CSCI-GA.3033.003 Scripting Languages

9/18/2013

Context and Modules (Perl)
Scripting as Glue

Web Server

- You will need to use this machine for your homework
- You must log in to create your account
- Do this now, so we can fix problems early.

Web Server

- Step 1: Install VPN:

http://www.it.cornell.edu/services/vpn/howto/install.cfm

- Step 2: ssh to machine:

```
$ ssh <netid>@en-cs-cs5142.coecis.cornell.edu
```

Step 3:

```
$ mkdir public_html
$ chmod 755 public_html
$ cd public_html
$ echo "Hello World" > index.html
$ chmod 604 index.html
```

Step 4: Go to URL:

http://en-cs-cs5142.coecis.cornell.edu/~<netid>/

Outline

- Text processing
- Type conversions

Idiomatic Perl

- Interpreter specification: `#!`
- Regular expressions extract data
  - Captured groups: `$1`, `$2`, …
- Hashes and arrays store data
  - Nested references build up data structures
- Default operand: `$_`
- Parameter array: `@_`
- String interpolations report results

Example from 8/28/2013, Revisited

```
#!/usr/bin/perl -w
%cup2g = ( flour => 110, sugar => 225, butter => 225 );
%volume = ( cup => 1, tbsp => 16, tsp => 48, ml => 236 );
%weight = ( lb => 1, oz => 16, g => 453 );
while (<>) {
  my ($qty, $unit, $ing) = /([0-9.]+) (\w+) (\w+)/;
  if ($cup2g{$ing} && $volume{$unit}) {
    $qty = 1.0 * $qty * $cup2g{$ing} / $volume{$unit};
    $unit = 'g';
  } elsif ($volume{$unit}) {
    $qty = 1.0 * $qty * $volume{ml} / $volume{$unit};
    $unit = 'ml';
  } elsif ($weight{$unit}) {
    $qty = 1.0 * $qty * $weight{g} / $weight{$unit};
    $unit = 'g';
  }
  printf("%d $unit $ing\n", $qty + .5);
}
```

Soap-box

Readable Perl

- Use pragmata: **`warnings`**, **`strict`**, maybe also **`diagnostics`**, **`sigtrap`**
- Avoid default **`$_`**
- Name your subroutine parameters
- Use **`my`**, avoid **`local`**
- Use **`/x`** modifier on regular expressions
  - Add whitespace, line breaks, comments
- Use modules for large projects

Chomsky Hierarchy

Type	Languages	Automata	Rules
0	Recursively enumerable	Turing machine	$\alpha \rightarrow \beta$
1	Context sensitive	Linear bounded Turing machine	$\alpha B \gamma \rightarrow \alpha \delta \gamma$
2	Context free	Pushdown automaton	$A \rightarrow \beta$ (BNF)
3	Regular	Finite state machine	$A \rightarrow b,\ C \rightarrow dE$ (regexp)

- Formal regexp only has “essentials”
- Perl adds shortcuts and non-regular features

Perl

Inside a Regular Expression

Kind	Construct	Syntax	Example	
			Pattern	Matches
Essentials	Character	Itself	`b`	`b`
	Concatenation	$e_1 e_2$	`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)

Non-Regular Perl Regex Features

- Non-regular = not essential feature, and can not be emulated by essential features
  - Backtracking engine, may take exponential time
- Backreferences in same pattern: `\1`, `\2`, …
- Zero-width assertions:
  - Look-ahead positive `(?=…)`, negative `(?!…)`
  - Look-behind positive `(?<=…)`, negative `(?<!…)`
- Match-time code execution: `(?{…})`
- Match-time interpolation: `(??{…})`
- Backtracking suppression: `(?>…)`

Non-Regular Perl Regex Examples

Description	Regex	Matched
Square (repeated)	`(\d+)x\1`	`123x123`
Palindrome (mirrored)	`(\w+)\w(??{reverse $1})`	`redivider`
Balanced parentheses	`(<+)x(??{'>' x length $1})`	`<<<x>>>`

Outline

- Text processing
- Type conversions

Natural and Artificial Languages

English		Perl	
Concept	Examples	Feature	Examples
Noun	fish	Variable	`$line`
Pronoun	it, they	Default variable	`$_`, `@_`
Inflection	...s	Sigil	`@`...
Singular	apple	Scalar	`$i`
Plural	oranges	Array	`@row`
Verb	cook	Function	`sort`
Irregular verb	read	Operator	`<...>`
Infinitive	to be	Function reference	`&cmp`
Context	go fish	Context	`if(@a)`...

Context

- When you use a value in a different context than the value's type, Perl converts it
  - E.g., using array as scalar -> int length
- Context provided by:
  - Operator: e.g., *string* **.** *string*
  - Function: e.g., **print** *list*
  - Assignment: e.g., **@a =** *list*
  - Statement: e.g., **if (***boolean***) {***...***}**
- Functions provide context for parameters, and can react to context for return value

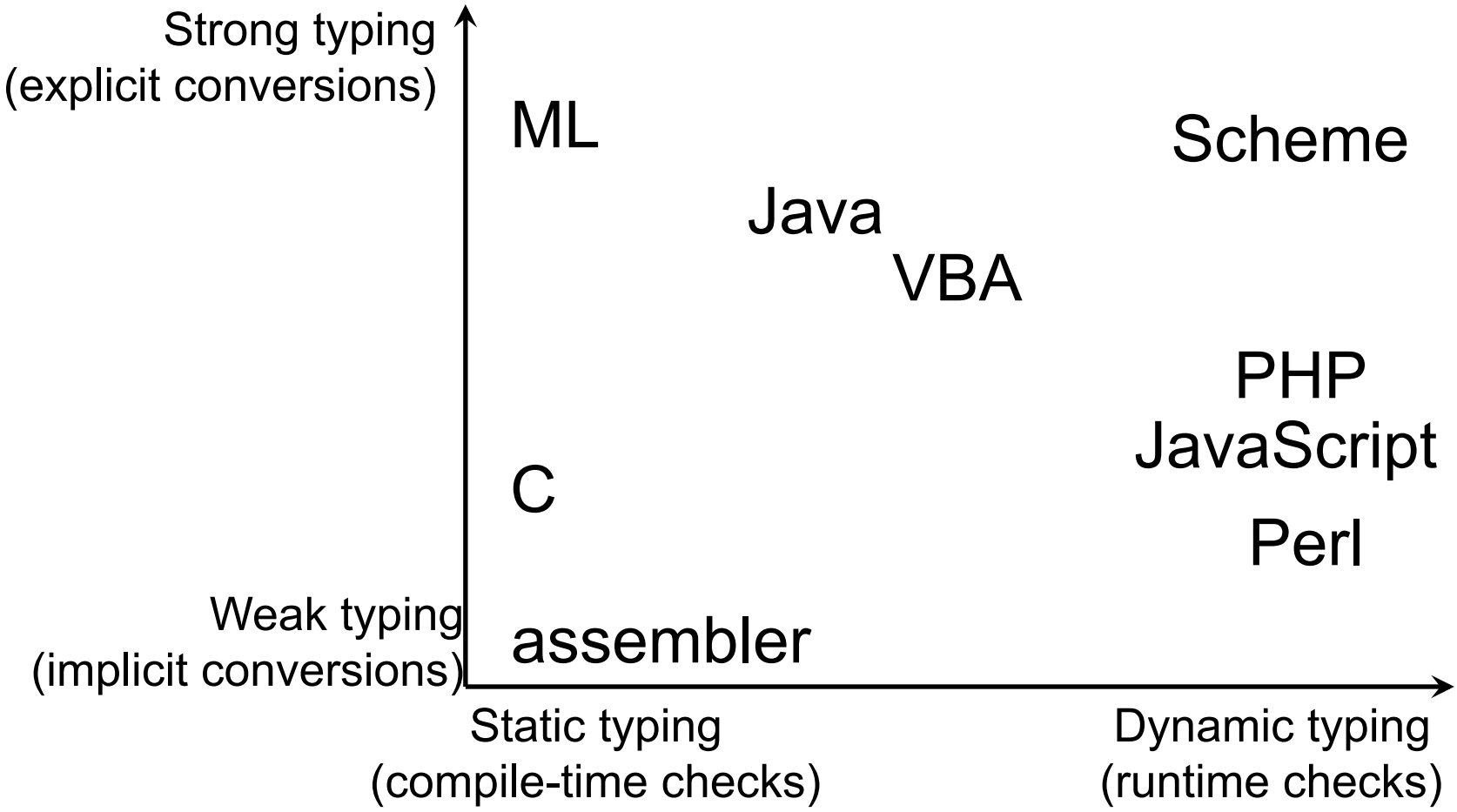
Primitive Conversions

Value		Context			
		Boolean	Number	String	Reference
Number	0	false	identity	""	undefined
	!=0	true		"value"	symbolic reference
String	""	false	0	identity	
	"0"	false	0		
	Other	true	prefix		
Undefined		false	0	""	undefined
Reference		true	addr	"*typ*(*adr*)"	identity

List Conversions

Value		Context	
		Scalar	List
Scalar		identity	1-element list
List literal		last element	flattened sublists
Array		length	identity
Hash	empty	`"0"`	empty list
	non-empty	"*size*/*buckets*"	k_1, v_1, k_2, v_2,...
`m//`		true iff match	all groups
<...>	end of file	`""`	empty array
	not EOF	line	all lines

Weak/Strong, Static/Dynamic Typing



Functions and Context

- Prototype provides context for parameters
  - **`sub`** [*id*] [*proto*] [*attrs*] [**`{`**...**`}`**]
  - No prototype: list operator
  - *proto* = **(***protoChar****)**
  - **`$`** scalar, **`@`** list, **`%`** hash, **`*`** file handle
  - **`&`** subroutine (in first slot: bare block, no comma)
  - **`;`** separates mandatory from optional arguments
  - **`\`** adds backslashes on actuals
- **`wantarray`** checks context for return value
  - true = list, false = array, undefined = void context

Prototype Example

- Example from page 227 of “Programming Perl”, 3rd ed. Wall, Christiansen, Orwant. O’Reilly, 2000.
- See also the Error module on CPAN.

Function definitions

```
sub try(&$) {
  my ($try, $catch) = @_;
  eval { &$try };
  if ($@) {
    local $_ = $@;
    &$catch;
  }
}
sub catch (&) { $_[0] }
```

Example usage

```
try {
  die "phooey";
} #not end of the call!
catch {
  /phooey/ and
    print "unphooey\n";
};
```

Outline

- Regular expressions (continued)
- Type conversions

Administrative

Last Slide

- Prelim 1 next week
- Today's lecture
  - Text processing
  - Context
- Next lecture
  - Modules
  - Object-oriented Perl

Anonymous functions

- Function definition can omit name
  `sub` [*id*] [*proto*] [*attrs*] [**`{`**...**`}`**]
- Example script:

Higher-order function: has function param or returns function result

```
#!/usr/bin/env perl
use warnings;
use strict;
sub callFn { my ($fn, $x) = @_; &$fn($x); }
sub printIt { my ($it) = @_; print "printIt $it\n"; }
#pass reference to named subroutine; prints "printIt Hello"
callFn(\&printIt, "Hello");
#pass reference to anonymous subroutine; prints "lambda Hi"
callFn(sub { my ($it) = @_; print "lambda $it\n" }, "Hi");
```

- Useful for call-backs, e.g., sort/compare