ФАКУЛЬТЕТ КИБЕРНЕТИКИ И ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

КАФЕДРА КИБЕРНЕТИКИ

Утверждено на заседании кафедры  
  
протокол №\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

от «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2014 г.

**ФОНД ОЦЕНОЧНЫХ СРЕДСТВ**

по дисциплине

ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

|  |  |
| --- | --- |
| Направление подготовки (специальность) | 09.03.04 Программная инженерия |
|  |  |
| Профиль подготовки (при его наличии) | Математическое и программное обеспечение вычислительных машин и компьютерных сетей |
|  |  |
| Наименование образовательной программы (специализация) | Математическое и программное обеспечение вычислительных машин и компьютерных сетей |
|  |  |
| Квалификация (степень) выпускника | бакалавр |
|  |  |
| Форма обучения | очная |

**ПАСПОРТ**

**фонда оценочных средств**

**по дисциплине «Функциональное программирование»**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Семестр** | **Интерактив** | **Трудоемкость, кред.** | **Общий объем курса, час.** | **Лекции, час.** | **Практич. занятия, час.** | **Лаборат. работы, час.** | **СРС, час.** | **КСР, час.** | **Форма(ы) контроля, экз./зач./КР/КП** |
| 7 |  | 2 | 72 | 16 | 16 | 0 | 40 | 0 | З |
| ИТОГО | 8 | 2 | 72 | 16 | 16 | 0 | 40 | 0 |  |

Группа: К07-221, К07-222, К07-223

**1.1. Область применения**

Фонд оценочных средств (ФОС) *–* является неотъемлемой частью учебно-методического комплекса учебной дисциплины «Введение в интеллектуальные системы и технологии» и предназначен для контроля и оценки образовательных достижений обучающихся, освоивших программу данной дисциплины.

**1.2. Цели и задачи фонда оценочных средств**

Целью Фонда оценочных средств является установление соответствия уровня подготовки обучающихся требованиям ОС НИЯУ МИФИ.

Для достижения поставленной цели Фондом оценочных средств по дисциплине «Введение в интеллектуальные системы и технологии» решаются следующие задачи:

– контроль и управление процессом приобретения обучающимися знаний, умений и навыков предусмотренных в рамках данного курса;

– контроль и оценка степени освоения общекультурных, общепрофессиональных и профессиональных компетенций предусмотренных в рамках данного курса;

– обеспечение соответствия результатов обучения задачам будущей профессиональной деятельности через совершенствование традиционных и внедрение инновационных методов обучения в образовательный процесс в рамках данного курса.

**1.3. Модели контролируемых компетенций**

В результате освоения дисциплины у выпускника формируются следующие компетенции:

ОПК-1, ОПК-2, ОПК-3, ПК-19

4. Структура и содержание учебной дисциплины

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№ п.п** | **Наименование раздела учебной дисциплины** | **Недели** | **Лекции, час.** | **Практ. занятия/ семинары, час.** | **Лабораторные работы, час.** | **Обязат. текущий контроль (форма\*, неделя)** | **Аттестация раздела (форма\*, неделя)** | **Максимальный балл за раздел\*\*** | **Компетенции по разделам, проверяемые при текущем и рубежном контроле** | **Компетенции, проверяемые на зач. /экз.** |
|  | *7 семестр* |  |  |  |  |  |  |  |  |  |
| 1 | Раздел 1 | 1-8 | 8 | 8 |  |  | КИ, 8 | 26 | ОПК-1, ОПК-2, ОПК-3, ПК-19 | ОПК-1, ОПК-2, ОПК-3, ПК-19 |
| 2 | Раздел 2 | 9-16 | 8 | 10 |  |  | КИ, 16 | 26 | ОПК-1, ОПК-2, ОПК-3, ПК-19 | ОПК-1, ОПК-2, ОПК-3, ПК-19 |
|  | *Итого за 7 семестр* |  | 16 | 16 | 0 |  |  | 52 |  |  |
|  | **Контрольные мероприятия после 7 семестра** |  |  |  |  |  | З | 48 | ОПК-1, ОПК-2, ОПК-3, ПК-19 | ОПК-1, ОПК-2, ОПК-3, ПК-19 |

\* – сокращенное наименование формы контроля

\*\* – сумма максимальных баллов должна быть равна 100 за семестр, включая зачет и (или) экзамен

Сокращение наименований форм текущего контроля и аттестации разделов:

КИ Контроль по итогам

КАЛЕНДАРНЫЙ ПЛАН

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Недели** | **Темы занятий / Содержание** | **Лек., час.** | **Пр./сем., час.** | **Лаб., час.** |
|  | *7 семестр* | 16 | 16 | 0 |
| 1 | **Тема 1. Исходные определения.** Введение. Структуры данных. Типы. Аппликация. Абстракция. Выражение в форме оператор/операнд. |  |  |  |
| 2 - 3 | **Тема 2. Абстрактный язык функционального программирования.** Нотация (клозы, образцы, охрана). Операционная семантика. Функции высших порядков. |  |  |  |
| 4 - 5 | **Тема 3. Приемы программирования.** Разбор случаев. Введение подфункций. Накапливающий параметр. Хвостовая рекурсия. Итерационная форма программ. |  |  |  |
| 6 - 7 | **Тема 4. Введение в Лисп.** Атомы и списки. Точечное представление. Базовые функции обработки данных и примитивы управления. Связывание, деструктивное присваивание. Имя и значение символа. Определение функций. Применяющие функционалы. Замыкания. Макросы. Ввод и вывод. |  |  |  |
| 8 | **Тема 5. Синтаксически-ориентированное программирование .** Язык абстрактного описания областей. Построение абстрактного описания (синтаксиса) списков, списочных структур, деревьев, бинарных деревьев. Построение типовых алгоритмов обработки данных для этих областей. |  |  |  |
| 9 - 10 | **Тема 6. Доказательства свойств программ.** Определение индукции по структурам данных. Списочная индукция, sexpr-индукция. Примеры. Ограничения. |  |  |  |
| 11 - 12 | **Тема 7. Тематические основания.** Формальная система. Аксиоматезируемость. Синтаксическое и семантическое равенство. Элементы лямбда-исчисления (выражение, подстановка, аксиомы, правила вывода, стратегии редукции, нормальная форма, Теорема Черча-Россера, пародоксальный комбинатор). |  |  |  |
| 13 - 14 | **Тема 8. Трансформации программ.** Эквивалентные и корректные преобразования. Система преобразований Берсталла и Дарлингтона: instantiation, folding, unfolding, low, abstraction. Трансформационный синтез программ и его проблемы. |  |  |  |
| 15 - 16 | **Тема 9. Частичные вычисления.** Остаточная программа. Интерпретатор. Компилятор. Генератор компиляторов. Проекции Футамуры. Принципы построения частичных вычислителей. |  |  |  |

ТЕМЫ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

|  |  |
| --- | --- |
| **Недели** | **Темы занятий / Содержание** |
|  | *7 семестр* |
| 1 - 2 | **Элементы программирования на абстрактном функциональном языке.** Элементы программирования на абстрактном функциональном языке. |
| 3 - 4 | **Программирование функций для работы со списками на Haskell.** Программирование функций для работы со списками на Haskell. |
| 5 - 6 | **Программирование функций вычисления сумм рядов** Программирование функций вычисления сумм рядов |
| 7 - 8 | **Программирование аналитических преобразований.** Программирование аналитических преобразований. |
| 9 - 10 | **Программирование аналитических преобразований.** Программирование аналитических преобразований. |
| 11 - 12 | **Программирование синтаксических анализаторов** Программирование синтаксических анализаторов |
| 13 - 14 | **Доказательство свойств функций.** Доказательство свойств функций. |
| 15 - 16 | **Реализация семестровых проектов по индивидуальным заданиям.** Реализация семестровых проектов по индивидуальным заданиям. |

**Контрольно-измерительные материалы по курсу «Функциональное программирование»**

**А**

Раздел А содержит вопросы с четырьмя вариантами ответов, один из которых является правильным.

1. Какими из следующих свойств могут обладать языки функционального программирования:
2. Краткость и простота
3. Строгая типизация
4. Модульность
5. Функции — это значения
6. Отсутствие побочных эффектов
7. Отложенные вычисления
8. Императивность
9. ADEFG
10. Все
11. ABCDEF
12. ACDEFG
13. В чистом функциональном программировании …
14. Отсутствует оператор присваивания
15. Допускается присваивание только локальным переменным
16. Всегда присутствует оператор присваивания
17. Не запрещается оператор присваивания
18. В чистом функциональном программировании …
19. Объекты можно изменять или уничтожать
20. Объекты нельзя изменять и уничтожать
21. Объекты можно изменять, но нельзя уничтожать
22. Объекты нельзя изменять, можно только уничтожать
23. В чистом функциональном программировании …
24. Отсутствует сборщик мусора
25. Сборщик мусора может работать параллельно с механизмом явного возврата памяти
26. Есть сбор­щик мусора
27. Автоматический сборщик мусора не нужен
28. В чистом функциональном программировании …
29. Функции могут давать побочный эффект
30. Побочные эффекты от выполнения функций допустимы
31. Только функции стандартной библиотеки свободны от побочных эффектов
32. Все функции свободны от побочных эффектов
33. Если функциональный язык не поддерживает отложенные вычисления, то он называет­ся
34. строгим
35. нестрогим
36. ленивым
37. императивным
38. Какая операция не является базисной для списков?
39. prefix
40. infix
41. head
42. tail
43. Какая операция является селектором для списков и S-выражений?
44. append
45. prefix
46. length
47. tail
48. Какая операция является конструктором S-выражений?
49. append
50. ++
51. prefix
52. []
53. Что не является S-выражением?
54. a : b
55. a : []
56. a b
57. [a, b]
58. Какое S-выражение не является списком:
59. []
60. a : []
61. a : b
62. a : (b : [])
63. Каким образом можно выбрать второй элемент списка L при условии, что он есть?
64. head (tail L)
65. tail (head L)
66. head (tail (head L))
67. tail L
68. Каким образом можно выбрать все элементы списка L, начиная с третьего?
69. head (head L)
70. tail (tail L)
71. tail (tail (head L))
72. head (head (tail L))
73. Выберите выражение, которое в абстрактном функциональном языке может быть использовано в качестве образца:
74. (X + 1 = 5)
75. (Y + X = 5)
76. (X + 1 = Y)
77. (Y + X = Z)
78. Какое из определений функции function недопустимо в Хаскелл?
79. function x [] = True
80. function x (x:xs) = False
81. function x (y:xs) = False
82. function x (n+1) = n
83. Образец в функциональном программировании это …
84. Выражение, построенное с помощью операций конструирования дан­ных
85. Ограничение, связывающее допустимые значения переменных
86. Механизм, позволяющий компилятору опре­делять типы констант, выражений и функций из контекста
87. Шаблон функции для построения итераторов над различными типами данных
88. Образец в функциональном программировании это …
89. Ограничение, связывающее допустимые значения переменных
90. Пример вызова функции
91. Механизм, позволяющий компилятору опре­делять типы констант, выражений и функций из контекста
92. Выражение, используемое для сопоставления с данными
93. Укажите верное высказывание о понятии «образец» в функциональном программировании
94. Сопоставление с данными возможно единственным образом
95. Все варианты сопоставления с данными перебираются встроенным механизмом перебора и отката
96. Образец может включать в себя переменные, константы и функции
97. Использование образцов для аргументов функции не влияет на порядок следования её клозов.
98. Какое высказывание верно при использовании образцов аргументов для определения функции в нескольких клозах:
99. Общность образца для аргумента не влияет на порядок следования клозов функции
100. Сначала следует записывать клозы с более общими образцами, снижая общность в последующих клозах функции
101. Сначала следует записывать клозы с наиболее конкретными образцами, повышая общность образцов в последующих клозах функции
102. Общность используемых образцов должна совпадать для всех клозов функции
103. Как осуществляется поиск и выбор клоза при вызове функции?
104. Просмотр клозов от первого к последнему до подходящего клоза
105. Просмотр клозов от первого к последнему до подходящего клоза, но если происходит ошибка, то выбирается следующий подходящий.
106. Просмотр клозов от последнего к первому до подходящего клоза
107. Просмотр клозов от последнего к первому до подходящего клоза, но если происходит ошибка, то выбирается следующий подходящий.
108. Сколько клозов выполняется при вызове функции?
109. Один
110. Один, если он подходит
111. Все, если они подходят
112. Все
113. Что происходит, когда ни один из клозов функции не подходит?
114. Считается, что функция возвращает константу False
115. Возникает ошибка времени выполнения
116. Осуществляется просмотр клозов функций-предков
117. Осуществляется поиск одноименных функций в других модулях
118. Какие конструкции могут быть фактическими аргументами функций?
119. образцы
120. функции
121. клозы
122. типы
123. Какие конструкции не могут быть фактическими аргументами функций?
124. переменные
125. типы
126. константы
127. образцы
128. Что в языке Хаскел можно обозначать знаком ::
129. типы функций
130. типы монад
131. типы клозов
132. целочисленное деление
133. Откуда интерпретатор Haskell не получает информацию о типах выражений?
134. С помощью механизма автоматического выведения типов
135. Из явного указания типов функций при их определении
136. Из явного указания типов констант при их использовании
137. Из явного указания типов выражений при их использовании
138. Общий вид рекурсивных определений, позволяющих при трансляции обеспечить вы­чис­ления в постоянном объёме памяти через итерацию, называется:
139. функциями с накопителем (аккумулятором)
140. равенствами в итеративной форме
141. лямбда-выражениями
142. равенствами в рекурсивной форме
143. Укажите лишнее правило построения определений с накапливающим параметром:
144. Равенства, соответствующие рекурсивному определению, выглядят как обращения к новой функции, в котором аккумулятор получает то значение, которое возвращается исходной функцией
145. Вводится новая функция с дополнительным аргументом (аккумулятором), в котором на­капливаются результаты вычислений. Начальное значение аккумулирующего аргумента задается в равенстве, связывающем старую и новую функции.
146. Те равенства исходной функции, которые соответствуют выходу из рекурсии, заменяются возвращением аккумулятора.
147. Те равенства исходной функции, которые соответствуют выходу из рекурсии, заменяются вызовом новой функции, в котором аккумулятор получает то значение, которое возвращается исходной функцией
148. Общий вид рекурсивных определений, позволяющих при трансляции обеспечить вычисления в постоянном объёме памяти через итерацию, называется равенствами в итеративной форме, которые представимы в виде fi (pij) = eij. Укажите вид выражения не в итеративной форме:
149. eij – выражение, не содержащее рекурсивные вызовы, а например, операции над данными.
150. eij – условное выражение, содержащее рекурсивные вызовы только в условии
151. eij имеет вид fk(vk), при этом vk — последовательность простых выражений.
152. eij – условное выражение с простым выражением в условии
153. Список из элементов a, b, c на языке Хаскел можно записать как:
154. (a b c)
155. [a, b, c]
156. [a b c]
157. (a:(b:c))
158. Список из элементов a, b на языке Хаскел можно записать как:
159. (a b)
160. (a, b)
161. [a b]
162. (a:(b:[]))
163. Полиморфная функция
164. изменяет типы своих переменных.
165. комбинирует данные различных типов.
166. может применяться к аргументам различных типов.
167. постепенно изменяет свой вид в процессе вычисления.
168. Функция на языке Хаскелл, имеющая тип [a] -> [[a]]
169. преобразует символ в строку.
170. выделяет подстроку из строки.
171. возвращает в качестве аргумента строку, изменяя порядок символов в ней.
172. оперирует строкой и возвращает последовательность подстрок.
173. Функция на языке Хаскелл, имеющая тип Eq a => a -> Bool,
174. требует аргумент с именем a
175. преобразует True в аргументы типа a
176. является полиморфной функцией
177. должна быть равна а
178. Что в абстрактном функциональном языке обозначает символ #:
179. знак альтерации
180. тип
181. начало идентификатора
182. операция сравнения «не равно»
183. В Haskell с заглавной буквы должны начинаться имена:
184. функций
185. переменных
186. констант
187. типов
188. В Haskell отсутствует возможность декларативного формирования:
189. бесконечных списков
190. структур данных
191. s-выражений
192. циклов
193. Вызов функции add с параметрами x и y в Haskell’е может выглядеть как:
194. Add x y
195. [add, x, y]
196. add (x, y)
197. add (x y)
198. Выберите неверное для Haskell утверждение
199. определения классов и их методов разделены
200. типы объектов не могут быть выведены неявно
201. реализован контроль доступа к методам классов
202. предоставляет механизм модуляризации программ
203. Что не является монадическим классом для монад в Haskell:
204. Functor
205. Maybe
206. Monad
207. MonadPlus
208. Математически монада определяется через набор правил, которые в первую очередь связывают…
209. операции, производимые над монадой
210. функции, уместные для монады
211. переменные, введённые в монаде
212. типы констант, введённые для монады
213. Какое утверждение неверно для понятия «монадический класс»?
214. не мо­жет являться предком для другого монадического класса
215. монадические классы наследуемы
216. монады являются типами, которые представляют собой экземпляры монадических классов
217. монадические классы предопределены
218. Монады в Haskell определяются как:
219. классы особого вида для реализации ввода-вывода и работы со списками
220. типы, которые представляют собой экземпляры одного из монадических классов
221. модули особого вида для реализации ввода-вывода и работы со списками
222. наборы инкапсулированных функций
223. Монада, по сути, представляет собой:
224. структуру, аналогичную понятию record или struct в императивном программировании
225. структуру, аналогичную понятию class в императивном программировании
226. набор инкапсулированных функций
227. набор правил, которые связывают операции, производимые над монадой
228. Для работы с файлами в Haskell необходимо явно:
229. импортировать модуль Prelude
230. импортировать модуль IO
231. экспортировать модуль Prelude
232. экспортировать модуль IO
233. Дайте наиболее точное определение понятия хвостовая рекурсия:
234. специальный вид рекурсии, в которой имеется единственный вызов рекурсивной функции.
235. вид рекурсии c единственным вызовом рекурсивной функции, который выполняется после остальных вычислений.
236. конструкция, при которой функция вызывает другую функцию, которая в свою очередь вызывает первую.
237. способ организации обработки данных, при котором функция вызывает сама себя непосредственно, либо с помощью других функций.
238. Какого соглашения по именованию нет в Haskell?
239. использование заглавной буквы в начале идентификатора
240. имена всех типов должны начинаться со строчной буквы
241. имена функций, переменных, констант должны начинаться со строчной буквы
242. возможно использование некоторых специальных знаков в качестве первого символа идентификатора
243. Список нечётных натуральных чисел до N = 20 в нотации Haskell можно записать как:
244. [1 .. 19]
245. [1 3 .. 20]
246. [1 3 .. 19]
247. [1, 3 .. 20]
248. В абстрактном функциональном языке клозы имеют вид: def f p1, ..., pn = expr, где:
249. pi – выражение
250. pi – переменная
251. pi – константа
252. pi – образец
253. Любую ли функцию можно преобразовать для вычисления с аккумуля­тором?
254. Да
255. Нет
256. Только функции, реализуемые с помощью хвостовой рекурсии
257. Только функции, осуществляющие арифметические действия
258. Прочтите выражение [ x | x <- xs, x > m, x < n ] :
259. переменная x, взятая из xs,при этом x больше m и x меньше n
260. cписок таких элементов из xs, что каждый из них больше m и меньше n
261. константа x, найденная в xs, большая m и меньшая n
262. cписок таких констант из xs, что каждая из них больше m и меньше n
263. При использовании локальных переменных в абстрактном функциональном языке используется тип записи:

let y = a \* b

f x = (x + y) / y

in f c + f d

f x y | y > z = y – z

| y == z = 0

| y < z = z – y

where z = x \* x

1. оба допустимы
2. оба недопустимы
3. Выберите неверное высказывания для понятия модуль в Haskell:
4. при помощи модулей можно контролировать пространства имён
5. при помощи модулей можно создавать абстрактные типы данных
6. имя модуля должно начинаться с заглавной буквы английского алфавита
7. в модуле допускается размещать только типы данных и функции
8. Какое правило не должно соблюдаться при именовании модулей в языке Haskell:
9. Имя файла, содержащего модуль, может не совпадать с названием модуля
10. Имя модуля может начинаться с прописной буквы
11. Имя файла с модулем может начинаться с прописной буквы
12. Имя файла с модулем может начинаться с заглавной буквы
13. В языке Haskell для понятия маски подстановки \_ (символ подчеркивания) следующее высказывание неверно:
14. маска подстановки заменяет любой образец
15. все переменные, замещенные маской, считаются равными (эквивалентными)
16. маска подстановки является своего рода анонимной переменной
17. если в теле клоза нет необходимости использования переменной образца, то её можно заменить маской подстановки
18. Каждое действие ввода/вывода возвращает:
19. Значение функции
20. Значение переменной
21. Значение константы
22. Может возвращать любое из перечисленных значений
23. В примере ( getChar :: IO Char ) показано, что:
24. функция getChar воз­вращает монадическое значение типа Char
25. тип функции getChar должен быть типом монады IO
26. точка входа в виде функции getChar должна быть определена в модуле с именем IO
27. функция getChar воз­вращает значение типа Char
28. Выберите неверное продолжение фразы «Действия ввода/вывода являются обычными значениями в терминах Haskell, т.е. …»
29. дей­с­твия можно передавать в функции в качестве параметров и возвращать как значения
30. действия можно заключать в структуры данных
31. действия можно использовать там, где можно использовать данные языка
32. действия можно инкапсулировать в объекты
33. Пусть некоторая функция f - это функция од­ной переменной типа A, принимающая значение типа B. Выберите верную запись на языке Хаскелл:
34. f :: A => B
35. f :: A -> B
36. f :: A x B
37. f :: B
38. Какое утверждение неверно для Хаскелл?
39. если функция не ис­поль­зует монадический тип IO, то она не может заниматься вводом/выводом
40. ес­ли функция возвращает монадический тип IO, то она должна подчиняться па­радигме дей­ствий
41. для того, чтобы возвращать монадические значения, существует спе­циальная функция return, которая из простого типа данных делает монадический
42. сис­тему операций ввода/вывода нельзя назвать полностью функциональной
43. Охарактеризуйте функцию сложения двух аргументов add (x, y)
44. Имеет тип (A->A)->A
45. Имеет тип A×A×A
46. Имеет тип А×А->A
47. Имеет тип A->(A->A)
48. Укажите неверную характеристику выражения (plus 5)
49. функция plus может иметь тип A->(A->A)
50. функция plus может иметь тип A->A
51. это вызов функции в кар­ри­ро­ванной форме
52. функция plus не может иметь других параметров
53. Охарактеризуйте функцию map типа (a->b)->[a]->[b] из стандартной библиотеки Хаскел
54. преобразует переданную функцию, действующую над атомарными типами, в другую функцию действующую над списковыми типами, и возвращает ее в качестве результата.
55. применяет переданную функцию ко всем элементам списка, переданного в качестве второго параметра, и возвращает список результатов.
56. применяет переданную функцию ко всем элементам списка, переданного в качестве второго параметра, а затем ко всем элементам списка, переданного в качестве третьего параметра.
57. применяет переданную функцию к списку, переданному в качестве второго параметра, и возвращает результирующий список.
58. Каков тип функции композиции (.) ?
59. a->b->c
60. b->a->c
61. (b->c)->(a->b)->(a->c)
62. (a->b)->(b->c)->(a->c)
63. Укажите неверную характеристику выражения \x -> x \* x \* x?
64. Определена анонимная функция
65. Описан функциональный тип
66. Определена функция одного аргумента
67. Функция возвращает куб от аргумента
68. Что можно сказать о списках [1, 2, 3] и [‘a’, ‘b’, ‘c’]?
69. списки записаны в каррированной форме
70. состоят из элементов полиморфного типа (целые числа можно трактовать как вещественные, а символы можно трактовать как строки из одного символа)
71. типы списков одинаковы, так как описываются одним шаблоном типа [a]
72. типы элементов не являются полиморфными
73. Что можно сказать о списках [1, 2, 3] и [1, ‘a’]?
74. списки записаны в каррированной форме
75. списки нельзя описать одним шаблоном типа
76. списки можно описать одним шаблоном типа
77. списки одного типа, так как содержат только атомы и не содержат вложенных списков
78. Что в следующем примере нельзя сказать об а?

class Eq a where

(==) :: a -> a -> Bool

1. а – шаблон типа
2. а – типовая переменная
3. a – конкретный экземпляр класса Eq
4. Для всех типов, являющихся экземплярами Eq, должна быть определена операция ==, указанного типа.
5. Что можно сказать о функции, определенной в следующих клозах?

GetN (N, []) = \_

GetN (1, H:T) = H

GetN (N, H:T) = GetN (N – 1, T)

1. возвращает голову списка, если число элементов списка больше N, и \_ в противном случае
2. осуществляет поиск элемента с номером N
3. вычитает единицу из всех элементов списка длиной N
4. осуществляет поиск числа N в списке
5. N-ый элемент какого из рядов вычисляет функция

Summ\_P (1) = 1

Summ\_P (N) = Summ\_T (N) + Summ\_P (N – 1)

* 1. an = xn
  2. an = Σi = 1,n i
  3. an = Σj = 1,n (Σi = 1,j i)
  4. an = Σi = 1,p n-i

1. N-ый элемент какого из рядов вычисляет функция

Power (X, 0) = 1

Power (X, N) = X \* Power (X, N – 1)

1. an = xn
2. an = Пi = 1,n i
3. an = nx
4. an = Пi = 1,p n-i
5. Что делает функция Set?

Set ([]) = []

Set (H:T) = Include (H, Set (T))

Include (A, []) = [A]

Include (A, A:L) = A:L

Include (A, B:L) = prefix (B, Include (A, L))

1. получает на вход списочную структуру и обращает все её элементы, а также её саму.
2. устанавливает номер первого вхождения заданного атома в спи­сок.
3. возвращает список, содержащий единичные вхождения атомов за­данного списка.
4. возвращает список пар (символ, частота). Каждая пара определяет атом из заданного списка и частоту его вхождения в этот список.
5. Если на языке Хаскелл оп­ре­де­лена каррированная функция, складывающая два числа, то:
6. не допускается передавать её в качестве аргумента в функцию map
7. при вызове функции параметры нужно перечислять через запятую в скобках
8. при вызове функции нужно передать ей два параметра
9. при вызове функции параметры записываются через пробел
10. Выберите неверное утверждение для функции двух аргументов в Хаскелл:
11. её нельзя вызывать в инфиксной форме пока это не будет разрешено явно при её определении.
12. если это операция, то для её вызова достаточно между параметрами просто записать её имя.
13. если её имя начинается с буквы, то для её вызова в инфиксной форме нужно между параметрами записать имя функции, заключённое в символы обратного апо­с­т­ро­фа `.
14. её можно вызывать в префиксной форме.
15. В языке Haskell есть зарезервированное слово infixr, которое:
16. используется для конструирования упорядоченной пары
17. является именем стандартной функции, преобразующей выражение из префиксной в инфиксную форму
18. используется для назначения приоритета выполнения определяемой операции
19. указывает на допустимость инфиксной формы записи определяемой операции
20. Укажите число охраняющих конструкций в функции:

sign x | x > 0 = 1

| x == 0 = 0

| x < 0 = -1

1. 3
2. 6
3. 1
4. 0
5. Можно ли считать, что выражение if Exp1 then Exp2 else Exp3 в Haskell является сокращением выражения:

case (Exp1) of

(True) -> Exp2

(False) -> Exp3

1. только если выражение Exp1 имеет тип Bool
2. только если типы выражений Exp2 и Exp3 совпадают
3. если выражение Exp1 имеет тип Bool, а типы выражений Exp2 и Exp3 совпадают
4. только если типы выражений Exp1, Exp2 и Exp3 совпадают

**B**

Раздел B содержит вопросы с открытым ответом, который является числом, короткой последовательностью символов. Все примеры даны на языке Хаскелл.

1. Сконструировать конечный список из N = 200 натуральных чисел, используя конструктор списка.

Возможные ответы:

[1..200]

или

[1,2..200]

1. Сконструировать конечный список из N = 200 нечётных натуральных чисел, используя конструктор списка.

Возможные ответы:

[1, 3..200]

или

[1, 3..199]

1. Сконструировать конечный список из N = 200 чётных натуральных чисел, используя конструктор списка.

Возможные ответы:

[2, 4..200]

или

[2, 4..201]

1. Сконструировать конечный список из N = 300 квадратов натуральных чисел, используя функцию map и лямбда-абстракцию.

Ответ:

map(\x->x\*x, [1,2..300])

1. Сконструировать конечный список из N = 300 квадратов натуральных чисел, используя функцию map и операцию ^.

Ответ:

map(^2, [1,2..300])

1. Требуется сконструировать конечный список из N = 200 кубов натуральных чисел. Какое лямбда-выражение пропущено?

map(…, [1,2..200])

Возможный ответ:

\x -> x\*x\*x

1. Требуется сконструировать конечный список из N = 500 факториалов. Какое выражение пропущено?

factorial (0) = 1

factorial (N) = N \* factorial (N – 1)

map(…, [1..500])

Ответ:

factorial

1. В каком порядке будет отсортирован список степеней двойки, полученный с помощью вызова следующей функции:

powerTwo 0 = []

powerTwo n = (2 ^ n) : powerTwo (n – 1)

Ответ:

По убыванию

1. Сконструировать бесконечный список факториалов.

Возможный ответ:

numbersFrom n = n : numbersFrom (n + 1)

map factorial (numbersFrom 1)

1. Сконструировать бесконечный список натуральных чисел c помощью конструктора упорядоченной пары.

Возможный ответ:

numbersFrom n = n : numberFrom (n + 1)

1. Сконструировать бесконечный список кубов натуральных чисел c помощью конструктора списка.

Возможный ответ:

[n^3 | n <- [1..]]

1. Сконструировать бесконечный список кубов натуральных чисел. Используемые функции вызывать в каррированной форме.

Ответ:

map (\x -> x \* x \* x) [1..]

1. Сконструировать бесконечный список степеней пятёрки.

Ответ:

map (5^) [0..]

1. Сконструировать бесконечный список дробей 1/n c помощью конструктора упорядоченной пары.

Возможный ответ:

fractionsFrom n = (1/n) : fractionsFrom (n + 1)

1. Сконструировать бесконечный список дробей 1/n с помощью конструктора списка.

Возможный ответ:

[1/n | n <- [1..]]

1. Объявите тип идентификатора Branch в определении бинарного дерева на Хаскелл:

data Tree a = Leaf a | Branch (Tree a) (Tree a)

Ответ:

Branch :: Tree a -> Tree a -> Tree a

1. Объявите тип идентификатора Leaf в определении бинарного дерева на Хаскелл:

data Tree a = Leaf a | Branch (Tree a) (Tree a)

Ответ:

Leaf :: a -> Tree a

1. Определите функцию add сложения двух аргументов через лямбда-абстракцию:

Возможный ответ:

add = \x y -> x + y

1. Определите функцию inc увеличения аргумента на 1 через лямбда-абстракцию:

Возможный ответ:

inc = \x -> x + 1

1. Допишите клоз функции reverse, обращающей список.

reverse :: [a] -> [a]

reverse [] = []

reverse (h:t) = …

Ответ:

reverse t ++ [h]

1. Допишите клоз функции map, применяющую другую переданную в качестве параметра функцию ко всем элементам заданного списка.

map :: (a -> b) -> [a] -> [b]

map f [] = []

map f (x:xs) = …

Ответ:

(f x) : (map f xs)

1. Допишите клоз функции, вычисляющей N-ый элемент ряда F(n) = en = Σi = 0,∞ (ni / i!). Можно использовать вспомогательные функции Power (N, P) и Factorial (X).

Exponent (N, 0) = 1

Exponent (N, P) = …

Ответ:

(Power (N, P) / Factorial (P)) + Exponent (N, P – 1)

**C**

Раздел С содержит сложные задачи, для которых требуется развернутый ответ, оцениваемый по наличию некоторых заранее определенных признаков. Возможные оценки: 0, 1, 2 балла.

1. Реализуйте функцию, вычисляющую N-ый член ряда Фибоначчи, в виде хвостовой рекурсии.

Возможный ответ:

fibb 1 prev pprev = prev

fibb n prev pprev = fibb (n-1) (prev+pprev) prev

fib n = fibb n 1 0

1 балл:

Построена функция с одним накопителем вместо двух. Введена функция с двумя накопителями, но работает некорректно.

2 балла:

Построена функция с двумя накопителями.

1. Реализуйте функцию Set(L), возвращающую список, содержащий единичные вхождения атомов за­данного списка. Тип функции объявить явно.

Возможный ответ на абстрактном функциональном языке со вспомогательной функцией:

Set ([]) = []

Set (H:T) = Include (H, Set (T))

Include (A, []) = [A]

Include (A, A:L) = A:L

Include (A, B:L) = B:(Include (A, L))

Возможный ответ на Хаскелл со вспомогательной функцией:

set :: [a] -> [a]

set [] = []

set (h:t) = include h (set t)

include :: a -> [a] -> [a]

include a [] = [a]

include a (b:t) | a == b = a : t

|otherwise = b : (include a t)

Возможный ответ на Хаскелл с помощью конструктора списков:

set :: [a] -> [a]

set [] = []

set (x:xs) = x : set [y | y<-xs, y /= x]

1 балл:

Клозы синтаксически корректны, верно описаны типы функций, но функция работает некорректно. Ошибочно использованы образцы на Хаскелл, например, недопустим клоз include a (a:t) = a:t . Ошибочно записан конструктор списка.

2 балла:

Построена функция set и вспомогательная функция расширения множества новым элементом include, которая не вставляет элемент в список, если он уже есть. Построена функция set с помощью конструктора списков.

1. Реализуйте функцию Frequency (L), возвращающую список пар (символ, частота). Каждая пара определяет атом из заданного списка и частоту его вхождения в этот список. Тип функции объявить явно.

Возможный ответ на абстрактном функциональном языке со вспомогательной функцией образования результирующих пар:

Frequency L = F ([], L)

F (L, []) = L

F (L, H:T) = F (Corrector (H, L), T)

Corrector (A, []) = [A:1]

Corrector (A, (A:N):T) = (A:N + 1): T

Corrector (A, H:T) = H : Corrector (A, T)

Возможный ответ на Хаскелл с помощью конструктора списков и вспомогательной функцией подсчёта:

counter :: Eq a => a -> [a] -> Int

counter e [] = 0

counter e (h:t) | h == e = 1 + (counter e t)

| otherwise = 0 + (counter e t)

frequency :: Eq a => [a] -> [(a, Int)]

frequency [] = []

frequency (h:t) = (h, (counter h (h:t))) : frequency [s | s <- t, s /= h]

1 балл:

Клозы синтаксически корректны, верно описаны типы функций, список пар некорректен. Ошибочно использованы образцы на Хаскелл, например, недопустим клоз corrector (a, (a:n):t) = (a:n + 1): t. Ошибочно записан конструктор списка.

2 балла:

Построены вспомогательные функции. Идеи верны. Список пар корректен.

1. Реализуйте функцию ListSumm(L1, L2) сложения элементов двух списков. Возвращает список, составленный из сумм элементов списков-параметров. Учесть, что переданные списки могут быть разной длины.

Возможный ответ на абстрактном функциональном языке:

ListSumm ([], L) = L

ListSumm (L, []) = L

ListSumm (H1:T1, H2:T2) = prefix ((H1 + H2), ListSumm (T1, T2))

Возможный ответ на Хаскелл:

listSumm :: Ord (a) => [a] -> [a] -> [a]

listSumm [] l = l

listSumm l [] = l

listSumm (h1:t1) (h2:t2) = (h1 + h2) : (listSumm t1 t2)

1 балл:

Не учтён случай списков разной длины, но для списков одной длины всё верно.

2 балла:

Функция корректна для всех случаев.

1. Реализуйте функцию OddEven(L) перестановки местами соседних чётных и нечётных элементов в заданном списке.

Возможный ответ на абстрактном функциональном языке:

OddEven ([]) = []

OddEven ([X]) = [X]

OddEven (H1:[H2:T]) = prefix (prefix (H2, H1), OddEven (T))

Возможный ответ на Хаскелл:

oddEven :: [a] -> [a]

oddEven [] = []

oddEven [x] = [x]

oddEven (h1:(h2:t)) = (h2:h1) : (oddEven t)

1 балл:

Не учтён случай нечётного количества элементов (отсутствует клоз для oddEven [x]). Конструируется S-выражение, а не список.

2 балла:

Функция корректна для всех случаев.

1. Построить функцию, вычисляющую N-ый элемент ряда F(n) = Σi = 1,p n-i

Возможный ответ на абстрактном функциональном языке:

Summ\_Power (N, 0) = 1

Summ\_Power (N, P) = (1 / Power (N, P)) + Summ\_Power (N, P – 1)

Возможный ответ на Хаскелл:

sum\_power n 1 = 1/n

sum\_power n p | p > 0 = (1 / (n^p)) + (sum\_power n (p-1))

1 балл:

Функция не рассчитывает частичную сумму, а рассчитывает иное, например, n-p.

2 балла:

Функция корректна.

1. Доказать, что .

Пример доказательства:

Для доказательства этого свойства можно использовать определение типа List (A) и самой функции append (в инфиксной записи используется символ \*).

1°. L = [] : [] \* [] = [] = L. Базис индукции доказан.

2°. . Теперь пусть применяется конструктор: a : L. Не­об­хо­димо доказать, что (a : L) \* [] = a : L. Это делается при помощи применения второго кло­за определения функции append:.

1 балл:

Доказан только базис индукции. Верно сформулирован шаг индукции. Доказательство не завершено.

2 балла:

Верно сформулированы и доказаны базис и шаг индукции.

1. Докажите эквивалентность ассоциативности функции Append (++), то есть, что для любых списков L1, L2, L3 истинно L1 ++ (L2 ++ L3) = (L1 ++ L2) ++ L3

Append ([], L2) = L2

Append (H:T, L2) = H : Append (T, L2))

Пример доказательства:

При доказательстве индукция будет проводиться по пер­во­му операнду, т.е. списку L1:

1°. L1 = []:

([] \* L2) \* L3 = (L2) \* L3 = L2 \* L3.

[] \* (L2 \* L3) = (L2 \* L3) = L2 \* L3.

2°. Пусть для списков L1, L2 и L3 ассоциативность функции append доказана. Необходимо доказать для (a : L1), L2 и L3:

((a : L1) \* L2) \* L3 = (a : (L1 \* L2)) \* L3 = a : ((L1 \* L2) \* L3).

(a : L1) \* (L2 \* L3) = a : (L1 \* (L2 \* L3)).

Как видно, последние два выведенных выражения равны, т.к. для списков L1, L2 и L3 ас­социативность полагается доказанной.

1 балл

Доказан только базис индукции. Верно сформулирован шаг индукции. Доказательство начато, но не завершено.

2 балла

Верно сформулированы и доказаны базис и шаг индукции.

1. Докажите эквивалентность двух определений reverse

Определение 1 (рекурсивное):

reverse [] = []

reverse (H : T) = (reverse T) \* [H]

Определение 2 (с аккумулятором):

reverse' L = rev L []

rev [] L = L

rev (H : T) L = rev T (H : L)

Требуется доказать, что.

Пример доказательства:

1°. Базис — L = []:

reverse [] = [].

reverse’ [] = rev [] [] = [].

2°. Шаг — пусть для списка L тождество функций reverse и reverse’ доказано. Необходимо доказать его для списка (H : L).

reverse (H : L) = (reverse L) \* [H] = (reverse’ L) \* [H].

reverse’ (H : L) = rev (H : L) [] = rev L (H : []) = rev L [H].

Теперь необходимо доказать равенство двух последних выведенных выражений для лю­бых списков над типом A. Это также делается по индукции:

2.1°. Базис — L = []:

(reverse’ []) \* [H] = (rev [] []) \* [H] = [] \* [H] = [H].

rev [] [H] = [H].

2.2°. Шаг — L = (A : T):

(reverse’ (A : T)) \* [H] = (rev (A : T) []) \* [H] = (rev T (A : [])) \* [H] = (rev T [A]) \* [H].

rev (A : T) [H] = rev L (A : H).

Если дальше пытаться проводить до­казательство по индукции для новых выведенных выражений, то эти самые выражения бу­дут все усложняться и усложняться. Нужна индукционная гипотеза, доказать которую проще, и которая обобщала бы шаг индукции (reverse’ L) \* [H] = rev L [H].

Индукционная гипотеза: (reverse’ L1) \* L2 = rev L1 L2.

Базис индукции для этой гипотезы очевиден. Шаг индукции в применении к вы­ра­же­нию в пункте 2.2 выглядит следующим образом:

(reverse’ (A : T)) \* L2 = (rev (A : T) []) \* L2 = (rev T [A]) \* L2 = ((reverse’ T) \* [A]) \* L2 = = (reverse’ T) \* ([A] \* L2) = (reverse’ T) \* (A : L2).

rev (A : T) L2 = rev T (A : L2) = (reverse’ T) \* (A : L2).

Что и требовалось доказать.

1 балл:

Доказан только базис индукции. Верно сформулирован шаг индукции, но доказательство остановилось на бесконечном развертывании.

2 балла:

Верно сформулированы и доказаны базисы индукции. Сформулирована и доказана дополнительная индукционная гипотеза. Доказан шаг индукции.

1. Докажите эквивалентность двух определений функций вычисления факториала.

Определение 1 (рекурсивное):

Factorial (0) = 1

Factorial (N) = N \* Factorial (N-1)

Определение 2 (с накопителем):

Factorial\_A (N) = F (N, 1)

F (0, A) = A

F (N, A) = F ((N – 1), (N \* A))

Пример доказательства:

1 балл:

Доказан только базис индукции. Верно сформулирован шаг индукции, доказательство начато, но не завершено.

2 балла:

Верно сформулированы и доказаны базис и шаг индукции.

1. Докажите эквивалентность двух определений функций возведения в степень.

Определение 1 (рекурсивное):

Power (X, 0) = 1

Power (X, N) = X \* Power (X, N – 1)

Определение 2 (с накопителем):

Power\_A (X, N) = P (X, N, 1)

P (X, 0, A) = A

P (X, N, A) = P (X, N – 1, X \* A)

Пример доказательства:

1 балл:

Доказан только базис индукции. Верно сформулирован шаг индукции, доказательство начато, но не завершено.

2 балла:

Верно сформулированы и доказаны базис и шаг индукции.

1. Докажите эквивалентность двух определений функций вычисления n-го члена ряда 

Определение 1 (рекурсивное):

Summ\_T (1) = 1

Summ\_T (N) = N + Summ\_T (N – 1)

Определение 2 (с накопителем):

Summ\_T\_A (N) = ST (N, 0)

ST (0, A) = A

ST (N, A) = ST (N – 1, N + A)

Пример доказательства:

1 балл:

Доказан только базис индукции. Верно сформулирован шаг индукции, доказательство начато, но не завершено.

2 балла:

Верно сформулированы и доказаны базис и шаг индукции.

1. Докажите эквивалентность двух определений функций возведения X в степень N.

Определение 1 (рекурсивное):

Power (X, 0) = 1

Power (X, N) = X \* Power (X, N – 1)

Определение 2 (с накопителем):

Power\_A (X, N) = P (X, N, 1)

P (X, 0, A) = A

P (X, N, A) = P (X, N – 1, X \* A)

Пример доказательства:

1 балл:

Доказан только базис индукции. Верно сформулирован шаг индукции. Доказательство начато, но не завершено.

2 балла:

Верно сформулированы и доказаны базис и шаг индукции.

1. Докажите эквивалентность двух определений функций вычисления N-го члена ряда Фибоначчи.

Определение 1 (рекурсивное):

Fib 1 = 1

Fib 2 = 1

Fib n = fib (n-1) + fib (n-2)

Определение 2 (с накопителем):

fibb 1 prev pprev = prev

fibb n prev pprev = fibb (n-1) (prev+pprev) prev

fib n = fibb n 1 0

Пример доказательства:

1 балл:

Доказан только базис индукции. Верно сформулирован шаг индукции, но доказательство остановилось на бесконечном развертывании.

2 балла:

Верно сформулированы и доказаны базисы индукции. Сформулирована и доказана дополнительная индукционная гипотеза. Доказан шаг индукции.

**Список вопросов к зачету по курсу «Функциональное программирование»**

***Вопрос 1.***

*Структуры данных. Типы. Аппликация. Абстракция. Выражение в форме оператор/операнд.*

***Вопрос 2.***

*Абстрактный язык функционального программирования. Нотация (клозы, образцы, охрана). Операционная семантика.*

***Вопрос3****.*

*Абстрактный язык функционального программирования. Нотация (клозы, образцы, охрана). Операционная семантика. Функции высших порядков.*

***Вопрос 4.***

*Приемы программирования. Введение подфункций.*

***Вопрос5.*** *Приемы программирования. Накапливающий параметр.*

***Вопрос6.*** *Приемы программирования. Итерационная форма программ.*

***Вопрос7.*** *Приемы программирования. Хвостовая рекурсия.*

***Вопрос 8.***

*Лисп. Атомы и списки. Точечное представление. Базовые функции обработки данных и примитивы управления. Связывание, деструктивное присваивание.*

***Вопрос 9.***

*Лисп. Имя и значение символа. Определение функций.*

***Вопрос 10.***

*Лисп. Применяющие функционалы. Замыкания. Макросы. Ввод и вывод.*

***Вопрос 11.***

*Синтаксически-ориентированное программирование. Язык абстрактного описания областей. Построение абстрактного описания (синтаксиса) списков, списочных структур. Построение типовых алгоритмов обработки данных для этих областей.*

***Вопрос 12.***

*Синтаксически-ориентированное программирование. Язык абстрактного описания областей. Построение абстрактного описания (синтаксиса) деревьев, бинарных деревьев. Построение типовых алгоритмов обработки данных для этих областей.*

***Вопрос 13.***

*Доказательства свойств программ. Определение индукции по структурам данных. Списочная индукция, sexpr-индукция. Примеры. Ограничения.*

***Вопрос 14.***

*Вопростические основания. Формальная система. Аксиоматезируемость. Синтаксическое и семантическое равенство*

***Вопрос 15.***

*Вопростические основания. Элементы лямбда-исчисления (выражение, подстановка, аксиомы, правила вывода, стратегии редукции, нормальная форма, Теорема Черча-Россера, пародоксальный комбинатор).*

***Вопрос 16.***

*Трансформации программ. Эквивалентные и корректные преобразования.*

***Вопрос 17.***

*Трансформации программ. Система преобразований Берсталла и Дарлингтона: instantiation, folding, unfolding, low, abstraction.*

***Вопрос 18.***

*Трансформации программ. Трансформационный синтез программ и его проблемы.*

***Вопрос 19.***

*Частичные вычисления. Остаточная программа. Интерпретатор. Компилятор. Генератор компиляторов.*

***Вопрос 20.***

*Частичные вычисления. Проекции Футамуры. Принципы построения частичных вычислителей.*