Задача A. Задача один

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

```
module Tasks where

import Prelude hiding (head, tail, take, drop, filter, foldl, foldr, concat, (++), zip, map)

-- В этом задании запрещается использовать какие-либо функции из стандартной библиотеки,
-- кроме конструкторов списков и операторов сравнения из ‘Ord‘.
-- Также запрещается использовать list comprehension.
-- Однако разрешается использовать дополнительные функции, реализованные самостоятельно.

-- 1. head’ возвращает первый элемент непустого списка
head’ :: [a] -> a
head’ = undefined

-- 2. tail’ возвращает список без первого элемента, для пустого - пустой
tail’ :: [a] -> [a]
tail’ = undefined

-- 3. take’ возвращает первые n >= 0 элементов исходного списка;
-- если n больше длины списка, то все элементы
take’ :: Int -> [a] -> [a]
take’ = undefined

-- 4. drop’ возвращает список без первых n >= 0 элементов;
-- если n больше длины списка, то пустой список.
drop’ :: Int -> [a] -> [a]
drop’ = undefined

-- 5. filter’ возвращает список из элементов, для которых f возвращает True
filter’ :: (a -> Bool) -> [a] -> [a]
filter’ = undefined

-- 6. zip’ принимает два списка [a1, a2, ...] и [b1, b2, ...]
-- и возвращает список [(a1, b1), (a2, b2), ...].
-- Размер итогого списка равен размеру меньшего из входных списков.
zip’ :: [a] -> [b] -> [(a, b)]
zip’ = undefined

-- 7. map’ принимает на вход функцию и список и применяет функцию ко всем элементам списка
map’ :: (a -> b) -> [a] -> [b]
map’ = undefined

-- 8. foldr’ последовательно применяет функцию f с конца
-- foldr’ (+) 0 [1, 2, 3] == (1 + (2 + (3 + 0)))
-- foldr’ (*) 4 [] == 4
foldr’ :: (b -> a -> a) -> a -> [b] -> a
foldr’ = undefined
```

```
-- 9. foldl' последовательно применяет функцию f с начала
-- foldl' (+) 0 [1, 2, 3] == (((0 + 1) + 2) + 3)
-- foldl' (*) 4 [] == 4
foldl' :: (a -> b -> a) -> a -> [b] -> a
foldl' = undefined

-- 10. concat' принимает на вход два списка и возвращает их конкатенацию
-- concat' [1,2] [3] == [1,2,3]
concat' :: [a] -> [a] -> [a]
concat' = undefined
```

Замечание

https://github.com/Peltorator/tinkoff_haskell_2020/blob/master/src/Tasks.hs

Задача B. Задача два

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

```
module Tasks2 where

-- Здесь уже можно пользоваться стандартными функциями из Prelude,
-- но нельзя подключать другие модули (Data.Numbers.Primes и т.п.).
--
-- Все функции необходимо реализовать в бесточечном стиле,
-- то есть без использования аргументов в определении.


-- 11. sumOfLists принимает на вход непустой список списков чисел
-- и возвращает сумму этих списков,
-- обрезанную по длине самого короткого списка,
-- то есть если на входе был список [[x1, x2, x3, x4], [y1, y2, y3], [z1, z2, z3, z4, z5]],
-- то итоговым результатом должен быть сисок [x1 + y1 + z1, x2 + y2 + z2, x2 + y3 + z3].
sum_of_lists :: Num a => [[a]] -> [a]
sum_of_lists = undefined

-- 12. n'thPrime принимает на вход число n >= 0 и возвращает n-е простое число.
n'th_prime :: Int -> Int
n'th_prime = undefined

-- 13. tails' возвращает все хвосты входного списка,
-- то есть если на входе был список [1, 2, 3],
-- то итоговым результатом должен быть список [[1, 2, 3], [2, 3], [3], []]
-- Необходимо реализовать функцию при помощи foldr.
tails' :: [a] -> [[a]]
tails' = undefined

-- 14. inits' возвращает все префиксы входного списка,
-- то есть если на входе был список [1, 2, 3],
-- то итоговым результатом должен быть список [[], [1], [1, 2], [1, 2, 3]]
-- Необходимо реализовать функцию при помощи foldr.
inits' :: [a] -> [[a]]
inits' = undefined

-- 15. reverse' переворачивает список, который был дан на входе.
-- Необходимо реализовать функцию при помощи foldr.
reverse' :: [a] -> [a]
reverse' = undefined

-- 16. reverse'' переворачивает список, который был дан на входе.
-- Необходимо реализовать функцию при помощи foldl.
reverse'' :: [a] -> [a]
reverse'' = undefined
```

Замечание

https://github.com/Peltorator/tinkoff_haskell_2020/blob/master/src/Tasks2.hs

Задача C. Задача три

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

```
module Tasks3 where

-- Здесь нельзя использовать deriving кроме Show.



-- 17. Тренировочная задача на знакомств с пользовательскими типами. Дан тип данных Person:
data Person = Person { firstName :: String, lastName :: String, age :: Int }
    deriving (Show, Eq)
-- Функция abbrFirstName сокращает имя мя до первой буквы с точкой,
-- то есть если имя было "John", то после применения этой функции, оно превратится в "J.".
-- Однако если имя было короче двух символов, то оно не меняется.
-- P.S. Если что, строка -- список символов.
abbrFirstName :: Person -> Person
abbrFirstName = undefined


-- Определим наше дерево, которое мы далее будем использовать:

data Tree a = Nil | Node (Tree a) a (Tree a)
    deriving Show

-- 18. Функция treeSum вычисляет сумму элементов дерва.
treeSum :: Tree Integer -> Integer
treeSum = undefined


-- 19. Функция treeHeight вычисляет максимальную высоту дерева.
treeHeight :: Tree a -> Int
treeHeight = undefined


-- 20. Сделайте Tree представителем класса типов Eq.
instance Eq a => Eq (Tree a) where
    (==) = undefined

-- Для реализации свертки двоичных деревьев нужно выбрать алгоритм обхода узлов дерева.
-- Сделайте дерево представителем класса типов Foldable несколькими способами.
-- Так как нельзя одно и то же дерево сделать Foldable несколькими способами,
-- мы заведем ему псевдонимы:
newtype Preorder a   = PreO   (Tree a) deriving (Eq, Show)
newtype Postorder a  = PostO  (Tree a) deriving (Eq, Show)
newtype Levelorder a = LevelO (Tree a) deriving (Eq, Show)
-- В данном контексте можно считать, что newtype -- это то же самое, что data,
-- Но конструктор данных только один. Это обертка над деревом.
--
-- Теперь сделайте 4 представителя класса типов Foldable:
```

```
-- Tree в порядке левое поддерево - вершина - правое поддерево;
-- Preorder в порядке вершина - левое поддерево - правое поддерево;
-- Postorder в порядке левое поддерево - правое поддерево - вершина;
-- Levelorder в порядке bfs (по уровням, на одном уровне -- слева направо).
--
-- 21.
instance Foldable Tree where
    foldr = undefined

-- 22.
instance Foldable Preorder where
    foldr = undefined

-- 23.
instance Foldable Postorder where
    foldr = undefined

-- 24.
instance Foldable Levelorder where
    foldr = undefined

-- 25. treeSum' вычисляет сумму элементов дерева. Примените foldr.
treeSum' :: Tree Integer -> Integer
treeSum' = undefined



-- Определим наш список, который мы далее будем использовать:
data MyList a = Empty | Cons a (MyList a)
    deriving Show

-- 26. Сделайте MyList представителем класса типов Eq.
instance Eq a => Eq (MyList a) where
    (==) = undefined

-- 27. Сделайте MyList представителем класса типов Ord. Достаточно реализовать оператор (<=).
instance Ord a => Ord (MyList a) where
    (<=) = undefined

-- 28. Сделайте MyList представителем класса типов Foldable.
instance Foldable MyList where
    foldr = undefined

-- 29. Сделайте MyList представителем класса типов Functor.
instance Functor MyList where
    fmap = undefined

-- 30. sum2D вычисляет сумму элементов двумерного списка.
-- Используйте реализованные выше instance'ы, чтобы сделать все в бесточечном стиле.
sum2D :: Num a => MyList (MyList a) -> a
sum2D = undefined
```

Замечание

https://github.com/Peltorator/tinkoff_haskell_2020/blob/master/src/Tasks3.hs

Задача D. A+B

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Вам заданы два целых числа a и b. Выведите $a + b$.

Формат входных данных

Единственная строка входных данных содержит два целых числа a и b ($-100 \leqslant a, b \leqslant 100$).

Формат выходных данных

Выведите $a + b$.

Примеры

стандартный ввод	стандартный вывод
7 8	15
-100 100	0
-7 -99	-106

Замечание

В первом примере $a = 7$ и $b = 8$. Таким образом, ответ равен $a + b = 7 + 8 = 15$.

Задача E. Переворот

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Дано натуральное число N и последовательность из N элементов. Требуется вывести эту последовательность в обратном порядке.

Формат входных данных

В первой строке входного файла записано натуральное число N ($N \leqslant 10^5$).

В следующих N строках идут N целых чисел, по модулю не превосходящих 10^9, — элементы последовательности.

Формат выходных данных

В выходной файл выведите заданную последовательность в обратном порядке.

Пример

стандартный ввод	стандартный вывод
3 6 8 -3	-3 8 6

Задача F. Дерево отрезков с операцией на отрезке

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	0.5 секунд
Ограничение по памяти:	256 мегабайт

Реализуйте эффективную структуру данных для хранения элементов и увеличения нескольких подряд идущих элементов на одно и то же число.

Формат входных данных

В первой строке вводится одно натуральное число N ($1 \leqslant N \leqslant 100\,000$) — количество чисел в массиве.

Во второй строке вводятся N чисел от 0 до 100 000 — элементы массива.

В третьей строке вводится одно натуральное число M ($1 \leqslant M \leqslant 30\,000$) — количество запросов.

Каждая из следующих M строк представляет собой описание запроса. Сначала вводится одна буква, кодирующая вид запроса (g — получить текущее значение элемента по его номеру, a — увеличить все элементы на отрезке).

Следом за g вводится одно число — номер элемента.

Следом за a вводится три числа — левый и правый концы отрезка и число add, на которое нужно увеличить все элементы данного отрезка массива ($0 \leqslant add \leqslant 100\,000$).

Формат выходных данных

Выведите в одну строку через пробел ответы на каждый запрос g.

Пример

стандартный ввод	стандартный вывод
5 2 4 3 5 2 5 g 2 g 5 a 1 3 10 g 2 g 4	4 2 14 5

Задача G. Двоичное дерево поиска

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Реализуйте сбалансированное двоичное дерево поиска.

Формат входных данных

Входной файл содержит описание операций с деревом, их количество не превышает 100000. В каждой строке находится одна из следующих операций:

- `insert` x — добавить в дерево ключ x. Если ключ x в дереве уже есть, то ничего делать не надо.
- `delete` x — удалить из дерева ключ x. Если ключа x в дереве нет, то ничего делать не надо.
- `exists` x — если ключ x есть в дереве, выведите «`true`», иначе «`false`»
- `next` x — выведите минимальный элемент в дереве, строго больший x, или «`none`», если такого нет.
- `prev` x — выведите максимальный элемент в дереве, строго меньший x, или «`none`», если такого нет.

Все числа во входном файле целые и по модулю не превышают 10^9.

Формат выходных данных

Выведите последовательно результат выполнения всех операций `exists`, `next`, `prev`. Следуйте формату выходного файла из примера.

Пример

стандартный ввод	стандартный вывод
insert 2 insert 5 insert 3 exists 2 exists 4 next 4 prev 4 delete 5 next 4 prev 4	true false 5 3 none 3