Styl kódování

Table of Contents

1. Úvod

Red je homoikonický jazyk, jehož kód je prezentován jako data. Důsledkem této vlastnosti je jeho téměř volná forma při formátování dat, stejně jako dostatečná flexibilita pro potřeby formátování v DSL. Následující obsah je *pouze jedním z přemnoha způsobů* formátování kódu Red.

Tento dokument popisuje *oficiální kódovací styl, použitý ve zdrojovém kódu Redu*, jehož dodržování je nezbytnou podmínkou u každého "pull request" pro připojení do repozitáře `red/red` na Github.

Protože Red/System je dialekt Redu, sdílí stejná pravidla skladby a kódovacího stylu. Specifická pravidla pro Red/System jsou označena.

Účelem následujících pravidel je maximalizace čitelnosti, včetně dodržování optimálního počtu řádků, viditelných v displeji editoru, jakož i minimalizace potřeby komentářů.

2. Délka řádku

Neexistuje striktně definovaný maximální počet sloupců pro řádek kódu, neboť ten může záviset na typu použitého fontu (velikost, proporcionální versus fixní šířka) nebo na zvýrazňovacích efektech. Mělo by být možné číst celý řádek kódu (vyjímaje komentáře) v editoru, zabírajícím maximálně polovinu monitoru šířky 1080 p. Na displejích, které používáme pro kódování základního kódu Redu, to je asi 100 sloupců. V následujícím popise se bude výraz *nadměrná velikost* nebo *příliš dlouhý* vztahovat na kód, nekonformní s výše uvedenými kriterii.

3. Odsazení

Základní kód (codebase) Redu používá pro odsazení kódu tabulaci o velikosti **4**. To poskytuje dobrý kompromis mezi příliš malým (např. 2 sloupce) nebo příliš velkým (např. 8 slopců) odsazením. Použití tabulátoru umožňuje akomodaci osobní preference uživatele daným pravidlům.

Všechny přispívající zdrojové soubory do repozitáře `red/red` by měly ve svém záhlaví obsahovat následující pole:

```
Tabs: 4
```

Při každém přechodu na nový řádek po otevření bloku či závorek se má použít odsazení o velikosti jednoho tabulátoru.

Správně

```
func [
    arg1
    arg2
    ...
][
    print arg1
    ...
]
```

Nesprávně

```
func [
arg1                ;-- chybějící odsazení!
arg2
...
][
    print arg1      ;-- nadměrné odsazení!
        ...
]
```

4. Uspořádání bloků

Všechna následující pravidla se vztahují na bloky `[]` a na závorky `()`.

Prázdné bloky neobsahují žádné mezery:

```
a: []
```

Navazující bloky nepotřebují mezeru mezi koncem jednoho a začátkem druhého.

```
[][]
[]()
```

```
[
   hello
][                      ;-- mezery nejsou požadovány
world
]
```

Je ovšem přípustné používat mezery mezi vnořenými bloky:

```
array: [[] [] [] []]
list:  [ [] [] [] [] ]

either a = 1 [["hello"]][["world"]]
either a = 1 [ ["hello"] ][ ["world"] ]
```

Malé bloky výrazů jsou obvykle otevřeny a uzavřeny na témže řádku

```
b: either a = 1 [a + 1][3]
```

Je-li řádek příliš dlouhý, má být blok zalomen (wrapped) do několika řádků s jednou úrovní odsazení:

Správně

```
b: either a = 1 [
    a + 1
][
    123 + length? mold a
]
```

Nesprávně

```
b: either a = 1
    [a + 1]
    [123 + length? mold a]
```

Tento styl je špatný, protože narušuje schopnost kopírování (copy/paste) kódu do konzoly Redu (řádek s `either` bude vyhodnocen před detekcí argumentů bloku).

Je-li první blok dostatečně malý a vejde se na řádek, potom jsou pouze následné bloky zalomeny do

několika řádků:

```
print either a = 1 ["hello"][
    append mold a "this is a very long expression"
]

while [not tail? series][
    print series/1
    series: next series
]
```

5. Určování jmen

Jména proměnných by měla být jednoslovná *podstatná jména*. Vybírejte slova krátká a výstižná. Přednost mají obecná slova (zejména pokud jsou již použita ve stejném kontextu v existujícím zdrojovém kódu Redu.) V případě potřeby je vhodné vyhledat nejlepší termín ve slovníku synonym. Jednopísmenová slova nebo zkratky nejsou pro označení vhodná.

Slova víceslovných názvů se oddělují pomlčkou `-`. Dvouslovné názvy se použijí v případě, že neexistuje jednoslovná varianta, nebo je jednoslovný název zaměnitelný s již existujícím. Názvy proměnných, složené z více než dvou slov, by se měly používat jen ve výjimečných případech. Používání jednoslovných názvů činí kód kompaktnější v horizontálním směru a tudíž mnohem čitelnější.

Správně

```
code: 123456
name: "John"
table: [2 6 8 4 3]
lost-items: []

unless tail? list [author: select list index]
```

Nesprávně

```
code_for_article: 123456
Mytable: [2 6 8 4 3]
lostItems: []

unless tail? list-of-books [author-property: select list-of-books selected-index]
```

Názvy funkcí by měly být jednoslovnými *slovesy*, naznačujícími zamýšlenou akci, byť dvou- nebo tříslovné názvy jsou často také nezbytné. Názvoslovná konvence se také vztahuje na na názvy funkcí. Zde je přípustné podstatné či přídavné jméno, doplněné otazníkem. Ten často naznačuje, že zpětná hodnota je typu `logic!`, což však není striktní pravidlo, protože je vhodné vytvářet jednoslovné názvy akcí pro zjištění vlastnosti (např. `length?`, `index?`). Při určování víceslovného

názvu funkce se vždy dává slovo na první místo. Jsou-li názvy proměnných a funkcí vybrány pečlivě, stává se kód téměř samodokumentační s malou potřebou komentářů.

Správně

```
make:   func [...
reduce: func [...
allow:  func [...
crunch: func [...
```

Nesprávně

```
length:    func [...
future:    func [...
position:  func [...
blue-fill: func [...     ;-- má být fill-blue
```

Existuje výjimka pro jména, související s OS nebo s API třetí strany. Pro snadnou odlišitelnost od regulerních jmen kódu Red nebo Red/System by se jejich původní jméno mělo stát součástí nově tvořeného jména, například:

```
tagMSG: alias struct! [
    hWnd    [handle!]
    msg     [integer!]
    wParam  [integer!]
    lParam  [integer!]
    time    [integer!]
    x       [integer!]
    y       [integer!]
]

#import [
    "User32.dll" stdcall [
        CreateWindowEx: "CreateWindowExW" [
            dwExStyle    [integer!]
            lpClassName  [c-string!]
            lpWindowName [c-string!]
            dwStyle      [integer!]
            x            [integer!]
            y            [integer!]
            nWidth       [integer!]
            nHeight      [integer!]
            hWndParent   [handle!]
            hMenu        [handle!]
            hInstance    [handle!]
            lpParam      [int-ptr!]
            return:      [handle!]
        ]
    ]
]
```

6. Velikost písmen

Všechny názvy proměnných a funkcí se píší malými písmeny, pokud není dobrý důvod použít velká písmena:

- jméno je obecně známým akronymem, např. GMT (Greenwich Mean Time)
- jméno je OS nebo (non-Red) API třetí strany

7. Makra (Red/System)

Pro názvy maker se obvykle používají velká písmena kvůli odlišení od zbytku kódu (pokud není přímo záměrem aby se nelišily od normálního kódu, 'like pseudo-custom datatype definitions'). Slova víceslovných názvů se oddělují podtržítkem _.

(TBD: extract all single-word names used in the Red codebase as examples)

8. Definice funkcí

Obecné pravidlo je udržet specifikaci bloku na jednom řádku. Tělo bloku může být na jednom či více řádcích. V případě Red/System, kde mají specifikace bloků tendenci být delší, je většina specifikací rozbalena (wrapped) do několika řádků a tak kvůli vizuální konzistenci jsou malé specifikace bloků rozbaleny.

Správně

```
do-nothing: func [][]
increment: func [n [integer!]][n + 1]

increment: func [n [integer!]][
    n + 1
]

increment: func [
    n [integer!]
][
    n + 1
]
```

Nesprávně

```
do-nothing: func [
][
]

do-nothing: func [

][

]

increment: func [
    n [integer!]
][n + 1]
```

Je-li specifikace bloku příliš dlouhá, měla by být rozvinuta (wrapped) přes několik řádek ale tak, aby každá definice typu byla na stejném řádku, jako jeho argumenty. Nepovinné argumenty mají být na samostatném řádku. Každé upřesnění začíná na novém řádku. Je-li následováno jediným argumentem, může tento být na témže řádku nebo na novém řádku s odsazením (nutno zachovat konsistenci s ostatními upřesněními téhož bloku). U upřesnění `/local` mohou být argumenty na témže řádku, pokud nejsou lokální slova doprovázena popisem typu.

Při rozbalování (wrappin) specifikace bloku přes několik řádek se pro usnadnění čtení doporučuje zarovnat definice následujících argumentů do sloupců. Takové zarovnání se přednostně provede s

použitím tabulátoru, případně mezerníkem.

Správně

```
make-world: func [
    earth   [word!]
    wind    [bitset!]
    fire    [binary!]
    water   [string!]
    /with
        thunder [url!]
    /only
    /into
        space [block! none!]
    /local
        plants animals men women computers robots
][
    ...
]
```

Nesprávně

```
make-world: func [
    [throw] earth [word!]       ;-- blok atributů nemá svůj vlastní řádek
    wind    [bitset!]
    fire [binary!]              ;-- nezarovnaná deklarace typu
    water   [string!]
    /with
        thunder [url!]
    /only
    /into space [block! none!]  ;-- nekonzistntní s upřesněním `/with`
    /local
        plants animals          ;-- příliš brzké zalomení řádku
        men women computers robots
][
    ...
]
```

Hlavní dokumentační řetězec (popisující funkci) má mít svůj vlastní řádek, je-li specifikační blok zalomen. Dokumentační řetězce argumentů a upřesnění by měly být na stejném řádku jako objekt, který popisují. Dokumentační řetězce začínají velkým písmenem a nemusí být ukončeny tečkou (jež se přidává automaticky po volání funkce `help` v konzole).

Správně

```
increment: func ["Add 1 to the argument value" n][n + 1]

make-world: func [
    "Build a new World"
    earth   [word!]     "1st element"
    wind    [bitset!]   "2nd element"
    fire    [binary!]   "3rd element"
    water   [string!]
    /with               "Additional element"
        thunder [url!]
    /only               "Not implemented yet"
    /into               "Provides a container"
        space [unset!]  "The container"
    /local
        plants animals men women computers robots
][
    ...
]
```

Nesprávně

```
make-world: func ["Build a new World"   ;-- má být na novém řádku
    earth   [word!]     "1st element"
    wind    [bitset!]     "2nd element" ;-- nadměrné odsazení
    fire    [binary!]
    "3rd element"               ;-- má být na témže řádku jako `fire`
    water   [string!]
    /with               "Additional element"
        thunder [url!]
    /only "Not implemented yet" ;-- má být zarovnáno s jinými docstringy
    /into
        "Provides a container"
        space [unset!]  "The container"
    /local
        plants animals men women computers robots
][
    ...
]
```

9. Volání funkcí

Argumenty se při volání funkce píší na stejný řádek jako název funkce. Stává-li se řádek příliš dlouhý, mohou být argumenty zalomeny do několika řádků (co argument, to řádek) s odsazením.

Správně

```
foo arg1 arg2 arg3 arg4 arg5

process-many
    argument1
    argument2
    argument3
    argument4
    argument5
```

Nesprávně

```
foo arg1 arg2 arg3
    arg4 arg5

foo
    arg1 arg2 arg3
    arg4 arg5

process-many
    argument1
        argument2
            argument3
            argument4
                argument5
```

U dlouhých výrazů s mnoha vloženými částmi, může být rozlišení hranic někdy obtížné. Používání závorek pro seskupení vložených volání s příslušnými argumenty je přijatelné (nikoliv povinné).

```
head insert (copy/part [1 2 3 4] 2) (length? mold (2 + index? find "Hello" #"o"))

head insert
    copy/part [1 2 3 4] 2
    length? mold (2 + index? find "Hello" #"o")
```

10. Komentáře

- text komentáře začíná předponou `;` nebo `;-- ` (silnější vizuální vodítko)
- jednořádkové komentáře začínají ve sloupci 57 (nebo 53)
- víceřádkové komentáře začínají předponou `;` na začátku každého řádku nebo se píší s použitím konstrukce `comment {···}`.

Zpravidla se má kvůli úspoře významného vertikálního místa vkládat komentář do stejného řádku jako počátek odpovídajícího kódu, místo na nový řádek. Pokud je však komentář použit také jako oddělovač odlišných porcí kódu, je jeho umístění na nový řádek vhodné.

11. Skladba řetězce

Pro jednořádkové řetězce se používají dvojité uvozovky "". Forma {} je vyhražena pro víceřádkové řetězce. Dodržování tohoto pravidla zajišťuje:

- konzistentnější prezentaci zdroje před a po kódu LOAD (?)
- lepší vyjádření významu

Výjimku z tohoto pravidla vytváří situace, když jednořádkový řetězec obsahuje znak ". V tom případě je lepší i pro jednořádkový řetězec použít formu {}, která je čitelnější než případné použití "escaping" znaků ^".

12. New line usage

TBD