

seqSymb :: IO()

seqSymb = do putStr "Filename> "

line <- getLine

file <- readFile line

let res = getRes file

writeFile "stdout.txt" res

getRes :: String -> String

getRes xs = unlines ([clearLine x| x<-(lines xs)])

clearLine :: String -> String

clearLine [] = []

clearLine xs | length xs < 3 = xs

| (xs!!0)==(xs!!1) && (xs!!1)==(xs!!2) = let (a,b) = clear xs (xs!!0) 0

in a ++ (clearLine b)

| otherwise = [head xs] ++ (clearLine (tail xs))

clear :: String -> Char -> Integer -> (String,String)

clear (x:xs) y n | y==x = clear xs y (n+1)

| otherwise = ("("++(show n)++")"++[y], [x]++xs)

clear [] y n = ("("++(show n)++")"++[y],"")

**2. Списки і функції**

**3. Бібліотека Parsec**

**Білет 13**

**1. Напишіть функцію stringToInt n xs, що переводить рядок, котрий задає деяке ціле число, в n-мірній системі числення (n <= 16). Вважається що для кодування чисел більших за 9 використовують символи ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’. Якщо рядок не задає число у відповідній системі числення, то результат Nothing.**

{-# OPTIONS\_GHC -Wall #-}

module B13 where

stringToInt :: Integer -> String -> Maybe Integer

stringToInt n xs | notSuit n xs = Nothing

| otherwise = let l = (length xs - 1)

in Just (calc n xs l)

calc :: Integer -> String -> Int -> Integer

calc n xs k | k==0 = findInt (head xs)

| otherwise = (findInt (head xs))\*(n^k) + (calc n (tail xs) (k-1))

notSuit::Integer->String->Bool

notSuit n xs = or [(findInt x)>= n|x<-xs]

findInt :: Char -> Integer

findInt x = case x of

'0'-> 0

'1'-> 1

'2'-> 2

'3'-> 3

'4'-> 4

'5'-> 5

'6' -> 6

'7' -> 7

'8' -> 8

'9'-> 9

'a' -> 10

'b' -> 11

'c' -> 12

'd' -> 13

'e' -> 14

'f' -> 15

\_ -> 0

**2. Експорт і імпорт модулів**

**3. Тип для аналізатора(Parser)**

**Білет 26**

**1. Напишіть функцію seqWord, котра просить ввести ім’я файлу, обробляє його рядки і виводить результат в стандартний файл виведення. Обробка файлу: вилучає в кожному рядку всі слова які раніше були в цьому ж рядку.**

{-# OPTIONS\_GHC -Wall #-}

module B26 where

seqDulb :: IO()

seqDulb = do putStr "Filename> "

line <- getLine

file <- readFile line

let res = getRes file

writeFile "stdout.txt" res

getRes :: String -> String

getRes xs = let lins = lines xs

in unlines [clear x|x<-lins]

clear :: String -> String

clear xs = let ys = split ' ' xs

un = unic ys

in concat [x++" "| x<-un]

unic :: (Ord a) => [a] -> [a]

unic (x:xs) | elem x xs = unic xs

| otherwise = [x] ++ unic xs

unic [] = []

split :: Char -> String -> [String]

split \_ [] = [""]

split ch (c:cs) | c == ch = "" : rest

| otherwise = (c : head rest) : tail rest

where rest = split ch cs

**2. Арифметичні послідовності**

**3. Поняття класу типів**

**Білет 9**

**1. Напишіть функцію intToString n m, що будує по цілому додатному числу його представлення в n-мірній системі числення (n <= 16). Вважається що для кодування чисел більших за 9 використовують символи ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’. Якщо рядок не задає число у відповідній системі числення, то результат Nothing.**

{-# OPTIONS\_GHC -Wall #-}

module B9 where

intToString :: Int -> Int -> String

intToString n m = let res = findRes n m

in reverse res

findRes :: Int -> Int -> String

findRes n m | m < n = [toChar (m `mod` n), toChar (div m n)]

| otherwise = [toChar (m `mod` n)] ++ (findRes n (div m n))

toChar :: Int -> Char

toChar x = case x of

0-> '0'

1-> '1'

2-> '2'

3-> '3'

4-> '4'

5-> '5'

6 -> '6'

7 -> '7'

8 -> '8'

9 -> '9'

10 -> 'a'

11 -> 'b'

12 -> 'c'

13 -> 'd'

14 -> 'e'

15 -> 'f'

\_ -> '0'

**2. Створення нових типів (data)**

**3. Монади Either**

**Білет 14**

**1. Написати функцію primeCnt xs, що вираховує кількість простих додатних чисел в списку цілих чисел xs.**

{-# OPTIONS\_GHC -Wall #-}

module B14 where

primeCnt :: [Int] -> [Int]

primeCnt xs = [x | x<- xs, isSimple x]

isSimple :: Int -> Bool

isSimple n = let ys = [2..(div n 2)]

bol = or [mod n x == 0 | x<-ys]

in not bol

**2. Конфлікти імен**

**3. Приклади монади State**

**Білет 10**

**1. Задано рядок xs, всі суфікси якого можна лексикографічно впорядкувати. Написати функцію lastTail xs, що знаходить останній після впорядкування. Наприклад: lastTail “introdaction”= “trodaction”, lastTail “tomat” = “tomat”**

{-# OPTIONS\_GHC -Wall #-}

module B10 where

lastTail :: String -> String

lastTail xs = let tails = getTails xs

in if tails==[] then ""

else getMax tails (tails!!0)

getTails :: String -> [String]

getTails xs = [drop x xs | x<-[0..(length xs - 1)]]

getMax :: [String] -> String -> String

getMax (x:xs) m | m>x = getMax xs m

| otherwise = getMax xs x

getMax [] m = m

**2. Ізоморфні типи даних (newtype)**

**3. Дії введення-виведення**

**Білет 21**

**1. Рядок можна розглядати як мультимножину символів. Наприклад «abac» - ожина що має 2 символи «a» і по одному символу «b» і «c». Напишіть функцію bagUnion st1 st2, котра бере мультимножини та повертає їх об’єднання. Об’єднання мультимножин містить всі елементи обох множин, кількість повторень елементу в об’єднанні це максимальна кількість його повторень в одній з множин.**

{-# OPTIONS\_GHC -Wall #-}

module B21 where

bagUnion :: String -> String -> String

bagUnion (x:xs) ys = let am1 = (amount xs x)+1

am2 = amount ys x

nxs = delete xs x

nys = delete ys x

res = if am1>am2 then (getAm am1 x)

else (getAm am2 x)

in res ++ (bagUnion nxs nys)

bagUnion [] ys = ys

amount :: String -> Char -> Int

amount (x:xs) y | y==x = 1 + (amount xs y)

| otherwise = amount xs y

amount [] \_ = 0

delete :: String -> Char -> String

delete (x:xs) y | x==y = delete xs y

| otherwise = [x]++(delete xs y)

delete [] \_ = []

getAm :: Int -> Char -> String

getAm n ch = [ch | \_<-[1..n]]

**2. Застосування функції**

**3. Монада State**

**Білет 22**

**1. Написати функцію, яка знаходить перетин між 2 стрінгами ("abbc" "ab" =ab)**

{-# OPTIONS\_GHC -Wall #-}

module B21 where

mainF :: String -> String -> String

mainF xs ys = merge (unic xs) (unic ys)

merge :: String -> String -> String

merge (x:xs) ys | elem x ys = [x] ++ (merge xs ys)

| otherwise = merge xs ys

merge [] \_ = []

unic :: (Ord a) => [a] -> [a]

unic (x:xs) | elem x xs = unic xs

| otherwise = [x] ++ unic xs

unic [] = []

**2. Базові типи даних**

**3. Монада Sparse**

**Білет 8**

**1. Напишіть функцію sum2, що вводить 2 рядки. Якщо кожен рядок містить ціле число, можливо зі знаком, то функція розпізнає цілі числа і виводить їх суму. В інших випадках виводить “error”. Допустимі проміжки перед числом і після числа**

{-# OPTIONS\_GHC -Wall #-}

module B8 where

import Data.Char

sum2 :: IO()

sum2 = do putStr "xs> "

xs <- getLine

putStr "ys> "

ys <- getLine

putStrLn (getRes xs ys)

getRes :: String -> String -> String

getRes xs ys = case (getNumber xs) of

Just a -> case(getNumber ys) of

Just b -> (show (a+b))

\_ -> "error"

\_ -> "error"

getNumber :: String -> Maybe Int

getNumber xs = let xs2 = removeSpaces xs

isMin = if xs2==[] then False else (head xs2)=='-'

nxs = if xs2==[] then [] else if ((head xs2)=='-') || ((head xs2)=='+')

then reverse (drop 1 xs2) else reverse xs2

cor = and [isDigit x| x<- nxs]

in if not cor then Nothing

else if isMin then Just ((getInt nxs 0)\*(-1)) else Just (getInt nxs 0)

getInt :: String -> Int -> Int

getInt (x:xs) n = (findInt x)\*(10^n) + (getInt xs (n+1))

getInt [] \_ = 0

removeSpaces :: String -> String

removeSpaces (x:xs) | x==' ' = removeSpaces xs

| xs == [] = [x]

| xs!!(length xs - 1)==' ' = removeSpaces [x]++[xs!!y|y<-[0..(length xs - 2)]]

| otherwise = [x]++xs

removeSpaces [] = []

findInt :: Char -> Int

findInt x = case x of

'1'-> 1

'2'-> 2

'3'-> 3

'4'-> 4

'5'-> 5

'6' -> 6

'7' -> 7

'8' -> 8

'9'-> 9

\_ -> 0

**2. Синоніми типів (type)**

**3. Нотація do**