MP20 @ II UWr 5 maja 2020 r.

Lista zagadnień nr 9

Mutowalne struktury danych

Ćwiczenie 1.

Zdefiniuj procedurę `mreverse!`, która odwraca listę mutowalną „w miejscu" (czyli nie tworzy nowych bloczków `mcons`-em, a odpowiednio przepina wskaźniki).

Ćwiczenie 2.

Podobnie jak pary, struktury też mogą mieć mutowalne pola. Np. *dwukierunkową listę* budujemy przy użyciu wartości `null` i następujących bloczków:

```
(struct bdlist (v [prev #:mutable] [next #:mutable]))
```

Racket prócz standardowych procedur stworzy też procedury `set-bdlist-prev!` oraz `set-bdlist-next!` do modyfikowania tych pól w istniejących strukturach.

Zdefiniuj procedury:

- `list->bdlist`, która przekształca zwykłą listę w listę dwukierunkową.
- `bdfilter`, która działa jak `filter`, ale dla list dwukierunkowych.

Ćwiczenie 3.

Do rozwiązania poprzedniego zadania dodaj procedury `cycle-bdlist!`, która bierze jako argument listę dwukierunkową i ją zayckla przepinając odpowiednie wskaźniki, oraz `decycle-bdlist!`, która z zacyklonej listy tworzy niezacykloną listę dwukierunkową.

Interpreter i procedury rekurencyjne

Ćwiczenie 4.

Rozbuduj nasz język programowania z pliku `letrec.rkt` o symbole. W wersji trudniejszej: o cytowanie.

Wskazówka: Żeby zdefiniować składnię konkretną, przyda się wiedzieć, że w Rackecie cytowanie zapisane jako `'expr` to lukier syntaktyczny dla formy specjalnej `quote`, np.

```
> (quote (+ 2 2))
'(+ 2 2)
> (list? ''x)
#t
> (car ''x)
'quote
> (cadr ''x)
'x
```

Ćwiczenie 5.

Napisz w naszym języku z pliku `letrec.rkt` procedury `map`, `filter` i `append`.

Ćwiczenie 6.

Napisz w naszym języku sortowanie przez wstawianie. Sprawdź eksperymentalnie, jaka jest różnica czasu wykonania między sortowaniem przez wstawianie zaimplementowanym bezpośrednio w Rackecie, a zaimplementowanym w naszym języku.

Ćwiczenie 7.

Rozbuduj nasz język o formy specjalne `display` i `read` (które możemy interpretować używając racketowych form specjalnych `display` i `read`), a także o blok `begin` (np. wyrażenie `(begin e1 e2 e3 e4)` wylicza po kolei wyrażenia `e1` aż do `e4`, a wartością całego bloku jest wartość wyrażenia `e4`).

Napisz program, który w kółko powtarza: wczytaj liczbę z wejścia i wypisz ją na wyjście. Pętle przerywa wczytanie liczby 0.

Ćwiczenie 8.

Dodaj do języka z pliku `letrec.rkt` lukier syntaktyczny dla definicji wzajemnie rekurencyjnych ze składnią konkretną jak poniżej:

```
(letrec* ([x e1]
          [y e2]
          [z e3])
  e)
```

Język WHILE

Ćwiczenie 9.

Napisz w języku WHILE program do liczenia n-tej liczby Fibonacciego i największego wspólnego dzielnika dwóch liczb.

Ćwiczenie 10.

Dodaj do języka pętlę `for` (np. z semantyką taką jak w C).

Ćwiczenie 11.

Dodaj do języka konstrukcję ++ ze składnią konkretną np. `(++ x)`, gdzie `x` jest zmienną. Semantyka to: zwiększ wartość zmiennej `x` o 1.

Ćwiczenie 12.

Dodaj do języka WHILE konstrukcję `break`, która zrywa wykonanie najbardziej wewnętrznej pętli.