テキスト処理 第5回 (2007-05-22)
compact, ack レポート説明

田中哲
産業技術総合研究所
情報技術研究部門
akr@isc.senshu-u.ac.jp
http://staff.aist.go.jp/tanaka-akira/textprocess-2007/

前回のレポート

- 以下の関数を実装してその内容を解説せよ
  - compact
  - アッカーマン関数ack
- レポートには以下の内容を含むものとする
  - 実装
  - 動作の様子
  - 解説
- ∠切 2007-05-22 16:20
- HIPLUS
- 拡張子が txt なテキストファイルが望ましい

compact

- Array#compact に似た機能を持つ関数
- 配列を引数として受け取り、nil を除いた新しい配列を返す
- def compact(ary) ... end と定義して ... を埋める
- Array#compact を使うのは禁止
- 動作の例

```
compact([])          #=> []
compact([1,nil,2])   #=> [1,2]
compact([nil,nil])   #=> []
```

compact の実装 (1)

```
def compact(ary)
  ary.find_all {|v| v != nil }
end
```

- find_all を使う
- reject なども使える

compact の実装 (2)

```
def compact(ary)
  ret = []
  ary.each {|v|
    ret << v if v != nil
  }
  ret
end
```

- ret に << v で加えていく
- Array#push も使える
- 講義資料では << を説明していない?

compact の実装 (3)

```
def compact(ary)
  ret = []
  ary.each {|v|
    ret += [v] if v != nil
  }
  ret
end
```

- each で繰り返す
- ret に += [v] で加えていく (お薦めできない)

注意(1) 返り値と表示

- 値を返すことと表示することは違う
- 100 を返す関数
```
def ret_hundred
  100
end
```
- 100 を表示して nil を返す関数
```
def prt_hundred
  p 100
end
```

注意(2) 破壊と非破壊

- 一般に引数を破壊することは避ける
- compact の場合は、Array#compact も元の配列を破壊しない

- 例

```
a = [1,nil,2]
p a.compact          #=> [1,2]
p a                  #=> [1,nil,2]
```

引数を破壊してしまう compact

```
def compact(ary)
  ary.delete(nil)
  ary
end
```

- Array#delete はレシーバを破壊的に変更する
- Array#delete_if も同様

その他の注意

- compact の中に
  print "processing...¥n"
  などと経過を表示するコードを入れた人がいた。
  そういう表示は不要。

トラブルについて

- トラブルが起きた場合、こちらで状況がわかれば助言できる
  - 状況が不明でいかんともしがたい記述の例
    「うまく動きませんでした」
  - 状況を再現できる良い記述の例:
    「ruby -e 'xx' としたら undefined local variable or method 'xx' for main:Object (NameError) というエラーが出ました」
- 参考:「技術系メーリングリストで質問するときのパターン・ランゲージ」
  http://www.hyuki.com/writing/techask.html

アッカーマン関数 ack

- 有名な再帰関数
- 帰納的定義
  - ack(m,n) = n + 1　　if m = 0
  - ack(m,n) = ack(m-1,1)　　if n = 0
  - ack(m,n) = ack(m-1, ack(m,n-1))　　otherwise
- 関数の値の例
  - ack(0,0)　　#=> 1
  - ack(1,1)　　#=> 3
  - ack(1,2)　　#=> 4
  - ack(2,1)　　#=> 5
  - ack(2,2)　　#=> 7
  - ack(3,4)　　#=> 125

ack の実装

```
def ack(m,n)
  if m == 0
    n + 1
  elsif n == 0
    ack(m-1, 1)
  else
    ack(m-1, ack(m, n-1))
  end
end
```

- 帰納的定義に対応するコードを書く
- return を使っても良い

ざっと眺めた結果

- 返り値と表示の説明が足りなかった?
- 破壊的な性質まで気が回らないひとがいた
  (ちゃんと気がついた人もいた)
- アッカーマンはあまりに自明すぎた?