ДАННЫЕ И ПЕРЕМЕННЫЕ В SCHEME

Выполнил Гребенкин Егор Дмитриевич
Группа 5030102/20202

Данные в Scheme: общая идея

Scheme — динамически типизированный язык:

- тип «сидит» в значении, а не в переменной
- одна переменная может ссылаться на значения разных типов — это нормальное поведение

Обобщённые (параметрические) типы как в C++/Rust/Java не входят в ядро Scheme.
Вместо этого полагаемся на:

- динамическую типизацию
- соглашения, проверки (error), контракты и тесты
- расширения реализаций (например, контракты в Racket)

Примитивные и составные типы данных

Примитивные (ядро):

- числа (целые, рациональные, вещественные, комплексные — зависит от реализации)
- булевы: #t, #f
- символы: 'name (очень важны в Lisp)
- строки: "text"
- символы/знаки: #\A
- процедуры (функции первого класса)

Составные:

- пары (cons) и списки (цепочки пар)
- векторы ((vector 1 2 3))

Пары, списки и векторы

Пары и списки:

- (cons a b) — пара
- (car pair), (cdr pair) — голова и хвост
- список (1 2 3) = (cons 1 (cons 2 (cons 3 '())))
- dotted pair: (cons 1 2) → (1 . 2) (не список!)

Векторы:

- создание: (vector 1 2 3)
- доступ: (vector-ref v 0)
- мутация: (vector-set! v 0 999)

Пользовательские типы и проверки

Пользовательские типы:

- struct в Racket (удобно)
- records по SRFI-9 в других реализациях
- или объекты через замыкания (позже)

Предикаты типов:

- number?, string?, symbol?, boolean?, pair?, list?, null?, vector?, procedure? и др.

Преобразования:

- stringnumber, symbolstring

Сравнение значений в Scheme: eq?, eqv?, equal?

Эта тема — один из самых частых источников ошибок у новичков.
Существуют три основных предиката сравнения:

- eq? — сравнение по идентичности объекта (один и тот же объект в памяти)
- eqv? — промежуточный вариант (как eq?, но учитывает числовое/символьное равенство)
- equal? — структурное (рекурсивное) сравнение содержимого

Сравнение значений: примеры и рекомендации

```
(eq? 'a 'a) ; => #t (    -)
(eq? "abc" "abc") ; => #f  #t (  !)
(equal? "abc" "abc") ; => #t (  )

(eq? '(1 2) '(1 2)) ; =>  #f ( )
(equal? '(1 2) '(1 2)) ; => #t (  )

(eqv? 1 1) ; => #t
(eqv? 1.0 1.0) ; => #t (   eq?   )
```

Практические правила:

- Для символов, булевых и идентичности — eq?
- Для строк, списков, векторов — equal?
- eqv? — редко (в основном для чисел/символов по стандарту)

Переменные и привязки

Глобальные:

```
(define x 10)
(define (f a b) (+ a b))
```

Локальные:

- let — параллельные (одновременно)
- let* — последовательные
- letrec — для взаимной рекурсии

Мутация:

- (set! x 42) — изменение переменной
- set-car!, vector-set! и др.

Мутабельность и семантика передачи

В Scheme память управляется GC, нет модели владения как в Rust.
Ключевой вывод:

- Если структура неизменяемая (или не мутируется) — можно думать «по значению»
- Если мутабельная — несколько переменных могут ссылаться на один объект → изменения видны везде

Это критично для отладки, конкурентности и проектирования API.

Числовая башня и точность

Scheme стремится к математической корректности (numeric tower):

- целые → рациональные → вещественные → комплексные

Exact vs inexact:

- exact — точные (1, 1/3)
- inexact — приближённые (1.0, float-вычисления)

Примеры:

- (/ 7 2) → 7/2 (exact)
- (/ 7 2.0) → 3.5 (inexact)
- (+ 0.1 0.2) → 0.30000000000000004

Вывод: для точной математики — exact; для физических расчётов — inexact + эпсилон-сравнения.

Что чаще всего используют в практике

- Символ ('alpha) — теги, enum, case
- Строка ("alpha") — текст, I/O
- Список — коллекция по умолчанию
- Вектор — индексируемая коллекция
- struct/record — именованные поля
- alist — простой «словарь» (список пар (key . value))

Предикаты + ручные проверки (error) — «контракты по соглашению».

Демонстрационные программы

- types-demo.scm — типы, сравнения, преобразования, мутабельность
- scope-demo.scm — области видимости, let*/letrec, set!
- records-demo.scm — пользовательский тип struct
- numbers-demo.scm — exact/inexact, проблемы точности

Основные выводы демо

types-demo.scm (фрагменты):

```
number: 42 symbol: alpha list: (1 2 3)
== ==
eq? symbols: #t
equal? strings: #t
equal? lists: #t
== ==
xs after set-car!: (999 2 3)
```

scope-demo.scm (фрагмент):

```
global x after let: 10 let*: b: 11
```

numbers-demo.scm:

```
(/ 7 2): 7/2
0.1 + 0.2: 0.30000000000000004 (= ... 0.3): #f
```

Как запустить демо

В DrRacket:

1) DrRacket
2) File Open... .scm 02/src
3) Language Choose Language... (Scheme/Racket)
4) Run

Из терминала (опционально):

```
racket types-demo.scm
racket scope-demo.scm
racket records-demo.scm
racket numbers-demo.scm
```

Литература и дополнительные темы

- references.md
- Документация Racket (типы, struct, сравнения)
- Scheme Reports (R5RS/R7RS) — семантика eq?/equal?, область видимости

Дополнительно (не вошли подробно):

- quote, quasiquote — данные как код
- write/read — round-trip сериализация
- association lists (alist) как простые словари