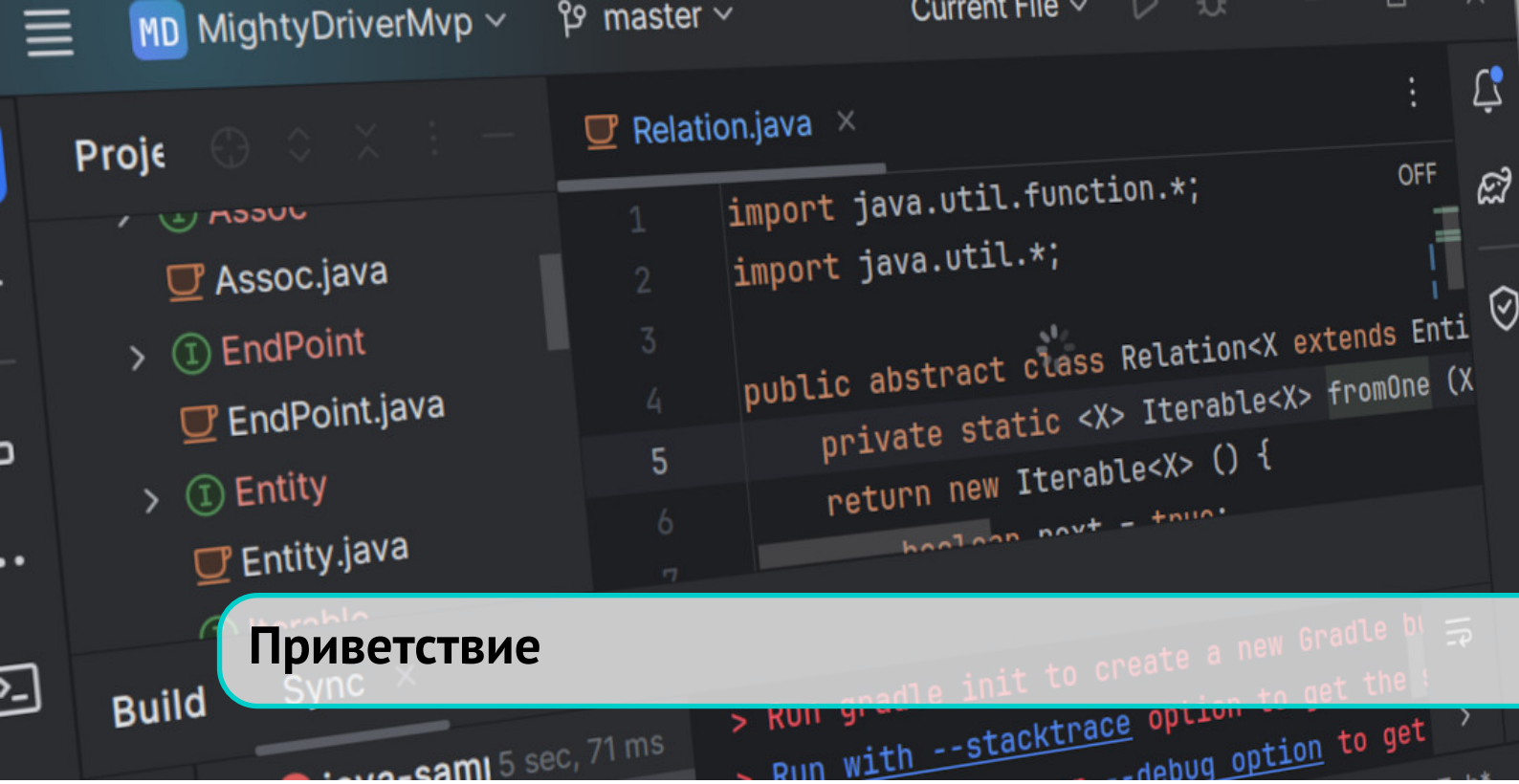
Весенняя проектная школа по системному программированию

Санкт-Петербург, 20–21 апреля 2024 года

МКН СПбГУ

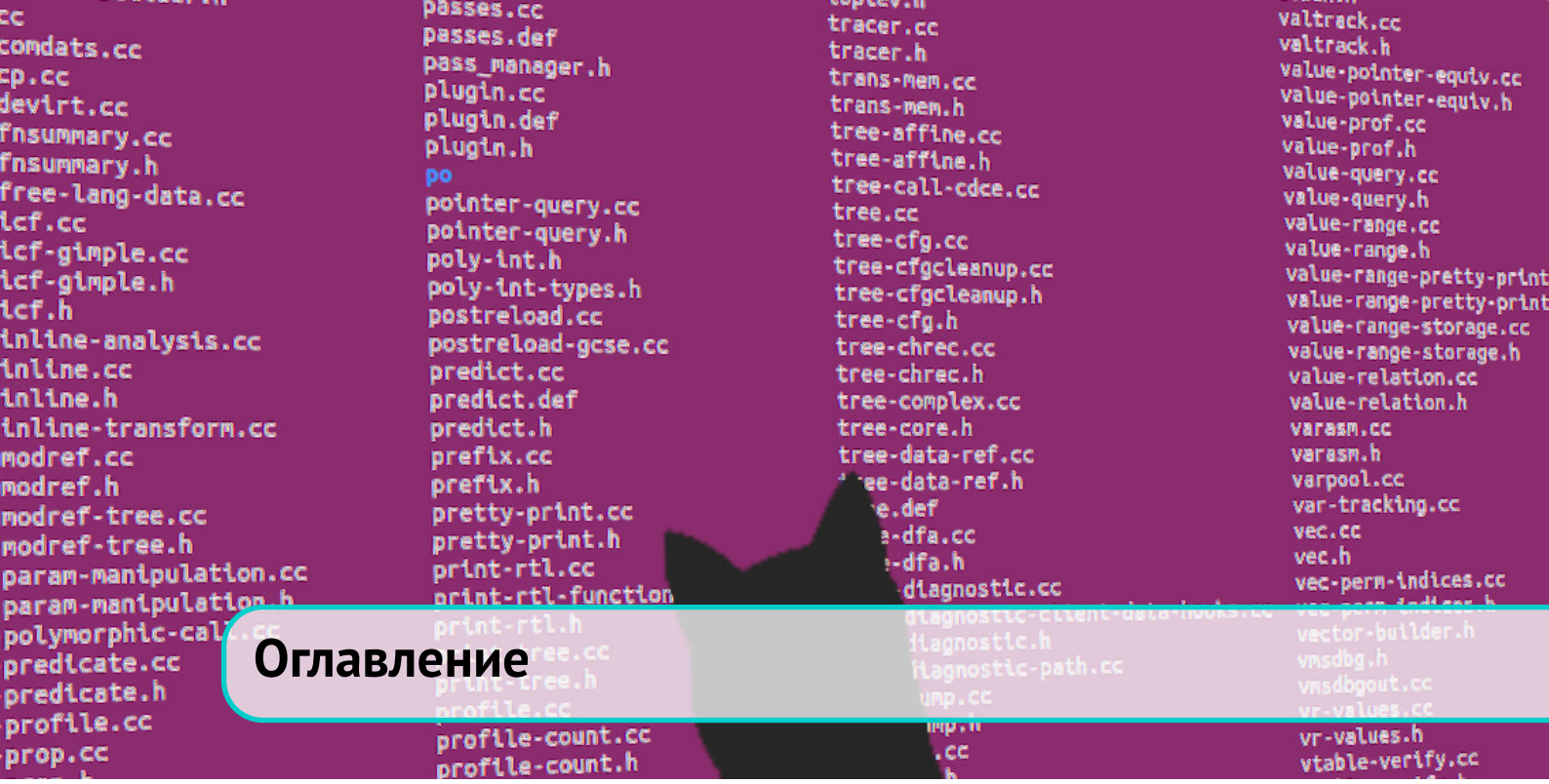


Приветствие

С 2022 года на факультете математики и компьютерных наук (МКН) СПбГУ существует магистерская програма “Разработка ПО и науки о данных”. Обучение в этой магистратуре осуществляется по трём направлениям: разработка больших систем, анализ и верификация программ, науки о данных. Весенняя проектная школа по системному программированию призвана привлечь внимание потенциальных абитуриентов к первым двум направлениям. В рамках школы слушатели познакомятся с интересными проблемами в соответствующих областях науки и инженерии, с общими подходами к их решению, понятийным и технологическим аппаратом. Проекты школы рассчитаны на то, что в течение сжатого промежутка времени их участники смогут не только усвоить необходимый теоретический материал, но и разработать соответствующее программное решение и провести его оценку.

К участию в школе приглашаются студенты бакалавриатов соответствующих специальностей.

Организаторы школы:
Дмитрий Сергеевич Шалымов
Дмитрий Юрьевич Булычев



Оглавление

1. Системное программное обеспечение

1.1 Сетевой обмен файлами по типу BitTorrent

Реализуйте свою систему обмена файлами в P2P сети по примеру BitTorrent. Набор всех пиров (хостов), участвующих в раздаче конкретного файла, называется торрентом. Пиры в торренте обмениваются друг с другом сегментами файла равной длины (например, 256 Кбайт). Когда пир впервые присоединяется к торренту, он не имеет сегментов. С течением времени он собирает их больше и больше. Каждый торрент имеет инфраструктурный узел, называемый трекером. Когда пир присоединяется к торренту, он регистрируется на трекере и периодически информирует его о своем присутствии в раздаче. Трекер отслеживает пиров-участников торрента. Когда новый участник присоединяется к торренту, трекер случайным образом выбирает последовательность пиров (например, 50 штук) из набора участвующих в раздаче и отправляет их IP-адреса новому участнику.

Особенности:

- Присоединяющийся к торренту узел (пир):
  - регистрируется трекером и получает список узлов, соединяется с набором узлов (соседями);
  - не имеет сегментов, но начинает получать их от других узлов.
- Узлы одновременно загружают и отдают сегменты;
- Узел может менять партнеров по обмену сегментами.

Запрос сегментов:

- В любой момент времени различные узлы хранят разный набор сегментов;
- Пусть одного из пользователей зовут Алиса. Алиса периодически запрашивает у каждого узла список сегментов, которые у него есть;
- Алиса запрашивает отсутствующие у нее сегменты по принципу “сначала редкие”. Тогда редкие сегменты начинают раздаваться быстрее, и это приводит к тому, что число копий каждого сегмента в торренте приблизительно уравнивается

Отправка “ты-мне-я-тебе”:

- Алиса отправляет сегменты четырем узлам, которые снабжают ее на максимальной скорости;
  - другие узлы Алиса блокирует и не получает от них сегменты;
  - каждые 10 секунд четверка топ-узлов пересматривается.
- Каждые 30 секунд случайно выбирается новый узел и начинается отправка ему сегментов;
  - это называется “оптимистическая разблокировка” узла;
  - новый узел присоединяется к топ-четверке, остальные “удушаются”.

Дополнительное задание. Реализуйте трекер в виде распределенной хеш-таблицы, как это сделано в BitTorrent (там используется Kademlia DHT).

Дмитрий Сергеевич Шалымов (СПбГУ)
Доцент СПбГУ. Руководитель ОП “Современное программирование” МКН СПбГУ

1.2 Система резервного копирования и хранения (Backup system)

Нужно реализовать резервную копию основного хранилища данных. При этом основное хранилище с течением времени (уже после создания копии) может изменяться. В случае, когда нужно создать резервную копию после изменения уже зареплицированного хранилища, должен создаваться и сохраняться инкремент (только та часть, где были изменены данные). Восстановление хранилища из резервной копии создается так же – сначала поднимается основная резервная копия, дальше к ней по очереди применяются инкременты. Для упрощения задачи можно работать только с текстовыми данными. Размер данных не ограничен.

Дмитрий Сергеевич Шалымов (СПбГУ)
Доцент СПбГУ. Руководитель ОП “Современное программирование” МКН СПбГУ

1.3 Сравнение двух файлов/директорий

Разработайте программу для сравнения двух текстовых файлов и директорий. Должна быть реализована возможность просматривать места с отличиями и переходить от одного к другому. Места, в которых отличаются файлы, нужно подсветить. В качестве примера можно рассмотреть программу WinDiff (https://en.wikipedia.org/wiki/WinDiff).

Дмитрий Сергеевич Шалымов (СПбГУ)
Доцент СПбГУ. Руководитель ОП “Современное программирование” МКН СПбГУ

1.4 Система контроля версий

Реализуйте в упрощенном виде систему хранения версий. Вы можете сами выбрать наиболее интересный функционал для реализации. Однако необходимо сохранять историю изменений (кто, что и когда поменял). В качестве полезного функционала может быть активная история изменений (возможность откатывать/накатывать изменения), функцию blame/annotate, работа с ветками, cherry pick и др. В качестве примера можно взять Git.

Дмитрий Сергеевич Шалымов (СПбГУ)
Доцент СПбГУ. Руководитель ОП “Современное программирование” МКН СПбГУ

1.5 Контейнер внедрения зависимостей (DI-контейнер)

Реализуйте свой DI-контейнер с функционалом, аналогичным существующим широко известным решениям, использующимся в индустрии. DI-контейнер — это библиотека, которая обеспечивает функциональность механизма внедрения зависимостей. В качестве примеров можно рассмотреть Autofac, Simple Injector

Должны быть поддержаны жизненные циклы:

- Singleton;
- Transient;
- Scoped.

Конфигурация

- в виде файла;
- в виде кода.

Контейнер должен уметь работать с циклическими зависимостями.

Дмитрий Сергеевич Шалымов (СПбГУ)
Доцент СПбГУ. Руководитель ОП “Современное программирование” МКН СПбГУ

$$p_{\mathcal{L}} = [\![\mathrm{p}_{\mathcal{L}}]\!]_{\mathcal{L}}$$

$$[\![spec^{\mathcal{L}}_{\mathcal{M}}\,(\mathbf{int}^{\mathcal{N}}_{\mathcal{L}} \times \mathrm{p}_{\mathcal{N}})]\!]_{\mathcal{L}}\,(x) = int^{\mathcal{N}}_{\mathcal{L}}\,(\mathrm{p}_{\mathcal{N}} \times x) = [\![\mathrm{p}_{\mathcal{N}}]\!]_{\mathcal{N}}\,(x) \qquad \text{(I)}$$

$$[\![[\![spec^{\mathcal{M}}_{\mathcal{K}}\,(\mathbf{spec}^{\mathcal{L}}_{\mathcal{M}} \times \mathbf{int}^{\mathcal{N}}_{\mathcal{L}})]\!]_{\mathcal{K}}\,(\mathrm{p}_{\mathcal{N}})]\!]_{\mathcal{L}}\,(x) =$$
$$[\![spec^{\mathcal{L}}_{\mathcal{M}}\,(\mathbf{int}^{\mathcal{N}}_{\mathcal{L}} \times \mathrm{p}_{\mathcal{N}})]\!]_{\mathcal{L}}\,(x) = [\![\mathrm{p}_{\mathcal{N}}]\!]_{\mathcal{N}}\,(x) \qquad \text{(II)}$$

$$[\![spec^{\mathcal{K}}_{\mathcal{L}}\,(\mathbf{spec}^{\mathcal{M}}_{\mathcal{K}} \times \mathbf{spec}^{\mathcal{L}}_{\mathcal{M}})]\!]_{\mathcal{T}}\,(\mathbf{int}^{\mathcal{N}}_{\mathcal{L}}) \cdot spec^{\mathcal{K}}_{\mathcal{L}}\,(\mathrm{p}_{\mathcal{M}} \times \mathbf{int}^{\mathcal{N}}_{\mathcal{L}}) \qquad \text{(III)}$$

2. Языки и инструменты программирования

2.1 Генерация кода на основе первой проекции Футамуры

Компиляция в машинный код представляет собой достаточно сложную задачу; общепринятая точка зрения заключается в том, что для реализации эффективного компилятора необходимы инфраструктуры–“бегемоты” вроде LLVM или GCC. Тем не менее иногда полезно отказаться от общепринятой точки зрения и подойти к вопросу с противоположной стороны.

В рамках данного проекта мы попробуем применить *первую проекцию Футамуры* для получения машинного кода программы путем специализации интерпретатора исходного языка.

Вкраце, проекции Футамуры представляют собой соотношения, связывающие различные *языковые проессоры*, такие как *интерпретаторы* и *компиляторы*. Ключевым понятием в проекциях Футамуры является *специализатор*, или *частичный вычислитель* — программа **spec**, которая для некоторой другой программы p и части её входных данных x строит *остаточную программу* **spec** $p\,x$, в которой (потенциально) произведена оптимизация вычислений, связанных с известными данными x. Определяющее соотношение для специализатора есть

$$(\mathbf{spec}\,p\,x)\,y = p\,x\,y$$

где y — “остаток” входа для p. Разработка нетривиальных специализаторов и исследование их свойств представляет собой отдельную область науки.

В свою очередь, первая проекция Футамуры есть специализация интерпретатора на интерпретируемую программу. При этом можно показать, что результатом является *код этой программы в языке реализации интерпретатора*. Несмотря на то, что данное соотношение может показаться совершенно абстрактным, оно используется на практике — именно такова технология раскрутки компилятора для нового языка в инфраструктуре Graal VM. Однако целевыми платформами для Graal VM являются виртуальные среды управляемого исполнения, мы же будем нацелены на получение непосредственно машинного кода.

Для того, чтобы получить машинный код некоторой программы, нам нужны
- некоторое внутреннее представление этой программы;
- интерпретатор этого внутреннего представления, написанный на машинном языке;
- специализатор для машинного языка.

Важно, что интерпретатор в этой схеме может быть (и, как правило, есть) фиксированный, написанный раз и навсегда. Следовательно, специализатор тоже может быть специализированный, приспособленный для эффективной специализации только одной определенной программы (*генерирующее расширение*), что существенно упрощает дело.

В рамках данного проекта мы реализуем машинный интерпретатор кода абстрактной машины для языка $\lambda^a\mathscr{M}^a$ и генерирующее расширение этого интерпретатора. В настоящий момент уже существует готовый компилятор из $\lambda^a\mathscr{M}^a$ в код для абстрактной машины, что сразу позволит проверить реализованный машинный компилятор на содержательных программах.

Петр Алексеевич Лозов (СПбГУ)
В 2018 году с отличием окончил магистратуру математико-механического факультета СПбГУ. В аспирантуре исследовал проблему эффективного исполнения реляционных программ и разработал метод динамического управления порядком исполнения конъюнктов в реляционной программе. В 2022 году защитил кандидатскую диссертацию на тему “Автоматизированное построение и эффективное исполнение реляционных программ”. Сейчас продолжает исследовать вопросы эффективного исполнения реляционных программ в прикладных задачах.

Дмитрий Юрьевич Булычев (СПбГУ)
Кандидат физ-мат. наук, доцент кафедры системного программирования математико-механического факультета СПбГУ, главный академический консультант «Техкомпании Хуавей», руководитель ОП “Разработка ПО и науки о данных” МКН СПбГУ. Область научных интересов — языки и инструменты программирования, компиляторы, функциональное, логическое и реляционное программирование, анализ и синтез программ.

2.2 Проверка типов с модальностью значений

Предлагается разработать проверку типов для языка, который позволяет указывать модальность значений. Если кратко, то существуют три оси модальностей.

Первая регламентирует локальность использования значений:
- *локальные значения* не покидают область видимости функций, где они объявлены (если покидают, то с помощью явного указания);
- *глобальные* не имеют такого ограничения.

Ось уникальности:
- уникальные аргументы функций:
  - нельзя использовать дважды;

  - можно изменять in-place без риска ошибок;
  - чтобы их передавать куда-то, нужен специальный синтаксис (“borrowing“).
- *эксклюзивные значения*, которые запрещено использовать одновременно в двух местах (например, `File.write` захватывает дескриптор файла эксклюзивно);
- “*shared*” — без ограничений.

“*Affinity*”:

- значения, которые можно использовать *единожды*;
- *разделенные* замыкания: запрещено создавать новые ссылки на замыкание во время его исполнения;
- *many* — без ограничений.

Идея проекта родилась из наблюдения тенденций развития функционального языка программирования OCaml. В компании Jane Street локальные переменные используются около года, а также они занимаются дальней разработкой видов модальности.

В рамках проекта предлагается реализовать проверку типов и их модальностей, а также всё для этого необходимое.

Дмитрий Сергеевич Косарев (СПбГУ)
Энтузиаст функционального программирования. Преподаватель на кафедре системного программирования мат.-мех. факультета СПбГУ, читает курсы "Функциональное программирование"и "Трансляция языков программирования".