Лекция 6б. Типы данных и типизация

Коновалов А. В.

27 сентября 2022 г.

Типы данных

Тип данных — множество значений, множество операций над ними и способ хранения в памяти компьютера (машинное представление).

Абстрактный тип данных — множество значений и множество операций над ними, т.е. способ хранения не задан.

Первая классификация типов данных:

1. Простые — неделимые порции данных: число, символ, литера.
2. Составные — содержащие значения других типов: cons-ячейка, список, вектор, строка.

Вторая классификация типов данных:

1. Встроенные типы данных — уже заранее есть в языке.
2. Пользовательские — их определяет пользователь.

В ряде языков программирования (например, в Си) есть встроенные в язык средства для определения пользовательских типов данных. Например, в Си встроены различные числовые типы. Пользователь на их основе может создавать массивы, массивы массивов, структуры, объединения и т.д.

В языке Scheme нет языковых средств для определения новых типов данных. Вместо этого пользователь придумывает способ представления некоторого значения при помощи встроенных типов данных и описывает операции над ним в виде набора процедур (иногда, макросов).

Т.е. проектируем представление типа данных и набор операций. При этом не рекомендуется работать с типом данных в обход предоставленных операций.

Если мы задокументируем только набор операций, но не опишем представление, то мы создали *абстрактный тип данных.*

Для типов данных языка Scheme обычно определены четыре вида операций:

- **конструктор** — процедура, имя которой имеет вид `make-‹имя-типа›`, например, `make-vector`, `make-set` (см. дз), конструктор предназначен для создания новых значений данного типа,
- **предикат типа** — процедура, возвращающая `#t`, если её аргумент является значением данного типа, имеет имя `‹имя-типа›?`: `vector?`, `set?` (см. дз), `multi-vector?` (см. дз),
- **модификаторы** — операции, меняющие на месте содержимое объекта, их имя имеет вид `‹тип›-‹операция›!`, например, `vector-set!`, `multivector-set!` (см. дз),
- **прочие операции** имеют имя вида `‹тип›-‹операция›`, `vector-ref`, `set-union`, `string-append` и т.д.

Пользовательские типы данных часто представляют как списки, первым элементом которых является символ с именем типа, а остальные — хранимые значения.

Пример. Тип данных — круг.

```
(define (make-circle x y r)
  (list 'cicrle x y r))

(define (circle? c)
  (and (list? c) (equal? (car c) 'circle)))

(define (circle-center c)
  (list (cadr c) (caddr c)))        ; (cadr xs) = (car (cdr xs

(define (circle-radius c)
  (cadddr c))
```

```
(define (circle-set-center! c p)
  (let ((x (car p))
        (y (cadr p)))
    (set-car! (cdr c) x)
    (set-car! (cddr c) y)
    c))

(define (circle-set-radius! c r)
  (set-car! (cdddr c) r)
  c)
```

Типизация и системы типов

Система типов — совокупность правил в языках программирования, назначающих свойства, именуемые типами, различным конструкциям, составляющим программу — переменные, выражения, функции и модули.

Определение по Пирсу, **система типов** — разрешимый синтаксический метод доказательства отсутствия определённых поведений программы путём классификации конструкции в соответствии с видами вычисляемых значений.

Классификации систем типов:

1. Наличие системы типов: есть/нет.
2. Типизация статическая/динамическая.
3. Типизация явная/неявная.
4. Типизация сильная/слабая.

Наличие системы типов:

- Нет: язык ассемблера, язык FORTH, язык B (Би) — предшественник Си.
- Есть: все остальные языки.

Статическая и динамическая типизация:

- **Статическая типизация** — у каждой именованной сущности (переменной, функции…) есть свой фиксированный тип, он не меняется в процессе выполнения программы. Примеры: Си, C++, Java, Haskell, Rust, Go.
- **Динамическая** — тип переменной/функции известен только во время выполнения программы. Примеры: Scheme, JavaScript, Python.

Явная и неявная типизация:

- **Явная** — тип данных для сущностей явно записывается в программе. Например, `int x` в языке Си. Языки с явной типизацией: Си, C++, Java и т.д.
- **Неявная** — тип данных можно не указывать. Неявная типизация характерна прежде всего для динамически типизированных языков. В статически типизированных языках используется совместно с выводом типов. Вывод типов переменных присутствует в следующих языках: C++ (ключевое слово `auto`), Go (когда тип переменной не указан), Rust, Haskell и т.д.

```c
/* Язык Си, тип указывается явно */
int x = 100;

// Язык C++
// тип выводится компилятором
auto x = 100;       /* int */
auto y = "abc";     /* const char * */

// Язык Go
var x int = 100
var y = 200         /* тип выведет компилятор */
```

Типизация сильная/слабая:

- **Сильная** — неявные преобразования типов запрещены. Например, нельзя сложить строку и число. Языки с сильной типизацией: Scheme, Python, Haskell.
- **Слабая типизация** — неявные преобразования допустимы. Например, в JavaScript при сложении строки с числом число преобразуется в строку. Если в JavaScript в переменной лежит строка с последовательностью цифр, то, при умножении её на число, она неявно преобразуется в число: `'1000' * 5` → `5000`. Примеры языков: JavaScript, Си, Perl, PHP.