Homework 7

March 8, 2014

1 Instruction

Write the answers to all questions in one text file of the name `hw7.rb`.

For help with Ruby constructs, read Section 5.

2 Ruby programming

This question is to define Ruby classes for vectors and matrices.

Define two classes `MyVector` and `MyMatrix` with the methods described below.

A `MyVector` object represents a row vector while a `MyMatrix` object represents a matrix that internally organized as an array of MyVector objects.

Methods for `MyVector`

1. The `initialize` method takes an array of integers as argument.
2. The `length` method returns the size of the vector
3. The `*` method takes an argument `a`:

   if `a` is a vector, then it returns the inner product of this vector and `a`. Verify that the two vectors have the same length and raise an exception otherwise.

   if `a` is a matrix, then it returns the product of this vector and the matrix. Verify that the vector length is equal to the number of rows in the matrix and raise an exception otherwise.
4. The `to_s` method returns the string representation of this vector.

Methods for `MyMatrix`

1. The `initialize` method takes an array of arrays as argument and convert each inner array into a row vector (of MyVector class) in the matrix.
2. The `transpose` method returns this matrix transposed

3. The * method takes a MyMatrix object argument and returns the product of the two matrices.

   Verify that the number of columns of this matrix is equal to the number of rows of the argument and raise an exception otherwise.

4. The `to_s` method returns string representation of this matrix.

3 Test code

```
v = MyVector.new([1,2,3])
puts "v = " + v.to_s

v1 = MyVector.new([2,3,4])
puts "v1 = " + v1.to_s

puts "v * v1 = " + (v * v1).to_s

m = MyMatrix.new([[1,2], [1, 2], [1, 2]])
puts "m = " + m.to_s + "\n"

puts "v * m = " + (v * m).to_s

m1 = MyMatrix.new([[1, 2, 3], [2, 3, 4]])
puts "m1 = " + m1.to_s + "\n"

puts "m * m1 = " + (m * m1).to_s
puts "m1 * m = " + (m1 * m).to_s
```

4 Output

```
v = 1 2 3
v1 = 2 3 4
v * v1 = 20
m =
1 2
1 2
1 2

v * m = 6 12
m1 =
1 2 3
2 3 4

m * m1 =
5 8 11
```

```
5 8 11
5 8 11
m1 * m =
6 12
9 18
```

5 Ruby constructs for this homework

This homework is posted before we get started on Ruby. However, it won't due until after we have two full lectures on Ruby. So you can wait until we have at least one lecture on Ruby. Or if you are curious, here are the list of Ruby features you will need for this homework.

5.1 Ruby class

A Ruby class is defined using the following syntax:

```
class MyClassName
  def initialize(argument)
    ...
  end

  def method1(arguments)
    ...
  end

  def method2(arguments)
    ...
  end
end
```

Notice that `initialize` is the fixed name for any class constructor in Ruby. They are all called this name. If you mispell it, then you just defined an ordinary method.

Also notice that there is no field declaration since fields are created implicitly. If you want to define a field with a value v, just write `@fieldName = v` and the field `fieldName` is created.

5.2 New object

To instantiate a class, use the syntax `Class1.new(arguments)` to call the constructor of the class `Class1`.

5.3 Reflection

There are many ways to dynamically inspect an object.

For example, you can write `a.instance_of?(Class1)` to test whether the object `a` is an instance of the class `Class1`.

5.4 Ruby method

You can define Ruby method inside a class using the syntax:

```
def method1(arguments)
  ...
end
```

What is inside a method is an expression. You can have more than one expressions inside a method and the last expression's value is the return value of the method. You can have a return statement though it is not always needed.

For example, the following method `is_even?` tests whether its argument is an even number of not. The result is stored in local variable `b`, which is the return value (the last line).

```
def is_even?(x)
  if x % 2 == 0
    b = true
  else
    b = false
  end

  b
end
```

5.5 Method call

Like ML functions, Ruby method calls do not require paranthesis if the method has no parameter or has just one parameter.

5.6 Exception

If you wish to raise an exception you can use the syntax:
`raise "some exception text"`.

5.7 Branch

There are many ways to write an if-then-else expression in Ruby.

For example,

```
if <some-boolean-expression>
  <expression1>
elsif
  <expression2>
```

```
else
  <expression3>
end
```

Note that new line is significant here. Also note that if you want to write nested if-then-else, you should use `elsif`. Note that keyword has no letter **e** in the middle.

5.8 Loop

Again, there are many ways to write a loop. A simple way is to use a while loop:

```
while <some-boolean-expression>
    <expression1>
    <expression2>
    ...
end
```

5.9 Array

Ruby has a builtin Array class that is similar to the Java array class. For example, it has a length method and you can use indices.

5.10 String

All Ruby class has a to-string method but it is called `to_s`. You can override it to print your own string.

You can use + operator for string concatenation.