Содержание

Оглавление

Требования к выполнению заданий

1. Необходимо выполнить одну курсовую задачу из предложенных, либо предложить свою формулировку, согласовав ее с преподавателем
2. Каждое задание состоит их базовых и дополнительных требований.
3. Требования предложенных задач также можно (и нужно) менять/уточнять. Обязательные требования, как правило, взаимосвязаны и их необходимо реализовать вместе (хотя их изменение также допускается). Из дополнительные требований следует выбирать некоторое подмножество, можно дополнить своими требования.
4. В формулировках большинства задач основным языком реализации предполагается Clojure (если не указано обратного). Это условие (как и все остальные) может быть изменено по согласованию с семинаристом. Следует иметь ввиду, что при этом может возникнуть необходимость в изменении формулировок остальных требований, либо сложность их реализации может существенно измениться (как правило - возрасти).
5. Задания выполняются вдвоем (в редких случаях допускаются исключения)
6. При выполнении задания каждый участник имеет за свою зону ответственности
7. Решение задачи предполагает как минимум три этапа:
    1. Формулировка задачи. Результатом этапа является документ, содержащий уточненную формулировку задачи, а также предполагаемый путь ее решения. Рекомендуемый объем этой части документа 2-3 страницы. Кроме того, документ должен содержать табличное описание (календарный план) оставшихся этапов: сроки, полученная функциональность, разделение ответственности между участниками.
    Срок: первая неделя ноября.

2. Реализация базовая функциональности. На этом этапе должны быть реализованы базовые требования.
   Срок: первая неделя декабря.
3. Расширенная функциональность (таких этапов может быть более одного). На этом этапе должны быть частично или полностью реализованы дополнительные требования.
   Срок: зачетная неделя.

8. Артефакты, получаемые по результатам этапов (кроме 1-го):
   - Программный код со сборочными файлами. Очень желательно, чтобы код собирался без использования IDE. В зависимости от языка реализации рекомендуется использовать Make, Maven, Gradle, Leiningen и т.д.
   - Покрытие компонентными тестами.
   - Автоматически генерируемая документация на программный интерфейс.
   - Примеры, демонстрирующие функциональность разработанной библиотеки. Для большинства задач модельные примеры следует придумать самостоятельно.

Примечание: все артефакты, включая документацию 1-го этапа, должны быть размещены в системе контроля версий (в какой – согласуется с семинаристом

Объектная модель

Требования:

1. Определите для языка Clojure объектную модель. Обеспечите поддержку следующих элементов:
   - класс;
   - атрибут (свойство, слот);
   - множественное наследование (в предположении что все ветки не пересекаются по атрибутам и методам);
   - диспетчеризацию обработки сообщений относительно одного аргумента (динамический полиморфизм).
2. Определите соответствующие функции и/или макросы для работы с перечисленными выше элементами.

Дополнительные требования

Определите поддержку следующих элементов:

- множественное наследование;
- диспетчеризацию обработки сообщений в иерархии с множественным наследованием (классов или примесей); реализуйте аналог call-next-method из CLOS (с фиксированной стратегией диспетчеризации по одному параметру);
- включите вспомогательные методы (**before**, **after**) в механизм диспетчеризации;
- обобщенные функции (диспетчеризация по нескольким параметрам);
- поддержка изменяемого состояния на основе транзакционной памяти (можно использовать встроенную в Clojure поддержку STM); обеспечьте реализацию

принципа Command-Query Separation на уровне декларации классов (и/или других элементов объектной модели).
- статическая типизация.

Рекомендации

Подробно ознакомьтесь с объектными моделями CLOS, Smalltalk и/или Eiffel

Представление данных в S-выражениях

Требования

1. Определите формат представления данных с древовидной структурой в виде S-выражений. В качестве семантической основы рекомендуется рассмотреть форматы XML и/или JSON. Допускается использование как классических S-выражений, так и расширенных, используемых в языке Clojure.
2. Разработайте язык навигации и простых запросов для заданного формата. Должны поддерживаться: относительный или абсолютный путь к узлу дерева, путь с условиями на свойства промежуточных узлов, путь с переменной вложенностью. В качестве семантической основы рекомендуется рассмотреть язык XPath для XML. Реализуйте функции поиска и модификации заданного формата посредством этого языка.
3. Разработайте представление схемы (по аналогии с XML Schema). Реализуйте функции проверки документа по схеме.

Дополнительные требования

- Разработайте язык трансформации в HTML, как упрощенный аналог XSLT.
- Реализуйте модель разбора по аналогии с XML SAX, обеспечьте поддержку трансформации и валидации по схеме в этом режиме.

Рекомендации

Ознакомьтесь XML-технологиями (SAX, DOM, XML Schema, XPath, XSLT)

Конвейерная обработка

Связанные термины: pipeline, workflow, Pipes and Filters pattern.

Требования

Конвейер состоит из связанных между собой узлов. Узел характеризуется набором входных и выходных каналов, по которым могут передаваться объекты. Узел ожидает появления

определенного набора объектов на своих входных каналах, после чего проводит вычисления и порождает объект(-ы) на своих выходных каналах (не обязательно всех). Например, узел "сумматор" имеет два входных канала и один выходной. Получая по одному объекту с каждого входного канала, он их суммирует и результат отправляет в выходной канал. При этом, если входные каналы не будут синхронизованы, то в одном из них могут "скапливаться" данные, ожидающие обработки. Определите проблемно-специфичный язык (DSL) для работы с конвейерными схемами обработки данных, поддерживающий следующие элементы:

- узлы с набором входных и выходных каналов;
- статическую типизацию каналов;
- формирование конвейера в форме связанных узлов в соответствии с типизацией;
- запуск конвейера и обработку данных;
- параллельное исполнение отдельных узлов конвейера;
- использование одинаковых узлов для формирования разных конвейеров.

Вариант модельной задачи

Продемонстрируйте работу разработанной вычислительной модели на примере обработки фотографий. Достаточно симулировать работу фильтров/преобразований без их реального использования. Вместо изображений можно использовать файлы с метаинформацией о том, какие преобразования были выполнены.
Типы фильтров:

- преобразование 1-в-1 (т.е. без изменения типа объекта): устранение шумов, blur и т.д.
- преобразование типа: png->jpg и т.д.
- преобразование n-в-1 (объекты поступают из одного канала): сшивка панорам, HDR-преобразование
- преобразование 1-в-n: нарезка на области
- выбор одного из n (объекты поступают из разных каналов): пытаемся устранить шумы различными методами, дальше сравниваем результаты, выбираем лучший.

Допускается придумать свою модельную задачу, согласовав ее с преподавателем.

Дополнительные требования

Обеспечьте поддержку следующих элементов:

- настраиваемые узлы, параметры которых можно менять в уже созданном конвейере;
- конвейеры с циклами;
- признак окончания работы конвейера (обратите внимание, что конвейер может прийти в состояние, когда его работа никогда не завершится, в этом случае необходимо диагностировать ошибку);
- продемонстрируйте работу дополнительных элементов на модельной задаче;
- многопоточные узлы (для модельной задачи это означает, что можно обрабатывать несколько фотографий одновременно), не нарушающие порядок.

Облачные вычисления

Требования

Реализуйте механизм, позволяющий выполнять ряд операций на удаленных компьютерах, предварительно зарегистрированных в рамках облака (кластера). С точки зрения прикладных программ вызов кода на удаленном компьютере ничем не должен отличаться от локального. Не должно также требоваться предварительного развертывания приложения на удаленном компьютере, т.е. требуется обеспечить маршалинг и передачу не только данных, но и кода. При оптимизации следует исходить из соображений, что удаленный вызов имеет смысл для больших объемов вычислений и существенных объемов данных.
Основные элементы:

1. базовый механизм передачи и запуска (код, данные), а также балансировки загрузки;
2. функция/макрос, обеспечивающая декларацию удаленно исполняемой функции;
3. вариант **map** с параллельным и распределенным исполнением (учитывайте многоядерность современных машин).

Дополнительные требования

Обеспечьте поддержку следующих элементов:

- оптимизация передачи данных по сети: диспетчер заданий будет стараться задачу отдать тому узлу, на котором уже развернут соответствующий код и есть соответствующие данные;
- сделайте механизм асинхронного запуска удаленных вычислений с поддержкой мониторинга прогресса;
- обеспечьте поддержку транспорта Java-объектов;
- обеспечьте поддержку транспорта Clojure- и Java-кода в форме байт-кода.

DI-контейнер

Требования

Требуется реализовать DI-контейнер для Clojure, обеспечивающий поддержку:

1. внедрения зависимости по заданному протоколу (Clojure Protocol)
2. конфигурации (также на Clojure), обеспечивающей:
    - управление жизненным циклом объектов (достаточно реализовать singleton и prototype в терминах Spring);
    - инициализация полей для типов, объявленных через defrecord (в т.ч. другими объектами, находящимися под управлением DI);
3. пользовательской инициализации/деинициализации объектов (по аналогии с аннотациями JavaEE @PostConstruct и @PreDestroy).

Дополнительные требования

Обеспечьте поддержку следующих возможностей:

- расширьте функциональность контейнера так чтобы поддерживалось внедрение экземпляров Java-классов через их интерфейсы;
- распространите действие механизма на Java-код, рекомендуется поддержать обработку стандартной аннотации javax.inject.Inject.

Рекомендации

С понятием DI рекомендуется подробнее ознакомиться здесь.
Конкретная реализация, которую можно рассматривать как образец.

Объектная персистентность

Требования

Реализовать библиотеку поддержки непрерывности существования (persistence) иерархии объектов с поддержкой объектной выборки в предметно-ориентированных терминах (DSL). Рекомендуется использовать языки Clojure/Java. Способ хранения объектов может быть выбран на усмотрение разработчика, однако он не должен влиять на API или объекты пользователя. Требования:

1. поддержка отношений между объектами:
    - One-To-One;
    - One-To-Many (с управлением жизненным циклом, т.е. агрегация или композиция);
    - Поддержка отношения Many-To-One;
2. библиотека должна предоставлять API с базовыми операциями "сохранить", "загрузить", "удалить";
3. библиотека должна предоставлять API для выполнения объектных запросов (query DSL) и (в зависимости от языка) механизмы генерации кода по объектной модели пользователя для построения внутреннего DSL (достаточно будет поиска объектов по совпадению атрибутов с поддержкой выражений NOT, AND, OR).

Дополнительные требования

- Реализовать поддержку наследования
- Реализовать поддержку ленивой инициализации коллекций объектов
- Реализовать в DSL с поддержку проекций, т.е. выборок определенных свойств объектов с формированием нового объекта.

Рекомендации

Ознакомьтесь с аналогичным механизмом в языке C# (язык запросов и т.д.)

Lisp-машина

Требования

Разработайте среду исполнения (runtime) Lisp. Трансляция должна осуществляться одним из следующих способов:

- В объектный код (набор объектов JVM или другой или любой другой платформы). Язык реализации можно выбрать любой. Самый простой вариант.
- В программу на языке C, Java байт-код, или Common Intemediate Language (.NET), которая далее собирается стандартными средствами. Это более сложный вариант.
- Непосредственно в нативный ассемблерный код. Самый сложный вариант, рекомендуется только в случае, если хорошо владеете данным видом языка ассемблера.

Реализация на языке среды исполнения (C, Java и т.д.) должна содержать минимальный набор элементов, которые не могут быть выражены на самом Lisp, все остальные необходимые элементы языка должны быть выражены в этом минимально базисе (определение функции, аппликация, определение списка, и ряд других). Может быть реализовано подмножество любого диалекта Lisp (можете придумать свой), рекомендуется Scheme. Необходимые элементы (не все они обязаны входить в базис):

- Определение функции
- Определение статического лексического контекста (let)
- Рекурсия
- Доступ к вызовам платформы (кроме native assembler)
- Ветвление, логические связки с отложенным порядком исполнения
- Цикл либо оптимизированная хвостовая рекурсия
- Присваивание для символов в определении let (императивный Lisp), либо для отдельного примитива, аналогичного atom в Clojure (чисто функциональный Lisp)
- Макрос
- Ввод-вывод

Дополнительные требования

Определите следующие элементы:

- Генерация и обработка исключений (совместимых с исключениями платформы исполнения)
- Для чисто-функционального Lisp: ленивые вычисления, отложенный порядок вычислений для вызовов функций (в качестве прототипа см. Haskell)
- Динамические лексические контексты (по аналогии с binding в Clojure)
- Механизм, подобный макросу, но генерирующий не Lisp-код, а непосредственно код для платформы исполнения

Рекомендации

Ознакомьтесь с главой 4 книги "Структура и интерпретация компьютерных программ" (желательно и со всеми предыдущими главами)

Неизменяемые структуры данных (Persistent Data Structures)

Требования

Реализуйте библиотеку со следующими структурами данных в persistent-вариантах:

- Массив (константное время доступа, переменная длинна)
- Двусвязный список
- Ассоциативный массив (на основе Hash-таблицы, либо бинарного дерева)

Требуется собственная реализация перечисленных структур. Найти соответствующие алгоритмы также является частью задания. Язык реализации не фиксируется, но рекомендуется Java/C#/C++. Для языков типа Clojure или Haskell задание бессмысленно, т.к. такие структуры и так встроены в язык. В динамических языках (Python, Ruby, JS и т.п.) реализация будет слишком медленная и неэффективная. В базовом варианте решения все структуры данных могут быть сделаны на основе fat-node. Требования:

- Должен быть единый API для всех структур, желательно использовать естественный API для выбранной платформы

Дополнительные требования

- Обеспечить произвольную вложенность данных (по аналогии с динамическими языками), не отказываясь при этом полностью от типизации посредством **generic/template**.
- Реализовать универсальный undo-redo механизм для перечисленных структур с поддержкой каскадности (для вложенных структур)
- Реализовать более эффективное по скорости доступа представление структур данных, чем fat-node
- Расширить экономичное использование памяти на операцию преобразования одной структуры к другой (например, списка в массив)
- Реализовать поддержку транзакционной памяти (STM)

Рекомендации

Для успешного выполнения задания следует самостоятельно подобрать и изучить публикации по теме Persistent Data Structures

Версионная память

Требования

Реализовать структуры данных с поддержкой одновременного многопоточного независимого изменения с возможностью слияния версий.

- Создание версии должно происходить при первом изменении без дополнительных указаний со стороны пользователя библиотеки, как если бы структура данных не была версионной.
- Должны поддерживаться такие структуры данных как множества, очереди и стеки. Способ реализации множества следует выбрать наиболее подходящим для реализации версионности.
- Отличие стека и очереди заключается в специфических операциях доступа, в соответствии с которыми должны применяться различные стратегии слияния версий.
- Демонстрация работоспособности должна быть проведена примере добавления данных конкурентными потоками исполнения, которые со случайными интервалами генерируют данные и помещают их в структуру (можно предложить свой вариант демонстрационной задачи)

Дополнительные требования

- Обеспечить детерминированность алгоритма слияния. Для одной программы вне зависимости от race-conditions должны получаться одни и те же результаты.
- Поддержка сбалансированных деревьев (особенность заключается в том, что после слияния дерево должно оставаться сбалансированным).
- Разработайте механизм для определения пользовательских стратегий объединения версий.
- Реализацию доп. требования продемонстрируйте на примере конкурентного построения дерева частотности.

Рекомендации

Для успешного выполнения задания следует самостоятельно подобрать и изучить публикации по теме. Для начала рекомендуется ознакомиться с этой статьей.