Próximo:Listas de construção, Anterior:Predicados relacionados à lista, Acima:Listas [Conteúdo][Índice

]

5.3 Acessando Elementos de Listas

Função: carro *cons-cell*

Esta função retorna o valor referido pelo primeiro slot da cons cell *cons-cell* . Em outras palavras, ele retorna o CAR de *cons-cell* .

Como um caso especial, se *cons-cell* for `nil`, esta função retornará `nil`. Portanto, qualquer lista é um argumento válido. Um erro é sinalizado se o argumento não for uma célula contra ou `nil`.

```
(carro '(abc))
     ⇒ a
(car '())
     ⇒ nil
```

Função: cdr *cons-cell*

Esta função retorna o valor referido pelo segundo slot da cons cell *cons-cell* . Em outras palavras, ele retorna o CDR de *cons-cell* .

Como um caso especial, se *cons-cell* for `nil`, esta função retornará `nil`; portanto, qualquer lista é um argumento válido. Um erro é sinalizado se o argumento não for uma célula contra ou `nil`.

```
(cdr '(abc))
     ⇒ (bc)
(cdr '())
     ⇒ nil
```

Função: *objeto* seguro para carro

Esta função permite tirar o CAR de uma célula de contras evitando erros para outros tipos de dados. Retorna o CAR do *objeto* se o *objeto* for uma célula contra, `nil`caso contrário. Isso contrasta com `car`, que sinaliza um erro se o *objeto* não for uma lista.

```
(car-safe object )
≡
(let ((x object ))
  (if (consp x)
      (car x)
    nil))
```

Função: *objeto* seguro para cdr

Esta função permite obter o CDR de uma célula de contras, evitando erros para outros tipos de dados. Ele retorna o CDR do *objeto* se o *objeto* for uma célula contra, `nil`caso contrário. Isso contrasta com `cdr`, que sinaliza um erro se o *objeto* não for uma lista.

```
( objeto seguro para cdr )
≡
(let ((x objeto ))
  (if (consp x)
      (cdr x)
    nil))
```

Macro: pop *listname*

Essa macro fornece uma maneira conveniente de examinar o CAR de uma lista e retirá-lo da lista, tudo de uma vez. Ele opera na lista armazenada em *listname* . Ele remove o primeiro elemento da lista, salva o CDR em *listname* e retorna o elemento removido.

No caso mais simples, *nomedalista* é um símbolo sem aspas que nomeia uma lista; nesse caso, essa macro é equivalente a `(prog1 (car listname) (setq listname (cdr listname)))`.

```
x
     ⇒ (abc)
(pop x)
     ⇒ a
x
     ⇒ (bc)
```

Mais geralmente, *listname* pode ser uma variável generalizada. Nesse caso, essa macro salva em *nomedalista* usando `setf`. Consulte Variáveis Generalizadas .

Para a `push`macro, que adiciona um elemento a uma lista, consulte Variáveis de lista .

Função: enésima *lista n*

Esta função retorna o *enésimo* elemento de *list* . Os elementos são numerados começando com zero, então o CAR da *lista* é o elemento número zero. Se o comprimento da *lista* for *n* ou menor, o valor será `nil`.

```
(nº 2 '(1 2 3 4))
     ⇒ 3
(nth 10 '(1 2 3 4))
     ⇒ nil

(nth nx) ≡ (car (nthcdr nx))
```

A função `elt`é semelhante, mas se aplica a qualquer tipo de sequência. Por razões históricas, toma seus argumentos na ordem oposta. Consulte Funções de seqüência .

Função: nthcdr *n lista*

Esta função retorna o *enésimo* CDR da *lista* . Em outras palavras, ele pula os primeiros *n* links da *lista* e retorna o que segue.

Se *n* for zero, `nthcdr`retorna tudo de *list* . Se o comprimento da *lista* for *n* ou menor, `nthcdr`retornará `nil`.

```
(nthcdr 1 '(1 2 3 4))
     ⇒ (2 3 4)
```

```
(nthcdr 10 '(1 2 3 4))
     ⇒ nil
(nthcdr 0 '(1 2 3 4))
     ⇒ (1 2 3 4)
```

Função: última *lista &opcional n*

Esta função retorna o último link da *lista* . O `car`deste link é o último elemento da lista. Se a *lista* for nula, `nil`será retornada. Se *n* for diferente `nil`de , o *n* -ésimo último link será retornado, ou toda a *lista* se *n* for maior que o comprimento da *lista* .

Função: *lista* de comprimento seguro

Esta função retorna o comprimento de *list* , sem risco de erro ou loop infinito. Geralmente, retorna o número de células contras distintas na lista. No entanto, para listas circulares, o valor é apenas um limite superior; muitas vezes é muito grande.

Se a *lista* não for `nil`ou uma célula contras, `safe-length` retornará 0.

A maneira mais comum de calcular o comprimento de uma lista, quando você não está preocupado que ela possa ser circular, é com `length`. Consulte Funções de seqüência .

Função: caar *cons-cell*

Este é o mesmo que . `(car (car cons-cell))`

Função: cadr *cons-cell*

Isso é o mesmo que ou . `(car (cdr cons-cell))` `(nth 1 cons-cell)`

Função: cdar *cons-cell*

Este é o mesmo que . `(cdr (car cons-cell))`

Função: cddr *cons-cell*

Isso é o mesmo que ou . `(cdr (cdr cons-cell))` `(nthcdr 2 cons-cell)`

Além do acima, 24 composições adicionais de `ca`e `cdr`são definidas como e , onde cada uma é ou . , , e escolha o segundo, terceiro ou quarto elementos de uma lista, respectivamente. `c`*xxxx*`r` `c`*xxxxx*`r` *x* `adcadrcaddrcadddr` `cl-lib`fornece o mesmo sob os nomes `cl-second`, `cl-third`e `cl-fourth`. Consulte Funções de lista em extensões do Common Lisp .

Função: butlast *x &opcional n*

Esta função retorna a lista *x* com o último elemento, ou os últimos *n* elementos, removidos. Se *n* for maior que zero faz uma cópia da lista para não danificar a lista original. Em geral, retornará uma lista igual a *x* . `(append (butlast x n) (last x n))`

Função: nbutlast *x &opcional n*

Esta é uma versão `butlast`que funciona modificando destrutivamente `cdr`o elemento apropriado, em vez de fazer uma cópia da lista.

Próximo:Listas de construção, Anterior:Predicados relacionados à lista, Acima:Listas [Conteúdo][Índice

]