Informatika pro moderní fyziky (6)
Vstupní soubory pro výpočetní programy
Tvorba textových dokumentů

František HAVLŮJ

e-mail: haf@ujv.cz

ÚJV Řež
oddělení Reaktorové fyziky a podpory palivového cyklu

akademický rok 2016/2017
9. listopadu 2016

1. Co jsme se naučili minule
2. Pracky pryč, padouchu!
3. Automatizace tvorby vstupů
4. Závěrečná zpráva
5. Výroba dokumentu v praxi
6. Na šablony chytře

Obsah

1 Co jsme se naučili minule

2 Pracky pryč, padouchu!

3 Automatizace tvorby vstupů

4 Závěrečná zpráva

5 Výroba dokumentu v praxi

6 Na šablony chytře

- načítání složitějšího datového souboru
- důkladné procvičení práce s hashi a poli

Obsah

1 Co jsme se naučili minule

2 Pracky pryč, padouchu!

3 Automatizace tvorby vstupů

4 Závěrečná zpráva

5 Výroba dokumentu v praxi

6 Na šablony chytře

Klávesnice a myš

- myš je dobrá na grafiku a jako alternativa k tabletu
- taky se hodí tam, kde se potřebuju přesouvat mezi položkami, které nemají jednoznačné pořadí nebo prostorový vztah (neseřazené ikony na ploše, rozhraní s mrakem oken atd.)
- případně ještě na použití menu pro úkony, které dělám jednou za uherský rok
- naopak na programování je nejlepší na myš vůbec nešahat a používat skoro jenom klávesnici
- extrémní školy dokonce brojí proti kurzorovým šipkám, protože (na velké klávesnici) nutí měnit polohu rukou, což je pomalé

Přepínání jazyků

- je dobré se mu vyhnout, protože to opravdu trochu otravuje (i když se s tím dá docela dobře žít, pokud máte dobrou klávesovou zkratku)
- rozhodně stojí za to zjistit – například pro psaní v LaTeXu – kde na české klávesnici máte potřebné speciální znaky (v tomto případě zejména backslash a složené závorky)
- chytré editory mají různé pochystávky a makra, která vám umožní se těmto speciálním znakům defacto vyhnout
- pokud můžu, pracuju celou dobu s anglickou (tj. pokud výjimečně nepíšu český dokument)

Klávesové zkratky

- jako s programováním – musím se něco naučit / zapamatovat, ale pak mi to ušetří hromadu času
- minimálně základní sadu stojí za to se naučit
- často jdou ručně editovat, ale většinou to není nutné (a je to stejně na houby, pokud zrovna nesedíte u svého počítače)
- jako s hudebním nástrojem – za čas už neznáte ty zkratky, ale prostě je umíte zmáčknout bez přemýšlení
- hodně jich je sdílených napříč programy a editory

Klávesové zkratky - MS Windows

- copy-paste `Ctrl+C/V`
- undo `Ctrl+Z`
- přepínání aplikací `Alt+Tab`
- přepínání oken v rámci aplikace `Ctrl+Tab`

Klávesové zkratky – Notepad++

- pohyb v textu po slovech a stránkách `Ctrl`+šipky, `PgUp/PgDn`
- uložení, otevření, zavření `Ctrl+S, O, W`
- změna odsazení bloku `Tab / Shift+Tab`
- přepínání mezi soubory
- zakomentovat/odkomentovat `Ctrl+Q`

Obsah

1 Co jsme se naučili minule

2 Pracky pryč, padouchu!

3 Automatizace tvorby vstupů

4 Závěrečná zpráva

5 Výroba dokumentu v praxi

6 Na šablony chytře

Určení poloh tyčí

Ve vstupním souboru si najdeme relevantní část:

```
c ---------------------------------
c polohy tyci (z-plochy)
c ---------------------------------
c
67 pz 47.6000     $ dolni hranice absoberu r1
68 pz 40.4980     $ dolni hranice hlavice r1
69 pz 44.8000     $ dolni hranice absoberu r2
70 pz 37.6980     $ dolni hranice hlavice r2
```

Určení poloh tyčí

příklad `c1_10_20`:

```
c -------------------------------
c polohy tyci (z-plochy)
c -------------------------------
c
67 pz 47.6000     $ dolni hranice absoberu r1
68 pz 40.4980     $ dolni hranice hlavice r1
69 pz 44.8000     $ dolni hranice absoberu r2
70 pz 37.6980     $ dolni hranice hlavice r2
```

Výroba šablon

Jak dostat polohy tyčí do vstupního souboru? Vyrobíme šablonu, tzn nahradíme

```
67 pz 47.6000     $ dolni hranice absoberu r1
```

Výroba šablon

Jak dostat polohy tyčí do vstupního souboru? Vyrobíme šablonu, tzn nahradíme

```
67 pz 47.6000     $ dolni hranice absoberu r1
```

nějakou značkou (*placeholder*):

```
67 pz %r1%     $ dolni hranice absoberu r1
```

Chytáky a zádrhele

- kromě samotné plochy konce absorbéru je nutno správně umístit i z-plochu konce hlavice o 7,102 cm níže
- obecně je na místě ohlídat si, že placeholder nebude kolidovat s ničím jiným

Doporučené nástroje jsou:

- již známá funkce `gsub` pro nahrazení jednoho řetězce jiným
- pro pragmatické lenochy funkce `File.read` načítající celý soubor do řetězce (na což nelze v Pascalu ani pomyslet)
- možno ovšem použít i `File.readlines` (v čem je to lepší?)

Realizace

```
DELTA = 44.8000 - 37.6980

template = File.read("template")
(0..10).each do |i1|
  (0..10).each do |i2|
    r1 = i1 * 50
    r2 = i2 * 50
    File.open("inputs/c_#{i1}_#{i2}", "w") do |f|
      s = template.gsub("%r1%", r1.to_s)
      s = s.gsub("%r1_%", (r1 - DELTA).to_s)
      s = s.gsub("%r2%", r2.to_s)
      s = s.gsub("%r2_%", (r2 - DELTA).to_s)
      f.puts template
    end
  end
end
```

A co takhle trocha zobecnění?

- když budu chtít přidat další tyče nebo jiné parametry, bude to děsně bobtnat
- funkce `process("template", "inputs/c_#{i1}_#{i2}", {”r1” => r1, ”r2” => r2, .....})`
- všechno víme, známe, umíme

Obsah

1 Co jsme se naučili minule

2 Pracky pryč, padouchu!

3 Automatizace tvorby vstupů

4 Závěrečná zpráva

5 Výroba dokumentu v praxi

6 Na šablony chytře

Jak vyrobit zprávu?

- potřebujeme udělat hezké PDF shrnující výsledky našich výpočtů
- takže úvod, popis toho co jsme dělali a pak přehled výsledků
- tabulka s hodnotami, 11+1 graf
- co by znamenalo to dělat ve Wordu ..... ?
- hodilo by se to zautomatizovat!

Jak vygenerovat text?

- zase potřebujeme lepší nástroj na text, než jsou WYSIWYG (What You See Is What You Get) editory
- ideálně něco, co bude mít plain-text vstup (který můžeme s úspěchem generovat v Ruby) a co se pak
- odpověď zní LaTeX
- nejvíc nejlepší text-processor na světě

Koncepce oddělení obsahu a formy

- když píšu, nechci říct, že je text tučně a o dva body větší, ale že je to nadpis kapitoly
- ideálně chci popsat někde, jak bude dokument vypadat a nemíchat vzhled s obsahem
- styly ve Wordu se tomu vzdáleně blíží
- v LaTeXu vlastně píšu jen obsah a o formu se musím starat jen hodně málo
- je samozřejmostí zadarmo obsah, rejstřík atd.
- kdo píše diplomku v něčem jiném, kazí si život

Příklad jednoduchého dokumentu

- viz `document.tex`
- není potřeba úplně všemu rozumět, zatím to jen budeme upravovat v mezích zákona
- všechny příkazy začínají zpětným lomítkem, parametry jsou ve složených závorkách
- napíšu `pdflatex document.tex` a dostanu pdfko!

Text

- konce řádku nejsou důležité
- nový odstavec se dělá prázdným řádkem
- nové (pod)kapitoly pomocí příkazů `section`, `subsection`

Tabulka

- prostředí (= `begin` ... `end`)
- sloupce se

```
\begin{tabular}{ll}
a & b \\
c & d \\
\end{tabular}
```

Úkol

- vyrobit PDF s výsledky
- 11 grafů keff
- 1 graf závislosti kritické polohy R2 na poloze R1
- tabulka keff
- tabulka kritických poloh
- (a my to dělat nebudeme)

Pozor na backslash

- v Ruby se v řetězci backslash `\` používá jako escape character
- např konec řádky je `\n`
- pokud chci vytiskout zpětné lomítko (což bude asi pro LaTeX potřeba), musím ho zdvojit: `\\`

Složitější práce

- LaTeX je ideální pro rozsáhlé texty
- výzkumák, diplomka, disertace se naformátuje sama a všechno funguje bez trápení a snažení
- odkazy, reference, citace .. všechno bez starostí

Prezentace

- balíček `beamer`
- bez nutnosti se uklikat to samo od sebe vypadá slušně
- viz tahle prezentace
- nevýhoda(?): obtížnost přizpůsobit rozmístění textu apod., ale na druhou stranu to aspoň drží jednotný styl

Obsah

1 Co jsme se naučili minule

2 Pracky pryč, padouchu!

3 Automatizace tvorby vstupů

4 Závěrečná zpráva

5 Výroba dokumentu v praxi

6 Na šablony chytře

Úkol na dnešek

- pro jeden blok JE mám provozní data - v určitých dnech hodnotu koncentrace kyseliny borité a axiálního ofsetu - pro několik kampaní (blíže neurčený počet)
- chci vyrobit přehledové PDF, které bude hezky prezentovat grafy obou veličin pro každou kampaň a k tomu i tabulky
- data pro jednotlivé kampaně mám v CSV souborech, každý má tři sloupce (datum, cB, AO)

Rozbor

- načíst tabulky a vykreslit grafy umíme
- převést tabulky v CSV na tabulky v LaTeXu se záhy naučíme
- vložit obrázek do latexu taky umíme
- předem neznámý počet souborů nás netrápí (`Dir["*.csv"]`)

Tak nejdřív ty grafy

- to už je vážně obehraná písnička, ale tady je aspoň trochu změna
- potřebujeme vybrat, které dva sloupce použít - parametr `using`
- `plot "data.csv" using 1:2`
- datum na vodorovné ose – potřeba načíst ve správném formátu atd.
- `set xdata time`
- `set timefmt "%m/%d/%Y"`
- s hvězdičkou: nastavit nadpis a popisky os

Ne tak úplně chytře

```
Dir["*.csv"].each do |fn|
  base = fn.split(".").first

  File.open("#{base}_bc.gp", "w") do |f|
    f.puts "set terminal png"
    f.puts "set xdata time"
    f.puts "set timefmt \"%m/%d/%Y\""
    f.puts "set output \"#{base}_bc.png\""
    f.puts "plot \"#{base}.csv\" using 1:2"
  end
  `gnuplot #{base}_bc.gp`

  File.open("#{base}_ao.gp", "w") do |f|
    f.puts "set terminal png"
    f.puts "set output \"#{base}_ao.png\""
    f.puts "set xdata time"
    f.puts "set timefmt \"%m/%d/%Y\""
    f.puts "plot \"#{base}.csv\" using 1:3"
  end
  `gnuplot #{base}_ao.gp`
end
```

DRY

- základní paradigma: DRY = don't repeat yourself
- nemá cenu psát dvě věci stejně
- použití `copy and paste` při programování je varovný signál
- pokud nejsou stejně, ale skoro stejně, je potřeba trochu chytrosti
- připomeňme si hash: `{"a" => 1, "b" => 2}`
- přes hash se dá iterovat:

```
{"a" => 1, "b" => 2}.each do |key, value|
```

Grafy – DRY

```
Dir["*.csv"].each do |fn|
  base = fn.split(".").first
  {"bc" => 2, "ao" => 3}.each do |var, col|
    File.open("#{base}_#{var}.gp", "w") do |f|
      f.puts "set terminal png"
      f.puts "set xdata time"
      f.puts "set timefmt \"%m/%d/%Y\""
      f.puts "set output \"#{base}_#{var}.png\""
      f.puts "plot \"#{base}.csv\" using 1:#{col}"
    end
    `gnuplot #{base}_#{var}.gp`
  end
end
```

Jak na tabulky

- tabulky budou dost rozsáhlé a montovat je přímo nějak do latexových vstupů je asi spíš nepraktické, naštěstí to jde i jinak
- naštěstí má LaTex příkaz `\input`, kterým můžeme prostě vložit do dokumentu nějaký externí soubor
- takže si nejdřív přichystáme soubory s tabulkami a pak se na ně budeme už jenom odkazovat

Jak na tabulky v LaTeXu (1)

Základem tabulky je prostředí `tabular` s definicí počtu a zarovnání sloupců:

```
\begin{tabular}{lrr}
  ...
\end{tabular}
```

Jak na tabulky v LaTeXu (2)

Uvnitř tabulky se sloupce oddělují ampersandem a řádky dvojitým backslashem:

```
\begin{tabular}{lrr}
  Data 1 & a & 1.0 \\
  Data 2 & b & 2.0 \\
  Data 3 & c & 3.0 \\
\end{tabular}
```

Jak na tabulky v LaTeXu (3)

Přidání mřížky je nesnadné, leč proveditelné a vlastně docela dobře vymyšlené - přidáváme jednotlivé čáry po sloupcích a řádcích:

```
\begin{tabular}{|l|r|r|}
  \hline
  Data 1 & a & 1.0 \\
  \hline
  Data 2 & b & 2.0 \\
  \hline
  Data 3 & c & 3.0 \\
  \hline
\end{tabular}
```

Úkol na teď: výroba tabulek

- vyrobit z CSV souboru (tři sloupce) dvě LaTeX tabulky (po dvou sloupcích)
- postarat se, aby byly hezké
- chytré je vyrobit tabulku třeba o šesti sloupcích (jakože tři dvousloupce), pak už se to na stránku v klidu vejde

Obsah

1 Co jsme se naučili minule

2 Pracky pryč, padouchu!

3 Automatizace tvorby vstupů

4 Závěrečná zpráva

5 Výroba dokumentu v praxi

6 Na šablony chytře

Úskalí šablon

- snadno umíme nahradit jeden řetězec druhým
- trochu méně pohodlné pro větší bloky textu
- navíc by se hodila nějaká logika (cyklus) přímo v šabloně
- naštěstí jsou na to postupy

ERb (Embedded Ruby)

- lepší šablona - “aktivní text”
- používá se například ve webových aplikacích
- hodí se ale i na generování latexových dokumentů, resp. všude, kde nám nesejde na whitespace
- poměrně jednoduchá syntax, zvládne skoro všechno (viz předmět MAA3)

Základní syntaxe ERb (1)

Jakýkoli Ruby příkaz, přiřazení, výpočet ...

```
<% a = b + 5 %>
<% list = ary * ", " %>
```

Základní syntaxe ERb (2)

Pokud chci něco vložit, stačí přidat rovnítko

```
<%= a %>
<%= ary[1] %>
<%= b + 5 %>
```

Základní syntaxe ERb (3)

Radost je možnost použít bloky a tedy i iterátory apod. v propojení s vkládaným textem:

```
<% (1..5).each do |i| %>
Number <%= i %>
<% end %>
<% ary.each do |x| %>
Array contains <%= x %>
<% end %>
```

ERb – shrnutí

- dobrý sluha, ale špatný pán
- můžu s tím vyrobit hromadu užitečných věcí na malém prostoru
- daň je velké riziko zamotaného kódu a nízké přehlednosti (struktura naprosto není patrná na první pohled, proto je namístě ji držet maximálně jednoduchou)

Důležité upozornění

- oddělení modelu a view
- přestože lze provádět zpracování dat a výpočty přímo v ERb, je to nejvíc nejhorší nápad
- je chytré si všechno připravit v modelu (tj. v Ruby skriptu, kterým data chystáme)
- a kód ve view (tj. v ERb šabloně) omezit na naprosté minimum

Jak ze šablony udělat výsledek

Příklad překladu ERb

```
require "erb_compiler"

erb(template, filename, {:x => 1, :y => 2})
```

Příklad – kreslení grafů z minula

template.gp

```
set terminal png
set output "plot_<%=n%>.png"
plot "data_<%=n%>.csv"
```

```
(1..10).each do |i|
  erb("template.gp", "plot_#{i}.gp", {:n => i})
end
```

Takže v latexu třeba

```
\subsection{Koncentrace kyseliny borité}

<% files.each do |f| %>

\subsubsection{Kampaň <%= f.split("_").last %>}
\begin{center}
\includegraphics[width=0.8\textwidth]{<%= f %>_bc.eps}
\end{center}
<% end %>
```

A teď už to jenom dejte dohromady...

1. připravit si základní kostru dokumentu v latexu
2. převést na šablonu: mít seznam souborů, správně generovat kapitoly
3. vyrobit grafy
4. vložit grafy do šablony
5. vyrobit tabulky
6. vložit tabulky do šablony
7. A JE TO!

A to je vše, přátelé!