リリース5: カード入力機能の実装

市東 亘

平成 30 年 1 月 9 日

1　概観

目 次

1　概観 **1**

2　カード入力機能の外部仕様 **1**

3　スタック項目の追加 **2**

4　新規カード **2**

5　リファクタリング **3**

6　カード編集機能の実装 **4**

6.1　`card.chooser()` ヘルパー関数の実装 4
6.2　`print.card.detail()` ヘルパー関数の実装 7
6.3　`edit.card()` ヘルパー関数の実装 7
6.4　`browse.card.details()` ヘルパー関数の実装 8
6.5　`edit.card.menu()` 関数の実装 9

2　カード入力機能の外部仕様

フラッシュ・カードのデータ入力機能としては，以下の 2 つが必要．

- 新規カード …… 表・裏面の記入．カテゴリの設定．
- カードの編集 …… 表・裏面の編集．カテゴリの変更．カードの削除．
  - 20 枚ごとにサマリ表示．「次ページ」「前ページ」でサマリページ移動．
  - サマリページで番号を選ぶとカードの編集画面に移動．n で次ページ，p で前ページに移動．
  - カードの削除 …… サマリ表示から削除するカードを番号で選択．削除前に確認メッセージを表示する．

これら 2 つのメニュー項目をメインメニューに追加する．

3　スタック項目の追加

リリース 1 で実装したカード操作関数にスタック項目を追加する．スタックは 1 から始まる整数で区別し，新規作成したカードのスタックは 1 に設定する．

```
# カード操作関数
add.card <- function(card.db, front, back, category.id) {
    timestamp <- as.numeric(Sys.time())
    id <- generateID(timestamp)
    list(db=
         rbind(card.db,
               list(ID=id, 表=front, 裏=back, カテゴリ=category.id,
                    作成日=timestamp, 更新日=0, 出題回数=0, 最終出題日=0,
                    スタック=1),   # ここを追加
               stringsAsFactors = FALSE),
         id=id)
}
```

正解した際にスタックを移動させるヘルパー関数 `move.to.next.stack()` を新たに作成する．指定されたカード ID のカードのスタック番号を 1 インクリメントすることによってスタックの移動を表現する．この関数は変更後のカードデータベース返すので純粋関数である．スタックの最大数は `MAX.STACK` グローバル変数に保持しておく．以下の例ではスタック数を 4 に設定している．

```
# スタックの移動
MAX.STACK <- 4
move.to.next.stack <- function (card.db, card.id) {
    index <- card.db$ID == card.id
    stack <- card.db[index, "スタック"]
    if ( stack < MAX.STACK)
        card.db[index, "スタック"] <- stack + 1
    card.db
}
```

4　新規カード

カードを新規作成する際にカテゴリを設定する必要がある．カテゴリを階層表示でメニューを作成する `category.choice.menu()` 関数をリリース 4，その 2 で以下のように作成した．

```
####
## 選択したカテゴリ ID を返す．-1 はキャンセルの選択を表す．
## show.no.parent=TRUE なら 0 は「親カテゴリなし」
category.choice.menu <- function(category.list, title=NULL, show.no.parent=TRUE) {
    if (show.no.parent)
        choice <- menu(c("【親カテゴリなし】", category.list$names),
                       title=title) - 1
    else
        choice <- menu(category.list$names,
                       title="\n カテゴリを番号で選択してください（0 でキャンセル）．")
    if (choice > 0)
        return(category.list$data[choice, "ID"])
    else if (show.no.parent)
        return(choice)   # 0 or -1
    else
        return(-1)
}
```

この時は，`show.no.parent` 引数で「親カテゴリなし」の表示の有無だけを選択できるようにした．一方，新規カード追加時には，カテゴリ一覧に加え「カテゴリの新規追加」も選択肢に含めた

い．そこで，category.choice.menu() 関数を以下のように変更する．変更のポイントは，これまでのコードに影響を与えないように show.no.parent 引数はいじらない点と，title 引数に値が設定されていない場合はデフォルトのタイトルを設定するように変更する．

```
####
## 選択したカテゴリ ID を返す．-1 はキャンセルの選択を表す．
## show.no.parent=TRUE なら 0 は「親カテゴリなし」
## show.add=TRUE なら 0 は「カテゴリの新規追加」を表す．
category.choice.menu <- function(category.list, title=NULL, show.no.parent=TRUE,
                                 show.add=FALSE) {
    if (is.null)
        title="\n カテゴリを番号で選択してください（0 でキャンセル）．"
    if (show.add) {
        choice <- menu(c("【カテゴリの新規追加】", category.list$names),
                       title=title) - 1
    } else if (show.no.parent)
        choice <- menu(c("【親カテゴリなし】", category.list$names),
                       title=title) - 1
    else
        choice <- menu(category.list$names, title)

    if (choice > 0)
        return(category.list$data[choice, "ID"])
    else if (show.no.parent || show.add)
        return(choice)   # 0 or -1
    else
        return(-1)
}
```

リリース 4，その 1 で「新規カード」メニュー項目を一時的に以下のように定義した．

```
# 新規カード（メインメニュー項目）
add.card.menu <- list(menu.title="新規カード",
                      fn=function() {
                          cat ("\n\n 処理: 新規カード作成中.....\n\n")
                      })
```

これを編集してさらに実際のカード編集機能を追加する．

```
## 新規カード（メインメニュー項目）
add.card.menu <- list(
    menu.title="新規カード",
    fn=function() {
        front <- readline(prompt="\n 表面を記入してください: ")
        back <- readline(prompt="裏面を記入してください: ")
        category.list <- make.category.list(CAT.DB)
        cat.id <-  category.choice.menu(category.list, show.add=TRUE)
        if (cat.id == 0)
            cat.id <- add.category.menu$fn()
        else if (cat.id < 0)
            return
        assign("CARD.DB", add.card(CARD.DB, front, back, cat.id)$db,
               envir=globalenv())
        SAVE.CARD()
    })
```

5　リファクタリング

DIY 原理を覚えているだろうか？assign() 関数を使う処理は今後も使いそうだ．

SAVE.CARD() をリファクタリングする．引数に更新したいデータベースを取り，Global Environment の CARD.DB に保存するように変更する．

```
# アプリ専用カード保存関数
SAVE.CARD <- function(db=NULL) {
    if (!is.null(db))
        assign("CARD.DB", db, envir=globalenv())
    save.card.db(CARD.DB, card.db.filename)  # 変更をファイルに保存
}
```

前節で作成した add.card.menu() の assign() を用いた箇所が SAVE.CARD() を用いて以下のように書き換わる．

```
## 新規カード（メインメニュー項目）
add.card.menu <- list(
    menu.title="新規カード",
    fn=function() {
        front <- readline(prompt="\n 表面を記入してください: ")
        back <- readline(prompt="裏面を記入してください: ")
        category.list <- make.category.list(CAT.DB)
        cat.id <-  category.choice.menu(category.list, show.add=TRUE)
        if (cat.id == 0)
            cat.id <- add.category.menu$fn()
        else if (cat.id < 0)
            return
        # assign("CARD.DB", add.card(CARD.DB, front, back, cat.id)$db,
        #        envir=globalenv())
        # SAVE.CARD()
        SAVE.CARD(add.card(CARD.DB, front, back, cat.id)$db)
    })
```

他にも assign() 関数を用いて CARD.DB を更新する箇所を各自で書き換えてみよう．また，SAVE.CATEGORY() も同じようにリファクタリングしてみよう．

6　カード編集機能の実装

補助関数（ヘルパー関数）として以下の関数を実装する．これらを組み合わせて最終的にカード編集機能を実装する．

- card.chooser() 関数 …… カードをサマリ表示し，カードを選択するためのプロンプトを表示する．関数はユーザが選択したカードのインデックス番号を返す．
- print.card.detail() 関数 …… カード 1 枚の詳細情報を表示する．
- edit.card() 関数 …… カードの内容を更新するユーザインタフェースを提供する．
- browse.card.details() 関数 …… カードの詳細情報を表示したまま，カードをブラウジングするユーザインタフェースを提供する．

6.1　card.chooser() ヘルパー関数の実装

card.chooser() 関数の主な機能は，

- カードのサマリ表示，
- サマリ表示のページ移動管理，
- カードの選択，

の 3 点．関数は選択されたカードの CARD.DB 内のインデックス番号を返す．カードが選択されなかった場合は 0 を返す．

```
## サマリ表示 1 ページに表示するカード数
ITEMS.PER.PAGE <- 20

## カードのサマリ表示
## ページ番号を指定するとカードのサマリ表示メニューが表示し，
## 選択されたカードの CARD.DB 上のインデックスを返す．
## 何も選ばれなかった場合は 0 を返す．
card.chooser <- function (page) {
    len <- nrow (CARD.DB)
    if (len == 0) {
        cat("\n 表示するカードがありません\n")
        return(0)
    }
    cat <- CAT.DB$カテゴリ名  # カテゴリの名前付きベクトルを作成
    names(cat) <- as.character(CAT.DB$ID)
    repeat {
        from <- ITEMS.PER.PAGE*(page-1)+1
        to <- min(ITEMS.PER.PAGE*page, len)
        maxpage <- ceiling(len / ITEMS.PER.PAGE)
        topage <- to%%ITEMS.PER.PAGE # ページ内の実際のカード数
        if (topage == 0)
            topage <- ITEMS.PER.PAGE
        cards <- CARD.DB[from:to, c("表", "裏", "カテゴリ")]
        cards$表 <- str_trunc(cards$表, 12, "right")
        cards$裏 <- str_trunc(cards$裏, 26, "right")
        cards$カテゴリ <- str_trunc(cat[as.character(cards$カテゴリ)],
                                    12, "right")
        cat("\n-----------------------------------------------\n")
        print.data.frame(cards, right=FALSE)
        cat("\n-----------------------------------------------\n")
        if (page == 1 && page == maxpage) # 1 ページ目かつ最終ページ
            prompt <- "[カード番号/キャンセル (c)] > "
        else if (page == 1) # 1 ページ目
            prompt <- "[番号/次ページ (n)/キャンセル (c)] > "
        else if (page == maxpage) # 最終ページ
            prompt <- "[番号/前ページ (p)/キャンセル (c)] > "
        else
            prompt <- "[番号/前ページ (p)/次ページ (n)/キャンセル (c)] > "
        repeat {
            choice <- readline(prompt)
            if (grepl("^[[:digit:]]{1,2}$", choice)) {
                num <- as.integer(choice)
                if (0 < num && num <= topage)
                    return (from + num - 1) # CARD.DB 内のインデックスを返す
            }
            if (choice == "p" && 1 < page) {
                page <- page - 1
                break
            }
            if (choice == "n" && page < maxpage) {
                page <- page + 1
                break
            }
            if (choice == "c") {
                return (0)
            }
        }
    }
}
```

関数内の最初の 5 行は CARD.DB データフレームの行数を len に設定している．もし len が 0 なら登録されたカードが 1 枚もないので 0 を返して関数から抜ける．

次に「CAT.DB$カテゴリ名」ベクトルを cat に格納し，cat の各要素にカテゴリ ID を文字列に変換した名前をつける．こうすることで，cat["<カテゴリ ID>"] でカテゴリ ID からカテゴリ名を取得できる．CARD.DB 内のカテゴリ列にはカテゴリ ID が格納されている．サマリ表示ではカテゴリ ID をカテゴリ名に置き換えて表示したいが，カテゴリ ID からカテゴリ名の変換で，カード 1 枚毎にカテゴリデータフレームから合致するインデックスの TRUE/FALSE ベクトルを作成してカテゴリ名を抽出するのは非効率である．そこで，名前付きベクトルを生成し，名前を使ってカテゴリを抽出するのだ．

repeat ブロック内でカードのサマリ一覧と入力プロンプトを表示する．ユーザからの入力がキャンセルを表す c でない限り，サマリ表示がリピートされる．

repeat ブロックの最初は指定されたページに表示されるカードのインデックスが何番目から何番目かを計算している．最終ページの場合，ITEMS.PER.PAGE 枚のカードがあるとは限らないため min() 関数で小さい方の値を to に設定している．maxpage と topage にはそれぞれ，サマリ表示の最大ページ数と，現在のページに表示されるカード数が設定される．

次に，各ページに表示するカードを CARD.DB から抽出し，cards データフレームを構築する．サマリ表示内容は cards データフレームをプリントすることで実現する．サマリ表示で使うのは「表」「裏」「カテゴリ」列の内容だけである．また，1 行に表示させるために str_trunc() 関数を使い，指定された文字数を超える場合には末尾に「...」を付けて文字列を省略表示する．

プロンプトでデータフレームを評価すると綺麗に整形されて表示されるが，プログラムないから明示的に表示させる場合は print.data.frame() 関数を使う．データフレームを画面に表示すると通常は右揃えになるが，ここでは左揃えで表示させないので，right 引数を FALSE に設定する．

次の if-else の連続では，表示ページ数に応じてユーザに表示するメニュー・プロンプトを設定している．ユーザからの入力はこれまで menu() 関数を使って取得していたが，menu() 関数は番号でしか選択できない．ここでは選択項目をアルファベットでも選択したかったため，画面にプロンプトを表示してユーザからの入力を読み込み文字列として返す readline(prompt="") 関数を使うことにする．

プロンプトで指定された以外の入力があった場合にプロンプトを繰り返し表示するために readline(prompt) を repeat ブロックで囲む．

最初の if(grepl("^[[:digit:]]{1,2}$", choice)) では，ユーザの入力が 1 から 2 桁の数字かどうかをテストしている．grepl(pattern, str) 関数は「正規表現」という記述方法で指定されたパターン（pattern）に文字列（str）が合致した場合は TRUE を返す．以下が 1 から 2 桁の数字にマッチするかどうか調べる実行例である．

```
grepl("^[[:digit:]]{2}$", "1a2")

## [1] FALSE

grepl("^[[:digit:]]{2}$", "12")

## [1] TRUE

grepl("^[[:digit:]]{2}$", "123")

## [1] FALSE
```

次の 2 つの if 節は前ページ（p），次ページ（n）が選択された時の処理で，page の値を更新し break で内側の repeat ブロックから抜け外側の repeat ブロックに戻る．ここで再びサマリ表示する．

最後の if 節はキャンセル（c）が選択された場合の処理で，return(0) によって card.chooser() 関数から 0 を返して終了する．

6.2 print.card.detail() ヘルパー関数の実装

次はカードデータの詳細を画面表示する print.card.detail() 関数を実装する．カードの詳細画面では各フィールドを縦に表示させたいので，データフレームの行データを転置させたデータフレーム，df，を作成する．あとはカテゴリ ID をカテゴリ名に置き換え，1970 年 1 月 1 日からの経過秒数で表されるタイムスタンプを文字列の日付に変更する．

```
## CARD.DB 内のインデックス番号を取り，カードデータの詳細を画面表示する．
print.card.detail <- function (index) {
    row <- CARD.DB[index,]
    df <- t(row)
    df["カテゴリ", 1] <- as.character(
        subset(CAT.DB, ID == row$カテゴリ, select=カテゴリ名))
    get.data.string <- function (colname) {
        time <- row[1, colname]
        if (time == 0)
            return ("")
        as.character(as.POSIXlt(time, origin="1970-01-01"))
    }
    df["作成日", 1] <- get.data.string("作成日")
    df["更新日", 1] <- get.data.string("更新日")
    df["最終出題日", 1] <- get.data.string("最終出題日")
    print(df)
}
```

6.3 edit.card() ヘルパー関数の実装

ここまでのコードを理解していれば edit.card() 関数の実装は難しくないはずだ．カテゴリの選択は以前作成した make.category.list() 関数と category.choice.menu() 関数を再利用する．

```
## CARD.DB 内のインデックス番目のカードを更新する．
edit.card <- function (index) {
    repeat {
        front <- readline(prompt="\n 表面を記入してください: ")
        back <- readline(prompt="裏面を記入してください: ")
        category.list <- make.category.list(CAT.DB)
        cat.id <-  category.choice.menu(category.list, show.add=TRUE,
                                        title="カテゴリを番号で選択してください（0 で変更なし）．")
        if (cat.id == 0)
            cat.id <- add.category.menu$fn()
        ## 最終確認
        repeat {
            choice <- readline(
                prompt="変更を保存しますか？[はい (y)/いいえ (n)/キャンセル (c)] > ")
            if (choice == "n")
                break
            if (choice == "c")
                return (NULL)
            if (choice == "y") {
                CARD.DB[index, "表"] <- front
                CARD.DB[index, "裏"] <- back
                if (cat.id > 0)
                    CARD.DB[index, "カテゴリ"] <- cat.id
                CARD.DB[index, "更新日"] <- as.numeric(Sys.time())
                SAVE.CARD(CARD.DB)
```

```
                return (NULL)
            }
        }
    }
}
```

6.4 browse.card.details() ヘルパー関数の実装

第??節で実装した card.chooser() は，カードをサマリ表示しながらページ移動する機能を提供した．一方 browse.card.details() は，カード詳細表示を 1 ページとして，ページ移動しながらカードをブラウズする機能を提供する．カードの詳細表示では先ほど作成した print.card.details() 関数を使用する．

詳細表示されたカードの下にユーザ入力用プロンプトを表示する．プロンプトで d が入力された場合にはカードを削除する．カードを削除する処理を入力処理の if 節内に記述するとごちゃごちゃしてしまうため，browse.card.details() 関数の先頭に delete.card.prompt() ローカル関数を定義し削除処理をまとめている．

```
## CARD.DB 内のインデックス番号を取り，カードデータの詳細画面を
## ブラウズする．ブラウズ中にカード編集画面にも遷移する．
## 返り値は現在のカードに対応するサマリ表示ページ番号．
## 存在しないカードが指定された場合，ページ番号 0 を返す．
browse.card.details <- function (index) {
    len <- nrow(CARD.DB)
    ## カード削除の確認と削除を担うローカル関数を定義
    ## Closure により外側の index にアクセスできる．ただし，copy-on-modify
    ## ルールにより書き込みはできない．
    ## 0 を返した場合削除なし，1 を返した場合削除成功．
    delete.card.prompt <- function () {
        repeat {
            choice <- readline(prompt="このカードを本当に削除してよろしいですか？[y/n] > ")
            if (choice == "n")
                return (0)
            if (choice == "y") {
                SAVE.CARD(CARD.DB[-index,])
                return (1)
            }
        }
    }
    repeat {
        if (len == 0 || index > len) {# 削除後にカードがなくなった場合 len==0
            cat("\n 表示するカードがありません\n")
            return (0)
        }
        cat("---------------------------------\n")
        print.card.detail(index)
        cat("---------------------------------\n")
        if (index == 1 && index == len) # 1 番目かつ最終カード
            prompt <- "[編集 (e)/削除 (d)/一覧へ戻る (c)] > "
        else if (index == 1) # 1 番目
            prompt <- "[編集 (e)/削除 (d)/次のカード (n)/一覧へ戻る (c)] > "
        else if (index == len) # 最終カード
            prompt <- "[編集 (e)/削除 (d)/前のカード (p)/一覧へ戻る (c)] > "
        else
            prompt <- "[編集 (e)/削除 (d)/前のカード (p)/次のカード (n)/一覧へ戻る (c)] > "
        repeat {
            choice <- readline(prompt)
            if (choice == "e") {
                edit.card(index)
                break
```

```
            }
            if (choice == "p" && 1 < index) {
                index <- index - 1
                break
            }
            if (choice == "n" && index < len) {
                index <- index + 1
                break
            }
            if (choice == "d") {
                if (delete.card.prompt() == 1)
                    len <- len - 1
                if (index > len)
                    index <- index - 1
                break
            }
            if (choice == "c") {
                ## index 番号からサマリ表示画面のページ番号を計算
                return (ceiling(index/ITEMS.PER.PAGE))
            }
        }
    }
}
```

6.5 edit.card.menu() 関数の実装

これまでに作成してきたヘルパー関数を使って，メインメニューの項目に「カードの編集・削除」を追加する．

```
## カードの編集・削除（メインメニュー項目）
edit.card.menu <- list(
    menu.title="カードの編集・削除",
    fn=function() {
        page <- 1
        repeat {
            choice <- card.chooser(page)
            if (choice == 0)
                break
            page <- browse.card.details(choice)
            if (page == 0)
                break
        }
    })
```

edit.card.menu() 関数をアプリのメニューに追加する．

```
# アプリ起動関数
run.app <- make.menu.func(list(add.card.menu, category.menus,
                               edit.card.menu, show.category.menu),
                          title="\n メインメニュー （0 で終了）")
```