Робота № 3

Типи i класи типiв

Варіант № 12

**Мета роботи**

Ознайомитись з системою типiв та класiв типiв. Набути досвiду визначення нових типiв та класiв типiв i їх використання.

**Завдання**

Записник. У записнику зберiгається iнформацiя про знайомих: телефон

(iм’я, телефон — один, або кiлька), нагадування про день народження (iм’я, дата — день та мiсяць), зустрiчi (дата та час, мiсце, опис — тема зустрiчi, зауваження про зустрiч — чи вiдбулась та iнше). Визначне функцiї для пошуку iмен за першою лiтерою iменi.

**Код програми**

**Модуль Main**

import System.IO

import System.Exit

import Control.Monad

import Phonebook

import Terminal

import Interface

import Person

import Meeting

import Date

main = forever $ showMenu "Меню"

[("Переглянути контакти", Interface.printContactsFile),

("Пошук контактів за даними", searchSubmenu Interface.pressEnter'),

("Додати контакт", Interface.addContact),

("Редагувати контакт", searchSubmenu Interface.editOrRemoveP),

("У кого сьогодні день народження?", Interface.whoseBirthday),

("Переглянути зустрічі", Interface.printMeetingsFile),

("Додати зустріч", Interface.addMeeting),

("Редагувати зустріч", searchSubmenuMeeting Interface.editOrRemoveMeeting),

("Вихід", exitSuccess)]

searchSubmenu nextFunction = showMenu "Пошук контактів по:"

[("Імʼя", Interface.find name nextFunction),

("Фамілія", Interface.find familyName nextFunction),

("Номер", Interface.find telephone nextFunction),

("Дата народження (d.m.rrrr)", Interface.find (printableDate.birthday) nextFunction),

("<- Повернутись", main)]

searchSubmenuMeeting nextFunction = showMenu "Пошук зустрічей по:"

[("Імʼя", Interface.findM namePlace nextFunction),

("Місце", Interface.findM place nextFunction),

("Дата проведення (d.m.rrrr)", Interface.findM (printableDate.dateMeeting) nextFunction),

("Проведено?", Interface.findM held nextFunction),

("<- Повернутись", main)]

**Модуль Interface**

module Interface (addContact,

addMeeting,

printContactsFile,

printMeetingsFile,

find,

findM,

pressEnter',

editOrRemoveP,

editOrRemoveMeeting,

whoseBirthday) where

import Phonebook

import Person

import Meeting

import Terminal

import Date

import DataStorage

import Data.Char

import Data.Maybe

dataFile = "contacts"

getBook = DataStorage.loadBook dataFile

saveNewBook newBook = DataStorage.overwriteBook newBook dataFile

getPerson book num = (getPList book) !! (num - 1)

getMeeting book num = (getMList book) !! (num - 1)

getPList (Phonebook pList mList) = pList

getMList (Phonebook pList mList) = mList

addContact :: IO ()

addContact = do book <- getBook

personToAdd <- getPersonData

if not (personToAdd `elem` (getPList book))

then saveNewBook $ addPerson book personToAdd

else putStrFlush "Такий контакт уже існує"

getPersonData = do

name <- promptString' "Імʼя" validName

familyName <- promptString' "Фамілія" validName

telephone <- promptString' "# Телефону" validPhone

birthday <- promptString' "Дата народження (dd.mm.rrrr)" validDate

return $ Person name familyName telephone (stringToDate birthday)

addMeeting :: IO ()

addMeeting = do book <- getBook

meetingToAdd <- getMeetingData

if not (meetingToAdd `elem` (getMList book))

then saveNewBook $ addMeetingData book meetingToAdd

else putStrFlush "Така зустріч вже існує"

getMeetingData = do

namePlace <- promptString' "Імʼя" validName

place <- promptString' "Місце" validName

dateMeeting <- promptString' "Дата (dd.mm.rrrr)" validDate

held <- promptLine "Held?"

return $ Meeting namePlace place (stringToDate dateMeeting) held

validName :: String -> Bool

validName x = (x /= "") && ( and $ map (\c -> isLetter c || c == ' ') x)

validPhone :: String -> Bool

validPhone x = and $ map isDigit x

validMail :: String -> Bool

validMail x = '@' `elem` x && '.' `elem` x

validDate :: String -> Bool

validDate x = x == "" || (isJust $ stringToDate x)

printContactsFile = getBook >>= showBook "Контакти" >> pressEnter

printMeetingsFile = getBook >>= showBookM "Зустрічі" >> pressEnter

find byWhat functionAtEnd= do book <- getBook

value <- promptLine "Префікс:"

showBook "Результати" (Phonebook (findPeopleBy byWhat value book) [])

functionAtEnd (Phonebook (findPeopleBy byWhat value book) [])

findM byWhat functionAtEnd = do

book <- getBook

value <- promptLine "Префікс:"

showBook "Результати" (Phonebook [] (findMeetingBy byWhat value book))

functionAtEnd (Phonebook [] (findMeetingBy byWhat value book))

pressEnter':: Phonebook -> IO ()

pressEnter' whatever = promptLine "ENTER.." >> return ()

whoFromResultsToEdit matchingGuysBook = if (inputLength > 1)

then do nr <- prompt' "Введи номер" (\c -> c >= 1 && c <= inputLength)

return $ getPerson matchingGuysBook nr

else return $ getPerson matchingGuysBook 1

where inputLength = length (getPList matchingGuysBook)

whoFromResultsToEditM matchingMeetings = if (inputLength > 1)

then do nr <- prompt' "Номер зустрічі" (\c -> c >= 1 && c <= inputLength)

return $ getMeeting matchingMeetings nr

else return $ getMeeting matchingMeetings 1

where inputLength = length (getMList matchingMeetings)

editOrRemoveP matchingGuysBook = do persona <- whoFromResultsToEdit matchingGuysBook

editOrDelete <- prompt' "Опції:\n 1) Редагувати\n 2) Видалити\n 3) <---\n " (\c -> c `elem` [1,2,3])

resultAction editOrDelete persona

where resultAction x persona = case x of

1 -> editContact persona

2 -> deleteContact persona

3 -> return ()

editOrRemoveMeeting matchingMeetings = do

meeting <- whoFromResultsToEditM matchingMeetings

editOrDelete <- prompt' "Опції:\n 1) Редагувати\n 2) Видалити\n 3) <---\n " (\c -> c `elem` [1,2])

resultAction editOrDelete meeting

where resultAction x meeting = case x of

1 -> Interface.editMeeting meeting

2 -> deleteMeeting meeting

3 -> return ()

editContact :: Person -> IO ()

editContact oldPerson = do book <- getBook

putStrFlush "Введіть нові дані:\n"

newPerson <- getPersonData

saveNewBook $ editPerson book oldPerson newPerson

putStrFlush "---------------> Контакт змінено\n" >> pressEnter

editMeeting :: Meeting -> IO ()

editMeeting oldM = do

book <- getBook

putStrFlush "Podaj nowe dane kontaktu:\n"

newM <- getMeetingData

saveNewBook $ Phonebook.editMeeting book oldM newM

putStrFlush "---------------> Контакт змінено\n" >> pressEnter

deleteContact :: Person -> IO ()

deleteContact personToDel = do book <- getBook

saveNewBook $ removePerson book personToDel

putStrFlush "---------------> Контакт змінено\n" >> pressEnter

deleteMeeting :: Meeting -> IO ()

deleteMeeting meetingToDel = do

book <- getBook

saveNewBook $ removeMeeting book meetingToDel

putStrFlush "---------------> Контакт змінено\n" >> pressEnter

whoseBirthday = do book <- getBook

listP <- findBirthdayPeople book

showBook "Сьогодні день народження" (Phonebook listP [])

pressEnter

**Модуль Phonebook**

module Phonebook where

import Data.List

import Data.Time

import Data.Maybe

import Person

import Meeting

import Date

data Phonebook = Phonebook [Person] [Meeting] deriving (Show, Read)

addPerson :: Phonebook -> Person -> Phonebook

addPerson (Phonebook pList mList) person = Phonebook (insert person pList) mList

addMeetingData :: Phonebook -> Meeting -> Phonebook

addMeetingData (Phonebook pList mList) meeting = Phonebook pList (insert meeting mList)

removePerson :: Phonebook -> Person -> Phonebook

removePerson (Phonebook pList mList) person = Phonebook (delete person pList) mList

removeMeeting :: Phonebook -> Meeting -> Phonebook

removeMeeting (Phonebook pList mList) meeting = Phonebook pList (delete meeting mList)

editPerson :: Phonebook -> Person -> Person -> Phonebook

editPerson book@(Phonebook pList mList) old new = Phonebook (replace pList old new) mList

editMeeting :: Phonebook -> Meeting -> Meeting -> Phonebook

editMeeting book@(Phonebook pList mList) old new = Phonebook pList (replace mList old new)

findPeopleBy :: (Person -> String) -> String -> Phonebook -> [Person]

findPeopleBy f value (Phonebook pList \_) = filter ((isPrefixOf value) . f) pList

findMeetingBy :: (Meeting -> String) -> String -> Phonebook -> [Meeting]

findMeetingBy f value (Phonebook \_ mList) = filter ((isPrefixOf value) . f) mList

findBirthdayPeople :: Phonebook -> IO [Person]

findBirthdayPeople (Phonebook pList \_) = do

c <- getCurrentTime

let today = Date $ utctDay c

return (filter (hasBirthday today) pList)

deleteAll :: (Eq a) => a -> [a] -> [a]

deleteAll d list = [x | x <- list, x /= d]

replace :: (Eq a, Ord a) => [a] -> a -> a -> [a]

replace list old new

| old `elem` list = if (compare old new) == EQ

then result

else sort result

| otherwise = error "Елемента не існує"

where

result = beginning ++ new : ending

beginning = take position list

ending = drop (position + 1) list

position = fromJust $ elemIndex old list

**Модуль Person**

module Person where

import Data.List

import Data.Time

import Data.Maybe

import Date

type Name = String

type FamilyName = String

type Telephone = String

type Birthday = Maybe Date

data Person = Person {

name :: Name,

familyName :: FamilyName,

telephone :: Telephone,

birthday :: Birthday

} deriving (Show, Read)

instance Ord Person where

compare a b

| a == b = EQ

| familyName a /= familyName b = compare (familyName a) (familyName b)

| name a /= name b = compare (name a) (name b)

| birthday a /= birthday b = compare (birthday a) (birthday b)

| otherwise = EQ

instance Eq Person where

(==) a b = (name a == name b) && (familyName a == familyName b) && (telephone a == telephone b)

printablePerson :: Person -> String

printablePerson p = concat $ intersperse " " [name p, familyName p, telephone p, printableDate (birthday p)]

hasBirthday :: Date -> Person -> Bool

hasBirthday (Date date) person

| isNothing $ birthday person = False

| otherwise = isAnniversary (fromJust $ birthday person) day month where

(\_, m, d) = toGregorian date

month = toInteger m

day = toInteger d

**Модуль Meeting**

*module Meeting where*

*import Data.List*

*import Data.Time*

*import Data.Maybe*

*import Date*

*type NamePlace = String*

*type Place = String*

*type DateMeeting = Maybe Date*

*type HeldMeeting = String*

*data Meeting = Meeting {*

*namePlace :: NamePlace,*

*place :: Place,*

*dateMeeting :: DateMeeting,*

*held :: HeldMeeting*

*} deriving (Show, Read)*

*instance Ord Meeting where*

*compare a b*

*| a == b = EQ*

*| held a /= held b = compare (held a) (held b)*

*| dateMeeting a /= dateMeeting b = compare (dateMeeting a) (dateMeeting b)*

*| namePlace a /= namePlace b = compare (namePlace a) (namePlace b)*

*| place a /= place b = compare (place a) (place b)*

*| otherwise = EQ*

*instance Eq Meeting where*

*(==) a b = (dateMeeting a == dateMeeting b) && (place a == place b) && (namePlace a == namePlace b) && (held a == held b)*

*printableMeeting :: Meeting -> String*

*printableMeeting p = concat $ intersperse " " [namePlace p, place p, printableDate (dateMeeting p), held p]*

**Модуль Date**

module Date where

import Locale

import Data.Time

import Data.Time.Format

import Data.Time.Calendar

import Data.Maybe

data Date = Date Day deriving (Show, Read, Eq, Ord)

dateFormat = "%-d.%-m.%Y"

printableDate :: Maybe Date -> String

printableDate Nothing = ""

printableDate (Just (Date x)) = formatTime defaultTimeLocale dateFormat x

stringToDate x

| isJust res = Just $ Date $ fromJust res

| otherwise = Nothing

where

res = parseTimeM True defaultTimeLocale dateFormat x :: Maybe Day

isAnniversary :: Date -> Integer -> Integer -> Bool

isAnniversary (Date date) day month = (dateDay == day) && (dateMonth == month) where

(\_, m, d) = toGregorian date

dateMonth = toInteger m

dateDay = toInteger d

**Модуль DataStorage**

module DataStorage (loadBook,overwriteBook) where

import Data.List

import System.IO

import System.Directory

import Terminal

import Phonebook

tempFile = "temp"

createFname fname = ("db/" ++ fname ++ ".data")

loadBook :: [Char] -> IO Phonebook

loadBook fname = do System.Directory.createDirectoryIfMissing True "db/"

handleFile <- openFile ("db/contacts.data") ReadWriteMode

idata <- loadData handleFile

return idata

loadData handleFile = do isEof <- hIsEOF handleFile

if isEof then

return (Phonebook [] [])

else do

contents <- hGetContents handleFile

return (read (contents)::Phonebook)

overwriteBook newbook dataFile = do

writeFile tempFile' (show newbook)

x <- doesFileExist dataFile'

if x

then removeFile dataFile' >> renameFile tempFile' dataFile'

else renameFile tempFile' dataFile'

where tempFile' = (createFname tempFile)

dataFile' = (createFname dataFile)

**Модуль Terminal**

module Terminal where

import System.IO

import Control.Exception

import Phonebook

import Person

import Meeting

putStrFlush str = do putStr str

hFlush stdout

promptLine what = do putStrFlush $ what ++ ": "

x <- System.IO.getLine

return x

prompt' :: Read b => [Char] -> (b -> Bool) -> IO b

prompt' text f = do putStrFlush $ text ++ ": "

x <- try readLn :: (Read t0) => IO (Either SomeException t0)

case x of

Left e -> invalid

Right v -> if (f v) then return v

else invalid

where invalid = putStrFlush "Помилка, спробуй ще раз: " >> prompt' text f

promptString' :: [Char] -> ([Char] -> Bool) -> IO [Char]

promptString' text f = do putStrFlush $ text ++ ": "

x <- getLine

if ( f x ) then return x

else putStrFlush "Помилка, спробуй ще раз: " >> promptString' text f

createLabel label size fill = let fillsize = (size - (length label) - 2)

halffillL = fillsize `div` 2

halffillR = halffillL + (fillsize `mod` 2)

in

(replicate halffillL fill) ++ " " ++ label ++ " " ++ (replicate halffillR fill)

showItems title itemsList = showItemsAndComment title itemsList []

showItemsAndComment title itemsList comment =

do putStrLn $ "\n" ++ createLabel title 100 '-'

if (comment /= []) then do

putStrFlush $ " \*) " ++ comment ++ "\n"

putStrFlush $ menuText itemsList 1

else

putStrFlush $ menuText itemsList 1

putStrLn $ createLabel "-" 100 '-'

where

menuText [] inum = []

menuText (i:is) inum = " " ++ (show inum) ++ ") " ++ i ++ "\n" ++ menuText is (inum + 1)

showBook title (Phonebook pList mList) =

do putStrLn $ "\n" ++ createLabel title 100 '-'

putStrFlush $ " \*) " ++ comment ++ "\n"

putStrFlush $ bookText pList 1

putStrLn $ createLabel "-" 100 '-'

where

bookText [] \_ = []

bookText (i:is) inum = " " ++ (show inum) ++ ") " ++ (printablePerson i) ++ "\n" ++ bookText is (inum + 1)

comment = "Імʼя | Фамілія | #Телефону | День народження | Група"

showBookM title (Phonebook pList mList) =

do putStrLn $ "\n" ++ createLabel title 100 '-'

putStrFlush $ " \*) " ++ comment ++ "\n"

putStrFlush $ bookTextM mList 1

putStrLn $ createLabel "-" 100 '-'

where

bookTextM [] \_ = []

bookTextM (i:is) inum = " " ++ (show inum) ++ ") " ++ (printableMeeting i) ++ "\n" ++ bookTextM is (inum + 1)

comment = "Назва | Місце | Дата | Проведено?"

pressEnter = promptLine "ENTER <--" >> return ()

showMenu :: [Char] -> [([Char], IO a)] -> IO a

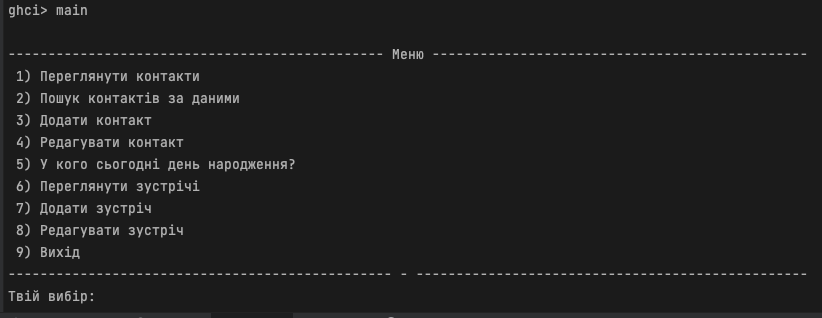
showMenu title actionList = do showItems title [fst i | i <- actionList]

choice <- prompt' "Твій вибір" (\c -> c >= 1 && c <= length actionList)

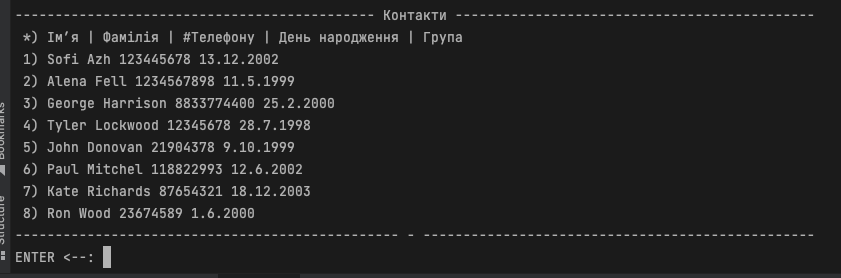
snd $ actionList !! (choice - 1)

**Приклад роботи**

Головне меню

****

Переглянути контакти

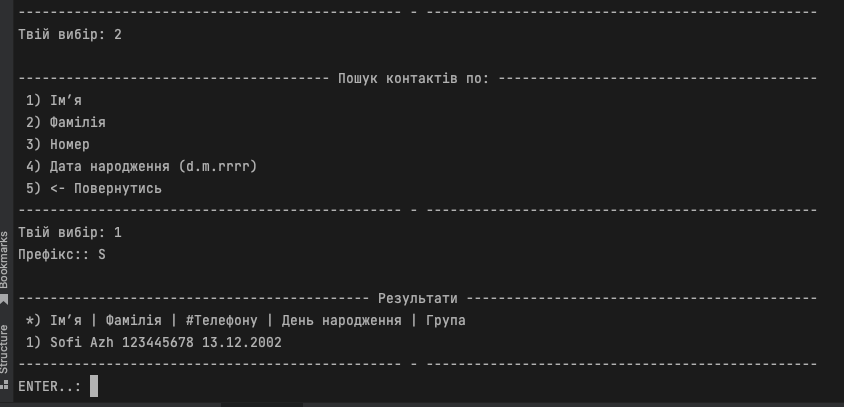


Enter

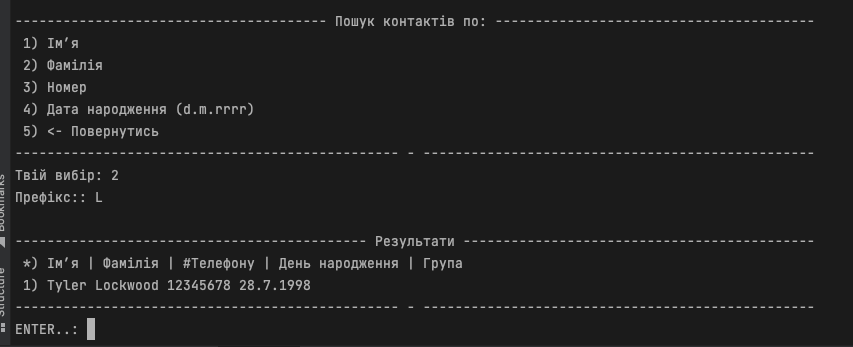


Пошук контактів за даними

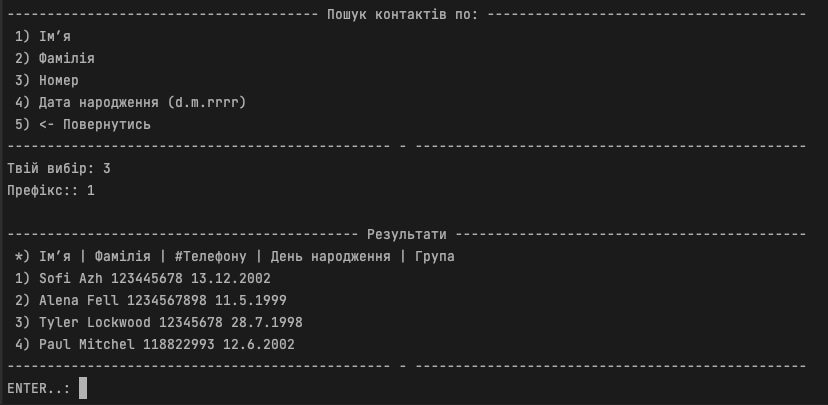
Пошук за імʼям



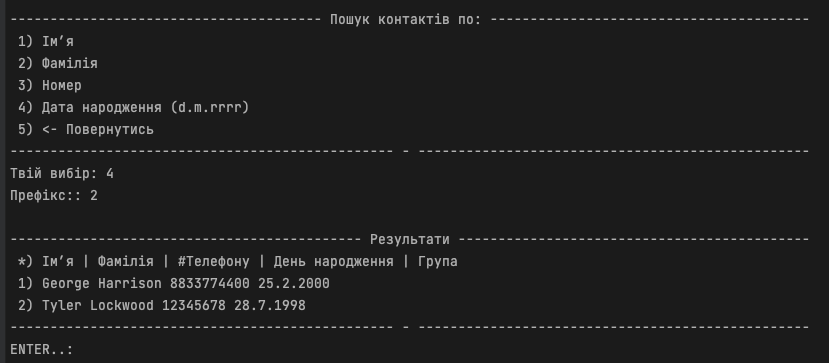
Пошук за фамілією



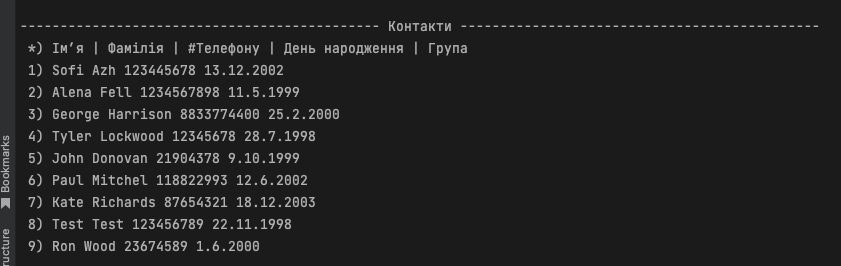
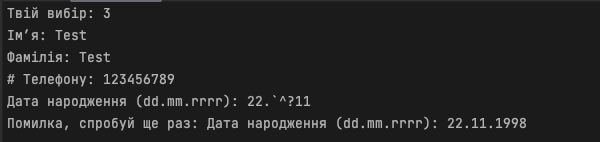
Пошук за номером



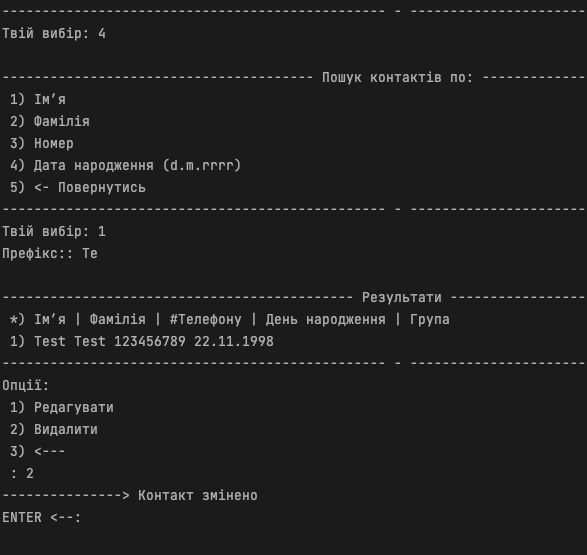
Пошук контактів за датою народження

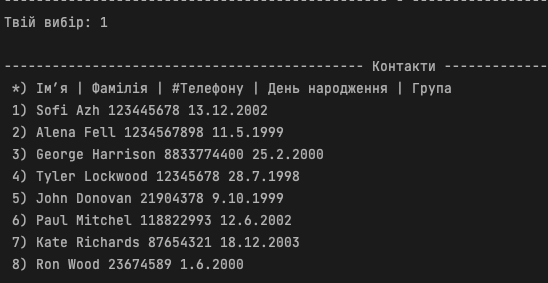


Додавання контакту

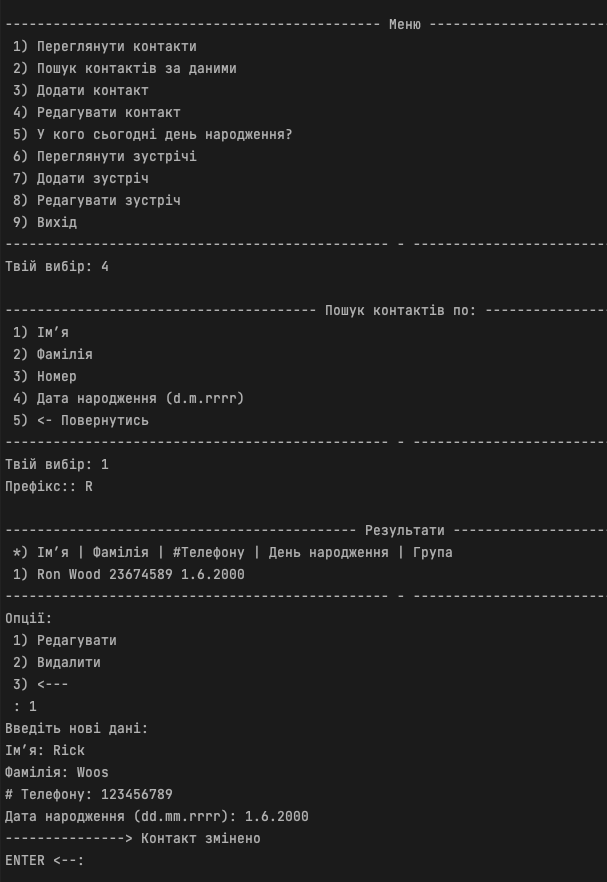


Видалення контакту

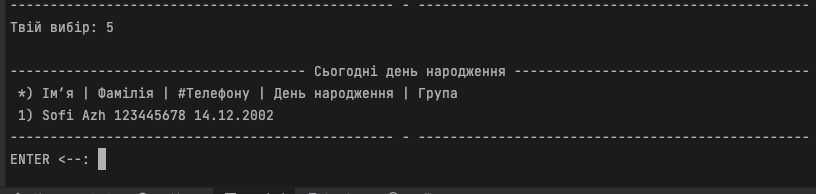




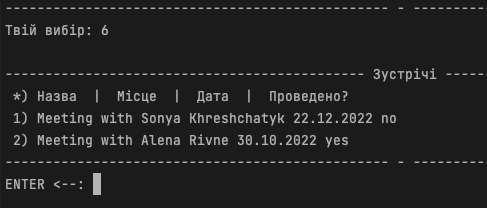
Редагування контакту



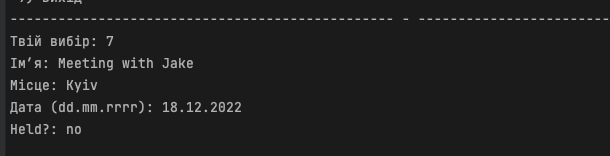
Нагадування про день народження

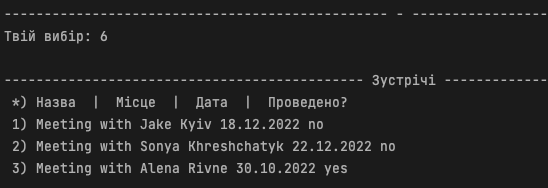


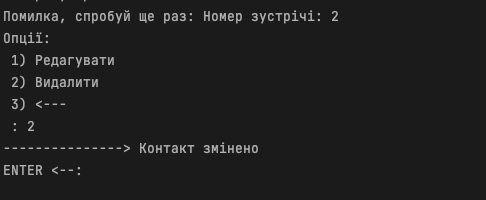
Перегляд зустрічей

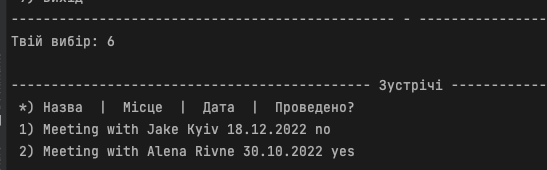


Додавання зустрічей





Видалення зустрічі 



Вихід

