CSCI-GA.3033.003 Scripting Languages

9/11/2013

Textual data processing (Perl)

Announcements

- If you did not get a PIN to enroll, contact Stephanie Meik

Outline

- Perl Basics (continued)
- Regular Expressions

Arrays

- Resizable
- Literals: list `@a=(1,3,5)`, range `@b=2..4`
- Indexing: e.g. `$a[1]`
  - Zero-based; negative index counts from end
  - `$#a` returns last index of `@a`, in this case, `2`
  - Write to non-existent index auto-vivifies
- Free: `undef @a`, truncate: `$#a=1`
- Array slice: using multiple indices, e.g., `@a[0,2]` or `@a[1..2]`
- Using array in scalar context: returns length
  - `scalar(@a); # 3 = size`

Hashes

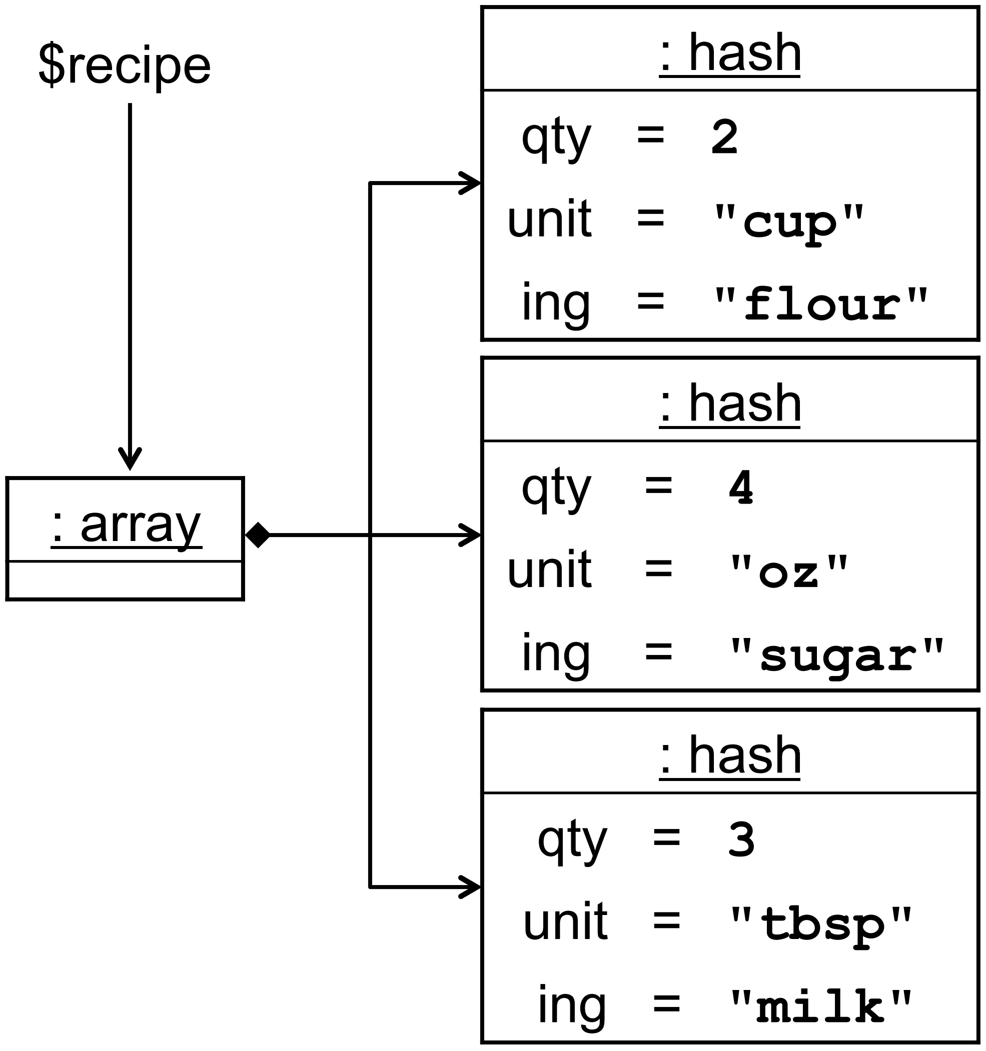
- Associative arrays, using hash tables
- Literals: none; use list literals instead
  - `%h = (lb => 1, oz => 16, g => 453);`
- Indexing: string, bare-word quotes optional
  - `$h{"oz"}`, `$h{oz}`
  - Write to non-existing index auto-vivifies
- Free: **`undef %h;`** delete: **`undef $h{oz};`**
- Hash slice: returns array, e.g.,
  `@a = @h{lb, g};` returns **`(1, 453)`**
- Using hash in scalar context: returns string
  - `scalar(%h); # ”2/8"=buckets/capacity`

Array and Hash References

- References are scalars, only scalars can be stored in other arrays/hashes
- Literals: "anonymous array/hash composers"
  - **`$ar=[1,3,5];`**, **`$hr={x=>2,y=>0};`**
- Indexing: e.g., **`$ar->[1]; $hr->{y};`**
  - Dereference operator **`->`** optional in nested chain, e.g., **`$rarh->[123]{abc}`**
  - Write to non-existent index auto-vivifies all indices on the way
- Using reference in string context: returns string describing the type and address
  - **`print $ar; # "ARRAY(0x18bef54)"`**

Data Structures

```
$recipe = [
  {
    qty  => 2,
    unit => "cup",
    ing  => "flour",
  },
  {
    qty  => 4,
    unit => "oz",
    ing  => "sugar",
  },
  {
    qty  => 3,
    unit => "tbsp",
    ing  => "milk",
  },
];
```



Control Statements

Proper statements (man perlsyn)	**`if(`***expr***`){…}`** [**`elsif(`***expr***`){…}`**] … [**`else{…}`**]
	[*label*`:`] **`while(`***expr***`){…}`** [**`continue{…}`**]
	[*label*`:`] **`until(`***expr***`){…}`** [**`continue{…}`**]
	[*label*`:`] **`for(`***expr*`;` *expr*`;` *expr***`)`** **`{…}`**
	[*label*`:`] **`foreach`** [*var*] **`(`***list***`)`** **`{…}`** [**`continue{…}`**]
	[*label*`:`] **`for`** [*var*] **`(`***list***`)`** **`{…}`** [**`continue{…}`**]
	[*label*`:`] **`{…}`** [**`continue{…}`**]
Modifiers	*stmt* (**`if`** | **`unless`** | **`while`** | **`until`** | **`foreach`**) *expr*
Operators (man perlfunc)	**`do`** | **`eval`** | **`sub`**
	`continue` | **`goto`** | **`last`** | **`next`** | **`redo`** | **`return`**
	`die` | **`dump`** | **`exit`**

What is Truth

- Strings `""` and **`"0"`** are false, everything else is true
- File read **<…>** returns `""` at end-of-file
- Number **`0`** converted to string **`"0"`**
- Array in scalar context **`0`** iff empty
- Hash in scalar context **`"0"`** iff empty
- Reference `""` iff **`undef`**

Writing Subroutines

- Declaration
  - **`sub`** [*id*] [*proto*] [*attrs*] [**`{`**…**`}`**]
  - No *id*: anonymous; no *proto*: list operator; no block: declaration without definition
  - To return a value: **`return`** *expr* **`;`**
  - If no explicit return: value of last expression
- Arguments
  - Array **`@_`**
  - Call-by-reference (**`$_[`***i***`]`** is alias of actual *i*)
  - Common idiom: copy arguments into named locals, e.g., **`my ($a,$b) = @_;`**

Blocks as Function Arguments

- **`do{…}`**, **`eval{…}`**, **`sub{…}`** look like control statements, but aren't
  - They are functions with block argument
  - Other examples: **`grep`**, **`map`**, **`sort`**
  - You can write such functions yourself
- This is a lightweight form of callback
- SmallTalk language does same, only more
  - SmallTalk has no control statements at all
  - Method on boolean with block argument
  - **`(x<y) ifTrue:[mx:=y] ifFalse:[mx:=x]`**

Default Variable `$_`

- Only angle operator in `while` condition: read line from file handle into `$_`
- No variable in `foreach`: use `$_` as iteration variable
- No binding in pattern match: use `$_`
- No explicit operand for certain functions: use `$_`, e.g., `print`, `chop`, `chomp`, `split`, ...
- Don't confuse `$_` with `@_` (arguments)

Outline

- Perl Basics (continued)
- Regular Expressions

Example of `m//` Matching

Input

```
-rw-r--r--   1 soule  soule     3998 Aug 29 12:49 hw01.html
-rw-r--r--@  1 soule  soule     4480 Sep  5 14:10 hw02.html
-rw-------   1 soule  soule     8868 Sep  9 12:09 index.html
```

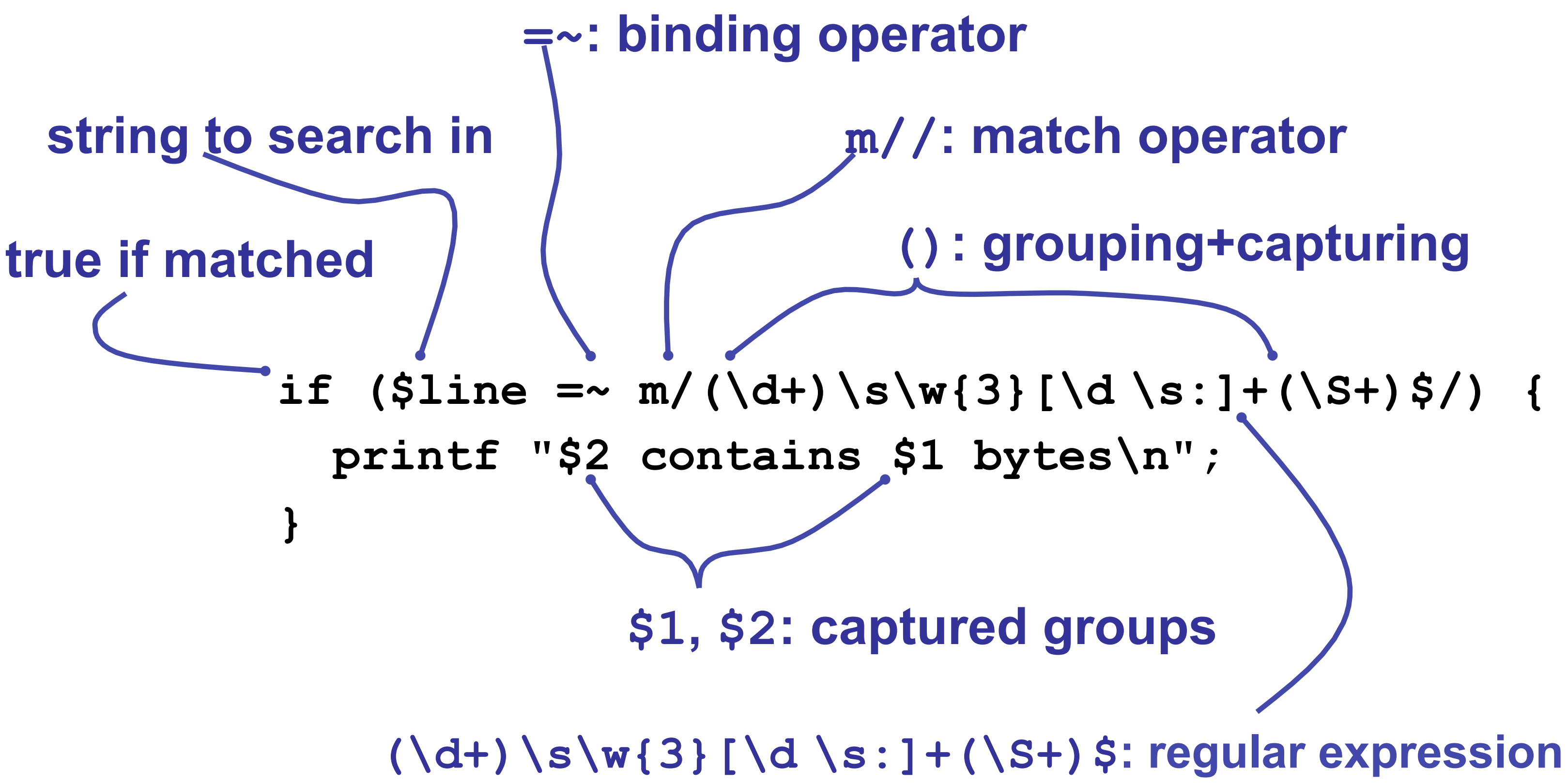
Script

```
#!/usr/bin/perl -w
while($line = <>) {
  if ($line =~ m/(\d+)\s\w{3}[\d\s:]+(\S+)$/) {
    printf "$2 contains $1 bytes\n";
  }
}
```

Output

```
hw01.html contains 3998 bytes
hw02.html contains 4480 bytes
index.html contains 8868 bytes
```

Explanation of `m//` Matching



```
if ($line =~ m/(\d+)\s\w{3}[\d \s:]+(\S+)$/) {
  printf "$2 contains $1 bytes\n";
}
```

Perl

Inside a Regular Expression

Kind	Construct	Syntax	Example	
			Pattern	Matches
Essentials	Character	Itself	**`b`**	**`b`**
	Concatenation	e_1e_2	**`bc`**	**`bc`**
	Alternative (or)	e_1 **|** e_2	**`a|bc`**	**`a`**, **`bc`**
	Repetition (≥0)	e**`*`**	**`a*`**	, **`a`**, **`aa`**, **`aaa`**
	Grouping	**`(`**e**`)`**	**`(a|b)c`**	**`ac`**, **`bc`**
Quantifier	Optional	e**`?`**	**`(a|b)?`**	, **`a`**, **`b`**
	Repetition (≥1)	e**`+`**	**`a+`**	**`a`**, **`aa`**, **`aaa`**
	Repetition	e**`{`**n**`}`**	**`a{3}`**	**`aaa`**
Char class	Custom class	**`[`**…**`]`**	**`[a-c]`**	**`a`**, **`b`**, **`c`**
	Wildcard character	**`.`**	**`.`**	**`a`**, **`3`**, **`:`**
	Shortcut class	**`\`**…	**`\d`**	**`0`**,**`1`**,**`2`**,…,**`9`**
Assertion	Zero-width anchor	**`$`**,**`^`**,**`\b`**,…	**`\b`**	(word boundary)

Explanation of `s///` substitution

string to search and modify

`=~`: binding operator

```
#!/usr/bin/perl -w
while($line = <>) {
  if ($line =~
      s/.+\s(\d+)\s\w{3}[\d\s:]+(\S+)$/$2 contains $1 bytes/) {
    print $line;
  }
}
```

`s///`: substitution operator

true if matched

`.+\s(\d+)\s\w{3}[\d \s:]+(\S+)$`: regular expression

`$2 contains $1 bytes`: replacement text

Outside a Regular Expression

- Match: `m//` or simply `//`; substitute: `s///`
- Binding: `=~` positive, `!~` negative
  - Or when missing: bound to `$_`
- Modifiers: e.g. `s///g` global substitute
- Return value: true if success, invert for `!~`
  - Or when assigned to list: groups, e.g.,

    ```
    my ($a,$b) = $s =~ /(\w+) (\d+)/;
    ```
- Output: `$&` entire match, `$1`, … groups
- Find out more: "man perlre"

Library Functions

- String: chomp, chop, length, substr, uc
- RegExp: m//, s///, split
- Math: abs, cos, exp, sin, sqrt
- Array: pop, push, shift, unshift
- Control: die, do, eval, last, next, sub
- Scoping: local, my, our, package, use
- Files: chdir, chmod, fork, printf, qx//
- And many more ("man perlfuc")

Perl Documentation

- Included: manpages perl, perldata, perlsyn, perlop, perlre, perlfunc, …
- Book: Programming Perl, 3rd edition. Larry Wall, Tom Christiansen, and Jon Orwant. O'Reilly, 2000 [safari].
- Online:
  - CPAN = Comprehensive Perl Archive Network, http://www.cpan.org
  - http://www.perl.com

Administrative

Last Slide

- Today's lecture
  - Associative arrays
  - Regular expressions
  - Basics of Perl
- Next lecture
  - Context
  - Typeglobs
  - Object-oriented Perl