Отчет по лабораторным работам

Дата: 15-12-2025
Семестр: 2 курс, 1 семестр
Группа: ПИН-Б-О-24-2
Дисциплина: Технологии программирования **Студент:** Осипов Александр Сергеевич

Цель работы

lab01: Введение в Haskell

Ознакомиться с основами функционального программирования, изучить базовый синтаксис Haskell, освоить основные концепции: чистые функции, иммутабельность, рекурсию, pattern matching и работу со списками.

lab02: Python и функциональные возможности

Изучить возможности функционального программирования в Python, освоить работу с функциями как объектами первого класса, изучить встроенные функции высшего порядка и научиться применять функциональные подходы в императивном языке.

lab03: JavaScript и современный фронтенд

Изучить применение функционального программирования в JavaScript, освоить современные подходы к разработке фронтенда с использованием функциональных концепций, изучить работу с асинхронными операциями в функциональном стиле.

lab04: Scala и большие данные

Изучить применение функционального программирования в Scala, освоить работу с коллекциями, option-типами и монадическими операциями. Изучить применение ФП в контексте обработки больших данных.

lab05: Rust и системное программирование

Изучить применение функционального программирования в Rust, освоить работу с системой владения, заимствования и временами жизни. Изучить функциональные подходы в контексте низкоуровневого и безопасного программирования.

lab06: Сравнительный анализ функционального программирования

Провести сравнительный анализ реализации концепций функционального программирования в изученных языках (Haskell, Python, JavaScript, Scala, Rust). Выявить сильные и слабые стороны каждого языка для решения практических задач в функциональном стиле.

Теоретическая часть

lab01: Введение в Haskell

Функциональное программирование - парадигма, где процесс вычисления рассматривается как вычисление значений функций. Основные понятия: - Чистые функции - для одинаковых входных данных всегда возвращают одинаковый результат - Иммутабельность - данные не изменяются после создания - Рекурсия - основной способ организации циклов - Pattern matching - сопоставление с образцом для декомпозиции данных - Функции высшего порядка - функции, принимающие другие функции как аргументы

lab02: Python и функциональные возможности

Python поддерживает функциональную парадигму, хотя не является чисто функциональным языком. Основные концепции: - Функции как объекты первого класса - можно присваивать переменным, передавать как аргументы - Lambda-функции - анонимные функции в одной строке - Замыкания - функции, запоминающие окружение создания - Декораторы - функции, модифицирующие поведение других функций - Генераторы - функции, возвращающие итератор

lab03: JavaScript и современный фронтенд

JavaScript поддерживает функциональную парадигму. ES6+ добавил множество возможностей для работы в функциональном стиле: - Функции первого класса - функции являются объектами - Стрелочные функции - компактный синтаксис - Неизменяемость - создание новых объектов вместо изменения существующих - Чистые функции - функции без побочных эффектов - Композиция функций - объединение простых функций

lab04: Scala и большие данные

Scala сочетает объектно-ориентированное и функциональное программирование, работая на JVM. Основные концепции: - Иммутабельность по умолчанию - val вместо var - Case classes - для создания неизменяемых данных - Pattern matching - мощный механизм декомпозиции данных - For-comprehensions - синтаксический сахар для работы с монадами - Type inference - компилятор автоматически выводит типы

lab05: Rust и системное программирование

Rust сочетает производительность системных языков с безопасностью и выразительностью функциональных языков. Основные концепции: - Иммутабельность по умолчанию - переменные неизменяемы без `mut` - Система владения - гарантии безопасности памяти без сборщика мусора - Трейты - аналоги type classes из Haskell - Алгебраические типы данных - enum с данными - Сопоставление с образцом - exhaustive pattern matching

lab06: Сравнительный анализ функционального программирования

Каждый язык имеет свои особенности реализации ФП: - **Haskell** - чисто функциональный, строгая типизация - **Python** - мультипарадигмальный с поддержкой ФП - **JavaScript** - ФП в веб-разработке - **Scala** - гибридный язык на JVM - **Rust** - системное ФП с гарантиями безопасности

Практическая часть

Выполненные задачи

lab01

- [x] Задача 1: Изучен базовый синтаксис Haskell и система типов
- [x] Задача 2: Освоено создание чистых функций и использование рекурсии
- [x] Задача 3: Научился работать со списками и pattern matching
- [x] Задача 4: Изучены основные функции высшего порядка
- [x] Задача 5: Освоена работа с кортежами и алгебраическими типами данных
- [x] Задача 6: Выполнены практические задания

lab02

- [x] Задача 1: Изучены функции как объекты первого класса в Python
- [x] Задача 2: Освоены встроенные функции высшего порядка: map, filter, reduce
- [x] Задача 3: Научился использовать lambda-функции и замыкания
- [x] Задача 4: Изучены генераторы и списковые включения
- [x] Задача 5: Освоено создание и использование декораторов
- [x] Задача 6: Выполнены практические задания

lab03

- [x] Задача 1: Изучены функции высшего порядка в JavaScript
- [x] Задача 2: Освоены методы работы с массивами: map, filter, reduce
- [x] Задача 3: Научился использовать стрелочные функции и замыкания
- [x] Задача 4: Изучены иммутабельные обновления
- [x] Задача 5: Выполнены практические задания

lab04

- [x] Задача 1: Изучен базовый синтаксис Scala и система типов
- [x] Задача 2: Освоена работа с коллекциями и функциями высшего порядка
- [x] Задача 3: Научился использовать option-типы и Either для обработки ошибок
- [x] Задача 4: Изучены case classes и pattern matching
- [x] Задача 5: Освоены базовые принципы работы с Apache Spark

lab05

- [x] Задача 1: Изучен базовый синтаксис Rust и система типов

- [x] Задача 2: Освоена система владения и заимствования
- [x] Задача 3: Научился работать с итераторами и замыканиями
- [x] Задача 4: Изучены алгебраические типы данных и pattern matching
- [x] Задача 5: Освоена обработка ошибок с помощью Result и Option

lab06

- [x] Задача 1: Реализована одинаковая задача на всех пяти языках
- [x] Задача 2: Сравнен синтаксис и выразительность кода
- [x] Задача 3: Проанализирована производительность решений
- [x] Задача 4: Оценена безопасность типов и надежность
- [x] Задача 5: Сформулированы рекомендации по выбору языка

Ключевые фрагменты кода

lab01: Введение в Haskell

Рекурсивный факториал:

```
factorial :: Integer -> Integer
factorial 0 = 1
factorial n = n * factorial (n - 1)
```

Функция высшего порядка map:

```
map' :: (a -> b) -> [a] -> [b]
map' _ [] = []
map' f (x:xs) = f x : map' f xs
```

Алгебраический тип данных:

```
data Point = Point Double Double
    deriving (Show)

distance :: Point -> Point -> Double
distance (Point x1 y1) (Point x2 y2) = sqrt ((x2 - x1)^2 + (y2 - y1)^2)
```

Практическое задание - подсчет четных чисел:

```
countEven :: [Int] -> Int
countEven [] = 0
countEven (x:xs)
    | even x    = 1 + countEven xs
    | otherwise = countEven xs
```

lab02: Python и функциональные возможности

Функции как объекты:

```
def apply_function(func, value):
    return func(value)

result = apply_function(square, 5)
```

Lambda и замыкания:

```
def create_counter():
    count = 0
    def counter():
        nonlocal count
        count += 1
        return count
    return counter
```

Функции высшего порядка:

```
student_names = list(map(lambda student: student['name'], students))
top_students = list(filter(lambda student: student['grade'] >= 90, students))
product = reduce(lambda x, y: x * y, numbers)
```

Декоратор:

```
def timer(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        print(f"Время: {time.time() - start_time:.4f} сек")
        return result
    return wrapper
```

lab03: JavaScript и современный фронтенд

Методы массивов:

```
const result = products
    .filter(product => product.inStock)
    .map(product => ({
        name: product.name.toUpperCase(),
        price: product.price
    }))
    .reduce((total, product) => total + product.price, 0);
```

Замыкания:

```
const createCounter = () => {
    let count = 0;
    return {
        increment: () => ++count,
        getCount: () => count
    };
};
```

Иммутабельные обновления:

```
const updatedUser = {
    ...user,
    name: 'Jane Doe',
    preferences: {
        ...user.preferences,
        theme: 'light'
    }
};
```

Дебаунсинг:

```
const debounce = (func, delay) => {
    let timeoutId;
    return (...args) => {
        clearTimeout(timeoutId);
        timeoutId = setTimeout(() => func.apply(null, args), delay);
    };
};
```

lab04: Scala и большие данные

Функции высшего порядка:

```
 val square: Int => Int = (x: Int) => x * x
def applyFunction(f: Int => Int, x: Int): Int = f(x)
```

Хвостовая рекурсия:

```
 @annotation.tailrec
def loop(acc: Int, n: Int): Int = {
  if (n <= 1) acc
  else loop(acc * n, n - 1)
}
```

lab05: Rust и системное программирование

Система владения:

```
 fn calculate_length(s: &String) -> usize {
    s.len()  // Заимствование без передачи владения
}
```

Функции высшего порядка:

```
 fn apply_function<F>(f: F, x: i32) -> i32
where
    F: Fn(i32) -> i32,
{
    f(x)
}
```

lab06: Сравнительный анализ функционального программирования

Результаты выполнения

Пример работы программы

lab01

```
 === Демонстрация работы функций ===

--- Базовые функции ---
25
"Good"

--- Рекурсия ---
120
15
13

--- Pattern matching ---
(4.0,6.0)
10
"Long list"

--- Функции высшего порядка ---
[1,4,9,16]
[2,4,6]
15

--- Алгебраические типы ---
5.0
True
False

--- Практические задания ---
3
[1,4,9]
[1,1,2,3,4,5,6,9]
```

lab02

```
 === Демонстрация функционального программирования в Python ===

1. Функции как объекты:
apply_function(square, 5) = 25

2. Lambda и замыкания:
Счетчик: 1, 2, 3

3. Функции высшего порядка:
Произведение чисел: 3628800
```

```
4. Генераторы и включения:
Четные квадраты: [4, 16, 36, 64, 100]

5. Декораторы:
Привет, Мария!
Привет, Мария!
Привет, Мария!
```

lab03

```
 Названия продуктов: ['iPhone', 'MacBook', 'T-shirt', 'Jeans', 'Book']
Доступные продукты: 3
Общая стоимость: 3121
Сумма доступных продуктов: 1107
Counter: 1, 2
Composed result: 20
```

lab04

```
 Квадрат 5: 25
Сложение 3 и 4: 7
Применение функции: 9
Удвоение 7: 14
Факториал 5: 120
Факториал хвостовой 5: 120
```

lab05

```
 Квадрат 5: 25
Сложение 3 и 4: 7
Применение функции: 9
Удвоение 7: 14
=== Система владения ===
s2 = hello
s2 = hello, s3 = hello
Длина 'hello' = 5
После модификации: hello, world!
```

lab06

Выводы по языкам

Haskell - максимальная выразительность и безопасность типов, но высокая кривая обучения. Идеален для академических задач.

Python - простота использования, огромная экосистема, но динамическая типизация. Отличен для прототипирования и Data Science.

JavaScript - доминирует в веб-разработке, хорошая поддержка ФП, но слабая типизация. Необходим для фронтенда.

Scala - баланс между выразительностью и производительностью, отличен для Big Data через Spark.

Rust - максимальная производительность и безопасность, но сложный синтаксис. Идеален для системного программирования.

Тестирование

lab01

- [x] Модульные тесты пройдены
- [x] Интеграционные тесты пройдены
- [x] Производительность соответствует требованиям

lab02

- [x] Модульные тесты пройдены
- [x] Интеграционные тесты пройдены
- [x] Производительность соответствует требованиям

lab03

- [x] Модульные тесты пройдены
- [x] Интеграционные тесты пройдены
- [x] Производительность соответствует требованиям

Выводы

Общие выводы по всем лабораторным работам

lab01

1. Haskell предоставляет мощную систему типов для безопасного программирования
2. Рекурсия является естественным способом организации циклов в функциональном программировании
3. Функции высшего порядка позволяют создавать абстрактные и переиспользуемые компоненты
4. Pattern matching упрощает работу со сложными структурами данных
5. Алгебраические типы данных дают возможность создавать выразительные модели данных

lab02

1. Python поддерживает основные концепции ФП, хотя и не является чисто функциональным языком
2. Функции высшего порядка делают код более декларативным и читаемым
3. Генераторы эффективны для работы с большими объемами данных
4. Декораторы позволяют добавлять функциональность без изменения исходного кода
5. Lambda-функции удобны для коротких операций, но могут ухудшать читаемость при усложнении

lab03

1. Современный JavaScript предоставляет мощные инструменты для ФП
2. Иммутабельность критически важна для предсказуемости состояния
3. Методы массивов делают код декларативным и читаемым
4. Замыкания позволяют создавать функции с состоянием
5. Дебаунсинг и другие утилиты улучшают производительность

lab04

1. Scala эффективно сочетает ООП и ФП парадигмы
2. For-comprehensions делают код с монадами читаемым
3. Система типов Scala помогает предотвращать ошибки на этапе компиляции

4. Хвостовая рекурсия оптимизируется компилятором
5. Scala отлично подходит для работы с большими данными через Spark

lab05

1. Система владения Rust обеспечивает безопасность без сборщика мусора
2. Итераторы в Rust эффективны благодаря нулевой стоимости абстракций
3. Pattern matching с enum мощнее, чем в большинстве языков
4. Компилятор предотвращает множество ошибок на этапе компиляции
5. Rust отлично подходит для системного программирования с гарантиями безопасности

lab06

1. Каждый язык имеет свои сильные стороны и оптимальные области применения
2. Выбор языка должен основываться на требованиях проекта, команды и экосистемы
3. Функциональное программирование доступно в разных формах в каждом языке
4. Безопасность типов важна для больших проектов
5. Производительность и простота использования часто противоречат друг другу

Ответы на контрольные вопросы

lab01

1. **Что такое чистая функция и каковы ее свойства?** - Чистая функция для одинаковых входных данных всегда возвращает одинаковый результат и не имеет побочных эффектов. Свойства: детерминированность, отсутствие побочных эффектов, возможность мемоизации.

2. **Чем рекурсия в Haskell отличается от итераций в императивных языках?** - В Haskell рекурсия оптимизируется через хвостовую рекурсию и ленивые вычисления. Нет изменяемых переменных, вместо этого создаются новые значения.

3. **Как работает pattern matching и для чего он используется?** - Pattern matching позволяет декомпозировать данные по образцу. Используется для работы с кортежами, списками, алгебраическими типами данных.

4. **Что такое функции высшего порядка? Приведите примеры.** - Функции, которые принимают другие функции в качестве аргументов или возвращают их. Примеры: map, filter, foldl.

5. **Какие преимущества дает статическая типизация в Haskell?** - Обнаружение ошибок на этапе компиляции, автоматический вывод типов, документация через типы, оптимизация компилятором.

lab02

1. **Что такое функции первого класса и как Python их поддерживает?** - Функции первого класса могут быть присвоены переменным, переданы как аргументы и возвращены из других функций. Python полностью поддерживает это.

2. **В чем разница между lambda-функциями и обычными функциями?** - Lambda-функции анонимны, ограничены одним выражением, не могут содержать операторы. Обычные функции могут содержать множественные операторы и имеют имя.

3. **Как работают замыкания и для чего они используются?** - Замыкания запоминают переменные из внешней области видимости. Используются для создания функций с состоянием, фабрик функций.

4. **Что такое декораторы и как создавать декораторы с параметрами?** - Декораторы - функции, модифицирующие другие функции. Декораторы с параметрами требуют тройной вложенности: внешняя функция принимает параметры, средняя - декорируемую функцию, внутренняя - обертку.

5. **Какие преимущества дают генераторы по сравнению со списками?** - Генераторы ленивы, экономят память, вычисляют значения по требованию. Списки создаются полностью в памяти сразу.

lab03

1. **Какие методы массивов в JavaScript относятся к функциям высшего порядка?** - map, filter, reduce, forEach, some, every, find, sort - все принимают функции как аргументы.

2. **Как работают замыкания и для чего они используются в React?** - Замыкания запоминают переменные из внешней области. В React используются для создания хуков, мемоизации, обработчиков событий.

3. **Что такое иммутабельность и почему она важна в React?** - Иммутабельность - создание новых объектов вместо изменения существующих. Важна для корректной работы React, определения изменений, оптимизации рендеринга.

4. **Как useMemo и useCallback помогают оптимизировать производительность?** - useMemo мемоизирует результаты вычислений, useCallback мемоизирует функции. Предотвращают ненужные пересчеты и рендеры.

5. **Какие преимущества дают функциональные компоненты перед классовыми?** - Меньше кода, проще тестировать, лучше производительность, хуки дают больше гибкости, проще понимать и поддерживать.

lab04

1. **Чем отличается var от val в Scala?** - val создает неизменяемую переменную, var - изменяемую. Рекомендуется использовать val для иммутабельности.

2. **Как работают for-comprehensions?** - For-comprehensions - синтаксический сахар для работы с монадами (Option, Either, List). Компилятор преобразует их в цепочку map/flatMap.

3. **В чем преимущество Option и Either?** - Явная обработка ошибок на уровне типов, компилятор заставляет обрабатывать все случаи, нет неожиданных исключений.

4. **Как pattern matching помогает в ФП?** - Позволяет декомпозировать данные по образцу, делает код декларативным и безопасным.

5. **Какие преимущества дает ФП в Apache Spark?** - Ленивые вычисления, распределенные преобразования, композиция операций, безопасность типов.

lab05

1. **Как система владения обеспечивает безопасность памяти?** - Компилятор отслеживает владение значениями, предотвращает использование после перемещения, гарантирует отсутствие гонок данных.

2. **В чем разница между `&T` и `&mut T`?** - `&T` - неизменяемая ссылка (можно иметь много), `&mut T` - изменяемая ссылка (может быть только одна).

3. **Как работают ленивые итераторы в Rust?** - Итераторы вычисляют значения по требованию, не создавая промежуточных коллекций. Цепочки преобразований оптимизируются компилятором.

4. **Какие преимущества дают алгебраические типы данных?** - Явное моделирование всех возможных состояний, компилятор заставляет обрабатывать все случаи, безопасность типов.

5. **Почему Rust предпочитает Result и Option вместо исключений?** - Явная обработка ошибок, нет скрытых исключений, компилятор заставляет обрабатывать ошибки, нулевая стоимость абстракций.

lab06

1. **Какой язык оказался наиболее выразительным?** - Haskell и Scala показали наибольшую выразительность благодаря мощным абстракциям.

2. **Какие компромиссы между безопасностью типов и продуктивностью?** - Статическая типизация замедляет разработку, но предотвращает ошибки. Динамическая типизация быстрее, но риск ошибок выше.

3. **Как разные языки решают проблему побочных эффектов?** - Haskell изолирует через IO монаду, Rust через систему владения, Python/JS полагаются на дисциплину программиста.

4. **Какой язык для high-performance приложения?** - Rust для максимальной производительности, Scala для Big Data, Haskell для математических вычислений.

5. **Какие особенности Rust делают его уникальным?** - Система владения обеспечивает безопасность памяти без сборщика мусора, нулевая стоимость абстракций, гарантии на этапе компиляции.

Структура проекта

```
.
├── lab01/
│   ├── project/
│   ├── report/
│   │   └── report.md
│   └── task.md
├── lab02/
│   ├── project/
│   ├── report/
│   │   └── report.md
│   └── task.md
├── lab03/
│   ├── project/
│   ├── report/
│   │   └── report.md
│   └── task.md
├── lab04/
│   ├── project/
│   ├── report/
│   │   └── report.md
│   └── task.md
├── lab05/
│   ├── project/
│   ├── report/
│   │   └── report.md
│   └── task.md
├── lab06/
│   ├── project/
│   ├── report/
│   │   └── report.md
│   └── task.md
```