Федеральное государственное бюджетное образовательное учреждение высшего образования
«РОССИЙСКАЯ АКАДЕМИЯ НАРОДНОГО ХОЗЯЙСТВА И ГОСУДАРСТВЕННОЙ СЛУЖБЫ ПРИ ПРЕЗИДЕНТЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»

Нижегородский институт управления – филиал РАНХиГС
(наименование института (факультета))
Кафедра информатики и информационных технологий
(наименование кафедры)

УТВЕРЖДЕНО

Директор Нижегородского института управления – филиала РАНХиГС

ПРОГРАММА БАКАЛАВРИАТА

Корпоративные информационные системы управления

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ
реализуемой с применением электронного (онлайн) курса

Б1.В.05 ОСНОВЫ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ
(индекс и наименование дисциплины (модуля), в соответствии с учебным планом)

направление подготовки
09.03.03 Прикладная информатика
(код и наименование направления подготовки)

очная, заочная
(форма(ы) обучения)

2024
(год набора)

Нижний Новгород, 2024 г.

Автор(ы)-составитель(и):

Заведующий кафедрой информатики и информационных технологий, кандидат технических наук, доцент *(ученая степень и(или) ученое звание, должность, наименование кафедры)*	И.И. Гребенюк *(Ф.И.О.)*
Доцент кафедры информатики и информационных технологий, кандидат технических наук, доцент *(ученая степень и(или) ученое звание, должность, наименование кафедры)*	А.Б. Гордеев *(Ф.И.О.)*
Заведующий кафедрой информатики и информационных технологий, кандидат технических наук, доцент *(ученая степень и(или) ученое звание, должность, наименование кафедры)*	И.И. Гребенюк *(Ф.И.О.)*

Изменения РПД Б1.В.05 «Основы функционального программирования» одобрены на заседании кафедры информатики и информационных технологий

Протокол от 20 февраля 2024 г. № 12.

Содержание

1. Перечень планируемых результатов обучения по дисциплине, соотнесенных с планируемыми результатами освоения программы 4

2. Объем и место дисциплины в структуре ОП ВО 6

3.Содержание и структура дисциплины 7

4. Материалы текущего контроля успеваемости обучающихся 9

5. Оценочные материалы промежуточной аттестации по дисциплине 37

6. Методические материалы по освоению дисциплины с применением ЭК 39

7. Учебная литература и ресурсы информационно-телекоммуникационной сети Интернет 41

7.1 Основная литература 41

7.3 Учебно-методическое обеспечение самостоятельной работы. 41

7.4 Нормативные правовые документы. 41

7.5. Интернет-ресурсы. 42

8. Материально-техническая база, информационные технологии, программное обеспечение и информационные справочные системы 42

1. Перечень планируемых результатов обучения по дисциплине, соотнесенных с планируемыми результатами освоения программы

1.1 Дисциплина «Основы функционального программирования» обеспечивает овладение следующими компетенциями:

Таблица 1

Код компетенции	Наименование компетенции	Код компонента компетенции	Наименование компонента освоения компетенции
ПКс-2	Способен разрабатывать и адаптировать прикладное программное обеспечение	**ПКс-2.2**	Способен устанавливать и настраивает программное обеспечение, необходимое для разработки и функционирования прикладного решения
ПКс-3	Способен проектировать ИС по видам обеспечения	**ПКс-3.1**	Способен выбирать и использовать современные технологии и программные средства проектирования информационных систем

1.2 В результате освоения дисциплины у студентов должны быть:

Таблица 2

ОТФ/ТФ/ трудовые или профессиональные действия	Код компонента компетенции	Результаты обучения
ОТФ: выполнение работ и управление работами по созданию (модификации) и сопровождению ИС, автоматизирующих задачи организационного управления и бизнес-процессы. ТФ: • определение первоначальных требований заказчика к ИС и возможности их реализации в ИС на этапе предконтрактных работ (С/01.6); • документирование существующих бизнес-процессов организации заказчика (реверс-инжиниринг бизнес-процессов организации) в рамках проекта создания (модификации) ИС (С/07.6); • разработка модели бизнес-процессов заказчика в рамках проекта создания (модификации) ИС (С/08.6); • адаптация бизнес-процессов заказчика ИС к возможностям ИС в рамках проекта создания (модификации) ИС (С/09.6); • выявление требований к ИС в рамках выполнения работ и управления работами по созданию (модификации) и сопровождению ИС (С/11.6); • разработка архитектуры ИС в рамках выполнения работ и управления работами по созданию (модификации) и сопровождению ИС (С/14.6); • разработка прототипов ИС в рамках выполнения работ и управления работами по созданию (модификации) и сопровождению ИС (С/15.6); • разработка баз данных ИС в рамках выполнения работ и управления работами по созданию (модификации) и сопровождению ИС (С/17.6); • оптимизация работы ИС заказчика в	ПКс-2.2	На уровне знаний: – предметная область автоматизации; – программные средства и платформы инфраструктуры информационных технологий организаций; – современные подходы и стандарты автоматизации организации; – возможности типовой ИС; – предметная область автоматизации; – инструменты и методы моделирования бизнес-процессов; – устройство и функционирование современных ИС; – технологии межличностной и групповой коммуникации в деловом взаимодействии, основы конфликтологии; – современные стандарты информационного взаимодействия систем; – программные средства и платформы инфраструктуры информационных технологий организаций; – современные объектно-ориентированные языки программирования; – современные структурные языки программирования; – языки современных бизнес-приложений. На уровне умений: – разрабатывать документы в рамках выполнения работ и управления работами по созданию (модификации) и сопровождению ИС; – кодировать на языках программирования в рамках выполнения работ и управления работами по созданию (модификации) и сопровождению ИС. На уровне навыков: – сбор исходных данных у заказчика; – согласование с заказчиком предлагаемых изменений;

ОТФ/ТФ/ трудовые или профессиональные действия	Код компонента компетенции	Результаты обучения
рамках выполнения работ и управления работами по созданию (модификации) и сопровождению ИС (С/26.6). ТД: • сбор исходных данных у заказчика ИС о его бизнес-процессах в рамках проекта создания (модификации) ИС (С/09.6); • анализ функциональных разрывов и корректировка на его основе существующей модели бизнес-процессов заказчика ИС в рамках проекта создания (модификации) ИС (С/09.6); • утверждение у заказчика предлагаемых изменений его бизнес-процессов для реализации их автоматизации в типовой ИС (С/09.6); • разработка прототипа ИС в соответствии с требованиями заказчика к ИС в рамках выполнения работ и управления работами по созданию (модификации) и сопровождению ИС (С/15.6).		– утверждение у заказчика предлагаемых изменений.
ОТФ: выполнение работ и управление работами по созданию (модификации) и сопровождению ИС, автоматизирующих задачи организационного управления и бизнес-процессы. ТФ: • определение первоначальных требований заказчика к ИС и возможности их реализации в ИС на этапе предконтрактных работ (С/01.6); • документирование существующих бизнес-процессов организации заказчика (реверс-инжиниринг бизнес-процессов организации) в рамках проекта создания (модификации) ИС (С/07.6); • разработка модели бизнес-процессов заказчика в рамках проекта создания (модификации) ИС (С/08.6); • адаптация бизнес-процессов заказчика ИС к возможностям ИС в рамках проекта создания (модификации) ИС (С/09.6); • выявление требований к ИС в рамках выполнения работ и управления работами по созданию (модификации) и сопровождению ИС (С/11.6); • разработка архитектуры ИС в рамках выполнения работ и управления работами по созданию (модификации) и сопровождению ИС (С/14.6); • оптимизация работы ИС заказчика в рамках выполнения работ и управления работами по созданию (модификации) и сопровождению ИС (С/26.6). ТД: • сбор исходных данных у заказчика ИС о его бизнес-процессах в рамках проекта создания (модификации) ИС (С/07.6,	ПКс-3.1	На уровне знаний: - возможности типовой ИС; - устройство и функционирование современных ИС; - инструменты и методы моделирования бизнес-процессов организации; - инструменты и методы выявления требований; - инструменты и методы оптимизации ИС. На уровне умений: - проектировать архитектуру ИС в рамках выполнения работ и управления работами по созданию (модификации) и сопровождению ИС. На уровне навыков: - разрабатывать прототипы ИС в соответствии с требованиями.

ОТФ/ТФ/ трудовые или профессиональные действия	Код компонента компетенции	Результаты обучения
С/08.6); • описание бизнес-процессов заказчика ИС на основе полученных исходных данных в рамках проекта создания (модификации) ИС (С/07.6); • согласование с заказчиком ИС описания его бизнес-процессов в рамках проекта создания (модификации) ИС (С/07.6, С/08.6); • утверждение у заказчика ИС описания его бизнес-процессов в рамках проекта создания (модификации) ИС (С/07.6, С/08.6); • разработка модели бизнес-процессов заказчика ИС в рамках проекта создания (модификации) ИС (С/08.6); • утверждение у заказчика ИС модели его бизнес-процессов в рамках проекта создания (модификации) ИС (С/08.6); • моделирование бизнес-процессов заказчика в ИС в рамках проекта создания (модификации) ИС (С/09.6); • создание вариантов архитектурных спецификаций ИС в рамках выполнения работ и управления работами по созданию (модификации) и сопровождению ИС (С/14.6); • выбор и согласование с заинтересованными сторонами оптимальной архитектурной спецификации ИС в рамках выполнения работ и управления работами по созданию (модификации) и сопровождению ИС(С/14.6).		

1.3. Студенты также должны овладеть навыками:

- сбора исходных данных у заказчика;
- согласование с заказчиком предлагаемых изменений;
- **утверждение у заказчика предлагаемых изменений;**
- разработки прототипов ИС в соответствии с требованиями;
- разработки и адаптации прикладных программных решений;
- описания и разработки информационных систем по видам обеспечения.

2. Объем и место дисциплины в структуре ОП ВО

Индекс дисциплины Б1.В.05 «Основы функционального программирования» является дисциплиной части, формируемой участниками образовательных отношений.

Общая трудоемкость дисциплины составляет 6 зачетных единиц, 216 академических часов / 162 астрономических часа.

По очной форме обучения: количество академических часов, выделенных на контактную работу с преподавателем составляет 98 часов, из них, лекции - 48 часов, практические занятия - 48 часов, консультация – 2 часа. Самостоятельная работа составляет - 82 часа, контроль – 36 часов.

По заочной форме обучения: количество академических часов, выделенных на контактную работу с преподавателем составляет 28 часов, из них, лекции – 12 часов,

практические занятия – 14 часов, консультация – 2 часа. Самостоятельная работа составляет 179 часа, контроль – 9 часов.

Форма промежуточной аттестации в соответствии с учебным планом: экзамен.

3.Содержание и структура дисциплины

3.1 Структура дисциплины (модуля)

Таблица 3

№ п/п	*Наименование тем (разделов)*	*Объем дисциплины, час.*						*Форма текущего контроля успеваемости и промежуточной аттестации*
		Всего	*Контактная работа обучающихся с преподавателем по видам учебных занятий*				*СР*	
			*Л/ЭО, ДОТ**	*ЛР/ ЭО ДОТ**	*ПЗ/ ЭО ДОТ**	*КСР*		
Очная форма обучения								
Тема 1	История и проблематика функционального программирования	17	4		4		9	Т
Тема 2	Основы лябда-исчисления.	17	4		4		9	Т
Тема 3	Основные понятия, синтаксис и базовые функции языка функционального программирования.	17	4		4		9	Т
Тема 4	Функции. Синтаксис. Аргументы и возвращаемые значения.	17	4		4		9	Пз
Тема 5	Логические функции. Управляющие структуры. Функции печати.	17	4		4		9	Пз
Тема 6	Циклические предложения и рекурсия.	17	4		4		9	Т
Тема 7	Поиск данных. Функционалы. Свойства символов.	26	8		8		10	Пз
Тема 8	Массивы. Макросы. Пример создания программы.	26	8		8		10	Пз
Тема 9	Компиляция программ. Отладка и тестирование.	24	8		8		8	Пз
Консультация		2						
Промежуточная аттестация		36						экзамен
Всего (в з.е./акад.часах/астр.часах)::		**6/216/ 162**	**48/36**		**48/36**		**82/61,5**	**36/27**
Заочная форма обучения								
Тема 1	История и проблематика функционального программирования	22	1		1		20	Т
Тема 2	Основы лямбда-исчисления.	22	1		1		20	Т
Тема 3	Основные понятия, синтаксис и базовые функции языка функционального программирования.	22	1		1		20	Т
Тема 4	Функции. Синтаксис. Аргументы и возвращаемые значения.	23	1		1		21	Пз
Тема 5	Логические функции. Управляющие структуры. Функции печати.	22	1		1		20	Пз
Тема 6	Циклические предложения и рекурсия.	22	1		2		19	Т

№ п/п	Наименование тем (разделов)	Объем дисциплины, час.						Форма текущего контроля успеваемости и промежуточной аттестации
		Всего	Контактная работа обучающихся с преподавателем по видам учебных занятий				СР	
			Л/ЭО, ДОТ*	ЛР/ ЭО ДОТ*	ПЗ/ ЭО ДОТ*	КСР		
Тема 7	Поиск данных. Функционалы. Свойства символов.	25	3		2		20	Пз
Тема 8	Массивы. Макросы. Пример создания программы.	27	3		3		21	Пз
Тема 9	Компиляция программ. Отладка и тестирование.	23	3		2		18	Пз
Консультация		2						
Промежуточная аттестация		9						экзамен
Всего (в з.е./акад.часах/астр.часах):		**6/216/ 162**	**12/9**		**14/ 7,5**	**4/3**	**179/ 134,25**	**9/6,75**

Примечание:

** – при применении электронного обучения, дистанционных образовательных технологий в соответствии с учебным планом*

*** – формы текущего контроля успеваемости: опрос (О), тестирование (Т),), оценка доклада (Д), оценка работы с документами в микрогруппе (М), оценка презентации(Пр), отчет по практическим занятиям (Пз), отчет по лабораторным работам (Лб), контрольная работа (Кр) .*

3.2 Содержание дисциплины

Тема 1 История и проблематика функционального программирования

Общее представление о функциональном программировании и сфере его применения, включая анализ основных понятий и принципов и их иллюстрацию на материале истории языка функционального программирования, его диалектов, наследников и реализаций. Роль функциональных программ в жизненном цикле программного обеспечения и эволюции информационных технологий, а также перспективы функционального подхода к совершенствованию информационных систем.

Тема 2 Основы лямбда-исчисления

Проблема разрешения. Основные понятия λ-исчисления (термы, синтаксис термов, базовые осглашения). Соглашения о приоритете операций. Каррирование. Области видимости. Редукция. Стратегии редукции. Две базовые теоремы λ-исчисления (Теоремы Карри и Чёрча-Россера).

Тема 3 Основные понятия, синтаксис и базовые функции языка функционального программирования.

Ключевой метод функционального программирования — выбор семантического базиса для класса решаемых задач на примере организации информационной обработки символьными выражениями в языке программирования. Формализм рекурсивных функций и простые алгоритмы символьной обработки данных. Анализ требований к полноте и эффективности их обработки.

Тема 4 Функции. Синтаксис. Аргументы и возвращаемые значения.

Техника использования параметров и функций при организации вычислений. Методы управления эффективностью и порядком вычислений, организованных как применение функций к заданным аргументам. Формирование результата в зависимости от параметризации форм, задающих вычисления.

Тема 5 Логические функции. Управляющие структуры. Функции печати.

Понятие логического выражения. Возвращаемые значения функций. Решение задач с использованием логических функций. Вывод результата с помощью функций отображения данных в различных форматах.

Тема 6 Циклические предложения и рекурсия.

Принципы реализации циклических процедур. Описание структур данных, удобных для динамической обработки информации. Рекурсивные процедуры.

Тема 7 Поиск данных. Функционалы. Свойства символов.

Программирование отображений и использование функционалов как метод повышения производительности программирования и эффективности отладки программ. Схемы отображений аргументов и формирования результатов на основе отображающих функций над компонентами структур данных и определения различных функциональных схем переработки данных.

Тема 8 Массивы. Макросы. Пример создания программы.

Поддержка полного жизненного цикла программ с помощью быстрого прототипирования и спецификации программ. Примеры и схемы с привлечением частичных функций на доступных типах данных с доведением до полных функций, приспособленных к обработке произвольных данных. Правила обработки одномерных и многомерных массивов. Создание макросов.

Тема 9 Компиляция программ. Отладка и тестирование.

Требования к компиляции функциональных программ. Определение компилятора. Разложение программы на функции с разным уровнем отладки. Компиляция программ как один из методов оптимизации процессов, осуществляемый как символьное преобразование — трансляция с исходного языка высокого уровня на язык низкого уровня. Примеры построения прототипов системы, детальная разработка алгоритмов и отладка программ.

4. Материалы текущего контроля успеваемости обучающихся

4.1. В ходе реализации дисциплины «Основы функционального программирования» используются следующие методы текущего контроля успеваемости обучающихся:

Таблица 4.

№	Тема (раздел)	Методы текущего контроля успеваемости
Тема 1	История и проблематика функционального программирования	Тестирование
Тема 2	Основы лямбда-исчисления	Тестирование
Тема 3	Основные понятия, синтаксис и базовые функции языка функционального программирования.	Тестирование
Тема 4	Функции. Синтаксис. Аргументы и возвращаемые значения.	Отчет по практическим занятиям
Тема 5	Логические функции. Управляющие структуры. Функции печати.	Отчет по практическим занятиям
Тема 6	Циклические предложения и рекурсия.	Тестирование
Тема 7	Поиск данных. Функционалы. Свойства символов.	Отчет по практическим занятиям
Тема 8	Массивы. Макросы. Пример создания программы.	Отчет по практическим занятиям
Тема 9	Компиляция программ. Отладка и тестирование.	Отчет по практическим занятиям

4.2. Типовые материалы текущего контроля успеваемости обучающихся

Примерные вопросы теста к теме 1

"Кто первым высказал идею, что проще описывать задачу математическими формулами, чем описывать ее на языке машинных кодов?"

1) "Джон Бэкус"
2) "Алан Тьюринг"
3) "Джон фон Нейман."
4) "Андрей Петрович Ершов"
5) "Хаскелл Карри"
6) "Мозес Шёнфинкель"

"Какой язык программирования был первым языком высшего уровня (в том числе частично реализовавшим и парадигму функционального программирования)?"

1. "FORTRAN"
2. "ADA"

3. "C"
4. "PASCAL"
5. "BASIC"
6. "LISP"
7. "Haskell"

"Какой язык программирования был первым языком функционального программирования?"

1. "FORTRAN"
2. "ADA"
3. "C"
4. "PASCAL"
5. "BASIC"
6. "LISP"
7. "Haskell"

"Кто является разработчиком Ля́мбда-исчисле́ния (λ-исчисление)?"

1. "Алонз Чёрч"
2. "Джон МакКарти"
3. "Хаскелл Карри"
4. "Алан Тьюринг"
5. "Давид Гильберт"

"Выберите функциональные языки программирования"

- "Haskell"
- "ML"
- "LISP"
- "ADA"
- "PASCAL"
- "BASIC"
- "Python"
- "Java"

Примерные вопросы теста к теме 2

"Что такое Проблема разрешения?"

- "Задача из области оснований математики, найти алгоритм, который бы принимал в качестве входных данных описание любой проблемы разрешимости (формального языка и математического утверждения «S» на этом языке) — и, после конечного числа шагов, останавливался бы и выдавал один из двух ответов: «Истина!» или «Ложь!», — в зависимости от того, истинно или ложно утверждение «S». Ответ не требует обоснований, но должен быть верным."
- "Утверждение о том, что любое чётное число, начиная с 4, можно представить в виде суммы двух простых чисел."
- "Утверждение о том, что всякое n-мерное многообразие гомотопически эквивалентно n-мерной сфере тогда и только тогда, когда оно гомеоморфно ей."
- "Утверждение, что в классической логике любое высказывание, вне зависимости от степени сложности, истинно или ложно и третьего не дано."

"Выберите определение Лямбда-исчисления."

- "Формальная система, разработанная для формализации и анализа понятия вычислимости, по сути описывает понятие алгоритма."
- "Аксиоматическая логическая система, интерпретацией которой является алгебра высказываний."
- "Символический метод записи чисел, представление чисел с помощью письменных знаков."

- "Комплекс определений, реализующий способ определять положение и перемещение точки или тела с помощью чисел или других символов. "

"Укажите факторы, которые сделали λ-исчисление основной формализаций, применяемой в исследованиях, связанных с языками программирования."

- "Это единственная формализация, которая, хотя и с некоторыми неудобствами, действительно может быть непосредственно использована для написания программ."
- "λ-исчисление дает простую и естественную модель для таких важных понятий, как рекурсия и вложенные среды."
- "Большинство конструкций традиционных языков программирования может быть более или менее непосредственно отображено в конструкции λ-исчисления."
- "Описание исследуемых явлений производится средствами уточненного естественного языка."
- "Предназначение создаваемых моделей состоит в выявлении тех существенных факторов, которые формируют интегральные свойства объекта-оригинала."
- "Процесс построения и использования модели, т.е. такого материального или искусственно созданного кем-то объекта, который в процессе познания (изучения) замещает его оригинал, сохраняя важные (по мнению автора) или типичные черты"

"Установить соответствие между понятиями"

- "Терм"
- "Переменная"
- "Применение функции (аппликация)"
- "λ-абстракция (анонимная функция)"
- "Программный код в λ-исчислении"

и

- "x"
- "F x, где F — функция, x — подставляемое в неё значение аргумента"
- "λx.F, где x — аргумент функции, F — её тело"
- "Если x – переменная, а M – терм, то (λx.M) – терм)"
- "Если M и N – термы, то (MN) – терм."

"Выберите основные правила написания программного кода в λ-исчислении."

- "Переменные x , y , z … являются термами."
- "Если M и N – термы, то (MN) – терм."
- "Если x – переменная, а M – терм, то (λx.M) – терм)"
- "Если M и N формулы, то формулами являются M v N, M ^ N, M -> N и M <-> N"
- "(M · N) · K = M · (N · K) = M · N ·K"

"Что такое аппликация в λ-исчислении?"

- "Применение или вызов функции по отношению к заданному значению."
- "Вырезание и наклеивание (нашивание) фигурок, узоров или целых картин из кусочков бумаги, ткани, кожи, растительных и прочих материалов на материал-основу (фон)"
- "Частичное наложение звуков одной морфемы на другую, в результате чего эти звуки выполняют двойную роль."
- "Функция полученная определенным образом."

"Укажите основные соглашения о приоритете операций в λ-исчислении."

- "Аппликация левоассоциативна. "
- "Аппликация забирает себе всё, до чего дотянется вправо."
- "Аппликация забирает себе всё, до чего дотянется влево."

- "Аппликация правоассоциативна. "
- "Аппликация коммуникативна. "

"Укажите правильную иллюстрацию того, что аппликация левоассоциативна:"
- "s t u — это тоже самое, что (s t) u"
- "s t u — это тоже самое, что s (t u)"
- "s t u — это тоже самое, что (s) (t) (u)"

"Укажите правильную иллюстрацию того, что область действия лямбда-выражения простирается вправо насколько возможно."
- "λx. λy. x y x означает то же самое, что λx. (λy. ((x y) x))"
- "λx. λy. x y x означает то же самое, что λx. (λy. ((x y) λx))"
- "λx. λy. x y x означает то же самое, что λx. (λy. (λ(x y) x))"

"Укажите определение каррирования"
- "Преобразование функции от многих переменных в функцию, берущую свои аргументы по одному."
- "Преобразование функции путем последовательного сокращения значений."
- "Преобразование функции путем последовательного объединения областей значения."
- "Преобразование функции путем исключения аргументов через вырождение."

"Правильно сгруппируйте определения и примеры."
- "Связанные переменные"
- "Свободные переменные"

и
- "Переменная x в терме λx. λy. x"
- "Переменные находятся в позиции, в которой они не связаны никакой вышележащей абстракцией."
- "Переменные по которым выше в дереве разбора были λ-абстракции."
- "Переменная y в терме λx. λy. x"

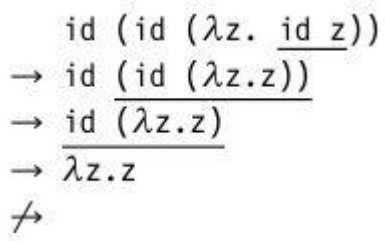
"Правильно сгруппируйте определения и примеры."
- "Редукция"
- "α -преобразование"

и
- "Процесс вычисления терма при котором в выражении (λx.t)y каждый шаг вычисления будет заключаться в замене всех вхождений переменной x внутри t на y. "
- "(λb.λx.bxx)z [b=z] <=> λx.zxx"
- "Замене в выражении λx.t имени переменной x на любое другое имя с одновременной заменой всех свободных вхождений этой переменной в выражение t, с целью ликвидации лишнего связывания переменных."
- "(λxy.x) y <=> (λxz.x) y"

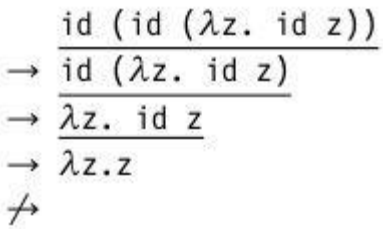
"Правильно сгруппируйте определения и примеры стратегий редукции. Примеры рассматриваются на базе терма (λx.x) ((λx.x) (λz. (λx.x) z)), содержащего три редекса:"
- "Полная β-редукция."
- "Нормальный порядок вычислений."

и

- "В этом случае каждый раз редекс внутри вычисляемого терма выбирается

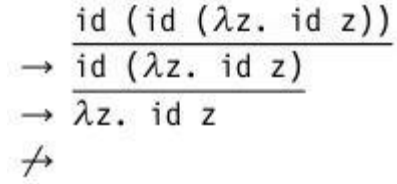


произвольным образом. Пример вычислен от внутреннего редекса к внешнему."
- "Первым всегда сокращается самый левый, самый внешний редекс."

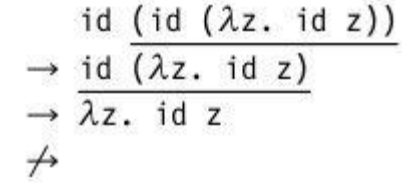


"Правильно сгруппируйте определения и примеры стратегий редукции. Примеры рассматриваются на базе терма (λx.x) ((λx.x) (λz. (λx.x) z)), содержащего три редекса:"
- "Вызов по имени."
- "Вызов по значению"

и
- "Порядок вычислений в этой стратегии аналогичен нормальному порядку вычислений, но к нему добавляется запрет на проведение сокращений внутри абстракции. "



- "Сокращение начинается с самого левого (внешнего) редекса, у которого в правой части стоит замкнутый терм, который нельзя вычислить далее. "



"Сопоставьте теоремы"
- "Теорема Карри"
- "Теорема Чёрча-Россера"

и
- "Если у терма есть нормальная форма, то последовательное сокращение самого левого внешнего редекса приводит к ней"
- "Если терм X редуцируется к термам Y1 и Y2 , то существует терм L , к которому редуцируются и терм Y1 и терм Y2. "
- "Каждый типизируемый терм имеет нормальную форму и каждая возможная последовательность редукций, начинающаяся с типизируемого терма, завершается"

Примерные вопросы теста к теме 3

"Что такое парадигма программирования?"

- "Совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию)"
- "Совокупность стилей, определяющих идеи написания компьютерных программ"
- "Совокупность фундаментальных научных установок, представлений и терминов, принимаемая и разделяемая научным сообществом и объединяющая большинство его членов"
- "Список словоформ, принадлежащих одной лексеме и имеющих разные грамматические значения"

"Укажите основные модели программирования"

2 "Императивное программирование"
3 "Декларативное программирование"
4 "Функциональное программирование"
5 "Объектно-ориентированное программирование"
6 "Линейное программирование"
7 "Последовательное программирование"
8 "Объектно-образное программирование"
9 "Каскадное программирование"
10 "Итеративное программирование"

"Установите соответствие между понятиями и определениями"

– "Императивное программирование"
– "Объектно-ориентированное программирование"
– "Декларативное программирование"
– "Функциональное программирование"

и

– "парадигма программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования"
– "парадигма программирования, в которой задаётся спецификация решения задачи, то есть описывается, ЧТО представляет собой проблема и ожидаемый результат"
– "парадигма программирования, которая описывает процесс вычисления в виде инструкций, изменяющих состояние программы, очень похожа на приказы, выражаемые повелительным наклонением в естественных языках, то есть это последовательность команд, которые должен выполнить компьютер"
– "парадигма программирования в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании"
– "парадигма разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков"

"Установить соответствие между понятиями и определениями"

- "Интерпретация"
- "Компиляция"

и

- "пооператорный (покомандный, построчный) анализ, обработка и тут же выполнение исходной программы или запроса"
- "трансляция программы, составленной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду"
- "трансляция команды составленной на исходном языке высокого уровня, в эквивалентную команду на низкоуровневом языке, близком машинному коду"
- "перевод исходного кода программы в байт-код"

"Укажите свойства функциональных языков"

1. "краткость и простота"
2. "строгая типизация"
3. "функции — это значения"
4. "чистота (отсутствие побочных эффектов)"
5. "отложенные (ленивые) вычисления"
6. "выразительность"
7. "концептуальная целостность"
8. "универсальность"

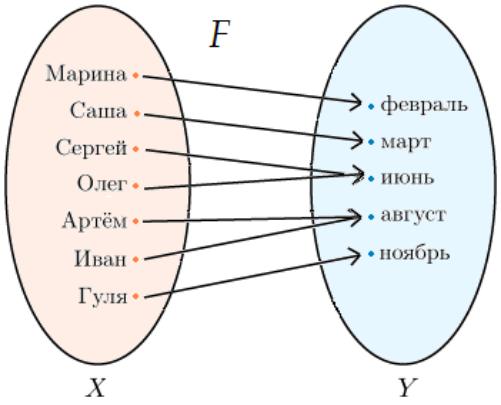
"Выберите определение функции"

1. "Соответствие между элементами двух множеств, установленное по такому правилу, что каждому элементу одного множества ставится в соответствие некоторый элемент из другого множества"
2. "Соответствие между двумя выражениями используемых в вычислении"
3. "Преобразование данных выполненное по определенному правилу"
4. "Преобразование двух множеств установленное по определенному правилу, при котором преобразование выполняется над каждым элементом преобразуемого множества"

"Что такое функция высшего порядка?"

- "Функция, принимающая в качестве аргументов другие функции или возвращающая другую функцию в качестве результата"
- "Функция, принимающая в качестве аргументов значения со степенью больше 1."
- "Функция, возвращающая в качестве результата значения со степенью больше 1."
- "Функция, принимающая в качестве аргументов производные и первообразные высших порядков"

"Установить соответствие обозначений и понятий на данном рисунке"



6. "X"
7. "Y"
8. "F"

и

1. "Область определения"
2. "Область значений"
3. "Функция"
4. "Область отображения"
5. "Правило обработки"

"Установить соответствие."

1. "f(a)=p, f(b)=q, f(c)=q, f(d)=r"
2. "f(x)=2x+5"

и

- "Определение функции перечислением"
- "Определение функции аналитически"
- "Графическое определение функции"
- "Словесное описание функции"

"Установить соответствие"

- "Функция частично определенная над множеством A"
- "Функция полностью определенная над множеством A"

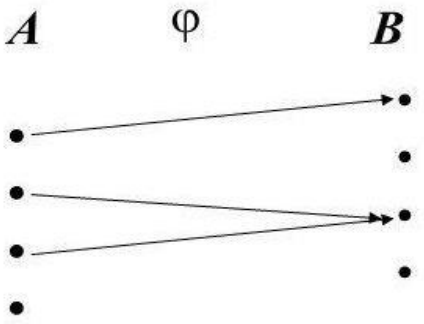
и

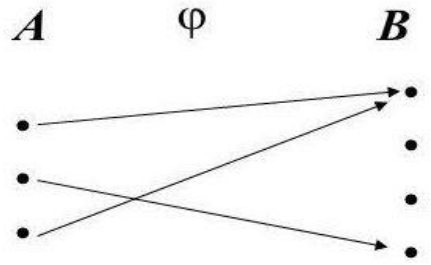
- "если в множестве A существуют элементы, для которых образ посредством этой функции не определен"
- "если она не является частичной над множеством"
- "если области определения и значения функции совпадают"
- "если области определения и значения функции не совпадают"
- "если области определения и значения функции пересекаются хотя бы в одной точке"

"Провести классификацию."

- "Частично определенная функция"
- "Полностью определенная функция"

и





"Провести классификацию. Символ ∈ означает «принадлежит». R - Множество вещественных чисел"

- "Частично определенная функция"
- "Полностью определенная функция"

и

- "f(x)=1/x, x ∈ R"
- "f(x)=sqr(x), x ∈ R (sqr - Извлечение квадратного корня)"

- "f(x)=x^2, x ∈ R"
- "f(x)=sqr(x), x ∈ R, x>0 (sqr - Извлечение квадратного корня)"
- "f(x)=1/x, x ∈ R, x≠0"

"Есть функция МАКС(a,b), которая выдает в качестве результата наибольшее из аргументов. Укажите правильно составленные функции."

- "НАИБ(a,b,c)=МАКС(МАКС(a,b),c)"
- "НАИБ(a,b,c)=МАКС(a,МАКС(b,c))"
- "НАИБ(a,b,c)=МАКС(МАКС(a,b),МАКС(b,c))"
- "НАИБ(a,b,c)=МАКС(a,b,c)"

"Есть функция МАКС(a,b), которая выдает в качестве результата наибольшее из двух аргументов, и функция НАИБ(a,b,c), которая выдает в качестве результата наибольшее из трех аргументов. Укажите правильно составленные функции находящие наибольшее из шести аргументов (a,b,c,d,e,f)"

- "МАКС(НАИБ(a,b,c),НАИБ(d,e,f))"
- "НАИБ(МАКС(a,b),МАКС(c,d),МАКС(e,f))"
- "МАКС(МАКС(МАКС(a,b),МАКС(c,d)),МАКС(e,f))"
- "НАИБ(НАИБ(a,b,c),НАИБ(d,e,f))"

"Есть функции: "ПЛЮС (x,y) = x+y","МИНУС (x,y) = x-y", "УМН (x,y) = x*y", "ДЕЛ (x,y) = x/y"
"Используя их запишите функцию a+b*c"

- "ПЛЮС (a,УМН (b,c))"
- "ПЛЮС (a,УМН (б,c))"
- "ПЛЮС (a,УМН (в,c))"

"Есть функции: "ПЛЮС (x,y) = x+y","МИНУС (x,y) = x-y", "УМН (x,y) = x*y", "ДЕЛ (x,y) = x/y"
"Используя их запишите функцию (2*x+3*y)/(x-y)"

- "ДЕЛ (ПЛЮС (УМН (2,x),УМН (3,y)),МИНУС (x,y))"
- "ДЕЛ (ПЛЮС (УМН (2,x),УМН (3,y)),МИНУС (x,y))"

Практическая работа к теме 4.

1. Разобрать работу с командной строкой. Вычисление простых математических выражений типа:

2*2

2+2*2 и т. д. Убедиться в том что haskell правильно понимает порядок вычислений и применения скобок.

2. Убедиться в том, что haskell правильно оценивает истинность. (3>2 и 2>3)

Идеология написания программ в Haskell – программы пишутся в блокноте и загружаются. Затем описанная функция вызывается с аргументами.

3. Написание функций вычисляющих математические выражения при заданных аргументах:

x^2, ax^2+bx+c:

*kvad (x) = x*x*

*kvadvyr (a,b,c,x) = a*x*x+b*x+c* или с использованием описанной функцией kvad - *a*kvad (x) b*x+c*

3. Рекурсивные функции. Вычисление степени числа, факториала числа, числа фибоначчи:

Математическая запись	Программа на haskell

n!	| 0!=1 | n!=(n-1)!*n	factorial (0)=1 factorial (n) = factorial (n-1)*n
x^y =	| x^0 = 1 |x^y= $x^{(y-1)}$*x	stepen (x,0) = 1 stepen (x,y) = stepen (x,y-1)*x
F_x =	|F_0=0 |F_1=1 |F_x=F_{x-2}+F_{x-1}	fib (0) = 0 fib (1) = 1 fib (x) = fib(x-2)+fib(x-1) -(это не список а вычисление x-ового числа фибоначчи)

5. Работа со списками. Убедиться в том, что haskell понимает списки как перечисления. Ввести в командной строке [1..10], [1,3..10], [2,4..10]

6. Написание функций выходящих на экран списки:

Все натуральные числа от 1 до n: *spisnat n = [1..n]* или *spisnat n =[x|x←[1..n]]*

Все четные числа от 1 до n: *spischet n = [2,4..n]* или *spischet n = [x|x←[2,4..n]]*

Все нечетные числа от 1 до n: *spisnechet n = [1,3..n]* или *spisnechet n = [x|x←[1,3..n]]*

7. Комбинирование функций:

Задание	Программа на haskell
Список квадратов натуральных чисел.	spiskvat n=[x*x|x<-[1..n]]
Список факториалов.	fact(0)=1 fact(x)=fact(x-1)*x spisfact n=[fact(x)|x<-[1..n]]
Список степеней двойки.	spisstepdvoik n =[2^x|x←[1..n]]
Список треугольных чисел Ферма.	treugferma n= [(x*(x+1))/2|x<-[1..n]]
Список пирамидальных чисел Ферма.	piramferma n= [(x*(3*x-1))/2|x<-[1..n]]

Практическая работа к теме 5.

1. Составить программу, которая выводит на экран все алфавитно-цифровые последовательности, содержащиеся в файле. Алфавитно-цифровой называется непрерывная последовательность цифр и букв латинского алфавита. Пример: если файл содержит символы asdf-+^&@qwert1y!@# то результатом работы программы должно быть: asdf qwert1y

2. Составить программу, принимающую в командной строке имя файла и один из двух символов ’u’ или ’l’. В зависимости от того, какой из символов передан, она преобразует содержимое файла к верхнему или нижнему регистру и выводит на экран.

Примерные вопросы теста к теме 6

"Что такое рекурсивная функция?"

- "Это функция, которая в своей записи содержит себя же"
- "Это композиция функций от одного аргумента"
- "Это функция вызываемая с различными аргументами"
- "Это функция модифицирующая себя в ходе вычисления"

"Что необходимо, чтобы рекурсивная функция вычислялась для любого N?"

- "Необходимо, чтобы для некоторых N функция была определена нерекурсивно."
- "Необходимо, чтобы N было целое и положительное"
- "Необходимо, чтобы N было вещественным"
- "Необходимо, чтобы N было ненулевое"

"Укажите основные проблемы рекурсии"

- "Повторные вычисления одних и тех же значений"
- "Дерево рекурсивных вызовов может оказаться бесконечным"

- "Рекурсия - неконтролируемый процесс"
- "Слишком длительное время компиляции кода"

"Укажите основные способы решения проблем рекурсии"

- "Использование накапливающего параметра — аккумулятора"
- "Использование хвостовой рекурсии, которая представляет собой специальный вид рекурсии, в которой имеется единственный вызов рекурсивной функции и при этом этот вызов выполняется после всех вычислений"
- "Полностью исключить рекурсию из вычислений"
- "Использование рекурсивных определений, позволяющих при трансляции обеспечить вычисления в постоянном объёме памяти через итерацию вызываемой функции"

"Сопоставьте обозначения и названия базисных операций"

- "prefix(x,y)≡x:y≡[x|y]"
- "head(x)≡h(x)"
- "tail(x)≡t(x)"

и

- "Операция отсечения головы"
- "Операция создания пары"
- "Операция отсечения хвоста"
- "Операция модификации списка"
- "Тождественная операция"

"Укажите основые аксиомы базисных операций"

- "head (x : y) = x"
- "tail (x : y) = y"
- "prefix (head (x : y), tail (x : y)) = (x : y)"
- "head (x : y) = y"
- "tail (x : y) = x"

"Укажите правильную запись функции определения длины списка Length, через базовые операции prefix, head, tail, где"

"prefix (x, y) - операция создания пары "
"head (x) - операция отсечения головы"
"tail (x) - операция отсечения хвоста"

- "Length ([ ]) = 0" "Length (L) = 1 + Length (tail (L))"
- "Length ([ ]) = 0" "Length (L) = 1 + Length (head (L))"
- "Length ([ ]) = 0" "Length (L) = Length (tail (L)) + Length (head (L))"

"Укажите правильную запись функции слияния двух списков Append, через базовые операции prefix, head, tail, где"

"prefix (x, y) - операция создания пары "
"head (x) - операция отсечения головы"
"tail (x) - операция отсечения хвоста"

- "Append ([ ], L2) = L2" "Append (L1, L2) = prefix (head (L1), Append (tail (L1), L2))"
- "Append ([ ], L2) = L2" "Append (L1, L2) = prefix (tail (L1), Append (head (L1), L2))"
- "Append ([ ], L2) = L2" "Append (L1, L2) = prefix (head (L2), Append (tail (L1), L2))"

"Сопоставьте понятия и определения"

- "Полиморфизм"
- "Параметрический полиморфизм"

- "Специальный (ad-hoc) полиморфизм"

и

- "это возможность использования в одном и том же контексте различных программных сущностей (объектов, типов данных и т. д.) с одинаковым интерфейсом."
- "это возможность определения обобщённых структур данных и функций, поведение которых не зависит от типов значений, которыми они оперируют."
- "(или «перегрузка имён») это возможность давать одинаковые имена программным сущностям с различным поведением"

Практическая работа к теме 7.

1. Составить программу, которая должна считывать с командной строки имена двух файлов и выводить на экран те строки этих файлов, которые отличаются друг от друга.
2. Составить программу, принимающую имя файла и выводящая его строки, перед каждой из которых записывается ее номер.

Практическая работа к теме 8.

1. Составить программу, которая считывает из файла матрицу и выводит на экран суммы столбцов этой матрицы.
2. Составить программу, которая считывает из файла матрицу и выводит на экран сумму диагональных элементов этой матрицы.
3. Составить программу, которая считывает две матрицы из файлов и записывает в третий файл матрицу, являющуюся их суммой.
4. Составить программу, которая считывает из одного файла матрицу, а из другого — вектор и записывает в третий файл результат умножения матрицы на вектор.

Практическая работа к теме 9.

Практическое задание выполняется индивидуально или в команде из двух человек.

Вариант задания и состав команды должны быть предварительно согласованы с преподавателем. Факт согласования отражается в таблице. **Несогласованные практические задания не принимаются!**

В случае командной работы задание разделяется на две части:

- базовая часть, которая совместно выполняется всеми участниками команды.
- индивидуальная часть, которая представляет собой расширение функционала базовой части и выполняется самостоятельно каждым участником команды.

Во время выполнения задания настоятельно рекомендуется следовать методическим указаниям.

По результатам работы должен быть составлен письменный отчёт, состоящий из следующих частей:

1. Постановка задачи, основной функционал приложения.
2. Модули, на который разделён проект, их взаимосвязь. Для каждого модуля должно быть указано:
   - предназначение модуля (например, работа с графикой, искусственный интеллект для игры и т.д.);
   - основные типы данных и функции, реализованные в модуле.
3. Используемые библиотеки.
4. Сценарии работы с приложением и примеры использования.

Попробуйте использовать LaTeX. Это даст вам отличную возможность освоить ещё один полезный навык, который скоро вам пригодится для написания курсовых и выпускных работ.

⚑ ⚑ ⚑ Фатальные недостатки

Ниже приведён список фатальных недостатков программы, при нахождении которых программа **не может быть принята** до их устранения (как бы хорошо она ни работала):

- наличие предупреждений при компиляции (обнаружить их можно, используя флаги -Wall и -Werror);
- использование транслита и/или русского языка в идентификаторах и/или комментариях;
- отсутствие проверки корректности входных данных (конфигурационные файлы, ввод из stdin, аргументы командной строки и т.д.) - программа должна не аварийно "падать", а выводить человеко-читаемое сообщение об ошибке;
- строчки кода длиной больше 100 символов;
- использование магических констант (исключением могут быть только константы, используемые для отрисовки: размеры объектов, RGB-значения для цвета и т.д.);
- использование функций undefined и unsafePerformIO.

Варианты заданий

Можно выбрать один из предложенных вариантов или придумать интересную для себя задачу!

Две различные команды не могут выбрать один вариант практического задания. Два участника одной команды не могут выбрать одну индивидуальную часть одного задания.

Если несколько студентов собираются индивидуально (не в команде) писать один вариант, то это возможно только при следующих ограничениях:

- один вариант могут писать не более чем n+1 человек, где n — количество индивидуальных частей в варианте;
- студент, который раньше всех занял вариант, пишет только базовую часть;
- остальные студенты должны выполнить также одну из индивидуальных частей, но у всех они должны быть разные;
- постановка индивидуальной задачи должна быть дополнительно согласована.

Возможные варианты заданий:

- ***Интерпретаторы***
  - λ-исчисление
  - Диалект Лисп
  - Машина Тьюринга
- ***Моделирование***
  - Игра "Жизнь"
  - Визуализация различных клеточных автоматов
- ***Игры***
  - Настольные игры (шашки, шахматы, реверси, нарды и т.п.)
  - Классические компьютерные игры (арканоид, тетрис, змейка, сапёр и т.п.)
  - Бесконечный платформер
- ***Головоломки***
  - Чайнворд
  - Японский кроссворд
  - Судоку
  - Лабиринты
  - Wordle
- Kanban-доска

Если вы собираетесь выполнить другое задание по своему выбору, необходимо составить краткое описание базовой и дополнительных частей и зафиксировать эту информацию (письмо преподавателю, файл README в проекте).

λ-исчисление

Описание

λ-исчисление (лямбда-исчисление) лежит в основе большинства функциональных языков программирования: семейства Лиспа (Common Lisp, Scheme, Clojure и др.) и семейства ML (Standard ML, Haskell, Agda и пр.).

λ-исчисление состоит из языка λ-выражений и набора правил преобразования. Базовые правила построения λ-выражений:

- переменная x является λ-выражением;
- если e — λ-выражение, а x — переменная, то λ x . e также λ-выражение (**лямбда-абстракция**);
- если e_1 и e_2 — λ-выражения, то e_1 e_2 также λ-выражение (**аппликация**).

Для удобства работы с λ-выражениями, при записи могут использоваться следующие упрощения:

- внешние скобки могут быть опущены (λ x . e_1) e_2;
- аппликация считается лево-ассоциативной e_1 e_2 e_3 ≡ ((e_1 e_2) e_3);
- тело λ-выражения распространяется вправо насколько возможно λ x . e_1 e_2 ≡ λ x . (e_1 e_2);
- последовательные λ-абстракции схлопываются в одну λ x . λ y . λ z . e ≡ λ x y z . e.

Оператор λ *связывает* переменную в выражении λ x . e. Переменные, подпадающие под какой-либо оператор λ-абстракции, называются *связанными*. Все прочие переменные называются *свободными*. Например, переменная y свободна в выражении λ x . y x. Переменная связывается *ближайшим* оператором λ. Например, единственное вхождение переменной x в выражении λ x . y (λ x . x) связано со вторым оператором λ.

Значение λ-выражения определяется правилами редукции:

- α-**конверсия**: переименовывание связанных переменных
- β-**редукция**: применение функции к аргументам
- η-**конверсия**: выражает принцип *две функции идентичны, если имеют одинаковый результат на всех входах*

Применение функции к аргументам осуществляется за счёт подстановки выражения-аргумента вместо связанной переменной в теле λ-выражения:

x[x -> e] ≡ e
y[x -> e] ≡ y, если y ≠ x
(e_1 e_2) [x -> e] ≡ (e_1 [x -> e]) (e_2 [x -> e])
(λ x . e_1) [x -> e] ≡ λ x . e_1
(λ y . e_1) [x -> e] ≡ λ y . (e_1 [x -> e]), если y ≠ x и y не входит свободно в e

Язык λ-выражений может быть расширен:

- базовыми типами данных (например, числа, булевы значения, строки);
- базовыми контейнерными типами (списки и кортежи);
- встроенными операциями (например, +, -, sin, cos, and, or, ++)
- специальными конструкциями (например, if e1 then e2 else e3 или let x1 = e1 in e2);
- системой типов;
- пользовательские структуры данных;
- и т.д.

Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна:

- определять структуру данных для синтаксического дерева;
- содержать раздельно парсер и интерпретатор (с любой стратегией редукций);
- предоставлять простую интерактивную среду программирования (REPL).

Расширенный парсер (индивидуальная часть)

Расширенный парсер должен добавлять хотя бы 2 различные возможности к базовому варианту. Ниже перечислены возможные варианты расширения парсера, однако этим списком они не ограничены:

- разбор расширенного λ-исчисления;
- восстановление после ошибок (например, если пользователь написал запятую (,) вместо точки (.), парсер может запомнить эту ошибку и продолжить разбор программы);
- поддержка пользовательских инфиксных операций с возможностью задать приоритет и ассоциативность;
- и т.д.

Система типов (индивидуальная часть)

Система типов помогает определить семантику λ-выражений, но также может быть использована для оптимизаций при интерпретации и генерации кода.

Система типов должна поддерживать хотя бы 2 различные возможности:

- базовые типы (числа, булев тип, строки);
- контейнерные типы (списки, кортежи, суммы);
- параметрический полиморфизм;
- пользовательские типы данных;
- механизм автоматического вывода типа для терма;
- и т.д.

Расширенная интерактивная среда (индивидуальная часть)

Расширенная интерактивная среда должна добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерактивной среды, однако этим списком они не ограничены:

- команды интерпретатора:
  - показать все возможные варианты редукции терма (одного шага) и выбрать один;
  - показать тип выражения;
  - поменять порядок редукции;
  - перевести терм из/в кодировку Чёрча;
  - загрузить программу из файла;
- интерпретация расширенного λ-исчисления;
- дружелюбные сообщения об ошибках (например, для замкнутых термов при опечатке в имени переменной можно предложить имена переменных, отличающихся одной буквой, которые находятся в области видимости);
- и т.д.

Диалект Лисп

Описание

В данном задании предлагается реализовать интерпретатор диалекта языка программирования Лисп.

Базовым объектом Лисп является S-выражение и в предлагаемом диалекте оно может быть представлено

- атомом (IAMATOM, numberp, setf),
- числовым литералом (10, 34.2),
- строковым литералом ("hello, world!"),
- пустым списком (nil, ()),
- точечной парой ((1 . 2)).

Как обычно, непустой список представляется точечной парой, вторым элементом которой является другой список: (1 "asd" atom) эквивалентно (1 . ("asd" . (atom . nil))).

Каждое S-выражение может быть вычислено по следующим правилам:

- числовые, строковые литералы, пустой список и атом вычисляются в себя;
- список (f ...) вычисляется применением функции (или раскрытием макроса) f к остальным элементам списка;
- функция перед применением вычисляет все свои аргументы;
- макросы и специальные формы не вычисляют аргументы перед применением.

Программа на этом диалекте состоит из последовательности S-выражений.

Диалект языка выражений может быть расширен:

- ***встроенными специальными формами:***

- quote (');
- if;
- let;
- macro;
- setf;
- ***пользовательскими функциями: lambda, defun;***
- ***лексическим/динамическим связыванием;***
- ***операциями ввода-вывода;***
- ***функциями с переменным числом аргументов;***
- ***и т.д.***

Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна:

- определять структуру данных для синтаксического дерева;
- содержать раздельно парсер и интерпретатор;
- предоставлять простую интерактивную среду программирования (REPL);
- предоставлять встроенные предикаты и функции:
  - atomp, numberp, listp, =;
  - car, cdr, cons.

Расширенный парсер (индивидуальная часть)

Расширенный парсер должен добавлять хотя бы 2 различные возможности к базовому варианту. Ниже перечислены возможные варианты расширения парсера, однако этим списком они не ограничены:

- разбор расширенного диалекта;
- восстановление после ошибок (например, если пользователь написал лишнюю закрывающую скобку, парсер может запомнить эту ошибку и продолжить разбор программы);
- и т.д.

Расширенная интерактивная среда (индивидуальная часть)

Расширенная интерактивная среда должна добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерактивной среды, однако этим списком они не ограничены:

- команды интерпретатора:
  - загрузить программу из файла;
  - провести один шаг вычисления;
  - раскрыть вызовы макросов;
- интерпретация расширенного диалекта;
- дружелюбные сообщения об ошибках (например, при опечатке в имени переменной можно предложить имена переменных, отличающихся одной буквой, которые находятся в области видимости);
- и т.д.

Генерация кода (индивидуальная часть)

Модуль генерации кода — предпоследний этап компиляции. Генерация кода может быть реализована многими способами, но чтобы простым образом получить портируемый компилятор, можно генерировать промежуточный код на низкоуровневом языке программирования, таком как C или еще ниже, например, LLVM.

Генерация кода должна переводить пользовательские функции в соответствующие функции целевого языка (для этого диалект должен быть расширен возможностью определения пользовательских функций).

Демонстрация генерации кода должна включать в себя программу на любом языке, использующую

Машина Тьюринга

Описание

Машина Тьюринга (МТ) — абстрактная вычислительная машина, предложенная Аланом Тьюрингом для формализации понятия алгоритма.

В состав машины Тьюринга входит неограниченная в обе стороны лента, разделённая на ячейки, и управляющее устройство, способное находиться в одном из конечного множества заранее определённых состояний.

Управляющее устройство может перемещаться влево и вправо по ленте, читать и записывать в ячейки символы некоторого конечного алфавита. Выделяется особый пустой символ, заполняющий все клетки ленты, кроме тех из них (конечного числа), на которых записаны входные данные.

МТ работает согласно правилам перехода. Каждое правило перехода предписывает машине, в зависимости от текущего состояния и наблюдаемого в текущей клетке символа, записать в эту клетку новый символ, перейти в новое состояние и переместиться на одну клетку влево или вправо, либо либо остаться на месте. Некоторые состояния машины Тьюринга могут быть помечены как терминальные, и переход в любое из них означает конец работы, остановку алгоритма. Результатом работы МТ является слово, записанное на ленте после остановки вычислителя.

В данном практическом задании предлагается реализовать систему, визуализирующую процесс вычислений МТ для произвольных программ и входных слов.

Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна включать:

- загрузку программы из файла;
- интерфейс для визуализации вычислений на МТ (лента, набор состояний, текущая конфигурация);
- возможность указать входное слово;
- возможность прервать/продолжить/остановить/начать вычисления заново.

Описание программы в виде файла должно также состоять из определения алфавитов и множества состояний, в которых выделены начальное и конечные состояния. Формат описания может быть как придуманным самостоятельно, так и одним из существующих языков разметки (например, JSON, YAML, TOML).

Расширенный интерфейс (индивидуальная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- настройки максимального количества шагов, скорости их отображения и т.д.;
- пошаговый режим;
- проверка корректности входных данных;
- и т.д.

Графический интерфейс для построения программ (индивидуальная часть)

Модуль должен включать в себя графический интерфейс для создания программ для МТ. У пользователя должна быть возможность:

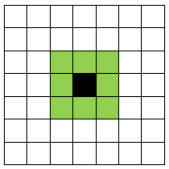
- указать используемый алфавит ленты и входных слов;
- построить таблицу переходов;
- обозначить начальное и конечные состояния;
- запустить программы сразу;
- сохранить программу в формате, воспринимаемом интерпретатором, для дальнейшей загрузки;

Игра «Жизнь»

Описание

Игра «Жизнь» — это клеточный автомат, придуманный английским математиком Джоном Конвеем в 1970 году.

Игра происходит во вселенной — плоскости или поверхности, разбитой на клетки. Каждая клетка в каждый момент времени может находиться в одном из двух состояний: быть «живой» или «мёртвой» (пустой). У каждой клетки есть соседи, определяемые окрестностью Мура.



Время в игре разбито на шаги, за каждый шаг все клетки одновременно обновляют своё состояние по следующим правилам:

- в пустой (мёртвой) клетке зарождается жизнь, если рядом находится ровно 3 живые клетки;
- если у живой клетки ровно 2 или 3 живых соседа, она остаётся жить; иначе она умирает (от одиночества или перенаселённости).

Игрок не принимает прямого участия в игре, а лишь расставляет или генерирует начальную конфигурацию живых клеток, которые затем взаимодействуют согласно правилам уже без его участия (он является наблюдателем).

В игре жизнь присутсвует множество интересных структур. Самым популярным, вероятно, является планер.



Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна содержать:

- визуализацию и моделирование заданной сцены;
- загрузку начального состояния сцены из файла.

Расширенный интерфейс (индивидуальная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- меню выбора сцены;
- меню выбора модификаций игры «Жизнь»;
- настройки поля-вселенной;
- редактор сцены с панелью заготовленных объектов;
- управление моделированием:
  - пауза/продолжение;
  - ускорение/замедление;
  - перемотка;
- интерфейс сохранения/загрузки;
- и т.д.

Работа с базой данных (индивидуальная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- база сцен;
- база объектов;
- и т.д.

Визуализация клеточных автоматов

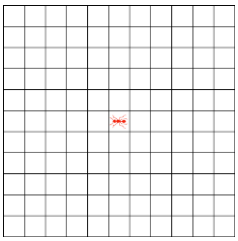
Описание

Клеточный автомат — дискретная модель, включающая в себя регулярную решётку ячеек, каждая из которых может находиться в одном из конечного множества состояний,

таких как 1 и 0. Решетка может быть любой размерности. Для каждой ячейки определено множество ячеек, называемых окрестностью. Для работы клеточного автомата требуется задание начального состояния всех ячеек и правил перехода ячеек из одного состояния в другое. На каждой итерации, используя правила перехода и состояния ячеек из окрестности, определяется новое состояние каждой ячейки.

В этом варианте задания предлагается реализовать приложение, визуализирующее эволюции различных клеточных автоматов из заранее выбранного набора. Этот набор может включать в себя такие автоматы, как:

- элементарные клеточные автоматы, например, правило 30 или правило 110;
- муравей Лэнгтона;
- черви Патерсона.



Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна содержать:

- выбор клеточного автомата из каталога с описанием (не менее 4 различных автоматов);
- визуализацию и моделирование фиксированной сцены для выбранного автомата;

Расширенный интерфейс (индивидуальная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- меню выбора сцены;
- интерфейс сохранения/загрузки;
- настройки поля-вселенной (размер поля, цвет клеток и т.п.);
- редактор сцены (указание начального состояния);
- управление моделированием:
  - пауза/продолжение;
  - ускорение/замедление;
  - перемотка;
- и т.д.

Работа с базой данных (индивидуальная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- база сцен;
- база объектов;
- и т.д.

Настольные игры

Описание

В данном задании предлагается реализовать любую из настольных игр (например, реверси, шашки, шахматы, нарды, го и пр.).

Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна:

- предоставлять возможность играть в игру в режиме «человек против человека», используя графический интерфейс;
- не допускать невозможных по правилам игры ходов игроков;
- определять момент победы или ничьей и демонстрировать пользователям результат.

Расширенный интерфейс (индивидуальная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- показ возможных ходов для выбранной фигуры/шашки;
- отмена хода/ходов;
- меню выбора режима игры:
  - человек против человека;
  - человек против компьютера;
  - компьютер против компьютера;
  - сетевая игра;
- меню настроек ИИ;
- меню выбора разновидности настольной игры (например, шашки с дамками или без);
- просмотр таблицы рекордов;
- проигрывание записанных игр;
- интерфейс сохранения/загрузки игр;
- и т.д.

Искусственный интеллект (индивидуальная часть)

Алгоритм, делающий ход, никак не оценивая ситуацию (например, случайный или первый доступный ход), не считается за реализацию искусственного интеллекта.

Реализация искусственного интеллекта должна предоставлять:

- настройки сложности;
- как минимум 2 различных стратегии (это может быть один алгоритм с разными эвристиками).

Клиент-серверная архитектура (индивидуальная часть)

Помимо возможности просто играть в настольную игру по сети (см. Минимальные требования), клиент-серверная архитектура должна предоставлять хотя бы 2 дополнительные возможности:

- поддержка нескольких игровых сессий одновременно;
- запуск ИИ на серверной стороне в качестве противника;
- регистрация, аутентификация и авторизация (вход в систему и права на доступ);
- доступ к таблице рекордов;
- сохранение/загрузка игр;
- и т.д.

Работа с базой данных (индивидуальная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- таблица рекордов;
- сохранения игровых сессий;
- база пользователей;
- и т.д.

Классические компьютерные игры

Описание

В этом варианте предлагается реализовать одну из "классических" компьютерных игр (например, змейка, тетрис, арканоид и т.п.).

Минимальные требования (базовая часть)

Базовая реализация проекта должна:

- предоставлять возможность играть в игру, используя графический интерфейс или консольные виджеты;
- не допускать невозможных по правилам игры действий игроков;
- определять момент завершения игры и демонстрировать пользователям результат.

Расширенный интерфейс (индивидуальная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Эти возможности во по большей части зависят от выбора конкретной игры. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- меню:
  - просмотр таблицы рекордов;
  - интерфейс сохранения/загрузки игр;
- расширения игры:
  - бонусные баллы;
  - выбор режима игры;
  - ограничение прохождения уровня по времени;
- и т.д.

Конструктор уровней (индивидуальная часть)

Конструктор уровней даёт игрокам возможность создавать свои собственные уровни, если это применимо к выбранной игре.

В модуле должно быть реализовано следующее:

- графический интерфейс для создания уровней;
- возможность сохранять уровень с названием и именем автора;
- возможность загрузить уровень для редактирования в конструкторе;
- пункт меню для выбора пользовательских уровней для игры.

Работа с базой данных (индивидуальная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- таблица рекордов;
- сохранения игровых сессий;
- база пользователей;
- и т.д.

Бесконечный платформер

Описание

Платформер — это жанр компьютерных игр, в которых основной чертой игрового процесса является прыгание по платформам, лазанье по лестницам, собирание предметов, обычно необходимых для завершения уровня.

Бесконечны платформер — это платформер, в котором персонаж бежит в одном направлении по миру, который генерируется по ходу движения. Таким образом, игровой мир потенциально бесконечен.

Основная цель игры — добежать как можно дальше.

В данном задании предлагается любую разновидность бесконечного платформера.



Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна:

- предоставлять простейшую генерацию игрового мира — повторение заданного шаблона;

- создавать игровой мир с возможность проиграть или двигаться дальше;
- предоставлять минимальное управление персонажем;
- определять момент поражения и подсчитывать пройденную дистанцию.

Расширенный интерфейс (индивидуальная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- меню:
  - просмотр таблицы рекордов;
  - интерфейс сохранения/загрузки игр;
  - выбор режимов игры:
    - однопользовательская/многопользовательская;
    - уровни сложности;
- расширения игры:
  - многопользовательский режим;
  - бонусы и специальные объекты, которые может подбирать персонаж;
  - различные препятствия;
  - отмотка времени;
- и т.д.

Генератор игрового мира (индивидуальная часть)

Генератор игрового мира должен постепенно повышать сложность трассы.

Для реализации различных режимов игры, генератор должен быть конфигурируемым. В частности, пользователь генератора должен иметь возможность

- ограничить объекты игрового мира, используемые для генерации;
- определить относительную сложность трассы, на которой может появляться заданный объект;
- определить частоту встречаемости объекта на трассе (в зависимости от сложности);
- и т.д.

Искусственный интеллект (индивидуальная часть)

Алгоритм поведения, никак не оценивающий ситуацию (например, случайное движение), не считается за реализацию искусственного интеллекта.

Искусственный интеллект в бесконечном платформере может управлять - активными противниками персонажа, встречающиеся на пути; - соперничающим персонажем (в многопользовательском режиме вместо второго игрока).

Реализация искусственного интеллекта должна предоставлять настройки сложности.

Клиент-серверная архитектура (индивидуальная часть)

Помимо возможности просто играть в игру по сети (см. Минимальные требования), клиент-серверная архитектура должна предоставлять хотя бы 2 дополнительные возможности:

- поддержка нескольких игровых сессий одновременно;
- запуск ИИ на серверной стороне;
- регистрация, аутентификация и авторизация (вход в систему и права на доступ);
- доступ к таблице рекордов;
- сохранение/загрузка игр;
- и т.д.

Работа с базой данных (индивидуальная часть)

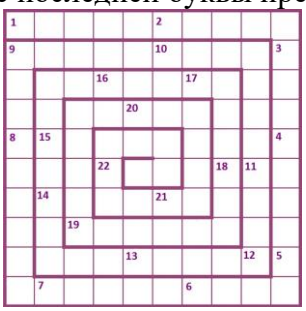
Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- таблица рекордов;
- сохранения игровых сессий;
- база пользователей;
- и т.д.

Чайнворд

Описание

Чайнворд — это разновидность кроссворда, где слова составляют единую цепочку. Каждое следующее слово начинается с последней буквы предыдущего слова.



Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна включать:

- интерфейс для разгадывания чайнворда (чайнворд + описания слов);
- загрузку головоломки из файла.

Расширенный интерфейс (индивидуальная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- показ верно угаданных слов;
- меню генерации чайнворда:
  - словарь;
  - сложность (например, количество слов);
  - вариант расположения (например, спираль, змейкой, лабиринтом, и т.д.)
- подсказки (показать одну букву);
- таймер решения головоломки;
- просмотр таблицы рекордов;
- интерфейс сохранения/загрузки игр;
- альтернативные способы отображения чайнворда (многоугольники, круги, спирали и пр.);
- редактор головоломок;
- и т.д.

Генерация головоломок (индивидуальная часть)

Генератор головоломок должен:

- использовать для генерации как минимум два параметра из трёх:
  - словари;
  - схемы расположения цепочки слов;
  - уровни сложности.
- по возможности, предоставлять равномерное распределение вероятности попадания слова из словаря в головоломку.

Выбор из нескольких заранее подготовленных головоломок не считается генерацией.

Работа с базой данных (индивидуальная часть)

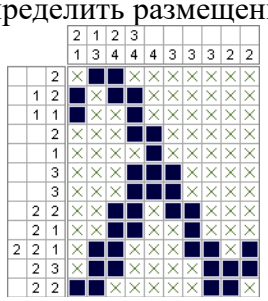
Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- таблица рекордов;
- сохранения игровых сессий;
- база головоломок;
- и т.д.

Японский кроссворд

Описание

Изображения зашифрованы числами, расположенными слева от строк, а также сверху над столбцами. Числа показывают, сколько групп чёрных (либо своего цвета, для цветных кроссвордов) клеток находятся в соответствующих строке или столбце и сколько слитных клеток содержит каждая из этих групп (например, набор чисел 4, 1, и 3 означает, что в этом ряду есть три группы: первая — из четырёх, вторая — из одной, третья — из трёх чёрных клеток). В чёрно-белом кроссворде группы должны быть разделены, как минимум, одной пустой клеткой, в цветном это правило касается только одноцветных групп, а разноцветные группы могут быть расположены вплотную (пустые клетки могут быть и по краям рядов). Необходимо определить размещение групп клеток.



Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна включать:

- интерфейс для разгадывания японского кроссворда;
- загрузку головоломки из файла.

Расширенный интерфейс (индивидуальная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- меню выбора головоломок;
- запуск автоматического решателя;
- таймер решения головоломки;
- просмотр таблицы рекордов и архива решенных головоломок;
- интерфейс сохранения/загрузки частичного решения;
- редактор головоломок;
- и т.д.

Генерация и автоматическое решение головоломок (индивидуальная часть)

В этой индивидуальной части требуется реализовать генерацию головоломок и автоматический решатель.

Генерация головоломок должна происходить по картинке, заданной в файле (например, псевдографикой в текстовом файле или в любом медиа-формате). После генерации головоломки, её сложность должна быть оценена при помощи автоматического решателя.

Автоматическое решение головоломки может быть реализовано несколькими способами. Чтобы использовать тот же алгоритм для оценки сложности головоломок, необходимо использовать стратегии, моделирующие решение человеком. При решении на каждом шаге пробуются стратегии, начиная с самой простой, и первая подходящая используется. Средняя сложность использованных стратегий может расцениваться как общая сложность головоломки.

Выбор из нескольких заранее подготовленных головоломок не считается генерацией.

Работа с базой данных (индивидуальная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- таблица рекордов;
- сохранения игровых сессий;
- база головоломок;

- и т.д.

Судоку

Описание

Игровое поле представляет собой квадрат размером 9×9, разделённый на меньшие квадраты со стороной в 3 клетки. Таким образом, всё игровое поле состоит из 81 клетки. В них уже в начале игры стоят некоторые числа (от 1 до 9), называемые подсказками. От игрока требуется заполнить свободные клетки цифрами от 1 до 9 так, чтобы в каждой строке, в каждом столбце и в каждом малом квадрате 3×3 каждая цифра встречалась бы только один раз.

Сложность судоку зависит не от количества изначально заполненных клеток, а от методов, которые нужно применять для её решения. Самые простые решаются дедуктивно: всегда есть хотя бы одна клетка, куда подходит только одно число. Некоторые головоломки можно решить за несколько минут, на другие можно потратить часы.

Правильно составленная головоломка имеет только одно решение.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна включать:

- интерфейс для разгадывания судоку;
- загрузку головоломки из файла.

Расширенный интерфейс (индивидуальная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- настройки сложности судоку;
- подсказки (например, показать, какие цифры можно поставить в данную клетку);
- запуск автоматического решателя;
- таймер решения головоломки;
- просмотр таблицы рекордов;
- интерфейс сохранения/загрузки игр;
- выбор разновидности головоломки;
- редактор головоломок;
- и т.д.

Генерация и автоматическое решение головоломок (индивидуальная часть)

В этой индивидуальная части требуется реализовать генерацию головоломок с заданной сложностью и автоматический решатель.

Простым способом генерации головоломки «Судоку» является случайная перестановка строк и столбцов и вычёркивание случайных клеток. Для генерации головоломок заданной сложности можно воспользоваться одним из двух вариантов:

- вычёркивать клетки по одной, каждый раз оценивая насколько игра усложняется (насколько трудно человеку будет проставить вычеркнутую цифру);
- генерировать головоломку, используя автоматический решатель.

Автоматическое решение головоломки может быть реализовано несколькими способами. Чтобы использовать тот же алгоритм для генерации головоломок заданной сложности, необходимо использовать стратегии, моделирующие решение человеком:

- при решении на каждом шаге пробуются стратегии, начиная с самой простой, и первая подходящая используется.

- при генерации на каждом шаге добавляется очередная цифра в поле, на основе одной из стратегий заданной сложности.

Выбор из нескольких заранее подготовленных головоломок не считается генерацией.

Работа с базой данных (индивидуальная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- таблица рекордов;
- сохранения игровых сессий;
- база головоломок;
- и т.д.

Лабиринты

Описание

В данном варианте предлагается реализовать систему генерации и прохождения двумерных лабиринтов.

Для генерации лабиринтов существует множество алгоритмов. Одним из самых простых для реализации на функциональном языке является алгоритм рекурсивного деления.

Работа алгоритма состоит в следующем:

1. В начале работы алгоритма в комнате нет стен.
2. Пространство комнаты разделяется стеной (одной или двумя перпендикулярными) в произвольном отношении.
3. В произвольном месте каждой стены выбирается проход.
4. Процесс рекурсивно повторяется для полученных комнат меньшего размера, пока размер комнаты не достигнет заранее установленного минимального размера.



Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна включать:

- генерацию лабиринта и визуализацию этого процесса;
- интерфейс для прохождения сгенерированного лабиринта (управление игроком);
- сохранение и загрузка лабиринтов из файла.

Выбор из нескольких заранее подготовленных лабиринтов не считается генерацией.

Расширенный интерфейс (индивидуальная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- настройки размеров лабиринта;
- выбор алгоритма генерации;
- таймер прохождения лабиринта;
- запуск автоматического решателя;
- подсказки (например, показать следующие несколько шагов);
- и т.д.

Автоматическое прохождение лабиринта (индивидуальная часть)

В этой индивидуальной части требуется реализовать автоматический решатель головоломки, который проходит лабиринт от текущего положения игрока до выхода из лабиринта и рисует полученный путь.

Можно использовать любой алгоритм нахождения пути в лабиринте или несколько из них, используя разные цвета для визуализации полученных путей.

Работа с базой данных (индивидуальная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- таблица рекордов;
- сохранения игровых сессий;
- база лабиринтов;
- и т.д.

Wordle

Описание

Wordle — это игра, где нужно угадать слово из пяти букв за шесть попыток.

Для начала нужно вписать любое слово в верхнюю строчку игрового поля. Если хотя бы одна буква в вашем слове соответствует загаданному, то она подсветится зелёным, когда стоит в нужном месте, и жёлтым, когда стоит не там, где надо. Если все ячейки вашего слова остаются серыми, значит в искомом слове вообще нет этих букв. Аналогичным образом подсвечиваются буквы на экранной клавиатуре.



Версия на английском и на русском.

В данном варианте предлагается реализовать эту игры. В оригинальной версии новое слово, которое нужно угадать, обновляется раз в сутки, для практического задания этого ограничения нет, слово должно произвольно выбираться из словаря при старте игры.

Набор возможных слов задаётся заранее, пользователь должен иметь возможность указать путь к файлу со словарём (или настройки подключения к базе данных со словарём при выборе соответствующей индивидуальной части).

Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна включать:

- интерфейс для отгадывания слова;
- загрузку словаря из файла.

Расширенный интерфейс (индивидуальная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- меню:
  - просмотр таблицы рекордов;

- интерфейс сохранения/загрузки игр;
- интерфейс для обновления загаданного слова или обнуления действий игрока;

- расширения игры:
  - подсчёт времени;
  - возможность выбрать размер поля;
  - возможность выбрать язык игры (должно поддерживаться как минимум 2 языка);
- интерфейс для работы со словарём;
- и т.д.

Работа с базой данных (индивидуальная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- таблица рекордов;
- сохранения игровых сессий;
- база пользователей;
- база слов;
- и т.д.

Kanban-доска

Описание

Kanban-доска - инструмент для управления проектами и визуализации рабочего процесса. Самые простые доски состоят из трех колонок: «сделать» (To Do), «в процессе» (In Progress) и «сделано» (Done). Каждая колонка заполняется карточками с описанием задач.



Новые задачи после создания помещаются в самую левую колонку. По мере обновления статуса задачи пользователи перемещают карточки из одной колонки в другую.

Карточка состоит из идентификатора (обычно числового) задачи, её заголовка и описания. У задачи может быть назначен исполнитель из заранее определённого списка пользователей.

Минимальные требования (базовая часть)

Базовая реализация проекта, в которой должны разбираться все участники, должна:

- визуализировать Kanban-доску как минимум с тремя фиксированными колонками;
- предоставлять интерфейс для создания новых задач, а также изменения и удаления существующих;
- предоставлять возможность переносить задачи между колонками, указывать приоритет и ответственного за задачу из заранее определённого списка пользователей.

Расширенный интерфейс (индивидуальная часть)

Расширенный интерфейс должен добавлять хотя бы 2 различные возможности к базовому интерфейсу. Ниже перечислены возможные варианты расширения интерфейса, однако этим списком они не ограничены:

- возможность конфигурировать колонки (добавлять новые, переименовывать и удалять имеющиеся);
- интерфейс для работы со списком пользователей;
- возможность выставлять дополнительные атрибуты карточек, например:

- дату дедлайна;
- тип задачи из определённого множества (баг/фича/вопрос/тестирование и т.п.);
- произвольные текстовые метки/теги (обычно указывают часть системы, к которой относится задача: backend, frontend, tests, integration, ...);
- и т.д.

Работа с базой данных (индивидуальная часть)

Модуль для работы с базой данных должен предоставлять хотя бы 2 различных возможности:

- сохранения и загрузка списка задач;
- сохранения и загрузка конфигурации колонок;
- база пользователей;
- и т.д.

5. Оценочные материалы промежуточной аттестации по дисциплине

5.1. Экзамен проводится с применением следующих методов (средств): проведение устного экзамена, включающего теоретическую часть и практико-ориентированное задание, требующее демонстрации уровня владения умениями/навыками.

5.2. Оценочные материалы промежуточной аттестации

Таблица 5.

Показатели и критерии оценивания

Компонент компетенции	Промежуточный индикатор оценивания	Критерий оценивания
ПКс-2.2 Способен устанавливать и настраивает программное обеспечение, необходимое для разработки и функционирования прикладного решения	ИПКс-2.2 Устанавливает и настраивает программное обеспечение, необходимое для разработки и функционирования прикладного решения	Разрабатывает и адаптирует программное обеспечение, необходимое для разработки и функционирования прикладного решения
ПКс-3.1 Способен выбирать и использовать современные технологии и программные средства проектирования информационных систем	ИПКс-3.1 Выбирает и использует современные технологии и программные средства проектирования информационных систем	Владеет навыками описания и разработки информационных систем по видам обеспечения

Перечень теоретических вопросов к экзамену

1. Декларативное и императивное программирование. Основные понятия.
2. Понятие функции. Синтаксис и аргументы..
3. λ-исчисление. Базовые понятия. Синтаксис и вычисление термов.
4. Каррирование и редукция. Стратегии редукции.
5. Символьное представление данных в языке программирования
6. Элементарные функции Работа со списками массивами.
7. Логические выражения и функции. Построение условных переходов в программных вычислениях.
8. Построение рекурсивных функций и процедур.
9. Функционалы и функции планирования.
10. Управляющие конструкции языка программирования.
11. Реализация методов поиска на языке программирования.
12. Процедурное и логическое программирование. Понятие логической программы.
13. Компиляторы и интерпретаторы. Функционал и особенности использования.
14. Отладка и тестирование программ.
15. Организация вывода результатов.

Перечень практико-ориентированных (практических) заданий к экзамену

ПКс-2.2 - способен устанавливать и настраивает программное обеспечение, необходимое для разработки и функционирования прикладного решения:

1. Составить программу с использованием конструкции «if-then-else». Адаптировать ее под различные начальные условия, вводимы с клавиатуры.

2.Разработайте программу для анализа и обобщения результатов опроса. Каждый вопрос имеет фиксированное количество ответов. Ответ одного человека представлен списком пар. Напишите функцию, которая по списку результатов опроса печатает таблицу ответов с процентами.

ПКс-3.1 - способен выбирать и использовать современные технологии и программные средства проектирования информационных систем:

3.Задан текстовый файл со списком имен и адресов электронной почты. Задан текстовый файл со списком адресов электронной почты и номеров ICQ. Создайте программу, которая на основе этих файлов создает web-страницу с гиперссылками на адреса электронной почты и выводит список в алфавитном порядке имен.

Шкала оценивания

Промежуточная аттестация проводится в форме экзамена без применения балльно-рейтинговой системы оценивания - посредством аттестационных испытаний в форме традиционного экзамена.

Результаты экзамена оцениваются по традиционной балльной системе оценивания («отлично», «хорошо», «удовлетворительно», «неудовлетворительно»), или традиционной балльной системе оценивания.

Наряду с оцениванием учебных достижений обучающихся по принятой образовательной организации системе оценок и использованием ее при оформлении официальных дипломов Академии о высшем образовании, вводят повышенный («Excellent» и «Good») и базовый («Very good» и «Satisfactory») уровень оценок «отлично» и «хорошо» для использования при оформлении ЕПД, согласно Положению об оформлении и выдаче ЕПД.

Таблица 6.

Соответствие академической оценке

Академическая оценка		Оценка по шкале ESTS	
зачтено	отлично	Passed	A (Excellent)
			B (Very Good)
	хорошо		C (Good)
			D (Satisfactory)
	удовлетворительно		E (Sufficient)
не зачтено	неудовлетворительно	-	-

Критерии оценивания в ходе промежуточной аттестации

Оценка **«отлично»** выставляется обучающемуся:

– если он глубоко и прочно усвоил программный материал, исчерпывающе, последовательно, четко и логически стройно излагает его на экзамене, умеет тесно увязывать теорию с практикой, свободно справляется с задачами, вопросами и другими видами знаний, причем не затрудняется с ответом при видоизменении заданий, правильно обосновывает принятое решение; учебные достижения и результаты текущего контроля успеваемости демонстрируют высокую степень сформированности соответствующих компетенций.

Оценка **«хорошо»** выставляется обучающемуся:

– если он твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопрос, правильно применяет теоретические положения при решении практических вопросов и задач, владеет необходимыми навыками и приемами их выполнения; учебные достижения и результаты текущего контроля успеваемости демонстрируют хорошую степень сформированности соответствующих компетенций.

Оценка **«удовлетворительно»** выставляется обучающемуся:

– если он имеет знания только основного материала, но не усвоил его деталей, допускает неточности, недостаточно правильные формулировки, нарушения логической последовательности в изложении программного материала, испытывает затруднения при выполнении практических работ; учебные достижения и результаты текущего контроля успеваемости демонстрируют удовлетворительную степень сформированности соответствующих компетенций.

Оценка **«неудовлетворительно»** выставляется обучающемуся:

– если он не знает значительной части программного материала, допускает существенные ошибки, неуверенно, с большими затруднениями выполняет практические работы; учебные достижения и результатами текущего контроля успеваемости демонстрируют низкую степень сформированности соответствующих компетенций.

6. Методические материалы по освоению дисциплины с применением ЭК

Обучение по дисциплине «Основы функционального программирования» предполагает изучение курса на аудиторных занятиях (лекции, практические) и самостоятельной работы студентов. Практические занятия дисциплины «основы функционального программирования» предполагают их проведение в различных формах с целью выявления полученных знаний, умений, навыков и компетенций.

Подготовка к лекции студентами заключается в следующем:

– повторить материал предыдущей лекции, прочитав его повторно;

– ознакомиться с темой предстоящей лекции (в рабочей программе учебной дисциплины);

– ознакомиться с учебными материалами по данной теме в соответствии с предложенным списком литературы в рабочей программе учебной дисциплины или с электронными материалами, предложенными лектором;

– записать возможные вопросы, которые можно будет задать лектору.

Подготовка к практическим (семинарским) занятиям:

– внимательно прочитать материал лекций, относящихся к данному занятию, ознакомиться с учебными материалами, включая электронные в соответствии с предложенным списком литературы в рабочей программе учебной дисциплины;

– подготовить развернутые ответы на вопросы, предложенные в рабочей программе дисциплины для обсуждения;

– выполнить задания, если они предусмотрены в письменной форме;

– понять, что для вас осталось неясными и постараться получить на них ответ заранее;

– готовиться к практическим/семинарским занятиям можно как индивидуально, так и в составе малой группы;

– рабочую программу учебной дисциплины необходимо использовать в качестве основного ориентира в организации обучения;

– электронная версия рабочей программы по дисциплине размещена на сайте Нижегородского института управления, к ней предоставлен авторизованный доступ.

Самостоятельная работа обучающегося:

Самостоятельная работа призвана закрепить теоретические знания и практические навыки, полученные на лекциях, практических и семинарских занятиях.

Самостоятельная работа проводится с целью:

- систематизации и закрепления полученных теоретических знаний и практических умений, обучающихся;
- углубления и расширения теоретических знаний;

- формирования умений использовать справочную документацию и специальную литературу;
- развития познавательных способностей и активности: творческой инициативы, самостоятельности, ответственности и организованности;
- формирования самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развития исследовательских умений.

Эффективность лекционных, семинарских и практических занятий по дисциплине во многом зависит от качества самостоятельной работы обучающихся, от их самоподготовки.

Часть времени, отведенного на самостоятельную работу должна использоваться на подготовку к аудиторным занятиям, другая часть на выполнение домашней работы, осмысление и оформление результатов практических занятий.

При подготовке к занятиям студенту полезно:

- изучить теоретический материал по данной теме (конспект занятия);
- ознакомиться с литературой, рекомендованной преподавателем;
- выполнить задания, предложенные преподавателем, к занятию;
- составить перечень вопросов, вызывающих затруднения, неясности или сомнения, обсудить их с преподавателем или на занятии;
- заниматься самостоятельным поиском дополнительной литературы по изучаемой теме.

Подготовка к экзамену. К экзамену необходимо готовиться целенаправленно, регулярно, систематически и с первых дней обучения по данной дисциплине. В самом начале учебного курса познакомьтесь со следующей учебно-методической документацией:

- программой дисциплины;
- перечнем знаний и умений, которыми студент должен владеть;
- тематическими планами лекций, семинарских занятий;
- учебником, учебными пособиями по дисциплине, а также электронными ресурсами;
- перечнем и тематикой письменных работ, а также методическими рекомендациями по их выполнению;
- перечнем вопросов.

Систематическое выполнение всех видов заданий на лекциях, практических занятиях, а также самостоятельная работа позволит успешно освоить дисциплину и создать хорошую базу для сдачи экзамена**.**

Вопросы (задания) для самостоятельной подготовки к занятиям

1. Написать программу формирования списка, содержащего информацию о количестве элементов-списков на каждом уровне вложенности: ((<уровень><количество элементов-списков>) ...).
2. Дан список. Написать программу, возвращающую максимальную глубину для входящих в список элементов-списков.
3. Написать программу, генерирующую все циклические перестановки списка. Элементами списка являются списки. Пример : ((a b)(c d)) ==>(((a b)(c d))((b a)(c d))((a b)(d c)) ..).
4. Описать функцию, которая находила бы сумму всех числовых элементов в списке. Список может содержать подсписки произвольной глубины.
5. Написать программу, которая, выдавала бы элемент списка по заданному номеру, считая от конца.

7. Учебная литература и ресурсы информационно-телекоммуникационной сети Интернет

7.1 Основная литература

1. Козырева Г.Ф. Функциональное и логическое программирование : учебно-методическое пособие / Козырева Г.Ф.. — Саратов : Ай Пи Эр Медиа, 2018. — 120 c. — ISBN 978-5-4486-0122-4. — Текст : электронный // IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/71596.html (дата обращения: 18.03.2024). — Режим доступа: для авторизир. пользователей. - DOI: https://doi.org/10.23682/71596.

2. Зыков, С. В. Программирование. Функциональный подход : учебник и практикум для вузов / С. В. Зыков. — 2-е изд. — Москва : Издательство Юрайт, 2024. — 150 с. — (Высшее образование). — ISBN 978-5-534-16942-3. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/537721 (дата обращения: 25.04.2024). — Режим доступа: для авторизир. пользователей.

3. Кубенский, А. А. Функциональное программирование : учебник и практикум для вузов / А. А. Кубенский. — Москва : Издательство Юрайт, 2024. — 348 с. — (Высшее образование). — ISBN 978-5-9916-9242-7. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/536996 (дата обращения: 25.04.2024). — Режим доступа: для авторизир. пользователей.

7.2 Дополнительная литература

1. Душкин Р.В. Функциональное программирование на языке Haskell / Душкин Р.В.. — Саратов : Профобразование, 2017. — 608 c. — ISBN 978-5-4488-0044-3. — Текст : электронный // IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/64053.html (дата обращения: 18.03.2023). — Режим доступа: для авторизир. пользователей.

2. Сузи Р.А. Язык программирования Python : учебное пособие / Сузи Р.А.. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 350 c. — ISBN 978-5-4497-0705-5. — Текст : электронный // IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/97589.html (дата обращения: 18.03.2024). — Режим доступа: для авторизир. пользователей.

7.3 Учебно-методическое обеспечение самостоятельной работы.

1. Электронные информационно-образовательные ресурсы Нижегородского института управления – филиала РАНХиГС - http://sdo.niu.ranepa.ru

2. Федоров, Д. Ю. Программирование на языке высокого уровня Python : учебное пособие для вузов / Д. Ю. Федоров. — 5-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2024. — 227 с. — (Высшее образование). — ISBN 978-5-534-17323-9. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/539651 (дата обращения: 25.04.2024).— Режим доступа: для авторизир. пользователей.

7.4 Нормативные правовые документы.

1.Федеральный закон от 29 декабря 2012 г. № 273-ФЗ "Об образовании в Российской Федерации"

2.Федеральный закон от 27 июля 2006 г. N 149-ФЗ "Об информации, информационных технологиях и о защите информации" (с изменениями и дополнениями)

3.Федеральный закон от 27 июля 2006 г. N 152-ФЗ "О персональных данных" (с изменениями и дополнениями)

4. Приказ Министерства образования и науки РФ от 05.04.2017 №301 «Об утверждении Порядка организации и осуществления образовательной деятельности по образовательным программам высшего образования - программам бакалавриата, программам специалитета, программам магистратуры»

5. Образовательный стандарт ФГБОУ ВО «Российской академии народного хозяйства и государственной службы при Президенте Российской Федерации»,

утвержденный приказом ректора Академии от 07.09.2023 № 01-24607 и разработанного на основе федерального государственного образовательного стандарта высшего образования – бакалавриат по направлению подготовки 09.03.03 Прикладная информатика, утвержденного приказом Министерства образования и науки Российской Федерации от 19.09.2017 №922 (зарегистрировано в Министерстве юстиции Российской Федерации 12.10.2017, регистрационный номер № 48531) и приказа Минобрнауки России от 26 ноября 2020 г. № 1456 «О внесении изменений в федеральные государственные образовательные стандарты высшего образования» (зарегистрировано в Минюсте России 27.05.2021 г., регистрационный № 63650).

7.5. Интернет-ресурсы.

1. Городняя Л.В., Березин Н.А. Введение в программирование на Лиспе https://www.iprbookshop.ru/101999.html
2. Антон Холомьёв. Учебник по Haskell. https://anton-k.github.io/ru-haskell-book/files/ru-haskell-book.pdf
3. Основы работы в OpenOffice. Web: http://www.intuit.ru/department/office/OpenOfficebasics .
4. Основы работы с Яндекс. Web: http://www.intuit.ru/department/internet/bwyandex/ .
5. Работа в Интернет с Microsoft Internet Explorer. Web: http://www.intuit.ru/department/internet/internetcert .
6. Современные офисные приложения. Web: http://www.intuit.ru/department/office/odpr/ .

8. Материально-техническая база, информационные технологии, программное обеспечение и информационные справочные системы

8.1. Материально-техническая база

Перечень материально-технического обеспечения:

1. Учебные аудитории, оборудованные для проведения учебных занятий лекционного и семинарского типа, групповых и индивидуальных консультаций, коллоквиумов, мероприятий текущего контроля и промежуточной аттестации, в том числе мультимедийным оборудованием для демонстрации электронных презентаций и аудио- и видеоматериалов.
2. Компьютерные классы для выполнения групповых тестовых и иных заданий, а также для самостоятельной работы обучающихся оснащенные компьютерной техникой и обеспечением доступа к сети «Интернет» и доступа в электронную информационно-образовательную среду организации.
3. Специализированные аудитории и лаборатории.
4. Библиотека с обеспечением печатными изданиями или электронно-библиотечная система обеспечивающая доступ к электронным изданиям (электронная библиотека).
5. Читальный зал.
6. Технические средства обучения: персональные компьютеры; компьютерные проекторы; звуковые динамики; программные средства, обеспечивающие просмотр видеофайлов в форматах AVI, MPEG-4, DivX, RMVB, WMV и др.

8.2. Информационные технологии, программное обеспечение:

При осуществлении образовательного процесса по дисциплине используется информационные технологии и программное обеспечение:

1. Современная операционная система.
2. Kaspersky Endpoint Security (или аналог).
3. Средство просмотра файлов формата pdf.
4. Современные офисные средства (текстовые и табличные редакторы, средства работы с презентационными материалами и т.д.).
5. Архиватор 7-Zip.
6. Система дистанционного обучения.
7. Автоматизированная библиотечная система.
8. СУБД PostgreSQL.
9. Lucidchart.
10. Eclipse
11. Система ADVANTA для управления проектами.

8.3. Информационные справочные системы:

1. https://www.urait.ru –Электронно-библиотечная система [ЭБС] Юрайт;
2. http://www.iprbookshop.ru – Электронно-библиотечная система [ЭБС] «IPR SMART» (ранее – IPRBooks)

3. https://e.lanbook.com - Электронно-библиотечная система [ЭБС] «Лань».
4. https://znanium.com - Электронно-библиотечная система [ЭБС] «Znanium.com».
5. https://www.book.ru - Электронно-библиотечная система [ЭБС] «Book.ru».
6. https://ibooks.ru - Электронно-библиотечная система [ЭБС] «ibooks.ru».
7. https://www.ezproxy.ranepa.ru:2443/login?url=https://dlib.eastview.com/ – East View. Полные тексты российских научных и практических журналов, а так же газет центральной прессы России.
8. https://www.ezproxy.ranepa.ru:2443/login?url=https://grebennikon.ru/ - Полные тексты 38 научно-практических журналов по маркетингу, менеджменту, финансам и управлению персоналом ИД «Гребенников»
9. http://www.consultant.ru/ - Справочно-правовая система «Консультант».
10. http://www.garant.ru/ Справочно-правовая система «Гарант».