

R でプログラミング

-R 入門書に書いていないプログラミングの常識-

名古屋大学大学院 生命農学研究科
森林生態生理学研究分野

安部 晃生

プログラミング

- コンピュータに対して処理を委譲
 - 反復処理
 - 演算速度
- プログラミング言語を用いてプログラム (指示書) を作成
 - 誰がプログラムを読み書きするのか？
 - ✓ 書く: 人間
 - ✓ 読む: コンピューター, 人間



人間の人間による人間のた めのコーディング

- なぜプログラムは難しく見えるのか？
 - 難しく見えるように書いている
- 難しくないように見えるプログラムとは？
 - 読みやすい (可読性)
 - 見やすい (視認性)
- 人間が読むことを意識したプログラムを書くことが大事
 - プロトコールは自分だけがわかれば良いのか？



可読性 (intelligibility)

- ソースコードを読むことにより, プログラムの目的・処理内容が容易に理解できること
 - 自然な処理の流れ
 - 適切な命名 (変数名, 関数名など)
 - コメント
- 可読性の低いソースコードの問題
 - 理解するための時間コスト
 - バグ発見の遅れ



可読性

処理の流れの例

```
three()  
five()  
two()  
one()  
four()
```

```
one()  
two()  
three()  
four()  
five()
```

わかりやすいのはどっち？

ただし，処理の順番に依存性がある場合は注意



可読性

命名の例

```
function (fp, rp, c)
{
  tm_f <- tm(fp, c)
  tm_r <- tm(rp, c)
  return abs(tm_f - tm_r)
}
```

わかりやすいのは
どっち？

```
function (primer.forward, primer.reverse, condition)
{
  tm.forward <- calculate.tm(primer.forward, condition)
  tm.reverse <- calculate.tm(primer.reverse, condition)
  return abs(tm.forward - tm.reverse)
}
```



可読性

コメントの例

```
cleanup <- function(file) {  
  ...  
}
```

わかりやすいのは
どっち？

```
# 一時データを保存したファイルをクリーンアップします。  
# クリーンアップされたファイルはサイズが 0 になりますが  
# 削除されずに残ります。  
# ファイルを削除するには base::unlink 関数を使用します。  
cleanup <- function(file) {  
  ...  
}
```



視認性 (visibility)

- ソースコードが文字・単語・ブロック単位ではっきりと読めること
 - インデント
 - 空行 (段落化)
 - 等幅フォント
- 視認性の低いソースコードの問題
 - バグ発見の遅れ
 - 眼精疲労



視認性 例

```
quicksort <- function(vector) {  
  if(length(vector) < 2) {  
    return(vector)  
  }  
  if(length(vector) == 2) {  
    return(  
      ifelse(vector[1] <= vector[2], vector, vector[2:1])  
    )  
  }  
  
  pivot <- sample(vector, 1)  
  parity <- vector[vector == pivot]  
  less <- quicksort(vector[vector < pivot])  
  greater <- quicksort(vector[vector > pivot])  
  return(c(less, parity, greater))  
}
```



視認性 例

```
quicksort<-function(vector){  
  if(length(vector)<2)return(vector)  
  if(length(vector)==2)return(ifelse(vector[1]<=vector[2],vector,vector[2:1]))  
  pivot<-sample(vector,1);parity<-vector[vector==pivot];  
  less<-quicksort(vector[vector<pivot]);greater<-quicksort(vector[vector>pivot]);  
  return(c(less,parity,greater))  
}
```

わかりやすいのはどっち？

エディタ

- エディタの機能
 - タブ切り替え (もしくはマルチバッファ)
 - (複数ファイルの) 正規表現検索・置換
 - オートインデント
 - シンタックスハイライト
 - コードフォールディング
 - アウトライン表示
 - 単語補完
 - 実行環境 (外部アプリケーションとの連携)



R で使われるエディタ

- Windows のメモ帳はメモのためのアプリケーションであって文書を書くためのアプリケーションではない！
- 目的にあったエディタを用いる
 - RjpWiki の「[エディタでR](#)」を参照
- 自分の好みのエディタを見つける
 - [投票所](#)によると Emacs や秀丸あたりが人気らしい
 - ちなみに発表者は Eclipse が好きです



Eclipse + StatET

The screenshot shows the Eclipse IDE with the StatET plugin. The main editor displays the R script `plot.maze.R`, which defines a function `plot.maze` for generating a maze. The function uses a recursive algorithm to create a maze of a given size. The R Graphics window shows the output of the function, a square maze with a start point 'S' at the top and a goal point 'G' at the bottom. The console window shows the execution of the function with the command `plot.maze(100)`.

```
1 #' 迷路を生成します。
2 #'
3 #' @param row 迷路的行数
4 #' @param column 迷路的列数
5 plot.maze <- function(row, column)
6 {
7   put_bar <- function(x, y)
8   {
9     LEFT <- c(-1, 0)
10    RIGHT <- c(1, 0)
11    TOP <- c(0, -1)
12    BOTTOM <- c(0, 1)
13
14    random <- runif(1)
15    bar_direction <- 1
16    if (random <= 1/4)
17    {
18      bar_direction <- LEFT
19    }
20    else if (random <= 1/2)
21    {
22      bar_direction <- RIGHT
23    }
24    else if (random <= 3/4)
25    {
26      bar_direction <- TOP
27    }
28    else
29    {
30      bar_direction <- BOTTOM
31    }
32
33    bar_destination <- c(0, 0)
34    segments(x / column, y / row, bar_destination, bar_direction)
35  }
36
37  plot.new()
38  segments(0, 0, 0, row)
39  segments(0, 0, column, 0)
40  segments(0, 0, column, row)
41  segments(0, row, column, row)
42  segments(0, 0, column, row)
43  segments(0, 0, column, row)
44  segments(0, 0, column, row)
45  segments(0, 0, column, row)
46  segments(0, 0, column, row)
47  segments(0, 0, column, row)
48  segments(0, 0, column, row)
49  segments(0, 0, column, row)
50  segments(0, 0, column, row)
51  segments(0, 0, column, row)
52  segments(0, 0, column, row)
53  segments(0, 0, column, row)
54  segments(0, 0, column, row)
55  segments(0, 0, column, row)
56  segments(0, 0, column, row)
57  segments(0, 0, column, row)
58  segments(0, 0, column, row)
59  segments(0, 0, column, row)
60  segments(0, 0, column, row)
61  segments(0, 0, column, row)
62  segments(0, 0, column, row)
63  segments(0, 0, column, row)
64  segments(0, 0, column, row)
65  segments(0, 0, column, row)
66  segments(0, 0, column, row)
67  segments(0, 0, column, row)
68  segments(0, 0, column, row)
69  segments(0, 0, column, row)
70  segments(0, 0, column, row)
71  segments(0, 0, column, row)
72  segments(0, 0, column, row)
73  segments(0, 0, column, row)
74  segments(0, 0, column, row)
75  segments(0, 0, column, row)
76  segments(0, 0, column, row)
77  segments(0, 0, column, row)
78  segments(0, 0, column, row)
79  segments(0, 0, column, row)
80  segments(0, 0, column, row)
81  segments(0, 0, column, row)
82  segments(0, 0, column, row)
83  segments(0, 0, column, row)
84  segments(0, 0, column, row)
85  segments(0, 0, column, row)
86  segments(0, 0, column, row)
87  segments(0, 0, column, row)
88  segments(0, 0, column, row)
89  segments(0, 0, column, row)
90  segments(0, 0, column, row)
91  segments(0, 0, column, row)
92  segments(0, 0, column, row)
93  segments(0, 0, column, row)
94  segments(0, 0, column, row)
95  segments(0, 0, column, row)
96  segments(0, 0, column, row)
97  segments(0, 0, column, row)
98  segments(0, 0, column, row)
99  segments(0, 0, column, row)
100 segments(0, 0, column, row)
101 }
```



Emacs

```
Emacs@y22a.local
r@y22a.local:~/Rproj/emacs_govt_data$
1 # このスクリプトでは樹木の直径成長をモデリングするためのインテネータを作成する。
2 # 仮定は
3 # 1) 樹木の年平均直径成長が樹木の直径の1/3程に比例する。
4 # 2) しかし、周辺の樹木密度が高い場所では成長が抑制される。
5 # 3) 年平均直径成長の真の値は正規分布に對する。
6 # 4) 年平均直径成長の観測値は真の値に平均の正誤差量が入ったものである。
7
8 ### 樹木を含む観測データの作成
9 library(spatstat)
10 set.seed(40)
11 plot.size <- 100 # プロットの1辺の長さ (100 x 100 mの1 haプロットとする)
12 coord.data <- rpoisapp(lambda = 1000/(plot.size^2),
13                       site = cwin(c(0, plot.size), c(0, plot.size)))
14 # ポアソン点過程データを1000個くらい作成
15 n <- coord.data$n # 全樹木数
16 x <- coord.data$x; y <- coord.data$y # xy座標
17
18 ### 各樹木のD値を設定
19 D.mean <- 15 # 直径樹木のD値の平均
20 D.var <- 25 # 直径樹木のD値の分散
21 D <- rgamma(n, shape = (D.mean^2)/D.var, rate = D.mean/D.var) # 直径樹木のD値
22 D[D < 5] <- 5 # 5 cm以下の樹木は5 cmとしおく
23
24 ### 各樹木の周辺樹木密度の計算 (プロット端は真正反転させて計算しておく、add=adjErrorの適用)
25 nd <- data.frame(x = c(x, -x, -x, -x, x, x, 200 - x, 200 - x, 200 - x),
26               y = c(y, 200 - y, y, y, 200 - y, y, 200 - y, y, y))
27 # 中心、左上、左、左下、上、下、右上、右、右下
28 r <- 5 # 周辺樹木密度を計算するときの半径
29 # そのまま計算すると9割の世界になるので必要なデータ以外を切り落としておく
30 nd <- nd[(r + ndfx & ndfx < 200 - r &
31         r + ndfy & ndfy < 200 - r, )
32 data <- sapply(1:nrow(nd), function(i) { # 半径r m以内の周辺樹木密度
33   sum(sqrt((ndfx[i] - ndfx[j])^2 +
34           (ndfy[i] - ndfy[j])^2) <= r) - 1}) # 最後に自身の数を抜く
35 if(0) { #以下3行は削除、一部コメントアウト開始
36   plot(nd)
37   points(nd[1:n, ], col = "red", pch = 16, cex = 0.5)
38   abline(h = c(0, 100), v = c(0, 100))
39   test.id <- 5
40   points(nd[test.id, ], col = "green")
41   points(nd[sqrt((ndfx - ndfx[test.id])^2 +
42               (ndfy - ndfy[test.id])^2) <= r, ],
43          col = "blue", pch = 16, cex = 0.5)
44 }
```

(玉木さん提供)



参考すると良いウェブ文献

- **コーディング技法** (<http://msdn.microsoft.com/ja-jp/library/aa291593.aspx>)
- **頑健なJavaプログラムの書き方**
(<http://www.alles.or.jp/~torutk/ojava/codingStandard/writingrobustjavacode.html>)
- **命名規則** ([http://ja.wikipedia.org/wiki/命名規則_\(プログラミング\)](http://ja.wikipedia.org/wiki/命名規則_(プログラミング)))
- **字下げスタイル** (<http://ja.wikipedia.org/wiki/字下げスタイル>)
- **プログラミング作法** (<http://ja.wikipedia.org/wiki/プログラミング作法>)



今後 R を習得する上で役に立つ項目

- 一貫性
- リファクタリング
- 単体テスト
 - R では RUnit というパッケージが存在
- バージョン管理システム
- バグ管理システム
 - バグ, デバッグ
- その他