ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

Цель лабораторной работы: научиться определять пользовательские типы данных. Получить представление об именованных типах данных.

Ход работы:

1. В современных web-магазинах часто продают книиги, видеокассеты и компакт-диски. База данных такого магазина для каждого типа товаров должна содержать следующие характеристики:
- книги: название и автор
- видеокассеты: название
- компакт-диски: название, исполнитель и количество композиций

1) Разработайте тип данных Product, который может представлять эти виды товаров.

```
data Product =
        Book{
           title     :: String,
           author :: String
        }
        | Videotape{
           title :: String
        }
        | Cd{
           title            :: String,
           performer     :: String,
           count_tracks :: Int
        }
        deriving (Eq, Show)
```

2) Определите функцию getTitle, возвращающую название товара.

```
getTitle :: Product -> String
getTitle product = title product
```

3) На ее основе определите функцию getTitles, которая по списку товаров возвращает список их названий.

```
getTitles :: [Product] -> [String]
getTitles products = [getTitle product | product <- products]
```

4) Определите функцию bookAuthors, которая по списку товаров возвращает список авторов книг.

```
bookAuthors :: [Product] -> [String]
bookAuthors products = [author product | product <- products]
```

5) Определите функцию
lookupTitle :: String -> [Product] -> Maybe Product
которая возвращает товар с заданным названием (обратите внимание на тип результата функции)

```
lookupTitle :: String -> [Product] -> Maybe Product
lookupTitle title products =
   case [product | product <- products, (getTitle product) == title] of
      [] -> Nothing
      [product] -> Just product
      prods -> Just (prods !! 0)
```

6) Определите функцию
lookupTitles :: [String] -> [Product] -> [Product]
Она принимает в качестве параметров список названий и список товаров и для каждого названия извлекает из второго списка соответствующие товары. Названия, которым не соответствует никакой товар, игнорируется. При определении функции обязательно используйте функцию lookupTitle

```
lookupTitles :: [String] -> [Product] -> [Product]
lookupTitles titles products = lookupTitles' titles products []
lookupTitles' [] products acc = reverse acc
lookupTitles' (title:rest) products acc =
   case lookupTitle title products of
      Just product ->
         lookupTitles' rest products (product:acc)
      _ ->
         lookupTitles' rest products acc
```

Пример работы:

```
*Main> let cd = Cd "Plug Me In" "AC/DC" 5
*Main> let video = Videotape "Gladiator"
*Main> let book = Book "Learn You a Haskell for Great Good!" "Miran Lipovaca"
*Main> book
Book {title = "Learn You a Haskell for Great Good!", author = "Miran Lipovaca"}
*Main> :t book
book :: Product
*Main> getTitle cd
"Plug Me In"
*Main> getTitle video
"Gladiator"
*Main> getTitle book
"Learn You a Haskell for Great Good!"
*Main> getTitles [book,video,cd]
["Learn You a Haskell for Great Good!","Gladiator","Plug Me In"]
*Main> lookupTitle "Plug Me In" [book,video,cd]
Just (Cd {title = "Plug Me In", performer = "AC/DC", count_tracks = 5})
*Main> let cd2 = Cd "Fly on the Wall" "AC/DC" 10
*Main> lookupTitles ["Fly on the Wall", "Plug Me In"] [book,cd,video,cd2]
[Cd {title = "Fly on the Wall", performer = "AC/DC", count_tracks = 10},Cd {titl
e = "Plug Me In", performer = "AC/DC", count_tracks = 5}]
*Main> let book2 = Book "Programming Erlang!" "Joe Armstrong"
*Main> bookAuthors [book,book2]
["Miran Lipovaca","Joe Armstrong"]
```

2. Определите тип данных, представляющий информацию о карте в карточной игре. Каждая карта характеризуется одной из четырех мастей. Карта может быть либо младшей (от двойки до десятки), либо картинкой (валет, дама, король, туз).

```
data GameCard = GameCard{ suit  :: String -- масть
                        , value :: Int -- значение
                        } deriving (Eq, Show)
```

Определите функции:

1) Функция isMinor, проверяющая, что ее аргумент является младшей картой.

```
isMinor :: GameCard -> Bool
isMinor GameCard{value = value} = if value < 11 then True else False
```

2) Функция sameSuit, проверяющая, что переданные в нее карты — одной масти.

```
sameSuit :: [GameCard] -> Bool
sameSuit cards@(head:_tail) =
    let GameCard{suit=firstSuit} = head in
    let res = [1 | GameCard{suit=suit} <- cards, suit == firstSuit] in
    sum res == length cards
```

3) Функция beats :: Card -> Card -> Bool, проверяющая, что карта, переданная ей в качестве первого аргумента, бьет карту, являющуюся вторым аргументом.

```
beats :: GameCard -> GameCard -> Bool
beats GameCard{value=v1} GameCard{value=v2}
    | (v1 == 11) && (v2 /= 11) = True -- если первая карта туз, а вторая нет
    | (v2 == 11) = False -- если вторая карта туз
    | (v1 > v2) = True -- если первая карта больше второй
    | otherwise = False
```

4) Функция beats2, аналогичная beats, но принимающая в качестве дополнительно аргумента козырную масть.

```
beats2 :: GameCard -> GameCard -> String -> Bool
beats2 GameCard{suit=s1, value=v1} GameCard{suit=s2, value=v2} trump
    | (t1 == True) && (t2 == False) = True -- если первая карта козырь, а вторая нет
    | (t1 == False) && (t2 == True) = False -- если первая карта простая, а вторая козырь

    | (t1 == True) && (v1 == 11) = True -- если первая карта козырный туз
    | (t2 == True) && (v2 == 11) = False -- если вторая карта козырный туз
    | (v1 > v2) && (t1 == True) && (t2 == True) = True -- если первая карта больше второй, обе
  козырные
    | (v1 < v2) && (t1 == True) && (t2 == True) = False -- если первая карта меньше второй, обе
  козырные

    | (v1 == 11) && (v2 /= 11) = True -- если первая карта туз, а вторая нет
    | (v2 == 11) = False -- если вторая карта туз
    | (v1 > v2) = True -- если первая карта больше второй
    | otherwise = False
    where
    t1 = (s1 == trump)
    t2 = (s2 == trump)
```

5) Функция beatsList, принимающая в качестве аргументов список карт, карту и козырную масть и возвращающая список тех карт из первого арнумента, которые бьют указанную карту с учетом козырной масти.

```
beatsList :: [GameCard] -> GameCard -> String -> [GameCard]
beatsList cards comparCard trump = [card | card <- cards, beats2 card comparCard trump]
```

6) Функция, по заданному списку карт возвращающая список чисел, каждое из которых является возможной суммой очков указанных карт, рассчитанных по правилам игры в «двадцать одно»: младшие карты считаются по номиналу, валет, дама и король считаются за 10 очков, туз может рассматриваться и как 1 и как 11 очков. Функция должна вернуть все возможные варианты.

```haskell
check21 :: [GameCard] -> [Int]
check21 cards =
  let values = [v | GameCard{value=v} <- cards] in --получает значение карт
  let newVal = map (\x -> if x > 11 then 10 else x) values in --преобразование вальтов, дам и
 королей
  if 11 `elem` newVal then --если в списке есть туз
    let newVal2 = map (\x -> if x == 11 then 1 else x) newVal in -- заменяет 11 на 1
    [sum newVal, sum newVal2] --считает обычную и новую суммы
  else --если в списке нет туза
    [sum newVal] --считает сумму
```

Пример работы:

```
*Main> let card = GameCard{suit="hearts",value=6}
*Main> card
GameCard {suit = "hearts", value = 6}
*Main> :t card
card :: GameCard
*Main> isMinor card
True
*Main> sameSuit [card,card]
True
*Main> let card2 = GameCard{suit="spades",value=6}
*Main> sameSuit [card,card2]
False
*Main> beats card card2
False
*Main> beats card2 card
False
*Main> let card2 = GameCard{suit="spades",value=11}
*Main> beats card2 card
True
*Main> beats2 card card2 "hearts"
True
*Main> beats2 card card2 "spades"
False
*Main> beatsList [card,card2] card "spades"
[GameCard {suit = "spades", value = 11}]
*Main> beatsList [card,card2] card "hearts"
[]
*Main> check21 [card,card2]
[17,7]
*Main> let card3 = GameCard{suit="spades",value=14}
*Main> check21 [card,card2,card3]
[27,17]_
```

Вывод: научился определять пользовательские типы данных. Получил представление об именованных типах данных.