

# Языки программирования (осень 2018)

В начало ► Мои курсы ► ЯП 2018 ► Оценка лабораторных работ ► Лабораторная работа №2 ► Работа

## Лабораторная работа №2

### Моя работа

#### Инструкции для работы ▼

Для того, чтобы отправить работу на оценку, нажмите "Начало подготовки Вашей работы".

На открывшейся странице:

- в поле **Название** появившегося окна укажите точное название загружаемого файла (строчными буквами, с расширением, без пробелов);
- поле **Содержимое работы** оставьте пустым;
- из папки с решением перетащите загружаемый файл в поле **Приложение** или загрузите файл в это поле, используя кнопку "Добавить.." в меню этого поля;
- выполнив перечисленные пункты нажмите кнопку "Сохранить".

При необходимости, пока не окончена фаза представления работ, можно откорректировать представление работы нажав кнопку "Редактировать работу"

### lab-2

представлено: Воскресенье, 7 Октябрь 2018, 18:32

-  lab-2.sml



#### Самооценка

от Максим Кулаков

Оценка: 88,13 из 100,00

#### Форма оценки ▼

#### Критерий 1

Функция `getNth` может иметь вид

```
fun getNth (c :: _, 0) = c
  | getNth (_ :: cs, n) = getNth (cs, n - 1)
  | getNth _ = raise List.Empty
```

- Не следует ставить больше 3-х баллов, если логика оцениваемой функции сложнее, чем у приведенной выше.

- Стоит снизить оценку на 1 балл, если при сравнениях с шаблонами в функции рассматривается более 3-х случаев.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 1

5

### Комментарий к Критерий 1

## Критерий 2

---

Функция `reverseAppend` может иметь вид

```
fun reverseAppend ([], rez)      = rez
  | reverseAppend (x :: xs, rez) = reverseAppend (xs, x :: rez)
```

- Не следует ставить больше 3-х баллов, если логика оцениваемой функции сложнее, чем у приведенной выше.
- Стоит снизить оценку на 1 балл, если при сравнениях с шаблонами в функции рассматривается более 2-х случаев.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 2

5

### Комментарий к Критерий 2

## Критерий 3

---

Функция `cardValue` может иметь вид

```
fun cardValue c =
  case c
  of (_, NUM n) => n
   | (BLACK, _) => 50
   | _          => 20
```

- Следует снизить оценку на 1 балл, если происходит сравнение более чем с тремя образцами (шаблонами).
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 3

4

### Комментарий к Критерий 3

Чёрные карты можно было объединить в один шаблон + лишние скобки

## Критерий 4

---

Функция `cardCount` может иметь вид

```
fun cardCount c =  
  case c  
  of (_, NUM 0) => 1  
    | (BLACK, _) => 4  
    | _         => 2
```

- Следует снизить оценку на 1 балл, если происходит сравнение более чем с тремя образцами (шаблонами).
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 4

4

### Комментарий к Критерий 4

Чёрные карты можно было объединить в один шаблон + лишние скобки

## Критерий 5

---

Функция `rankColors` может иметь вид

```
fun rankColors r =  
  case r  
  of WILD           => [BLACK]  
    | WILD_DRAW_FOUR => [BLACK]  
    | _             => [RED, GREEN, BLUE, YELLOW]
```

- Следует снизить оценку на 1 балл, если происходит сравнение более чем с тремя образцами (шаблонами).
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 5

4

### Комментарий к Критерий 5

лишние скобки

## Критерий 6

---

Функция `sumCards` может иметь вид

```
fun sumCards cs =  
  let  
    fun sumCardsIter ([], accum) = accum  
      | sumCardsIter (c :: cs, accum) =  
          sumCardsIter (cs, cardValue c + accum)  
  in sumCardsIter (cs, 0)  
  end
```

- Не следует ставить больше 3-х баллов, если логика оцениваемой функции сложнее, чем у приведенной выше.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 6

5

### Комментарий к Критерий 6

## Критерий 7

---

Функция `removeNth` может иметь вид

```
fun removeNth (cs, n) =  
  let  
    fun removeNthIter (cs', c :: cs, 0) = reverseAppend (cs', cs)  
      | removeNthIter (cs', c :: cs, n) =  
          removeNthIter (c :: cs', cs, n - 1)  
      | removeNthIter _ = raise List.Empty  
  in removeNthIter ([], cs, n)  
  end
```

- Не следует ставить больше 3-х баллов, если в функции не используется `reverseAppend`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 7

5

### Комментарий к Критерий 7

## Критерий 8

---

Функция `removeCard` может иметь вид

```
fun removeCard (cs, c, e) =  
  let  
    fun removeCardIter (_, []) = raise e  
      | removeCardIter (seen, s :: unseen) =  
          if isSameCard (s, c)  
          then reverseAppend (seen, unseen)  
          else removeCardIter (s :: seen, unseen)  
  in removeCardIter ([], cs)  
  end
```

- Не следует ставить больше 3-х баллов, если в функции не используется `isSameCard` или `reverseAppend`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 8

5

### Комментарий к Критерий 8

## Критерий 9

---

Функция `insertElem` может иметь вид

```
fun insertElem (cs, c, n) =  
  let  
    fun insertElemIter (cs', cs, 0) = reverseAppend (cs', c :: cs)  
      | insertElemIter (cs', c' :: cs, n) =  
          insertElemIter (c' :: cs', cs, n - 1)  
      | insertElemIter _ = raise List.Empty  
  in insertElemIter ([], cs, n)  
  end
```

- Не следует ставить больше 3-х баллов, если в функции не используется `reverseAppend`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 9

5

### Комментарий к Критерий 9

## Критерий 10

---

Функция `interchange` может иметь вид

```
fun interchange (cs, i, j) =
  let
    val a = getNth (cs, i)
    val b = getNth (cs, j)
    fun interchangeAux (i, j, a, b) =
      let
        val cs' = removeNth (removeNth (cs, j), i)
      in insertElem (insertElem (cs', b, i), a, j)
      end
  in
    if i = j then cs
    else if i < j then interchangeAux (i, j, a, b)
    else interchangeAux (j, i, b, a)
  end
```

Другой вариант функции - без вспомогательной функции

```
fun interchange (cs, i, j) =
  let
    val a = getNth (cs, i)
    val b = getNth (cs, j)
    val tempList1 = insertElem (removeNth (cs, i), b, i)
    val tempList2 = insertElem (removeNth (tempList1, j), a, j)
  in tempList2
  end
```

- Не следует ставить больше 3-х баллов, если в функции не используется `insertElem` или `removeNth`.
- Не стоит ставить больше 4-х баллов, если решение представляет более сложный алгоритм, чем представленные выше.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 10

4

## Комментарий к Критерий 10

форматирование

## Критерий 11

Функция `shuffleList` может иметь вид

```

fun shuffleList l =
  let
    val seed = seed ()
    val rnd = Random.rand (seed mod 67, seed)
    val maxNum = length l - 1
    fun getNum i = Random.randRange (i, maxNum) rnd
    fun shuffleIter (~1, l) = l
      | shuffleIter (i, l) =
          shuffleIter (i - 1, interchange (l, i, getNum i))
  in shuffleIter (maxNum - 1, l)
  end

```

- Обратите внимание, что случайная позиция для перестановки должна генерироваться для всех элементов исходного списка, кроме одного. Если функция генерации случайной позиции вызывается меньшее количество раз, то решение стоит оценить не более чем на 2 балла.
- Если элементы списка просматриваются для `i` от `n - 2` до `0` (или наоборот), то номер случайной позиции должен выбираться от `i` до `n - 1`. Если элементы списка просматриваются для `i` от `n - 1` до `1` (или наоборот), то номер случайной позиции должен выбираться от `0` до `i`. Если выбор случайной позиции происходит из другого диапазона, то решение следует оценить не более чем на 2 балла.
- Определение источника случайных чисел (результат `Random.rand`) должно происходить вне функции, организующей цикл. В противном случае случайные числа извлекаются из разных источников на разных итерациях и числа перестают быть "случайными". Если это не так, оценку за решение следует снизить на 2 балла.
- Если нет вызова `seed()` для генерации аргумента функции `rand`, то решение следует оценить не более чем на 3 балла.
- Если в решении не используется функция `rand` или `randRange`, то решение следует оценить не более чем на 2 балла.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 11

5

## Комментарий к Критерий 11

## Критерий 12

Функция `allRankColors` может иметь вид

```

fun allRankColors r =
  map (fn col => (col, r)) (rankColors r)

```

- Не следует ставить больше 3-х баллов, если в функции не используется `rankColors` или `map`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 12

## Комментарий к Критерий 12

## Критерий 13

Функция `copyCardNTimes` может иметь вид

```
fun copyCardNTimes c =  
  let  
    fun copyNTimesIter (0, l) = l  
      | copyNTimesIter (i, l) = copyNTimesIter (i - 1, c :: l)  
  in copyNTimesIter (cardCount c, [])  
  end
```

- Так как в списке-результате будут содержаться карты только одного достоинства, то порядок элементов в списке не важен. Поэтому, если в решении используется функция `reverseAppend` или ей подобная, то оценку следует снизить на 1 балл.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 13

5

## Комментарий к Критерий 13

## Критерий 14

Список `deck` может определяться следующим образом

```
val deck = foldl (op @) [] (map copyCardNTimes  
                           (foldl (op @) [] (map allRankColors ranks)))
```

- В решении могут присутствовать дополнительные обозначения для повышения читабельности кода.
- Не стоит оценивать решение более чем на 3 балла, если не используется `map`, `allRankColors` или `copyCardNTimes`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 14

5

## Комментарий к Критерий 14

## Критерий 15



Функция `getSameRank` может иметь вид

```
fun getSameRank (r, cs) =  
  List.filter (fn (_, r') => isSameRank (r', r)) cs
```

- Не стоит оценивать решение более чем на 3 балла, если в нем не используется `List.filter` или `isSameRank`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 15

4

### Комментарий к Критерий 15

форматирование отступов

## Критерий 16

Функция `getSameColor` может иметь вид

```
fun getSameColor (col, cs) =  
  List.filter (fn (col', _) => isSameColor (col', col)) cs
```

- Не стоит оценивать решение более чем на 3 балла, если в нем не используется `List.filter` или `isSameColor`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 16

4

### Комментарий к Критерий 16

форматирование отступов

## Критерий 17

Функция `hasRank` может иметь вид

```
fun hasRank (r, cs) =  
  isSome (List.find (fn (_, r') => isSameRank (r', r)) cs)
```

- Не стоит оценивать решение более чем на 3 балла, если в нем не используется `List.find` или `isSameRank`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 17

4

### Комментарий к Критерий 17

форматирование отступов

## Критерий 18

---

Функция `hasColor` может иметь вид

```
fun hasColor (col, cs) =  
  isSome (List.find (fn (col', _) => isSameColor (col', col)) cs)
```

- Не стоит оценивать решение более чем на 3 балла, если в нем не используется `List.find` или `isSameColor`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 18

4

### Комментарий к Критерий 18

форматирование отступов

## Критерий 19

---

Функция `hasCard` может иметь вид

```
fun hasCard (c, cs) =  
  isSome (List.find (curry_2 isSameCard c) cs)
```

- Не стоит оценивать решение более чем на 3 балла, если в нем не используется `List.find` или `isSameCard`.
- Не стоит оценивать решение более чем на 4 балла, если в нем не используется `curry2`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 19

4

### Комментарий к Критерий 19

форматирование отступов

## Критерий 20

---

Функция `countColor` может иметь вид

```
fun countColor (col, cs) =  
  length (getSameColor (col, cs))
```

- Не стоит оценивать решение более чем на 3 балла, если в нем не используется `getSameColor` или `length`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 20

4

### Комментарий к Критерий 20

форматирование отступов

## Критерий 21

---

Функция `maxColor` может иметь вид

```
fun maxColor cs =  
  let  
    fun countColors cs =  
      map (fn c => (c, countColor (c, cs)))  
        [RED, GREEN, BLUE, YELLOW]  
    fun greatest (p1 as (_, n1), p2 as (_, n2)) =  
      if n1 > n2 then p1 else p2  
    fun max (p :: ps) =  
      foldl greatest p ps  
      | max _ = raise IllegalGame  
    val (col, _) = max (countColors cs)  
  in col  
  end
```

- Максимум в решении может искаться по разному. Вспомогательные функции не обязаны присутствовать. Но обязательно должна проходить (явно или неявно) обработка списка `[RED, GREEN, BLUE, YELLOW]`. Если ее нет, возможно стоит снизить балл.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 21

5

### Комментарий к Критерий 21

## Критерий 22

---

Функция `deal` может иметь вид

```
fun deal names =
  let
    fun dealIter ([], ps, c :: d) =
      makeDesk (reverseAppend (ps, []), [c], d, PROCEED)
    | dealIter ((n, f) :: ns, ps, d) =
      dealIter ( ns
                , makePlayer (n, List.take (d, 7), f) :: ps
                , List.drop (d, 7) )
    | dealIter _ = raise IllegalGame
  in dealIter (names, [], shuffleList deck)
end
```

- Дайте за решение не более 2-х баллов, если колода тасуется более одного раза.
- Обратите внимание, что должен быть реализован итерационный процесс, при котором каждому игроку выделяется семь карт из колоды, а из колоды, соответственно, семь картудаляются.
- Дайте за решение не более 1 балла, если в результате раздачи карт порядок игроков меняется.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 22

4

## Комментарий к Критерий 22

неправильные переносы

## Критерий 23

---

Функция `getPlayersFirst` может иметь вид

```
fun getPlayersFirst dsk =
  let val p :: _ = getDeskPlayers dsk in p end
```

- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 23

5

## Комментарий к Критерий 23

## Критерий 24

---

Функция `getPileTop` может иметь вид

```
fun getPileTop dsk =  
  let val c :: _ = getDeskPile dsk in c end
```

- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 24

5

### Комментарий к Критерий 24

## Критерий 25

---

Функция `nextPlayer` может иметь вид

```
fun nextPlayer dsk =  
  let val p :: ps = getDeskPlayers dsk  
  in setDeskPlayers (dsk, ps @ [p])  
  end
```

- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 25

5

### Комментарий к Критерий 25

## Критерий 26

---

Функция `changeDirection` может иметь вид

```
fun changeDirection dsk =  
  let val p :: ps = getDeskPlayers dsk  
  in setDeskPlayers (dsk, p :: reverseAppend (ps, []))  
  end
```

- Не следует ставить больше 3-х баллов, если в функции не используется `reverseAppend`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 26

5

## Критерий 27

Функция `start` может иметь вид

```
fun start dsk =
  let val {players = ps, pile = [c], deck = ds, ...} = dsk
  in case c
    of (_, REVERSE) => changeDirection dsk
      | (BLACK, _)   => let val c' :: ds' = shuffleList (c :: ds)
                        in start (makeDesk (ps, [c'], ds', PROCEED))
                        end
      | (_, NUM _)   => nextPlayer dsk
      | _            => setDeskState (nextPlayer dsk, EXECUTE)
  end
end
```

- Не стоит ставить более 4-х баллов, если при сравнении с шаблонами рассматривается больше 4-х случаев.
- Не следует ставить больше 4-х баллов, если в случае черной карты наверху колоды `pile` нет рекурсивного вызова.
- Не следует ставить больше 3-х баллов, если в функции не используется `changeDirection`, `makeDesk`, `nextPlayer` или `setDeskState`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 27

4

### Комментарий к Критерий 27

не неверная табуляция

## Критерий 28

Функция `takeOne` может иметь вид

```
fun takeOne dsk =
  case dsk
  of {players = ps, pile = c :: cs, deck = [], state = move} =>
      takeOne (makeDesk (ps, [c], shuffleList cs, move))
      | {players = p :: ps, pile = cs, deck = d :: ds, state = move} =>
          makeDesk ( setPlayerCards (p, d :: getPlayerCards p) :: ps
                    , cs, ds, move )
      | _ => raise IllegalGame
```

- Не следует ставить больше 4-х баллов, если в случае отсутствия карт в колоде `deck` нет рекурсивного вызова.
- Не следует ставить больше 3-х баллов, в функции не используется `makeDesk` или `setPlayerCards`.

- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 28

4

### Комментарий к Критерий 28

неверный перенос

## Критерий 29

Функция `takeTwo` может иметь вид

```
val takeTwo = takeOne o takeOne
```

- Не следует ставить больше 4-х баллов, если не используется операция композиции `o`.
- Стоит снизить оценку на один балл, если в решении присутствует дополнительное "оборачивание" в функцию, т.е. решение представлено в виде, подобном следующему:

```
fun takeTwo dsk = (takeOne o takeOne) dsk
```

- Не следует ставить больше 3-х баллов, в функции не используется `takeOne`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 29

4

### Комментарий к Критерий 29

дополнительное "оборачивание" в функцию

## Критерий 30

Функция `takeFour` может иметь вид

```
val takeFour = takeTwo o takeTwo
```

- Не следует ставить больше 4-х баллов, если не используется операция композиции `o`.
- Стоит снизить оценку на один балл, если в решении присутствует дополнительное "оборачивание" в функцию, т.е. решение представлено в виде, подобном следующему:

```
fun takeFour dsk = (takeTwo o takeTwo) dsk
```

- Не следует ставить больше 3-х баллов, в функции не используется `takeTwo`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 30

4

### Комментарий к Критерий 30

дополнительное "оборачивание" в функцию

## Критерий 31

Функция `pass` может иметь вид

```
fun pass dsk =  
  case (getDeskState dsk, getPileTop dsk)  
  of (EXECUTE, (_, DRAW_TWO)) =>  
      setDeskState (nextPlayer (takeTwo dsk), PROCEED)  
  | (EXECUTE, (_, SKIP))      => setDeskState (nextPlayer dsk, PROCEED)  
  | _                        => nextPlayer dsk
```

- Не следует ставить больше 3-х баллов, в функции не используется `setDeskState` или `nextPlayer`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 31

4

### Комментарий к Критерий 31

не хватает пробелов

## Критерий 32

Функция `requiredColor` может иметь вид

```
fun requiredColor dsk =  
  case (getPileTop dsk, getDeskState dsk)  
  of (_, GIVE col) => col  
  | ((col, _), _) => col
```

- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 32

4

### Комментарий к Критерий 32

не хватает пробелов

## Критерий 33



## Критерий 33

Функция `playableCards` может иметь вид

```
fun playableCards dsk =
  let
    val filterFunc =
      case (getDeskState dsk, getPileTop dsk)
      of (GIVE col, _) =>
          (fn (col', _) => isSameColor (col, col')
            orelse isSameColor (BLACK, col'))
        | (EXECUTE, (_, r)) =>
          (fn (_, r') => isSameRank (r, r'))
        | (PROCEED, (col, r)) =>
          (fn (col', r') => isSameColor (col, col')
            orelse isSameRank (r, r')
            orelse isSameColor (BLACK, col'))
      in List.filter filterFunc (getPlayerCards (getPlayersFirst dsk))
    end
```

- Не следует ставить больше 3-х баллов, в функции не используется `List.filter`, `isSameColor` или `isSameRank`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 33

5

### Комментарий к Критерий 33

## Критерий 34

Функция `countCards` может иметь вид

```
val countCards =
  (map (length o getPlayerCards)) o getDeskPlayers
```

- Не следует ставить больше 4-х баллов, если не используется операция композиции `o`.
- Не следует ставить больше 3-х баллов, в функции не используется `getDeskPlayers` или `getPlayerCards`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 34

5

### Комментарий к Критерий 34

## Критерий 35

Функция `hasNoCards` может иметь вид

```
val hasNoCards = null o getPlayerCards
```

- Не следует ставить больше 4-х баллов, если не используется операция композиции `o`.
- Стоит снизить оценку на один балл, если в решении присутствует дополнительное "оборачивание" в функцию, т.е. решение представлено в виде, подобном следующему:

```
fun hasNoCards p = (null o getPlayerCards) p
```

- Не следует ставить больше 3-х баллов, в функции не используется `getPlayerCards` или `null`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 35

4

### Комментарий к Критерий 35

дополнительное "оборачивание" в функцию

## Критерий 36

Функция `countLoss` может иметь вид

```
val countLoss =  
  let  
    fun playerLoss p =  
      let val cs = getPlayerCards p  
      in (getPlayerName p, sumCards cs)  
      end  
  in map playerLoss  
  end
```

- Не следует ставить больше 3-х баллов, в функции не используется `getPlayerName` или `sumCards`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 36

5

### Комментарий к Критерий 36

## Критерий 37

Функция `naiveStrategy` может иметь вид

```
fun naiveStrategy (move, cs, c as (col, r), is) =
  let
    val black = getSameColor (BLACK, cs)
    val maxCol = maxColor cs
    fun greatest (c1, c2) =
      if cardValue c1 > cardValue c2 then c1 else c2
    fun maxCard (c :: cs) = foldl greatest c cs
      | maxCard _ = raise Empty
  in
    case move
    of GIVE col' =>
      let val sameColor = getSameColor (col', cs)
      in
        if null sameColor
        then if hasRank (WILD_DRAW_FOUR, black)
          then ((BLACK, WILD_DRAW_FOUR), maxCol)
          else ((BLACK, WILD), maxCol)
        else if not (isSameColor (maxCol, col'))
          andalso hasRank (WILD, black)
          then ((BLACK, WILD), maxCol)
          else (maxCard sameColor, col)
        end
      | EXECUTE =>
        let val (firstSame :: _) = getSameRank (r, cs)
        in (firstSame, col)
        end
      | PROCEED =>
        let
          val sameColR = getSameColor (col, cs) @ getSameRank (r, cs)
        in
          if not (null sameColR)
          then (maxCard sameColR, col)
          else if hasRank (WILD_DRAW_FOUR, black)
            then ((BLACK, WILD_DRAW_FOUR), maxCol)
            else ((BLACK, WILD), maxCol)
          end
        end
    end
  end
```

- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 37

5

### Комментарий к Критерий 37

## Критерий 38

Функция `play` может иметь вид

```
fun play (dsk, playCards) =
  let
    val {players = p :: ps, pile = cs as (c :: _), deck = ds, state = m} = dsk
    val name = getPlayerName p
    val (playCard, wantCol) =
      (getPlayerStrategy p) (m, getPlayerCards p, c, countCards dsk)
    val _ = removeCard (playCards, playCard, IllegalMove name)
    val newPlayerCards =
      removeCard (getPlayerCards p, playCard, IllegalMove name )
    val newPlayers = setPlayerCards (p, newPlayerCards) :: ps
    val newPile = playCard :: cs
    val newDesk = makeDesk (newPlayers, newPile, ds, GIVE wantCol)
  in
    case playCard
    of (_, WILD_DRAW_FOUR) =>
      if hasColor (requiredColor dsk, playCards)
      then nextPlayer (takeFour newDesk)
      else nextPlayer (takeFour (nextPlayer newDesk))
    | (BLACK, _)    => nextPlayer newDesk
    | (_, NUM _)    => nextPlayer (setDeskState (newDesk, PROCEED))
    | (_, REVERSE) => nextPlayer (changeDirection
      (setDeskState (newDesk, PROCEED)))
    | _             => nextPlayer (setDeskState (newDesk, EXECUTE))
  end
```

- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 38

5

## Комментарий к Критерий 38

## Критерий 39

---

Функция `gameStep` может иметь вид

```

fun gameStep dsk =
  let
    val playCards = playableCards dsk
  in
    case (playCards, getDeskState dsk)
    of (_ :: _, _) => play (dsk, playCards)
      | ([], EXECUTE) => pass dsk
      | _ => let
          val dsk = takeOne dsk
          val playCards = playableCards dsk
        in
          if null playCards
          then pass dsk
          else play (dsk, playCards)
        end
      end
  end
end

```

- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 39

5

### Комментарий к Критерий 39

## Критерий 40

Функция `game` может иметь вид

```

fun game dsk =
  let
    val desk = start dsk
    fun gameIter dsk =
      let
        val pls = getDeskPlayers dsk
        val winner = List.find hasNoCards pls
      in
        case winner
        of SOME p => (getPlayerName p, countLoss pls)
          | NONE   => gameIter (gameStep dsk)
      end
    end
  in gameIter desk
  end
end

```

- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 40

**Комментарий к Критерий 40**

пробел

**НАВИГАЦИЯ**

В начало

■ Личный кабинет


Страницы сайта

Мои курсы

Графика Осень 2018

ЯП 2018

Участники

 Значки Компетенции Оценки


Общее

Форумы курса

Материалы по тематике курса

Лабораторные работы

Оценка лабораторных работ

 Лабораторная работа №0 Лабораторная работа №1 **Лабораторная работа №2**

■ Моя работа

 Лабораторная работа №3 Лабораторная работа №4 Лабораторная работа №5 Лабораторная работа №6

Раздел 1. Standard ML

Раздел 2. Haskell

Раздел 3. LISP

Раздел 4. Ruby

Раздел 5. PROLOG

Реферат

Аттестация

Бонусы

Напоминалки

ИКБ

On-line

