

CMSC 330: Organization of Programming Languages

Ruby Regular Expressions
and other topics

Reminders

- If you have questions about projects or homework, please use the online forum
- No bragging about project progress

Review

- formal parameters vs. actual parameters
- control statement (definition and examples)
- deep vs. shallow copy
- deep vs. shallow equality

CMSC 330

3

Standard Library: String

- `"hello".index("l", 0)`
 - Return index of the first occurrence of string "l" in "hello", starting at 0
- `"hello".sub("h", "j")`
 - Replace first occurrence of "h" by "j" in string (not permanent)
 - Use `gsub` ("global" sub) to replace all occurrences
- `"r1\tr2\ttr3".split("\t")`
 - Return array of substrings delimited by tab
- Consider these three examples again
 - All involve *searching* in a string for a certain pattern
 - What if we want to find more complicated patterns?
 - Find first occurrence of "a" or "b"
 - Split string at tabs, spaces, and newlines

CMSC 330

4

Regular Expressions

- A way of describing patterns or sets of strings
 - Searching and matching
 - Formally describing strings
 - The symbols (lexemes or tokens) that make up a language
- Common to lots of languages and tools
 - awk, sed, perl, grep, Java, OCaml, C libraries, etc.
- Based on some really elegant theory
 - We'll see that soon

CMS 330

5

Example Regular Expressions in Ruby

- `/Ruby/`
 - Matches exactly the string "Ruby"
 - Regular expressions can be delimited by `/`'s
 - Use `\` to escape `/`'s in regular expressions
- `/(Ruby|OCaml|Java)/`
 - Matches either "Ruby", "OCaml", or "Java"
- `/(Ruby|Regular)/` or `/R(uby|egular)/`
 - Matches either "Ruby" or "Regular"
 - Use `()`'s for grouping; use `\` to escape `()`'s

CMS 330

6

Using Regular Expressions

- Regular expressions are instances of `Regexp`
 - we'll see use of a `Regexp.new` later
- Basic matching using `=~` method of `String`

```
line = gets          # read line from standard input
if line =~ /Ruby/ then # returns nil if not found
  puts "Found Ruby"
end
```

- Can use regular expressions in index, search, etc.

```
offset = line.index(/(MAX|MIN)/) # search starting from 0
line.sub(/(Perl|Python)/, "Ruby") # replace
line.split(/(\t|\n| )/)          # split at tab, space,
                                # newline
```

CMSC 330

7

Using Regular Expressions (cont'd)

- Invert matching using `!~` method of `String`
 - Matches strings that *don't* contain an instance of the regular expression

CMSC 330

8

Repetition in Regular Expressions

- `/(Ruby)*/`
 - { "", "Ruby", "RubyRuby", "RubyRubyRuby", ... }
 - `*` means *zero or more occurrences*
- `/Ruby+/`
 - { "Ruby", "Rubyy", "Rubyyy", ... }
 - `+` means *one or more occurrence*
 - so `/e+/` is the same as `/ee*/`

CMSC 330

9

Repetition in Regular Expressions

- `/(Ruby)?/`
 - { "", "Ruby" }
 - `?` means *optional*, i.e., zero or one occurrence
- `/(Ruby){3}/`
 - { "RubyRubyRuby", "RubyRubyRubyRuby", ... }
 - `{x}` means repeat the search for at least x occurrences
- `/(Ruby){3, 5}/`
 - { "RubyRubyRuby", "RubyRubyRubyRuby", "RubyRubyRubyRubyRuby" }
 - `{x, y}` means repeat the search for at least x occurrences and at most y occurrences

CMSC 330

10

Watch Out for Precedence

- `/(Ruby)*/` means `{ "", "Ruby", "RubyRuby", ... }`
 - But `/Ruby*/` matches `{ "Rub", "Ruby", "Rubyy", ... }`
- In general
 - `* {x}` and `+` bind most tightly
 - Then concatenation (adjacency of regular expressions)
 - Then `|`
- Best to use parentheses to disambiguate

CMSC 330

11

Character Classes

- `/[abcd]/`
 - `{ "a", "b", "c", "d" }` (Can you write this another way?)
- `/[a-zA-Z0-9]/`
 - Any upper or lower case letter or digit
- `/[^0-9]/`
 - Any character except 0-9 (the `^` is like not and must come first)
- `/[\t\n]/`
 - Tab, newline or space
- `/[a-zA-Z_\\$][a-zA-Z_\\$0-9]*/`
 - Java identifiers (`$` escaped...see next slide)

CMSC 330

12

Special Characters

| | |
|-----|------------------------------|
| . | any character |
| ^ | beginning of line |
| \$ | end of line |
| \\$ | just a \$ |
| \d | digit, [0-9] |
| \s | whitespace, [\t\r\n\f] |
| \w | word character, [A-Za-z0-9_] |
| \D | non-digit, [^0-9] |
| \S | non-space, [^\t\r\n\f] |
| \W | non-word, [^A-Za-z0-9_] |

CMSC 330

13

Potential Character Class Confusions

- ^ inside character classes: not
outside character classes: beginning of line
- [] inside regular expressions: character class
outside regular expressions: array
note: [a-z] does not make a valid array,
/(0..2)/ does not mean 012
- () inside character classes: literal characters ()
outside character classes: used for grouping
- inside character classes: range (ex: a to z given by [a-z])
outside character classes: subtraction

CMSC 330

14

Regular Expression Practice

With a neighbor, make regular expressions representing the following ideas:

- All lines beginning with **a** or **b** `/^(a|b)/`
- All lines containing at least two (only alphabetic) words separated by white-space `/[a-zA-Z]+\s+[a-zA-Z]+/`
- All lines where **a** and **b** alternate and appear at least once `/^((ab)+a?)|(ba)+b?)$/`
- An expression which would match both of these lines (but not radically different ones):

CMSC330: Organization of Programming Languages: Fall 2007
CMSC351: Algorithms: Fall 2007

CMSC 330

15

Regular Expression Coding Readability

```
> ls -l
drwx-----  2 sorelle  sorelle  4096 Feb 18 18:05 bin
-rw-----  1 sorelle  sorelle   674 Jun  1 15:27 calendar
drwx-----  3 sorelle  sorelle  4096 May 11 12:19 cmesc311
drwx-----  2 sorelle  sorelle  4096 Jun  4 17:31 cmesc330
drwx-----  1 sorelle  sorelle  4096 May 30 19:19 cmesc630
drwx-----  1 sorelle  sorelle  4096 May 30 19:20 cmesc631
```

What if we want to specify the format of this line exactly?

```
/^(d|)(r|)(w|)(x|)(r|)(w|)(x|)(r|)(w|)(x|)
(\s+)(\d+)(\s+)(\w+)(\s+)(\w+)(\s+)(\d+)(\s+)(Jan|Feb
|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)(\s+)(\d\d)
(\s+)(\d\d:\d\d)(\s+)(\S+)$/
```

This is unreadable!

CMSC 330

16

Regular Expression Coding Readability

Instead, we can do each part of the expression separately and then combine them:

```
oneperm_re = '((r|-)(w|-)(x|-))'
permissions_re = '(d|-)' + oneperm_re + '{3}'
month_re = '(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)'
day_re = '\d{1,2}'; time_re = '(\d{2}:\d{2})'
date_re = month_re + '\s+' + day_re + '\s+' + time_re
total_re = '\d+'; user_re = '\w+'; group_re = '\w+'
space_re = '\d+'; filename_re = '\S+'

line_re = Regexp.new('^' + permissions_re + '\s+' + total_re
  + '\s+' + user_re + '\s+' + group_re + '\s+' +
  space_re + '\s+' + date_re + '\s+' + filename_re + '$')

if line =~ line_re
  puts "found it!"
end
```

CMSC 330

17

Method 1: Back-references

Two options to extract substrings based on R.E.'s:

- Use *back-references*
 - Ruby remembers which strings matched the parenthesized parts of r.e.'s
 - These parts can be referred to using special variables called back-references (named \$1, \$2,...)

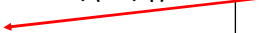
CMSC 330

18

Back-reference Example

- Extract information from a report

```
gets =~ /^Min: (\d+) Max: (\d+)$/
min, max = $1, $2
```



```
sets min = $1
and max = $2
```

- Warning:** Despite their names, \$1 etc are *local* variables

```
def m(s)
  s =~ /(Foo)/
  puts $1 # prints Foo
end
m("Foo")
puts $1 # prints nil
```

CMSC 330

19

Another Back-reference Example

- Warning 2: If another search is done, all back-references are reset to nil

| | |
|---|---|
| <pre>gets =~ /(h)e(ll)o/ puts \$1 puts \$2 gets =~ /h(e)llo/ puts \$1 puts \$2 gets =~ /hello/ puts \$1</pre> | <pre>hello h ll hello e nil hello nil</pre> |
|---|---|

CMSC 330

20

Method 2: String.scan

- Also extracts substrings based on regular expressions
- Can optionally use parentheses in regular expression to affect how the extraction is done
- Has two forms which differ in what Ruby does with the matched substrings
 - The first form returns an array
 - The second form uses a code block
 - We'll see this later

CMSC 330

21

First Form of the scan Method

- `str.scan(regex)`
 - If `regex` doesn't contain any parenthesized subparts, returns an array of matches
 - An array of all the substrings of `str` which matched
- ```
s = "CMSC 330 Fall 2007"
s.scan(/\S+ \S+/)
returns array ["CMSC 330", "Fall 2007"]
```
- Note: these string are chosen sequentially from as yet unmatched portions of the string, so while “330 Fall” *does* match the regular expression above, it is *not* returned since “330” has already been matched by a previous substring.

CMSC 330

22

## First Form of the scan Method... part 2

- If *regexp* contains parenthesized subparts, returns an array of arrays

- Each sub-array contains the parts of the string which matched one occurrence of the search

```
s = "CMSC 330 Fall 2007"
s.scan(/(\S+) (\S+)/) # [{"CMSC", "330"},
 # ["Fall", "2007"]]
```

- Each sub-array has the same number of entries as the number of parenthesized subparts
- All strings that matched the first part of the search (or \$1 in back-reference terms) are located in the first position of each sub-array

CMSC 330

23

## Practice with scan and back-references

```
> ls -l
drwx----- 2 sorelle sorelle 4096 Feb 18 18:05 bin
-rw----- 1 sorelle sorelle 674 Jun 1 15:27 calendar
drwx----- 3 sorelle sorelle 4096 May 11 2006 cmisc311
drwx----- 2 sorelle sorelle 4096 Jun 4 17:31 cmisc330
drwx----- 1 sorelle sorelle 4096 May 30 19:19 cmisc630
drwx----- 1 sorelle sorelle 4096 May 30 19:20 cmisc631
```

Extract just the file or directory name from a line using

- scan

```
name = line.scan(/(\S+$/)) # ["bin"]
```

- back-references

```
if line =~ /(\S+$/)/
 name = $1 # "bin"
end
```

CMSC 330

24

## Standard Library: Array

- Arrays of objects are instances of class `Array`
  - Arrays may be heterogeneous  
`a = [1, "foo", 2.14]`
  - C-like syntax for accessing elements, indexed from 0  
`x = a[0]; a[1] = 37`
- Arrays are *growable*
  - Increase in size automatically as you access elements  
`irb(main):001:0> b = []; b[0] = 0; b[5] = 0; puts b.inspect`  
`[0, nil, nil, nil, nil, 0]`
  - `[]` is the empty array, same as `Array.new`

CMSC 330

25

## Standard Library: Arrays (cont'd)

- Arrays can also shrink
  - Contents shift left when you delete elements  
`a = [1, 2, 3, 4, 5]`  
`a.delete_at(3)`            `# delete at position 3; a = [1,2,3,5]`  
`a.delete(2)`            `# delete element = 2; a = [1,3,5]`
- Can use arrays to model stacks and queues  
`a = [1, 2, 3]`  
`a.push("a")`    `# a = [1, 2, 3, "a"]`  
`x = a.pop`      `# x = "a"`  
`a.unshift("b")` `# a = ["b", 1, 2, 3]`  
`y = a.shift`    `# y = "b"`

**note:** push, pop, shift, and unshift all permanently modify the array

CMSC 330

26

## Iterating through Arrays

- It's easy to iterate over an array with **while**

```
a = [1,2,3,4,5]
i = 0
while i < a.length
 puts a[i]
 i = i + 1
end
```

- Looping through all elements of an array is very common
  - And there's a better way to do it in Ruby

CMSC 330

27

## Iteration and Code Blocks

- The **Array** class also has an **each** method, which takes a code block as an argument

```
a = [1,2,3,4,5]
a.each { |x| puts x }
```

code block delimited by  
{}'s or do...end

parameter name

body

CMSC 330

28

## More Examples of Code Blocks

- Sum up the elements of an array

```
a = [1,2,3,4,5]
sum = 0
a.each { |x| sum = sum + x }
printf("sum is %d\n", sum)
```

- Print out each segment of the string as divided up by commas (commas are printed trailing each segment)
  - Can use any delimiter

```
s = "Student,Sally,099112233,A"
s.each(',') { |x| puts x }
```

("delimiter" = symbol used to denote boundaries)

CMS 330

29

## Yet More Examples of Code Blocks

```
3.times { puts "hello"; puts "goodbye" }
5.upto(10) { |x| puts(x + 1) }
[1,2,3,4,5].find { |y| y % 2 == 0 }
[5,4,3].collect { |x| -x }
```

- `n.times` runs code block `n` times
- `n.upto(m)` runs code block for integers `n..m`
- `a.find` returns first element `x` of array such that the block returns true for `x`
- `a.collect` applies block to each element of array and returns new array (`a.collect!` modifies the original)

CMS 330

30

## Still Another Example of Code Blocks

```
File.open("test.txt", "r") do |f|
 f.readlines.each { |line| puts line }
end
```

- `open` method takes code block with file argument
  - File automatically closed after block executed
- `readlines` reads all lines from a file and returns an array of the lines read
  - Use `each` to iterate

CMSC 330

31

## Using Yield to Call Code Blocks

- Any method can be called with a code block. Inside the method, the block is called with `yield`.
- After the code block completes, control returns to the caller after the `yield` instruction.

```
def countx(x)
 for i in (1..x)
 puts i
 yield
 end
end

countx(4) { puts "foo" }
```

```
1
foo
2
foo
3
foo
4
foo
```

CMSC 330

32



## So What are Code Blocks?

---

- A code block is just a special kind of method
  - `{ |y| x = y + 1; puts x }` is almost the same as
  - `def m(y) x = y + 1; puts x end`
- The `each` method takes a code block as an argument
  - This is called *higher-order programming*
    - In other words, methods take other methods as arguments
    - We'll see a lot more of this in OCaml
- We'll see other library classes with `each` methods
  - And other methods that take code blocks as arguments
  - As we saw, your methods can use code blocks too!

CMSC 330

33

## Second Form of the scan Method

---

- Remember the scan method?
  - Gave back an array of matches
  - Can also take a code block as an argument
- `str.scan(regex) { |match| block }`
  - Applies the code block to each match
  - Short for `str.scan(regex).each { |match| block }`
  - The regular expression can also contain parenthesized subparts

CMSC 330

34

## Example of Second Form of scan

```
12 34 23
19 77 87
11 98 3
2 45 0
```

input file:  
will be read line by line, but  
column summation is desired

```
sum_a = sum_b = sum_c = 0
while (line = gets)
 line.scan(/(\d+)\s+(\d+)\s+(\d+)/) { |a,b,c|
 sum_a += a.to_i
 sum_b += b.to_i
 sum_c += c.to_i
 }
end
printf("Total: %d %d %d\n", sum_a, sum_b, sum_c)
```

converts the string  
to an integer

Sums up three columns of numbers

CMSC 330

35

## Standard Library: Hash

- A hash acts like an associative array
  - Elements can be indexed by any kind of values
  - Every Ruby object can be used as a hash key, because the `Object` class has a `hash` method
- Elements are referred to using `[]` like array elements, but `Hash.new` is the `Hash` constructor

```
italy["population"] = 58103033
italy["continent"] = "europe"
italy[1861] = "independence"
```

CMSC 330

36

## Hash (cont'd)

---

- The `Hash` method `values` returns an array of a hash's values (in some order)
- And `keys` returns an array of a hash's keys (in some order)
- Iterating over a hash:

```
italy.keys.each {
 |key| puts("key: #{key}, value: #{italy[key]}")
}
```

CMSC 330

37

## Hash (cont'd)

---

Convenient syntax for creating literal hashes

- Use `{ key => value, ... }` to create hash table

```
credits = {
 "cmisc131" => 4,
 "cmisc330" => 3,
}

x = credits["cmisc330"] # x now 3
credits["cmisc311"] = 3
```

CMSC 330

38

## Standard Library: File

- Lots of convenient methods for IO

```
File.new("file.txt", "rw") # open for rw access
f.readline # reads the next line from a file
f.readlines # returns an array of all file lines
f.eof # return true if at end of file
f.close # close file
f << object # convert object to string and write to f
$stdin, $stdout, $stderr # global variables for standard UNIX IO
 By default stdin reads from keyboard, and stdout and stderr both
 write to terminal
```

- **File** inherits some of these methods from **IO**

CMSC 330

39

## Exceptions

- Use **begin...rescue...ensure...end**
  - Like try...catch...finally in Java

```
begin
 f = File.open("test.txt", "r")
 while !f.eof
 line = f.readline
 puts line
 end
rescue Exception => e
 puts "Exception:" + e.to_s +
 " (class " + e.class.to_s + ")"
ensure
 f.close
end
```

Class of exception  
to catch

Local name  
for exception

Always happens

CMSC 330

40

## Practice: Amino Acid counting in DNA

Write a function that will take a filename and read through that file counting the number of times each group of three letters appears so these numbers can be accessed from a hash.

(assume: the number of chars per line is a multiple of 3)

```
cgggcattcagcaccggtatactgttaagcaatccagatTTTTgtgtataacataccggc
catactgaagcattcattgaggctagcgctgataacagtagcgctaacaatgggggaatg
tggcaatacgggtgcgattactaagagccgggaccacacccccgtaaggatggagcgtgg
taacataataatccggttcaagcagtgggcgaaggtggagatgtccagtaagaatagtgg
gggcctactacccatggtacataattaagagatcgtcaatcttgagacgggtcaatggtac
cgagactatatcactcaactccggacgtatgcgcttactgggtcacctcgttactgacgga
```

CMS3 330

41

## Practice: Amino Acid counting in DNA

get the  
file  
handle

array of  
lines  
from the  
file

for each  
line in  
the file

for each  
triplet  
in the  
line

```
def countaa(filename)
 file = File.new(filename, "r")
 arr = file.readlines
 hash = Hash.new
 arr.each{ |line|
 acids = line.scan(/.../)
 acids.each{ |aa|
 if hash[aa] == nil
 hash[aa] = 1
 else
 hash[aa] += 1
 end
 }
 }
end
```

initialize  
the hash, or  
you will get  
an error  
about trying  
to index into  
an array with  
a string

get an array  
of triplets  
in the line

CMS3 330

42