Bahasa Pemrograman :: Pemrograman List

Julio Adisantoso
ILKOM IPB

2 Maret 2011

Ekspresi

- Ekspresi Scheme dituliskan dengan menggunakan notasi Cambridge-prefix.
- Seluruh instruksi dalam Scheme membentuk pola list, dimana data dan program (atau fungsi) direpresentasikan sebagai list.
- Contoh

```
(+)                     ;0
(+ 5)                   ;5
(+ 5 4 3 2 1)           ;15
(*)                     ;1
(* 5)                   ;5
(* 1 2 3 4 5)           ;120
(+ (* 5 4) (- 6 2))     ;24
```

Define

- Variabel global didefinisikan dalam Scheme menggunakan fungsi define.
- Contoh

```
(define f 120)          ;membuat f bernilai 120
(define x f)            ;membuat x bernilai 120
(define x 'f)           ;membuat x bernilai simbol f
(define warna 'merah)   ;membuat warna bernilai merah
(define warna merah)    ;error, belum terdefinisi
```

Evaluasi Ekspresi

Ada tiga aturan

- Nama atau simbol diganti oleh nilai yang bersesuaian. Contoh:

```
f               ;bernilai 120
(+ f 5)         ;bernilai 125
```

- **list** dievaluasi sebagai pemanggilan fungsi yang ditulis dalam bentuk notasi Cambridge-prefix:

```
(+)                 ;panggil fungsi + tanpa argumen
(+ 5)               ;panggil fungsi + dengan 1 argumen
(+ 5 4 3 2 1)       ;panggil fungsi + dengan 5 argumen
(+ (5 4 3 2 1))     ;error, evaluasi 5 sebagai fungsi
(f)                 ;error, f bukan fungsi
```

Evaluasi Ekspresi

Ada tiga aturan

- Nama atau simbol diganti oleh nilai yang bersesuaian nilai konstanta mengevaluasi nilai dirinya sendiri. Contoh:

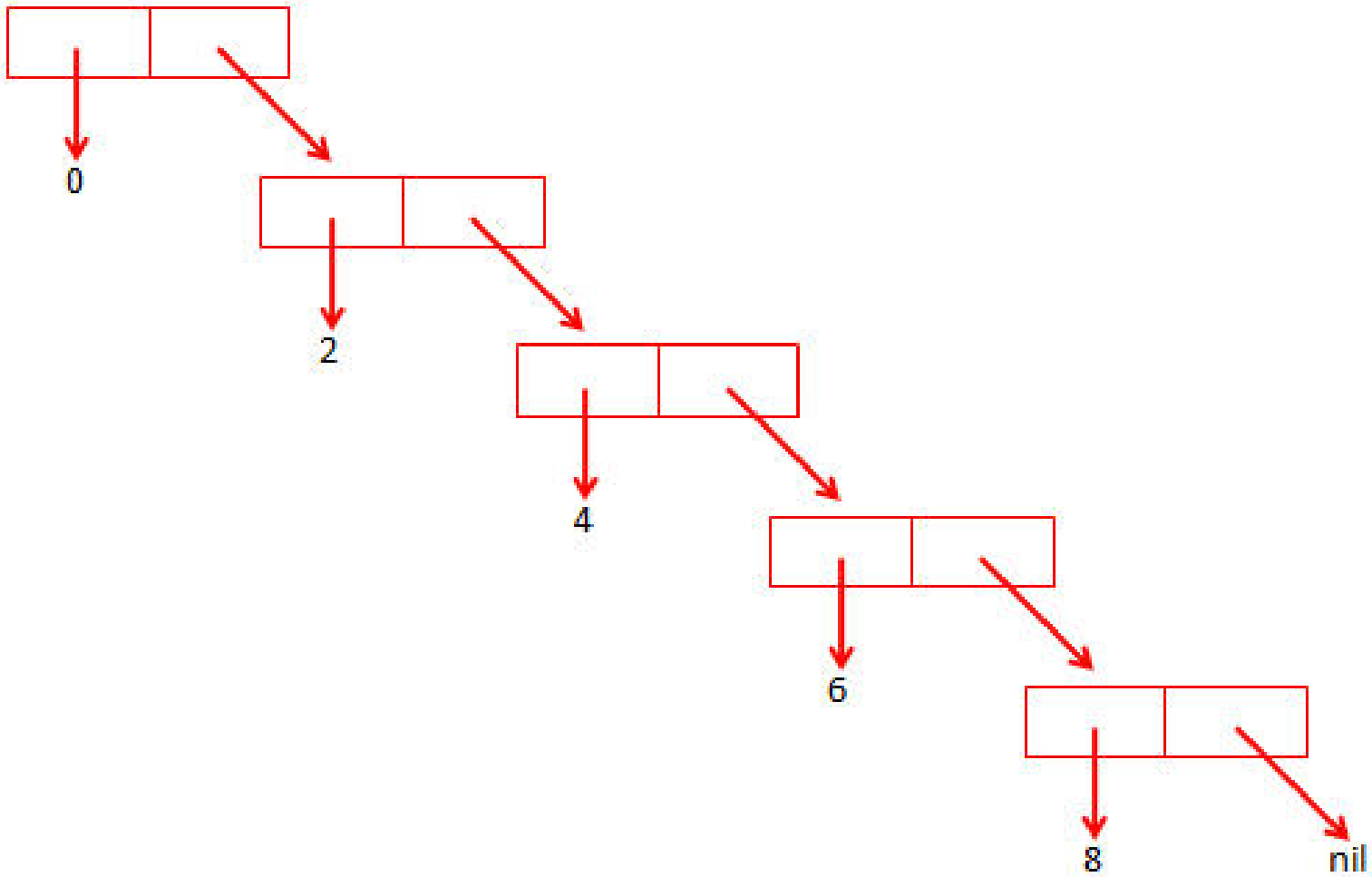
```
5       ;bernilai 5
#f      ;bernilai false
#t      ;bernilai true
```

List

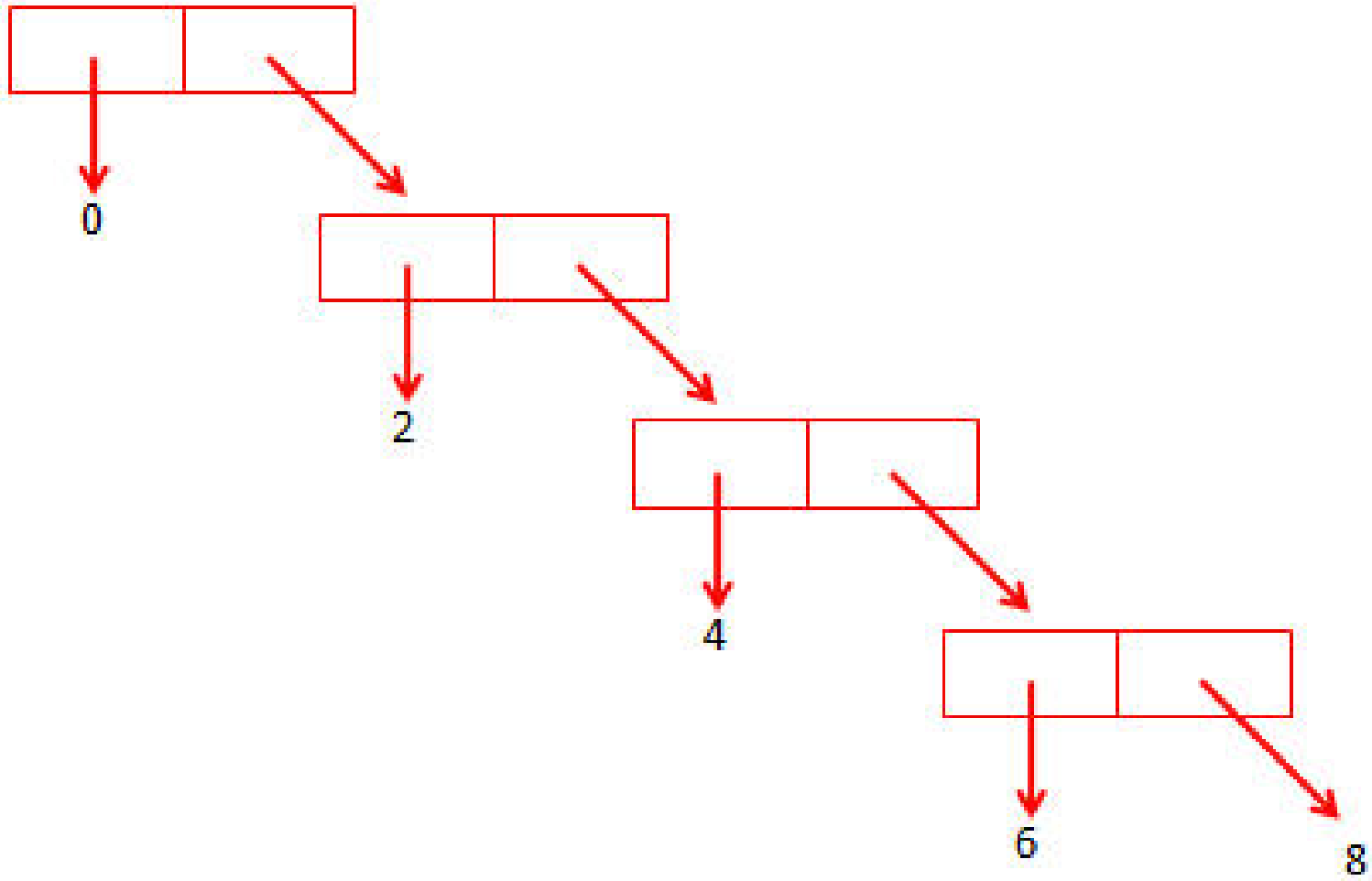
- Struktur data dasar dalam Scheme adalah list
- List digunakan untuk instruksi dan data
- Simbol () melambangkan list kosong, atau juga dapat dituliskan sebagai null
- Contoh:

```
(define genap '(0 2 4 6 8))
(define ganjil '(1 3 5 7 9))
(define genap2 '(0 2 4 6 . 8))
(define kosong '())
(define kosong2 null)
```

(0 2 4 6 8)



(0 2 4 6 . 8)



Fungsi Dasar List

- Fungsi dasar yang digunakan untuk menyusun suatu list adalah cons
- Membutuhkan dua argumen dimana argumen kedua haruslah list.
- Contoh:

```
(cons 8 null)                          ;(8)
(cons 6 (cons 8 null))                 ;(6 8)
(cons 4 (cons 6 (cons 8 null)))        ;(4 6 8)
(cons 4 (cons 6 (cons 8 9)))           ;(4 6 8 . 9)
```

Fungsi car dan cdr

- Node di dalam list mempunyai dua bagian, elemen pertama disebut sebagai head atau kepala, dan list selain elemen pertama disebut tail atau ekor.
- Fungsi car mengembalikan head atau elemen pertama dari list
- Fungsi cdr mengembalikan list selain elemen pertama atau tail.
- Contoh:

```
(car genap)          ;0
(cdr genap)          ;(2 4 6 8)
(car (cdr genap))    ;2, bisa ditulis (cadr genap)
(cdr (cdr genap))    ;(4 6 8), atau (cddr genap)
(car '(6 8))         ;6
(car (cons 6 8))     ;6
(cdr '(6 8))         ;(8)
(cdr (cons 6 8))     ;8, bukan (8)
(car '(8))           ;8
(cdr '(8))           ;()
```

Fungsi list dan append

- Fungsi tingkat tinggi untuk menyusun list adalah fungsi list dan append.
- Fungsi list menyusun list dari sekumpulan argumen yang diberikan:

```
(list 1 2 3 4)              ;(1 2 3 4)
(list '(1 2) '(3 4) 5)      ;((1 2) (3 4) 5)
(list genap ganjil)         ;((0 2 4 6 8) (1 3 5 7 9))
(list 'genap 'ganjil)       ;(genap ganjil)
```

Fungsi list dan append

- Fungsi append mengambil elemen-elemen dari dua list argumen dan menyusun list baru
- atau menggabungkan elemen list pada argumen kedua ke dalam list pada argumen pertama.
- Contoh:

```
(append '(1 2) '(3 4))      ;(1 2 3 4)
(append genap ganjil)       ;(0 2 4 6 8 1 3 5 7 9)
(append '(1 2) '())         ;(1 2)
(append '(1 2) (list 3)     ;(1 2 3)
```

Fungsi memeriksa list

```
(null? '())           ;#t
(null? genap)         ;#f
(null? '(1 2 3))      ;#f
(null? 5)             ;#f

(equal? 5 5)                    ;#t
(equal? 5 1)                    ;#f
(equal? '(1 2) '(1 2))          ;#t
(equal? 5 '(5))                 ;#f
(equal? '(1 2 3) '(1 (2 3)))    ;#f
(equal? '(1 2) '(2 1))          ;#f
(equal? '() '())                ;#t
```

Mendefinisikan Fungsi

- Fungsi dalam Scheme didefinisikan dengan menggunakan define dengan bentuk umum seperti berikut:

```
(define ( name arguments ) function-body )
```

- Jadi, fungsi **min** untuk menentukan bilangan minimum (terkecil) dari dua bilangan adalah:

```
(define (min x y)
   (if (< x y) x y)
)
```

Fungsi Rekursif

- Fungsi untuk menghitung nilai faktorial dari sebuah bilangan bulat positif adalah:

```
(define (faktorial n)
   (if (equal? n 0)
       1
       (* n (faktorial (- n 1))))
   )
)
```

- Fungsi ini sebagai implementasi dari fungsi faktorial yang biasa dituliskan dalam notasi matematika sebagai berikut:

$$f(n) = \begin{cases} 1 & ; \text{jika } n = 0 \\ n * f(n-1) & ; \text{selainnya} \end{cases}$$

Fungsi Rekursif List

- Fungsi rekursif juga sering digunakan untuk mengolah list.
- Sebagai contoh, berikut adalah fungsi untuk menghitung penjumlahan dari setiap elemen list sederhana:

```
(define (sum alist)
   (if (null? alist) 0
       (+ (car alist) (sum (cdr alist)))
   )
)
```

- Sehingga jika diproses (sum '(2 4 6 8)) akan menghasilkan nilai 20.

Latihan

- Buat program menentukan nilai absolut (harga mutlak) dari sebuah bilangan bulat.

```
> (abs -10)
10
```

- Buat program menghitung banyaknya elemen list yang nilainya kurang dari suatu nilai tertentu

```
> (hitung 10 '(1 2 15 7 25 10))
3
```

- Buat program memeriksa keanggotaan suatu elemen dalam list sederhana

```
> (anggota? 10 '(5 10 15 20))
#t
```

Latihan

- Buat program menghapus elemen list yang sama dan bersebelahan

```
> (hapus '("a" "b" "b" "b" "c" "c"))
("a" "b" "c")
```

- Buat program mengganti elemen list

```
> (subst 'x 2 '(1 2 3 2 1))
(1 x 3 x 1)
```