МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГУ»)

УТВЕРЖДАЮ

Заведующий кафедрой

теории функций и геометрии

 Е.М. Семенов

23.03.2025 г.

РАБОЧАЯ ПРОГРАММА УЧЕБНОЙ ДИСЦИПЛИНЫ

Б1.В.1.ДВ.03.01. Типовое λ-исчисление и язык программирования Haskell

1. Код и наименование специальности: 01.05.01 Фундаментальные математика и механика

2. Специализация: Современные методы теории функций в математике и механике

3. Квалификация выпускника: Математик. Механик. Преподаватель

4. Форма обучения: Очная

5. Кафедра, отвечающая за реализацию дисциплины: Кафедра теории функций и геометрии

6. Составители программы: Морозов Александр Вячеславович, преподаватель кафедры теории функций и геометрии

7. **Рекомендована:** Научно-методическим Советом математического факультета, протокол № 0500-03 от 18.03.2025 г.

8. Учебный год: 2028/2029 **Семестр(-ы):** 7

9. Цели и задачи учебной дисциплины:

Целями освоения учебной дисциплины являются:

– формирование у студентов теоретических знаний и практических навыков программирования на функциональном языке Haskell;

– изучение основ типового λ-исчисления как теоретического фундамента функционального программирования.

Задачи дисциплины:

– раскрыть основы функционального и логического программирования, структуру языка Haskell, его синтаксис и семантические конструкции;

– сформировать у студентов теоретическую базу и практические навыки, необходимые для работы в парадигме функционального программирования;

– выработать у обучающихся навыки самостоятельно ориентироваться в языках функционального программирования и понимать их основные принципы работы.

10. Место учебной дисциплины в структуре ОПОП:

Дисциплина «Типовое λ-исчисление и язык программирования Haskell» относится к дисциплинам по выбору по специальности 01.05.01 – Фундаментальные математика и механика. Дисциплина «Типовое λ-исчисление и язык программирования Haskell» базируется на знаниях, полученных в рамках изучения дисциплин «Технология программирования и работа на ЭВМ», «Математический практикум», «Математическая логика», а также предшествующих математических дисциплин, использующих соответствующие методы.

11. Планируемые результаты обучения по дисциплине/модулю (знания, умения, навыки), соотнесенные с планируемыми результатами освоения образовательной программы (компетенциями выпускников):

Код	Название компетенции	Код(ы)	Индикатор(ы)	Планируемые результаты обучения
ПК-2	Способен проводить исследования по обработке и анализу научной информации и результатов иссл едований методами теории функций.	ПК-2.2	Умеет разрабатывать математические модели в области естествознания, экономики и управления, а также реализовывать алгоритмы математических моделей на базе пакетов прикладных программ моделирования.	Знать: - современные методы разработки и реализации моделей, используя теорию функций. Уметь: - разрабатывать математические модели в области естествознания, экономики и управления, а также реализовывать алгоритмы математических моделей на базе пакетов прикладных программ моделирования. Владеть навыками: -проведения научно-исследовательской деятельности в области

				решения задач аналитического характера
ПК-3	Способен к построению моделей и оптимальному решению теоретических и прикладных задач математики и механики на основе методов теории функций и геометрии.	ПК-3.1	Знает современные методы разработки и реализации математических моделей.	Знать: - современные методы разработки и реализации математических моделей. Уметь: - строить модели и оптимальные решения теоретических и прикладных задач математики и механики на основе методов теории функций и геометрии. Владеть навыками: - построения моделей прикладных процессов; - навыками применения современных инструментальных средств к решению прикладных задач.
		ПК-3.2	Владеет навыками построения моделей прикладных процессов и навыками применения современных инструментальных средств к решению прикладных задач.	

12. Объем дисциплины в зачетных единицах/часах в соответствии с учебным планом — 3/108.

Форма промежуточной аттестации зачет.

13. Трудоемкость по видам учебной работы:

Вид учебной работы	Трудоемкость (часы)	
	Всего	По семестрам
		7 сем.
Аудиторные занятия	34	34
в том числе: лекции	-	-
практические	34	34
лабораторные	-	-
курсовая работа	-	-
Самостоятельная работа	74	74
Промежуточная аттестация	-	-
Итого:	108	108

13.1. Содержание дисциплины:

№ п/п	Наименование раздела дисциплины	Содержание раздела дисциплины	Реализация раздела дисциплины с помощью онлайн-курса, ЭУМК

Практические занятия			
1	Основы функционального программирования: типизированное и чистое лямбда-исчисление, комбинаторная логика.	**Тема 1** Введение: функциональное и императивное программирование. Лямбда-исчисление.Применение и абстракция. Свободные и связанные переменные. Комбинаторы. Функции нескольких переменных, каррирование. Подстановка, лемма подстановки. Бета-преобразование. Эта-преобразование. Расширение чистого лямбда-исчисления. **Тема 2** Теорема о неподвижной точке, Y-комбинатор. Редексы. Одношаговая и многошаговая редукция. Нормальная форма. Редукционные графы. Теорема Чёрча-Россера. Следствия: редуцируемость к нормальной форме, единственность нормальной формы. Стратегии редукции. Головная и слабая головная нормальные формы. Теорема о нормализации. Механизмы вызова в функциональных языках. **Тема 3** Роль типов в языках программирования. Предтермы. Утверждения о типизации. Контексты. Правила типизации по Карри и по Чёрчу. Деревья вывода типов. Свойства типизированного лямбда-исчисления. Связь между системами Карри и Чёрча. Проблемы разрешимости. Сильная и слабая нормализация. Соответствие Карри-Говарда.	-
2	Язык программирования Haskell: синтаксис, семантика, стандартная библиотека.	**Тема 1** Стандарт языка Haskell. Компилятор GHC. Интерпретатор GHCi. Hoogle. Связывание. Кодирование функций. Рекурсия. Основные синтаксические конструкции. Базовые типы. Система модулей. Частичное применение, каррирование. Инфиксные операторы, сечения. Бесточечный стиль. Кодирование рекурсивных функций. **Тема 2** Ошибки. Основание. Строгие и нестрогие функции. Ленивое и энергичное исполнение. Функция seq. Списки, стандартные функции для работы с ними. Функции высших порядков над списками. Бесконечные структуры данных. Арифметические последовательности. Генерация (выделение) списков. **Тема 3** Алгебраические типы данных. Типы суммы и произведения. Параметризованные и рекурсивные типы. Сопоставление с образцом, его семантика. Объявления type и newtype. Метки полей.	-

		Тема 4 Параметрический и специальный полиморфизм. Классы типов. Объявления представителей (instance declaration). Методы по умолчанию. Расширение класса типов (class extinsion). Пример: классы Eq и Ord. Операторы над типами как параметры в определении класса. Класс типов Functor. Стандартные классы типов: Enum, Bounded, Show и Read. Класс Num и его расширения. Стандартная внутренняя реализация классов типов. **Тема 5** Свёртки списков. Правая и левая свёртки. Ленивые и энергичные версии свёрток. Развертки. Моноиды. Представители класса типов Monoid. Класс типов Foldable. Минимальная полная реализация. Реализация методов по умолчанию. Универсальность правой свертки. Свойство слияния (fusion law) для свертки. Свойство foldr-map.	
3	Управление эффектами с помощью аппликативных функторов и монад.	**Тема 1** Полное определение класса типов Functor. Стандартные представители класса Functor. Законы для функторов. Класс типов Pointed и законы для него. Класс типов Applicative и его представители. Два способа объявления списка аппликативным функтором. Законы Applicative и Applicative/Functor. Полное определение класса типов Applicative. Тип композиции однопараметрических конструкторов типа как функтор и аппликативный функтор. **Тема 2** Аппликативные парсеры. Класс типов Alternative. Законы Alternative. Интерфейс Alternative для аппликативного парсера. Класс типов Traversable. Законы Traversable. Реализация по умолчанию для методов базовых классов. Реализация представителей Traversable. Функтор Const и реализации методов Foldable по умолчанию. Моноидальные функторы. Тип композиции однопараметрических конструкторов типа как Traversable. **Тема 3** Монады. Класс типов Monad. Пример: монада Identity. Законы класса типов Monad. do-нотация. Монада Maybe: вычисления, которые могут завершиться неудачей. Монада списка: вычисления со множественными результатами. Программирование с помощью стандартных монад. Связь Monad и Functor. Связь Monad и Applicative. Монадические комбинаторы и их свойства. Законы класса типов Monad в разных представлениях.	-

		Тема 4 Монады для записи в лог, чтения из внешнего окружения и работы с изменяемым состоянием: Reader, Writer, State. Ввод-вывод в чистых языках. Монада IO. Тип IORef. Работа с псевдослучайными величинами. Взаимодействие с файловой системой. **Тема 5** Класс MonadPlus и его законы. Стандартные функции, использующие контекст MonadPlus/Alternative. Монада Except: вычисление, допускающее возбуждение и перехват исключений. Мультипараметрические классы типов. Проблема комбинирования монадических эффектов. **Тема 6** Трансформеры монад: монадические вычисления со множественными эффектами. Законы для класса типов MonadTrans. Библиотеки mtl и transformers: сравнение. Трансформеры и ввод-вывод.	
4	Системы типов функциональных языков.	**Тема 1** Вывод типов. Главный (наиболее общий) тип. Свойства подстановки типов. Композиция подстановок. Унификатор, теорема унификации. Главная пара. Алгоритм Хиндли-Дамаса-Милнера. let-полиморфизм и типы высших рангов. **Тема 2** Алгебраические типы. Экспоненциальные типы. Параметризованные типы. Изоморфизм между типами. Рекурсивные типы. Мю-нотация. Оператор неподвижной точки для типов. Рекурсивный тип как неподвижная точка нерекурсивного функтора. Катаморфизм, F-алгебры. Анаморфизм, F-коалгебры. Гилеморфизм. **Тема 3** Зипперы. Дифференцирование алгебраических типов. Линзы ван Лаарховена. Библиотека lens. Template Haskell. Пользовательские линзы.	-

13.2. Темы (разделы) дисциплины и виды занятий:

№ п/п	Наименование раздела дисциплины	Виды занятий (часов)				
		Лекции	Практические	Лабораторные	Самостоятельная работа	Всего
1	Основы функционального программирования: типизированное и чистое лямбда-исчисление, комбинаторная логика.	-	7	-	14	21

2	Язык программирования Haskell: синтаксис, семантика, стандартная библиотека.	-	10	-	23	33
3	Управление эффектами с помощью аппликативных функторов и монад.	-	10	-	23	33
4	Системы типов функциональных языков.	-	7	-	14	21
Итого		-	34	-	74	108

14. Методические указания для обучающихся по освоению дисциплины:

В процессе преподавания дисциплины используются такие виды учебной работы, как практические занятия, а также различные виды самостоятельной работы обучающихся.

Методические рекомендации студентам к практическим занятиям

Практические занятия представляют собой важнейший компонент учебного процесса. Они выполняют ключевую функцию в углубленном освоении студентами теоретического материала, представленного в рамках лекционных курсов.

Структура практического занятия выстроена следующим образом. На начальном этапе преподаватель проводит обсуждение темы, сопровождая его разбором конкретных примеров и задач, а также предоставляет методические рекомендации по выполнению предстоящей практической работы. Данный этап позволяет студентам прояснить сложные аспекты изучаемого материала и более эффективно подготовиться к самостоятельной деятельности. Кроме того, преподаватель обозначает перечень рекомендуемой литературы, формулирует цели и задачи ее изучения, что способствует систематизации знаний.

На следующем этапе студенты получают индивидуальные или групповые задания для выполнения практической работы и приступают к их решению, применяя усвоенные теоретические положения. Этот процесс направлен на достижение глубокого понимания учебного материала через его практическую апробацию.

После завершения самостоятельной работы проводится коллективный разбор аналогичного задания под руководством преподавателя, что служит дополнительным механизмом закрепления изученного материала. В заключительной части занятия студенты получают домашнее задание, целью которого является самостоятельное применение приобретенных знаний, что способствует их дальнейшему углублению и систематизации.

Курс построен таким образом, чтобы позволить студентам максимально проявить способность к самостоятельной работе. Для успешной самостоятельной работы предполагается тесный контакт с преподавателем.

Изучение дисциплины следует начинать с проработки настоящей рабочей программы, особое внимание, уделяя целям и задачам, структуре и содержанию курса.

Студентам рекомендуется получить в библиотеке учебную литературу по дисциплине, необходимую для эффективной работы на всех видах аудиторных занятий, а также для самостоятельной работы по изучению дисциплины.

Своевременное и качественное выполнение самостоятельной работы базируется на соблюдении настоящих рекомендаций и изучении рекомендованной литературы. Студент может дополнить список использованной литературы современными источниками, не представленными в списке рекомендованной

литературы, и в дальнейшем использовать собственные подготовленные учебные материалы при написании курсовых и дипломных работ.

Успешное освоение курса предполагает активное, творческое участие студента путем планомерной, повседневной работы.

Все выполняемые студентами самостоятельно задания (выполнение домашних заданий) подлежат последующей проверке преподавателем.

15. Перечень основной и дополнительной литературы, ресурсов интернет, необходимых для освоения дисциплины:

а) основная литература:

№ п/п	Источник
1	*Барендрегт Х. Ламбда-исчисление. Его синтаксис и семантика / пер. с англ. — М.: Мир, 1985. — 606 с.*
2	*Hutton G. Programming in Haskell. — Cambridge: Cambridge University Press, 2007. — 171 p.*
3	*Берд Р. Жемчужины проектирования алгоритмов. Функциональный подход / пер. с англ. — М.: ДМК Пресс, 2013. — 312 с.*
4	*Bragilevsky V. Haskell in Depth. — New York: Manning Publications, 2019. — 625 p.*

б) дополнительная литература:

№ п/п	Источник
1	*Маклейн С. Категории для работающего математика / пер. с англ. — М.: Физматлит, 2024. — 351 с.*
2	*Липовача М. Изучай Haskell во имя добра! / пер. с англ. Д. Леушина, А. Синицына, Я. Арсанукаева. — М.: ДМК Пресс, 2017. — 490 с.: ил.*
3	*Milewski B. Category Theory for Programmers. — 2nd ed. — CC BY-SA 4.0, 2020. — 480 p.*
4	*Vene V. Categorical programming with inductive and coinductive types: PhD Thesis. — Tartu: University of Tartu, 2000. — 116 p.*

в) информационные электронно-образовательные ресурсы**:**

№ п/п	Источник
1	*Racket Documentation: Category Theory for Programmers [Электронный ресурс]. – URL: https://docs.racket-lang.org/ctp/index.html (дата обращения: 10.06.2025).*
2	*Черевиченко Г. Учебник по теории категорий для программистов и математических логиков [Электронный ресурс]. – URL: https://github.com/George66/Textbook (дата обращения: 10.06.2025).*

3	*Москвин Д. Н. Основы языка Haskell [Электронный ресурс] // Stepik. – URL: https://stepik.org/course/75 (дата обращения: 10.06.2025).*
4	*Москвин Д. Н. Основы языка Haskell (часть 2) [Электронный ресурс] // Stepik. – URL: https://stepik.org/course/693 (дата обращения: 10.06.2025).*

16. Перечень учебно-методического обеспечения для самостоятельной работы:

Курс дисциплины построен таким образом, чтобы позволить студентам максимально проявить способность к самостоятельной работе. Для успешной самостоятельной работы предполагается тесный контакт с преподавателем.

Изучение дисциплины следует начинать с проработки настоящей рабочей программы, особое внимание, уделяя целям и задачам, структуре и содержанию курса.

Студентам рекомендуется получить в библиотеке учебную литературу по дисциплине, необходимую для эффективной работы на всех видах аудиторных занятий, а также для самостоятельной работы по изучению дисциплины.

Своевременное и качественное выполнение самостоятельной работы базируется на соблюдении настоящих рекомендаций и изучении рекомендованной литературы. Студент может дополнить список использованной литературы современными источниками, не представленными в списке рекомендованной литературы, и в дальнейшем использовать собственные подготовленные учебные материалы при написании курсовых и дипломных работ.

Успешное освоение курса предполагает активное, творческое участие студента путем планомерной работы.

№ п/п	Источник
1	*Липовача М. Изучай Haskell во имя добра! / пер. с англ. Д. Леушина, А. Синицына, Я. Арсанукаева. — М.: ДМК Пресс, 2017. — 490 с.: ил.*
2	*Milewski B. Category Theory for Programmers. — 2nd ed. — CC BY-SA 4.0, 2020. — 480 p.*
3	*Положение об организации самостоятельной работы обучающихся в Воронежском государственном университете.*

17. Образовательные технологии, используемые при реализации учебной дисциплины, включая дистанционные образовательные технологии (ДОТ), электронное обучение (ЭО), смешанное обучение):

При реализации дисциплины используются следующие образовательные технологии: логическое построение дисциплины, установление межпредметных связей, актуализация личного и учебно-профессионального опыта обучающихся, включение элементов дистанционных образовательных технологий.

Осуществляется интерактивная связь с преподавателем через сеть интернет, проводятся индивидуальные онлайн консультации. Практические работы выполняются на компьютерной технике с использованием различных информационных технологий.

Перечень необходимого программного обеспечения: актуальная операционная система на базе ядра Linux, ONLYOFFICE или аналоги, браузер Mozilla Firefox, Chromium или аналоги, GHC, GHCi, Visual Studio или любые другие редакторы кода с подсветкой синтаксиса, экран, ноутбук, мультимедиапроектор.

18. Материально-техническое обеспечение дисциплины:

Для проведения практических занятий используются: Компьютерный класс: специализированная мебель, маркерная доска, персональные компьютеры
Ubuntu (бесплатное и/или свободное ПО, лицензия: https://ubuntu.com/download/desktop) 20.04
Visual Studio Community (бесплатное и/или свободное ПО, лицензия: https://visualstudio.microsoft.com/ru/vs/community/)
LibreOffice (GNU Lesser General Public License (LGPL), бесплатное и/или свободное ПО, лицензия: https://ru.libreoffice.org/about-us/license/)
GHC (BSD 3, бесплатное и/или свободное ПО, лицензия: https://gitlab.haskell.org/ghc/ghc/-/blob/master/LICENSE)

Для самостоятельной работы используются классы с компьютерной техникой, оснащенные необходимым программным обеспечением, электронными учебными пособиями и законодательно-правовой и нормативной поисковой системой, имеющий выход в глобальную сеть.

19. Оценочные средства для проведения текущей и промежуточной аттестаций:

Порядок оценки освоения обучающимися учебного материала определяется содержанием следующих разделов дисциплины:

№ п/п	Наименование раздела дисциплины (модуля)	Компетенция(и)	Индикатор(ы) достижения компетенции	Оценочные средства
1	Основы функционального программирования: типизированное и чистое лямбда-исчисление, комбинаторная логика.	ПК-2, ПК-3	ПК-2.2, ПК-3.1, ПК-3.2	Домашняя работа № 1
2	Язык программирования Haskell: синтаксис, семантика, стандартная библиотека.	ПК-2, ПК-3	ПК-2.2, ПК-3.1, ПК-3.2	Домашняя работа № 2
3	Управление эффектами с помощью аппликативных функторов и монад.	ПК-2, ПК-3	ПК-2.2, ПК-3.1, ПК-3.2	Домашняя работа № 3
4	Системы типов функциональных языков.	ПК-2, ПК-3	ПК-2.2, ПК-3.1, ПК-3.2	Домашняя работа № 4

№ п/п	Наименование раздела дисциплины (модуля)	Компетенция(и)	Индикатор(ы) достижения компетенции	Оценочные средства
Промежуточная аттестация форма контроля – зачет				Список вопросов к зачету

20. Типовые оценочные средства и методические материалы, определяющие процедуры оценивания

20.1. Текущий контроль успеваемости

ДОМАШНЕЕ ЗАДАНИЕ №1

Задача 1. *(1 балл)* Выделите свободные и связанные переменные в термах и выполните указанные подстановки:

1. λyz.xyw(zx) [x := λy.yw];
2. λxy.xy(λx.xy)x [x := λz.z];
3. xy(λxz.xyz)y [y := xz].

Определите, возможно ли в получившемся терме выполнить β-преобразование.

Задача 2. *(1 балл)* Уберите лишние скобки и при возможности выполните β - преобразование

1. (x(λx.((xy)x))y);
2. ((λp.(λq.((q(pr))s)))((q(pr))s)).

Задача 3. *(1 балл)* Покажите, что для любых M и N выполняется

λx.MN = S(λx.M)(λx.N).

Задача 4. *(2 балла)* Покажите равенство комбинаторов

1. SKK = I;
2. B = S(KS)K.

Задача 5. *(2 балла)* Реализуйте функцию возведения в степень для чисел Чёрча.

Задача 6. *(2 балла)* Пусть имеются взаимно-рекурсивное определение функций f и g:

f = Ffg, g = Gfg.

Используя Y-комбинатор, найдите не рекурсивные определения для f и g.

Задача 7. *(1 балл)* Найдите замкнутые термы в нормальной форме, являющиеся обитателями типа

1. α→β→γ→α
2. α→β→γ→β
3. α→β→α→α
4. α→α→α→α

Сколько разных (с точностью до αβ-эквивалентности) термов каждого типа вы можете привести?

Задача 8. *(2 балла)* Найдите замкнутые термы в нормальной форме, являющиеся обитателями типа

1. (δ→δ→α)→(α→β→γ)→(δ→β)→δ→γ
2. (δ→δ→α)→(γ→α)→(α→β)→δ→γ→β

Критерии оценивания и шкала оценки домашнего задания №1.

Оценка	Критерии выставления оценки
«Отлично» (7-12 баллов)	Все конструкции соответствуют спецификации, а преобразования проведены аккуратно и снабжены комментариями.
«Хорошо» (4-6 баллов)	В преобразованиях или реализации есть неточности, переходы в доказательствах не обоснованы.
«Удовлетворительно» (2-3 балла)	Решение в целом соответствует базовым требованиям, однако содержит существенные погрешности: преобразования выполнены с ошибками, часть ключевых шагов опущена или недостаточно аргументирована, доказательства носят фрагментарный характер.
«Неудовлетворительно» (0 баллов)	Неустранимые ошибки в преобразованиях или реализации.

ДОМАШНЕЕ ЗАДАНИЕ №2

Задача 1. *(1 балл)* Какому известному вам библиотечному оператору, конструктору или функции эквивалентно выражение uncurry (flip const) ?

Задача 2. *(1 балл)* В модуле Data.Tuple стандартной библиотеки определена функция swap :: (a,b) ->(b,a) , переставляющая местами элементы пары:

GHCi> swap (1,'A')
('A', 1)

Эта функция может быть выражена в виде swap = f (g h) , где f , g и h — некоторые идентификаторы из следующего набора:

{curry, uncurry, (,), const, flip}

Укажите через запятую подходящую тройку f , g , h .

Задача 3. *(1 балл)* Определите функцию вычисляющую двойной факториал, то есть произведение натуральных чисел, не превосходящих заданного числа и имеющих ту же четность. Например:

7!! = 7·5·3·1, 8!! = 8·6·4·2.

Сигнатура:

doubleFact :: Integer -> Integer

** Предполагается, что аргумент функции может принимать только неотрицательные значения.*

Задача 4. *(1 балл)* Реализуйте функцию, находящую элементы следующей рекуррентной последовательности

a_0 = 1;
a_1 = 2;
a_2 = 3;
a_{k+3} = a_{k+2} + a_{k+1} − 2a_k .

Сигнатура:

```
seqA :: Integer -> Integer
```

Найдите эффективное решение со сложностью не хуже линейной по числу вызовов арифметических операторов. Пример:

```
GHCi> seqA 7
-10
```

Задача 5. *(2 бала)* Понятие чисел Фибоначчи можно расширить, потребовав, чтобы рекуррентное соотношение выполнялось для произвольных целых значений аргумента, в том числе и отрицательных. Реализуйте функцию, вычисляющую числа Фибоначчи так, чтобы она удовлетворяла этому требованию.

```
fibonacci :: Integer -> Integer
fibonacci n = undefined
```

Реализация должна иметь сложность не хуже линейной по числу вызовов оператора сложения.

```
GHCi> fibonacci (-99)
218922995834555169026
```

Задача 6. *(1 балл)* Составить список сумм соответствующих элементов трех заданных списков. Длина результирующего списка должна быть равна длине самого длинного из заданных списков, при этом «закончившиеся» списки не должны давать вклада в суммы.

```
sum3 :: Num a => [a] -> [a] -> [a] -> [a]
sum3 = undefined
```

Задача 7. *(1 балл)* Сформируйте список цифр заданного целого числа.

```
digits :: Integer -> [Integer]

digits = undefined
```

Задача 8. *(1 балл)* Определите, содержит ли заданное целое число все цифры от 1 до 9. (Используйте функцию digits из предыдущего задания.)

```
containsAllDigits :: Integer -> Bool
containsAllDigits = undefined
```

Задача 9. *(1 балл)* Определите, содержит ли заданное целое число все цифры от 1 до 9 в точности по одному разу. (Используйте функцию digits из предыдущего задания.)

```
containsAllDigitsOnes :: Integer -> Bool
containsAllDigitsOnes = undefined
```

Критерии оценивания и шкала оценки домашнего задания №2.

Оценка	Критерии выставления оценки
«Отлично» (7-10)	Программа компилируется и выполняется без ошибок. Логика программы полностью соответствует поставленной задаче, нет семантических ошибок. Код грамотно структурирован. Переменные и функции имеют осмысленные, корректные имена. Соблюдается единая стилистика. Присутствуют минимально необходимые комментарии для пояснения не очевидных решений.
«Хорошо» (4-6)	Программа работает, но содержит незначительные логические недочёты. Структура кода в целом понятна, но есть небольшие нарушения. Имена переменных/функций в основном адекватны, но встречаются не информативные варианты. Отсутствует общая стилистика кода. Комментарии недостаточны или избыточны.
«Удовлетворительно» (2-3)	Программа компилируется, но содержит серьёзные логические ошибки. Код плохо структурирован. Имена переменных/функций не информативны. Стиль не соблюдается (хаотичные отступы, смешение стилей). Комментарии отсутствуют или не соответствуют коду.
«Неудовлетворительно» (0-1)	Программа не компилируется или не запускается. Критические логические ошибки, делающие решение нерабочим. Полное отсутствие структуры и читаемости. Имена переменных/функций не несут смысла. Грубые нарушения стиля.

ДОМАШНЕЕ ЗАДАНИЕ №3

Задача 1. *(1 балл)* «Окружите» каждый элемент списка заданными «скобками», используя монаду списка и do - нотацию:

```
surround :: a -> a -> [a] -> [a]
surround x y zs = do undefined
```

Пример:

```
GHCi> surround '{' '}' "abcd"
"{a}{b}{c}{d}"
```

Задача 2. *(1 балл)* Ассоциативным списком называют список пар (ключ, значение). Реализуйте функцию поиска в таком списке, возвращающую список всех значений с заданным ключом. Используйте монаду списка и do -нотацию.

```
lookups :: (Eq a) => a -> [(a, b)] -> [b]
```

```
lookups x ys = do undefined
```

Пример:

GHCi> lookups 2 [(1, "one"), (2, "two"), (3, "three"), (2, "two'")]
["two", "two'"]

Задача 3. *(2 балла)* Разложите положительное целое число на два сомножителя всевозможными способами, используя монаду списка и do -нотацию.

```
factor2 :: Integer -> [(Integer, Integer)]
factor2 n = do undefined
```

Пары должны быть уникальными, первый элемент пары не должен превышать второй, результат следует упорядочить лексикографически, в возрастающем порядке.

Пример:

GHCi> factor2 45
[(1, 45), (3, 15), (5, 9)]

Задача 4. *(2 балла)* Вычислите модули разностей между соседними элементами списка, используя монаду списка и do -нотацию.

```
absDiff :: Num a => [a] -> [a]
absDiff xs = do undefined
```

Пример:

GHCi> absDiff [2, 7, 22, 9]
[5, 15, 13]

Задача 5. *(2 балла)* Для типа данных

```
data OddC a = Un a | Bi a a (OddC a) deriving (Eq, Show)
```

(контейнер-последовательность, который по построению может содержать только нечетное число элементов) реализуйте функцию со следующей сигнатурой

```
concat3OC :: OddC a -> OddC a -> OddC a -> OddC a
```

конкатенирующую три таких контейнера в один:

GHCi> tst1 = Bi 'a' 'b' (Un 'c')
GHCi> tst2 = Bi 'd' 'e' (Bi 'f' 'g' (Un 'h'))
GHCi> tst3 = Bi 'i' 'j' (Un 'k')
GHCi> concat3OC tst1 tst2 tst3
Bi 'a' 'b' (Bi 'c' 'd' (Bi 'e' 'f' (Bi 'g' 'h' (Bi 'i' 'j' (Un 'k')))))

Обратите внимание, что соображения четности запрещают конкатенацию двух контейнеров OddC .

Задача 6. *(1 балл)* Напишите функцию, вычисляющую числа Фибоначчи с использованием монады State.

```
fib :: Int -> Integerfib n = fst $ execState (replicateM n fibStep) (0,1)
fibStep :: State (Integer,Integer) ()
fibStep = do undefined
```

Задача 7. *(2 балла)* Напишите функцию

```
while :: IORef a -> (a -> Bool) -> IO () -> IO ()
while ref p action = do undefined
```

позволяющую кодировать «императивные циклы» следующего вида:

```
factorial :: Integer -> IO Integer
factorial n = do
        r <- newIORef 1
        i <- newIORef 1
        while i (<= n) ( do
                ival <- readIORef i
                modifyIORef' r (* ival)
                modifyIORef' i (+ 1)
          )
        readIORef r
```

Задача 8. *(2 балла)* Вычислите усреднённый по сериям модуль отклонения количества орлов от своего теоретического среднего значения в серии (предполагается, что эта величина известна и равна половине длины серии n). Используйте глобальный системный генератор случайных чисел.

```
avgdev :: Int -> Int -> IO Double
avgdev k n = undefined
```

Критерии оценивания и шкала оценки домашнего задания №3.

Оценка	Критерии выставления оценки
«Отлично» (7-13)	Программа компилируется и выполняется без ошибок. Логика программы полностью соответствует поставленной задаче, нет семантических ошибок. Код грамотно структурирован. Переменные и функции имеют осмысленные, корректные имена. Соблюдается единая стилистика. Присутствуют минимально необходимые комментарии для пояснения не очевидных решений.
«Хорошо» (4-6)	Программа работает, но содержит незначительные логические недочёты. Структура кода в целом понятна, но есть небольшие нарушения. Имена переменных/функций в основном адекватны, но встречаются не информативные варианты. Отсутствует общая стилистика кода. Комментарии недостаточны или избыточны.
«Удовлетворительно» (2-3)	Программа компилируется, но содержит серьёзные логические ошибки. Код плохо структурирован. Имена переменных/функций не информативны. Стиль не соблюдается (хаотичные отступы, смешение стилей). Комментарии отсутствуют или не соответствуют коду.

«Неудовлетворительно» (0-1)	Программа не компилируется или не запускается. Критические логические ошибки, делающие решение нерабочим. Полное отсутствие структуры и читаемости. Имена переменных/функций не несут смысла. Грубые нарушения стиля.

ДОМАШНЕЕ ЗАДАНИЕ №4

Задача 1. *(1 балл)* Реализуйте следующий набор вспомогательных функций. Функция freeVars возвращает список свободных переменных терма

```
freeVars :: Expr -> [Symb]
freeVars = undefined
```

Функция freeTVars возвращает список свободных переменных типа (в STT все переменные типа свободные)

```
freeTVars :: Type -> [Symb]
freeTVars = undefined
```

Функция extendEnv расширяет контекст переменной с заданным типом

```
extendEnv :: Env -> Symb -> Type -> Env
extendEnv = undefined
```

Функция freeTVarsEnv возвращает список свободных типовых переменных контекста

```
freeTVarsEnv :: Env -> [Symb]
freeTVarsEnv = undefined
```

Задача 2. *(1 балл)* Реализуйте функцию appEnv , позволяющую использовать контекст как частичную функцию из переменных в типы:

```
appEnv :: (MonadError String m) => Env -> Symb -> m Type
appEnv (Env xs) v = undefined
```

Обратите внимание на механизм обработки исключений, который мы будем использовать и в дальнейшем. Информация об исключении передается в виде строки; конкретная монада-обработчик исключений не фиксируется.

Ожидаемое поведение:

```
GHCi> let tx = (TVar "a" :-> TVar "b") :-> TVar "c"
GHCi> let ty = TVar "a" :-> TVar "b"
GHCi> let env = Env [("y", ty), ("x", tx)]
GHCi> let Right res = appEnv env "x" in res
(TVar "a" :-> TVar "b") :-> TVar "c"
GHCi> let Right res = appEnv env "y" in res
TVar "a" :-> TVar "b"
GHCi> let Left res = appEnv env "z" in res
"There is no variable \"z\" in the enviroment."
```

Задача 3. *(1 балл)* Реализуйте функции, осуществляющие подстановку типов вместо переменных типа в тип (appSubsTy) и подстановку типов вместо переменных типа в контекст (appSubsEnv):

```
appSubsTy :: SubsTy -> Type -> Type
appSubsTy = undefined

appSubsEnv :: SubsTy -> Env -> Env
appSubsEnv = undefined
```

Задача 4. *(2 балла)* Реализуйте функцию, выполняющую композицию двух подстановок (носитель композиции является объединением носителей двух этих подстановок):

```
composeSubsTy :: SubsTy -> SubsTy -> SubsTy
composeSubsTy = undefined
```

Используя эту функцию сделайте тип SubsTy представителем класса типов Monoid.

Задача 5. *(2 балла)* Реализуйте алгоритм унификации, возвращающий для двух переданных типов наиболее общий унификатор или сообщение об ошибке, если унификация невозможна.

```
unify :: (MonadError String m) => Type -> Type -> m SubsTy
unify = undefined
```

Ожидаемое поведение:

```
GHCi> let Right sbs = unify (TVar "a" :-> TVar "b") (TVar "c" :-> TVar "d") in sbs
SubsTy [("a", TVar "c"),("b", TVar "d")]
GHCi> let Left tst = unify (TVar "a") (TVar "a" :-> TVar "a") in tst
"Can't unify (TVar \"a\") with (TVar \"a\" :-> TVar \"a\")!"
```

Критерии оценивания и шкала оценки домашнего задания №4.

Оценка	Критерии выставления оценки
«Отлично» (5-7)	Программа компилируется и выполняется без ошибок. Логика программы полностью соответствует поставленной задаче, нет семантических ошибок. Код грамотно структурирован. Переменные и функции имеют осмысленные, корректные имена. Соблюдается единая стилистика. Присутствуют минимально необходимые комментарии для пояснения не очевидных решений.
«Хорошо» (2-4)	Программа работает, но содержит незначительные логические недочёты. Структура кода в целом понятна, но есть небольшие нарушения. Имена переменных/функций в основном адекватны, но встречаются не информативные варианты. Нарушается общая стилистика. Комментарии недостаточны или избыточны.
«Удовлетворительно» (1)	Программа компилируется, но содержит серьёзные логические ошибки. Код плохо структурирован. Имена переменных/функций не информативны. Стиль не

	соблюдается (хаотичные отступы, смешение стилей). Комментарии отсутствуют или не соответствуют коду.
«Неудовлетворительно» (0)	Программа не компилируется или не запускается. Критические логические ошибки, делающие решение нерабочим. Полное отсутствие структуры и читаемости. Имена переменных/функций не несут смысла. Грубые нарушения стиля.

20.2. Промежуточная аттестация

Промежуточная аттестация по дисциплине осуществляется с помощью следующих оценочных средств:

20.2.1. Перечень вопросов к зачету

λ-исчисление

1. Сравнение функционального и императивного подходов к программированию.
2. Основы λ-исчисления. λ-термы, свободные и связанные переменные.
3. Подстановка λ-терма. Лемма подстановки.
4. α- и β-конверсии. η-конверсия и экстенсиональная эквивалентность.
5. Расширения чистого лямбда-исчисления. δ-конверсия.
6. Кодирование булевых значений, кортежей в чистом бестиповом λ-исчислении.
7. Кодирование чисел Чёрча в чистом бестиповом λ-исчислении.
8. Теорема о неподвижной точке. Y-комбинатор.
9. Редексы. Одношаговая и многошаговая редукция. Нормальная форма. Редукционные графы.
10. Теорема Чёрча-Россера и её следствия.
11. Стратегии редукции. Теорема о нормализации. Механизмы вызова в функциональных языках.
12. Функция предшествования для чисел Чёрча. Комбинатор примитивной рекурсии.
13. Просто типизированное λ-исчисление в стиле Карри. Предтермы. Утверждения о типизации. Контексты. Правила типизации.
14. Просто типизированное λ-исчисление в стиле Чёрча. Предтермы. Утверждения о типизации. Контексты. Правила типизации.
15. Свойства систем просто типизированного λ-исчисления.
16. Связь между системами Карри и Чёрча. Проблемы разрешимости. Сильная и слабая нормализация.
17. Свойство универсальности правой свертки. Закон слияния foldr-map.
18. Понятие главного (наиболее общего) типа. Подстановки типа и их композиция. Унификаторы.
19. Алгоритм унификации.
20. Алгоритм построения системы ограничений.
21. Теорема Хиндли-Милнера.
22. Рекурсивные типы. μ-нотация.
23. Катаморфизмы.
24. Анаморфизмы и гилеоморфизмы.
25. Зипперы. Контексты с дыркой.

Haskell

1. Основы программирования на Haskell. Связывание. Рекурсия. Базовые конструкции языка.
2. Основные встроенные типы языка Haskell. Система модулей. Частичное применение, каррирование.
3. Операторы и их сечения в Haskell. Бесточечный стиль.
4. Ошибки. Основание. Строгие и нестрогие функции. Ленивое и энергичное исполнение.
5. Алгебраические типы данных. Сопоставление с образцом, его семантика.
6. Объявления type и newtype . Метки полей.
7. Списки, стандартные функции для работы с ними. Генерация (выделение) списков.
8. Специальный полиморфизм. Классы типов. Объявления представителей. Классы типов Eq , Ord , Enum и Bound .
9. Внутренняя реализация классов типов.
10. Стандартные классы типов: Num и его наследники, Show и Read .
11. Моноиды. Представители класса типов Monoid .
12. Свёртки списков. Правая и левая свёртки. Энергичные версии. Развертки.
13. Класс типов Foldable и его представители.
14. Класс типов Functor и его представители.
15. Класс типов Applicative и его представители.
16. Классы типов Alternative и MonadPlus .
17. Аппликативные парсеры.
18. Класс типов Traversable и его представители.
19. Монады. Класс типов Monad . Законы для монад. do-нотация.
20. Стандартные монады: Maybe и списки.
21. Ввод-вывод в чистых языках. Монада IO . Взаимодействие с файловой системой.
22. Линзы ван Лаарховена.

Описание технологии проведения

Промежуточная аттестация предназначена для определения уровня освоения всего объема учебной дисциплины. Промежуточная аттестация по дисциплине «Типовое λ-исчисление и язык программирования Haskell» проводится в форме зачета.

Промежуточная аттестация, как правило, осуществляется в конце семестра. При проведении зачета учитываются результаты выполнения домашних работ.

К промежуточной аттестации допускаются студенты, выполнившие все, предусмотренные планом домашние работы. В случае отсутствия не более двух домашних работ, студент может быть допущен к промежуточной аттестации с добавлением двух дополнительных вопросов к типовому КИМ промежуточной аттестации.

Промежуточная аттестация проводится в формате собеседования с преподавателем. Обучающийся получает два вопроса (первый из блока «λ-исчисление», второй из блока «Haskell»). Время подготовки к ответу не должно превышать 0,75 часа. При желании, студент может начать ответ без подготовки. При необходимости, преподаватель может задавать уточняющие, а в случае отсутствия оценки по контрольным точкам дополнительные вопросы.

Требования к выполнению заданий, шкалы и критерии оценивания

На основании критериев оценивания, приведенных ниже, преподаватель выставляет обучающемуся оценку по дисциплине.

20.2.2. Критериии оценивания

Оценка	Критерии выставления оценки
«Зачтено»	Дан верный ответ на поставленный вопрос. Материал изложен последовательно или с незначительными нарушениями логики. Выводы присутствуют в явном или неявном виде; общая аргументация подтверждает правильность ответа.
«Не зачтено»	Ответ неверный, неполный или не соответствует поставленному вопросу. Материал изложен хаотично, без четкой структуры. Выводы отсутствуют или не соответствуют содержанию ответа.