CMSC 330: Organization of Programming Languages

Ruby Hash Tables
(and other topics)
Standard Library: Hash

• A Hash acts like an associative array or dictionary lookup
  – 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"
Hash (cont’d)

Convenient syntax for creating literal hashes
– Use \{ key => value, ... \} to create hash table

```ruby
credits = {
  "cmsc131" => 4,
  "cmsc330" => 3,
}

x = credits["cmsc330"]  # x now 3
credits["cmsc311"] = 3
```
Hash (cont’d)

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

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

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

- Array and Hash structures can be sorted:
  \[7, 1, 5].sort \Rightarrow [1, 5, 7]\]

- Sorting a Hash returns a nested Array, sorted by key:
  \{"bob"=>4, "sue"=>2, "bill"=>7\}.sort
  \Rightarrow [["bill", 7], ["bob", 4], ["sue", 2]]

- To sort by value, use sort_by and a code block:
  people = {"bob"=>4,"sue"=>2,"bill"=>7}
  people.sort_by { |key,value| value } 
  \Rightarrow [["sue", 2], ["bob", 4], ["bill", 7]]
Standard Library: File

- **Lots of convenient methods for IO**
  
  ```ruby
  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**
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
```
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)

gcggcattcagcacccgtataactgttaagcaatccagatttttgtgtataacatcaacctggc
catactgaagcatttcttgaggctagcgtgataacagtagcgcgttaacaatggggaatg
tggcaatacggtgctgattaactaagagagcggggaccacacaccccgttaaggatggagcgtg
taatcattaatcgggttcacaagcagtgggctggaggagatgtccagtaagaatagttg
gggcctactacccatgttgtacataattaagagatcgctcaatctttagagacggtcaatggtac
cgagactatatatcactaactccggacgtaatggcgcttacttggtcactcgttactgacgga
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 when trying to index into an array with a string
get the file handle
array of lines from the file
for each line in the file
for each triplet in the line
get an array of triplets in the line
Global Variables in Ruby

- Ruby has two kinds of global variables
  - Class variables beginning with @@
  - Global variables across classes beginning with $

```ruby
class Global
  @@x = 0

  def Global.inc
    @@x = @@x + 1; $x = $x + 1
  end

  def Global.get
    return @@x
  end
end
```

```ruby
$x = 0
Global.inc
$x = $x + 1
Global.inc
puts(Global.get)
puts($x)
```

define a class ("singleton") method
Special Global Variables

• Ruby has a bunch of global variables that are implicitly set by methods

• The most insidious one: \$_
  – Default method return, argument in many cases
  – Can lead to shorter programs, but more confusion

```ruby
gets    # implicitly reads input into $
print   # implicitly writes $
```

• More useful ones: $1, $2, $3, etc.
  – Matches from regular expressions (=~ or !~)

• $0 is the name of the Ruby script
CMSC 330: Organization of Programming Languages

Ruby Objects and Inheritance
Classes and Objects

• Class names begin with an uppercase letter
• The “new” method creates an object
  – \( s = \text{String}.\text{new} \) creates a new \text{String} and makes \( s \) refer to it
• Every class inherits from \text{Object}
Everything is an Object

• In Ruby, everything is in fact an object
  – (-4).abs
    • integers are instances of Fixnum
  – 3 + 4
    • infix notation for “invoke the + method of 3 on argument 4”
  – "programming".length
    • strings are instances of String
  – String.new
    • classes are objects with a new method
  – (4.13).class
    • use the class method to get the class for an object
      • floating point numbers are instances of Float
Objects and Classes

• Objects are data
• Classes are types of data
• But in Ruby, classes themselves are objects!

<table>
<thead>
<tr>
<th>Object</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Fixnum</td>
</tr>
<tr>
<td>-3.30</td>
<td>Float</td>
</tr>
<tr>
<td>“CMSC 330”</td>
<td>String</td>
</tr>
<tr>
<td>String.new</td>
<td>String</td>
</tr>
<tr>
<td>Fixnum</td>
<td>Class</td>
</tr>
<tr>
<td>String</td>
<td>Class</td>
</tr>
</tbody>
</table>

• Fixnum, Float, String, etc., (including Class), are objects of type Class
Two Cool Things to Do with Classes

• Since classes are objects, you can manipulate them however you like

```ruby
if p then x = String else x = Time end  # Time is
...  # another class
y = x.new  # creates a String or a Time,
# depending upon p
```

• You can get names of all the methods of a class
  – Object.methods
    • => ["send", "name", "class_eval", "object_id", "new",
    "autoload?", "singleton_methods", ... ]
The nil Object

• Ruby uses a special object nil
  – All uninitialized fields set to nil (@ refers to a class field)
    ```ruby
    irb(main):004:0> @x
    => nil
    ```
  – Like NULL or 0 in C/C++ and null in Java
• nil is an object of class NilClass
  – It’s a singleton object – there is only one instance of it
    • NilClass does not have a new method
  – nil has methods like to_s, but not other methods that don’t make sense
    ```ruby
    irb(main):006:0> @x + 2
    NoMethodError: undefined method `+' for nil:NilClass
    ```
What is a Program?

• In C/C++, a program is...
  – A collection of declarations and definitions
  – With a distinguished function definition
    • int main(int argc, char *argv[]) { ... }
  – When you run a C/C++ program, it’s like the OS calls main(...)

• In Java, a program is...
  – A collection of class definitions
  – With a class Cl that contains a method
    • public static void main(String[] args)
  – When you run java Cl, the main method of class Cl is invoked
A Ruby Program is...

- The class **Object**
  - When the class is loaded, any expressions not in method bodies are executed

```ruby
def sayN(message, n)
i = 0
  while i < n
    puts message
    i = i + 1
  end
  return i
end

x = sayN("hello", 3)
puts(x)
```

invokes self.sayN
Ruby is Dynamically Typed

- Recall we don’t declare types of variables
  - But Ruby does keep track of types at run time
    
    ```ruby
    x = 3; x.foo
    NoMethodError: undefined method 'foo' for 3:Fixnum
    ```

- We say that Ruby is *dynamically typed*
  - Types are determined and checked at run time

- Compare to C, which is *statically typed*

```ruby
# Ruby
x = 3
x = "foo"  # gives x a new type

/* C */
int x;
x = 3;
x = "foo"; /* not allowed */
```
Types in Java and C++

• Are Java and C++ statically or dynamically typed?
  – A little of both
  – Many things are checked statically
    ```java
    Object x = new Object();
    x.println("hello"); // No such method error at compile time
    ```
  – But other things are checked dynamically
    ```java
    Object o = new Object();
    String s = (String) o; // No compiler warning, fails at runtime
    // (Some Java compilers may be smart enough to warn about above cast)
    ```
Tradeoffs: Static vs. Dynamic

- Ease of coding
- Error-proneness
- Usefulness of type checking
Classes and Objects in Ruby

class Point
  def initialize(x, y)
    @x = x
    @y = y
  end
  
  def addX(x)
    @x += x
  end
  
  def to_s
    return "(" + @x.to_s + "," + @y.to_s + ")"
  end
end

p = Point.new(3, 4)
p.addX(4)
puts(p.to_s)
Notes

• Parentheses are optional for method calls
• Ruby does not support method overloading
  – There can only be one initialize method
  – The last initialize method defined is used
• Recall classes begin with an uppercase letter
• `inspect` converts *any* instance to a string
  ```ruby
  irb(main):033:0> p.inspect
  => "#<Point:0x54574 @y=4, @x=7>"
  ```
• Instance variables are prefixed with `@`
  – Compare to local variables with no prefix
  – *Cannot be accessed outside of class*
• The `to_s` method can be invoked implicitly
  – Could have written `puts(p)`
    • Like Java’s `toString()` methods
Inheritance

• Recall that every class inherits from Object

class A
  def plusplus(x)
    return x + 1
  end
end

class B < A
  def plusplus(y)
    return (super(y) + 1)
  end
end

b = B.new
puts(b.plusplus(3))
super() in Ruby

• Within the body of a method, a call to super acts just like a call to that original method, except that the search for a method body starts in the superclass.
Inheritance (cont’d)

• Ruby uses **single inheritance**; i.e. each class may only have a single parent:

```
class A < B  # (B is the parent of A)
A.superclass  # (returns B)
```
**Inheritance (cont’d)**

- Ruby supports *mixins* and *modules*, which are similar to Java interfaces (modules cannot be instantiated):

```ruby
module M1
  def f1
    end
end

module M2
  def f2
    end
end

class A
  include M1
  include M2
end
```

- Modules can be mixed in to particular object instances during execution:

```ruby
obj = “test”
obj.extend M1
```

- Class A will have both f1 and f2
- obj will have f1
Single vs. Multiple Inheritance

• Single inheritance: one parent per class
• Multiple inheritance: unlimited parents per class

• “Diamond problem”
  – If B and C both override method foo() in A, which version is called by D?
  – In what order are constructors called?

• Three ways to fix:
  – Explicit namespaceing
  – Linear prioritization
  – Non-instantiation of superclasses