Правила оформления решений на языке Haskell

Содержание

1 Общие правила оформления

§ 1.1 Внимательно читайте задание. Посылаемое на проверку решение должно удовлетворять всем условиям, представленным в задании. В том числе это относится к имени основной функции, последовательности и типу ее аргументов и типу ее результата (если таковые указаны в задании).

§ 1.2 Каждая строка программы не должна превышать по длине 80 символов (включая отступы в начале строки).

§ 1.3 Код программы не должен содержать символов табуляции. Для отступов необходимо использовать пробелы. Текстовый редактор необходимо настроить, чтобы при нажатии клавиши табуляции вместо символа табуляции вставлялось 2 пробела.

§ 1.4 Программа должна компилироваться. Любая программа, посылаемая на проверку, должна проходить успешную компиляцию *без ошибок и предупреждений* (errors или warnings). Без исключений. После внесения изменений в код программы, даже самых незначительных, убедитесь в том, что программа проходит успешную компиляцию, прежде чем посылать ее на проверку.

§ 1.5 При написании программ разрешается использовать только двумерный синтаксис.

§ 1.6 Каждая функция должна быть короткой, лаконичной, делать только одну вещь. Каждая функция должна занимать не более 50 строк, делать только одну вещь, но делать ее хорошо.

§ 1.7 Каждая значимая часть кода должна быть снабжена соответствующим примером, демонстрирующим возможные варианты функционирования этой части. Все примеры должны быть описаны в конце файла с программой. Демонстрационные результаты должны быть выведены на экран с помощью функции `print` в теле определения значения `main`.

```haskell
-- ПРИМЕР ВЫВОДА ДЕМОНСТРАЦИОННОГО РЕЗУЛЬТАТА

main = do
  -- примеры работы функции удаления подряд идущих
  -- одинаковых элементов списка
  print $ delDuplicates [12]  -- в списке всего лишь один элемент
  print $ delDuplicates [1, 2, 3, 4, 5] -- в списке нет повторяющихся элементов
```

```
  -- в списке есть несколько серий повторяющихся элементов
  print $ delDuplicates [111, 111, 111, 2, 2, 2, 2, 3, 3, 3, 2, 2, 2, 1, 1, 1,
                         2, 3, 3, 43, 4, 5, 6, 6, 6, 6, 4, 4, 4, 4, 4, 4, 3, 3, 3, 2]
```

§ 1.8 Избегайте использования одновременного использования в одном выражении конструкций `let` и `where`.

§ 1.9 Избегайте повторения одного и того же кода. Если вам кажется, что код который вы пишите, повторяет уже существующий, то попытайтесь оптимизировать структуру программы для того, чтобы избежать повторения одних и тех же действий.

§ 1.10 Любой циклический процесс должен быть реализован в виде рекурсии. Если в задании не оговаривается, какую рекурсию организовывать, — хвостовую или нехвостовую, следует производить выбор ориентируясь на минимизацию количества операций для получения результата.

2 Комментарии

§ 2.1 В программе, код которой не является очевидным, каждая функция должна быть прокомментирована. Из комментариев должно быть понятно, что функция получает в качестве аргументов и что функция возвращает в качестве результата. В случае наличия сложного/громоздкого тела функции необходимо прокомментировать ход получения результата функции.

§ 2.2 Следует избегать комментариев, повторяющих выражение языка программирования.

```
-- ПРИМЕР ПЛОХОГО КОММЕНТАРИЯ

A = B + C  -- свяжем имя A со значением суммы значений B и C
```

§ 2.3 Правильно оформляйте многострочные комментарии. При просмотре в редакторе не имеющем подсветки синтаксиса или при черно-белой печати код программы сливается с многострочными комментариями. Чтобы выделить такие комментарии следует придерживаться одного из двух стилей их оформления:

1. Начинайте каждую новую строку комментария с символа `-`, например:

   ```
   {- This is one of those rare but long comments
    - that need to span multiple lines because
    - the code is unusually complex and requires
    - extra explanation. -}
   ```

   Такие комментарии легко читать, но не слишком легко редактировать.
2. Альтернатива

   ```
   {-----------------------------------------------
   This is one of those rare but long comments that
   need to span multiple lines because the code is
   unusually complex and requires extra explanation.
   -----------------------------------------------}
   ```

§ 2.4 Отдавайте предпочтение однострочным комментариям. Более предпочтительны короткие комментарии, начинающиеся с `--` и продолжающиеся до конца строки.

§ 2.5 Используйте линию комментариев для отделения друг от друга частей программы. Линия

```
-------------------------------------------------
```

является прекрасным заменителем пустой строки для отделения одной части программы от другой.

3 Отступы и пробелы

§ 3.1 Не используйте символ табуляции для отступов.

§ 3.2 Делайте отступы постоянной длины. Обычный отступ — 2 пробела. Можно делать отступы большего размера, но при этом вы можете превысить ограничение на длину строки и, чтобы этого избежать, придется разрывать выражения.

§ 3.3 Используйте пустые строки разумно. Пустые строки используются, в основном, на верхнем уровне, для отделения одного определения (группы определений) от другого.

Пустые строки могут повысить читабельность программы, но есть и негативный эффект: текстовый редактор отображает ограниченное количество строк. Если необходимо прочитать и понять за раз функцию, занимающую большое количество строк, лишние пустые строки могут помешать в этом.

Не должно быть пустых строк в выражении (включая выражение — тело функции).

§ 3.4 Используйте пробелы.

1. Ставьте пробел между функцией и ее аргументом.
2. Ставьте пробел между инфиксным оператором и его аргументом.
3. Ставьте пробел после запятой.

```
1 + fun (height l, height r)      -- ХОРОШО

1+fun(height l,height r)          -- 4 раза ПЛОХО
```

4 Имена и объявления

§ 4.1 Соглашение для идентификаторов:

Токен	*Соглашение*	*Пример*
Имя переменной, имя функции	Последовательность строчных символов, начинающаяся со строчной буквы. В именах, состоящих из нескольких слов, второе и последующие слова начинаются с заглавной буквы.	`getItem`
Имя типа, Имя класса, Имя конструктора, Имя модуля	Последовательность строчных букв, начинающаяся с заглавной буквы. В именах, состоящих из нескольких слов, каждое слово начинается с заглавной буквы.	`PriorityQueue`

§ 4.2 Допускаются следующие общепринятые обозначения (обычно, для аргументов функции): `f` для функций, `s` для строковой переменной, `n` для целочисленной переменной, `x` и `y` для переменной любого типа, `a` или `b` для переменной любого типа, `xs` или `ys` для списков, `xss` для списка списков, и т. д.

5 Короче — лучше

§ 5.1 Оценивайте стоимость использования промежуточных результатов.

§ 5.2 Не перегружайте базовые функции. Не пишите свои варианты базовых функций (в особенности `filter`, `map` и т. п.).

§ 5.3 Упрощайте условные выражения, возвращающие логическое значение.

§ 5.4 Упрощайте все. Если выражение возможно упростить, необходимо написать его проще.

§ 5.5 Не создавайте лишних λ-выражений.