卒業論文

my_help の階層表示の作成

関西学院大学理工学部

情報科学科 西谷研究室

27019610 佐々木 陸

2023 年 3 月

概 要

西谷研究室では my_help をメモを残す時に主に使用している．本研究ではこの my_help を list 表示する際に，指定した layer に応じて，階層表示できるように改良していくことを目的とした．現在の my_help の list 表示は 1 階層のみである．これによってどこにどのメモを残しているかを見つけるのが大変であることが課題となっている．

my_help を改良していくにあたって，テキストを Hash に格納するためのプログラムである md2hash_new.rb と org2hash_new.rb，階層表示にするためのプログラムである print_contents.rb の開発とそれらに対するテストを作った．また，my_help の list 表示の処理を行う list.rb の書き換えを行い，指定した layer に応じて階層表示できるよう改良した．

階層表示できるよう改良したことによって，わかりやすい構造になり，課題を解決できたと考えている．

目 次

図 目 次

第1章　序論

西谷研究室では my_help というキーボードのみで操作可能な CUI(CLI) ヘルプの Usage 出力を真似て, ユーザー独自の help を作成・提供する Ruby 言語用の外部ライブラリである gem を使用している．my_help はメモを残したいときに使用していて，key(記憶のとっかかり) を提供してくれる．

現在の my_help は list を表示した場合 1 階層のみの表示となっており，どこにどのメモを残したのかを見つけるのが難しい．

そこで，ディレクトリやディレクトリに含まれるファイルを tree コマンドで tree 状に表示させられるように, my_help で list 表示する場合に，階層表示させる．そうすることで，分かりやすく整理されてどこに何があるか見つけやすい構造になるため，より使い勝手が良くなると考えた．

そこで，my_help で list 表示する際に見たい場所を指定して，それに応じて階層表示することを本研究の目的とした．

第2章　my_help

2.1　my_helpの特徴

my_help は独自の help を提供することを目的としている．これを西谷研究室では講義の内容や 1 度参考にしたサイトのリンクなどをメモすることに活用している．

my_help の主な特徴として

- ユーザーが自身の help(メモ) を作成できる
- テンプレートを提供する
- 同じ形式，見た目，操作，階層である
- 読み書きしやすい
- 新規作成，削除，編集ができる

がある [2].

2.2　my_helpの使い方

my_help のコマンドの一覧は以下のコマンドをターミナル上で入力すると出力される．

```
> my_help
Commands:
  my_help delete [HELP]       # delete HELP
  my_help edit [HELP]         # edit help
  my_help git  [pull|push]    # git operations
  my_help hello               # hello
  my_help help [COMMAND]      # Describe available commands or one specific c...
  my_help init                # initialize my_help environment
  my_help list [HELP] [ITEM]  # list helps
```

```
my_help new  [HELP]          # mk new HELP
my_help set  [:key] [VAL]    # set editor or ext
my_help version              # show version
```

第3章　手法

3.1　RSpec

Ruby や Ruby on Rails で作ったクラスやメソッドをテストするためのドメイン特化言語を使ったフレームワークである RSpec を使用する.

3.1.1　テストの進め方

- test を作る
- エラーを出す (red)
- エラーを無くす (green)
- code をきれいにする (refactoring)

このような coding の仕方を TDD(Test Driven Development： テスト駆動開発) と呼ぶ. この手法でプログラムの開発を進めていく.

3.1.2　テストの中身

RSpec の中身について実際のコードを用いて具体的な例を紹介する. ここで示すのは”Effective Testing with RSpec 3”で紹介されている理想的なサンドウィッチについてである [3, p.5].

```
1  sandwich = Struct.new(:taste, :toppings)
2  RSpec.describe "An ideal sandwich" do
3   before { @sandwich = sandwich.new("delicious", []) }
4
5   it "is delicious" do
6     taste = @sandwich.taste
```

```
7     expect(taste).to eq("delicious")
8   end
9  end
```

1 行目はテストするプログラムである．ここでは構造体クラスを定義している．変数名を sandwich として，引数を Symbol(:taste, :toppings) として構造体を生成している．
2 行目以降が RSpec のコードである．RSpec は describe と it という単語を使って，会話形式で概念を表現する．2 行目の describe でテストの対象を示す．ここでは”An ideal sandwich”に関する記述 (describe) であることを明示している．3 行目の before によってテストを行う前 (before) に準備する環境が記述されている．1 行目の sandwich を”delicious”と [] を引数として呼び出した．これは，:toppings を空で生成していることを意味している．5 行目の it で”An ideal sandwich”がどうあるべきか，つまりプログラムの振る舞いを文章で記述することで仕様を示す．ここではそれが”is delicious”であることを確かめる文章となっている．7 行目の expect では実行結果 (actual) と期待値 (expeced) が一致しているかを to eq によって確認している．
このコードの結果は以下のようになり，RSpec はテストの内容とテスト結果が直感的に表示される．

```
riku@rikuShigetos-MacBook-Air ~/m/m/r/d/c/spec (main|MERGING)> rspec sandwich_spec.rb --format documentation

An ideal sandwich
  is delicious
```

図 3.1: 理想的なサンドウィッチの RSpec の実行結果.

3.2 FSM(Finite State Machine)

my_help で list を指定した layer に応じて階層表示するように改良していくにあたって，markdown または org で記述されたテキストを Hash に格納する必要がある．そこで，Org2Hash および Md2Hash という Class を用意して，それぞれのテキストから Hash へ格納する．そこでは，有限状態機械 (Finite State Machine:FSM) を用いて実装する．

FSM とは

有限状態機械は基本的に, イベントに対してどのように振る舞うのかを規定したものでしかありません. これは一連の状態で構成されており, そのいずれかが「現在の状態」となります. そして, それぞれの状態について, その状態 にとって意味を持つ一連のイベントがあります. そして, それぞれのイベントごとに, システムが遷移する次の状態が定義されています [4].

と定義される.

3.2.1 FSM の処理

FSM が実際にどのような処理をしているか Pragmatic Progratmmer で紹介されている FSM の例を用いて説明していく [4, p.175].

例えば, ウェブソケットから複数のパーツに分割されたメッセージ (multipart メッセージ) が到来する可能性があるとします. 最初のメッセージはヘッダーです. その後に, 任意の数のデータメッセージが, 最後にトレーラーが到来します. これを FSM で表現すると以下のようになります.

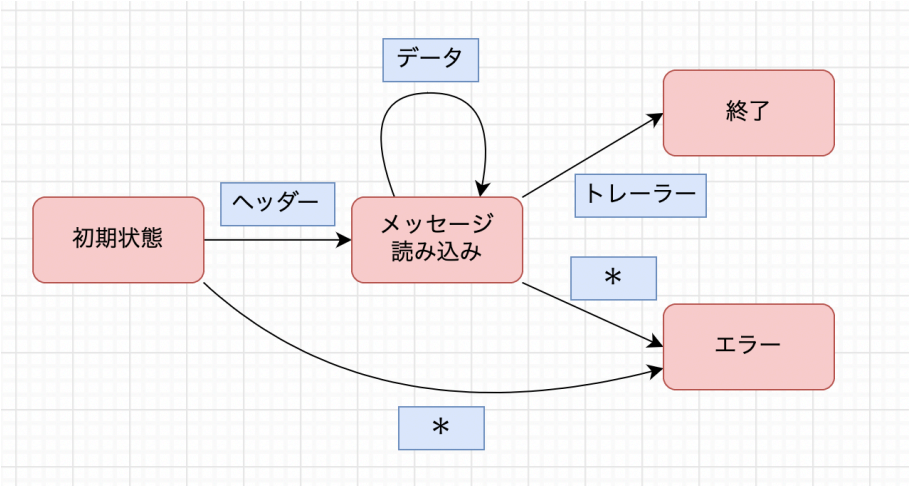

図 3.2: ”達人プログラマー ”で紹介されている FSM 図 ([4, p.175] に倣って再作成).

まず最初の状態は「初期状態」です. ヘッダーメッセージが到来すると, 状態が「メッセージ読み込み」状態に遷移します. 初期状態にある時に何か別のメッセージが到来した場合, アスタリスク (*) の付いた矢印を経由して 「エラー」に状態が遷移し, そこで終了します. 「メッセージ読み込み」状態にあ

る時に受け付けられるのは, データメッセージ (その場合, 状態「メッセージ読み込み」にとどまり続けます) か, トレーラー (その場合, 状態は「完了」に遷移します) であり, それ以外の場合は「エラー」状態に遷移します [4].

この例での FSM の処理は初期状態から始まり，ヘッダーが来た場合に，メッセージ読み込み状態に遷移し，そこで，データが来た場合は自己遷移するという流れは，本研究で実装した fsm_2 メソッドに似ている．

本研究で実装した fsm_2 メソッドには 2 つの主要な状態があるが，そのうち 1 つは内部に子の状態を持つコンポジット状態である．またイベントも異なっている．他にも異なる点があるため，本研究ではこの例の通りではなく，参考にした上で FSM の実装を行う．

第4章　結果

4.1　md2hash.rb

まずは md2hash.rb から実装した．ここでは markdown で記述したテキストを Hash に格納することを目的としている．その挙動を確認するためのテストは md2hash_spec.rb を用意する．

4.1.1　Hash

org2hash.rb，md2hash.rb のコードではテキストを Hash に格納していく．

Hash はデータを管理しているものであり，key(数値，文字，シンボル) によってデータを呼び出せる．Ruby では {} が Hash を意味する．

4.1.2　FSM 図

md2hash.rb で実装した fsm_2 メソッドを drawio を用いて FSM 図として示した．図に沿って解説していく．

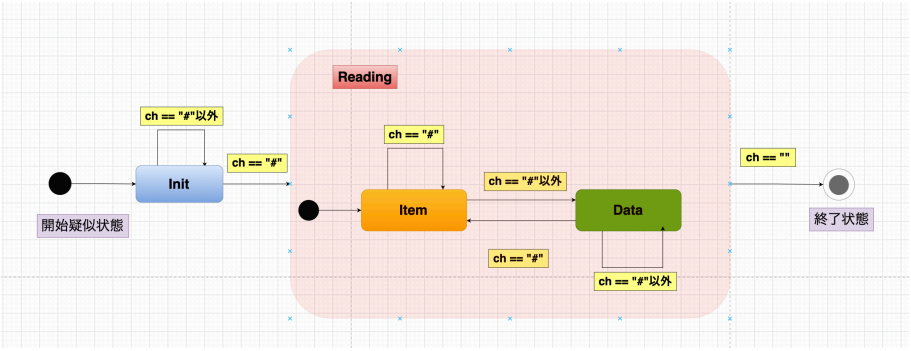

図 4.1: md2hash.rb で実装した fsm_2 メソッドの FSM 図.

1．fsm_2 メソッドでは Init(イベントを待っている)，Reading(読み込んでいる) という 2 つの主要な状態がある．

2．開始疑似状態から Init 状態へと遷移する．

3．Init 状態では default("#"以外) を受けた場合は自己遷移し，"#"を受けた場合に Reading 状態に遷移する．

4．Init 状態からコンポジット状態 (内部に子の状態を持つ状態) である Reading 状態に遷移後は，通常の開始疑似状態を経由して，Item 状態 (item の追加をしている) へと遷移する．

5．Item 状態時に"#"を受けた場合は自己遷移し，default を受けた場合は Data 状態 (data の追加をしている) へと遷移する．

6．Data 状態時に default を受けた場合には自己遷移し，"#"を受けた場合には Item 状態へと遷移する．

7．イベントがある場合はこれらの動作を繰り返し，イベントがなければ，Reading 状態を抜けて終了するという処理になっている．

fsm_2 メソッドではこのような処理をするよう実装していく．

4.1.3 状態遷移

md2hash.rb，org2hash.rb では FSM を用いて実装した．実装した fsm_2 メソッドではイベントによって状態遷移が引き起こされるという処理のループになっている．各遷移の結果として，新たな状態 とアクション名の双方が得られ，このアクション名を使ってコードを実行し，ループの続きを実行する．

まず，FSM を実装するにあたって，以下のコードのように TRANS をあらかじめ用意して，イベントと遷移先，action を明示する．

```
1  # md2hash.rb
2    TRANS = {
3      #current
```

```
4       #            new         action
5       #----------------------------------
6       init: {
7         :default => [:init, :ignore],
8         "#" => [:reading, :item],
9       },
10
11      reading: {
12        "#" => [:reading, :item],
13        :default => [:reading, :data],
14      },
15    }
```

Hash である TRANS を用意し，その中でさらに init 状態と reading 状態の Hash を用意した．

以下の表では md2hash.rb の TRANS で明示したイベントが来た場合の遷移先と action である．

	"#"	default
init 状態	reading 状態	init 状態
	item	ignore
reading 状態	reading 状態	reading 状態
	item	data

これを fsm_2 メソッドで活用していく．

4.1.4 fsm_2 メソッド

my_help には元々org2hash.rb があり，そこでは，org で記述したテキストを Hash に格納するために fsm メソッドを実装した．それと同じように md2hash.rb でも fsm メソッドの実装を行なったが，fsm メソッドでは指定した layer に応じて，階層表示させるのが，難しかったため，fsm メソッドを改良し，fsm_2 メソッドを作った．

以下のコードは md2hash.rb の fsm_2 メソッドの一部である．

```
1  state = :init
2  @md_text.each do |line|
```

```
 3    state, action = TRANS[state][line[0]] || TRANS[state][:default]
 4
 5    results << [line, state, action]
 6    current_item = case action
 7    when :ignore
 8    when :item
 9      m = line.match(/^(#*) (.+)/)
10      head, item = m[1], m[2]
11      head_num = head.chomp.size || 0
12      contents[item] = { head_num: head_num, cont: [] }
13      contents[item]
14    when :data
15      contents[item][:cont] << line
16    end
17  end
```

最初の状態は init 状態である．2 行目で markdown で記述したテキストを 1 行ずつ処理し，3 行目で状態と受け取ったイベントによって，遷移先と action を定める．action が ignore の時，何も処理しない．ここまでは fsm メソッドと同じである．fsm メソッドから fsm_2 メソッドに変更した点は，action が item の場合と data の場合の処理である．

action が item の場合，

```
m = line.match(/^(.*) (.+)/)
head, item = m[1], m[2]
head_num = head.chomp.size || 0
```

によって，見出し ((例)## head1-2) を"##"とそれに続く見出し名 (head1-2) に分けた．"##"は変数 head に，見出し名は変数 item に代入する．そして，head の数 ("##"の場合は 2) を変数 head_num に代入する．

そして

```
contents[item] = { head_num: head_num, cont: [] }
```

によって，Hash である contents に item と head_num と cont を格納する．cont は配列であり，action がデータの場合にデータを入れていくため初期化しておく．そして，action が data の場合は cont にデータを入れていく．

fsm メソッドでは markdown で記述したテキストの item(見出し名) と cont(テキストの内容) を Hash に格納したが，指定した layer に応じて，階層表示するために fsm_2 メソッドでは item と cont だけでなく head_num も Hash に格納した．これで md2hash.rb の実装は以上となる．

4.1.5 md2hash_spec.rb

以下は md2hash.rb に対してテストする md2hash_spec.rb のコードである．

```
 1  # md2hash_spec
 2  RSpec.describe "Md2Hash" do
 3    context "hierarchy hash" do
 4      before {
 5        @md_text = <<HEREDOC
 6  # head1
 7  ## head1-2
 8    - hage
 9  HEREDOC
10        @Md2Hash = Md2Hash.new(@md_text)
11      }
12      it "reads text" do
13        # results = @Md2Hash.results
14        contents = @Md2Hash.contents
15        expected = {
16   "head1" => { :head_num => 1, :cont => [] },
17   "head1-2" => { :head_num => 2, :cont => ["  - hage"] },
18        }
19        expect(contents).to eq(expected)
20      end
21    end
22  end
```

ここでのテストの対象は”Md2Hash”である．3 行目の context はテスト対象のメソッドをどのような条件で実行するかを明示する．ここでは”hierarchy hash”とする．

4 行目の before の中では markdown で記述したテキストを HEREDOC を用いて，インスタンス変数である@md_text に格納した．HEREDOC はソースコードの中に改行を含む

文字列を記述する特別な記法である．そして，@md_text を引数として，md2hash.rb で用意した Md2Hash という Class を呼び出した．

12 行目の it で”read text” と仕様を示す．Class Md2Hash の中で作成した contents メソッドを呼び出す．それが実行結果となる．そして，実行結果 (contents) と期待値 (expected) が一致するかを to eq によって確認する．このテストが通るよう md2hash.rb を実装した．

4.1.6 md2hash.rb の結果

md2hash.rb のコードをテストすると

```
riku@rikuShigetos-MacBook-Air ~/g/r/m/d6_md2hash (main)> rspec spec/md2hash_new_
spec.rb -e reads
Run options: include {:full_description=>/reads/}

md2hash
  hierarchy hash
{"head1"=>{:head_num=>1, :cont=>[]}, "head1-2"=>{:head_num=>2, :cont=>["  - hage
"]}}
    reads text

Finished in 0.00051 seconds (files took 0.04602 seconds to load)
1 example, 0 failures
```

図 4.2: md2hash_spec の実行結果.

と正常に動作することが確認できる．

4.2 Org2Hash.rb

org2hash.rb は org で記述したテキストを Hash に格納することを目的としている．md2hash.rb のコードと共通する部分が非常に多い．しかし，md2hash.rb で実装した fsm_2 メソッドでは "#"がイベントとなるのに対して，org2hash.rb の fsm_2 メソッドでは"*"がイベントとなるため変更していく． 以下のコードが org2hash.rb の変更した部分である．

```
TRANS = {
    #current
    #            new         action
    #---------------------------------
    init: {
      :default => [:init, :ignore],
      "*" => [:reading, :item],
    },
```

```
  reading: {
    "*" => [:reading, :item],
    :default => [:reading, :data],
  },
}
```

Hash である TRANS の中で定義したイベントを"#"から"*"に変更した．markdown では"#"が見出しを意味するが，org では"*"が見出しであるためである．

以下の表では org2hash.rb の TRANS で明示したイベントが来た場合の遷移先と action である．

	"*"	default
init 状態	reading 状態	init 状態
	item	ignore
reading 状態	reading 状態	reading 状態
	item	data

ここではイベントを"#"から"*"へ変更したが，遷移後の状態や action は変わらないため，イベントの部分が主な変更点である．これによって，md2hash.rb と同様に org で記述したテキストを Hash に格納することができる．

4.3 print_contents.rb

print_contents.rb では list 表示する際に指定した layer に応じて，階層表示するように実装した．また，その挙動を確認するためのテストは print_contents_spec.rb を用意する．

4.3.1 print_contents.rb の list メソッド

print_contents.rb の list メソッドでは階層表示するようにプログラムを書いた．

以下のコードが list メソッドである．

```
1    #print_contents.rb
2   def list(layer = 1)
```

```
 3    output = ""
 4    @contents.each do |key, val|
 5      n = @contents[key][:head_num]
 6      key, desc = key.split(":")
 7      desc ||= ""
 8
 9      if n <= layer
10        s1 = ""
11        for i in 1..n - 1
12   if n == 1
13     break
14   end
15   s1 += "     "
16        end
17
18        s2 = format("- %20s : %s\n", key, desc)
19
20        output += s1 + s2
21      end
22    end
23    return output
24  end
```

list メソッドでは layer という引数を受け取り，layer に応じて階層表示させる．4 行目の

```
@contents.each do |key, val|
  n = @contents[key][:head_num]
```

では，@contents は org2hash.rb あるいは md2hash.rb でテキストを Hash に格納したものに対して処理を行い， 変数 n に head_num(org の場合"*"の数，md の場合"#"の数) を代入する．

そして，9 行目の

```
if n <= layer
```

以降で，指定した layer に応じて階層表示するようにプログラムを書いていく．11 行目の for 文の中で，head_num - 1 回 ((例)head_num が 2 の場合 1 回) ループを回し，空白を入れていく．そのため，head_num が 1 の場合は空白を入れない．

そして，空白と見出し名を連結させると，階層に応じて，見出しの前に空白が入り階層表示するようになった．これで print_contents.rb の実装が以上となる．

4.3.2 print_contents_spec.rb

以下のコードは print_contents.rb をテストする print_contents_spec.rb である．

```
1      #print_contents_spec
2      RSpec.describe "Print" do
3    before {
4      @contents = { "head1" => { :head_num => 1, :cont => [] },
5     "head1-2" => { :head_num => 2, :cont => ["  - hage"] },
6     "head2" => { :head_num => 1, :cont => [] } }
7    }
8
9    it "階層表示 " do
10     layer = 2
11     puts actual = Print.new(@contents).list(level)
12     output = <<HEREDOC
13 -                head1 :
14       -              head1-2 :
15 -                head2 :
16 HEREDOC
17     expect(actual).to eq(output)
18   end
19 end
```

ここでのテストの対象は”Print”である．3 行目の before で print_contentss.rb で用意した Print という Class を呼び出す時の引数となる@contents の中身を定義した．
9 行目の it で”階層表示”と仕様を示す．10 行目で変数 layer を指定する．Class Print の中で作成した list メソッドを layer を引数として呼び出す．それが実行結果 (actual) となる．そして，それと期待値 (output) が一致するかを to eq によって確認した．このテストが通るよう print_contents を実装する．

4.3.3 print_contents.rb の結果

print_contents.rb のコードを RSpec でテストすると

```
[riku@rikuShigetos-MacBook-Air ~/m/m/r/d/d6_md2hash (main|MERGING)> rspec spec/pr]
int_contents_spec.rb

Print
-               head1 :
    -             head1-2 :
-               head2 :
  階層表示

Finished in 0.00057 seconds (files took 0.046 seconds to load)
1 example, 0 failures
```

図 4.3: print_contents_spec.rb の実行結果.

と正常に動作することが確認できる.

4.4 my_help に組み込む

実装した org2hash.rb，md2hash.rb と print_contents.rb を my_help に組み込む．my_help には元々org2hash.rb があり，そのまま組み込むと重複してしまうため，新しく組み込んだ org_2hash.rb と md2hash.rb をそれぞれ org2hash_new.rb，md2hash_new.rb とした.

以下のコードは lib/my_help/list.rb のファイルを読み込む処理を行う read_help メソッドの一部である.

```
1  if @ext == ".org"
2      info[:items] = Org2Hash_new.new(File.read(file)).fsm_2
3  end
4  if @ext == ".md"
5      info[:items] = Md2Hash_new.new(File.read(file)).fsm_2
6  end
```

ここで org2hash_new.rb で用意した Class Org2Hash_new もしくは md2hash_new.rb で用意した Class Md2Hash_new を呼び出せるようにした．また，拡張子が”.md”もしくは”.org”である場合に list 表示させるため，指定した拡張子によって，どちらかを呼び出せるようにした.

このような改良を加えたことによって，元々，実装されていたspec/unit/list_spec.rbのテストが通らなくなる．実行結果は以下のようになる．

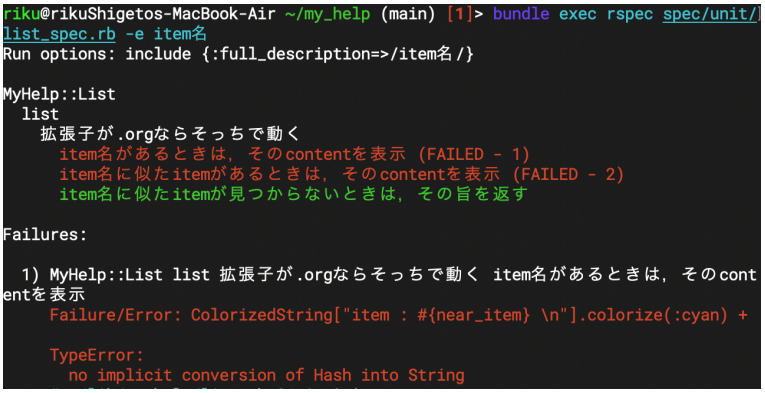

図 4.4: list.rbのエラー．

このエラーを修正していく．以下のコードはlist.rbのfind_nearメソッドの一部である．

```
ColorizedString["item : #{near_item} \n"].colorize(:cyan) +
  @help_info[:items][near_item]
```

このhelp_infoの仕様をlayerに対応させるために変更したところで全てのエラーが出ているため，

```
@help_info[:items][near_item][:cont][0]
```

と明示すると正常に動作します．このように対応箇所を全て変更することで，
list_spec.rbのテストは通るようになる．次は，指定したlayerに従って階層表示するように改良していく．

4.5 layerに従って階層表示

my_helpはlist表示をするにあたって，指定したlayerに対応して階層表示することができず，1階層のみの表示となっていた．

list.rbのindexから呼び出すlist_help_withメソッド，org2hash_new.rbとmd2hash_new.rbのfsm_2メソッド，print_contents.rbのlistメソッドの書き換えを行うことで，指定したlayerに応じて階層表示するように改良していく．

4.5.1 list.rb の list_help_with メソッドの改良

まず，list.rb の list_help_with メソッドの書き換えを行なった．ここで変更したのは，item 名がない時の処理である．以下のコードが変更していく部分である．

```
1  # lib/my_help/list.rb の list_help_with メソッド
2      if item.nil?
3   @help_info[:items].each_pair do |item, _val|
4     item, desc = item.split(":")
5     desc ||= ""
6     output << format("- %20s : %s\n", item, desc)
7   end
```

変更前のコードでは item 名がない場合，全ての見出しを表示するとしている．

そこで，以下のコードに変更した．

```
1  if item.nil?
2   contents = @help_info[:items]
3   #p contents
4   output << Print.new(contents).list(@layer)
```

@help_info[:items] は list.rb のファイルを読み込む read_help メソッドで呼び出した org2hash_new によってテキストを Hash に格納したものである．それを引数として，階層表示させるためのプログラムである print_contents.rb の list メソッドを呼び出した．ここで list メソッドを呼び出したことで，item 名がない場合，layer を指定すると，それに応じて階層表示にできるようになった．

4.5.2 org2Hash_new.rb,md2hash_new.rb の fsm_2 メソッドの改良

次に org2hash_new.rb と md2hash_new.rb の書き換えを行う．変更する部分は共通しており，どちらも fsm_2 メソッドの”action が item の時”の処理で正規表現を指定した部分である．

下の markdown で書かれた sample を md2hash_new.rb で実行した場合，

```
#sample コード
# head1 : a
## head1-2
```

正しい実行結果は本来,

```
1  > ruby lib/my_help/md2hash_new.rb
2  {"head1 : a"=>{:head_num=>1, :cont=>["- hoge "]},
3  "head1-2 "=>{:head_num=>2, :cont=>[]}, }
```

となるが，改良前のコードの実行結果は

```
1  > ruby lib/my_help/md2hash_new.rb
2  {"a"=>{:head_num=>9, :cont=>["- hoge "]},
3  "head1-2 "=>{:head_num=>2, :cont=>[]}, }
```

となり，2 行目の出力に異常が出た．head : a に対して，異なる結果が出た．これは org2hash_new.rb と md2hash_new.rb の fsm_2 メソッドで指定した正規表現に原因があるため変更していく．

以下のコードは，md2hash_new.rb の fsm_2 メソッドで用いた元々の正規表現である．org2hash_new.rb でもこの正規表現を用いている．

```
#  md2hash_new の fsm_2 メソッドの一部
 m = line.match(/^(.*) (.+)/)
```

この正規表現では"^"は行の先頭を示す．前の"(.*)"では，任意の 1 文字が 0 または 1 個以上連続した文字列だけを取り出している．

後ろの"(.+)"は任意の 1 文字が 最低 1 個以上連続した文字列だけを取り出している．これによって,

1. # head1 :
2. a

となる．

本来"#"だけを取り出せるようにしたいため，前の"(.*)"の部分を以下のように変更した．

```
# md2hash_new fsm_2 メソッドの一部
 m = line.match(/^(#+) (.+)/)
```

前を任意の文字ではなく，"#"だけを取り出せるように"(#+)"に変更した．

これによって，

1. #
2. head1 : a

と正しく認識され，正しい動作をした．

org2hash_new.rb の正規表現も変更した．以下のコードが変更後となる．

```
m = line.match(/^(\*+) (.+)/)
```

変更したのは，md2hash_new.rb と同じく前の部分である．org2hash_new.rb では"(*+)"とした． 正規表現では"*"は 0 または 1 個以上連続するという意味で使われている．しかし，"*"のように前に"\"が付くことによって，"*"は正規表現ではなく，単なる一つの文字として認識され，"*"の文字列だけを取り出せた．

これで md2hash_new.rb と org2hash_new.rb が正しい動作をするようになった．

4.5.3 print_contents.rb の list メソッドでの出力を改良

print_contents.rb では出力の形式を改良していく．改良前の print_contents.rb を以下の markdown で記述した sample コードを用いて，layer は 3 と指定し実行した結果は

```
#sample コード
# head1 : a
## head1-2 : b
### head1-3 :
#### head1-4 : c
- hoge
##### head1-5
# head2 :
- hage
- hage
## head2-2 :
### head2-3 : d
```

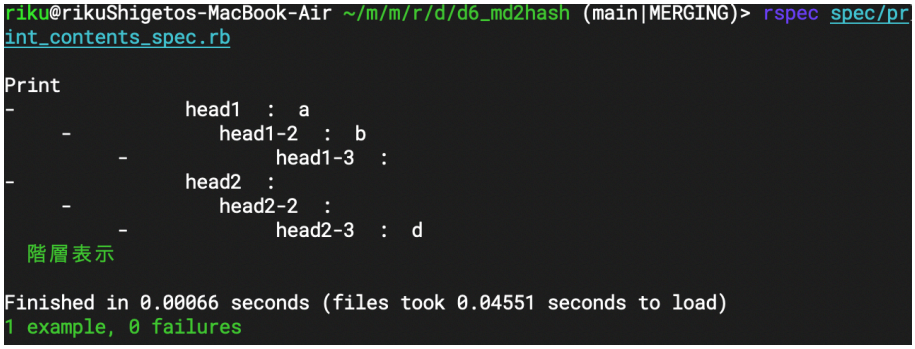

図 4.5: 改良前の rint_contents を 3 階層まで出力.

となる．階層に応じて，見出しの前に空白を入れることで，段差をつけて，階層表示に見えるようにした．しかし，あまり綺麗なデザインではないため改良を加えた．

以下のコードが改良後である．

```
1  #print_contents  改良後
2  if n <= layer
3     s1 = ""
4     for i in 1..n
5       s1 += "*"
6     end
7
8     s2 = ""
9     for j in 1..layer - n
10      s2 += " "
11    end
12
13    s3 = format("%20s : %s\n", key, desc)
14
15    output += s1 + s2 + s3
16  end
```

4 行目の for 文の中で階層数に応じて，見出しの前に"*"を出力する．

階層によって"*"の数は異なり，出力がずれてしまうのはあまり綺麗ではないため，全ての見出し ((例)head : a) の出力が":"で揃うようにする．そのため，9 行目の for 文の中で，指定した layer と"*"の数の差だけ空白を入れることで出力が揃うようにし，"*"と空白，見出し名の文字列を連結させた．

改良後に同じ sample コードで layer を 3 と指定して実行した結果は

```
my_help called with name : sample, item :
*               head1  :  a
**             head1-2  :  b
***            head1-3  :
*               head2  :
**             head2-2  :
***            head2-3  :  d
      layerが指定されているときは，指定されたlayerに合うように階層表示
```

図 4.6: 改良後の print_contents を 3 階層まで出力.

となる．階層数は"*"の数で示し，見出し名は":"で揃えたことでより綺麗なデザインの階層表示になった.

4.5.4 指定した layer が-1 である場合の出力

次に指定した layer が-1 である場合，全ての list を表示するように print_contents.rb を改良していく.

まず，maximize メソッドを追加した.

```
1  def maximize()
2        max = 0
3        @contents.each do |key, val|
4   n = @contents[key][:head_num]
5
6   if max < n
7     max = n
8   end
9        end
10       max
11     end
```

ここでは， head_num(org の場合 "*"の数，md の場合 "#"の数) の 最大値を返している．head_num の最大値は list メソッドで用いるために追加した.

次に，layer が-1 の場合に全ての list を表示させるために print_contents.rb の list メソッドを改良した．以下のコードが list メソッドに追加した部分となる.

```
1  if layer == -1
2    s1 = ""
```

```
 3    for i in 1..n
 4      s1 += "*"
 5    end
 6
 7    s2 = ""
 8    for j in 1..maximize()  - n
 9      s2 += " "
10    end
11
12    s3 = format("%20s : %s\n", key, desc)
13    output += s1 + s2 + s3
14  end
```

layer が-1 の場合，3 行目の for 文の中で "*"を head_num と同じ数出力する．

見出し名の":"で出力を揃うように出力させたい．しかし，階層によって"*"の数は異なるため，出力がずれてしまう．

そこで，8 行目の for 文の中で maximize メソッドを呼び出し，head_num の最大値と"*"の数の差だけ空白を入れることで出力が揃うようにした．そして空白と見出し名の文字列を連結させて出力した．

以下の markdown で記述した sample コードを用いて，layer が-1 を指定した場合の実行結果は

```
# head1 : a
## head1-2 : b
### head1-3 :
#### head1-4 : c
- hoge
##### head1-5
# head2 :
- hage
- hage
## head2-2 :
### head2-3 : d
```

```
my_help called with name : sample, item :
*                  head1  :  a
**                 head1-2  :  b
***                head1-3  :
****               head1-4  :  c
*****              head1-5  :
*                  head2  :
**                 head2-2  :
***                head2-3  :  d
      layerが-1と指定したとき，全て表示する
```

図 4.7: print_contents layer を-1 と指定した時の出力.

となる．全ての見出しを出力し，出力の形式についても，階層表示と同様 見出し名の":" で揃えることで綺麗な出力となった.

4.5.5 改良後の list_spec.rb の結果

本研究では my_help で list の表示する際に階層表示を行えるように改良した.

以下は改良後の list.rb をテストする list_spec.rb の結果である.

```
riku@rikuShigetos-MacBook-Air ~/my_help (main)> bundle exec rspec spec/unit/list
_spec.rb

MyHelp::List
  list
    拡張子が.orgならそっちで動く
      ヘルプ名がないときは，全てのヘルプとその簡単な内容説明を表示
      ヘルプ名があるときは，その中の全てのitemを表示
      item名があるときは，そのcontentを表示
      item名に似たitemがあるときは，そのcontentを表示
      item名に似たitemが見つからないときは，その旨を返す
      layerが指定されているときは，指定されたlayerに合うように階層表示
      指定したlayerが2の時,2階層まで表示
      layerが-1と指定したとき，全て表示する
    拡張子が.mdならそっちで動く
      ヘルプ名がないときは，全てのヘルプとその簡単な内容説明を表示
      ヘルプ名があるときは，その中の全てのitemを表示
      item名があるときは，そのcontentを表示
      item名に似たitemがあるときは，そのcontentを表示
      item名に似たitemが見つからないときは，その旨を返す
      layerが指定されているときは，指定されたlayerに合うように階層表示
      指定したlayerが2の時,2階層まで表示
      layerが-1と指定したとき，全て表示する

Finished in 0.0034 seconds (files took 0.09389 seconds to load)
16 examples, 0 failures
```

図 4.8: list_spec.rb の結果.

このように全てのテスト項目に対して通るように改良できた．また，本研究によって，新しく layer を指定した場合の階層表示と拡張子が”.md”の場合も list 表示できるようになったことが分かる.

第5章　今後

本研究では list 表示させる場合に layer を指定すると，階層表示するように改良を加えた．しかし，item 名がない場合のみの改良となっている．

以下のコードは list_spec.rb で新しく追加したい item 名と layer を指定した場合のテストである．

```
1  it "help 名と item 名と layer が指定された時，その layer の contents を表示する" do
2    output = "- content_c"
3    ext = ".md"
4    layer = 2
5    help_options = "example b_item"
6    expect(List.new(templates_path, ext, layer).list(help_options)).to be_include(output)
7  end
```

以下の markdown で記述した sample は 5 行目の help_options で指定した”example”の中身である．

```
# head
- help_title example
# license
-      cc by Shigeto R. Nishitani, 2016-22
# a_item
- content_a
# b_item
- content_b
## c_item
- content_c
```

ここでは item 名がある場合に layer は 2，拡張子を”.md” と指定した場合 b_item の 2 階層目にある c_item の中の

```
- content_c
```

だけを表示したいと考えている．今後はこのテストコードに対して通るように list メソッドを改良する必要があると考えている．

第6章　総括

本研究では，my_help をより使いやすくするために改良を行なった．開発内容は以下の通りである．

- 階層表示にするためのプログラムである print_contents.rb の開発
- markdown で記述したテキストを Hash に格納するためのプログラムである md2hash_new.rb の開発
- org で記述したテキストを Hash に格納するためのプログラムである org2hash_new.rb の開発
- lib/my_help/list.rb のファイルから読み込む read_help メソッド，list を表示する list_helps メソッド，index から呼び出す list_help_with メソッド，item 名がある場合の処理をする find_near メソッドの書き換え
- spec/unit で開発したプログラムに対するテストの作成

これらによって，これまで my_help は org のみの list 表示となっていたが，拡張子を”.md”と指定することで markdown で記述したテキストも list 表示することが可能となった．

また，list 表示させる際に見たいメモをすぐに見つけれるよう指定した layer に応じて，階層表示するように改良したため，どこにどのメモがあるか見つけやすくなっている．これらによって my_help はより使い勝手の良いものとなった．

謝辞

本研究を行うにあたり，終始丁寧なご指導をして下さった西谷滋人教授に深く感謝申し上げます．また，同研究室の皆様にも研究を進めるにあたって多大なご助言とご協力頂き深く感謝申し上げます．本当にありがとうございました．

参考文献

[1] RubyGems - https://rubygems.org/ (accessd on 20 Jan 2021).

[2] ”my_help/README.org” Daddygongon, https://github.com/daddygongon/my_help.

[3] ”Effective Testing with RSpec 3”,M. Marston and E. Dees,(Pragmatic Bookshelf, 2017).

[4] ”達人プログラマー -熟達に向けたあなたの旅-” 第 2 版, David Thomas and Andrew Hunt (著), 村上雅章 (訳) ,(オーム社,2020).

[5] ”UML ユーザガイド 第 2 版”, グラディー · ブーチ, ジェームズ · ランボー, イヴァー · ヤコブソン,(ピアソン桐原, 2010/3/1).

[6] ”テスト駆動開発 Kindle 版”, Kent Beck (著), 和田卓人 (翻訳), (オーム社, 2017/10/13).