Практика программироавния. Заметки.

Семён Григорьев

9 марта 2021 г.

Оглавление

Глава 1

рактика программирования, семестр 1

1.1 Лекция 1: Введение

Программирование — не только написание кода. Документация, сборка, тестирование, версионирование, обработка отзывов пользователей.

Инфраструктура проекта, рабочее окружение, система контроля версий, непрерывная сборка.

Соответствующие решения на примере инфраструктуры вокруг GitHub. GithubActions, внешние сервисы для CI (`https://travis-ci.org/`, `https://www.appveyor.com/`, `https://circleci.com/`).

Практика развёртывания соответствующей инфраструктуры.

1. Для начала, завести аккаунт на GitHub (`https://github.com/`). Важно, чтобы имя аккаунта (логин) было `NameSurname` или `Name_Surname`.

2. Создаём репозиторий для проекта (для всех домашних работ). Название должно отражать сожержимое репозитория. Не забываем добавить описание. Лицензию, readme и gitignore лучше не добавлять.

3. Устанавливаем git (`https://git-scm.com/`) и графическую оболочку для работы с ним (если кому нужно).

4. Теперь пора приступать к созданию проекта. Так как дальше мы будем пользоваться F#, то в качестве шаблона предлагается использовать `https://github.com/TheAngryByrd/MiniScaffold`.

   (a) Установить .NET Core: `https://dotnet.microsoft.com/download`

   (b) Прочитать инструкции (https://github.com/TheAngryByrd/MiniScaffold#install-the-dotnet-template-from-nuget) и выполнить соответсвующие шаги. Нам нужно создать консольное приложение. Это может занять некоторое время.

5. Устанавливаем связь только что созданного локального репозитория с удалённым репозиторием: https://docs.github.com/en/github/importing-your-projects-to-github/adding-an-existing-project-to-github-using-the-command-line

С этого момента домашние работы только через GitHub с налаженной сборкой.

1.2 Домашняя работа 1

Задачи:

1. **(1 балл)** Инициализировать рабочее окружение: репозиторий на GitHub, CI, readme, лицензия. Добавить преподавателя в совладельцы. Оформить тестовый pull request: например, оформленное readme (поправить описание проекта, удалить лишнее, что досталось от шаблона, поправить ссылку на статус сборки). Запросить ревью у преподавателя.

1.3 Лекция 2

1.3.1 Первое знакомство с F#

Общие сведения о платформе .NET, общие представления об архитектуре платформы.

F# — это один из языков платформы. Некоторые ресурсы дл того, чтобы начать изучать F#.

- Главный сайт сообщества: `https://fsharp.org/`. Там много чего интересного, от библиотек и инструментов до ссылок на другие ресурсы.
- Неплохая книга по основам языка: `https://en.wikibooks.org/wiki/F_Sharp_Programming`
- Подборка обучающих материалов от сообщества: `https://fsharp.org/learn/`
- Официальная документация от Microsoft: `https://docs.microsoft.com/ru-ru/dotnet/fsharp/`

Структура проекта вообще и на языке F# в частности. Разбиение кода на модули, файлы, библиотеки. Переиспользование кода. Зависимости между модулями, библиотеками. Пространства имён.

Основные особенности F#, примеры кода, базовые языковые конструкции и типы.

Примитивные типы: логисекие, числовые, символы, строки. Массивы, основные функции работы с ними: инициализация, взятие элемента, запись элемента.

Базовые констукции управления: ветвления (if), циклы (различные варианты for, while). Двумерный синтаксис.

Структура программы. Точка входа, модули, функции. Модуль и пространство имён.

Основы работы с консолью, библиотека Argu.

1.3.2 Тестирование программ

Тестирование программ: ручное, автоматизированное, автоматическое.

Доказательство корректности vs тестирование. Тестирование не есть доказательство корректности. А можно ли всё таки доказать корректность? Да (иногда).

- Формальная верификация используя внешние инструменты.

- Использование специальных языков программирования, таких как Coq (на самом деле это целая система, так называемый proof assistant), Agda, Idris, F*,

Типы тестов и особенности их применения: модульные, интеграционные, unit и т.д. Автоматизация создания тестов. Intellitest — пример инстумента для автоматической генерации тестов. Примеры инструментов для тестирования: Expecto, FsUnit, NUnit, FsCheck, Canopy.

С этого момента все домашние работы должны быть снабжены автоматически запускаемыми при сборке тестами.

1.3.3 Отладка кода

Отладка кода. Некоторые методы отладки: отладочная печать, логгирование, использование пошаговых отладчиков. Некоторые шаги отладки: формулировка гипотезы и её проверка, локализация ошибки, работа с тестами. Практика по использованию отладчика.

1.4 Домашняя работа 2

В задачах, связанных с обработкой массивов на вход необходимо принимать длину массива и затем создавать случайный массив соответствующей длины. Для всех задач необходимо реализовать чтение входных данных из консоли и вывод результата в консоль.

Задачи:

1. **(1 балл)** Реализовать функцию, вычисляющую значение выражения $x^4+x^3+x^2+x+1$ “наивным” способом.
2. **(1 балл)** Реализовать функцию, вычисляющую значение выражения $x^4+x^3+x^2+x+1$, применив минимальное число умножений и сложений.
3. **(1 балл)** Вычислить индексы элементов массива, не больших, чем заданное число.
4. **(1 балл)** Вычислить индексы элементов массива, лежащих вне диапазона, заданного двумя числами.
5. **(1 балл)** Дан массив длины 2. Поменять местами нулевой и первый элементы, не используя дополнительной памяти/переменных.
6. **(1 балл)** Поменять местами i-й и j-й элементы массива не используя дополнительной памяти/переменных.

1.5 Лекция 3

Ещё раз произменения: в реквесте должно быть только то, что непосредственно относится к сдаваемой домашке.

Ещё раз про функции, про то, как выделять и разделять функциональность, не надо запихивать всё в одну функцию. Про то, где должны быть проверки.

Про консоль.

Про обработку крайних случаев. Про исключения.

Про тесты и ошибки: нашёл ошибку — создал тест.

Про стиль кодирования: про пробелы вокруг скобок и операций, про отступы и переводы строк. Про соглашения о наименовании. camlCase CamlCase

Про единицы измерения.

Базовые структуры данных, алгоритмы и их выражение в F#. Функция. Рекурсия и итерация. Базовые типы и основы работы с ними: матрицы, массивы, списки, структуры.

Числа Фибоначчи.

1.6 Домашняя работа 3

Для всех задач обеспечить чтение n из консоли и печать результата в консоль.

1. **(1 балл)** Реализовать вычисление n-ого числа Фибоначчи рекурсивным методом.

2. **(1 балл)** Реализовать вычисление n-ого числа Фибоначчи итеративным методом.

3. **(1 балл)** Реализовать вычисление n-ого числа Фибоначчи используя хвостовую рекурсию (не используя `mutable` и других изменяемых структур). Подсказка: нужно использовать рекурсию с аккумулятором.

4. **(2 балла)** Реализовать вычисление n-ого числа Фибоначчи через перемножение матриц "наивным" методом. Функции построения единичной матрицы, умножения и возведения в степень должны быть реализованы в общем виде.

5. **(2 балла)** Реализовать вычисление n-ого числа Фибоначчи через перемножение матриц за логарифм.

6. **(1 балл)** Реализовать вычисление всех чисел Фибоначчи до n-ого включительно.

1.7 Лекция 4

Рассказать про то, что выделенные значения — это плохо. Немного про исключительные ситуации. Про создание проектов. Про версии пакетов и вообще версии артефактов. Про то, что заимствование кода не поощрается. Тем более неправомерное заимствование. Про классный пример форматирования `x*x*x + x*x + x + 1`.

Работа с файлами.

Сортировки: пузырьком, вставкой, Хоара. Различные сценарии использования: поддержание отсортированного набора, сортировка всего набора целиком. Некоторые особенности реализации: наивная функциональная реализация Хоара, реализация на массиве.

Основы машинного представления данных. Представления чисел. Представление чисел с плавающей точкой. Проблемы переполнения. Битовые операции. Строки, кодировки.

1.8 Домашняя работа 4

Во всех задачах на сортировку необходимо реализовать чтение массива из файла и печать результата в файл. Функции чтения и записи необходимо переиспользовать.

В задачах на битовые операции продолжаем рабоать с консолью: чтение данных с консоли и печать результата туда же.

Для данной домашней работы необходимо создать отдельный проект.

При создании тестов необходимо, в задачах на сортировку, убедиться, что, во-первых, сортировки ведут себя одинаоково на одинаковых данных, во-вторых, что они ведут себя так же, как системные сортировки для соответствующих коллекций. Для задачи о запаковке и распаковке надо проверить, что реализованные функции являются взаимно обратными. FsCheck (testProperty) в помощь.

1. **(1 балл)** Реализовать сортировку пузырьком массива.
2. **(1 балл)** Реализовать сортировку пузырьком списка.
3. **(1 балл)** Реализовать быструю сортировку для списка.
4. **(1 балл)** Реализовать быструю сортировку для массива.
5. **(1 балл)** Реализовать запаковку двух 32-битных чисел в одно 64-битное и распаковку обратно.
6. **(1 балл)** Реализовать запаковку четырёх 16-битных чисел в одно 64-битное и распаковку обратно.

1.9 Лекция 5

Основы анализа алгоритмов. Модель вычислителя. Понятие элементарной операции. Асимптотика, “O”-символика.

Постановка эксперимента и оформление результатов. Эксперимент по сравнению и анализу производительности. Точность проведения замеров. “Масштабы времени”, цель эксперимента и точность измерений, инструменты измерений. Базовая статистическая обработка данных: медиана и среднее, выбросы, распределение. Проверка гипотез.

Технические средства. О системе вёрски LaTeX. TexLive, Overleaf. Способы визуализации результатов. Python + Matplotlib, другие библиотеки. GnuPlot.

Отдельно про скрипты сборки. Теперь мы руками попробуем сделать свой маленький скрипт. Shell, cmd (PowerShell).

1.10 Домашняя работа 5

В данной задаче предолагается, что для обработки сырых данных и отрисовки графиков используется Python и Matplotlib, однако другие сравнимые по выразительности и качеству результата пакеты разрешены. Исользование офисных пакетов (типа LibreOffice, MS Office и т.д.) запрещено. Шрафики должны быть векторными. Внимательно ледите за тем, что в репозиторий должны быть только исходники, сгенерированные файлы в репозиторий попасть не должны.

1. **(5 баллов)** Провести сравнительное исследование реализованных в предыдущей домашней работе сортировок и стандартных реализаций сортировок соответствующих коллекций. Оформить отчёт: введение, детали реализации, постановка эксперимента, результаты экспериментов, анализ результатов. Отчёт оформляется в LaTeX, исходники выкладываются так же как и обычный код, снабжаются скриптом сборки (Shell), который генерирует графики и собирает pdf-документ.

1.11 Лекция 6

11. Контрольная работа.

1. **[3 балла]** Что такое хвостовая рекурсия? Чем отна отличается от "обычной" рекурсии? Привелите пример кода на F# для следующих функций, реализованных с использованием хвостовой рекурсии
   (a) сумма элементов списка,
   (b) минимальный элемент списка,
   (c) максимальный элемент списка,
   (d) факториал,
   (e) длина списка
2. **[3 балла]**
   (a) Расскажите про этапы жизни программнго продукта (проекта). Опишите каждый этап.
   (b) Для каких целей может проводиться экспериментальное исследование реализации алгоритма. В чём отличие между теоретическим и экспериментальным исследованием?
   (c) Что такое отсортированная коллекция? Каким условиям должны удовлетворять элементы коллекции, чтобы коллекцию из этих элементов можно было отсортировать? Какие алгоритмы сортировки вы знаете? Чем они отличаются?
   (d) Перечислите компоненты инфраструктуры проекта. Какую роль они выполняют? Приведите примеры конкретных реализаций компонент.
   (e) Что такое отладка? Что такое тестирование? Чем отладка отличается от тестирования? Какие средства отладки вы знаете? Какие средства тестирования вы знаете?
3. **[4 балла]**
   (a) Сколько операций сложения и умножения чисел требуется для перемножения двух матриц? Почему?
   (b) Сколько операций сравнения потребуется для сотрировки списка длины n сортировкой Хоара? Почему?
   (c) Сколько операций сравнения потребуется для сотрировки списка длины n сортировкой пузырьком? Почему?
   (d) Сколько операций сложения потребуется для вычисления n-ого числа Фибоначчи наивным рекурсивным методом? Почему?

(e) Сколько операций сложения и умножения чисел потребуется для вычисления n-ого числа Фибоначчи через умнжение матриц “умным” (не наивным) способом? Почему?

12. Про анализ результатов экспериментов. Бокс-плот. Придумать что-то про JIT?

1.12 Лекция 7

Пример рефакторинга разбора аргументов командной строки. И то же самое с тестами.

Понятие типа данных. Системы типов: статические, динамические, строгие, нестрогие. Примеры языков с разными системами типов. Понятие о разной выразительности (“мощности”) систем. Пример кода на $F^\star$.

Приведение типов: автоматическое, ручное.

Обобщённые типы данных. Понятие о полиморфизме.

Алгебраические типы данных: кортежи, Discriminated Unions (рамеченные объединения). Примеры на F#. Единицы измерения.

1.13 Домашняя работа 6

1. **(1 балл)** Предположим, что мы храним булевы матрицы в виде списка координат ячеек, значение которых `true`. Необходимо реализовать соответствующие типы: единицы измерения для строк и столбцов, пара “строка-столбец”, список пар “строка-столбец”.

2. **(2 балла)** Реализовать функцию, перемножающую две матрицы, заданных в формате, описанном в предыдущей задаче. Не забыть проверку корректности входных данных. Реализовать подгрузку матриц из файла и запись результата в файл. Файлы с данными и результатом указываются через консоль. Формат хранения матрицы из m строк и n столбцов: в файле m строк, каждая строка состоит из n символов 0 или 1.

1.14 Лекция 8

Про то, что теперь не указываем типы там, где это не нужно. А точнее, указываем только там, где это необходимо. Про думать головой над постановками задачи, а не просто кодить.

Ещё раз про полиморфизм, Ad-hoc полиморфизм и бинарные операции. Типовые параметры и ограничения на них в F# (`https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/generics/constraints`).

Списки, деревья: как формальные объекты, структуры данных и как примеры алгебраических обобщённых типов. Реализация списка и дерева. Обходы списков и деревьев.

Про fold, map, iter.

Длинная арифметика. Практика работы со списками. Ещё раз о проблеме переполнения. Целочисленная арифметика на списках.

1.15 Домашняя работа 7

1. **(1 балл)** Реализовать самостоятельно полиморфный непустой список (далее будем называть этот тип `MyList`). Реализовать для него функции сортировки, вычисления длины, конкатенации, map, iter. Реализовать преобразование из стандартного списка в MyList.

2. **(1 балл)** На основе `MyList` реализовать тип `MyString`, представляющий строку как список символов. Реализовать преобразование стандартной строки в `MyString` и конкатенацию строк для `MyString`.

3. **(1 балл)** Реализовать тип дерева с произвольным количеством потомков в каждом узле (использовать `MyList`) `MyTree`. Каждый узел должен хранить данные произвольного типа.

4. **(1 балл)** Пусть есть `MyTree`, хранящий в узлах целые числа. Реализовать функции, которые находят максимальный хранимый элемент, среднее значение всех хранимых элементов.

1.16 Лекция 9

Ещё раз про то, как условный оператор форматировать.

Граф как формальный объект и как структура данных. Понятие о бинарном отношении и его свойствах: транзитивность, рефлексивность, симметричность. (Не)Ориентированные, (не)помеченные графы. Способы представления графов: список рёбер, список смежности, матрица смежности.

GraphViz

Базовые алгоритмы на графах. Обходы в глубину и ширину, построение транзитивного замыкания, поиск кратчайшего пути.

Линейная алгебра. Основы: матрица, вектор, полукольцо, кольцо, монид, полугруппа. Сведение некоторых задач к операциям линейной алгебры (транзитивное замыкание, кратчайшие пути, пересечение автоматов). Особенности практического использования такого подхода: разреженные структуры данных, абстрактность, композициональность.

Множество S, с заданными на нем бинарными операциями $+$ и $\cdot$, называется полукольцом, если для любых элементов a, b, c верно следующее:

1. $\langle S, + \rangle$ — коммутативный моноид. То есть имеют место свойства:
   (a) Коммутативности: $a + b = b + a$
   (b) Ассоциативности: $(a + b) + c = a + (b + c)$
   (c) Существования нейтрального элемента (нуля): $a + 0 = 0 + a = a$
2. $\langle S, \cdot \rangle$ — полугруппа. Необходимо свойство ассоциативности: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
3. Умножение дистрибутивно относительно сложения:
   (a) Левая дистрибутивность: $a \cdot (b + c) = a \cdot b + a \cdot c$
   (b) Правая дистрибутивность: $(a + b) \cdot c = a \cdot c + b \cdot c$

4. Мультипликативное свойство нуля: $a \cdot 0 = 0 \cdot a = 0$

Моноид — это полугруппа с нейтральным элементом. Кольцо, в отличие от полукольца, по сложению образуеи комутативную группу (содержит обратные по сложению).

В отечественной культуре в полукольце нет нейтрального по умножению, зато есть полукольцо с единицей — полукольцо с нейтральным по умножению. Фактически, умножение начинает задавать моноид. Однако часто определение полукольца включает требование наличие нейтрального по умножению.

Полукольцо называют коммутативным, если операция умножения в нём коммутативна.

Полукольцо называют идемпотентным, если для любого $s \in S, s + s = s$

Дерево квадрантов.

Примеры матриц смежности и тензорное произведение можно посмотреть в этом документе. Соответственно, разделы 2.1 “Основные определения” и 7.2 “Тензорное произведение”.

1.17 Домашняя работа 8

При создании новых структур данных необходимо расширять библиотеку, созданную в предыдущей домашней работе.

1. **(5 баллов)** Используя тип MyList из предыдущей домашней работы, реализовать целочисленную длинную арифметику: операции сложения, умножения, вычитания, целочисленного деления.

2. **(2 балла)** Реализовать представление разреженных матриц в виде дерева квадрантов. Реализовать функцию поэлементного сложения двух матриц в таком формате.

3. **(3 балла)** Реализовать функцию умножения двух матриц в формате дерева квадрантов.

4. **(3 балла)** Реализовать функцию тензорного умножения двух матриц в формате дерева квадрантов.

5. **(5 баллов)** Реализовать построение транзитивного замыкания ориентированного графа через произведение матриц. Использовать представление матриц из второй задачи. Визуализировать результат с помощью GraphViz: исходный граф и выделенные рёбра, появившиеся в результате транзитивного замыкания. Для задания графа использовать формат из задачи 2 6-й домашней работы.

6. **(5 баллов)** Реализовать вычисление кратчайших путей между всеми парами вершин в ориентированном графе. Использовать представление матриц из второй задачи. Визуализировать результат с помощью GraphViz: исходный граф и выделенные рёбра со значением кратчайшего пути между соответствующей парой вершин. Для задания графа использовать формат, аналогичный формату из задачи 2 6-й домашней работы.

1.18 Лекция 10

Рассказать про иерархию типов и про коллекции (Seq)

Дальше будет много про формальные языки. Неплохой конспект на русском: `https://neerc.ifmo.ru/wiki/index.php?title=%D0%9A%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D1%8F:%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D1%85_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2`. Там, правда, без линейной алгебры.

Регулярные выражения и конечные автоматы. Определения. Построение автомата по регулярному выражению. Применения регулярных выражений (поиск, анализ текста, моделирование систем, анализ программного кода).

Устройство языков программирования: лексика, синтаксис, семантика. Определения, примеры. Шаги обработки кода: лексический и синтаксический анализы, “семантический” анализ.

Лексический и синтаксический анализ. Введение в формальные языки как способ описания синтаксиса. Контекстно-свободные граммтики как система переписываний. Способы реализации лексических и синтаксических анализаторов. ANTLR, fslex+fsyacc, FParsec.

1.19 Лекция 11

Проделжение про FsLex+FsYacc, полноценный пример интерпретатора.

Интерпретация и компиляция. Особенности, разновидности интерпретаторов, основные шаги. Особенности, разновидности компиляторов, основные шаги. Примеры, пример реализации простого интерпретатора.

Устройство сред разработки и компиляторов, интерпретаторов: общие шаги, классические возможности, JIT/AOT. Примеры из .NET, F#, JVM, LLVM.

F# $\rightarrow$ IL $\rightarrow$ C# $\rightarrow$ ASM

```
module M

type Lst<'a> =
    | Nill
    | Cons of 'a * Lst<'a>

let rec map f l =
    match l with
    | Nill -> Nill
    | Cons (x,tl) -> Cons(f x, map f tl)

Cons(1,Nill)
|> map ((+)1)
|> printfn "%A"
```

Этот код на sharplab.io

```
module M

let f x =
    let h y = if y / 2 = 0 then y * x else y + x
    if x > 5 then x + 2 else h (x * 6)

let rec g y = f (y + g 2)
```

Этот код на sharplab.io

1.20 Домашняя работа 9

В задачах ниже необходимо максмально переиспольховать результаты предыдущих домашних работ. Для реализации синтаксического нализатора можно использовать не только FsYacc, рассматриваемый на паре, но и другие инструменты (например FParsec). Нельзя писать анализатор руками.

1. **(4 балла)** Разработать библиотеку конечных автоматов, использующую разреженные матрицы из пердыдущей работы для пердставления переходов автомата, и предоставляющую следующие возможности.
   - Построение автомата по регулярному выражению. Можно ограничится заданием регулярного выражения через конструкторы типа.
   - Построение пересечения двух автоматов.
   - Возможность проверять, принимается ли строка автоматом.
2. **(7 баллов)** Релизовать синтаксический анализатор регулярных выражений, позволяющий в предыдущей задаче задавать регулярное выражение как строку. поддерживаемые операции регулярных выражений: конкатенация, альтернатива, звезда Клини. Алфавит регулярных выражений — строчные и прописные латинские символы, цифры, арифметические знаки, знаки припенания. Обеспечить построение автомата по регулярному выражению. Предусмотреть возможность задавать регулярное выражение с консоли, а на выход получать представление автомата в DOT.
3. **(7 баллов)** Реализовать синтаксический анализатор для арифметических выражений над целыми числами. Поддерживаемые операции: сложение, умножение, вычитание, деление. Также бывают группирующие скобки. Числа могут быть очень большими (но целыми). Реализовать вычисление значения выражения на основе операций длинной арифметики. Предусмотреть возможность задавать выражение с консоли, а на выход получать результат его вычисления (в консоль) и дерево разбора в формате DOT.
4. **(8 баллов)** Расширить язык регулярных выражения следующими конструкциями.
   - Операцией пересечения, повторения один или более раз, повторения 0 или 1 раз. Все эти операции могут встречаться в произвольном месте выражения.
   - Функцией проверки, что строка принадлежит языку, задаваемому выражением.
   - Функцией поиска всех подстрок, удовлетворяющих заданному регулярному выражению.

- Функцией печати атомата, задаваемого выражением, в файл в формате DOT.
- Функцией печати результата в консоль.
- Переменными. Переменные могут использоваться в правых частях всех выражений.

Реализовать интерпретатор получившегося языка. Предусмотреть возможность его консольного запуска: на входе файл с кодом на нашем языке, на выходе — результат интерпретации.

5. **(8 баллов)** Расширить язык арифметики следующими конструкциями.

   - Операцией возведения в степень, взятия остатка (от целочисленного деления), модуля, добавить унарный минус.
   - Функцией перевода числа в двоичную систему исчисления.
   - Функцией печати результата в консоль.
   - Переменными. Переменные могут использоваться в правых частях всех выражений.

   Реализовать интерпретатор получившегося языка. Предусмотреть возможность его консольного запуска: на входе файл с кодом на нашем языке, на выходе — результат интерпретации.

1.21 Лекция 12

Парадигмы программирования. Структурное программирование: машины Тьюринга, архитектура фон Неймана, языки-представители.

Начать с автомата (трансдьюсера). Сложение в унарной систее исчисления.

Машина Тьюринга. Машина Поста, нормальные алгорифмы Маркова.

Машина Тьюринга $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$:

- Q — конечное не пустое множество состояний;
- Γ — конечное не пустое множество символов (алфавит);
- $b \in \Gamma$ специальный пробельный символ;
- $\Sigma \subseteq \Gamma \setminus \{b\}$ входной алфавит (то, что можно написать на ленту “снаружи” перед стартом);
- $q_0 \in Q$ — стартовое состояние;
- $F \subseteq Q$ Множество финальных состяний. Говорят, что вход принимается машиной M ечли она оказывается в одном из финальных состояний F.
- $\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ — частично определённая функция переходоа. L и R — команды перемещения головки (влево и вправо соответственно). Если δ не определена, то машина останавливается.

Эмулятор машин Тьюринга: `https://turingmachinesimulator.com/`

Архитектура фон Неймана (Принстон).

- Однородность памяти: данные и команды — одно и то же, хранятся в одной памяти, обрабатываются по общим принципам.
- Адресность: память — это набор занумерованных ячеек.
- Программное управление: все действия описываются программой.

Узкое место — канал передачи данных.

Гарвард.

- данные и команды разделены.

Проблема: сложнее и дороже.

Классика: TTA (`http://openasip.org/`).

1.21.1 Busy beaver

- У машины n состояний + 1 финальное
- У машины одна бесконечная в обе стороны лента
- Алфавит ленты: $\{0, 1\}$, 0 — пробельный символ.
- Функция перехода получает состояние и текущий символ на ленте, переходит в новое состояние, пишет что-то на ленту, сдвигает головку (налево или направо).

Задача: найти терминирующуюся машинутьюринга, записывающую максимальное число 1 на ленту.

Невычислимая функция.

1.21.2 Лямбда-исчисление

Применение и абстракция. Констант не предполагается.

Сложене в арифметике Пеано.

α-эквивалентность — переименование переменных.

β-редукция — вычисление. $(\alpha x.t)a = t[x := a]$

η-преобразование — заворачивание-разворачивание функций. $\lambda x.fx = f$. Если нет свободных вхождений x в f

Лямбда-процессоры.

Reduceron: `https://www.cs.york.ac.uk/fp/reduceron/` A Platform for Full-Stack Functional Programming: `https://ieeexplore.ieee.org/abstract/document/9180772` ACQuA: A Parallel Accelerator Architecture for Pure Functional Programs: `https://ieeexplore.ieee.org/abstract/document/9155051`

Рекурсивные функции (Гёдель, теория вычислимости).

Кодирование по Чёрчу (Church encoding).

Number	Function definition	Lambda expression
0	$0\ f\ x = x$	$0 = \lambda f.\lambda x.x$
1	$1\ f\ x = f\ x$	$1 = \lambda f.\lambda x.f\ x$
2	$2\ f\ x = f\ (f\ x)$	$2 = \lambda f.\lambda x.f\ (f\ x)$
3	$3\ f\ x = f\ (f\ (f\ x))$	$3 = \lambda f.\lambda x.f\ (f\ (f\ x))$
$\vdots$	$\vdots$	$\vdots$
n	$n\ f\ x = f^n\ x$	$n = \lambda f.\lambda x.f^{\circ n}\ x$

$$\text{plus} \equiv \lambda m.\lambda n.\lambda f.\lambda x.m\ f\ (n\ f\ x)$$

$$\text{succ} \equiv \lambda n.\lambda f.\lambda x.f\ (n\ f\ x)$$

$$\text{mult} \equiv \lambda m.\lambda n.\lambda f.\lambda x.m\ (n\ f)\ x$$

Функциональное программирование. Понятие лямбда-исчисления, основные принципы и особенности функционального программирования. Языки представители, Haskell, F#, Ocaml.

1.21.3 Парадигмы

Объектно-ориентированное программирование, основные понятия, инкапсуляция, наследование, полиморфизм. Языки-представители. Пример объектно-ориентированного кода на F#.

Логическое программирование, Пролог. SWI-prolog:`https://swish.swi-prolog.org/`

```
a(1,2).
a(2,3).
a(3,4).
a(4,1).
b(1,5).
b(5,6).
b(6,1).

reachable(X,Y) :- a(X,Z),b(Z,Y).
reachable(X,Y) :- a(X,Z),b(W,Y),reachable(Z,W).
```

функционально-логический (лямбда-пролог, ещё кто-то)

Стековое программирование, Форт. Forth (Форт): `https://www.forth.com/resources/forth-programming-language/`

Визуальное программирование, визуальное моделирование, UML, предметно-ориентированное моделирование.

UML examples: `https://www.uml-diagrams.org/index-examples.html`

Trik Studio: `https://github.com/trikset/trik-studio` MetaEdit+: `https://www.metacase.com/download/metaedit/moremacosx45.html`

Глава 2

Практика программирования, семестр 2

2.1 Лекция 13

И тут начался второй секместр.

Программный продукт, проект.

1. Программа, проект, продукт – что есть что, различия. Жизненный цикл продукта.
2. Открытый исходный код: окружение, инструменты, лицензии. Экосистема проектов с открытым исходным кодом. Непрерывная интеграция: задачи, облачный сервис AppVeyor, настройка сборки, матрица сборки. Облачный сервис Travis. GitHub Actions. Инструменты анализа качества, линтеры, покрытие тестами. Инструменты планирования и управления проектом: Trello, Pivotal Tracker. Средства коммуникации: Slack, Gitter. Багтрекер GitHub Issues. Другие средства управления проектом GitHub. Авторское право и лицензии.
3. Документация, комментирование, автоматическая генерация документации по комментариям. Публикация документации на gh-pages.
4. Визуальное моделирование, UML. Метафора моделирования, цель моделирования. Диаграммы UML. Диаграмма классов: синтаксис, синтаксис свойств, агрегация и композиция. Диаграмма компонентов. Диаграмма случаев использования. Диаграммы активностей, последовательностей, конечных автоматов. Генерация кода по диаграммам конечных автоматов. Диаграммы развёртывания. Примеры CASE-инструментов. Предметно-ориентированные визуальные языки.

FSharpLint: `https://fsprojects.github.io/FSharpLint/`.

Пример матрицы и установки FSharpLint: `https://github.com/YaccConstructor/Brahma.FSharp/blob/master/.github/workflows/build.yml`

Покрытие тестами уже есть в build.fsx.

Лицензия: если просто выложили на гитхаб, то, вообще говоря, ПО не открытое и не свободное. Скорее всего им никто не сможет воспользоваться.

Документация

ReadTheDocs (`https://readthedocs.org/`) — хостинг для документации.

Автоматическая генерация документации: Doxigen(`https://www.doxygen.nl/index.html`), JavaDoc(`https://docs.oracle.com/javase/8/docs/technotes/guides/javadoc/index.html`)

BuildDocs

ReleaseDocs

Release:https://www.jimmybyrd.me/MiniScaffold/Tutorials/Getting_Started_With_Libraries.html#Making-a-Release

UML

draw.io

https://www.lucidchart.com/

https://www.lucidchart.com/pages/uml-component-diagram

https://www.lucidchart.com/pages/uml-class-diagram#section_2

https://www.lucidchart.com/pages/uml-use-case-diagram

https://www.lucidchart.com/pages/uml-sequence-diagram

Примеры оформленных студенческих репозиториев. https://github.com/vdshk/graph-database

https://github.com/SergeyKuz1001/formal_languages_autumn_2020

https://github.com/AnzhelaSukhanova/Minimal_GDB

2.1.1 Публикация Nuget-пакета

Централизованный репозиторий: NuGet (.NET).

Ещё примеры: PyPI (Python);

Hackage (Haskell);

npm (JavaScript)

О том, как публиковать пакеты на GitHub и о том, как их оттуда ставить: https://docs.github.com/en/packages/guides/configuring-dotnet-cli-for-use-with-github-packages

2.2 Практика 10

1. **[5 баллов]** Оформление калькулятора или регулярных выражений как отдельного проекта. Создать репозиторий, снабдить всеми необходимыми элементами экосистемы: сборка, тесты, лицензия, readme. Не забыть подключить FsLint.

2. **[5 баллов]** Создать документацию. Описать цели и задачи проекта, конкретный синтаксис языка, привести примеры.

3. **[3 балла]** Создать диаграмму (наиболее подходящего типа), описывающую структуру проекта, выбранного выше. Добавить её в документацию. По итогу опубликовать nuget-пакет с интерпрететором на GitHub.

2.3 Лекция 14

Раздел 2: Объектно-ориентированное программирование 1. Основы. Инкапсуляция и наследование. Интерфейс. Множественное наследование и множественная реализация интерфейса. Абстрактный класс. Примеры ООП на F#. 2. Некоторые базовые паттерны проектирования. 3. Исключения и обработка ошибок. Бросание и обработка исключений. Перебрасывание

исключений. Объявление своих классов-исключений. Некоторые особенности использования исключений (легковесность в OCaml vs тяжеловесность в .NET). 4. GUI (winforms, GTK и т.д). Основы событийно-ориентированного программирования. Основы разработки GUI (вёрстки). Домашняя работа 2

2.4 Практика 11

Мини-IDE для языка из предыдущей работы. Проект оформляется в отдельном репозитории по всем правилам. Процесс сдачи задач прежний.

В качестве интерпретатора используется пакет напарника. Все необходимые улучшения интерпретатора, добавление новых функций, исправления оформляются как issues на GitHub в соответсвующес проекте.

1. **[2 балла]** Расширить синтаксис соответствующего языка логическими выражениями (с переменными) и условными операторами.
2. **[9 баллов]** Разработать среду разработки для полученного языка. Среда должна предоставлять следующие возможности:
   - Редактировать код.
   - Работать с файлами: создать новый, открыть существующий, сохранить изменения.
   - Выводить сообщения о (синтаксических) ошибках.
   - Запустить программу на исполнение.
   - Видеть результат исполнения в «консоли».
3. **[3 балла]** Расширить IDE возможностью подсветки синтаксиса.
4. **[8 баллов]** Расширить IDE возможностью устанавливать точки останова. В момент остановки должна быть возможность просмотреть значения всех “живых” переменных.

2.5 Практика 12

1. **[4 балла]** Реализовать консольный генератор матриц. На вход принимается размер матрицы, тип данных, количество матриц, метрика разреженности, возможно другие необходимые параметры. В результате генерируется набор файлов с матрицами в формате из первого семестра.
2. **[9 баллов]** Реализовать параллельное умножение для плотных матриц. Исследовать варианты с распараллеливанием различных циклов. Для исследования использовать созданный ранее генератор. Оформить соответствующий отчёт.
3. **[6 баллов]** Реализовать параллельное умножение матриц, представленных в виде дерева квадрантов.
4. **[10 баллов]** Сравнить производительность решений из первых двух пунктов на разных типах матриц. Для исследования использовать созданный ранее генератор. Оформить соответствующий отчёт.

2.6 Практика 11

1. **[7 баллов]** На основе Mailboxprocessor или Hopac реализовать решение, в котором есть следующие конкурентно выполняющиеся типы задачи:

   - Подгрузка пар матриц из файлов (для генерации использовать генератор из предыдущей работы)
   - Различные алгоритмы перемножения матриц (для разреженных, для плотных параллельно и последовательно)
   - Балансировщик, знающий, кому какие матрицы отправлять для обработки.

   Предусмотреть два режима работы:

   - Обработать все матрицы, доступные на входе
   - Обработать заданное количество пар матриц

2. **[14 баллов]** Проанализировать масштабируемость полученной системы. Какое количество конкурентных задач оптимально для определённой конфигурации системы? Оформить соответствующий отчёт.

Глава 3

Практика программирования, семестр 2

Глава 4

САКОД

4.1 Введение

Баллы можно получать за домашние задачи и доклады.

Домашние задачи сдаются через оформленный на GitHub репозиторий. Кроме кода, репозиторий должен содержать readme с информацией о проекте, инструкциями по запуску и установке, тесты, подключённый и настроенный CI, build script для автоматизации установки и сборки. Разрабатываемые решения должны быть кроссплатформенными (или хотя бы безболезненно запускаться на Ubuntu).

Сдача домашней работы — запрос ревью соответствующего реквеста. Реквест должен содержать только изменения, имеющие непосредственное отношение к сдаваемой работе. Работа состоит из нескольких задач и сдавать можно отдельные задачи. У каждой работы есть дедлайн, после которого любая сданная задача оценивается в четверть от полного балла.

Все отчёты и презентации готовятся в TeX и снабжаются соответствующим скриптом сборки. Публикуются на GitHub как исходники, так и результирующий PDF-файл.

Темы докладов:

1. Инфраструктура проекта на примере F#: FAKE, Scaffold, FsCheck
2. Веб-приложения в комбинаторном стиле: Suave
3. Программирование, ориентированное на обработку данных: Type Providers, F# Data
4. Распределённое программирование на F#: MBrace
5. Программирование GPGPU на F#: Alea CUDA
6. Мапрограммирование на F#: F# code quotations

- 91–100 : A (отл)
- 81–90 : B (хор)
- 71–80 : C (хор)
- 61–70 : D (удвл)
- 51–60 : E (удвл)
- 0–50 : F (неуд)

4.2 Основы обработки изображений

4.2.1 Форматы изображений

Векторный, растровый.

Нас интересует растровый.

И для простоты сразу битмапа (bmp): двумерный массив пикселей, где для каждого хранится цвет. Естественно, ещё и метаданные вокруг, но они, в основном, про то, как ситать файл, а не про само изображение. Цвет — либо RGB, либо градации серого (Grayscale). На пиксель от 1 до 64 бит. В grayscale 16 или 32.

https://docs.microsoft.com/en-us/dotnet/api/system.drawing.bitmap?view=dotnet-plat-ext-5.0

4.2.2 Цифровые фильтры изображений

Собель для поиска границ, размытие по Гауссу, машинное обучение.

Примеры:

- `https://www.codingame.com/playgrounds/2524/basic-image-manipulation/filtering`
- `https://lodev.org/cgtutor/filtering.html`

4.2.3 Домашняя работа 1

1. **4 балла.** Реализовать приложение с графическим интерфейсом пользователя, позволяющее открыть папку с изображениями, выбрать изображение, просмотреть его, просмотреть информацию о нём (размер в пикселях, размер в мегабайтах).

2. **3 балла.** Расширить приложение графической компонентой задания матричного фильтра. Необходимо предусмотреть возможность выбора типа фильтра, дефолтных значений, размера фильтра, корректировку весов.

3. **3 балла.** Расширить приложение возможностью отображать одновременно два изображения: до и после применения фильтра. Предусмотреть возможность сохранять результат применения фильтра.

4. **8 баллов.** Реализовать применение матричных фильтров с использованием GPGPU. Интегрировать с разработанным графическим интерфейсом. Предусмотреть возможность применения нескольких фильтров последовательно.

5. **8 баллов.** Расширить разрабатываемое приложение возможностью потоковой обработки изображений: выбираем папку с изображениями и ко всем применяем заданные фильтры. Результаты применения фильтров сохраняются в отдельную выбранную папку.

6. **10 баллов.** Подготовить отчёт с анализом производительности и масштабируемости полученного решения.

4.3 Лекция 2: Структуры данных и алгоритмы линейной алгебры

Основы линейной алгебры: примитивы (матрицы, вектора, поля, кольца, полукольца) и их свойства (конечность и идемпотентность, коммутативность и т.д.), операции над матрицами и векторами: поэлементные, умножение матриц, умножение матрицы на вектор, тензорное произведение, транспонирование.

Разреженное представление матриц и векторов. Основные форматы разреженного представления матриц: покоординатный, CSR, Quad-tree. Специализированные форматы: диагональные матрицы, HiCOO, и др. Их преимущества и недостатки.

4.4 Лекция 2: Структуры данных и алгоритмы линейной алгебры

Параллельная обработка разреженных матриц и векторов. Особенности соответствующих алгоритмов для GPGPU.

Прикладные задачи, сводимые к линейной алгебре. Обработка графов, GraphBLAS API. BFS, транзитивное замыкание, кратчайшие пути, подсчёт треугольников, минимальное остовное дерево. Пересечение автоматов, объединение автоматов.

4.4.1 Домшняя работа 2

Данная работа посвящена реализации алгоритмов анализа графов с использованием операций линейной алгебры. Необходимо выбрать минимум 3 различных алгоритма (требующих различных операций). Неполный список: подсчёт треугольников, BFS, минимальное остовное дерево, поиск кратчайших путей. Можно предложить свой. Выбранные алгоритмы необходимо реализовать с использованием различных библиотек и сравнить их производительность.

1. **3 балла.** Реализовать выбранные алгоритмы на (py)graphblas.
2. **3 балла.** Реализовать выбранные алгоритмы на sciPy.
3. **3 балла.** Реализовать выбранные на стандартной библиотеке для анализа графов (можно выбрать в зависимости от языка).
4. **10 баллов.** Сравнить производительность полученных реализаций, составить отчёт.

4.5 Лекция 3: Структуры данных и алгоритмы линейной алгебры

Раздел 3: Основы анализа сложности алгоритмов. 1. Введение. Классическая теория сложности, анализ сложности алгоритмов в теории и что это значит для практики. 2. Основы fine-grained complexity. 3. Основы анализа сложности параллельных алгоритмов. Домашняя работа 5. Выбрать любой алгоритм на графах из реализованных и провести его анализ сложности. 4. Проверочная работа

4.5.1 Домшняя работа 3

1. **7 баллов.** Бибилиотека плотных операций на ГПУ.
2. **5 баллов.** Задачи на графах.
3. **10 баллов.** Производительность.
4. **13 баллов.** Попробовать что-то реализовать на разреженной алгебре и сравнить производиительность.