CMSC 330: Organization of Programming Languages

Introduction to Ruby

Introduction

- Ruby is an object-oriented, imperative scripting language
  - “I wanted a scripting language that was more powerful than Perl, and more object-oriented than Python. That’s why I decided to design my own language.”
  - “I believe people want to express themselves when they program. They don’t want to fight with the language. Programming languages must feel natural to programmers. I tried to make people enjoy programming and concentrate on the fun and creative part of programming when they use Ruby.”
    — Yukihito Matsumoto (“Matz”)

Books on Ruby

- Earlier version of Thomas book available on web
  - See course web page

Applications of Scripting Languages

- Scripting languages have many uses
  - Automating system administration
  - Automating user tasks
  - Quick-and-dirty development

- Major application
  - Text processing

Output from Command-Line Tool

% wc *
271  474    5323 AST.c
100  392    3219 AST.h
117 1458  226768 AST.o
1874 5463    47641 AST_defs.c
1375 6317    52667 AST_defs.h
375  844    49631 AST_parent.c
810 3136    34569 AST_print.c
640 1070    30530 AST_types.h
590  844     7041 AST_util.c
98  274    2154 AST_util.h
58  416    26754 AST_util.o
844 2755    25873 Makefile
270 725    5578 Makefile.am
844 2743    27200 Makefile.in
38  175    1154 sitran.s
2035 4514    47751 alias_types.c
86  300    3204 alias_types.h
104 1011    68648 alias_types.o

...
Climate Data for IAD in August, 2005

CMSC 330 9

CMSC 330 10

A Simple Example

- Let's start with a simple Ruby program

```
ruby1.rb:

# This is a ruby program
x = 37
y = x + 5
print(y)
print("\n")
```

Run Ruby, Run

- There are three ways to run a Ruby program
  - `ruby -w filename` – execute script in `filename`
    - tip: the `-w` will cause Ruby to print a bit more if something bad happens
  - `irb` – launch interactive Ruby shell
    - can type in Ruby programs one line at a time, and watch as each line is executed
      `irb(main):001> 3+4
      => 7`
  - `irb(main):002> print("hello")`

Run Ruby, Run (cont.)

- Suppose you want to run a Ruby script as if it were an executable

```
# This is a ruby program
x = 37
y = x + 5
print(y)
print("\n")
```

CMSC 330 10

CMSC 330 11

Language Basics

- comments begin with `#`, go to end of line
- variables need not be declared
- no special main() function or method
- line break separates expressions (can also use `;` to be safe)

CMSC 330 11

CMSC 330 12

Run Ruby, Run (cont.)

- Suppose you want to run a Ruby script as if it were an executable

```ruby
#!/usr/local/bin/ruby -w
print("Hello, world!\n")
```

Run Ruby, Run (cont.)

- `filename` – run program
  - The first line ("shebang") tells the system where to find the program to interpret this text file
  - Must `chmod u+x filename` first
  - Or `chmod a+x filename` so everyone has exec permission
  - Warning: Not very portable
  - Depends on location `/usr/local/bin/ruby`

CMSC 330 13
Explicit vs. Implicit Declarations

- Java and C/C++ use explicit variable declarations
  - Variables are named and typed before they are used
    - `int x, y; x = 37; y = x + 5;`

- In Ruby, variables are implicitly declared
  - First use of a variable declares it and determines type
    - `x = 37; y = x + 5;`
  - `x, y` exist, will be integers

Tradeoffs?

<table>
<thead>
<tr>
<th>Explicit Declarations</th>
<th>Implicit Declarations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Higher overhead</td>
<td>Lower overhead</td>
</tr>
<tr>
<td>Helps prevent typos</td>
<td>Easy to mistype variable name</td>
</tr>
<tr>
<td>Forces programmer to document types</td>
<td>Figures out types of variables automatically</td>
</tr>
</tbody>
</table>

Methods in Ruby

Methods are declared with `def...end`

```
def sayHello(message, n)
  i = 0
  while i < n
    puts message
    i = i + 1
  end
  return i
end
x = sayHello("hello", 3)
pputs(x)
```

List parameters at definition

Invoke method

May omit parens on call

(Methods should begin with lowercase letter and be defined before they are called)

More Control Statements in Ruby

- A control statement is one that affects which instruction is executed next
  - We’ve seen two so far in Ruby
    - `while` and `function call`

- Ruby also has conditionals

```
if grade >= 90 then
  puts "You got an A"
elsif grade >= 80 then
  puts "You got a B"
elsif grade >= 70 then
  puts "You got a C"
else
  puts "You’re not doing so well"
end
```

What is True?

- The guard of a conditional is the expression that determines which branch is taken

```
if grade >= 90 then
end
```

Guard

- The `true` branch is taken if the guard evaluates to anything except
  - `false`
  - `nil`
- Warning to C programmers: `0` is not `false`
Yet More Control Statements in Ruby

- `unless cond then stmt-f else stmt-t end`
  - Same as "if not cond then stmt-t else stmt-f end"

- `until cond body end`
  - Same as "while not cond body end"

Using If and Unless as Modifiers

- Can write if and unless after an expression
  - puts "You got an A" if grade >= 90
  - puts "You got an A" unless grade < 90

- Why so many control statements?
  - Is this a good idea?
  - Advantages? Disadvantages?

Other Useful Control Statements

- `for elt in [1, "math", 3.4] puts elt.to_s end`
- `while i > 0 break next puts message redo end`
- `for i in (1..3) puts i end`
- `O.foreach(filename) { |elt| puts elt }`
- `(1..3).each { |x| case x when 1, 3..5 when 2, 6..8 puts x end end end end end end` executes `code block` without `break`.

Using Ruby Control Statements

Ruby function to print all even numbers from 1 to some given value `x`

```ruby
def even(x)
  return nil unless x > 0
  (1..x).each{|i|
    if i % 2 == 0
      puts i
    end
  }
end
```

Classes and Objects

- Class names begin with an uppercase letter
- The "new" method creates an object
  - `s = String.new` creates a new `String` and makes `s` refer to it
- Every class inherits from `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 (the kind of data which things are)
- 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>&quot;CMSC 330&quot;</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
  - 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)
    - ib(main):004.0> @x
      - => nil
  - Like NULL or 0 in C/C++ and null in Java

- nil is an object of class NilClass
  - If 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
    - ib(main):006.0> @x = 2
    - NilClassError: 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 MyClass that contains a method
    - public static void main(String[] args)
  - When you run java MyClass, the main method of class MyClass 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
end

puts "Hello", sayN("hello", 1)  # sayN("hello", 1) does not return anything
```

- Defines a method of Object
- Invokes self.sayN
- Invokes self.puts (part of Object)

Ruby is Dynamically Typed

- Recall we don’t declare types of variables
  - But Ruby does keep track of types at run time
    - 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
puts 1 + 2  # 3
```
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 run time
    // (Some Java compilers may be smart enough to warn about above cast)
    ```

Tradeoffs?

<table>
<thead>
<tr>
<th>Static types</th>
<th>Dynamic types</th>
</tr>
</thead>
<tbody>
<tr>
<td>More work to do when writing code</td>
<td>Less work when writing code</td>
</tr>
<tr>
<td>Helps prevent some subtle errors</td>
<td>Can use objects incorrectly and not realize until execution</td>
</tr>
<tr>
<td>Fewer programs type check</td>
<td>More programs type check</td>
</tr>
</tbody>
</table>

Classes and Objects in Ruby

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

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

- Class contains method/constructor definitions
- Instance variables prefixed with "@"
- Method with no arguments
- Instantiation
- Invoking no-arg method

Notes For Java Programmers

- Ruby does not support method overloading
  - A typical Java class might have two or more constructors
  - Since Ruby does not support method overloading there can only be one initialize method
- Ruby does issue an exception or warning if classes defines more than one initialize method
  - But last initialize method defined is the valid one

Classes and Objects in Ruby (cont’d)

- 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`
  ```ruby
  class A
    def add(y)
      return x + 1
    end
  end
  class B < A
    def add(y)
      return (super + 1)
    end
  end
  b = B.new
  puts(b.add(3))
  ```
  - Extend superclass
  - Invoke add method of parent
super( ) in Ruby

- Within the body of a method
  - Call to super( ) acts just like a call to that original method
  - Except that search for method body starts in the superclass of the object that was found to contain the original method

Global Variables in Ruby

- Ruby has two kinds of global variables
  - Class variables beginning with @@ (static in Java)
  - 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

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

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
- Example
  ```ruby
  gets # implicitly reads input into $_
  print # implicitly writes $_
  ```
- Using $_ leads to shorter programs
  - And confusion
  - It's suggested you avoid using it

Creating Strings in Ruby

- Substitution in double-quoted strings with #{ }
  - course = "330"; msg = "Welcome to #{course}"
  - "It is now #{Time.now}"  
  - The contents of #{ } may be an arbitrary expression
  - Can also use single-quote as delimiter
    - No expression substitution, fewer escaping characters
- Here-documents
  ```ruby
  $ = <<END
  This is a text message on multiple lines and typing \n is annoying
  END
  ```

Substitution in Ruby Strings

- Writing elt as #{elt} makes it clear that it is a variable to be evaluated, not a literal word to be printed. This is a cleaner way to express output; it builds a single string and presents it as a single argument to puts.

```ruby
ruby> for elt in [100, -9.6, "pickle"]
|   puts "#{elt} is #\{elt.class\}"
| end
100  (Fixnum)
-9.6  (Float)
pickle (String)
```

Creating Strings in Ruby (cont.)

- Ruby also has printf and sprintf
  ```ruby
  printf("Hello, %s\n", name);
  sprintf("%d: %s", count, Time.now)
  ```
  - Returns a string
- The to_s method returns a String representation of a class object
Standard Library: String

- The String class has many useful methods
  - s.length  # length of string
  - s1 == s2  # structural equality (string contents)
  - s = "A line\n"; s.chomp  # returns "A line"
    > Return new string with s's contents except newline at end of line removed
  - s = "A line\n"; s.chomp!
    > Destructively removes newline from s
  - Convention: methods ending in ! modify the object
  - Another convention: methods ending in ? observe the object
  - "r1\tr2\tr3\.each("\t") { |rec| puts rec }
    > Apply code block to each tab-separated substring

Object Copy vs. Reference Copy

- Consider the following code
  - Assume an object/reference model like Java or Ruby
    > Or even two pointers pointing to the same structure
    ```ruby
    x = "groundhog" ; y = x
    ```
  - Which of these occur?
    - Object copy
    - Reference copy

Physical vs. Structural Equality

- Consider these cases again:
  - If we compare x and y, what is compared?
    > The references, or the contents of the objects they point to?
  - If references are compared (physical equality) the first would return false but the second true
  - If objects are compared both would return true

String Equality

- In Java, x == y is physical equality, always
  - Compares references, not string contents
- In Ruby, x == y for strings uses structural equality
  - Compares contents, not references
  - == is a method that can be overridden in Ruby!
  - To check physical equality, use the equal? method
    > Inherited from the Object class
- It's always important to know whether you're doing a reference or object copy
  - And physical or structural comparison

Object Copy vs. Reference Copy (cont.)

- For
  - ```ruby
    x = "groundhog" ; y = x
    ```
    > Ruby and Java would both do a reference copy
  - But for
    ```ruby
    x = "groundhog"
    y = String.new(x)
    ```
    > Ruby would cause an object copy
    > Unnecessary in Java since Strings are final
## Comparing Equality

<table>
<thead>
<tr>
<th>Language</th>
<th>Physical equality</th>
<th>Structural equality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>a == b</td>
<td>a.equals(b)</td>
</tr>
<tr>
<td>C</td>
<td>a == b</td>
<td>*a == *b</td>
</tr>
<tr>
<td>Ruby</td>
<td>a.equal?(b)</td>
<td>a == b</td>
</tr>
<tr>
<td>Ocaml</td>
<td>a == b</td>
<td>a = b</td>
</tr>
<tr>
<td>Python</td>
<td>a is b</td>
<td>a == b</td>
</tr>
<tr>
<td>Scheme</td>
<td>(eq? a b)</td>
<td>(equal? a b)</td>
</tr>
<tr>
<td>Visual Basic .NET</td>
<td>a ls b</td>
<td>a = b</td>
</tr>
</tbody>
</table>

## Summary

- **Scripting languages**
- **Ruby language**
  - Implicit variable declarations
  - Many control statements
  - Classes & objects
  - Strings