**Санкт-Петербургский государственный университет**

**Р А Б О Ч А Я П Р О Г Р А М М А**

**УЧЕБНОЙ ДИСЦИПЛИНЫ**

Функциональное программирование

Functional Programming

**Язык(и) обучения**

русский

Трудоемкость в зачетных единицах: 2

Регистрационный номер рабочей программы: 003589

**Раздел 1. Характеристики учебных занятий**

**1.1. Цели и задачи учебных занятий**

Развитие современных популярных языков программирования в значительной мере основывается на заимствовании идей из других языков программирования, которые по тем или иным причинами менее распространены и популярны. Серьёзным источником новых идей и подходов (реактивное программирование, сопоставление с образцом, концепты С++, интерфейсы преобразования последовательностей данных в С#, и т.п.) являются функциональные языки программирования.

Можно выделить две классификации функциональных языков программирования.

Языки из семейства Scheme (1970-e) наследуют синтаксис в виде s-выражений (изначально появившийся в языке Lisp), традиционно не знамениты развитой системой типов, и полагаются на проектирование встроенных предметно-ориентированных языков, чтобы облегчать написание кода и минимизировать возможность для совершения ошибок.

Второе семейство — это языки, развивающие идеи, заложенные в язык ML (1973); например, OCaml, Haskell и Scala. Они полагаются на развитую систему статической типизации, которая предохраняет программиста от совершения некоторых классов ошибок. Хотя существуют не функциональные языки, где заявляется статическая типизация (C++, C#), компиляторы языков из семейства ML подходят к корректности более серьёзно, поэтому у начинающих обучающихся часто возникает ощущение (правда, в общем случае неправильное): «если программа компилируется, то работает как ожидалось».

В рамках учебных занятий обучающиеся познакомятся с языками второго семейства, хотя подобный курс можно организовать и с помощью языков из первого семейства. Выбор сделан из личных предпочтений составителей курса.

Функциональные языки программирования традиционно хорошо подходят для задач, где происходит символьное преобразование данных, или же ужесточены требования по корректности программ. Примерами могут быть области, где происходит символьное оперирование с данными (системы (полу-)автоматического доказательства теорем и системы-помощники в доказательстве теорем, компиляторы) или требуются гарантии корректности (различные блокчейн-платформы).

Во время курса обучающийся будет развивать навыки программирования на задачах из области компиляторных технологий: преимущественно из так называемого frontend компилятора, частично из middleend компилятора. Глубоких предварительных знаний для выполнения домашнего задания не потребуется. Выбор именно этой области основан на личных предпочтениях составителей курса.

«Функциональное программирование» является одной из важных дисциплин образовательной программы 09.03.04, предназначенной для подготовки специалиста в области информационных технологий. Она представляет собой комплекс знаний, умений и навыков, позволяющих овладеть основами программирования на функциональных языках, а также обрести навыки программирования с использованием только чистых функций. Отдельные параметры курса могут варьироваться по степени сложности в зависимости от уровня подготовки обучающихся.

Цель изучения дисциплины: ознакомление обучающихся с основными понятиями функционального программирования, получение практических навыков программирования в рамках этой парадигмы программирования.

**1.2. Требования подготовленности обучающегося к освоению содержания учебных занятий (пререквизиты)**

Программа курса предназначена для обучающихся 2 курса, освоивших программу курсов «Алгебра и геометрия» и «Основы программирования» (или аналогичных).

Максимальная эффективность программы будет обеспечена при условии, что обучающийся:

* владеет основами алгебры;
* владеет основами программирования;
* владеет навыками чтения технической документации на английском языке;
* владеет навыками работы с системами контроля версий (обусловлено способом приёмки домашнего задания).

**1.3. Перечень результатов обучения (learning outcomes)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Наименование категории (группы) компетенций | Код и наименование компетенции | Планируемые результаты обучения, обеспечивающие формирование компетенции | Код индикатора и индикатор достижения универсальной компетенции |
|  | 1 | 2 | 3 | 4 |
| 1 | Общепрофессиональные компетенции | ОПК-1 – способен применять естественнонаучные и общеинженерные знания, методы математического анализа и моделирования, теоретического и экспериментального исследования в профессиональной деятельности | Знать основы функциональной парадигмы программирования | ОПК-1.1 Уметь идентифицировать возможные проблемы и пути их решения |
| 2 | Общепрофессиональные компетенции | ОПК-2 Способен понимать принципы работы современных информационных технологий и программных средств, в том числе отечественного производства, и использовать их при решении задач профессиональной деятельности | Знать основы функциональной парадигмы программирования | ОПК-2.2 Уметь проводить формализацию и алгоритмизацию поставленных задач |
| 3 | Общепрофессиональные компетенции | ОПК-6 – способен разрабатывать алгоритмы и программы, пригодные для практического использования, применять основы информатики и программирования к проектированию, конструированию и тестированию программных продуктов | Навыки программирования на функциональном языке | ОПК-6.1 Уметь проверять и отлаживать программный код |
| 4 | Профессиональные компетенции | ПКА-1 – способен использовать в педагогической деятельности научные основы образования в сфере ИКТ | Знать основы функциональной парадигмы программирования | ПКА-1.1 Быть способным осуществлять организационное и технологическое обеспечение кодирования на языках программирования |
| 5 | Профессиональные компетенции | ПКП-1 – способен проектировать программные системы | Навыки программирования на функциональном языке | ПКП-1.1 Уметь разрабатывать технические спецификации на программные компоненты и их взаимодействие |
| 6 | Профессиональные компетенции | ПКП-2 – способен использовать основные модели информационных технологий и способы их применения для решения задач в предметных областях | Навыки программирования на функциональном языке | ПКП-2.1 Уметь описывать алгоритмы компонентов, включая методы и схемы |

**1.4. Перечень и объём активных и интерактивных форм учебных занятий**

Аудиторная учебная работа:

* лекционные занятия в объеме 2 часов в неделю;
* текущее тестирование, комплексное тестирование (экзамен) в конце семестра.

Самостоятельная работа:

* без участия преподавателя: решение семестровой задачи (или задач) для подготовки к промежуточной аттестации.

**Раздел 2. Организация, структура и содержание учебных занятий**

**2.1. Организация учебных занятий**

**2.1.1**  **Основной курс**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Трудоёмкость, объёмы учебной работы и наполняемость групп обучающихся | | | | | | | | | | | | | | | | | | |
| Код модуля в составе дисциплины,  практики и т.п. | Контактная работа обучающихся с преподавателем | | | | | | | | | | | | Самостоятельная работа | | | | Объём активных и интерактивных  форм учебных занятий | Трудоёмкость |
| лекции | семинары | консультации | практические  занятия | лабораторные работы | контрольные работы | коллоквиумы | текущий контроль | промежуточная  аттестация | итоговая аттестация | под руководством преподавателя | в присутствии  преподавателя | сам. раб. с использованием  методических материалов | текущий контроль (сам.раб.) | промежуточная аттестация (сам.раб.) | итоговая аттестация  (сам.раб.) |
| ОСНОВНАЯ ТРАЕКТОРИЯ | | | | | | | | | | | | | | | | | | |
| Форма обучения: очная | | | | | | | | | | | | | | | | | | |
| Семестр 3 | 30 |  | 2 |  |  |  |  |  | 2 |  |  |  | 18 |  | 20 |  | 10 | 2 |
|  | 2-42 |  | 2-25 |  |  |  |  |  | 2-25 |  |  |  | 1-1 |  | 1-1 |  |  |  |
| ИТОГО | 30 |  | 2 |  |  |  |  |  | 2 |  |  |  | 18 |  | 20 |  |  | 2 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Виды, формы и сроки текущего контроля успеваемости и промежуточной аттестации | | | | | | |
| Код модуля в составе дисциплины, практики и т.п. | Формы текущего контроля успеваемости | | Виды промежуточной аттестации | | Виды итоговой аттестации  (только для программ итоговой аттестации и дополнительных образовательных программ) | |
| Формы | Сроки | Виды | Сроки | Виды | Сроки |
| ОСНОВНАЯ ТРАЕКТОРИЯ | | | | | | |
| Форма обучения очная | | | | | | |
| Семестр 3 |  |  | экзамен, устно, традиционная форма | по графику промежуточной аттестации |  |  |

**2.2. Структура и содержание учебных занятий**

**Основной курс Основная траектория Очная форма обучения**

Период обучения (модуль): Семестр 3

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Наименование темы (раздела, части) | Вид учебных занятий | Количество часов |
| I. | Введение. Особенности синтаксиса и философия парадигмы | лекции | 2 |
| II. | Проектирование типов в функциональном программировании. Алгебраические типы данных. Изоморфизм типов. Полукольцо типов. Дифференцирование типов. Структура данных «зиппер» (zipper) | лекции | 3 |
| III. | Лямбда-исчисление. Различные стратегии и интерпретаторы | лекции | 2 |
| IV. | Различные виды интерпретаторов лямбда-исчисления. Неподвижные точки | лекции | 2 |
| V. | Парсер-комбинаторы. Синтаксический анализ с их помощью | лекции | 2 |
| VI. | Ленивые последовательности. Программирование в стиле оригами. Введение в схемы рекурсии | лекции | 4 |
| VII. | Вывод типа по телу функции. Унификация | лекции | 2 |
| VIII. | Чисто функциональные структуры данных | лекции | 2 |
| IX. | Функторы. Монады. Законы. Примеры. Do-нотация | лекции | 2 |
| X. | Краткий обзор теоретических аспектов: α, β, η - правила в лямбда-исчислении. STLC. Система Hindley-Milner-Damas. Let-полиморфизм | лекции | 2 |
| XII. | Введение в трансформеры монад | лекции | 4 |
| XIII. | Равенство и эквивалентность типов. GADT и создание DSL. Initial & final embedding. Expression problem. | лекции | 3 |
| XIV. | Промежуточная аттестация | самостоятельная работа | 20 |
| консультация | 2 |
| экзамен | 2 |

Ниже изложен примерный план проведения занятий. Изучение тем II-V окажет наибольшую помощь обучающемуся, без знания данных тем обучающийся скорее всего испытает непреодолимые проблемы при практической работе в рамках данного курса.

Остальные темы являются теоретическими в большей степени, чем темы II-IV. Изучение этих тем не является строго необходимым для выполнения самостоятельной работы, но использование полученных знаний при практической работе настоятельно рекомендуется. Преподаватель оставляет за собой право попросить обучающегося применить знания по некоторой теме в своей практической работе, ради проверки усвоения конкретной темы. Обучающимся стоит быть готовым произвести изменения в своей практической работе даже в день аттестации. Преподавателю рекомендуется заявлять свои просьбы до проведения аттестации, чтобы у обучающегося было достаточно времени на исправления (хотя в реальности осуществление таких пожеланий может быть затруднительно: неудобные даты пересдач, болезни той или иной стороны, и т.п.).

Раздел XIII не является обязательным и введён, если по тем или иным причинам в курсе остаётся дополнительное время. На усмотрение преподавателя в нём могут фигурировать и иные темы из области функционального программирования.

1. **Введение**. Особенности синтаксиса и философия парадигмы.

В данном разделе дается общее видение разновидностей языков функционального программирования, история эволюции, их положение среди множества других языков и кратко основные особенности, сильные и слабые стороны. Данная лекция является не очень технической и легко может быть передвинута позже по календарю занятий.

1. **Проектирование типов.** Далее обучающиеся знакомятся с основным средством для проектирования пользовательских типов данных – алгебраическими типами данных – и с сопоставлением с образцом. Также вводится понятие абстрактного синтаксического дерева и показываются примеры того, как это связано с проектированием типов данных. Сразу же после этого можно рассказывать про структуру данных «зиппер», или же отложить изложение этого материала на более поздний срок ради более быстрого изложения необходимого минимума материала для эффективного освоение практического материала (домашнего задания). Также в этом разделе объясняется метод подсчета количества “жителей” типа.
2. **Лямбда-исчисление. Различные стратегии и интерпретаторы. Различные виды интерпретаторов лямбда-исчисления. Неподвижные точки.**

Вводится синтаксис лямбда-исчисления и стратегии редукции лямбда-термов. Рекомендуется изложить минимум два стратегии: вызов по значению (call-by-name) и вызов по необходимости (call-by-name и call-by-needed). Желательно также коснуться аппликативной (applicative order) и нормальной стратегии (normal order).  
В качестве примеров лямбда-выражений рекомендуется использовать необходимый конструкции для демонстрации того, что лямбда-исчисление – это универсальный вычислитель. В минимум таких конструкций входят: представления чисел, ветвлений, булевых констант, а также способ организации неограниченной рекурсии (так называемые Y-комбинатор и Z-комбинатор). Также рекомендуется объяснить проблему, которая обычно решается с помощью “подстановки без захвата переменных” (capture avoiding substituition), индексов или уровней де Брёйна.

1. **Парсер-комбинаторы. Синтаксический анализ с их помощью.**

На парсер-комбинаторах демонстрируется комбинаторный подход к проектированию интерфейсов (другими примерами комбинаторного подхода могут быть принтер-комбинаторы или matching-комбинаторы). Рекомендуется показать проектирование парсер-комбинаторной библиотеки с нуля восходящим образом, чтобы пояснить обучающимся как синтаксический анализ проводится в деталях. Во время практической работы обучающиеся могут либо пользоваться готовой библиотекой парсер-комбинаторов, либо реализовать свои, при условии, что имена и типы основных комбинаторов будут взяты из какой-нибудь достаточно популярной библиотеки парсер-комбинаторов.

Примерами языков, на которых будет демонстрироваться работа парсер комбинаторов, рекомендуется взять язык арифметических выражений с приоритетами операций (так как он достаточно простой), а также язык лямбда-термов.  
Изложение тем «парсер-комбинаторы» и «проектирование типов» достаточно для того, чтобы обучающиеся могли содержательно начать практическую работу.

1. **Ленивые последовательности. Программирование в стиле оригами. Введение в схемы рекурсии.**

Ленивые последовательности являются подходом в функциональном программировании, который наиболее часто перенимается нефункциональными языками программирования (например, интерфейс IEnumerable языка C#). Необходимо показать стандартные операции со списками, универсальные свойства этих операций, и причины, из-за которых ленивые последовательности могут быть эффективнее строгих. Показать, как можно реализовать сортировки списков с помощью операций fold&unfold («свернуть» и «развернуть», отсюда название «программирование в стиле оригами») на основе статьи «Программирование в стиле оригами» (Origami programming, J.Gibbons, 2004).

1. **Вывод типа по телу функции. Унификация.**

Автоматический вывод типов является отличительной особенностью языков семейства ML. Здесь необходимо пояснить суть алгоритма синтаксической унификации и показать, как он применяется при выводе типов компилятором в языках семейства ML. На аттестации обучающийся должен быть готов продемонстрировать свои навыки вывода типа функции по её реализации с помощью ручки и бумаги (т.е. без использования электронных помощников).

1. **Чисто функциональные структуры данных.**

Обучающийся знакомится с подходом к проектированию чисто функциональных структур данных, то есть без использования присваивания. Примерами таких структур данных стоит выбрать списки (так как они достаточно просты) и самобалансирующиеся красно-черные деревья (так как их реализация в чисто функциональном подходе выглядит существенно проще и короче, чем при традиционных подходах с присваиванием). Основной литературой по данному вопросу является книга К.Окасаки «Чисто функциональные структуры данных». При наличии дополнительного лекционного времени рекомендуется рассказать обучающемуся про другие чисто функциональные структуры данных из вышеупомянутой книги.

1. **Функторы. Монады. Законы. Примеры. Do-нотация.**

Обучающийся знакомится с понятием функтора (например, упомянутые выше списки и ленивые последовательности) и монады, а также с законами функторов и монад. Законы функторов должны выглядеть знакомыми обучающемуся, так как они неявно использовались в рамках темы V. Обучающийся знакомится с законами монад, изучает основные примеры монад: Reader, Writer, State, List, Cont, а также их применение при написании интерпретатора лябмда-исчисления со сквозной функциональностью (с так называемыми «эффектами»).

После изучения данной темы обучающийся должен:

* 1. Уметь описывать свои типы данных, так, чтобы они были функторами.
  2. Уметь применять функторы и монады из списка выше при практической работе (уметь описывать новые монады не требуется).

1. **Краткий обзор теоретических аспектов: α, β, η – правила в лямбда-исчислении. STLC. Система Hindley-Milner-Damas. Let-полиморфизм.**

Здесь читатель кратко знакомится с наиболее известными системами типов для лямбда-исчисления и приобретает более детальное понимание того, как компилятор проводит автоматический вывод типов. Рекомендованной литературой является книга Бенджамина Пирса «Типы в языках программирования».

1. **Введение в трансформеры монад.**

Программирование с использование только чистых функций является не очень удобным без использования монад, а использование нескольких монад одновременно не очень удобно без использования трансформеров монад (monad transformers). В теме VIII законы монад постулировались как необходимое требование, полезность которого не очевидна, а в рамках данной темы обучающийся сможет увидеть как законы монад можно применить для написания более эффективного кода на функциональном языке программирования. Рекомендуется строить изложение материала на статье Ральфа Хинзе «Deriving backtracking monad transformers» 2000 года.

1. **Равенство и эквивалентность типов. GADT и создание DSL. Initial & final embedding. Expression problem. Либо запасные часы лекций.**

Данную тему рекомендуется спрашивать только у обучающихся, которые претендуют на максимальную итоговую оценку. В ней рекомендуется объяснить способ описания термов лямбда-исчисления, которые типизируются в системе типов STLC и корректны по построению. В случае нехватки времени стоит ограничиться объяснением типа данные Eq, и его связью с отношением равенства (типов) из алгебры. В случае сильной нехватки времени предусмотренные часы лекций стоит уделить закреплению предыдущего материала.

**Раздел 3. Обеспечение учебных занятий**

**3.1. Методическое обеспечение**

**3.1.1** **Методические указания по освоению дисциплины**

Занятия проходят в форме лекций. На лекционных занятиях преподаватель рассказывает материал курса согласно содержанию в разбивке по разделам, приведенному в разделе 2.2. Изложение материала в целом классическое, согласно таковому в источниках, приведенных в обязательном и дополнительном списках литературы.

Успешное освоение дисциплины возможно благодаря посещению занятий, участию в обсуждении вопросов, подготовленных к занятию; самостоятельной работе, включающей в себя чтение специальной литературы по разделам темы и выполнению практических заданий.

Изложение лекционного материала (как и выполнение практических занятий) может происходить с использованием различных статически типизированных языков функционального программирования. Ниже список наиболее известных из них с комментариями. Выбор языка для курса должен осуществляться с оглядкой на компетенцию преподавателя.

1. Язык Haskell, возможно, является самым распространенным языком для преподавания функционального программирования. Он отличается наличием большого количества интересных возможностей в нём, а также самым распространенным кандидатом для добавления в него новых интересных синтаксических конструкций, апробации способов программирования и проектирования, совершения экспериментов с новыми разновидностями систем типов. Не смотря на перечисленные плюсы для практического программирования, при обучении описанные выше возможности могут вызываться сложности, например, потому что для детального понимания сути относительно простых конструкций (например, вывод данных на печать или арифметические операции с числами) обучающемуся необходимо иметь представление о некоторых (достаточно абстрактных) концепциях, присущих языку программирования Haskell (монады и классы типов, соответственно). Несмотря на выше сказанное, Haskell *является допустимым кандидатом* на функциональный язык программирования, на основе которого будет излагаться курс.
2. Язык OCaml — это ещё один распространенный вариант для преподавания функционального программирования. В нём меньше продвинутых возможностей, некоторые вещи приходится реализовывать более явным и длинным способом, чем в Haskell. Однако эти недостатки делают его более удачным языком для обучения, чем Haskell, так как

* объем необходимым предварительных знаний для него ниже;
* некоторые вещи приходится реализовывать явно, что в большей степени требует от обучающегося понимания, что он делает;
* при проектировании синтаксиса языка были заложены некоторые особенности, облегчающие на него переход, после использования языка Си.

Также у языка OCaml есть некоторое количество диалектов:

* Bucklescript/ReScript позволяет компилировать программы для исполнения в вэб-среде, в то время как по умолчанию OCaml компилируется в платформо-зависимый код. Возможность осуществлять практическую часть работы, тестируя свой код в веб-браузере, может показать приближенность ReScript к индустрии, а также может вызывать большую вовлеченность обучающихся, которые уже интересуются вэб-программированием.
* ReasonML является разновидностью ReScript, где синтаксис языка OCaml несколько изменен, чтобы быть более похожим на современный Javascript. При этом все синтаксические конструкции OCaml можно автоматически преобразовывать в ReasonML и наоборот. Это изменение в синтаксисе, с одной стороны, может благоприятно отразиться на обучении, так как если обучающийся уже знаком с языками программирования с Си-подобным синтаксисом, то ему будет проще освоить ReasonML. С другой стороны, в измененном синтаксисе некоторые конструкции функционального программирования могут потерять часть своей привлекательности.

По мнению авторов РПД OCaml *является рекомендованным языком* для изложения лекционного материала. Использование его диалектов во время лекций или практических занятий отдается на усмотрение и компетенцию преподавателя.

1. Составители РПД рекомендует *использование языка F# в крайнем случае*.
2. Использования языка *Scala 2 не рекомендуется*, Scala 3 – в крайнем случае.

В списке выше не представляется возможным охватить все возможные кандидаты на основной язык преподавания, поэтому составители программы выбрали и перечислили наиболее вероятные кандидаты. Также не перечислены наиболее подходящие кандидаты из языков семейства Scheme, потому что при составлении РПД подразумевались статически типизированные языки функционального программирования.

Составители программы настоятельно не рекомендуют использование языков программирования, где основная парадигма не является функциональной, так как такое изложение материала может существенно исказить основные концепции.

**3.1.2 Методическое обеспечение самостоятельной работы**

При самостоятельном изучении теоретического материала и решении семестровой задачи целесообразно использовать рекомендованную литературу. Дополнительно по согласованию с преподавателем обучающийся может осваивать теоретическую часть курса по одному или нескольким из онлайн-курсов, представленных на образовательных платформах в п. 3.4.3. При этом преподаватель обязан сообщить обучающимся, какие разделы выбранных ими онлайн-курсов недостаточно полно раскрывают ту или иную тему курса и порекомендовать дополнительные источники по данной теме. Преподаватель вправе отказать обучающемуся в самостоятельном освоении теоретической части дисциплины по онлайн-курсам в случае нахождения в нем существенных расхождений с содержанием курса в разделе 2 и п. 3.1.1.

В качестве самостоятельной работы в данном РПД подразумевается реализация на утвержденном функциональном языке программирования мини-проекта на тему компиляторов. Глубоких знаний об устройстве компиляторов обучающемуся не требуется. Минимальные знания “что такое компилятор и зачем они нужны?” обучающийся должен был получить после прохождения курса “Основы программирования”. Необходимую информацию о синтаксическом анализе он должен получить в лекции “Парсер-комбинаторы”. Порядок выполнения заданий, связанных с middle-end компилятора, должен быть понятен непосредственно из описания задания. Написание интерпретатора языка программирования рассказывается в лекциях на примере лямбда-исчисления.

В качестве мини-проекта предлагается использовать реализацию упрощенной версии некоторого языка программирования. Подробнее про это написано в разделе 3.1.4.2.

**3.1.3** **Методика проведения текущего контроля успеваемости и промежуточной аттестации и критерии оценивания**

В ходе самостоятельной работы обучающийся получает баллы, выполняя задания, которые в течение семестра оцениваются преподавателем. За что именно начисляются баллы указано в разделе 3.1.4.2. К аттестации баллы позволяют оценить итоговый процент освоения курса с помощью следующей таблицы:

|  |  |
| --- | --- |
| Баллы | Итоговый процент освоения курса |
| 0-4 | 0-49% |
| 5-6 | 50-59% |
| 7 | 60-69% |
| 8-10 | 70-79% |
| 11-12 | 80-89% |
| ≥13 | 90-100% |

Самостоятельная работа предоставляется на оценку преподавателю (или его ассистентам) заранее, т.е. за некоторое время до промежуточной аттестации. Преподаватель оставляет за собой право установить в начале семестра крайний срок подачи результатов самостоятельной работы, например, три недели до аттестации.

Самостоятельную работу рекомендуется подавать в электронном виде с помощью выбранной системы контроля версий, основы работы с которыми обучающийся уже освоил в курсе “Основы программирования” (или аналогичным ему). Предоставляемые решения (пулл-реквесты) должны компилироваться; быть добавляемыми в основной репозиторий, т.е. слияние (англ. merging) должно проводиться без ошибок; решения не должны быть списанными с другого обучающегося этого или предыдущих годов обучения. Если какой-то из этих пунктов не выполнен, то преподаватель вправе не проверять задание и отклонить решение.

Преподаватель в праве выбрать стиль кодирования для всех обучающихся группы и требовать его соблюдения при самостоятельной работе. Преподаватель в праве установить список разрешенных (или запрещенных) библиотек языка, которые можно использовать при выполнении практической работы.

При проведении аттестации обучающийся:

* беседует с преподавателем, отвечая на теоретические вопросы, чтобы показать усвоение лекционных знаний;
* отвечает на вопросы по реализации его самостоятельной работы, чтобы показать, что её выполнял он, а не кто-либо ещё;
* решает простые 5-минутные задачи по теме курса для демонстрации своих навыков.

Во время беседы с преподавателем, обучающийся поясняет содержимое некоторых (возможно, всех) теоретических вопросов, заготовленных к аттестации. Выдачи билетов и времени на подготовку не предусмотрено.

Также преподаватель может потребовать проведения изменений в реализации самостоятельной работы для демонстрации усвоения курса. Например, это может быть “переписывание некоторой функции из реализации самостоятельной работы с использованием монады State”. Обучающийся должны быть готов провести изменения на месте, преподавателю рекомендуется требовать такого рода изменения заранее, в течение семестра.

К аттестации приготовлен отдельный вопрос, неудачный ответ на который, обнуляет все набранные баллы.

Ответы обучающегося на аттестации оцениваются следующим способом:

1. Ответ правильный, полный и получен без помощи преподавателя.
2. Правильный (или полный) получен с минимальной помощью преподавателя.
3. Иное.

Если все ответы обучающегося можно оценить первым и вторым способом, то считается, что обучающийся подтвердил набранные баллы. В ином случае обучающийся не подтвердил баллы, набранные за самостоятельную работу.

В случае подтверждения, обучающийся получает оценку ECTS, соответствующую тому количеству баллов, сколько он набрал за самостоятельное практическое задание. В случае неуспешного подтверждения обучающийся получает оценку ECTS на ранг ниже, чем та, на которую он претендовал по результатам самостоятельной работы.

Перевод итогового процента освоения курса в оценки делается по стандартной методике согласно приказу №7293/1 от 20.07.2018:

|  |  |  |
| --- | --- | --- |
| Итоговый процент освоения курса, % | Оценка СПбГУ при проведении зачёта | Оценка ECTS |
| 0-49 | Не зачтено | F |
| 50-60 | Зачтено | E |
| 61-69 | Зачтено | D |
| 70-79 | Зачтено | C |
| 80-89 | Зачтено | B |
| 90-100 | Зачтено | A |

По желанию преподавателя на аттестацию допустимо приглашать других преподавателей с квалификацией не ниже, изложенной в разделе 3.2.1, как для независимого оценивания ответов обучающихся, так и для коллегиального. В последнем случае оценка за зачет ставится на основании голосования простого большинства. В спорных ситуациях преподаватель, ведущий дисциплину, имеет право принятия окончательного решения.

**3.1.4 Методические материалы для проведения текущего контроля успеваемости и промежуточной аттестации (контрольно-измерительные материалы, оценочные средства)**

#### **3.1.4.1** **Формируемые дисциплиной компетенции**

|  |  |  |
| --- | --- | --- |
| № | Код индикатора и индикатор достижения универсальной компетенции | Контрольно-измерительные материалы (КИМ) (тестовые вопросы, контрольные задания, кейсы и пр.) |
|  | 1 | 2 |
| 1 | ОПК-1.1 Уметь идентифицировать возможные проблемы и пути их решения | Шкала оценивания: кусочно-линейная, вычисляется по формуле MAX(0, (N – 0.6)) \* 2.5 \* 100%, где N – доля успешно выполненных заданий, проверяющих данную компетенцию, от 0 до 1 |
| 2 | ОПК-2.2 Уметь проводить формализацию и алгоритмизацию поставленных задач | Шкала оценивания: кусочно-линейная, вычисляется по формуле MAX(0, (N – 0.6)) \* 2.5 \* 100%, где N – доля успешно выполненных заданий, проверяющих данную компетенцию, от 0 до 1 |
| 3 | ОПК-6.1 Уметь проверять и отлаживать программный код | Шкала оценивания: кусочно-линейная, вычисляется по формуле MAX(0, (N – 0.6)) \* 2.5 \* 100%, где N – доля успешно выполненных заданий, проверяющих данную компетенцию, от 0 до 1 |
| 4 | ПКА-1.1 Быть способным осуществлять организационное и технологическое обеспечение кодирования на языках программирования | Шкала оценивания: кусочно-линейная, вычисляется по формуле MAX(0, (N – 0.6)) \* 2.5 \* 100%, где N – доля успешно выполненных заданий, проверяющих данную компетенцию, от 0 до 1 |
| 5 | ПКП-1.1 Уметь разрабатывать технические спецификации на программные компоненты и их взаимодействие | Шкала оценивания: кусочно-линейная, вычисляется по формуле MAX(0, (N – 0.6)) \* 2.5 \* 100%, где N – доля успешно выполненных заданий, проверяющих данную компетенцию, от 0 до 1 |
| 6 | ПКП-2.1 Уметь описывать алгоритмы компонентов, включая методы и схемы | Шкала оценивания: кусочно-линейная, вычисляется по формуле MAX(0, (N – 0.6)) \* 2.5 \* 100%, где N – доля успешно выполненных заданий, проверяющих данную компетенцию, от 0 до 1 |

#### **3.1.4.2** **Контрольно-измерительные материалы (примеры)**

Примерный список теоретических вопросов для финальной аттестации.

1. Функции в программировании и функции в математике. Сходства и отличия. Понятие чистой функции.
2. Алгебра типов. Простейшие типы. Как из простейших типов построить что-то изоморфное данному алгебраическому типу? Zippers для списков и различных видов двоичных деревьев.
3. Эквивалентность, равенство и изоморфизм вообще и применительно к типам.
4. Понятие мемоизации. Пример: эффективное вычисление чисел Фибоначчи.
5. Continuation passing style. Преобразование фукнкций из дефотлного (direct) стиля в CPS
6. Понятие исчисления. Аксиомы, правила вывода, посылки и заключения. Доказательства.
7. Лямбда-исчисление. Три правила (α, β, η) преобразования лямбда-термов.
8. Лямбда исчисление как универсальный язык программирования. Числа Чёрча, арифметика, ветвления. Проблема останова.
9. Стратегии вычисления лямбда-термов: вызовы по имени (CBN), значению (CBV), необходимости; нормальная (NO) и аппликативная стратегии (AO). Достоинства и недостатки.
10. Написания рекурсивных функций без использования рекурсии. Y- и Z- комбинаторы. Пример
11. Унификация и подстановки. Occurs check. Capture avoiding substitution. Индексы де Брёйна.
12. Стандартные функции над списками. Универсальные свойства.
13. PFDS. Понятие неизменяемых и устойчивых (persistent) типов данных.
14. PFDS. Чисто функциональные очереди. Простой вариант и оптимизированный (извлечение за амортизированную константу)
15. PFDS. Понятие префиксных деревьев и HAMT.
16. Монады. Законы монад. Стандартные монады: List, LazyList, Maybe, Parser.
17. Монады. Законы монад. Выражение функторов, если монада уже описана. Стандартные монады: Reader, Writer, State, List, Maybe, Cont, IO.
18. Deep и shallow embedding. Понятия трансформеров монад. Законы. Прокомментировать реализацию.
19. Схемы рекурсии. На примере списков и деревьев. Ката- и анаморфизм.
20. Схемы рекурсии. Хиломорфизм. Решения задач: фибоначчи, binary partition, LCS.

Теоретические вопросы проверяют компетенции ОПК-1, ПКА-1 и ПКП-2.

Отдельный вопрос, который рекомендуется задавать всем (компетенции ОПК-1, ПКА-1, ПКП-2):

“С использованием только лямбда-исчисления и некоторой стратегии из лекций реализуйте функцию вычисления факториала n (или вычисления n-го числа Фибоначчи). С помощью только ручки и бумаги протрассируйте вычисления факториала трёх (или 4го числа Фибоначчи).”

Критерии оценивания были приведены в разделе 3.1.3.

Короткие задачи на 5 минут проверяют компетенции ОПК-2, ОПК-6 и ПКП-1. Критерии оценивания такие же как для теоретических вопросов (были перечислены в разделе 3.1.3). Примеры:

1. Узнать монаду по реализации, или объяснить, почему то, что указано, не является монадой.
2. Построение ленивой последовательности из чисел Фибоначчи.
3. Реализовать синтаксический анализатор языка a^nb^nc^n.
4. Реализуйте сортировку иммутабельного списка, используя идеи так называемой “быстрой сортировки”.
5. По типу полиморфной функции (например, a->b->a) оцените количество различных его “жителей”.
6. Перепишите некоторое лямбда-выражение с использованием индексов де Брёйна.

Ниже следуют задачи, рекомендованные для проверки остаточных знаний в ФОС ООП. Критерии оценивания: по шкале от 0 (решение отсутствует или обладает существенными недостатками) до 1 (решение существенными недостатками не обладает).

1. Протрассируйте применение комбинатора Ω=(λx.xx) к самому себе с использованием стратегии вычислений CBN. Объясните результат.  
   Проверяемые компетенции: ПКА-1.  
   Варианты ответов:
   1. Ω Ω
   2. Ω
   3. Y
   4. Вычисление не закончится
2. Реализуйте сортировку (неизменяемого) списка (или ленивой последовательности) с использованием рекурсии, операции фильтрации и конкатенации списков, использую идеи так называемой “быстрой сортировки” (в качестве pivot можно взять головной элемент).   
   Проверяемые компетенции: ПКП-2, ОПК-6.
3. Поясните, как факт поставки вместе c компилятором статически типизированного компилирующегося языка функционального программирования позволяет использовать данный язык в скриптах.  
   Проверяемые компетенции: ПКП-2, ОПК-2.
4. По типу полиморфной функции (например, a\*a->a\*a) посчитайте количество различных его “жителей”.  
   Проверяемые компетенции: ОПК-1.  
   Варианты ответов:
   1. 0
   2. 1
   3. 2
   4. 4

Самостоятельная работа проверяет компетенции ОПК-1, ОПК-2, ОПК-6, ПКП-1, ПКП-2.

Перед началом самостоятельной работы обучающийся должен выбрать не более одного мини-языка для выполнения мини-проекта (по усмотрению преподавателя выдаваемые языки могут повторяться). Такими языками могут быть упрощенные варианты

1. С#: основные конструкции императивного программирования + объявление классов и методов без наследования + заранее заготовленные шаблонные (generics) классы (например, массив и список).
2. C#: как в предыдущем пункте, но шаблонные классы заменяются на поддержку работы с исключениями.
3. С#: основные конструкции + классы с публичным наследованием.
4. OCaml: основные конструкции + встроенные типы данных (список, пары).
5. Haskell: аналогично OCaml.
6. Gallina: аналогично OCaml.
7. Java: аналогично C#.
8. Python: основные конструкции + добавление в объекты методов во время исполнения.
9. Ruby: аналогично Python.
10. Lua: основные конструкции + массивы (комбинирование обычных и ассоциативных).
11. Assembler: минимальный набор простых операций + поддержка регистров + SIMD операции.
12. Скриптовые языки: bash, awk.
13. Scheme-подобные языки: основные конструкции + различные усложнения (delimited continuations, call/cc или т.п.)
14. Си: основные конструкции + указатели.
15. Forth: стековое программирование.
16. Javascript: основные конструкции с поддержкой интересных примеров неявного преобразования типов данных.
17. SQL: стандартные операции ADD/DELETE/CREATE/SELECT/JOIN + хранимые процедуры на уровне факториала.

Этот список допустимо дополнять и изменять в зависимости от подготовки группы и количества обучающихся.

Детали задания по реализации каждого мини-языка будут различаться по причине того, что языки отличаются друг от друга, и некоторые требования в одном языке могут быть не вполне адекватны при реализации другого мини-языка. Из-за большого количества мини-языков полное перечисление особенностей и требований ко всем мини-языкам не представляется возможным и адекватным в рамках данного документа. Однако, мы разделим задания для самостоятельной работы на основные части и опишем требования по ним.

|  |  |  |
| --- | --- | --- |
| №п/п | Название | Баллы |
| 1 | Абстрактное синтаксическое дерево | 0 (зачет/незачет) |
| 2 | Синтаксический анализатор (парсер) | 1-2 |
| 3 | Интерпретатор | 4-5 |
| 4 | Тесты | 2 |
| 5 | Стандартная библиотека | 1 |
| 6 | REPL | 1 |
| 7 | Pretty-printer | 1 |
| 8 | Трансформации программ | 2-4 |
| 9 | Компиляция в представление какой-либо платформы | 4 |
|  | Итого | 0-20 |

Части самостоятельной работы можно оценивать независимо. В случае адекватного выполнения частей самостоятельной работы обучающийся получает баллы. Способ пересчета условных баллов в финальную оценку за предмет “Функциональное программирование” представлены в следующем разделе документа.

Преподаватель вправе варьировать количество баллов в таблице от мини-языка к мини-языку, однако при выборе мини-языка обучающемуся необходимо сообщить распределение баллов по нему. Например, программы на языке Forth записываются в обратной польской нотации (за исключением нескольких крайних случаев), что делает задачу синтаксического анализа вырожденной. Простота синтаксиса языков, родственных Scheme, делают задачу синтаксического анализа почти вырожденной.

Далее мы более подробно опишем составные части самостоятельной работы и стандартные недочеты по ним. Преподаватель в праве отправить задание на доделку при наличии, по его мнению, существенных недочетов. Или же преподаватель может зачесть задачу в присутствии недочётов, если обучающийся сможет предъявить достаточно вескую причину, почему надо делать так, а не иначе (вескость оценивается по усмотрению преподавателя).

1. **Абстрактное синтаксическое дерево** (AST) является необходимым пререквизитом для выполнения всех остальных заданий и поэтому за данное задание не назначаются баллы. От обучающегося требуется описать синтаксис своего мини-языка с помощью типов данных.   
   При оценки данного задания проверяющему (а также обучающемуся) стоит обратить на следующие стандартные ошибки и недочёты, которые могут привести к не зачтению данного задания.
   1. AST не должно позволять представить большее разнообразие синтаксиса, чем следует. Например, для конструкции языка C# new Classname(args), узел типа данных для AST должен содержать имя и набор выражений, а не выражение и набор выражений, так как последнее позволит типизированно представить некоторые AST, которые для С# не имеют смысла.
   2. Одна и та же программа в AST должна представляться единственным способом. Иначе обучающемуся придется несколько раз реализовывать одинаковую функциональность при написании интерпретатора мини-языка.
   3. К конструкторам (алгебраических и не только) типов в AST должен прилагаться комментарий с пояснением какую конкретную конструкцию мини-языка они обозначают. (Примеры можно найти в коде компилятора OCaml)
   4. В AST желательно пользоваться синонимами типов для более удобного понимания AST. Например, для имен переменных желательно завести тип “имя” который будет синонимом типа “строка”, вместо явного использования типа “строка” в AST.
2. При реализации **синтаксического анализатора (парсера)** обучающийся может использовать готовую библиотеку парсер-комбинаторов в рамках языка реализации, или же реализовать парсер-комбинаторы самостоятельно. В последнем случае, обучающемуся необходимо посмотреть на интерфейсы уже имеющихся библиотек парсер-комбинаторов и соблюдать именование парсер-комбинаторов и их типы. Без этого обучающийся может реализовать что-то, только отдаленно напоминающее парсер-комбинаторы, и не получить за это баллы.
3. **Интерпретатор** является самым объемной частью самостоятельной работы, и поэтому он награждается большим количеством баллов.   
   Здесь сложно перечислить какие-то виды явных ошибок, скажем только что интерпретатор должен быть для выбранного мини-языка, а не какого-то другого. В интерпретаторе должны поддерживаться все требуемые возможности мини-языка. В противном случае обучающийся рискует получить за интерпретатор не максимальные баллы. Для проверки корректности мини-интерпретатора рекомендуется сравнивать результат его выполнения с результатом полноценного интерпретатора: если мини-интерпретатор выдает ответ, то оригинальный интерпретатор должен выдавать такой же ответ (обратное неверно).
4. Функциональные **тесты** также являются важной частью самостоятельной работы так как позволяют обучающемуся проверять то, что он сделал, а также облегчают убеждение преподавателя в корректности интерпретатора, синтаксического анализатора и всего остального. Чем больше разнотипных тестов будет – тем лучше. Особенно важными тесты будут при реализации мини-Javascript, из-за наличия большого количества неявных преобразований данных в нём.  
   В большинстве задач тестирование стоит начинать с программ на мини-языке, которые занимаются вычислением факториала и т.п.
5. **Стандартная библиотека** является сопутствующим заданием к практической работе. Реализация этой части сильно поможет обучающемуся, так как облегчит тестирование и вводить меньшее количество текста при демонстрации корректности программ.
6. **REPL** также является сопутствующим заданием, так как он существенно поможет обучающемуся в исследовании правильности своей реализации.
7. **Pretty-printer** позволяет распечатывать программу в красивом, удобном для восприятия человека виде. Существенно облегчит жизнь при отладке синтаксического анализатора.
8. **Трансформации программ** заключаются выполнении различного вида анализов над AST программ. Примерами могут служить:
   1. На вход поступает положение идентификатора в исходном код программы и его новое имя, а на выходе выдаются необходимые изменения (например, в формате DIFF), которые нужно осуществить чтобы переименовать идентификатор в новое имя.
   2. Анализ объявленных функций языка Haskell и проверка возможности изменения стратегии вычисления данных функций с CBN на CBV (без нарушения корректности) ради повышения производительности.
   3. Преобразования программ для улучшения производительности (сворачивание констант, выделение общих повыражений, и т.п).
   4. Вывод типов по коду программы (наиболее актуально для функциональных языков программирования).

По разрешению преподавателя обучающиеся могут повторно сдавать новые трансформации программ, получая дополнительные баллы.

1. **Компиляция в представление какой-либо платформы** по сути является другим способом написания интерпретатора. Например, обучающийся, который реализует мини-C# может преобразовать программу в формат, байт-код JVM машины. Непосредственно преобразование можно осуществлять самостоятельно или с использованием сторонней библиотеки. Основная сложность заключается в исследовании возможностей целевой платформы, поэтому задача награждается бОльшим количеством баллов.

**3.1.5 Методические материалы для оценки обучающимися содержания и качества учебного процесса**

Для оценки обучающимися содержания и качества учебного процесса применяется анкетирование в соответствии с методикой и графиком, утвержденными в установленном порядке.

**3.2. Кадровое обеспечение**

**3.2.1** **Образование и (или) квалификация штатных преподавателей и иных лиц, допущенных к проведению учебных занятий**

К проведению лекционных занятий должны привлекаться преподаватели, имеющие диплом о высшем образовании по соответствующему направлению.

**3.2.2**  **Обеспечение учебно-вспомогательным и (или) иным персоналом**

Учебно-вспомогательный и инженерно-технический персонал должен иметь соответствующее образование и обладать навыками организации работы с пользовательскими программными продуктами в локальной сети компьютерного класса и в Интернете.

**3.3. Материально-техническое обеспечение**

**3.3.1** **Характеристики аудиторий (помещений, мест) для проведения занятий**

Учебные аудитории для проведения учебных занятий, оснащенные стандартным оборудованием, используемым для обучения в СПбГУ в соответствии с требованиями материально-технического обеспечения

**3.3.2 Характеристики аудиторного оборудования, в том числе неспециализированного компьютерного оборудования и программного обеспечения общего пользования**

Стандартное оборудование, используемое для обучения в СПбГУ. MS Windows, MS Office, Mozilla FireFox, Google Chrome, Acrobat Reader DC, WinZip, Антивирус Касперского

**3.3.3 Характеристики специализированного оборудования**

Специализированное оборудование не требуется.

**3.3.4 Характеристики специализированного программного обеспечения**

Для компьютерного класса рекомендуется использовать программное обеспечение под операционной системой GNU/Linux; в случае невозможности использования этого варианта рекомендуется Windows с установленной WSL или MacOS.

Для редактирования программ подойдут: Visual Studio Code (рекомендуется), vim или emacs.

**Для OCaml.** Необходимо присутствие в системе пакетного менеджера OPAM (<https://opam.ocaml.org/>) версии 2 и выше, а также желательно компилятора OCaml версии 4.10 или выше. (Технически компилятор может быть установлен обучающимся самостоятельно с помощью OPAM при условии наличия установленных системных пакетов (m4, gcc, make и т.п.) и достаточности свободного места на диске (1,5ГБ для установленного OCaml и наиболее популярных библиотек)).

**Для Haskell.** Необходим установленный пакетный менеджер stack (<https://docs.haskellstack.org/>) версии 2 и выше, а также компилятор GHC версии 8.6 и выше.

**3.3.5 Перечень и объёмы требуемых расходных материалов**

Для аудиторий с маркерными досками необходимы стирающиеся маркеры в объеме, достаточном для проведения курса. Для аудиторий с меловыми досками необходим мел в объеме, достаточном для проведения курса.

**3.4. Информационное обеспечение**

1. Курт Уилл: Программируй на Haskell, 2019.
2. Джон Харрисон: Введение в функциональное программирование, 1997.
3. Окасаки Крис: Чисто функциональные структуры данных, 2016.
4. Миран Липовача: Изучай Haskell во имя добра!, ДМК Пресс, 2017
5. Бенджамин Пирс: Типы в языках программирования. Перевод с английского языка. Издательство «Лямбда пресс» & «Добросвет», Москва, 2011.
6. Ричард Бёрд: Жемчужины проектирования алгоритмов: функциональный подход, ДМК Пресс, Москва, 2013.
7. Conor Mcbride: The Derivative of a Regular Type is its Type of One-Hole Contexts (Extended Abstract), 2001.
8. Will Kurt. Get Programming with Haskell. Manning Publications, 2018.
9. Bartosz Meliwski. Category theory for programmers. ImageWrap, 2018.
10. Ярон Мински, Анил Мадхавапедди, Джейсон Хикки: Программирование на языке OCaml, 2014.
11. <https://stepik.org/course/75/promo> – курс функционального программирования на платформе Stepik, часть 1
12. <https://stepik.org/course/693/promo> – курс функционального программирования на платформе Stepik, часть 2.
13. Сайт Научной библиотеки им. М. Горького СПбГУ: <http://www.library.spbu.ru/>
14. Электронный каталог Научной библиотеки им. М. Горького СПбГУ: <http://www.library.spbu.ru/cgi-bin/irbis64r/cgiirbis_64.exe?C21COM=F&I21DBN=IBIS&P21DBN=IBIS>
15. Перечень электронных ресурсов, находящихся в доступе СПбГУ: <http://cufts.library.spbu.ru/CRDB/SPBGU/>
16. Перечень ЭБС, на платформах которых представлены российские учебники, находящиеся в доступе СПбГУ: <http://cufts.library.spbu.ru/CRDB/SPBGU/browse?name=rures&resource%20type=8>

**Раздел 4. Разработчики программы**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ФИО | Ученая  степень | Ученое звание | Должность | Контакты |
| Булычев Дмитрий Юрьевич | кандидат физ.-мат. наук | доцент | доцент кафедры системного программирования | d.bylychev@spbu.ru |
| Сартасов  Станислав Юрьевич |  |  | ст.преподаватель кафедры системного программирования | s.sartasov@spbu.ru |
| Косарев Дмитрий Сергеевич |  |  | инженер | --- |