Thank you for your feedback with regards to the technical test. Having read it, I have decided to address it. To help explain the reasons behind these changes, I have created this document.

Having made these changes, I believe that your feedback has helped to improve the code quality.

V2 Feedback

The feedback highlighted the following:

1. Solution can be improved from an Object-Orientated perspective
2. File structure is unconventional
3. File naming is unconventional

V2 Approach

Acting on feedback

Having looked at the feedback, I agreed with all of it and hence decided to take the initiative to act upon it, in an effort to improve the code base.

I have approached the feedback as follows:

Solution can be improved from an Object-Orientated perspective

I agree with this statement. Whilst designing the code in V1, I was trying to decide between two approaches:

1) Using Classes
2) Using Modules

At the time, I decided to go with modules, since I thought it would reduce the 'footprint' of setting up the objects. By that, I mean there is less work to be done to carry out the processing of characters, without having to instantiate additional object. In hindsight, I realize that this might not have been the best decision, since, by including modules into a class, it adds more responsibility to the class – something that I was trying to encapsulate by creating the modules in the first place.

Taking this feedback into consideration, I have changed the following modules to classes:

1) Parser
2) Processors (Integer, Multiplication, Integer)

By doing this change, although the 'footprint' of setting up the objects is more expensive, the use of it is much better.

Previous usage:

```
processor_for(character).process(@stack, character)
```

New usage:

```
processor_for(character).process(character)
```

As you can see, we no longer would need to keep passing the stack around, since the object has been initialized with the stack.

File structure is unconventional

This is true. This was done by mistake/oversight on my behalf. To rectify, this, I have added a directory within the lib folder, which is named stack_machine (same as the name of the gem). In addition, this was also mirrored in the structure of the spec directory.

Previous structure	New structure
`├── Gemfile`	`├── Gemfile`
`├── Gemfile.lock`	`├── Gemfile.lock`
`├── LICENSE.txt`	`├── LICENSE.txt`
`├── README.md`	`├── README.md`
`├── Rakefile`	`├── Rakefile`
`├── lib`	`├── lib`
`|   ├── parser.rb`	`|   ├── stack_machine`
`|   ├── processors.rb`	`|   |   ├── parser.rb`
`|   ├── stack.rb`	`|   |   ├── processors.rb`
`|   ├── stack_machine`	`|   |   ├── stack.rb`
`|   |   └── version.rb`	`|   |   └── version.rb`
`|   └── stack_machine.rb`	`|   └── stack_machine.rb`
`├── spec`	`├── spec`
`|   ├── parser.rb`	`|   ├── spec_helper.rb`
`|   ├── processors.rb`	`|   ├── stack_machine`
`|   ├── spec_helper.rb`	`|   |   ├── parser_spec.rb`
`|   ├── stack.rb`	`|   |   ├── processors_spec.rb`
`|   └── stack_machine.rb`	`|   |   └── stack_spec.rb`
` └── stack_machine.gemspec`	`|   └── stack_machine_spec.rb`
	`└── stack_machine.gemspec`

File naming is unconventional

This is also true – in particular the spec files names did not have '_spec' appended to them. This has been rectified as follows:

Previous structure	New structure
`├── spec`	`├── spec`
`|   ├── parser.rb`	`|   ├── spec_helper.rb`
`|   ├── processors.rb`	`|   ├── stack_machine`
`|   ├── spec_helper.rb`	`|   |   ├── parser_spec.rb`
`|   ├── stack.rb`	`|   |   ├── processors_spec.rb`
`|   └── stack_machine.rb`	`|   |   └── stack_spec.rb`
` └── stack_machine.gemspec`	`|   └── stack_machine_spec.rb`
	`└── stack_machine.gemspec`

Additional Changes

Although this additional change was not mentioned in the feedback, having looked at the solution, I felt like it would be beneficial.

Change:

No longer using a map constant to figure out processor based on character, but instead using a function.

Previous usage:

```
OPERATOR_SIGN_TO_PROCESSOR_MAP = {
          '+' => 'Addition',
          '*' => 'Multiplication'
          } unless defined?(OPERATOR_SIGN_TO_PROCESSOR_MAP)

   def processor_for(character)
     if is_operator?(character)
       processor =
"StackMachine::Processors::#{OPERATOR_SIGN_TO_PROCESSOR_MAP[character]}"
     else
       processor = "StackMachine::Processors::Integer"
     end
     StackMachine.const_get(processor).new
   end
```

New usage:

```
  def processor_for(character)
     if character =~ /^[+]$/
       StackMachine::Processors::Addition.new(@stack)
     elsif character =~ /^[*]$/
       StackMachine::Processors::Multiplication.new(@stack)
     elsif character =~ /^[0123456789]$/
       StackMachine::Processors::Integer.new(@stack)
     end
   end
```

By making this change, we have been able to remove a constant and also the additional 'is_operator?' function from the Parser class.

The new function, works by matching the character against a series of regex's to figure out which processor is the correct one to use. By doing this approach, we will be able to add new character types in the future (should we wish to) with ease (i.e. a processor for subtractions or vision). In addition, we have been able to remove dynamic Ruby methods and constantizeation (which are slower to run).