Apuntes de la Asignatura Procesadores de Lenguajes

Casiano R. León [1]

12 de mayo de 2014

[1]DEIOC Universidad de La Laguna

Índice general

Índice de figuras

Índice de cuadros

A Juana

For it is in teaching that we learn
And it is in understanding that we are understood

Agradecimientos/Acknowledgments

I'd like to thank

A mis alumnos de Procesadores de Lenguajes del Grado de Informática de la Escuela Superior de Informática en la Universidad de La Laguna

Parte I

PARTE: APUNTES DE PROCESADORES DE LENGUAJES

Capítulo 1

Expresiones Regulares y Análisis Léxico en JavaScript

1.1. Mozilla Developer Network: Documentación

1. RegExp Objects
2. exec
3. search
4. match
5. replace

1.2. Práctica: Conversor de Temperaturas

Véase https://bitbucket.org/casiano/pl-grado-temperature-converter/src.

```
[~/srcPLgrado/temperature(master)]$ pwd -P
/Users/casiano/local/src/javascript/PLgrado/temperature # 27/01/2014
```

index.html

```
<html>
  <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>JavaScript Temperature Converter</title>
      <link href="global.css" rel="stylesheet" type="text/css">

     <script type="text/javascript" src="temperature.js"></script>
  </head>
  <body>
    <h1>Temperature Converter</h1>
    <table>
      <tr>
        <th>Enter  Temperature (examples: 32F, 45C, -2.5f):</th>
        <td><input id="original" onchange="calculate();"></td>
      </tr>
      <tr>
        <th>Converted Temperature:</th>
```

```
        <td><span class="output" id="converted"></span></td>
      </tr>
    </table>
  </body>
</html>
```

global.css

```
th, td      { vertical-align: top; text-align: right; font-size:large; }
#converted  { color: red; font-weight: bold; font-size:large;          }
input       { text-align: right;  border: none; font-size:large;       }
body
{
 background-color:#b0c4de;  /* blue */
 font-size:large;
}
```

temperature.js

```
"use strict"; // Use ECMAScript 5 strict mode in browsers that support it
function calculate() {
  var result;
  var original       = document.getElementById("........");
  var temp = original.value;
  var regexp = /.............................../;

  var m = temp.match(......);

  if (m) {
    var num = ....;
    var type = ....;
    num = parseFloat(num);
    if (type == 'c' || type == 'C') {
      result = (num * 9/5)+32;
      result = .............................
    }
    else {
      result = (num - 32)*5/9;
      result = ............................
    }
    converted.innerHTML = result;
  }
  else {
    converted.innerHTML = "ERROR! Try something like '-4.2C' instead";
  }
}
```

Despliegue

- Deberá desplegar la aplicación en GitHub Pages como página de proyecto. Vea la sección *GitHub Project Pages* 13.4.4.

Mocha y Chai Mocha is a test framework while Chai is an expectation one.

Let's say Mocha setups and describes test suites and Chai provides convenient helpers to perform all kinds of assertions against your JavaScript code.

Pruebas: Estructura

Instale `mocha`.

```
$ npm install -g mocha
```

Creamos la estructura para las pruebas:

```
$ mocha init tests

$ tree tests
tests
|-- index.html
|-- mocha.css
|-- mocha.js
`-- tests.js
```

Añadimos `chai.js` (Véase http://chaijs.com/guide/installation/) al directorio `tests`.

The latest tagged version will be available for hot-linking at http://chaijs.com/chai.js.

If you prefer to host yourself, use the `chai.js` file from the root of the github project.

```
[~/srcPLgrado/temperature(master)]$ tree tests/
tests/
|-- chai.js
|-- index.html
|-- mocha.css
|-- mocha.js
`-- tests.js

0 directories, 5 files
```

Podemos encontrar un ejemplo de unit testing en JavaScript en el browser con el testing framework Mocha y Chai en el repositorio https://github.com/ludovicofischer/mocha-chai-browser-demo.

Pruebas: `index.html`

Modificamos `index.html` para

- Cargar `chai.js`
- Cargar `temperature.js`
- Usar el estilo `mocha.setup('tdd')`:
- Imitar la página `index.html` con los correspondientes `input` y `span`:

```
<input id="original" placeholder="32F" size="50">
<span class="output" id="converted"></span>
```

```
[~/srcPLgrado/temperature(master)]$ cat tests/index.html
<!DOCTYPE html>
<html>
  <head>
    <title>Mocha</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="mocha.css" />
  </head>
  <body>
    <div id="mocha"></div>
    <input id="original" placeholder="32F" size="50">
    <span class="output" id="converted"></span>

    <script src="chai.js"></script>
    <script src="mocha.js"></script>
    <script src="../temperature.js"></script>
    <script>mocha.setup('tdd')</script>
    <script src="tests.js"></script>

    <script>
      mocha.run();
    </script>
  </body>
</html>
```

Pruebas: Añadir los tests

The "TDDïnterface provides

- suite()
- test()
- setup()
- teardown().

```
[~/srcPLgrado/temperature(master)]$ cat tests/tests.js
var assert = chai.assert;

suite('temperature', function() {
    test('32F = 0C', function() {
        original.value = "32F";
        calculate();
        assert.deepEqual(converted.innerHTML, "0.0 Celsius");
    });
    test('45C = 113.0 Farenheit', function() {
        original.value = "45C";
        calculate();
        assert.deepEqual(converted.innerHTML, "113.0 Farenheit");
    });
    test('5X = error', function() {
        original.value = "5X";
        calculate();
        assert.match(converted.innerHTML, /ERROR/);
```

```
    });
});
```

Pruebas: Véase

- Testing your frontend JavaScript code using mocha, chai, and sinon by Nicolas Perriault

1.3. Práctica: Comma Separated Values. CSV

Donde

```
[~/srcPLgrado/csv(master)]$ pwd -P
/Users/casiano/local/src/javascript/PLgrado/csv
```

Véase https://bitbucket.org/casiano/pl-grado-csv/src y https://github.com/crguezl/csv.

Introducción al formato CSV
Véase Comma Separated Values en la Wikipedia:
A comma-separated values (CSV) file stores tabular data (numbers and text) in plain-text form. A CSV file consists of any number of records, separated by line breaks of some kind; each record consists of fields, separated by a comma. All records have an identical sequence of fields.

Ejemplo de ejecución
Véase la página en http://crguezl.github.io/csv/. Pruebe a dar como entrada cualquiera de estas dos

```
[~/srcPLgrado/csv(gh-pages)]$ cat input.txt
"producto",           "precio"
"camisa",             "4,3"
"libro de O\"Reilly", "7,2"
```

```
[~/srcPLgrado/csv(gh-pages)]$ cat input2.txt
"producto",           "precio"  "fecha"
"camisa",             "4,3",    "14/01"
"libro de O\"Reilly", "7,2"     "13/02"
```

Pruebe también a dar alguna entrada errónea.

Aproximación al análisis mediante expresiones regulares de CSV Una primera aproximación sería hacer `split` por las comas:

```
> x = '"earth",1,"moon",9.374'
'"earth",1,"moon",9.374'
> y = x.split(/,/)
[ '"earth"', '1', '"moon"', '9.374' ]
```

Esta solución deja las comillas dobles en los campos entrecomillados. Peor aún, los campos entrecomillados pueden contener comas, en cuyo caso la división proporcionada por `split` sería errónea:

```
> x = '"earth, mars",1,"moon, fobos",9.374'
'"earth, mars",1,"moon, fobos",9.374'
> y = x.split(/,/)
[ '"earth', ' mars"', '1', '"moon', ' fobos"', '9.374' ]
```

La siguiente expresión regular reconoce cadenas de comillas dobles con secuencias de escape seguidas opcionalmente de una coma:

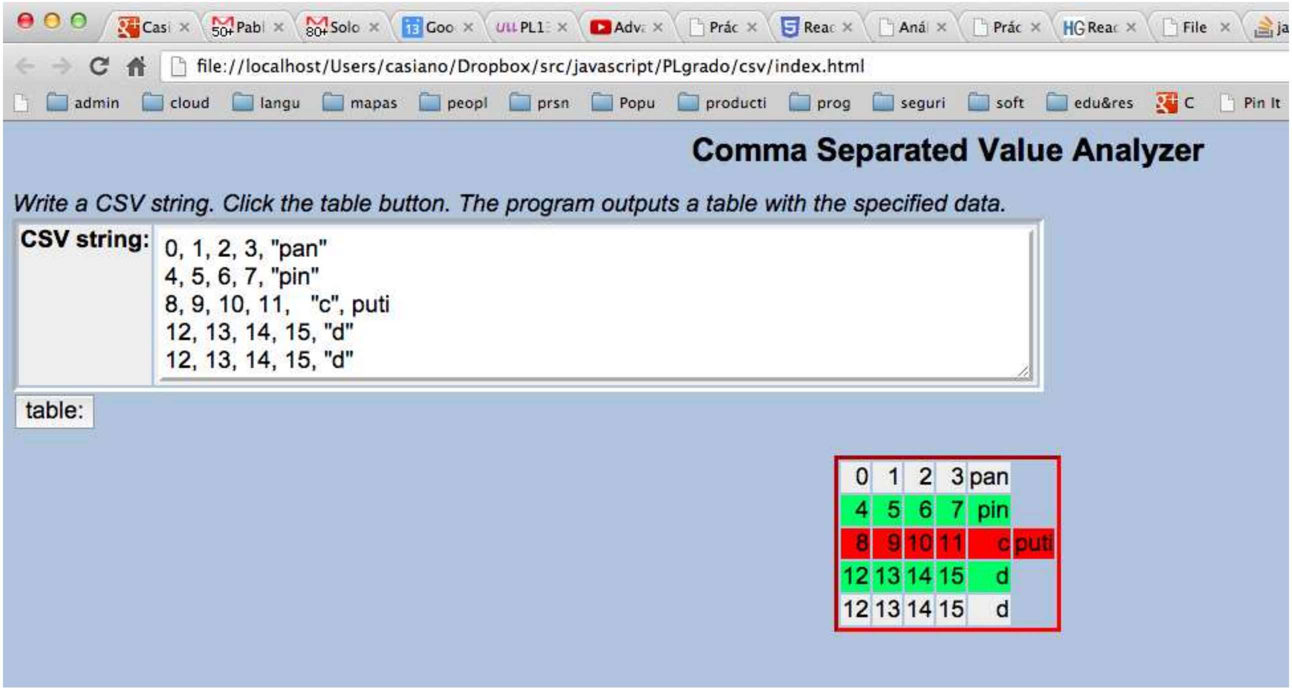


Figura 1.1: Ejemplo de pantalla de La aplicación para el Análisis de Datos en Formato CSV

```
> x = ’"earth, mars",1,"moon, fobos",9.374’
’"earth, mars",1,"moon, fobos",9.374’
> r = /"((?:[^"\\]|\\.)*)"\s*,?/g
/"((?:[^"\\]|\\.)*)"\s*,?/g
> w = x.match(r)
[ ’"earth, mars",’, ’"moon, fobos",’ ]
```

If your regular expression uses the `g` flag, you can use the `exec` or `match` methods multiple times to find successive matches in the same string. When you do so, the search starts at the substring of string specified by the regular expression's `lastIndex` property.

Javascript sub-matches stop working when the `g` modifier is set:

```
> text = ’test test test test’
’test test test test’
> text.match(/t(e)(s)t/)
[ ’test’, ’e’, ’s’, index: 0, input: ’test test test test’ ]
> text.match(/t(e)(s)t/g)
[ ’test’, ’test’, ’test’, ’test’ ]
```

Sin embargo el método `exec` de las expresiones regulares si que conserva las subexpresiones que casan con los paréntesis:

```
> r = /t(e)(s)t/g
/t(e)(s)t/g
> text = ’test test test test’
’test test test test’
> while (m = r.exec(text)) {
... console.log(m);
... }
[ ’test’, ’e’, ’s’, index: 0, input: ’test test test test’ ]
[ ’test’, ’e’, ’s’, index: 5, input: ’test test test test’ ]
```

```
[ 'test', 'e', 's', index: 10, input: 'test test test test' ]
[ 'test', 'e', 's', index: 15, input: 'test test test test' ]
undefined
```

Another catch to remember is that `exec()` doesn't return the matches in one big array: it keeps returning matches until it runs out, in which case it returns `null`.

Véase

- Javascript Regex and Submatches en StackOverflow.
- La sección *Ejercicios* 1.5

Esta otra expresión regular `/([^,]+),?|\s*,/` actúa de forma parecida al `split`. Reconoce secuencias no vacías de caracteres que no contienen comas seguidas opcionalmente de una coma o bien una sóla coma (precedida opcionalmente de blancos):

```
> x = '"earth, mars",1,"moon, fobos",9.374'
'"earth, mars",1,"moon, fobos",9.374'
> r = /([^,]+),?|\s*,/g
/([^,]+),?|\s*,/g
> w = x.match(r)
[ '"earth,', ' mars",', '1,', '"moon,', ' fobos",', '9.374' ]
```

La siguiente expresión regular es la unión de dos:

- Cadenas de dobles comillas seguidas de una coma opcional entre espacios en blanco
- Cadenas que no tienen comas

```
>  x = '"earth, mars",1,"moon, fobos",9.374'
'"earth, mars",1,"moon, fobos",9.374'
> r = /\s*"((?:[^"\\]|\\.)*)"\s*,?|\s*([^,]+),?|\s*,/g
/\s*"((?:[^"\\]|\\.)*)"\s*,?|\s*([^,]+),?|\s*,/g
> w = x.match(r)
[ '"earth, mars",', '1,', '"moon, fobos",', '9.374' ]
```

El operador `|` trabaja en circuito corto:

```
> r = /(ba?)|(b)/
/(ba?)|(b)/
> r.exec("ba")
[ 'ba', 'ba', undefined, index: 0, input: 'ba' ]
> r = /(b)|(ba?)/
/(b)|(ba?)/
> r.exec("ba")
[ 'b', 'b', undefined, index: 0, input: 'ba' ]
```

Si usamos `exec` tenemos:

```
> x = '"earth, mars",1,"moon, fobos",9.374'
'"earth, mars",1,"moon, fobos",9.374'
> r = /\s*"((?:[^"\\]|\\.)*)"\s*,?|\s*([^,]+),?|\s*,/g
/\s*"((?:[^"\\]|\\.)*)"\s*,?|\s*([^,]+),?|\s*,/g
> while (m = r.exec(x)) { console.log(m); }
[ '"earth, mars",', 'earth, mars', undefined, index: 0,
  input: '"earth, mars",1,"moon, fobos",9.374' ]
[ '1,', undefined, '1', index: 14,
  input: '"earth, mars",1,"moon, fobos",9.374' ]
```

```
[ '"moon, fobos",', 'moon, fobos', undefined, index: 16,
  input: '"earth, mars",1,"moon, fobos",9.374' ]
[ '9.374', undefined, '9.374', index: 30,
  input: '"earth, mars",1,"moon, fobos",9.374' ]
undefined
```

1. RegExp Objects

   The `RegExp` constructor creates a regular expression object for matching text with a pattern.

   Literal and constructor notations are possible:

   ```
   /pattern/flags;
   new RegExp(pattern [, flags]);
   ```

   - The literal notation provides compilation of the regular expression when the expression is evaluated.
   - Use literal notation when the regular expression will remain constant.
   - For example, if you use literal notation to construct a regular expression used in a loop, the regular expression won't be recompiled on each iteration.

   - The constructor of the regular expression object, for example, `new RegExp("ab+c")`, provides runtime compilation of the regular expression.
   - Use the constructor function when you know the regular expression pattern will be changing, or you don't know the pattern and are getting it from another source, such as user input.
   - When using the constructor function, the normal string escape rules (preceding special characters with \ when included in a string) are necessary. For example, the following are equivalent:

     ```
     var re = /\w+/;
     var re = new RegExp("\\w+");
     ```

2. exec

3. search

   `str.search(regexp)`

   If successful, search returns the index of the regular expression inside the string. Otherwise, it returns `-1`.

   When you want to know whether a pattern is found in a string use search (similar to the regular expression `test` method); for more information (but slower execution) use `match` (similar to the regular expression `exec` method).

4. match

5. replace

   The `replace()` method returns a new string with some or all matches of a pattern replaced by a replacement. The pattern can be a string or a `RegExp`, and the replacement can be a string or a function to be called for each match.

   ```
   > re = /apples/gi
   /apples/gi
   > str = "Apples are round, and apples are juicy."
   'Apples are round, and apples are juicy.'
   > newstr = str.replace(re, "oranges")
   'oranges are round, and oranges are juicy.'
   ```

The replacement string can be a function to be invoked to create the new substring (to put in place of the substring received from parameter #1). The arguments supplied to this function are:

Possible name	Supplied value
match	The matched substring. (Corresponds to $&.)
p1, p2, ...	The nth parenthesized submatch string, provided the first argument to re (Corresponds to $1, $2, etc.) For example, if /(\a+)(\b+)/, was given, p p2 for \b+.
offset	The offset of the matched substring within the total string being examine string was abcd, and the matched substring was bc, then this argument w
string	The total string being examined

```
[~/javascript/learning]$ pwd -P
/Users/casiano/local/src/javascript/learning
[~/javascript/learning]$ cat f2c.js
#!/usr/bin/env node
function f2c(x)
{
  function convert(str, p1, offset, s)
  {
    return ((p1-32) * 5/9) + "C";
  }
  var s = String(x);
  var test = /(\d+(?:\.\d*)?)F\b/g;
  return s.replace(test, convert);
}

var arg = process.argv[2] || "32F";
console.log(f2c(arg));

[~/javascript/learning]$ ./f2c.js 100F
37.77777777777778C
[~/javascript/learning]$ ./f2c.js
0C
```

index.html

```
<html>
  <head>
     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
     <title>CSV Analyzer</title>
     <link href="global.css" rel="stylesheet" type="text/css">

     <script type="text/javascript" src="../../underscore/underscore.js"></script>
     <script type="text/javascript" src="../../jquery/starterkit/jquery.js"></script>
     <script type="text/javascript" src="csv.js"></script>
  </head>
  <body>
    <h1>Comma Separated Value Analyzer</h1>
    <div>
      <i>Write a CSV string. Click the table button. The program outputs a table with the spec
    </div>
    <table>
      <tr>
```

```
        <th>CSV string:</th> <!-- autofocus attribute is HTML5 -->
        <td><textarea autofocus cols = "80" rows = "5" id="original"></textarea></td>
      </tr>
    </table>
    <button type="button">table:</button><br>
    <span class="output" id="finaltable"></span>
  </body>
</html>
```

jQuery

jQuery (Descarga la librería)

jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML.

- It was released in January 2006 at BarCamp NYC by John Resig.
- It is currently developed by a team of developers led by Dave Methvin.
- jQuery is the most popular JavaScript library in use today
- jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications.
- The set of jQuery core features — DOM element selections, traversal and manipulation — enabled by its selector engine (named "Sizzle"from v1.3), created a new "programming style", fusing algorithms and DOM-data-structures; and influenced the architecture of other JavaScript frameworks like YUI v3 and Dojo.
-

How JQuery Works

- Véase How jQuery Works
- https://github.com/crguezl/how-jquery-works-tutorial en GitHub
-

```
[~/javascript/jquery(master)]$ pwd -P
/Users/casiano/local/src/javascript/jquery
```

```
~/javascript/jquery(master)]$ cat index.html
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Demo</title>
</head>
<body>
    <a href="http://jquery.com/">jQuery</a>
    <script src="starterkit/jquery.js"></script>
    <script>

    // Your code goes here.

    </script>
</body>
</html>
```

To ensure that their code runs after the browser finishes loading the document, many JavaScript programmers wrap their code in an onload function:

```
window.onload = function() { alert( "welcome" ); }
```

Unfortunately, the code doesn't run until all images are finished downloading, including banner ads. To run code as soon as the document is ready to be manipulated, jQuery has a statement known as the ready event:

```
$( document ).ready(function() {
    // Your code here.
});
```

For `click` and most other events, you can prevent the default behavior by calling event.preventDefault() in the event handler. If this method is called, the default action of the event will not be triggered. For example, clicked anchors will not take the browser to a new URL.

```
[~/javascript/jquery(master)]$ cat index2.html
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Demo</title>
</head>
<body>
    <a href="http://jquery.com/">jQuery</a>
    <script src="starterkit/jquery.js"></script>
    <script>

    $( document ).ready(function() {
        $( "a" ).click(function( event ) {
            alert( "The link will no longer take you to jquery.com" );
            event.preventDefault();
        });
    });

    </script>
</body>
</html>
```

Borrowing from CSS 1–3, and then adding its own, jQuery offers a powerful set of tools for matching a set of elements in a document.

See jQuery Category: Selectors.

Another common task is adding or removing a class. jQuery also provides some handy effects.

```
[~/javascript/jquery(master)]$ cat index3.html
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <style>
        a.test { font-weight: bold; }
    </style>
    <title>Demo</title>
</head>
```

```
<body>
    <a href="http://jquery.com/">jQuery</a>
    <script src="starterkit/jquery.js"></script>
    <script>

    $( document ).ready(function() {
        $( "a" ).click(function( event ) {
            $( "a" ).addClass( "test" );
            alert( "The link will no longer take you to jquery.com" );
            event.preventDefault();
            $( "a" ).removeClass( "test" );
            $( this ).hide( "slow" );
            $( this ).show( "slow" );
        });
    });

    </script>
</body>
</html>
```

- In JavaScript `this` always refers to the *owner* of the function we're executing, or rather, *to the object that a function is a method of.*
- When we define our function `tutu()` in a page, its owner is the page, or rather, the `window` object (or `global` object) of JavaScript.
- An `onclick` property, though, is owned by the HTML element it belongs to.
- The method .addClass(className) adds the specified class(es) to each of the set of matched elements.

  `className` is a String containing one or more space-separated classes to be added to the class attribute of each matched element.

  This method does not replace a class. It simply adds the class, appending it to any which may already be assigned to the elements.
- The method `.removeClass( [className ] )` removes a single class, multiple classes, or all classes from each element in the set of matched elements.

  If a class name is included as a parameter, then only that class will be removed from the set of matched elements. If no class names are specified in the parameter, all classes will be removed.

  This method is often used with `.addClass()` to switch elements' classes from one to another, like so:

  ```
  $( "p" ).removeClass( "myClass noClass" ).addClass( "yourClass" );
  ```

JavaScript enables you to freely pass functions around to be executed at a later time. A *callback* is a function that is passed as an argument to another function and is usually executed after its parent function has completed.

Callbacks are special because they wait to execute until their parent finishes or some event occurs. Meanwhile, the browser can be executing other functions or doing all sorts of other work.

```
[~/javascript/jquery(master)]$ cat app.rb
require 'sinatra'

set :public_folder, File.dirname(__FILE__) + '/starterkit'
```

```
get '/' do
  erb :index
end

get '/chuchu' do
  if request.xhr?
    "hello world!"
  else
    erb :tutu
  end
end

__END__

@@layout
  <!DOCTYPE html>
  <html>
    <head>
        <meta charset="utf-8" />
        <title>Demo</title>
    </head>
    <body>
        <a href="http://jquery.com/">jQuery</a>
        <div class="result"></div>
        <script src="jquery.js"></script>
        <%= yield %>
    </body>
  </html>

@@index
  <script>
  $( document ).ready(function() {
      $( "a" ).click(function( event ) {
          event.preventDefault();
          $.get( "/chuchu", function( data ) {
            $( ".result" ).html( data );
            alert( "Load was performed." );
          });
      });
  });
  </script>

@@tutu
  <h1>Not an Ajax Request!</h1>
```

- `jQuery.get( url [, data ] [, success(data, textStatus, jqXHR) ] [, dataType ] )` load data from the server using a HTTP GET request.
- `url`

  Type: String

  A string containing the URL to which the request is sent.
- `data`

Type: PlainObject or String

A plain object or string that is sent to the server with the request.

- `success(data, textStatus, jqXHR)`

  Type: Function()

  A callback function that is executed if the request succeeds.

- `dataType`

  Type: String

  The type of data expected from the server. Default: Intelligent Guess (`xml`, `json`, `script`, or `html`).

To use callbacks, it is important to know how to pass them into their parent function.
Executing callbacks with arguments can be tricky.
This code example will not work:

```
$.get( "myhtmlpage.html", myCallBack( param1, param2 ) );
```

The reason this fails is that the code executes

```
myCallBack( param1, param2)
```

immediately and then passes `myCallBack()`'s return value as the second parameter to `$.get()`.
We actually want to pass the function `myCallBack`, not `myCallBack( param1, param2 )`'s return value (which might or might not be a function).
So, how to pass in `myCallBack()` and include arguments?
To defer executing `myCallBack()` with its parameters, you can use an anonymous function as a wrapper.

```
[~/javascript/jquery(master)]$ cat app2.rb
require 'sinatra'

set :public_folder, File.dirname(__FILE__) + '/starterkit'

get '/' do
  erb :index
end

get '/chuchu' do
  if request.xhr? # is an ajax request
    "hello world!"
  else
    erb :tutu
  end
end

__END__

@@layout
  <!DOCTYPE html>
  <html>
    <head>
        <meta charset="utf-8" />
        <title>Demo</title>
```

```
  </head>
  <body>
      <a href="http://jquery.com/">jQuery</a>
      <div class="result"></div>
      <script src="jquery.js"></script>
      <%= yield %>
  </body>
</html>

@@tutu
  <h1>Not an Ajax Request!</h1>

@@index
  <script>
    var param = "chuchu param";
    var handler = function( data, textStatus, jqXHR, param ) {
      $( ".result" ).html( data );
      alert( "Load was performed.\n"+
             "$data = "+data+
             "\ntextStatus = "+textStatus+
             "\njqXHR = "+JSON.stringify(jqXHR)+
             "\nparam = "+param );
    };
    $( document ).ready(function() {
        $( "a" ).click(function( event ) {
            event.preventDefault();
            $.get( "/chuchu", function(data, textStatus, jqXHR ) {
              handler( data, textStatus, jqXHR, param);
            });
        });
    });
  </script>
```

El ejemplo en `app2.rb` puede verse desplegado en Heroku: http://jquery-tutorial.herokuapp.com/

JSON.stringify() The JSON.stringify() method converts a value to `JSON` , optionally replacing values if a `replacer function` is specified, or optionally including only the specified properties if a `replacer array` is specified.

```
JSON.stringify(value[, replacer [, space]])
```

- `value`

  The value to convert to a JSON string.

- `replacer`

  - If a function, transforms values and properties encountered while stringifying;
  - if an array, specifies the set of properties included in objects in the final string.

- `space`

  Causes the resulting string to be pretty-printed.

See another example of use in http://jsfiddle.net/casiano/j7tsF/. To learn to use JSFiddle wath the YouTube video How to use JSFiddle by Jason Diamond

Underscore

Underscore: is a utility-belt library for JavaScript that provides a lot of the functional programming support that you would expect in Ruby.

- Underscore provides functions that support methods like:

  `map`, `select`, `invoke`

- as well as more specialized helpers:

  function binding, javascript templating, deep equality testing, and so on.

- Cargando la librería:

```
[~/javascript/jquery(master)]$ node
> 2+3
5
> _
5
> uu = require('underscore')
{ [Function]
  _: [Circular],
  VERSION: '1.5.2',
  forEach: [Function],
  each: [Function],
  collect: [Function],
  map: [Function],
  inject: [Function],
  reduce: [Function],
  .................
  chain: [Function] }
```

- `each`:

```
> uu.each([1, 2, 3], function(x) { console.log(x*x); })
1
4
9
```

- `map`:

```
> uu.map([1, 2, 3], function(num){ return num * 3; })
[ 3, 6, 9 ]
```

- `invoke`

```
> z = [[6,9,1],[7,3,9]]
[ [ 6, 9, 1 ], [ 7, 3, 9 ] ]
> uu.invoke(z, 'sort')
[ [ 1, 6, 9 ], [ 3, 7, 9 ] ]
> uu.invoke(z, 'sort', function(a, b) { return b-a; })
[ [ 9, 6, 1 ], [ 9, 7, 3 ] ]
```

- `reduce`:

```
> uu.reduce([1, 2, 3, 4], function(s, num){ return s + num; }, 0)
10
> uu.reduce([1, 2, 3, 4], function(s, num){ return s * num; }, 1)
24
> uu.reduce([1, 2, 3, 4], function(s, num){ return Math.max(s, num); }, -1)
4
> uu.reduce([1, 2, 3, 4], function(s, num){ return Math.min(s, num); }, 99)
1
```

- `filter`: (`select` is an alias for `filter`)

```
> uu.filter([1, 2, 3, 4, 5, 6], function(num){ return num % 2 == 0; })
[ 2, 4, 6 ]
```

- `isEqual`

```
> a = {a:[1,2,3], b: { c: 1, d: [5,6]}}
{ a: [ 1, 2, 3 ],
  b: { c: 1, d: [ 5, 6 ] } }
> b = {a:[1,2,3], b: { c: 1, d: [5,6]}}
{ a: [ 1, 2, 3 ],
  b: { c: 1, d: [ 5, 6 ] } }
> a == b
false
> uu.isEqual(a,b)
true
```

- `bind`

```
> func = function(greeting){ return greeting + ': ' + this.name }
[Function]
> func = uu.bind(func, {name: 'moe'})
[Function]
> func('hello')
'hello: moe'
> func = uu.bind(func, {name: 'moe'}, 'hi')
[Function]
> func()
'hi: moe'
>
```

Templates en Underscore

- Underscore: template

  `_.template(templateString, [data], [settings])`

  Compiles JavaScript templates into functions that can be evaluated for rendering. Useful for rendering complicated bits of HTML from a JavaScript object or from *JSON* data sources.

  JSON, or *JavaScript Object Notation*, is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is used primarily to transmit data between a server and web application, as an alternative to XML. Although originally derived from the JavaScript scripting language, JSON is a language-independent data format, and code for parsing and generating JSON data is readily available in a large variety of programming languages.

- Template functions can both interpolate variables, using `<%= ... %>`,

```
> compiled = uu.template("hello: <%= name %>")
{ [Function]
  source: 'function(obj){
    var __t,__p=\'\', __j=Array.prototype.join, i
        print=function(){__p+=__j.call(arguments,\'\');};
    with(obj||{}){
      __p+=\'hello: \'+ ((__t=( name ))==null?\'\':__t)+ \'\';
    }
    return __p;
  }'
}
> compiled({name: 'moe'})
'hello: moe'
```

- as well as execute arbitrary JavaScript code, with `<% ... %>`.

```
> uu = require('underscore')
> list = "\
... <% _.each(people, function(name) { %>\
..... <li><%= name %></li>\
... <% }); %>"
'<% _.each(people, function(name) { %> <li><%= name %></li> <% }); %>'
> uu.template(list, {people: ['moe', 'curly', 'larry']})
' <li>moe</li>  <li>curly</li>  <li>larry</li> '
```

- When you evaluate a template function, pass in a data object that has properties corresponding to the template's free variables.

  If you're writing a one-off, like in the example above, you can pass the data object as the second parameter to template in order to render immediately instead of returning a template function.

- If you wish to interpolate a value, and have it be HTML-escaped, use `<%- ... %>`

```
> template = uu.template("<b><%- value %></b>")
{ [Function]
  source: 'function(obj){
    var __t,__p=\'\',__j=Array.prototype.join,print=function(){__p+=__j.call(argument
    with(obj||{}){
      __p+=\'<b>\'+
      ((__t=( value ))==null?\'\':_.escape(__t))+
      \'</b>\';
    }
    return __p;
  }'
}
> template({value: '<script>'})
'<b>&lt;script&gt;</b>'
```

- The `settings` argument should be a hash containing any `_.templateSettings` that should be overridden.

```
_.template("Using 'with': <%= data.answer %>", {answer: 'no'}, {variable: 'data'});
=> "Using 'with': no"
```

  Another example:

```
template = uu.template("<b>{{ value }}</b>",{value: 4 },
                       { interpolate: /\{\{(.+?)\}\}/g })
'<b>4</b>'
```

You can also use `print` from within JavaScript code. This is sometimes more convenient than using `<%= ... %>`.

```
> compiled = uu.template("<% print('Hello ' + epithet); %>")
{ [Function]
  source: 'function(obj){\n
    var __t,__p=\'\',
        __j=Array.prototype.join,print=function(){
            __p+=__j.call(arguments,\'\');};\n
            with(obj||{}){\n
              __p+=\'\';\n print(\'Hello \' + epithet); \n
              __p+=\'\';\n}\n
              return __p;\n
  }'
}
> compiled({ epithet : 'stooge' })
'Hello stooge'
>
```

If ERB-style delimiters aren't your cup of tea, you can change Underscore's template settings to use different symbols to set off interpolated code:

- Define an `interpolate` regex to match expressions that should be interpolated verbatim,
- an `escape` regex to match expressions that should be inserted after being HTML escaped, and
- an `evaluate` regex to match expressions that should be evaluated without insertion into the resulting string.
- You may define or omit any combination of the three.
- For example, to perform `Mustache.js` style templating:

  ```
  _.templateSettings = {
    interpolate: /\{\{(.+?)\}\}/g
  };

  var template = _.template("Hello {{ name }}!");
  template({name: "Mustache"});
  => "Hello Mustache!"
  ```

- escape:

  ```
  > uu.templateSettings.escape = /\{\{-(.*?)\}\}/g
  /\{\{-(.*?)\}\}/g
  > compiled = uu.template("Escaped: {{- value }}\nNot escaped: {{ value }}")
  { [Function]
    source: 'function(obj){\nvar __t,__p=\'\',__j=Array.prototype.join,print=function()
  > compiled({value: 'Hello, <b>world!</b>'})
  'Escaped: Hello, &lt;b&gt;world!&lt;/b&gt;\nNot escaped: {{ value }}'
  ```

- Another example:

```
> uu.templateSettings = {
.....       interpolate: /\<\@\=(.+?)\@\>/gim,
.....       evaluate: /\<\@(.+?)\@\>/gim
.....   }
{ interpolate: /\<\@\=(.+?)\@\>/gim,
  evaluate: /\<\@(.+?)\@\>/gim }
> s = " <@ _.each([0,1,2,3,4], function(i) { @>  <p><@= i @></p> <@ }); @>"
' <@ _.each([0,1,2,3,4], function(i) { @>  <p><@= i @></p> <@ }); @>'
> uu.template(s,{})
'   <p>0</p>   <p>1</p>   <p>2</p>   <p>3</p>   <p>4</p> '
```

By default, template places the values from your data in the local scope via the `with` statement. The `with` statement adds the given object to the head of this scope chain during the evaluation of its statement body:

```
> with (Math) {
... s = PI*2;
... }
6.283185307179586
> z = { x : 1, y : 2 }
{ x: 1, y: 2 }
> with (z) {
... console.log(y);
... }
2
undefined
```

However, you can specify a single variable name with the variable `setting`. This improves the speed at which a template is able to render.

```
_.template("Using 'with': <%= data.answer %>", {answer: 'no'}, {variable: 'data'});
=> "Using 'with': no"
```

- JSFIDDLE: underscore templates
- Stackoverflow::how to use Underscore template

Content delivery network or content distribution network (CDN)

Una CDN que provee `underscore` esta en http://cdnjs.com/:

```
<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.5.2
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
```

A content delivery network or content distribution network (CDN) is a large distributed system of servers deployed in multiple data centers across the Internet. The goal of a CDN is to serve content to end-users with high availability and high performance. CDNs serve a large fraction of the Internet content today, including

- web objects (text, graphics and scripts),
- downloadable objects (media files, software, documents), applications (e-commerce, portals),
- live streaming media, on-demand streaming media, and social networks.

Google provee también un servicio CDN para los desarrolladores en https://developers.google.com/speed/librar

textarea, autofocus y button

1. textarea:

   The `<textarea>` tag defines a multi-line text input control.

   A *text area* can hold an unlimited number of characters, and the text renders in a fixed-width font (usually Courier).

   The size of a text area can be specified by the `cols` and `rows` attributes, or through CSS' `height` and `width` properties.

   `cols` and `rows` consider the font size. `height` and `width` aren't.

2. autofocus.

   The autofocus attribute is a boolean attribute.

   When present, it specifies that the text area should automatically get focus when the page loads.

   Véase también [1]

3. button:

   The `<button>` tag defines a clickable button.

   Inside a `<button>` element you can put content, like text or images.

Local Storage (HTML5 Web Storage)

Web storage and *DOM storage* (document object model) are web application software methods and protocols used for storing data in a web browser.

- Web storage supports persistent data storage, similar to cookies but with a greatly enhanced capacity and no information stored in the HTTP request header.
- Local Storage nos permite almacenar hasta 5MB del lado del cliente por dominio, esto nos permite ahora hacer aplicaciones mas robustas y con mas posibilidades. Las Cookies ofrecen algo parecido, pero con el limite de 100kb.
- There are two main web storage types: *local storage* and *session storage*, behaving similarly to persistent cookies and session cookies respectively.
- Unlike cookies, which can be accessed by both the server and client side, web storage falls exclusively under the purview of client-side scripting
- The HTML5 localStorage object is isolated per domain (the same segregation rules as the same origin policy).

  The same-origin policy permits scripts running on pages originating from the same site – a combination of scheme, hostname, and port number – to access each other's DOM with no specific restrictions, but prevents access to DOM on different sites.

Véase:

- Ejemplo en GitHub: https://github.com/crguezl/web-storage-example

  ```
  [~/javascript/local_storage(master)]$ pwd -P
  /Users/casiano/local/src/javascript/local_storage
  ```

- Como usar localstorage
- HTML5 Web Storage
- W3C Web Storage

- Using HTML5 localStorage To Store JSON Options for persistent storage of complex JavaScript objects in HTML5 by Dan Cruickshank
- HTML5 Cookbook. Christopher Schmitt, Kyle Simpson .ºReilly Media, Inc.", Nov 7, 2011 Chapter 10. Section 2: LocalStorage

While Chrome does not provide a UI for clearing localStorage, there is an API that will either clear a specific key or the entire localStorage object on a website.

```
//Clears the value of MyKey
window.localStorage.clear("MyKey");

//Clears all the local storage data
window.localStorage.clear();
```

Once done, localStorage will be cleared. Note that this affects all web pages on a single domain, so if you clear localStorage for `jsfiddle.net/index.html` (assuming that's the page you're on), then it clears it for all other pages on that site.

global.css

```
html *
{
   font-size: large;
   /* The !important ensures that nothing can override what you've set in this style (unless i
   font-family: Arial;
}

h1            { text-align: center; font-size: x-large; }
th, td        { vertical-align: top; text-align: right; }
/* #finaltable  * { color: white; background-color: black; }   */

/* #finaltable table { border-collapse:collapse; } */
/* #finaltable table, td { border:1px solid white; } */
#finaltable:hover td { background-color: blue; }
tr:nth-child(odd)    { background-color:#eee; }
tr:nth-child(even)    { background-color:#00FF66; }
input        { text-align: right;  border: none;       }     /* Align input to the right  */
textarea     { border: outset; border-color: white;       }
table        { border: inset; border-color: white; }
table.center { margin-left:auto; margin-right:auto; }
#result      { border-color: red; }
tr.error       { background-color: red; }
body
{
 background-color:#b0c4de;  /* blue */
}
```

1. Introducción a las pseudo clases de CSS3

   Una pseudo clase es un estado o uso predefinido de un elemento al que se le puede aplicar un estilo independientemente de su estado por defecto. Existen cuatro tipos diferentes de pseudo clases:

   - Links: Estas pseudo clases se usan para dar estilo al enlace tanto en su estado normal por defecto como cuando ya ha sido visitado, mientras mantenemos el cursor encima de él o cuando hacemos click en él

- Dinamicas: Estas pseudo clases pueden ser aplicadas a cualquier elemento para definir como se muestran cuando el cursor está situado sobre ellos, o haciendo click en ellos o bien cuando son seleccionados
- Estructurales: Permiten dar estilo a elementos basándonos en una posición numérica exacta del elemento
- Otras: Algunos elementos pueden ser estilizados de manera diferente basándonos en el lenguaje o que tipo de etiqueta no son

2. CSS pattern matching

   In CSS, *pattern matching* rules determine which style rules apply to elements in the document tree. These patterns, called selectors, may range from simple element names to rich contextual patterns. If all conditions in the pattern are true for a certain element, the selector matches the element.

   The universal selector, written `*`, matches the name of any element type. It matches any single element in the document tree.

   For example, this rule set will be applied to every element in a document:

   ```
   * {
     margin: 0;
     padding: 0;
   }
   ```

3. CSS class selectors

   Working with HTML, authors may use the period (`.`) notation as an alternative to the `~=` notation when representing the class attribute. Thus, for HTML, `div.value` and `div[class~=value]` have the same meaning. The attribute value must immediately follow the *period* (`.`).

4. CSS3: nth-child() selector The `:nth-child(n)` selector matches every element that is the `n`th child, regardless of type, of its parent.

   `n` can be a number, a keyword, or a formula.

5. The CSS border properties allow you to specify the style and color of an element's border. The border-style property specifies what kind of border to display. For example, `inset:` Defines a 3D inset border while `:outset` defines a 3D outset border. The effect depends on the border-color value

   See CSS: border

6.

csv.js

```
// See http://en.wikipedia.org/wiki/Comma-separated_values
"use strict"; // Use ECMAScript 5 strict mode in browsers that support it

$(document).ready(function() {
   $("button").click(function() {
     calculate();
   });
 });

function calculate() {
  var result;
```

```
  var original       = document.getElementById("original");
  var temp = original.value;
  var regexp = /____________________________________________/g;
  var lines = temp.split(/\n+\s*/);
  var commonLength = NaN;
  var r = [];
  // Template using underscore
  var row = "<%% _.each(items, function(name) { %>"     +
            "                   <td><%%= name %></td>" +
            "             <%% }); %>";

  if (window.localStorage) localStorage.original  = temp;

  for(var t in lines) {
    var temp = lines[t];
    var m = temp.match(regexp);
    var result = [];
    var error = false;

    if (m) {
      if (commonLength && (commonLength != m.length)) {
        //alert('ERROR! row <'+temp+'> has '+m.length+' items!');
        error = true;
      }
      else {
        commonLength = m.length;
        error = false;
      }
      for(var i in m) {
        var removecomma = m[i].replace(/,\s*$/,'');
        var remove1stquote = removecomma.replace(/^\s*"/,'');
        var removelastquote = remove1stquote.replace(/"\s*$/,'');
        var removeescapedquotes = removelastquote.replace(/\"/,'"');
        result.push(removeescapedquotes);
      }
      var tr = error? '<tr class="error">' : '<tr>';
      r.push(tr+_.template(row, {items : result})+"</tr>");
    }
    else {
      alert('ERROR! row '+temp+' does not look as legal CSV');
      error = true;
    }
  }
  r.unshift('<p>\n<table class="center" id="result">');
  r.push('</table>');
  //alert(r.join('\n')); // debug
  finaltable.innerHTML = r.join('\n');
}

window.onload = function() {
  // If the browser supports localStorage and we have some stored data
  if (window.localStorage && localStorage.original) {
    document.getElementById("original").value = localStorage.original;
```

```
  }
};
```

1. Tutorials:Getting Started with jQuery

Tareas

- Añada pruebas usando Mocha y Chai

1.4. Comentarios y Consejos

How can I push a local Git branch to a remote with a different name easily?

```
$ git branch -a
* gh-pages
  remotes/origin/HEAD -> origin/gh-pages
  remotes/origin/gh-pages
```

Of course a solution for this way to work is to rename your master branch:

```
$ git branch -m master gh-pages
[~/Downloads/tmp(gh-pages)]$ git branch
* gh-pages
```

Otherwise, you can do your initial push this way:

```
$ git push -u origin master:gh-pages
```

Option `-u`: for every branch that is up to date or successfully pushed, add `upstream` (tracking) reference, used by argument-less `git-pull`.

- How can I push a local Git branch to a remote with a different name easily?

favicons y shortcut icons

- A *favicon* (short for *Favorite icon*), also known as a *shortcut* icon, is a file containing one or more small icons, most commonly 16×16 pixels, associated with a particular Web site or Web page.
- A web designer can create such an icon and install it into a Web site (or Web page) by several means, and graphical web browsers will then make use of it.
- Browsers that provide favicon support typically display a page's favicon in the browser's address bar (sometimes in the history as well) and next to the page's name in a list of bookmarks.
- Browsers that support a tabbed document interface typically show a page's favicon next to the page's title on the tab
- Some services in the cloud to generate favicons:
  - Favicon Generator
  - favicon.cc
- En `index.html` poner una línea como una de estas:

```
<link rel="shortcut icon" href="etsiiull.png" type="image/x-icon">
<link rel="shortcut icon" href="logo.png" />
<link href="images/favicon.ico" rel="icon" type="image/x-icon" />
```

1.5. Ejercicios

1. Paréntesis:

```
> str = "John Smith"
'John Smith'
> newstr = str.replace(re, "$2, $1")
'Smith, John'
```

2. El método `exec`.

   If your regular expression uses the `g` flag, you can use the `exec` method multiple times to find successive matches in the same string. When you do so, the search starts at the substring of str specified by the regular expression's `lastIndex` property.

```
> re = /d(b+)(d)/ig
/d(b+)(d)/gi
> z = "dBdxdbbdzdbd"
'dBdxdbbdzdbd'
> result = re.exec(z)
[ 'dBd', 'B', 'd', index: 0, input: 'dBdxdbbdzdbd' ]
> re.lastIndex
3
> result = re.exec(z)
[ 'dbbd', 'bb', 'd', index: 4, input: 'dBdxdbbdzdbd' ]
> re.lastIndex
8
> result = re.exec(z)
[ 'dbd', 'b', 'd', index: 9, input: 'dBdxdbbdzdbd' ]
> re.lastIndex
12
> z.length
12
> result = re.exec(z)
null
```

3. JavaScript tiene lookaheads:

```
> x = "hello"
'hello'
> r = /l(?=o)/
/l(?=o)/
> z = r.exec(x)
[ 'l', index: 3, input: 'hello' ]
```

4. JavaScript no tiene lookbehinds:

```
> x = "hello"
'hello'
> r = /(?<=l)l/
SyntaxError: Invalid regular expression: /(?<=l)l/: Invalid group
> .exit
```

```
[~/Dropbox/src/javascript/PLgrado/csv(master)]$ irb
ruby-1.9.2-head :001 > x = "hello"
 => "hello"
ruby-1.9.2-head :002 > r = /(?<=l)l/
 => ll
ruby-1.9.2-head :008 > x =~ r
 => 3
ruby-1.9.2-head :009 > $&
 => "l"
```

5. El siguiente ejemplo comprueba la validez de números de teléfono:

```
[~/local/src/javascript/PLgrado/regexp]$ pwd -P
/Users/casiano/local/src/javascript/PLgrado/regexp
[~/local/src/javascript/PLgrado/regexp]$ cat phone.html
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <meta http-equiv="Content-Script-Type" content="text/javascript">
    <script type="text/javascript">
      var re = /\(?\d{3}\)?([-\/\.])\d{3}\1\d{4}/;
      function testInfo(phoneInput){
        var OK = re.exec(phoneInput.value);
        if (!OK)
          window.alert(RegExp.input + " isn't a phone number with area code!");
        else
          window.alert("Thanks, your phone number is " + OK[0]);
      }
    </script>
  </head>
  <body>
    <p>Enter your phone number (with area code) and then click "Check".
        <br>The expected format is like ###-###-####.</p>
    <form action="#">
      <input id="phone"><button onclick="testInfo(document.getElementById('phone'));">Che
    </form>
  </body>
</html>
```

6. ¿Con que cadenas casa la expresión regular `/^(11+)\1+$/`?

```
> '1111'.match(/^(11+)\1+$/) # 4 unos
[ '1111',
  '11',
  index: 0,
  input: '1111' ]
> '111'.match(/^(11+)\1+$/) # 3 unos
null
> '11111'.match(/^(11+)\1+$/) # 5 unos
null
> '111111'.match(/^(11+)\1+$/) # 6 unos
[ '111111',
  '111',
```

```
  index: 0,
  input: '111111' ]
> '11111111'.match(/^(11+)\1+$/) # 8 unos
[ '11111111',
  '1111',
  index: 0,
  input: '11111111' ]
> '1111111'.match(/^(11+)\1+$/)
null
>
```

Busque una solución al siguiente ejercicio (véase 'Regex to add space after punctuation sign' en PerlMonks) Se quiere poner un espacio en blanco después de la aparición de cada coma:

7.
```
> x = "a,b,c,1,2,d, e,f"
'a,b,c,1,2,d, e,f'
> x.replace(/,/g,", ")
'a, b, c, 1, 2, d,  e, f'
```

pero se quiere que la sustitución no tenga lugar si la coma esta incrustada entre dos dígitos. Además se pide que si hay ya un espacio después de la coma, no se duplique.

a) La siguiente solución logra el segundo objetivo, pero estropea los números:

```
> x = "a,b,c,1,2,d, e,f"
'a,b,c,1,2,d, e,f'
> x.replace(/,(\S)/g,", $1")
'a, b, c, 1, 2, d, e, f'
```

b) Esta otra funciona bien con los números pero no con los espacios ya existentes:

```
> x = "a,b,c,1,2,d, e,f"
'a,b,c,1,2,d, e,f'
> x.replace(/,(\D)/g,", $1")
'a, b, c,1,2, d,  e, f'
```

c) Explique cuando casa esta expresión regular:

```
> r = /(\d[,.]\d)|(,(?=\S))/g
/(\d[,.]\d)|(,(?=\S))/g
```

Aproveche que el método `replace` puede recibir como segundo argumento una función (vea replace):

```
> z = "a,b,1,2,d, 3,4,e"
'a,b,1,2,d, 3,4,e'
> f = function(match, p1, p2, offset, string) { return (p1 || p2 + " "); }
[Function]
> z.replace(r, f)
'a, b, 1,2, d, 3,4, e'
```

1.6. Práctica: Palabras Repetidas

Se trata de producir una salida en las que las palabras repetidas consecutivas sean reducidas a una sola aparición. Rellena las partes que faltan.

Donde

```
[~/srcPLgrado/repeatedwords(master)]$ pwd -P
/Users/casiano/local/src/javascript/PLgrado/repeatedwords
[~/srcPLgrado/repeatedwords(master)]$ git remote -v
origin  ssh://git@bitbucket.org/casiano/pl-grado-repeated-words.git (fetch)
origin  ssh://git@bitbucket.org/casiano/pl-grado-repeated-words.git (push)
```

Véase: https://bitbucket.org/casiano/pl-grado-repeated-words

Ejemplo de ejecución

Estructura

index.html

```
[~/Dropbox/src/javascript/PLgrado/repeatedwords(master)]$ cat index.html
<html>
  <head>
     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
     <title>File Input</title>
     <link href="global.css" rel="stylesheet" type="text/css">

     <script type="text/javascript" src="../../underscore/underscore.js"></script>
     <script type="text/javascript" src="../../jquery/starterkit/jquery.js"></script>
     <script type="text/javascript" src="repeated_words.js"></script>
  </head>
  <body>
    <h1>File Input</h1>
    <input type="file" id="fileinput" />
    <div id="out" class="hidden">
    <table>
      <tr><th>Original</th><th>Transformed</th></tr>
      <tr>
        <td>
          <pre class="input" id="initialinput"></pre>
        </td>
        <td>
          <pre class="output" id="finaloutput"></pre>
        </td>
      </tr>
    </table>
    </div>
  </body>
</html>
```

1. Tag input

global.css

Rellena los estilos para `hidden` y `unhidden`:

```
[~/Dropbox/src/javascript/PLgrado/repeatedwords(master)]$ cat global.css
html *
{
```

```
   font-size: large;
   /* The !important ensures that nothing can override what you've set in this style
      (unless it is also important). */
   font-family: Arial;
}

.thumb {
    height: 75px;
    border: 1px solid #000;
    margin: 10px 5px 0 0;
  }

h1            { text-align: center; font-size: x-large; }
th, td        { vertical-align: top; text-align: right; }
/* #finaltable  * { color: white; background-color: black; }   */

/* #finaltable table { border-collapse:collapse; } */
/* #finaltable table, td { border:1px solid white; } */
#finaltable:hover td { background-color: blue; }
tr:nth-child(odd)    { background-color:#eee; }
tr:nth-child(even)    { background-color:#00FF66; }
input        { text-align: right;  border: none;       }     /* Align input to the right  */
textarea     { border: outset; border-color: white;       }
table        { border: inset; border-color: white; }
.hidden      { display: ____; }
.unhidden    { display: _____; }
table.center { margin-left:auto; margin-right:auto; }
#result      { border-color: red; }
tr.error       { background-color: red; }
pre.output   { background-color: white; }
span.repeated { background-color: red }

body
{
 background-color:#b0c4de;  /* blue */
```

1. CSS display Property
2. Diferencias entre "Displayz "Visibility"

repeated_words.js

Rellena las expresiones regulares que faltan:

```
[~/srcPLgrado/repeatedwords(master)]$ cat repeated_words.js
"use strict"; // Use ECMAScript 5 strict mode in browsers that support it

$(document).ready(function() {
   $("#fileinput").change(calculate);
});

function generateOutput(contents) {
  return contents.replace(/____________________/__,'_________________________________');
}
```

```
function calculate(evt) {
  var f = evt.target.files[0];
  var contents = '';

  if (f) {
    var r = new FileReader();
    r.onload = function(e) {
      contents = e.target.result;
      var escaped  = escapeHtml(contents);
      var outdiv = document.getElementById("out");
      outdiv.className = 'unhidden';
      finaloutput.innerHTML = generateOutput(escaped);
      initialinput.innerHTML = escaped;

    }
    r.readAsText(f);
  } else {
    alert("Failed to load file");
  }
}

var entityMap = {
    "&": "&amp;",
    "<": "&lt;",
    ">": "&gt;",
    '"': '&quot;',
    "'": '&#39;',
    "/": '&#x2F;'
  };

function escapeHtml(string) {
  return String(string).replace(/_________/g, function (s) {
    return ____________;
  });
```

1. jQuery event.target
2. HTML 5 File API
3. HTML 5 File API: FileReader
4. HTML 5 File API: FileReader
5. element.className
6. HTML Entities
7. Tutorials:Getting Started with jQuery
8. Underscore: template

Ficheros de Entrada

```
[~/Dropbox/src/javascript/PLgrado/repeatedwords(master)]$ cat input2.txt
habia una vez
vez un viejo viejo
```

```
hidalgo que vivia
vivia
[~/Dropbox/src/javascript/PLgrado/repeatedwords(master)]$ cat input.txt
one one
nothing rep
is two three
three four
[~/Dropbox/src/javascript/PLgrado/repeatedwords(master)]$ cat inputhtml1.txt
habia => una vez
vez & un viejo viejo <puchum>
hidalgo  & <pacham> que vivia
vivia </que se yo>
```

1.7. Ejercicios

El formato *INI* es un formato estandar para la escritura de ficheros de configuración. Su estrucutra básica se compone de "seccionesz "propiedades". Véase la entrada de la wikipedia INI.

```
; last modified 1 April 2001 by John Doe
[owner]
name=John Doe
organization=Acme Widgets Inc.

[database]
; use IP address in case network name resolution is not working
server=192.0.2.62
port=143
file = "payroll.dat"
```

1. Escriba un programa javascript que obtenga las cabeceras de sección de un fichero INI
2. Escriba un programa javascript que case con los bloques de un fichero INI (cabecera mas lista de pares `parámetro=valor`)
3. Se quieren obtener todos los pares nombre-valor, usando paréntesis con memoria para capturar cada parte.
4. ¿Que casa con cada paréntesis en esta regexp para los pares nombre-valor?

```
> x = "h     = 4"
> r = /([^=]*)(\s*)=(\s*)(.*)/
> r.exec(x)
>
```

1.8. Ejercicios

1. Escriba una expresión regular que reconozca las cadenas de doble comillas. Debe permitir la presencia de comillas y caracteres escapados.
2. ¿Cual es la salida?

```
> "bb".match(/b|bb/)

> "bb".match(/bb|b/)
```

1.9. Práctica: Ficheros INI

Donde

```
[~/srcPLgrado/ini(develop)]$ pwd -P
/Users/casiano/local/src/javascript/PLgrado/ini
[~/srcPLgrado/ini(develop)]$ git remote -v
origin  ssh://git@bitbucket.org/casiano/pl-grado-ini-files.git (fetch)
origin  ssh://git@bitbucket.org/casiano/pl-grado-ini-files.git (push)
```

Véase

- Repositorio conteniendo el código (inicial) del analizador de ficheros `ini`: https://github.com/crguezl/pl-grad
- Despliegue en GitHub pages: http://crguezl.github.io/pl-grado-ini-files/
- Repositorio privado del profesor: https://bitbucket.org/casiano/pl-grado-ini-files/src.

index.html

```
[~/javascript/PLgrado/ini(master)]$ cat index.html
<html>
  <head>
     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
     <title>INI files</title>
     <link href="global.css" rel="stylesheet" type="text/css">
<!--
     <link rel="shortcut icon" href="logo.png" />
-->
     <link rel="shortcut icon" href="etsiiull.png" type="image/x-icon">

     <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/
     <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>

     <script type="text/javascript" src="ini.js"></script>
  </head>
  <body>
    <h1>INI files</h1>
    <input type="file" id="fileinput" />
    <div id="out" class="hidden">
    <table>
      <tr><th>Original</th><th>Tokens</th></tr>
      <tr>
        <td>
          <pre class="input" id="initialinput"></pre>
        </td>
        <td>
          <pre class="output" id="finaloutput"></pre>
        </td>
      </tr>
    </table>
    </div>
  </body>
</html>
```

Ficheros

Vease

- Reading files in JavaScript using the File APIs by Eric Bidelman.

  Source code in files-in-javascript-tut

- W3C File API

- Ejemplo `FileList` en

  - github
  - en acción en gh-pages.
  - Tambien en jsfiddle
  - o bien

    ```
    [~/src/javascript/fileapi/html5rocks]$ pwd -P
    /Users/casiano/local/src/javascript/fileapi/html5rocks
    [~/src/javascript/fileapi/html5rocks(master)]$ ls -l filelist.html
    -rw-r--r--  1 casiano  staff  767 15 feb 17:21 filelist.html
    ```

  - The EventTarget.addEventListener() method

    ```
    target.addEventListener(type, listener[, useCapture]);
    ```

    registers the specified listener on the `EventTarget` it's called on. The event target may be an Element in a document, the Document itself, a Window, or any other object that supports events (such as `XMLHttpRequest`).

  - ```
    > date = new Date(Date.UTC(2012, 11, 12, 3, 0, 0));
    Wed Dec 12 2012 03:00:00 GMT+0000 (WET)
    > date.toLocaleDateString()
    "12/12/2012"
    ```

  - Date.prototype.toLocaleDateString()

- Ejemplo de Drag and Drop en

  - GitHub
  - gh-pages
  - jsfiddle

  o bien en:

  ```
  [~/src/javascript/fileapi/html5rocks]$ pwd -P
  /Users/casiano/local/src/javascript/fileapi/html5rocks
  [~/src/javascript/fileapi/html5rocks]$ ls -l dragandrop.html
  -rw-r--r--  1 casiano  staff  1535 15 feb 18:25 dragandrop.html
  ```

  - `stopPropagation` stops the event from bubbling up the event chain.

    Suppose you have a table and within that table you have an anchor tag. Both the table and the anchor tag have code to handle mouse clicks. When the user clicks on the anchor tag, which HTML element should process the event first? Should it be the table then the anchor tag or vice versa?

    Formally, the event path is broken into three phases.
  ```

  - In the *capture phase*, the event starts at the top of the DOM tree, and propagates through to the parent of the target.
  - In the *target phase*, the event object arrives at its target. This is generally where you will write your event-handling code.
  - In the *bubble phase*, the event will move back up through the tree until it reaches the top. Bubble phase propagation happens in reverse order to the capture phase, with an event starting at the parent of the target and ending up back at the top of the DOM tree.
  - jsfiddle

  These days, there's a choice to register an event in either the capture phase or the bubble phase. If you register an event in the capture phase, the parent element will process the event before the child element.

- `preventDefault` prevents the default action the browser makes on that event.

- After you've obtained a `File` reference, instantiate a `FileReader` object to read its contents into memory.

```
var reader = new FileReader();
```

to read the file we call one of the readAs... For example readAsDataURL is used to starts reading the contents of the specified Blob or File:

```
reader.readAsDataURL(f);
```

- Methods to remember:

  - `FileReader.abort()` Aborts the read operation. Upon return, the `readyState` will be DONE.
  - `FileReader.readAsArrayBuffer()` Starts reading the contents of the specified `Blob`, once finished, the `result` attribute contains an `ArrayBuffer` representing the file's data.
  - `FileReader.readAsBinaryString()` Starts reading the contents of the specified `Blob`, once finished, the result attribute contains the raw binary data from the file as a string.
  - `FileReader.readAsDataURL()` Starts reading the contents of the specified `Blob`.

    When the read operation is finished, the `readyState` becomes `DONE`, and the `loadend` is triggered. At that time, the result attribute contains a URL representing the file's data as base64 encoded string.
  - `FileReader.readAsText()` Starts reading the contents of the specified `Blob`, once finished, the result attribute contains the contents of the file as a text string.

  Once one of these read methods is called on your `FileReader` object, the `onloadstart`, `onprogress`, `onload`, `onabort`, `onerror`, and `onloadend` can be used to track its progress.

- When the load finishes, the reader's `onload` event is fired and its result attribute can be used to access the file data.

```
reader.onload = function(e) {
  var contents = e.target.result;

  ....
}
```

See

- jsfiddle
- GitHub
- gh-pages
- or

  ```
  [~/src/javascript/fileapi/html5rocks]$ pwd -P
  /Users/casiano/local/src/javascript/fileapi/html5rocks
  [~/src/javascript/fileapi/html5rocks]$ ls -l readimages.html
  -rw-r--r--  1 casiano  staff  1530 15 feb 21:00 readimages.html
  ```

- base64 testing image jsfiddle
- The `insertBefore()` method inserts a node as a child, right before an existing child, which you specify. See

  ```
  [~/src/javascript/fileapi/html5rocks]$ ls -l readimages.html
  -rw-r--r--  1 casiano  staff  1530 15 feb 21:00 readimages.html
  ```

global.css

```
[~/javascript/PLgrado/ini(master)]$ cat global.css
html *
{
   font-size: large;
   /* The !important ensures that nothing can override what you've set in this style (unless i
   font-family: Arial;
}

.thumb {
    height: 75px;
    border: 1px solid #000;
    margin: 10px 5px 0 0;
  }

h1            { text-align: center; font-size: x-large; }
th, td        { vertical-align: top; text-align: left; }
/* #finaltable  * { color: white; background-color: black; }   */

/* #finaltable table { border-collapse:collapse; } */
/* #finaltable table, td { border:1px solid white; } */
#finaltable:hover td { background-color: blue; }
tr:nth-child(odd)    { background-color:#eee; }
tr:nth-child(even)    { background-color:#00FF66; }
input        { text-align: right;  border: none;       }     /* Align input to the right  */
textarea     { border: outset; border-color: white;       }
table        { border: inset; border-color: white; }
.hidden      { display: none; }
.unhidden    { display: block; }
table.center { margin-left:auto; margin-right:auto; }
#result      { border-color: red; }
tr.error       { background-color: red; }
```

```
pre.output   { background-color: white; }
/*
span.repeated { background-color: red }
span.header { background-color: blue }
span.comments { background-color: orange }
span.blanks { background-color: green }
span.nameEqualValue { background-color: cyan }
span.error { background-color: red }
*/
body
{
 background-color:#b0c4de;  /* blue */
}
```

Ficheros de Prueba

```
~/Dropbox/src/javascript/PLgrado/ini(master)]$ cat input.ini
; last modified 1 April 2001 by John Doe
[owner]
name=John Doe
organization=Acme Widgets Inc.

[database]
; use IP address in case network name resolution is not working
server=192.0.2.62
port=143
file = "payroll.dat"

$ cat input2.ini
[special_fields]
required = "EmailAddr,FirstName,LastName,Mesg"
csvfile = "contacts.csv"
csvcolumns = "EmailAddr,FirstName,LastName,Mesg,Date,Time"

[email_addresses]
sales = "jack@yahoo.com,mary@my-sales-force.com,president@my-company.com"

$ cat inputerror.ini
[owner]
name=John Doe
organization$Acme Widgets Inc.

[database
; use IP address in case network name resolution is not working
server=192.0.2.62
port=143
file = "payroll.dat"
```

ini.js

```
[~/javascript/PLgrado/ini(master)]$ cat ini.js
"use strict"; // Use ECMAScript 5 strict mode in browsers that support it

$(document).ready(function() {
```

```
   $("#fileinput").change(calculate);
});

function calculate(evt) {
  var f = evt.target.files[0];

  if (f) {
    var r = new FileReader();
    r.onload = function(e) {
      var contents = e.target.result;

      var tokens = lexer(contents);
      var pretty = tokensToString(tokens);

      out.className = 'unhidden';
      initialinput.innerHTML = contents;
      finaloutput.innerHTML = pretty;
    }
    r.readAsText(f);
  } else {
    alert("Failed to load file");
  }
}

var temp = '<li> <span class = "<%= token.type %>"> <%= match %> </span>\n';

function tokensToString(tokens) {
   var r = '';
   for(var i=0; i < tokens.length; i++) {
     var t = tokens[i]
     var s = JSON.stringify(t, undefined, 2);
     s = _.template(temp, {token: t, match: s});
     r += s;
   }
   return '<ol>\n'+r+'</ol>';
}

function lexer(input) {
  var blanks         = /^\s+/;
  var iniheader      = /^\[([^\]\r\n]+)\]/;
  var comments       = /^[;#](.*)/;
  var nameEqualValue = /^([^=;\r\n]+)=([^;\r\n]*)/;
  var any            = /^(.|\n)+/;

  var out = [];
  var m = null;

  while (input != '') {
    if (m = blanks.exec(input)) {
      input = input.substr(m.index+m[0].length);
      out.push({ type : 'blanks', match: m });
    }
    else if (m = iniheader.exec(input)) {
```

```
      input = input.substr(m.index+m[0].length);
      out.push({ type: 'header', match: m });
    }
    else if (m = comments.exec(input)) {
      input = input.substr(m.index+m[0].length);
      out.push({ type: 'comments', match: m });
    }
    else if (m = nameEqualValue.exec(input)) {
      input = input.substr(m.index+m[0].length);
      out.push({ type: 'nameEqualValue', match: m });
    }
    else if (m = any.exec(input)) {
      out.push({ type: 'error', match: m });
      input = '';
    }
    else {
      alert("Fatal Error!"+substr(input,0,20));
      input = '';
    }
  }
  return out;
}
```

Véase la sección *JSON.stringify()* 1.3 para saber mas sobre `JSON.stringify`.

Dudas sobre la Sintáxis del Formato `INI` La sintáxis de `INI` no está bien definida. Se aceptan decisiones razonables para cada una de las expresiones regulares. Si quiere ver un parser en acción puede instalar la gema `inifile` (Ruby).

Una opción que no hemos contemplado en nuestro código es la posibilidad de hacer que una línea de asignación se expanda en varias líneas. En `inifile` el carácter `\` indica que la línea continúa en la siguiente:

```
[~/javascript/PLgrado/inifile(master)]$ cat test/data/good.ini
[section_one]
one = 1
two = 2

[section_two]
three =         3
multi = multiline \
support

; comments should be ignored
[section three]
four   =4
five=5
six =6

[section_four]
   [section_five]
 seven and eight= 7 & 8
```

```
[~/javascript/PLgrado/inifile(master)]$ pry
[2] pry(main)> require 'inifile'
=> true
```

```
[3] pry(main)> p = IniFile.new(:filename => 'test/data/good.ini')
=> #<IniFile:0x007fba2f41a500
 @_line=" seven and eight= 7 & 8",
 @_section={"seven and eight"=>"7 & 8"},
 @comment=";#",
 @content=
  "[section_one]\none = 1\ntwo = 2\n\n[section_two]\nthree =         3\nmulti = multiline \\\n
 @default="global",
 @encoding=nil,
 @escape=true,
 @filename="test/data/good.ini",
 @ini=
  {"section_one"=>{"one"=>"1", "two"=>"2"},
   "section_two"=>{"three"=>"3", "multi"=>"multiline support"},
   "section three"=>{"four"=>"4", "five"=>"5", "six"=>"6"},
   "section_four"=>{},
   "section_five"=>{"seven and eight"=>"7 & 8"}},
 @param="=">

[4] pry(main)> p["section_two"]
=> {"three"=>"3", "multi"=>"multiline support"}
[5] pry(main)> p[:section_two]
```

Tareas

Es conveniente que consiga estos objetivos:

- Pueden comenzar haciendo un fork del repositorio https://github.com/crguezl/pl-grado-ini-files.
- La entrada debería poder leerse desde un fichero. Añada drag and drop.
- Use Web Storage igual que en la anterior
- Escriba las pruebas
- Use templates externos `underscore` para estructurar la salida
- Añada soporte para multilíneas en las asignaciones (Véase la sección 1.9)

```
> s = 'a=b\\\nc'
'a=b\\\nc'
> n2 = /^([^=;#\r\n]+)=((?:[^;#\r\n]*\\\n)*[^;#\r\n]*)/
/^([^=;#\r\n]+)=((?:[^;#\r\n]*\\\n)*[^;#\r\n]*)/
> m = n2.exec(s)
[ 'a=b\\\nc', 'a', 'b\\\nc',
  index: 0, input: 'a=b\\\nc' ]
> d = m[2]
'b\\\nc'
> d.replace(/\\\n/g,' ')
'b c'
```

Véase

1. JSON.stringify
2. www.json.org
3. JSON in JavaScript

4. Underscore: template

5. Stackoverflow::how to use Underscore template

1.10. Práctica: Analizador Léxico para Un Subconjunto de JavaScript

TDOP, Top Down Operator Precedence Vamos a trabajar a partir de este repo de Douglas Crockford:

- https://github.com/douglascrockford/TDOP
- Autor: Douglas Crockford, douglas@crockford.com
- Fecha que figura en el repo: 2010-11-12
- Descripción:
  - tdop.html contains a description of Vaughn Pratt's Top Down Operator Precedence, and describes the parser whose lexer we are going to write in this lab. Is a simplified version of JavaScript.
  - The file `index.html` parses `parse.js` and displays its AST.
  - The page depends on on `parse.js` and `tokens.js`.
  - The file `tdop.js` contains the Simplified JavaScript parser.
  - tokens.js. produces an array of token objects from a string. This is the file we are going to work in this lab.

Objetivos de la Práctica

Douglas Crockford escribió su analizador léxico tokens.js sin usar expresiones regulares. Eso hace que sea extenso (268 líneas). Su analizador es un subconjunto de JS que no tiene - entre otras cosas - expresiones regulares ya que uno de sus objetivos era que el analizador se analizara a si mismo.

Reescriba el analizador léxico en tokens.js. usando expresiones regulares.

1. Evite que se hagan copias de la cadena siendo procesada. Muévase dentro de la misma cadena usando `lastIndex`

2. Añada botones/enlaces/menu de selección que permitan cargar un fichero específico de una lista de ficheros en la `texarea` de entrada.

   Vea el ejemplo en https://github.com/crguezl/loadfileontotexarea.

   En este caso en vez de un fichero `index.html` arrancamos desde un programa Ruby `app.rb`. Para verlo en ejecución instale primero las dependencias:

   ```
   [~/javascript/jquery/loadfileontotexarea(master)]$ bundle install
   Using daemons (1.1.9)
   Using eventmachine (1.0.3)
   Using rack (1.5.2)
   Using rack-protection (1.5.2)
   Using tilt (1.4.1)
   Using sinatra (1.4.4)
   Using thin (1.6.1)
   Using bundler (1.3.5)
   Your bundle is complete!
   Use `bundle show [gemname]` to see where a bundled gem is installed.
   ```

Para ejecutar puede llamar a la aplicación así:

```
[~/javascript/jquery/loadfileontotexarea(master)]$ bundle exec rackup
Thin web server (v1.6.1 codename Death Proof)
Maximum connections set to 1024
Listening on 0.0.0.0:9292, CTRL+C to stop
```

Ahora visite en su navegador la URL `http://localhost:9292`.

Puede ver también la aplicación corriendo en los servidores de Heroku en http://pllexer.herokuapp.com/. Visite los enlaces `withajax.html` y `withget.html`.

3. Añada pruebas
4. Haga el despliegue de su aplicación en Heroku. Para ver como hacerlo siga las indicaciones en la sección *Heroku* 31 en estos apuntes
5. Una primera solución de la que puede partir se encuentra en: https://github.com/crguezl/ull-etsii-grado-pl-mi en github. Veala en funcionamiento en GitHub Pages
6. El método `tokens` retorna el array de tokens. Puede encontrarlo en tokens.js.
7. Mejore la solución en https://github.com/crguezl/ull-etsii-grado-pl-minijavascript/tree/gh-pages
8. Para esta práctica es necesario familiarizarse con la forma en que funciona la OOP en JS. Vea este jsfiddle

Capítulo 2

Expresiones Regulares en C

2.1. Expresiones Regulares Posix en C

Las Expresiones Regulares 'a la Perl' no forman parte de ANSI C. La forma mas sencilla de usar regexps en C es utilizando la versión POSIX de la librería que viene con la mayoría de los Unix. Sigue un ejemplo que usa regex POSIX en C:

```
[~/src/C/regexp]$ cat use_posix.c
#include <stdio.h>
#include <stdlib.h>
#include <regex.h>

int main() {
       regex_t regex;
       int reti;
       char msgbuf[100];

/* Compile regular expression */
        reti = regcomp(&regex, "^a[[:alnum:]]", 0);
        if( reti ){ fprintf(stderr, "Could not compile regex\n"); exit(1); }

/* Execute regular expression */
        reti = regexec(&regex, "abc", 0, NULL, 0);
        if( !reti ){
                puts("Match");
        }
        else if( reti == REG_NOMATCH ){
                puts("No match");
        }
        else{
                regerror(reti, &regex, msgbuf, sizeof(msgbuf));
                fprintf(stderr, "Regex match failed: %s\n", msgbuf);
                exit(1);
        }

/* Free compiled regular expression if you want to use the regex_t again */
    regfree(&regex);
}
```

Compilación y ejecución:

```
[~/src/C/regexp]$ cc use_posix.c  -o use_posix
```

```
[~/src/C/regexp]$ ./use_posix
Match
```

Enlaces Relacionados

1. Regular Expression Matching in GNU C
2. Pattern Matching in GNU C
3. PCRE
4. PCRE doc

2.2. Expresiones Regulares en Flex

Puede encontrar los ejemplos de este capítulo en https://github.com/crguezl/flex-examples.

Un lenguaje regular es aquel que puede ser descrito mediante expresiones regulares como las que se utilizan en `ex`, `vi`, `sed`, `perl` y en tantas otras utilidades UNIX. Dado un lenguaje regular, un analizador léxico es un programa capaz de reconocer las entradas que pertenecen a dicho lenguaje y realizar las acciones semánticas que se hayan asociado con los estados de aceptación. Un generador de analizadores léxicos es una herramienta que facilita la construcción de un analizador léxico. Un generador de analizadores léxicos parte, por tanto, de un lenguaje adecuado para la descripción de lenguajes regulares (y de su semántica) y produce como salida una función (en C, por ejemplo) que materializa el correspondiente analizador léxico. La mayor parte de los generadores producen a partir del conjunto de expresiones regulares los correspondientes tablas de los autómatas finitos deterministas. Utilizando dichas tablas y un algoritmo de simulación genérico del autómata finito determinista se obtiene el analizador léxico. Una vez obtenido el estado de aceptación a partir de la entrada es posible, mediante una sentencia `switch` ejecutar la acción semántica asociada con la correspondiente expresión regular.

2.2.1. Estructura de un programa LEX

Estructura de un programa

`LEX` y `FLEX` son ejemplos de generadores léxicos. `Flex` lee desde la entrada estándar si no se especifica explícitamente un fichero de entrada. El fichero de entrada `reglen.l` (se suele usar el tipo `l`) debe tener la forma:

```
%{
declaration C1
.
.
.

declaration CM
%}
macro_name1 regular_definition1
.
.
.

macro_nameR regular_definitionR

%x exclusive_state
%s inclusive_state
%%
```

```
regular_expression1 { action1(); }
.
.
.

regular_expressionN { actionN(); }

%%
support_routine1() {
}
.
.
.

support_routineS() {
}
```

Como vemos, un programa `LEX` consta de 3 secciones, separadas por `%%`. La primera sección se denomina *sección de definiciones*, la segunda *sección de reglas* y la tercera *sección de código*. La primera y la última son opcionales, así el programa legal `LEX` mas simple es:

```
%%
```

que genera un analizador que copia su entrada en `stdout`.

Compilación
Una vez compilado el fichero de entrada `regleng.l` mediante la correspondiente orden:

`flex reglen.l`

obtenemos un fichero denominado `lex.yy.c`. Este fichero contiene la rutina `yylex()` que realiza el análisis léxico del lenguaje descrito en `regleng.l`. Supuesto que una de las `support_routines` es una función `main()` que llama a la función `yylex()`, podemos compilar el fichero generado con un compilador C para obtener un ejecutable `a.out`:

`cc lex.yy.c -lfl`

La inclusión de la opción `-fl` enlaza con la librería de `flex`, que contiene dos funciones: `main` y `yywrap()`.

Ejecución
Cuando ejecutamos el programa `a.out`, la función `yylex()` analiza las entradas, buscando la secuencia mas larga que casa con alguna de las expresiones regulares (`regular_expressionK`) y ejecuta la correspondiente acción (`actionK()`). Si no se encuentra ningun emparejamiento se ejecuta la *regla por defecto*, que es:

`(.|\n)  { printf("%s",yytext); }`

Si encuentran dos expresiones regulares con las que la cadena mas larga casa, elige la que figura primera en el programa `lex`. Una vez que `yylex()` ha encontrado el *token*, esto es, el patrón que casa con la cadena mas larga, dicha cadena queda disponible a través del puntero global `yytext` , y su longitud queda en la variable entera global `yyleng` .

Una vez que se ha ejecutado la correspondiente acción, `yylex()` continúa con el resto de la entrada, buscando por subsiguientes emparejamientos. Asi continúa hasta encontrar un `end of file`, en cuyo caso termina, retornando un cero o bien hasta que una de las acciones explicitamente ejecuta una sentencia `return`.

Sección de definiciones

La primera sección contiene, si las hubiera, las definiciones regulares y las declaraciones de los estados de arranque.

Las definiciones tiene la forma:

name regular_definition

donde *name* puede ser descrito mediante la expresión regular:

`[a-zA-Z_][a-zA-Z_0-9-]*`

La *regular_definition* comienza en el primer carácter no blanco que sigue a *name* y termina al final de la línea. La definición es una expresión regular extendida. Las subsiguientes definiciones pueden "llamar" a la macro `{name}` escribiéndola entre llaves. La macro se expande entonces a `(regular_definition)` en `flex` y a `regular_definition` en `lex`.

El código entre los delimitadores `%{` y `%}` se copia verbatim al fichero de salida, situándose en la parte de declaraciones globales. Los delimitadores deben aparecer (sólos) al comienzo de la línea.

El Lenguaje de las Expresiones Regulares Flex

La sintáxis que puede utilizarse para la descripción de las expresiones regulares es la que se conoce como "extendida":

- `x` Casa con 'x'
- `.` Cualquier carácter, excepto `\n`.
- `[xyz]` Una "clase"; en este caso una de las letras `x`, `y`, `z`
- `[abj-oZ]` Una "clase" con un rango; casa con `a`, `b`, cualquier letra desde `j` hasta `o`, o una `Z`
- `[^A-Z]` Una "Clase complementada" esto es, todos los caracteres que no están en la clase. Cualquier carácter, excepto las letras mayúsculas. Obsérvese que el retorno de carro `\n` casa con esta expresion. Así es posible que, ¡un patrón como `[^"]+` pueda casar con todo el fichero!.
- `[^A-Z\n]` Cualquier carácter, excepto las letras mayúsculas o un `\n`.
- `[[:alnum:]]` Casa con cualquier caracter alfanumérico. Aqui `[:alnum:]` se refiere a una de las clases predefinidas. Las otras clases son: `[:alpha:]` `[:blank:]` `[:cntrl:]` `[:digit:]` `[:graph:]` `[:lower:]` `[:print:]` `[:punct:]` `[:space:]` `[:upper:]` `[:xdigit:]`. Estas clases designan el mismo conjunto de caracteres que la correspondiente función C `isXXXX`.
- `r*` Cero o mas `r`.
- `r+` Una o mas `r`.
- `r?` Cero o una `r`.
- `r{2,5}` Entre 2 y 5 `r`.
- `r{2,}` 2 o mas `r`. `r{4}` Exactamente 4 `r`.
- `{macro_name}` La expansión de `macro_name` por su *regular_definition*
- `"[xyz]\"foo"` Exactamente la cadena: [xyz]"foo
- `\X` Si `X` is an `a`, `b`, `f`, `n`, `r`, `t`, o `v`, entonces, la interpretación ANSI-C de `\x`. En cualquier otro caso `X`.
- `\0` El carácter NUL (ASCII 0).
- `\123` El carácter cuyo código octal es 123.
- `\x2a` El carácter cuyo código hexadecimal es 2a.

- `(r)` Los paréntesis son utilizados para cambiar la precedencia.
- `rs` Concatenation
- `r|s` Casa con `r` o `s`
- `r/s` Una `r` pero sólo si va seguida de una `s`. El texto casado con `s` se incluye a la hora de decidir cual es el emparejamiento mas largo, pero se devuelve a la entrada cuando se ejecuta la acción. La acción sólo ve el texto asociado con `r`. Este tipo de patrón se denomina *trailing context* o *lookahead* positivo.
- `^r` Casa con `r`, al comienzo de una línea. Un `^` que no aparece al comienzo de la línea o un `$` que no aparece al final de la línea, pierde su naturaleza de "ancla" y es tratado como un carácter ordinario. Asi: `foo|(bar$)` se empareja con `bar$`. Si lo que se quería es la otra interpretación, es posible escribir `foo|(bar\n)`, o bien:

```
foo |
bar$   { /* action */ }
```

- `r$` Casa con `r`, al final de una línea. Este es también un operador de *trailing context*. Una regla no puede tener mas de un operador de *trailing context*. Por ejemplo, la expresión `foo/bar$` es incorrecta.
- <s>r Casa con `r`, pero sólo si se está en el estado `s`.
- <s1,s2,s3>r Idem, si se esta en alguno de los estados `s1`, `s2`, or `s3`
- <*>r Casa con `r` cualquiera que sea el estado, incluso si este es exclusivo.
- <<EOF>> Un final de fichero.
- <s1,s2><<EOF>> Un final de fichero, si los estados son `s1` o `s2`

Los operadores han sido listados en orden de precedencia, de la mas alta a la mas baja. Por ejemplo `foo|bar+` es lo mismo que `(foo)|(ba(r)+)`.

Las Acciones Semánticas
Cada patrón regular tiene su correspondiente acción asociada. El patrón termina en el primer espacio en blanco (sin contar aquellos que están entre comillas dobles o prefijados de secuencias de escape). Si la acción comienza con `{`, entonces se puede extender a través de multiples líneas, hasta la correspondiente `}`. El programa `flex` no hace un análisis del código C dentro de la acción. Existen tres directivas que pueden insertarse dentro de las acciones: `BEGIN`, `ECHO` y `REJECT`. Su uso se muestra en los subsiguientes ejemplos.

La sección de código se copia verbatim en `lex.yy.c`. Es utilizada para proveer las funciones de apoyo que se requieran para la descripción de las acciones asociadas con los patrones que parecen en la sección de reglas.

2.2.2. Versión Utilizada

Todos los ejemplos que aparecen en este documento fueron preparados con la versión 2.5.4 de *flex* en un entorno Linux

```
$ uname -a
Linux nereida.deioc.ull.es 2.2.12-20 #10 Mon May 8 19:40:16 WEST 2000 i686 unknown
$ flex --version
flex version 2.5.4
```

y con la versión 2.5.2 en un entorno Solaris

```
> uname -a
SunOS fonil 5.7 Generic_106541-04 sun4u sparc SUNW,Ultra-5_10
> flex --version
flex version 2.5.2
```

2.2.3. Espacios en blanco dentro de la expresión regular

La expresión regular va desde el comienzo de la línea hasta el primer espacio en blanco no escapado. Todos los espacios en blanco que formen parte de la expresión regular deben ser escapados o protegidos entre comillas. Así, el siguiente programa produce un error en tiempo de compilación C:

```
> cat spaces.l
%%
one two { printf("spaces\n"; }
%%
nereida:~/public_html/regexpr/lex/src> flex spaces.l
nereida:~/public_html/regexpr/lex/src> gcc lex.yy.c
spaces.l: In function ‘yylex’:
spaces.l:2: ‘two’ undeclared (first use in this function)
spaces.l:2: (Each undeclared identifier is reported only once
spaces.l:2: for each function it appears in.)
spaces.l:2: parse error before ‘{’
spaces.l:4: case label not within a switch statement
lex.yy.c:632: case label not within a switch statement
lex.yy.c:635: case label not within a switch statement
lex.yy.c:757: default label not within a switch statement
lex.yy.c: At top level:
lex.yy.c:762: parse error before ‘}’
```

Deberíamos escapar el blanco entre `one` y `two` o bien proteger la cadena poniéndola entre comillas: `"one two"`.

2.2.4. Ejemplo Simple

Este primer ejemplo sustituye las apariciones de la palabra *username* por el *login* del usuario:

```
$ cat subst.l
%option main
%{
#include <unistd.h>
%}
%%
username    printf( "%s",  getlogin());
%%
$ flex -osubst.c subst.l
$ gcc -o subst subst.c
$ subst
Dear username:
Dear pl:
```

He presionado CTRL-D para finalizar la entrada.
Observe el uso de la opción `%option main` en el fichero *subst.l* para hacer que *flex* genere una función *main*. También merece especial atención el uso de la opción `-osubst` para cambiar el nombre del fichero de salida, que por defecto será *lex.yy.c*.

2.2.5. Suprimir

Al igual que en *sed* y *awk*, es muy sencillo suprimir las apariciones de una expresión regular.

```
$ cat delete.l
/* delete all entries of zap me */
%%
"zap me"
$ flex delete.l ; gcc lex.yy.c -lfl; a.out
this is zap me a first zap me phrase
this is  a first  phrase
```

2.2.6. Declaración de yytext

En la sección de definiciones es posible utilizar las directivas `%pointer` o `%array`. Estas directivas hacen que `yytext` se declare como un puntero o un *array* respectivamente. La opción por defecto es declararlo como un puntero, salvo que se haya usado la opción `-l` en la línea de comandos, para garantizar una mayor compatibilidad con `LEX`. Sin embargo, y aunque la opción `%pointer` es la mas eficiente (el análisis es mas rápido y se evitan los *buffer overflow*), limita la posible manipulación de *yytext* y de las llamadas a `unput()`.

```
$ cat yytextp.l
%%
hello {
        strcat(yytext, " world");
        printf("\n%d: %s\n",strlen(yytext),yytext);
      }
$ flex yytextp.l ; gcc lex.yy.c -lfl ; a.out
hello

11: hello world

fatal flex scanner internal error--end of buffer missed
```

Este error no aparece si se utiliza la opción `%array`:

```
$ cat yytext.l
%array
%%
hello {
        strcat(yytext, " world");
        printf("\n%d: %s\n",strlen(yytext),yytext);
      }
$ flex yytext.l; gcc lex.yy.c -lfl; a.out
hello

11: hello world
```

Además, algunos programs `LEX` modifican directamente `yytext`, utilizando la declaración:
`extern char yytext[]`
que es incompatible con la directiva `%pointer` (pero correcta con `%array`). La directiva `%array` define `yytext` como un *array* de tamaño `YYLMAX`. Si deseamos trabajar con un mayor tamaño, basta con redefinir `YYLMAX`.

2.2.7. Declaración de yylex()

Por defecto la función *yylex()* que realiza el análisis léxico es declarada como `int yylex()`. Es posible cambiar la declaración por defecto utilizando la macro *YY_DECL*. En el siguiente ejemplo la definición:

```
#define YY_DECL char *scanner(int *numcount, int *idcount)
```

hace que la rutina de análisis léxico pase a llamarse *scanner* y tenga dos parametros de entrada, retornando un valor de tipo `char *`.

```
$ cat decl.l
%{
#define YY_DECL char *scanner(int *numcount, int *idcount)
%}

num [0-9]+
id [a-z]+
%%
{num} {(*numcount)++;}
halt  {return ((char *) strdup(yytext));}
{id}  {(*idcount)++;}
%%
main() {
  int a,b;
  char *t;

  a = 0; b = 0;
  t = scanner(&a, &b);
  printf("numcount = %d, idcount = %d, yytext = %s\n",a,b,t);
  t = scanner(&a, &b);
  printf("numcount = %d, idcount = %d, yytext = %s\n",a,b,t);
}

int yywrap() {
  return 1;
}
```

La ejecución del programa anterior produce la siguiente salida:

```
$ decl
a b 1 2 3 halt
     numcount = 3, idcount = 2, yytext = halt

e 4 5 f

numcount = 5, idcount = 4, yytext = (null)
$ decl
a b 1 2 3 halt
     numcount = 3, idcount = 2, yytext = halt

e 4 f 5 halt
    numcount = 5, idcount = 4, yytext = halt
```

2.2.8. yywrap()

Cuando el analizador léxico alcanza el final del fichero, el comportamiento en las subsiguientes llamadas a *yylex* resulta indefinido. En el momento en que *yylex()* alcanza el final del fichero llama a la función *yywrap*, la cual retorna un valor de 0 o 1 según haya mas entrada o no. Si el valor es 0, la función *yylex* asume que la propia *yywrap* se ha encargado de abrir el nuevo fichero y asignarselo a `yyin` . Otra manera de continuar es haciendo uso de la función *yyrestart(FILE *file)*. El siguiente ejemplo cuenta el número de líneas, palabras y caracteres en una lista de ficheros proporcionados como entrada.

```
%{
unsigned long charCount = 0, wordCount = 0, lineCount = 0;
%}

word [^ \t\n]+
eol \n

%%
{word} { wordCount++; charCount += yyleng; }
{eol} { charCount++; lineCount++; }
. charCount++;

%%

char **fileList;
unsigned nFiles;
unsigned currentFile = 0;
unsigned long totalCC = 0;
unsigned long totalWC = 0;
unsigned long totalLC = 0;

main ( int argc, char **argv) {
  FILE *file;

  fileList = argv + 1; nFiles = argc - 1;

  if (nFiles == 0) {
    fprintf(stderr,"Usage is:\n%s file1 file2 file3 ...\n",argv[0]);
    exit(1);
  }
  file = fopen (fileList[0], "r");
  if (!file) {
      fprintf (stderr, "could not open %s\n", argv[1]);
      exit (1);
  }
  currentFile = 1; yyrestart(file);
  yylex ();
  printf ("%8lu %8lu %8lu %s\n", lineCount, wordCount,
    charCount, fileList[currentFile - 1]);
  if (argc > 2) {
      totalCC += charCount; totalWC += wordCount; totalLC += lineCount;
      printf ("%8lu %8lu %8lu total\n", totalLC, totalWC, totalCC);
  }
  return 0;
```

```
}

int yywrap () {
  FILE *file;

  if (currentFile < nFiles) {
     printf ("%8lu %8lu %8lu %s\n", lineCount, wordCount,
       charCount, fileList[currentFile - 1]);
     totalCC += charCount; totalWC += wordCount; totalLC += lineCount;
     charCount = wordCount = lineCount = 0;
     fclose (yyin);

     while (fileList[currentFile] != (char *) 0) {
       file = fopen (fileList[currentFile++], "r");
       if (file != NULL) { yyrestart(file); break; }
  fprintf (stderr, "could not open %s\n", fileList[currentFile - 1]);
     }
     return (file ? 0 : 1);
    }
    return 1;
}
```

La figura muestra el proceso de compilación y la ejecución:

```
$ flex countlwc.l;gcc lex.yy.c; a.out *.l
      58       179      1067 ape-05.l
      88       249      1759 countlwc.l
      11        21       126 magic.l
       9        17       139 mgrep.l
       9        16       135 mlg.l
       5        15       181 ml.l
       7        12        87 subst.l
     187       509      3494 total
```

La diferencia esencial entre asignar *yyin* o llamar a la función *yyrestart* es que esta última puede ser utilizada para conmutar entre ficheros en medio de un análisis léxico. El funcionamiento del programa anterior no se modifica si se se intercambian asignaciones a *yyin* (*yyin = file*) y llamadas a *yyrestart(file)*.

2.2.9. unput()

La función *unput(c)* coloca el carácter *c* en el flujo de entrada, de manera que será el primer carácter leído en próxima ocasión.

```
$ cat unput2.l
%array
%%
[a-z] {unput(toupper(yytext[0]));}
[A-Z] ECHO;
%%
$ flex unput2.l ; gcc lex.yy.c -lfl;a.out
abcd
ABCD
```

Un problema importante con *unput* es que, cuando se utiliza la opción `%pointer`, las llamadas a *unput* destruyen los contenidos de *yytext*. Es por eso que, en el siguiente ejemplo se hace una copia de *yytext*. La otra alternativa es, por supuesto, usar la opción `%array`.

```
$ cat unput.l
%%
[0-9]+ {
  int i;
  char *yycopy = (char *) strdup(yytext);

  unput(')');
  for(i=strlen(yycopy)-1; i>=0; --i)
    unput(yycopy[i]);
  unput('(');
  free(yycopy);
}
\([0-9]+\) printf("Num inside parenthesis: %s\n",yytext);
.|\n
$ flex unput.l ; gcc lex.yy.c -lfl ; a.out
32
Num inside parenthesis: (32)
(43)
Num inside parenthesis: (43)
```

2.2.10. input()

La función *input()* lee desde el flujo de entrada el siguiente carácter. Normalmente la utilizaremos si queremos tomar "personalmente el control" del análisis. El ejemplo permite "engullir" los comentarios (no anidados):

```
$ cat input.l
%%
"/*" {
        int c;
        for(;;) {
          while ((c=input()) != '*' && c != EOF)
            ;
          if (c == '*')  {
            while ((c = input()) == '*')
              ;
            if (c == '/') break;
          }
          if (c == EOF) {
            fprintf(stderr,"Error: EOF in comment");
            yyterminate();
          }
        }
      }
```

La función `yyterminate()` termina la rutina de análisis léxico y devuelve un cero indicándole a la rutina que llama que todo se ha acabado. Por defecto, `yyterminate()` es llamada cuando se encuentra un final de fichero. Es una macro y puede ser redefinida.

```
$ flex input.l ; gcc lex.yy.c -lfl ; a.out
hello /* world */
hello
unfinished /* comment
unfinished Error: EOF in comment
```

He presionado CTRL-D después de entrar la palabra *comment*.

2.2.11. REJECT

La directiva `REJECT` le indica al analizador que proceda con la siguiente regla que casa con un prefijo de la entrada. Como es habitual en *flex*, se elige la siguiente regla que casa con la cadena mas larga. Consideremos el siguiente ejemplo:

```
$ cat reject.l
%%
a    |
ab   |
abc  |
abcd ECHO; REJECT; printf("Never seen\n");
.|\n
```

La salida es:

```
$ gcc lex.yy.c -lfl;a.out
abcd
abcdabcaba
```

Observe que `REJECT` supone un cambio en el flujo de control: El código que figura después de `REJECT` no es ejecutado.

2.2.12. yymore()

La función `yymore()` hace que, en vez de vaciar *yytext* para el siguiente *matching*, el valor actual se mantenga, concatenando el valor actual de *yytext* con el siguiente:

```
$ cat yymore.l
%%
mega- ECHO; yymore();
kludge ECHO;

$ flex yymore.l ; gcc lex.yy.c -lfl ; a.out
mega-kludge
mega-mega-kludge
```

La variable `yyleng` no debería ser modificada si se hace uso de la función `yymore()`.

2.2.13. yyless()

La función `yyless(n)` permite retrasar el puntero de lectura de manera que apunta al carácter n de *yytext*. Veamos un ejemplo:

```
$ cat yyless.l
%%
foobar ECHO; yyless(4);
[a-z]+ ECHO;

$ flex yyless.l; gcc lex.yy.c -lfl; a.out
foobar
foobarar
```

Veamos un ejemplo mas "real". supongamos que tenemos que reconocer las cadenas entre comillas dobles, pero que pueden aparecer en las mismas secuencias de escape `\"`. La estrategia general del algoritmo es utilizar la expresión regular `\"[^"]+\"` y examinar si los dos últimos carácteres en `yytext` son `\"`. En tal caso, se concatena la cadena actual (sin la `"` final) como prefijo para el próximo emparejamiento (utilizando `yymore`). La eliminación de la `"` se hace a través de la ejecución de `yyless(yyleng-1)`, que al mismo tiempo garantiza que el próximo emparejamiento tendrá lugar con este mismo patrón `\"[^"]+\"`.

```
$ cat quotes.l
%%
\"[^"]+\" {
            printf("Processing string. %d: %s\n",yyleng,yytext);
            if (yytext[yyleng-2] =='\\') {
             yyless(yyleng-1); /* so that it will match next time */
             yymore(); /* concatenate  with current yytext */
             printf("After yyless. %d: %s\n",yyleng,yytext);
            } else {
              printf("Finished. The string is: %s\n",yytext);
            }
          }
```

El ejemplo no puede entenderse si no se tiene en cuenta que `yyless(yyleng-1)` actualiza los valores de *yyleng* y *yytext*, como muestra la salida.
¿Qué ocurre si intercambiamos las posiciones de `yymore()` e `yyless(yyleng-1)` en el código? ¿Cambiara la salida? La respuesta es que no. Parece que la concatenación se hace con el valor final de *yytext* y no con el valor que este tenía en el momento de la llamada a *yymore*.
Otra observación a tener en cuenta es que *yyless()* es una macro y que, por tanto, sólo puede ser utilizada dentro del fichero *lex* y no en otros ficheros fuentes.

En general, el uso de estas funciones nos puede resolver el problema de reconocer límites que de otra forma serían difíciles de expresar con una expresión regular.

```
$ flex quotes.l ; gcc lex.yy.c -lfl ; a.out
"Hello \"Peter\", nice to meet you"
Procesing string. 9: "Hello \"
After yyless. 8: "Hello \
Procesing string. 16: "Hello \"Peter\"
After yyless. 15: "Hello \"Peter\
Procesing string. 35: "Hello \"Peter\", nice to meet you"
Finished. The string is: "Hello \"Peter\", nice to meet you"
```

2.2.14. Estados

Las expresiones regulares pueden ser prefijadas mediante *estados*. Los estados o condiciones de arranque, se denotan mediante un identificador entre ángulos y se declaran en la parte de las definiciones. Las declaraciones se hacen mediante `%s` para los estados "inclusivos" o bien `%x` para los estados "exclusivos", seguidos de los nombres de los estados. No pueden haber caracteres en blanco antes de la declaración. Un *estado* se activa mediante la acción BEGIN *estado*. A partir de ese momento, las reglas que esten prefijadas con el estado pasan a estar activas. En el caso de que el estado sea inclusivo, las reglas no prefijadas también permanecen activas. Los estados exclusivos son especialmente útiles para especificar "sub analizadores" que analizan porciones de la entrada cuya estructura "sintáctica" es diferente de la del resto de la entrada.

El ejemplo "absorbe" los comentarios, conservando el numero de líneas del fichero en la variable `linenum`

```
$ cat comments.l
%option noyywrap
%{
  int linenum = 0;
%}
%x comment
%%

"/*" BEGIN(comment); printf("comment=%d, YY_START = %d, YYSTATE = %d",comment,YY_START,YYSTATE
```

```
<comment>[^*\n]* /* eat anything that is not a star * /
<comment>"*"+[^*/\n]* /* eat up starts not followed by / */
<comment>\n ++linenum; /* update number of lines */
<comment>"*"+"/" BEGIN(0);

\n ECHO; linenum++;
.  ECHO;
%%
main() {
  yylex();
  printf("\n%d lines\n",linenum);
}
```

La opción `noyywrap` hace que `yylex()` no llame a la función `yywrap()` al final del fichero y que asuma que no hay mas entrada por procesar.

Los estados se traducen por enteros, pudiendo ser manipulados como tales. La macro `INITIAL` puede utilizarse para referirse al estado 0. Las macros `YY_START` y `YYSTATE` contienen el valor del estado actual.

```
$ flex comments.l ; gcc lex.yy.c ; a.out < hello.c
main() <%
int a<:1:>; comment=1, YY_START = 1, YYSTATE = 1
  a<:0:> = 4; comment=1, YY_START = 1, YYSTATE = 1
  printf("hello world! a(0) is %d\n",a<:0:>);
%>

6 lines
$ cat hello.c
main() <%
int a<:1:>; /* a comment */
  a<:0:> = 4; /* a comment in
                 two lines */
  printf("hello world! a(0) is %d\n",a<:0:>);
%>
```

En *flex* es posible asociar un ámbito con los estados o condiciones iniciales. Basta con colocar entre llaves las parejas *patrón acción* gobernadas por ese estado. El siguiente ejemplo procesa las cadenas *C*:

```
$ cat ststring.l
%option main
%x str
%{
#define MAX_STR_CONST 256

  char string_buffer[MAX_STR_CONST];
  char *string_buf_ptr;
%}

%%
\"  string_buf_ptr  = string_buffer; BEGIN(str);
<str>{
\"             {BEGIN (INITIAL); *string_buf_ptr = '\0'; printf("%s",string_buffer); }
\n             { printf("Error: non terminated string\n"); exit(1); }
\\[0-7]{1,3}   { int result; /* octal escape sequence */
```

```
                        (void) sscanf(yytext+1,"%o",&result);
                         if (result > 0xff) {printf("Error: constant out of bounds\n"); exit(2
                         *string_buf_ptr++ = result;
                    }
\\[0-9]+        { printf("Error: bad escape sequence\n"); exit(2); }
\\n             {*string_buf_ptr++ = '\n';}
\\t             {*string_buf_ptr++ = '\t';}
\\b             {*string_buf_ptr++ = '\b';}
\\r             {*string_buf_ptr++ = '\r';}
\\f             {*string_buf_ptr++ = '\f';}
\\(.|\n)        {*string_buf_ptr++ = yytext[1];}
[^\\\n\"]+      {char *yptr = yytext; while(*yptr) *string_buf_ptr++ = *yptr++; }
}
(.|\n)
%%
$ flex ststring.l ; gcc lex.yy.c ; a.out < hello.c
        hello
world! a(0) is %d
$ cat hello.c
main() <%
int a<:1:>; /* a comment */
  a<:0:> = 4; /* a comment in
                 two lines */
  printf("\thell\157\nworld! a(0) is %d\n",a<:0:>);
%>
```

Obsérve la conducta del programa ante las siguientes entradas:

- Entrada:

  ```
  "hello \
  dolly"
  ```

  ¿Cuál será la salida? ¿Que patrón del programa anterior es el que casa aqui?

- Entrada: `"hello\ndolly"`. ¿Cuál será la salida? ¿Que patrón del programa anterior es el que casa aqui?

- ```
  |
  "hello
  ```

  Donde hay un retorno del carro después de `hello`. ¿Cuál será la salida?

2.2.15. La pila de estados

Mediante el uso de la opción
`%option stack`
tendremos acceso a una pila de estados y a tres rutinas para manipularla:

- `void yy_push_state(int new_state)`
  Empuja el estado actual y bifurca a `new_state`.

- `void yy_pop_state()`
  Saca el estado en el *top* de la pila y bifurca a el mismo.
  ```

- `int yy_top_state()`
  Nos devuelve el estado en el *top* de la pila, sin alterar los contenidos de la misma.

Ejemplo

El siguiente programa `flex` utiliza las funciones de la pila de estados para reconocer el lenguaje (no regular) $\{a^n b^n \ / \ n \in N\}$

```
%option main
%option noyywrap
%option stack
%{
#include <stdio.h>
#include  <stdlib.h>
%}
%x estado_a
%%
^a { yy_push_state(estado_a);}
<estado_a>{
a      { yy_push_state(estado_a); }
b      { yy_pop_state();  }
b[^b\n]+    {   printf ("Error\n");
        while (YYSTATE != INITIAL)
          yy_pop_state();
        while (input() != '\n') ;
      }
(.|\n)  {  printf ("Error\n");
        while (YYSTATE != INITIAL)
          yy_pop_state();
        while (input() != '\n') ;
      }
}
.      {  printf ("Error\n");
        while (input() != '\n') ;
      }
\n      { printf("Aceptar\n");}
%%
```

2.2.16. Final de Fichero

El patrón `<<EOF>>` permite asociar acciones que se deban ejecutar cuando se ha encontrado un *end of file* y la macro `yywrap()` ha devuelto un valor no nulo.

Cualquiera que sea, la acción asociada deberá de optar por una de estas cuatro alternativas:

- Asignar `yyin` a un nuevo fichero de entrada
- Ejecutar `return`
- Ejecutar `yyterminate()` (véase la sección 2.2.10)
- Cambiar de *buffer* de entrada utilizando la función `yy_switch_buffer` (véase la sección 2.2.21).

El patrón `<<EOF>>` no puede usarse con otras expresiones regulares. Sin embargo, es correcto prefijarlo con estados.

Si `<<EOF>>` aparece sin condiciones de arranque, la regla se aplica a todos los estados que no tienen una regla `<<EOF>>` específica. Si lo que se quiere es que la regla se restringa al ámbito del estado inicial se deberá escribir:

<INITIAL><<EOF>>

Sigue un programa que reconoce los comentarios anidados en C. Para detectar comentarios incacabados usaremos <<EOF>>.

```
%option stack
%x comment
%%
"/*"   { yy_push_state(comment); }
(.|\n) ECHO;
<comment>"/*"     { yy_push_state(comment); }
<comment>"*/"     { yy_pop_state(); }
<comment>(.|\n)   ;
<comment><<EOF>>  { fprintf(stderr,"Error\n"); exit(1); }
%%
```

```
$ cat hello.c
main() {
int a[1]; /* a /* nested comment */. */
  a[0] = 4; /* a  /* nested comment in
                 /* two */ lines  */ *****/
}
$ flex nestedcom.l ; gcc lex.yy.c -lfl ; a.out < hello.c
main() {
int a[1];
  a[0] = 4;
}
$ cat hello4.c
main() {
int a[1]; /* a /* nested comment */. */
  a[0] = 4; /* an  /* incorrectly nested comment in
                 /* two  lines  */ *****/
}
$ a.out < hello4.c
main() {
int a[1];
Error
  a[0] = 4;
```

2.2.17. Uso de Dos Analizadores

La opción `-Pprefix` de flex cambia el prefijo por defecto `yy` para todas las variables globales y funciones. Por ejemplo `-Pfoo` cambia el nombre de `yytext footext`. También cambia el nombre del fichero de salida de `lex.yy.c` a `lex.foo.c`. Sigue la lista de identificadores afectados:

```
yy_create_buffer
yy_delete_buffer
yy_flex_debug
yy_init_buffer
yy_flush_buffer
yy_load_buffer_state
yy_switch_to_buffer
yyin
yyleng
yylex
yylineno
```

```
yyout
yyrestart
yytext
yywrap
```

Desde dentro del analizador léxico puedes referirte a las variables globales y funciones por cualquiera de los nombres, pero externamente tienen el nombre cambiado. Esta opción nos permite enlazar diferentes programas flex en un mismo ejecutable.

Sigue un ejemplo de uso de dos analizadores léxicos dentro del mismo programa:

```
$ cat one.l
%%
one {printf("1\n"); return 1;}
.   {printf("First analyzer: %s\n",yytext);}
%%

int onewrap(void) {
  return 1;
}

$ cat two.l
%%
two {printf("2\n"); return 2;}
.   {printf("Second analyzer: %s\n",yytext);}
%%
int twowrap(void) {
  return 1;
}
$ cat onetwo.c
main() {
  onelex();
  twolex();
}
```

Como hemos mencionado, la compilación *flex* se debe realizar con el opción `-P`, que cambia el prefijo por defecto `yy` de las funciones y variables accesibles por el usuario. El mismo efecto puede conseguirse utilizando la opción `prefix`, escribiendo `%option prefix="one"` y `%option prefix="two"` en los respectivos programas `one.l` y `two.l`.

```
$ flex -Pone one.l
$ flex -Ptwo two.l
$ ls -ltr | tail -2
-rw-rw----   1 pl       casiano     36537 Nov  7 09:52 lex.one.c
-rw-rw----   1 pl       casiano     36524 Nov  7 09:52 lex.two.c
$ gcc onetwo.c lex.one.c lex.two.c
$ a.out
two
First analyzer: t
First analyzer: w
First analyzer: o

one
1
one
Second analyzer: o
```

```
Second analyzer: n
Second analyzer: e

two
2
$
```

2.2.18. La Opción `outfile`

Es posible utilizar la opción `-ooutput.c` para escribir el analizador léxico en el fichero `output.c` en vez de en `lex.yy.c`. El mismo efecto puede obtenerse usando la opción `outfile="output.c"` dentro del programa `lex`.

2.2.19. Leer desde una Cadena: YY_INPUT

En general, la rutina que hace el análisis léxico, *yylex()*, lee su entrada a través de la macro `YY_INPUT`. Esta macro es llamada con tres parámetros

`YY_INPUT(buf,result,max)`

el primero, *buf* es utilizado para guardar la entrada. el tercero *max* indica el número de caracteres que *yylex()* pretende leer de la entrada. El segundo *result* contendrá el número de caracteres realmente leídos. Para poder leer desde una cadena (*string*) basta con modificar `YY_INPUT` para que copie los datos de la cadena en el *buffer* pasado como parámetro a `YY_INPUT`. Sigue un ejemplo:

```
$ cat string.l
%{
#undef YY_INPUT
#define YY_INPUT(b,r,m) (r = yystringinput(b,m))
#define min(a,b) ((a<b)?(a):(b))
%}

%%
[0-9]+ printf("Num-");
[a-zA-Z][a-zA-Z_0-9]* printf("Id-");
[ \t]+
. printf("%c-",yytext[0]);
%%

extern char string[];
extern char *yyinputptr;
extern char *yyinputlim;

int yystringinput(char *buf, int maxsize) {
  int n = min(maxsize, yyinputlim-yyinputptr);

  if (n > 0) {
    memcpy(buf, yyinputptr, n);
    yyinputptr += n;
  }
  return n;
}

int yywrap() { return 1; }
```

Este es el fichero conteniendo la función *main*:

```
$ cat stringmain.c
char string[] = "one=1;two=2";
char *yyinputptr;
char *yyinputlim;

main() {
  yyinputptr = string;
  yyinputlim = string + strlen(string);
  yylex();
  printf("\n");
}
```

Y esta es la salida:

```
$ a.out
Id-=-Num-;-Id-=-Num-
```

La cadena `string = "one=1;two=2"` definida en la línea 2 ha sido utilizada como entrada para el análisis léxico.

2.2.20. El operador de "trailing context" o "lookahead" positivo

En el lenguaje FORTRAN original los "blancos" no eran significativos y no se distinguía entre mayúsculas y minúsculas. Así pues la cadena `do i = 1, 10` es equivalente a la cadena `DOI=1,10`. Un conocido conflicto ocurre entre una cadena con la estructura `do i = 1.10` (esto es `DOI=1.10`) y la cadena anterior. En la primera `DO` e `I` son dos "tokens" diferentes, el primero correspondiendo a la palabra reservada que indica un bucle. En la segunda, `DOI` constituye un único "token" y la sentencia se refiere a una asignación. El conflicto puede resolverse utilizando el operador de "trailing" `r/s`. Como se mencionó, el operador de "trailing"`r/s` permite reconocer una `r` pero sólo si va seguida de una `s`. El texto casado con `s` se incluye a la hora de decidir cual es el emparejamiento mas largo, pero se devuelve a la entrada cuando se ejecuta la acción. La acción sólo ve el texto asociado con `r`. El fichero `fortran4.l` ilustra una posible solución:

```
 cat fortran4.l
%array
%{
#include <string.h>
#undef YY_INPUT
#define YY_INPUT(buf,result,max) (result = my_input(buf,max))
%}
number [0-9]+
integer [+-]?{number}
float ({integer}\.{number}?|\.{number})(E{integer})?
label [A-Z0-9]+
id   [A-Z]{label}*
%%
DO/{label}={number}\, { printf("do loop\n"); }
{id} { printf("Identifier %s\n",yytext); }
{number} { printf("Num %d\n",atoi(yytext)); }
{float} { printf("Float %f\n",atof(yytext)); }
(.|\n)
%%

int my_input(char *buf, int max)
{
```

```
  char *q1, *q2, *p = (char *) malloc(max);
  int i;
  if ('\0' != fgets(p,max,yyin)) {
    for(i=0, q1=buf, q2=p;(*q2 != '\0');q2++) {
      if (*q2 != ' ') { *q1++ = toupper(*q2); i++; };
    }
    free(p);
    return i;
  }
  else exit(1);
}
```

La función

`char *fgets(char *s, int size, FILE *stream)`

lee a lo mas uno menos que `size` caracteres desde `stream` y los almacena en el *buffer* apuntado por `s`. La lectura termina después de un `EOF` o un retorno de carro. Si se lee un `\n`, se almacena en el *buffer*. La función pone un carácter nulo `\0` como último carácter en el *buffer*.

A continuación, puedes ver los detalles de una ejecución:

```
$ flex fortran4.l; gcc lex.yy.c -lfl; a.out
do j = 1 . 10
Identifier DOJ
Float 1.100000
do k = 1, 5
do loop
Identifier K
Num 1
Num 5
```

2.2.21. Manejo de directivas `include`

El analisis léxico de algunos lenguajes requiere que, durante la ejecución, se realice la lectura desde diferentes ficheros de entrada. El ejemplo típico es el manejo de las directivas *include file* existentes en la mayoría de los lenguajes de programación.
¿Donde está el problema? La dificultad reside en que los analizadores generados por *flex* proveen almacenamiento intermedio (*buffers*) para aumentar el rendimiento. No basta con reescribir nuestro propio *YY_INPUT* de manera que tenga en cuenta con que fichero se esta trabajando. El analizador sólo llama a *YY_INPUT* cuando alcanza el final de su *buffer*, lo cual puede ocurrir bastante después de haber encontrado la sentencia *include* que requiere el cambio de fichero de entrada.

```
$ cat include.l
%x incl
%{
#define yywrap() 1
#define MAX_INCLUDE_DEPTH 10
YY_BUFFER_STATE include_stack[MAX_INCLUDE_DEPTH];
int include_stack_ptr = 0;
%}
%%
include          BEGIN(incl);
.                ECHO;
<incl>[ \t]*
<incl>[^ \t\n]+  { /* got the file name */
                   if (include_stack_ptr >= MAX_INCLUDE_DEPTH) {
                     fprintf(stderr,"Includes nested too deeply\n");
```

```
                    exit(1);
                  }
                  include_stack[include_stack_ptr++] = YY_CURRENT_BUFFER;
                  yyin = fopen(yytext,"r");
                  if (!yyin) {
                    fprintf(stderr,"File %s not found\n",yytext);
                    exit(1);
                  }
                  yy_switch_to_buffer(yy_create_buffer(yyin, YY_BUF_SIZE));
                  BEGIN(INITIAL);
                }
<<EOF>> {
          if ( --include_stack_ptr < 0) {
            yyterminate();
          } else {
            yy_delete_buffer(YY_CURRENT_BUFFER);
            yy_switch_to_buffer(include_stack[include_stack_ptr]);
          }
        }
%%
main(int argc, char ** argv) {

  yyin = fopen(argv[1],"r");
  yylex();
}
```

La función `yy_create_buffer(yyin, YY_BUF_SIZE));` crea un *buffer* lo suficientemente grande para mantener `YY_BUF_SIZE` caracteres. Devuelve un `YY_BUFFER_STATE`, que puede ser pasado a otras rutinas. `YY_BUFFER_STATE` es un puntero a una estructura de datos opaca (`struct yy_buffer_state`) que contiene la información para la manipulación del *buffer*. Es posible por tanto inicializar un puntero `YY_BUFFER_STATE` usando la expresión `((YY_BUFFER_STATE) 0)`.

La función `yy_switch_to_buffer(YY_BUFFER_STATE new_buffer);` conmuta la entrada del analizador léxico. La función `void yy_delete_buffer( YY_BUFFER_STATE buffer )` se usa para recuperar la memoria consumida por un *buffer*. También se pueden limpiar los contenidos actuales de un buffer llamando a: `void yy_flush_buffer( YY_BUFFER_STATE buffer )`

La regla especial `<<EOF>>` indica la acción a ejecutar cuando se ha encontrado un final de fichero e `yywrap()` retorna un valor distinto de cero. Cualquiera que sea la acción asociada, esta debe terminar con uno de estos cuatro supuestos:

1. Asignar `yyin` a un nuevo fichero de entrada.
2. Ejecutar `return`.
3. Ejecutar `yyterminate()`.
4. Cambiar a un nuevo buffer usando `yy_switch_to_buffer()`.

   La regla `<<EOF>>` no se puede mezclar con otros patrones.

Este es el resultado de una ejecución del programa:

```
$ cat hello.c
#include hello2.c
main() <%
int a<:1:>; /* a comment */
  a<:0:> = 4; /* a comment in
                 two lines */
```

```
  printf("\thell\157\nworld! a(0) is %d\n",a<:0:>);
%>
$ cat hello2.c
#include hello3.c
/* file hello2.c  */
$ cat hello3.c
/*
third file
*/
$ flex include.l ; gcc lex.yy.c ; a.out hello.c
##/*
third file
*/

/* file hello2.c  */

main() <%
int a<:1:>; /* a comment */
  a<:0:> = 4; /* a comment in
                 two lines */
  printf("\thell\157\nworld! a(0) is %d\n",a<:0:>);
%>
```

Una alternativa a usar el patrón `<<EOF>>` es dejar la responsabilidad de recuperar el *buffer* anterior a `yywrap()`. En tal caso suprimiríamos esta parajea patrón-acción y reescribiríamos `yywrap()`:

```
%x incl
%{
#define MAX_INCLUDE_DEPTH 10
YY_BUFFER_STATE include_stack[MAX_INCLUDE_DEPTH];
int include_stack_ptr = 0;
%}
%%
include          BEGIN(incl);
.                ECHO;
<incl>[ \t]*
<incl>[^ \t\n]+  { /* got the file name */
                   if (include_stack_ptr >= MAX_INCLUDE_DEPTH) {
     fprintf(stderr,"Includes nested too deeply\n");
     exit(1);
   }
   include_stack[include_stack_ptr++] = YY_CURRENT_BUFFER;
   yyin = fopen(yytext,"r");
   if (!yyin) {
     fprintf(stderr,"File %s not found\n",yytext);
     exit(1);
   }
   yy_switch_to_buffer(yy_create_buffer(yyin, YY_BUF_SIZE));
   BEGIN(INITIAL);
 }
%%
main(int argc, char ** argv) {

  yyin = fopen(argv[1],"r");
```

```
  yylex();
}

int yywrap() {
  if ( --include_stack_ptr < 0) {
    return 1;
  } else {
    yy_delete_buffer(YY_CURRENT_BUFFER);
    yy_switch_to_buffer(include_stack[include_stack_ptr]);
    return 0;
  }
}
```

2.2.22. Análisis Léxico desde una Cadena: `yy_scan_string`

El objetivo de este ejercicio es mostrar como realizar un análisis léxico de los argumentos pasados en la línea de comandos. Para ello *flex* provee la función `yy_scan_string(const char * str)`. Esta rutina crea un nuevo *buffer* de entrada y devuelve el correspondiente manejador `YY_BUFFER_STATE` asociado con la cadena `str`. Esta cadena debe estar terminada por un carácter `\0`. Podemos liberar la memoria asociada con dicho *buffer* utilizando `yy_delete_buffer(BUFFER)`. La siguiente llamada a `yylex()` realizará el análisis léxico de la cadena `str`.

```
$ cat scan_str.l
%%
[0-9]+     printf("num\n");
[a-zA-Z]+  printf("Id\n");
%%
main(int argc, char ** argv) {
int i;

  for(i=1;i<argc;i++) {
    yy_scan_string(argv[i]);
    yylex();
    yy_delete_buffer(YY_CURRENT_BUFFER);
  }
}

int yywrap() { return 1; }
$ flex scan_str.l ; gcc lex.yy.c ; a.out Hello World! 1234
Id
Id
!num
```

Alternativamente, la función `main()` podría haber sido escrita asi:

```
main(int argc, char ** argv) {
int i;
YY_BUFFER_STATE p;

  for(i=1;i<argc;i++) {
    p = yy_scan_string(argv[i]);
    yylex();
    yy_delete_buffer(p);
  }
}
```

La función `yy_scan_bytes(const char * bytes, int len)` hace lo mismo que `yy_scan_string` pero en vez de una cadena terminada en el carácter nulo, se usa la longitud `len`. Ambas funciones `yy_scan_string(const char * str)` y `yy_scan_bytes(const char * bytes, int len)` hacen una copia de la cadena pasada como argumento.

Estas dos funciones crean una copia de la cadena original. Es mejor que sea asi, ya que `yylex()` modifica los contenidos del *buffer* de trabajo. Si queremos evitar la copia, podemos usar

`yy_scan_buffer(char *base, yy_size_t size)`,

la cual trabaja directamente con el *buffer* que comienza en `base`, de tamaño `size` *bytes*, los últimos dos de los cuáles deben ser `YY_END_OF_BUFFER_CHAR` (ASCII NUL). Estos dos últimos *bytes* no son "escaneados". El área de rastreo va desde `base[0]` a `base[size-2]`, inclusive. Si nos olvidamos de hacerlo de este modo y no establecemos los dos *bytes* finales, la función `yy_scan_buffer()` devuelve un puntero nulo y no llega a crear el nuevo buffer de entrada. El tipo `yy_size_t` es un tipo entero. Como cabe esperar, `size` se refiere al tamaño del *buffer*.

2.2.23. Análisis de la Línea de Comandos y 2 Analizadores

El objetivo de este ejercicio es mostrar como realizar un análisis léxico de los argumentos pasados en la línea de comandos. Para ello diseñaremos una librería que proporcionará un función `yylexarg(argc,argv)` que hace el análisis de la línea de acuerdo con la especificación *flex* correspondiente. En el ejemplo, esta descripción del analizador léxico es proporcionada en el fichero *fl.l*. Para complicar un poco mas las cosas, supondremos que queremos hacer el análisis léxico de un fichero (especificado en la línea de comandos) según se especifica en un segundo analizador léxico *trivial.l*. El siguiente ejemplo de ejecución muestra la conducta del programa:

```
$ fl -v -V -f tokens.h
verbose mode is on
version 1.0
File name is: tokens.h
Analyzing tokens.h
#-id-blanks-id-blanks-int-blanks-#-id-blanks-id-blanks-int-blanks-#-id-blanks-id-blanks
-int-blanks-#-id-blanks-id-blanks-int-blanks-#-id-blanks-id-blanks-int-blanks-
```

Los contenidos del fichero *Makefile* definen las dependencias y la estructura de la aplicación:

```
$ cat Makefile
LIBS=-lflarg
CC=gcc -g
LIBPATH=-L. -L~/lib
INCLUDES=-I. -I~/include

fl: main.c lex.arg.c lex.yy.c libflarg.a tokens.h
        $(CC) $(LIBPATH) $(INCLUDES) main.c lex.arg.c lex.yy.c $(LIBS) -o fl
lex.arg.c: fl.l
        flex -Parg fl.l
lex.yy.c: trivial.l tokens.h
        flex trivial.l
libflarg.a: flarg.o
        ar r libflarg.a flarg.o
flarg.o: flarg.c
        $(CC) -c flarg.c
clean:
$ make clean;make
rm lex.arg.c lex.yy.c *.o  fl
flex -Parg fl.l
flex trivial.l
```

```
gcc -g -c flarg.c
ar r libflarg.a flarg.o
gcc -g -L. -L~/lib -I. -I~/include main.c lex.arg.c lex.yy.c -lflarg -o fl
```

Observa el uso de la opción `-Parg` en la traducción del fichero *fl.l*. Así no solo el fichero generado por *flex*, sino todas las variables y rutinas accesibles estarán prefijadas por *arg* en vez de *yy*. La librería la denominamos *libflarg.a*. (*flarg* por flex arguments). El correspondiente fichero cabecera será *flarg.h*. Los fuentes de las rutinas que compondrán la librería se mantendrán en el fichero *flarg.c*.

Lo que haremos será redefinir `YY_INPUT(buf, result, max)` para que lea su entrada desde la línea de argumentos.

```
$ cat flarg.h
int yyarglex(int argc, char **argv);
int YY_input_from_argv(char *buf, int max);
int argwrap(void);

#undef YY_INPUT
#define YY_INPUT(buf,result,max) (result = YY_input_from_argv(buf,max))
```

La función `int YY_input_from_argv(char *buf, int max)` utiliza los punteros `char **YY_targv` y `char **YY_arglim` para moverse a través de la familia de argumentos. Mientras que el primero es utilizado para el recorrido, el segundo marca el límite final. Su inicialización ocurre en

`yyarglex(int argc, char **argv)`

con las asignaciones:

```
  YY_targv = argv+1;
  YY_arglim = argv+argc;
```

despues, de lo cual, se llama al analizador léxico generado, *arglex* .

```
$ cat flarg.c
char **YY_targv;
char **YY_arglim;

int YY_input_from_argv(char *buf, int max)
{
  static unsigned offset = 0;

  int len, copylen;

    if (YY_targv >= YY_arglim) return 0;        /* EOF */
    len = strlen(*YY_targv)-offset;     /* amount of current arg */
    if(len >= max) {copylen = max-1; offset += copylen; }
    else copylen = len;
    if(len > 0) memcpy(buf, YY_targv[0]+offset, copylen);
    if(YY_targv[0][offset+copylen] == '\0') {   /* end of arg */
      buf[copylen] = ' '; copylen++; offset = 0; YY_targv++;
    }
    return copylen;
}

int yyarglex(int argc, char **argv) {
  YY_targv = argv+1;
  YY_arglim = argv+argc;
```

```
  return arglex();
}

int argwrap(void) {
  return 1;
}
```

El fichero *fl.l* contiene el analizador léxico de la línea de comandos:

```
$ cat fl.l
%{
unsigned verbose;
unsigned thereisfile;
char *progName;
char fileName[256];
#include "flarg.h"
#include "tokens.h"
%}

%%
-h      |
"-?"    |
-help   { printf("usage is: %s [-help | -h | -? ] [-verbose | -v]"
         " [-Version | -V]"
         " [-f filename]\n", progName);
        }

-v      |
-verbose { printf("verbose mode is on\n"); verbose = 1; }

-V      |
-version { printf("version 1.0\n"); }

-f[[:blank:]]+[^ \t\n]+ {
              strcpy(fileName,argtext+3);
              printf("File name is: %s\n",fileName);
              thereisfile = 1;
            }
.

\n
```

Observe el uso de la clase `[:blank:]` para reconocer los blancos. Las clases son las mismas que las introducidas en *gawk*.

El análisis léxico del fichero que se lee después de procesar la línea de comandos es descrito en *trivial.l*. Partiendo de *trivial.l*, la ejecución del *Makefile* da lugar a la construcción por parte de *flex* del fichero *lex.yy.c* conteniendo la rutina *yylex*.

```
$ cat trivial.l
%{
#include "tokens.h"
%}
digit [0-9]
id [a-zA-Z][a-zA-Z0-9]+
blanks [ \t\n]+
```

```
operator [+*/-]
%%
{digit}+ {return INTTOKEN; }
{digit}+"."{digit}+ {return FLOATTOKEN; }
{id} {return IDTOKEN;}
{operator} {return OPERATORTOKEN;}
{blanks} {return BLANKTOKEN;}
. {return (int) yytext[0];}
%%
int yywrap() {
  return 1;
}
```

El fichero *tokens.h* contiene la definición de los *tokens* y es compartido con *main.c*.

```
$ cat tokens.h
#define INTTOKEN  256
#define FLOATTOKEN 257
#define IDTOKEN 258
#define OPERATORTOKEN 259
#define BLANKTOKEN 260
```

Nos queda por presentar el fichero *main.c*:

```
$ cat main.c
#include <stdio.h>
#include "flarg.h"
#include "tokens.h"
extern unsigned verbose;
extern unsigned thereisfile;
extern char *progName;
extern char fileName[256];
extern FILE * yyin;

main(int argc, char **argv) {
  unsigned lookahead;
  FILE * file;

  progName = *argv;
  yyarglex(argc,argv);
  if (thereisfile) {
    if (verbose) printf("Analyzing %s\n",fileName);
    file = (fopen(fileName,"r"));
    if (file == NULL) exit(1);
    yyin = file;
    while (lookahead = yylex()) {
      switch (lookahead) {
        case INTTOKEN:
            printf("int-");
            break;
          case FLOATTOKEN:
            printf("float-");
            break;
          case IDTOKEN:
            printf("id-");
```

```
          break;
        case OPERATORTOKEN:
          printf("operator-");
          break;
        case BLANKTOKEN:
          printf("blanks-");
          break;
        default: printf("%c-",lookahead);
      }
    } /* while */
    printf("\n");
  } /* if */
}
```

2.2.24. Declaraciones pointer y array

Como se comentó, las opciones `%pointer` y `%array` controlan la definición que *flex* hace de *yytext*. en el caso en que eligamos la opción `%array` la variable `YYLMAX` controla el tamaño del *array*. Supongamos que en el fichero *trivial.l* del ejemplo anterior introducimos las siguientes modificaciones:

```
$ cat trivial.l
%array
%{
#undef YYLMAX
#define YYLMAX 4
#include "tokens.h"
%}
digit [0-9]
id [a-zA-Z][a-zA-Z0-9]+
blanks [ \t\n]+
operator [+*/-]
%%
{digit}+ {return INTTOKEN; }
{digit}+"."{digit}+ {return FLOATTOKEN; }
{id} {return IDTOKEN;}
{operator} {return OPERATORTOKEN;}
{blanks} {return BLANKTOKEN;}
. {return (int) yytext[0];}
%%
int yywrap() {
  return 1;
  }
```

En tal caso, la definición excesivamente pequeña de YYLMAX provoca un error en tiempo de ejecución:

```
$ fl -V -f tokens.h
version 1.0
File name is: tokens.h
token too large, exceeds YYLMAX
```

2.2.25. Las Macros `YY_USER_ACTION`, `yy_act` e `YY_NUM_RULES`

La macro `YY_USER_ACTION` permite ejecutar una acción inmediatamente después del "emparejamiento" y antes de la ejecución de la acción asociada. cuando se la invoca, la variable `yy_act` contiene el número de la regla que ha emparejado (las reglas se numeran a partir de uno). La macro `YY_NUM_RULES` contiene el número de reglas, incluyendo la regla por defecto.

El siguiente programa aprovecha dichas macros para mostrar las frecuencias de uso de las reglas.

```
$ cat user_action.l
%array
%{
#include <string.h>

int ctrl[YY_NUM_RULES];
#undef YY_USER_ACTION
#define YY_USER_ACTION { ++ctrl[yy_act]; }
%}
number [0-9]+
id   [a-zA-Z_]+[a-zA-Z0-9_]*
whites [ \t\n]+
%%
{id}
{number}
{whites}
.
%%

int yywrap() {
  int i;

  for(i=1;i<YY_NUM_RULES;i++)
    printf("Rule %d: %d occurrences\n",i,ctrl[i]);
}

$ flex user_action.l ; gcc lex.yy.c -lfl ; a.out
a=b+2*(c-4)
Rule 1: 3 occurrences
Rule 2: 2 occurrences
Rule 3: 1 occurrences
Rule 4: 6 occurrences
```

2.2.26. Las opciones `interactive`

La opción `option always-interactive` hace que `flex` genere un analizador que considera que su entrada es "interactiva". Concretamente, el analizador para cada nuevo fichero de entrada, intenta determinar si se trata de un a entrada interactiva o desde fichero haciendo una llamada a la función `isatty()`. Vea un ejemplo de uso de esta función:

```
$ cat isatty.c
#include <unistd.h>
#include <stdio.h>
main() {

  if (isatty(0))
    printf("interactive\n");
  else
    printf("non interactive\n");
}
$ gcc isatty.c; a.out
interactive
```

```
$ a.out < isatty.c
non interactive
$
```

cuando se usa la opción `option always-interactive`, se elimina esta llamada.

2.2.27. La macro YY_BREAK

Las acciones asociadas con los patrones se agrupan en la rutina de análisis léxico `yylex()` en una sentencia `switch` y se separan mediante llamadas a la macro `YY_BREAK`. Asi, al compilar con `flex` el siguiente fichero `.l`

```
$ cat interactive.l
%%
. printf("::%c",yytext[0]);
\n printf("::%c",yytext[0]);
```

tenemos el fichero de salida `lex.yy.c` que aparece a continuación (hemos omitido las líneas de código en las que estamos menos interesados, sustituyendolas por puntos suspensivos)

```
/* A lexical scanner generated by flex */
....
#define YY_NUM_RULES 3
#line 1 "interactive.l"
#define INITIAL 0
#line 363 "lex.yy.c"
....
YY_DECL {
 ....
#line 1 "interactive.l"
#line 516 "lex.yy.c"
 ....
 if ( yy_init ) {
   yy_init = 0;
#ifdef YY_USER_INIT
   YY_USER_INIT;
#endif
   if ( ! yy_start ) yy_start = 1;  /* first start state */
   if ( ! yyin ) yyin = stdin;
   if ( ! yyout ) yyout = stdout;
   if ( ! yy_current_buffer ) yy_current_buffer = yy_create_buffer( yyin, YY_BUF_SIZE );
   yy_load_buffer_state();
 }
 while ( 1 )    /* loops until end-of-file is reached */ {
   ...........................
yy_match:
   do {
    .....
   }
   ...........
yy_find_action:
   ...........
   YY_DO_BEFORE_ACTION;
   do_action:  /* This label is used only to access EOF actions. */
     switch ( yy_act ) { /* beginning of action switch */
```

```
          case 0:
            ..................
            goto yy_find_action;
        case 1:
        YY_RULE_SETUP
        #line 2 "interactive.l"
        printf("::%c",yytext[0]);
          YY_BREAK
        case 2:
        YY_RULE_SETUP
        #line 3 "interactive.l"
        printf("::%c",yytext[0]);
          YY_BREAK
        case 3:
        YY_RULE_SETUP
        #line 4 "interactive.l"
        ECHO;
          YY_BREAK
        #line 614 "lex.yy.c"
        case YY_STATE_EOF(INITIAL):
          yyterminate();
          case YY_END_OF_BUFFER:
            { .....  }
        default:
          YY_FATAL_ERROR("fatal flex scanner internal error--no action found");
  } /* end of action switch */
 } /* end of scanning one token */
} /* end of yylex */

#if YY_MAIN
int main()
  { yylex(); return 0; }
#endif
#line 4 "interactive.l"
```

Por defecto, la macro `YY_BREAK` es simplemente un `break`. Si cada acción de usuario termina en un `return`, puedes encontrarte con que el compilador genera un buen número de `warning! unreachable code`. Puedes entonces redefinir `YY_BREAK` a vacío y evitar estos mensajes.

Capítulo 3

Expresiones Regulares en Perl

3.1. Introducción

Los rudimentos de las expresiones regulares pueden encontrarse en los trabajos pioneros de McCullogh y Pitts (1940) sobre redes neuronales. El lógico Stephen Kleene definió formalmente el algebra que denominó *conjuntos regulares* y desarrollo una notación para la descripción de dichos conjuntos, las *expresiones regulares*.

Durante las décadas de 1960 y 1970 hubo un desarrollo formal de las expresiones regulares. Una de las priemras publicaciones que utilizan las expresiones regulares en un marco informático es el artículo de 1968 de Ken Thompson *Regular Expression Search Algorithm* en el que describe un compilador de expresiones regulares que produce código objeto para un IBM 7094. Este compilador dió lugar al editor *qed*, en el cual se basó el editor de Unix *ed*. Aunque las expresiones regulares de este último no eran tan sofisticadas como las de *qed*, fueron las primeras en ser utilizadas en un contexto no académico. Se dice que el comando global `g` en su formato `g/re/p` que utilizaba para imprimir (opción `p`) las líneas que casan con la expresión regular `re` dió lugar a un programa separado al que se denomino `grep`.

Las expresiones regulares facilitadas por las primeras versiones de estas herramientas eran limitadas. Por ejemplo, se disponía del cierre de Kleene `*` pero no del cierre positivo `+` o del operador opcional `?`. Por eso, posteriormente, se han introducido los metacaracteres `\+` y `\?`. Existían numerosas limitaciones en dichas versiones, por ej. `$` sólo significa "final de línea" al final de la expresión regular. Eso dificulta expresiones como

```
grep 'cierre$\|^Las' viq.tex
```

Sin embargo, la mayor parte de las versiones actuales resuelven correctamente estos problemas:

```
nereida:~/viq> grep 'cierre$\|^Las' viq.tex
Las expresiones regulares facilitadas por las primeras versiones de estas herramientas
eran limitadas. Por ejemplo, se disponía del cierre de Kleene \verb|*| pero no del cierre
nereida:~/viq>
```

De hecho AT&T Bell labs añadió numerosas funcionalidades, como por ejemplo, el uso de `\{min, max\}`, tomada de *lex*. Por esa época, Alfred Aho escribió *egrep* que, no sólo proporciona un conjunto mas rico de operadores sino que mejoró la implementación. Mientras que el *grep* de Ken Thompson usaba un autómata finito no determinista (NFA), la versión de *egrep* de Aho usa un autómata finito determinista (DFA).

En 1986 Henry Spencer desarrolló la librería *regex* para el lenguaje `C`, que proporciona un conjunto consistente de funciones que permiten el manejo de expresiones regulares. Esta librería ha contribuido a "homogeneizar" la sintáxis y semántica de las diferentes herramientas que utilizan expresiones regulares (como *awk*, *lex*, *sed*, ...).

Véase También

- La sección *Expresiones Regulares en Otros Lenguajes* 3.3

- Regular Expressions Cookbook. Jan Goyvaerts, Steven Levithan
- PCRE (Perl Compatible Regular Expressions) en la Wikipedia
- PCRE (Perl Compatible Regular Expressions)
- Java Regular Expressions
- C# Regular Expressions
- .NET Framework Regular Expressions

3.1.1. Un ejemplo sencillo

Matching en Contexto Escalar

```
pl@nereida:~/Lperltesting$ cat -n c2f.pl
  1    #!/usr/bin/perl -w
  2    use strict;
  3
  4    print "Enter a temperature (i.e. 32F, 100C):\n";
  5    my $input = <STDIN>;
  6    chomp($input);
  7
  8    if ($input !~ m/^([-+]?[0-9]+(\.[0-9]*)?)\s*([CF])$/i) {
  9      warn "Expecting a temperature, so don't understand \"$input\".\n";
 10    }
 11    else {
 12      my $InputNum = $1;
 13      my $type = $3;
 14      my ($celsius, $farenheit);
 15      if ($type eq "C" or $type eq "c") {
 16        $celsius = $InputNum;
 17        $farenheit = ($celsius * 9/5)+32;
 18      }
 19      else {
 20        $farenheit = $InputNum;
 21        $celsius = ($farenheit -32)*5/9;
 22      }
 23      printf "%.2f C = %.2f F\n", $celsius, $farenheit;
 24    }
```

Véase también:

- `perldoc perlrequick`
- `perldoc perlretut`
- `perldoc perlre`
- `perldoc perlreref`

Ejecución con el depurador:

```
pl@nereida:~/Lperltesting$ perl -wd c2f.pl
Loading DB routines from perl5db.pl version 1.28
Editor support available.
```

```
Enter h or `h h' for help, or `man perldebug' for more help.
main::(c2f.pl:4):       print "Enter a temperature (i.e. 32F, 100C):\n";
DB<1> c 8
Enter a temperature (i.e. 32F, 100C):
32F
main::(c2f.pl:8):       if ($input !~ m/^([-+]?[0-9]+(\.[0-9]*)?)\s*([CF])$/i) {
DB<2> n
main::(c2f.pl:12):        my $InputNum = $1;
DB<2> x ($1, $2, $3)
0  32
1  undef
2  'F'
DB<3> use YAPE::Regex::Explain
DB<4> p YAPE::Regex::Explain->new('([-+]?[0-9]+(\.[0-9]*)?)\s*([CF])$')->explain
The regular expression:
(?-imsx:([-+]?[0-9]+(\.[0-9]*)?)\s*([CF])$)
matches as follows:

NODE                     EXPLANATION
----------------------------------------------------------------------
(?-imsx:                 group, but do not capture (case-sensitive)
                         (with ^ and $ matching normally) (with . not
                         matching \n) (matching whitespace and #
                         normally):
----------------------------------------------------------------------
  (                        group and capture to \1:
----------------------------------------------------------------------
    [-+]?                    any character of: '-', '+' (optional
                             (matching the most amount possible))
----------------------------------------------------------------------
    [0-9]+                   any character of: '0' to '9' (1 or more
                             times (matching the most amount
                             possible))
----------------------------------------------------------------------
    (                        group and capture to \2 (optional
                             (matching the most amount possible)):
----------------------------------------------------------------------
      \.                       '.'
----------------------------------------------------------------------
      [0-9]*                   any character of: '0' to '9' (0 or
                               more times (matching the most amount
                               possible))
----------------------------------------------------------------------
    )?                       end of \2 (NOTE: because you're using a
                             quantifier on this capture, only the
                             LAST repetition of the captured pattern
                             will be stored in \2)
----------------------------------------------------------------------
  )                        end of \1
----------------------------------------------------------------------
  \s*                      whitespace (\n, \r, \t, \f, and " ") (0 or
                           more times (matching the most amount
                           possible))
```

```
----------------------------------------------------------------------
  (                          group and capture to \3:
----------------------------------------------------------------------
    [CF]                       any character of: 'C', 'F'
----------------------------------------------------------------------
  )                          end of \3
----------------------------------------------------------------------
  $                          before an optional \n, and the end of the
                             string
----------------------------------------------------------------------
)                          end of grouping
----------------------------------------------------------------------
```

Dentro de una expresión regular es necesario referirse a los textos que casan con el primer, paréntesis, segundo, etc. como `\1`, `\2`, etc. La notación `$1` se refieré a lo que casó con el primer paréntesis en el último *matching*, no en el actual. Veamos un ejemplo:

```
pl@nereida:~/Lperltesting$ cat -n dollar1slash1.pl
   1    #!/usr/bin/perl -w
   2    use strict;
   3
   4    my $a = "hola juanito";
   5    my $b = "adios anita";
   6
   7    $a =~ /(ani)/;
   8    $b =~ s/(adios) *($1)/\U$1 $2/;
   9    print "$b\n";
```

Observe como el `$1` que aparece en la cadena de reemplazo (línea 8) se refiere a la cadena `adios` mientras que el `$1` en la primera parte contiene `ani`:

```
pl@nereida:~/Lperltesting$ ./dollar1slash1.pl
ADIOS ANIta
```

Ejercicio 3.1.1. *Indique cuál es la salida del programa anterior si se sustituye la línea 8 por*

```
$b =~ s/(adios) *(\1)/\U$1 $2/;
```

Número de Paréntesis

El número de paréntesis con memoria no está limitado:

```
pl@nereida:~/Lperltesting$ perl -wde 0
main::(-e:1):   0
            123456789ABCDEF
DB<1> $x = "123456789AAAAAA"
                   1  2  3  4  5  6  7  8  9 10 11  12
DB<2> $r = $x =~ /(.)(.)(.)(.)(.)(.)(.)(.)(.)(.)(.)\11/; print "$r\n$10\n$11\n"
1
A
A
```

Véase el siguiente párrafo de `perlre` (sección Capture buffers):

> *There is no limit to the number of captured substrings that you may use. However Perl also uses* `\10`*,* `\11`*, etc. as aliases for* `\010`*,* `\011`*, etc. (Recall that* `0` *means octal, so* `\011` *is the character at number* `9` *in your coded character set; which would be the 10th*

character, a horizontal tab under ASCII.) Perl resolves this ambiguity by interpreting `\10` *as a backreference only if at least* `10` *left parentheses have opened before it. Likewise* `\11` *is a backreference only if at least* `11` *left parentheses have opened before it. And so on.* `\1` *through* `\9` *are always interpreted as backreferences.*

Contexto de Lista

Si se utiliza en un contexto que requiere una lista, el "pattern match" retorna una lista consistente en las subexpresiones casadas mediante los paréntesis, esto es `$1`, `$2`, `$3`, ... Si no hubiera emparejamiento se retorna la lista vacía. Si lo hubiera pero no hubieran paréntesis se retorna la lista `($&)`.

```
pl@nereida:~/src/perl/perltesting$ cat -n escapes.pl
#!/usr/bin/perl -w
use strict;

my $foo = "one two three four five\nsix seven";
my ($F1, $F2, $Etc) = ($foo =~ /^\s*(\S+)\s+(\S+)\s*(.*)/);
print "List Context: F1 = $F1, F2 = $F2, Etc = $Etc\n";

# This is 'almost' the same than:
($F1, $F2, $Etc) = split(/\s+/, $foo, 3);
print "Split: F1 = $F1, F2 = $F2, Etc = $Etc\n";
```

Observa el resultado de la ejecución:

```
pl@nereida:~/src/perl/perltesting$ ./escapes.pl
List Context: F1 = one, F2 = two, Etc = three four five
Split: F1 = one, F2 = two, Etc = three four five
six seven
```

El modificador `s`

La opción `s` usada en una regexp hace que el punto `'.'` case con el retorno de carro:

```
pl@nereida:~/src/perl/perltesting$ perl -wd ./escapes.pl
main::(./escapes.pl:4): my $foo = "one two three four five\nsix seven";
DB<1> c 9
List Context: F1 = one, F2 = two, Etc = three four five
main::(./escapes.pl:9): ($F1, $F2, $Etc) = split(' ',$foo, 3);
DB<2> ($F1, $F2, $Etc) = ($foo =~ /^\s*(\S+)\s+(\S+)\s*(.*)/s)
DB<3> p "List Context: F1 = $F1, F2 = $F2, Etc = $Etc\n"
List Context: F1 = one, F2 = two, Etc = three four five
six seven
```

La opción `/s` hace que `.` se empareje con un `\n`. Esto es, casa con cualquier carácter.

Veamos otro ejemplo, que imprime los nombres de los ficheros que contienen cadenas que casan con un patrón dado, incluso si este aparece disperso en varias líneas:

```
#!/usr/bin/perl -w
#use:
#smodifier.pl 'expr' files
#prints the names of the files that match with the give expr
undef $/; # input record separator
my $what = shift @ARGV;
while(my $file = shift @ARGV) {
  open(FILE, "<$file");
```

```
    $line =  <FILE>;
    if ($line =~ /$what/s) {
      print "$file\n";
    }
  }
```

Ejemplo de uso:

```
> smodifier.pl 'three.*three' double.in split.pl doublee.pl
double.in
doublee.pl
```

Vea la sección 3.4.2 para ver los contenidos del fichero `double.in`. En dicho fichero, el patrón `three.*three` aparece repartido entre varias líneas.

El modificador `m`

El modificador `s` se suele usar conjuntamente con el modificador `m`. He aquí lo que dice la seccion *Using character classes* de la sección 'Using-character-classes' en `perlretut` al respecto:

- `m` *modifier (*`//m`*): Treat string as a set of multiple lines.* `'.'` *matches any character except* `\n`*.* `^` *and* `$` *are able to match at the start or end of any line within the string.*
- *both* `s` *and* `m` *modifiers (*`//sm`*): Treat string as a single long line, but detect multiple lines.* `'.'` *matches any character, even* `\n` *.* `^` *and* `$` *, however, are able to match at the start or end of any line within the string.*

Here are examples of //s and //m in action:

```
$x = "There once was a girl\nWho programmed in Perl\n";

$x =~ /^Who/; # doesn't match, "Who" not at start of string
$x =~ /^Who/s; # doesn't match, "Who" not at start of string
$x =~ /^Who/m; # matches, "Who" at start of second line
$x =~ /^Who/sm; # matches, "Who" at start of second line

$x =~ /girl.Who/; # doesn't match, "." doesn't match "\n"
$x =~ /girl.Who/s; # matches, "." matches "\n"
$x =~ /girl.Who/m; # doesn't match, "." doesn't match "\n"
$x =~ /girl.Who/sm; # matches, "." matches "\n"
```

Most of the time, the default behavior is what is wanted, but `//s` *and* `//m` *are occasionally very useful. If* `//m` *is being used, the start of the string can still be matched with* `\A` *and the end of the string can still be matched with the anchors* `\Z` *(matches both the end and the newline before, like* `$`*), and* `\z` *(matches only the end):*

```
$x =~ /^Who/m; # matches, "Who" at start of second line
$x =~ /\AWho/m; # doesn't match, "Who" is not at start of string

$x =~ /girl$/m; # matches, "girl" at end of first line
$x =~ /girl\Z/m; # doesn't match, "girl" is not at end of string

$x =~ /Perl\Z/m; # matches, "Perl" is at newline before end
$x =~ /Perl\z/m; # doesn't match, "Perl" is not at end of string
```

Normalmente el carácter `^` casa solamente con el comienzo de la cadena y el carácter `$` con el final. Los `\n` empotrados no casan con `^` ni `$`. El modificador `/m` modifica esta conducta. De este modo `^` y `$` casan con cualquier frontera de línea interna. Las anclas `\A` y `\Z` se utilizan entonces para casar con el comienzo y final de la cadena. Véase un ejemplo:

```
nereida:~/perl/src> perl -de 0
  DB<1> $a = "hola\npedro"
  DB<2> p "$a"
hola
pedro
  DB<3> $a =~ s/.*/x/m
  DB<4> p $a
x
pedro
  DB<5> $a =~ s/^pedro$/juan/
  DB<6> p "$a"
x
pedro
  DB<7> $a =~ s/^pedro$/juan/m
  DB<8>  p "$a"
x
juan
```

El conversor de temperaturas reescrito usando contexto de lista

Reescribamos el ejemplo anterior usando un contexto de lista:

```
casiano@millo:~/Lperltesting$ cat -n c2f_list.pl
#!/usr/bin/perl -w
use strict;

print "Enter a temperature (i.e. 32F, 100C):\n";
my $input = <STDIN>;
chomp($input);

my ($InputNum, $type);

($InputNum, $type) = $input =~ m/^
                                   ([-+]?[0-9]+(?:\.[0-9]*)?) # Temperature
                                   \s*
                                   ([cCfF]) # Celsius or Farenheit
                                $/x;

die "Expecting a temperature, so don't understand \"$input\".\n" unless defined($InputN

my ($celsius, $fahrenheit);
if ($type eq "C" or $type eq "c") {
  $celsius = $InputNum;
  $fahrenheit = ($celsius * 9/5)+32;
}
else {
  $fahrenheit = $InputNum;
  $celsius = ($fahrenheit -32)*5/9;
}
printf "%.2f C = %.2f F\n", $celsius, $fahrenheit;
```

La opción `x`

La opción `/x` en una regexp permite utilizar comentarios y espacios dentro de la expresión regular. Los espacios dentro de la expresión regular dejan de ser significativos. Si quieres conseguir un espacio que sea significativo, usa `\s` o bien escápalo. Véase la sección 'Modifiers' en `perlre` y la sección 'Building-a-regexp' en `perlretut`.

Paréntesis sin memoria

La notación `(?: ... )` se usa para introducir paréntesis de agrupamiento sin memoria. `(?: ...)` Permite agrupar las expresiones tal y como lo hacen los paréntesis ordinarios. La diferencia es que no "memorizan" esto es no guardan nada en `$1`, `$2`, etc. Se logra así una compilación mas eficiente. Veamos un ejemplo:

```
> cat groupingpar.pl
#!/usr/bin/perl

  my $a = shift;

  $a =~ m/(?:hola )*(juan)/;
  print "$1\n";
nereida:~/perl/src> groupingpar.pl 'hola juan'
juan
```

Interpolación en los patrones: La opción `o`

El patrón regular puede contener variables, que serán interpoladas (en tal caso, el patrón será recompilado). Si quieres que dicho patrón se compile una sóla vez, usa la opción `/o`.

```
pl@nereida:~/Lperltesting$ cat -n mygrep.pl
#!/usr/bin/perl -w
my $what = shift @ARGV || die "Usage $0 regexp files ...\n";
while (<>) {
  print "File $ARGV, rel. line $.: $_" if (/$what/o); # compile only once
}

```

Sigue un ejemplo de ejecución:

```
pl@nereida:~/Lperltesting$ ./mygrep.pl
Usage ./mygrep.pl regexp files ...
pl@nereida:~/Lperltesting$ ./mygrep.pl if labels.c
File labels.c, rel. line 7:        if (a < 10) goto LABEL;
```

El siguiente texto es de la sección 'Using-regular-expressions-in-Perl' en `perlretut`:

> *If* `$pattern` *won't be changing over the lifetime of the script, we can add the //o modifier, which directs Perl to only perform variable substitutions once*

Otra posibilidad es hacer una compilación previa usando el operador `qr` (véase la sección 'Regexp-Quote-Like-Operators' en `perlop`). La siguiente variante del programa anterior también compila el patrón una sóla vez:

```
pl@nereida:~/Lperltesting$ cat -n mygrep2.pl
#!/usr/bin/perl -w
my $what = shift @ARGV || die "Usage $0 regexp files ...\n";
$what = qr{$what};
while (<>) {
  print "File $ARGV, rel. line $.: $_" if (/$what/);
}
```

Véase

- El nodo en perlmonks /o is dead, long live qr//! por diotalevi

Cuantificadores greedy

El siguiente extracto de la sección *Matching Repetitions* en la sección 'Matching-repetitions' en `perlretut` ilustra la semántica *greedy* de los operadores de repetición `*+{}?` etc.

> *For all of these quantifiers, Perl will try to match as much of the string as possible, while still allowing the regexp to succeed. Thus with* `/a?.../`*, Perl will first try to match the regexp with the a present; if that fails, Perl will try to match the regexp without the a present. For the quantifier* `*` *, we get the following:*

```
1. $x = "the cat in the hat";
2. $x =~ /^(.*)(cat)(.*)$/; # matches,
3. # $1 = 'the '
4. # $2 = 'cat'
5. # $3 = ' in the hat'
```

> *Which is what we might expect, the match finds the only cat in the string and locks onto it. Consider, however, this regexp:*

```
1. $x =~ /^(.*)(at)(.*)$/; # matches,
2. # $1 = 'the cat in the h'
3. # $2 = 'at'
4. # $3 = '' (0 characters match)
```

> *One might initially guess that Perl would find the* `at` *in* `cat` *and stop there, but that wouldn't give the longest possible string to the first quantifier* `.*`*. Instead, the first quantifier* `.*` *grabs as much of the string as possible while still having the regexp match. In this example, that means having the* `at` *sequence with the final* `at` *in the string.*
>
> *The other important principle illustrated here is that when there are two or more elements in a regexp, the leftmost quantifier, if there is one, gets to grab as much the string as possible, leaving the rest of the regexp to fight over scraps. Thus in our example, the first quantifier* `.*` *grabs most of the string, while the second quantifier* `.*` *gets the empty string. Quantifiers that grab as much of the string as possible are called maximal match or greedy quantifiers.*
>
> *When a regexp can match a string in several different ways, we can use the principles above to predict which way the regexp will match:*
>
> - **Principle 0:** *Taken as a whole, any regexp will be matched at the earliest possible position in the string.*
> - **Principle 1:** *In an alternation* `a|b|c...` *, the leftmost alternative that allows a match for the whole regexp will be the one used.*
> - **Principle 2:** *The maximal matching quantifiers* `?`, `*`, `+` *and* `{n,m}` *will in general match as much of the string as possible while still allowing the whole regexp to match.*
> - **Principle 3:** *If there are two or more elements in a regexp, the leftmost greedy quantifier, if any, will match as much of the string as possible while still allowing the whole regexp to match. The next leftmost greedy quantifier, if any, will try to match as much of the string remaining available to it as possible, while still allowing the whole regexp to match. And so on, until all the regexp elements are satisfied.*

Regexp y Bucles Infinitos

El siguiente párrafo está tomado de la sección 'Repeated-Patterns-Matching-a-Zero-length-Substring' en `perlre`:

> *Regular expressions provide a terse and powerful programming language. As with most other power tools, power comes together with the ability to wreak havoc.*
>
> *A common abuse of this power stems from the ability to make infinite loops using regular expressions, with something as innocuous as:*

```
'foo' =~ m{ ( o? )* }x;
```

The `o?` *matches at the beginning of* `'foo'` *, and since the position in the string is not moved by the match,* `o?` *would match again and again because of the* `*` *quantifier.*

Another common way to create a similar cycle is with the looping modifier `//g` *:*

```
@matches = ( 'foo' =~ m{ o? }xg );
```

or

```
print "match: <$&>\n" while 'foo' =~ m{ o? }xg;
```

or the loop implied by `split()`*.*

... Perl allows such constructs, by forcefully breaking the infinite loop. The rules for this are different for lower-level loops given by the greedy quantifiers `*+{}` *, and for higher-level ones like the* `/g` *modifier or* `split()` *operator.*

The lower-level loops are interrupted (that is, the loop is broken) when Perl detects that a repeated expression matched a zero-length substring. Thus

```
m{ (?: NON_ZERO_LENGTH | ZERO_LENGTH )* }x;
```

is made equivalent to

```
m{ (?: NON_ZERO_LENGTH )*
|
(?: ZERO_LENGTH )?
}x;
```

The higher level-loops preserve an additional state between iterations: whether the last match was zero-length. To break the loop, the following match after a zero-length match is prohibited to have a length of zero. This prohibition interacts with backtracking (see Backtracking), and so the second best match is chosen if the best match is of zero length.

For example:

```
$_ = 'bar';
s/\w??/<$&>/g;
```

results in `<><b><><a><><r><>` *. At each position of the string the best match given by non-greedy* `??` *is the zero-length match, and the second best match is what is matched by* `\w` *. Thus zero-length matches alternate with one-character-long matches.*

Similarly, for repeated `m/()/g` *the second-best match is the match at the position one notch further in the string.*

The additional state of being matched with zero-length is associated with the matched string, and is reset by each assignment to `pos()`*. Zero-length matches at the end of the previous match are ignored during* `split`*.*

Ejercicio 3.1.2.

- *Explique la conducta del siguiente matching:*

```
  DB<25> $c = 0

  DB<26>  print(($c++).": <$&>\n") while 'aaaabababab' =~ /a*(ab)*/g;
0: <aaaa>
1: <>
2: <a>
3: <>
4: <a>
5: <>
6: <a>
7: <>
8: <>
```

Cuantificadores *lazy*

Las expresiones *lazy* o *no greedy* hacen que el NFA se detenga en la cadena mas corta que casa con la expresión. Se denotan como sus análogas *greedy* añadiéndole el postfijo `?`:

- `{n,m}?`
- `{n,}?`
- `{n}?`
- `*?`
- `+?`
- `??`

Repasemos lo que dice la sección Matching Repetitions en la sección 'Matching-repetitions' en `perlretut`:

> *Sometimes greed is not good. At times, we would like quantifiers to match a minimal piece of string, rather than a maximal piece. For this purpose, Larry Wall created the minimal match or non-greedy quantifiers* `??` *,*`*?`*,* `+?`*, and* `{}?`*. These are the usual quantifiers with a ? appended to them. They have the following meanings:*
>
> - `a??` *means: match 'a' 0 or 1 times. Try 0 first, then 1.*
> - `a*?` *means: match 'a' 0 or more times, i.e., any number of times, but as few times as possible*
> - `a+?` *means: match 'a' 1 or more times, i.e., at least once, but as few times as possible*
> - `a{n,m}?` *means: match at least n times, not more than m times, as few times as possible*
> - `a{n,}?` *means: match at least n times, but as few times as possible*
> - `a{n}?` *means: match exactly n times. Because we match exactly n times, an? is equivalent to an and is just there for notational consistency.*
>
> *Let's look at the example above, but with minimal quantifiers:*
>
> ```
> 1. $x = "The programming republic of Perl";
> 2. $x =~ /^(.+?)(e|r)(.*)$/; # matches,
> 3. # $1 = 'Th'
> 4. # $2 = 'e'
> 5. # $3 = ' programming republic of Perl'
> ```

The minimal string that will allow both the start of the string `^` *and the alternation to match is* `Th` *, with the alternation* `e|r` *matching* `e`*. The second quantifier* `.*` *is free to gobble up the rest of the string.*

```
1. $x =~ /(m{1,2}?)(.*?)$/; # matches,
2. # $1 = 'm'
3. # $2 = 'ming republic of Perl'
```

The first string position that this regexp can match is at the first `m` *in programming . At this position, the minimal* `m{1,2}?` *matches just one* `m` *. Although the second quantifier* `.*?` *would prefer to match no characters, it is constrained by the end-of-string anchor* `$` *to match the rest of the string.*

```
1. $x =~ /(.*?)(m{1,2}?)(.*)$/; # matches,
2. # $1 = 'The progra'
3. # $2 = 'm'
4. # $3 = 'ming republic of Perl'
```

In this regexp, you might expect the first minimal quantifier `.*?` *to match the empty string, because it is not constrained by a* `^` *anchor to match the beginning of the word. Principle 0 applies here, however. Because it is possible for the whole regexp to match at the start of the string, it will match at the start of the string. Thus the first quantifier has to match everything up to the first m. The second minimal quantifier matches just one* `m` *and the third quantifier matches the rest of the string.*

```
1. $x =~ /(.??)(m{1,2})(.*)$/; # matches,
2. # $1 = 'a'
3. # $2 = 'mm'
4. # $3 = 'ing republic of Perl'
```

Just as in the previous regexp, the first quantifier `.??` *can match earliest at position* `a` *, so it does. The second quantifier is greedy, so it matches mm , and the third matches the rest of the string.*

We can modify principle 3 above to take into account non-greedy quantifiers:

- **Principle 3:** *If there are two or more elements in a regexp, the leftmost greedy (non-greedy) quantifier, if any, will match as much (little) of the string as possible while still allowing the whole regexp to match. The next leftmost greedy (non-greedy) quantifier, if any, will try to match as much (little) of the string remaining available to it as possible, while still allowing the whole regexp to match. And so on, until all the regexp elements are satisfied.*

Ejercicio 3.1.3. *Explique cuál será el resultado de el segundo comando de matching introducido en el depurador:*

```
casiano@millo:~/Lperltesting$ perl -wde 0
main::(-e:1):   0
  DB<1> x ('1'x34) =~ m{^(11+)\1+$}
0  11111111111111111
  DB<2> x ('1'x34) =~ m{^(11+?)\1+$}
????????????????????????????????????
```

Descripción detallada del proceso de matching Veamos en detalle lo que ocurre durante un matching. Repasemos lo que dice la sección Matching Repetitions en la sección 'Matching-repetitions' en `perlretut`:

> *Just like alternation, quantifiers are also susceptible to backtracking. Here is a step-by-step analysis of the example*
>
> ```
> 1. $x = "the cat in the hat";
> 2. $x =~ /^(.*)(at)(.*)$/; # matches,
> 3. # $1 = 'the cat in the h'
> 4. # $2 = 'at'
> 5. # $3 = '' (0 matches)
> ```
>
> 1. *Start with the first letter in the string 't'.*
> 2. *The first quantifier '.*' starts out by matching the whole string 'the cat in the hat'.*
> 3. *'a' in the regexp element 'at' doesn't match the end of the string. Backtrack one character.*
> 4. *'a' in the regexp element 'at' still doesn't match the last letter of the string 't', so backtrack one more character.*
> 5. *Now we can match the 'a' and the 't'.*
> 6. *Move on to the third element '.*'. Since we are at the end of the string and '.*' can match 0 times, assign it the empty string.*
> 7. *We are done!*

Rendimiento

La forma en la que se escribe una regexp puede dar lugar agrandes variaciones en el rendimiento. Repasemos lo que dice la sección Matching Repetitions en la sección 'Matching-repetitions' en `perlretut`:

> *Most of the time, all this moving forward and backtracking happens quickly and searching is fast. There are some pathological regexps, however, whose execution time exponentially grows with the size of the string. A typical structure that blows up in your face is of the form*
>
> ```
> /(a|b+)*/;
> ```
>
> *The problem is the nested indeterminate quantifiers. There are many different ways of partitioning a string of length n between the* `+` *and* `*`*: one repetition with* `b+` *of length n, two repetitions with the first* `b+` *length k and the second with length $n-k$, m repetitions whose bits add up to length n, etc.*
>
> *In fact there are an exponential number of ways to partition a string as a function of its length. A regexp may get lucky and match early in the process, but if there is no match, Perl will try every possibility before giving up. So be careful with nested* `*`*'s,* `{n,m}`*'s, and* `+` *'s.*
>
> *The book Mastering Regular Expressions by Jeffrey Friedl [2] gives a wonderful discussion of this and other efficiency issues.*

Eliminación de Comentarios de un Programa C

El siguiente ejemplo elimina los comentarios de un programa `C`.

```
casiano@millo:~/Lperltesting$ cat -n comments.pl
   1    #!/usr/bin/perl -w
   2    use strict;
```

```

my $progname = shift @ARGV or die "Usage:\n$0 prog.c\n";
open(my $PROGRAM,"<$progname") || die "can't find $progname\n";
my $program = '';
{
  local $/ = undef;
  $program = <$PROGRAM>;
}
$program =~ s{
  /\*  # Match the opening delimiter
  .*?  # Match a minimal number of characters
  \*/  # Match the closing delimiter
}[]gsx;

print $program;
```

Veamos un ejemplo de ejecución. Supongamos el fichero de entrada:

```
> cat hello.c
#include <stdio.h>
/* first
comment
*/
main() {
  printf("hello world!\n"); /* second comment */
}
```

Entonces la ejecución con ese fichero de entrada produce la salida:

```
> comments.pl hello.c
#include <stdio.h>

main() {
  printf("hello world!\n");
}
```

Veamos la diferencia de comportamiento entre `*` y `*?` en el ejemplo anterior:

```
pl@nereida:~/src/perl/perltesting$ perl5_10_1 -wde 0
main::(-e:1):   0
  DB<1>  use re 'debug'; 'main() /* 1c */ { /* 2c */ return; /* 3c */ }' =~ qr{(/\*.*\*/)}; pr
Compiling REx "(/\*.*\*/)"
Final program:
   1: OPEN1 (3)
   3:   EXACT </*> (5)
   5:   STAR (7)
   6:     REG_ANY (0)
   7:   EXACT <*/> (9)
   9: CLOSE1 (11)
  11: END (0)
anchored "/*" at 0 floating "*/" at 2..2147483647 (checking floating) minlen 4
Guessing start of match in sv for REx "(/\*.*\*/)" against "main() /* 1c */ { /* 2c */ return;
Found floating substr "*/" at offset 13...
Found anchored substr "/*" at offset 7...
Starting position does not contradict /^/m...
Guessed: match at offset 7
```

```
Matching REx "(/\*.*\*/)" against "/* 1c */ { /* 2c */ return; /* 3c */ }"
   7 <in() > </* 1c */ {>    |  1:OPEN1(3)
   7 <in() > </* 1c */ {>    |  3:EXACT </*>(5)
   9 <() /*> < 1c */ { />    |  5:STAR(7)
                                  REG_ANY can match 36 times out of 2147483647...
  41 <; /* 3c > <*/ }>       |  7:  EXACT <*/>(9)
  43 <; /* 3c */> < }>       |  9:  CLOSE1(11)
  43 <; /* 3c */> < }>       | 11:  END(0)
Match successful!

/* 1c */ { /* 2c */ return; /* 3c */
Freeing REx: "(/\*.*\*/)"

  DB<2>  use re 'debug'; 'main() /* 1c */ { /* 2c */ return; /* 3c */ }' =~ qr{(/\*.*?\*/)}; p
Compiling REx "(/\*.*?\*/)"
Final program:
   1: OPEN1 (3)
   3:   EXACT </*> (5)
   5:   MINMOD (6)
   6:   STAR (8)
   7:     REG_ANY (0)
   8:   EXACT <*/> (10)
  10: CLOSE1 (12)
  12: END (0)
anchored "/*" at 0 floating "*/" at 2..2147483647 (checking floating) minlen 4
Guessing start of match in sv for REx "(/\*.*?\*/)" against "main() /* 1c */ { /* 2c */ return
Found floating substr "*/" at offset 13...
Found anchored substr "/*" at offset 7...
Starting position does not contradict /^/m...
Guessed: match at offset 7
Matching REx "(/\*.*?\*/)" against "/* 1c */ { /* 2c */ return; /* 3c */ }"
   7 <in() > </* 1c */ {>    |  1:OPEN1(3)
   7 <in() > </* 1c */ {>    |  3:EXACT </*>(5)
   9 <() /*> < 1c */ { />    |  5:MINMOD(6)
   9 <() /*> < 1c */ { />    |  6:STAR(8)
                                  REG_ANY can match 4 times out of 4...
  13 <* 1c > <*/ { /* 2c>    |  8:  EXACT <*/>(10)
  15 <1c */> < { /* 2c *>    | 10:  CLOSE1(12)
  15 <1c */> < { /* 2c *>    | 12:  END(0)
Match successful!

/* 1c */
Freeing REx: "(/\*.*?\*/)"

  DB<3>
```

Véase también la documentación en la sección 'Matching-repetitions' en `perlretut` y la sección 'Quantifiers' en `perlre`.

Negaciones y operadores ***lazy*** A menudo las expresiones `X[^X]*X` y `X.*?X`, donde `X` es un carácter arbitrario se usan de forma casi equivalente.

- La primera significa:

  Una cadena que no contiene `X` *en su interior y que está delimitada por* `X`*s*

- La segunda significa:

  Una cadena que comienza en `X` *y termina en la* `X` *mas próxima a la* `X` *de comienzo*

Esta equivalencia se rompe si no se cumplen las hipótesis establecidas.

En el siguiente ejemplo se intentan detectar las cadenas entre comillas dobles que terminan en el signo de exclamación:

```
pl@nereida:~/Lperltesting$ cat -n negynogreedy.pl
#!/usr/bin/perl -w
use strict;

my $b = 'Ella dijo "Ana" y yo contesté: "Jamás!". Eso fué todo.';
my $a;
($a = $b) =~ s/".*?!"/-$&-/;
print "$a\n";

$b =~ s/"[^"]*!"/-$&-/;
print "$b\n";
```

Al ejecutar el programa obtenemos:

```
> negynogreedy.pl
Ella dijo -"Ana" y yo contesté: "Jamás!"-. Eso fué todo.
Ella dijo "Ana" y yo contesté: -"Jamás!"-. Eso fué todo.
```

Copia y sustitución simultáneas El operador de *binding* `=~` nos permite "asociar" la variable con la operación de casamiento o sustitución. Si se trata de una sustitución y se quiere conservar la cadena, es necesario hacer una copia:

```
$d = $s;
$d =~ s/esto/por lo otro/;
```

en vez de eso, puedes abreviar un poco usando la siguiente "perla":

```
($d = $s) =~ s/esto/por lo otro/;
```

Obsérvese la asociación por la izquierda del operador de asignación.

Referencias a Paréntesis Previos

Las referencias relativas permiten escribir expresiones regulares mas reciclables. Véase la documentación en la sección 'Relative-backreferences' en `perlretut`:

> *Counting the opening parentheses to get the correct number for a backreference is error-prone as soon as there is more than one capturing group. A more convenient technique became available with Perl 5.10: relative backreferences. To refer to the immediately preceding capture group one now may write* `\g{-1}` *, the next but last is available via* `\g{-2}`*, and so on.*
>
> *Another good reason in addition to readability and maintainability for using relative backreferences is illustrated by the following example, where a simple pattern for matching peculiar strings is used:*
>
> ```
> 1. $a99a = '([a-z])(\d)\2\1'; # matches a11a, g22g, x33x, etc.
> ```
>
> *Now that we have this pattern stored as a handy string, we might feel tempted to use it as a part of some other pattern:*

```
$line = "code=e99e";
if ($line =~ /^(\w+)=$a99a$/){ # unexpected behavior!
  print "$1 is valid\n";
} else {
  print "bad line: '$line'\n";
}
```

But this doesn't match – at least not the way one might expect. Only after inserting the interpolated `$a99a` *and looking at the resulting full text of the regexp is it obvious that the backreferences have backfired – the subexpression* `(\w+)` *has snatched number 1 and demoted the groups in* `$a99a` *by one rank. This can be avoided by using relative backreferences:*

```
$a99a = '([a-z])(\d)\g{-1}\g{-2}'; # safe for being interpolated
```

El siguiente programa ilustra lo dicho:

```
casiano@millo:~/Lperltesting$ cat -n backreference.pl
   1    use strict;
   2    use re 'debug';
   3
   4    my $a99a = '([a-z])(\d)\2\1';
   5    my $line = "code=e99e";
   6    if ($line =~ /^(\w+)=$a99a$/){ # unexpected behavior!
   7      print "$1 is valid\n";
   8    } else {
   9      print "bad line: '$line'\n";
  10    }
```

Sigue la ejecución:

```
casiano@millo:~/Lperltesting$ perl5.10.1 -wd backreference.pl
main::(backreference.pl:4):     my $a99a = '([a-z])(\d)\2\1';
  DB<1> c 6
main::(backreference.pl:6):     if ($line =~ /^(\w+)=$a99a$/){ # unexpected behavior!
  DB<2> x ($line =~ /^(\w+)=$a99a$/)
  empty array
  DB<4> $a99a = '([a-z])(\d)\g{-1}\g{-2}'
  DB<5> x ($line =~ /^(\w+)=$a99a$/)
0  'code'
1  'e'
2  9
```

Usando Referencias con Nombre (Perl 5.10)

El siguiente texto esta tomado de la sección 'Named-backreferences' en `perlretut`:

Perl 5.10 also introduced named capture buffers and named backreferences. To attach a name to a capturing group, you write either `(?<name>...)` *or* `(?'name'...)`*. The backreference may then be written as* `\g{name}` *.*

It is permissible to attach the same name to more than one group, but then only the leftmost one of the eponymous set can be referenced. Outside of the pattern a named capture buffer is accessible through the `%+` *hash.*

Assuming that we have to match calendar dates which may be given in one of the three formats `yyyy-mm-dd`*,* `mm/dd/yyyy` *or* `dd.mm.yyyy`*, we can write three suitable patterns where we use* `'d'`*,* `'m'` *and* `'y'` *respectively as the names of the buffers capturing the pertaining components of a date. The matching operation combines the three patterns as alternatives:*

```
$fmt1 = '(?<y>\d\d\d\d)-(?<m>\d\d)-(?<d>\d\d)';
$fmt2 = '(?<m>\d\d)/(?<d>\d\d)/(?<y>\d\d\d\d)';
$fmt3 = '(?<d>\d\d)\.(?<m>\d\d)\.(?<y>\d\d\d\d)';
for my $d qw( 2006-10-21 15.01.2007 10/31/2005 ){
  if ( $d =~ m{$fmt1|$fmt2|$fmt3} ){
    print "day=$+{d} month=$+{m} year=$+{y}\n";
  }
}
```

If any of the alternatives matches, the hash `%+` *is bound to contain the three key-value pairs.*

En efecto, al ejecutar el programa:

```
casiano@millo:~/Lperltesting$ cat -n namedbackreferences.pl
     1  use v5.10;
     2  use strict;
     3
     4  my $fmt1 = '(?<y>\d\d\d\d)-(?<m>\d\d)-(?<d>\d\d)';
     5  my $fmt2 = '(?<m>\d\d)/(?<d>\d\d)/(?<y>\d\d\d\d)';
     6  my $fmt3 = '(?<d>\d\d)\.(?<m>\d\d)\.(?<y>\d\d\d\d)';
     7
     8  for my $d qw( 2006-10-21 15.01.2007 10/31/2005 ){
     9    if ( $d =~ m{$fmt1|$fmt2|$fmt3} ){
    10      print "day=$+{d} month=$+{m} year=$+{y}\n";
    11    }
    12  }
```

Obtenemos la salida:

```
casiano@millo:~/Lperltesting$ perl5.10.1 -w namedbackreferences.pl
day=21 month=10 year=2006
day=15 month=01 year=2007
day=31 month=10 year=2005
```

Como se comentó:

... It is permissible to attach the same name to more than one group, but then only the leftmost one of the eponymous set can be referenced.

Veamos un ejemplo:

```
pl@nereida:~/Lperltesting$ perl5.10.1 -wdE 0
main::(-e:1):   0
DB<1>  # ... only the leftmost one of the eponymous set can be referenced
DB<2> $r = qr{(?<a>[a-c])(?<a>[a-f])}
DB<3> print $+{a} if 'ad' =~ $r
a
DB<4> print $+{a} if 'cf' =~ $r
c
DB<5> print $+{a} if 'ak' =~ $r
```

Reescribamos el ejemplo de conversión de temperaturas usando paréntesis con nombre:

```
pl@nereida:~/Lperltesting$ cat -n c2f_5_10v2.pl
#!/usr/local/bin/perl5_10_1 -w
use strict;

print "Enter a temperature (i.e. 32F, 100C):\n";
my $input = <STDIN>;
chomp($input);

$input =~ m/^
            (?<farenheit>[-+]?[0-9]+(?:\.[0-9]*)?)\s*[fF]
            |
            (?<celsius>[-+]?[0-9]+(?:\.[0-9]*)?)\s*[cC]
         $/x;

my ($celsius, $farenheit);
if (exists $+{celsius}) {
  $celsius = $+{celsius};
  $farenheit = ($celsius * 9/5)+32;
}
elsif (exists $+{farenheit}) {
  $farenheit = $+{farenheit};
  $celsius = ($farenheit -32)*5/9;
}
else {
  die "Expecting a temperature, so don't understand \"$input\".\n";
}

printf "%.2f C = %.2f F\n", $celsius, $farenheit;
```

La función `exists` retorna verdadero si existe la clave en el hash y falso en otro caso.

Grupos con Nombre y Factorización

El uso de nombres hace mas robustas y mas factorizables las expresiones regulares. Consideremos la siguiente regexp que usa notación posicional:

```
pl@nereida:~/Lperltesting$ perl5.10.1 -wde 0
main::(-e:1):   0
  DB<1> x "abbacddc" =~ /(.)(.)\2\1/
0  'a'
1  'b'
```

Supongamos que queremos reutilizar la regexp con repetición

```
  DB<2> x "abbacddc" =~ /((.)(.)\2\1){2}/
  empty array
```

¿Que ha ocurrido? La introducción del nuevo paréntesis nos obliga a renombrar las referencias a las posiciones:

```
  DB<3> x "abbacddc" =~ /((.)(.)\3\2){2}/
0  'cddc'
1  'c'
2  'd'
  DB<4> "abbacddc" =~ /((.)(.)\3\2){2}/; print "$&\n"
abbacddc
```

Esto no ocurre si utilizamos nombres. El operador `\k<a>` sirve para hacer referencia al valor que ha casado con el paréntesis con nombre `a`:

```
  DB<5> x "abbacddc" =~ /((?<a>.)(?<b>.)\k<b>\k<a>){2}/
0  'cddc'
1  'c'
2  'd'
```

El uso de grupos con nombre y `\k`[1] en lugar de referencias numéricas absolutas hace que la regexp sea mas reutilizable.

LLamadas a expresiones regulares via paréntesis con memoria

Es posible también llamar a la expresión regular asociada con un paréntesis.

Este parrafo tomado de la sección 'Extended-Patterns' en `perlre` explica el modo de uso:

`(?PARNO) (?-PARNO) (?+PARNO) (?R) (?0)`

> `PARNO` *is a sequence of digits (not starting with 0) whose value reflects the paren-number of the capture buffer to recurse to.*
>
> *....*
>
> *Capture buffers contained by the pattern will have the value as determined by the outermost recursion. ....*
>
> *If* `PARNO` **is preceded by a plus or minus sign then it is assumed to be relative, with negative numbers indicating preceding capture buffers and positive ones following**. *Thus* `(?-1)` *refers to the most recently declared buffer, and* `(?+1)` *indicates the next buffer to be declared.*
>
> **Note that the counting for relative recursion differs from that of relative backreferences, in that with recursion unclosed buffers are included.**

Veamos un ejemplo:

```
casiano@millo:~/Lperltesting$ perl5.10.1 -wdE 0
main::(-e:1):   0
  DB<1> x "AABB" =~ /(A)(?-1)(?+1)(B)/
0  'A'
1  'B'
  # Parenthesis:       1   2   2                  1
  DB<2> x 'ababa' =~ /^((?:([ab])(?1)\g{-1}|[ab]?))$/
0  'ababa'
1  'a'
  DB<3> x 'bbabababb' =~ /^((?:([ab])(?1)\g{-1}|[ab]?))$/
0  'bbabababb'
1  'b'
```

Véase también:

- Perl Training Australia: Regular expressions in Perl 5.10
- Perl 5.10 Advanced Regular Expressions by Yves Orton
- Gabor: Regular Expressions in Perl 5.10

[1] Una diferencia entre $\backslash k$ y $\backslash g$ es que el primero sólo admite un nombre como argumento mientras que $\backslash g$ admite enteros

Reutilizando Expresiones Regulares

La siguiente reescritura de nuestro ejemplo básico utiliza el módulo `Regexp::Common` para factorizar la expresión regular:

```
casiano@millo:~/src/perl/perltesting$ cat -n c2f_5_10v3.pl
#!/soft/perl5lib/bin/perl5.10.1 -w
use strict;
use Regexp::Common;

print "Enter a temperature (i.e. 32F, 100C):\n";
my $input = <STDIN>;
chomp($input);

$input =~ m/^
            (?<farenheit>$RE{num}{real})\s*[fF]
            |
            (?<celsius>$RE{num}{real})\s*[cC]
         $/x;

my ($celsius, $farenheit);
if ('celsius' ~~ %+) {
  $celsius = $+{celsius};
  $farenheit = ($celsius * 9/5)+32;
}
elsif ('farenheit' ~~ %+) {
  $farenheit = $+{farenheit};
  $celsius = ($farenheit -32)*5/9;
}
else {
  die "Expecting a temperature, so don't understand \"$input\".\n";
}

printf "%.2f C = %.2f F\n", $celsius, $farenheit;
```

Véase:

- La documentación del módulo `Regexp::Common` por Abigail
- Smart Matching: Perl Training Australia: Smart Match
- Rafael García Suárez: la sección 'Smart-matching-in-detail' en `perlsyn`
- Enrique Nell (Barcelona Perl Mongers): Novedades en Perl 5.10

El Módulo Regexp::Common

El módulo `Regexp::Common` provee un extenso número de expresiones regulares que son accesibles vía el hash `%RE`. sigue un ejemplo de uso:

```
casiano@millo:~/Lperltesting$ cat -n regexpcommonsynopsis.pl
use strict;
use Perl6::Say;
use Regexp::Common;

while (<>) {
    say q{a number}              if /$RE{num}{real}/;

```

```
      say q{a ['"`] quoted string} if /$RE{quoted}/;

      say q{a /.../ sequence}      if m{$RE{delimited}{'-delim'=>'/'}};

      say q{balanced parentheses}  if /$RE{balanced}{'-parens'=>'()'}/;

      die q{a #*@%-ing word}."\n"  if /$RE{profanity}/;

}

```

Sigue un ejemplo de ejecución:

```
casiano@millo:~/Lperltesting$ perl regexpcommonsynopsis.pl
43
a number
"2+2 es" 4
a number
a ['"`] quoted string
x/y/z
a /.../ sequence
(2*(4+5/(3-2)))
a number
balanced parentheses
fuck you!
a #*@%-ing word
```

El siguiente fragmento de la documentación de `Regexp::Common` explica el modo simplificado de uso:

> *To access a particular pattern,* `%RE` *is treated as a hierarchical hash of hashes (of hashes...), with each successive key being an identifier. For example, to access the pattern that matches real numbers, you specify:*
>
> ```
> $RE{num}{real}
> ```
>
> *and to access the pattern that matches integers:*
>
> ```
> $RE{num}{int}
> ```
>
> *Deeper layers of the hash are used to specify flags: arguments that modify the resulting pattern in some way.*
>
> - *The keys used to access these layers are prefixed with a minus sign and may have a value;*
> - *if a value is given, it's done by using a multidimensional key.*
>
> *For example, to access the pattern that matches base-2 real numbers with embedded commas separating groups of three digits (e.g.* `10,101,110.110101101`*):*
>
> ```
> $RE{num}{real}{-base => 2}{-sep => ','}{-group => 3}
> ```
>
> *Through the magic of Perl, these flag layers may be specified in any order (and even interspersed through the identifier keys!) so you could get the same pattern with:*

```
$RE{num}{real}{-sep => ','}{-group => 3}{-base => 2}
```

or:

```
$RE{num}{-base => 2}{real}{-group => 3}{-sep => ','}
```

or even:

```
$RE{-base => 2}{-group => 3}{-sep => ','}{num}{real}
```

etc.

Note, however, that the relative order of amongst the identifier keys is significant. That is:

```
$RE{list}{set}
```

would not be the same as:

```
$RE{set}{list}
```

Veamos un ejemplo con el depurador:

```
casiano@millo:~/Lperltesting$ perl -MRegexp::Common -wde 0
main::(-e:1):   0
  DB<1> x 'numero: 10,101,110.110101101 101.1e-1 234' =~ m{($RE{num}{real}{-base => 2}{-sep =>
0  '10,101,110.110101101'
1  '101.1e-1'
```

La expresión regular para un número real es relativamente compleja:

```
casiano@millo:~/src/perl/perltesting$ perl5.10.1 -wd c2f_5_10v3.pl
main::(c2f_5_10v3.pl:5):     print "Enter a temperature (i.e. 32F, 100C):\n";
  DB<1> p $RE{num}{real}
(?:(?i)(?:[+-]?)(?:(?=[0123456789]|[.])(?:[0123456789]*)(?:(?:[.])(?:[0123456789]{0,}))?)(?:(?
```

Si se usa la opción `-keep` el patrón proveído usa paréntesis con memoria:

```
casiano@millo:~/Lperltesting$ perl -MRegexp::Common -wde 0
main::(-e:1):   0
DB<2> x 'one, two, three, four, five' =~ /$RE{list}{-pat => '\w+'}/
0  1
DB<3> x 'one, two, three, four, five' =~ /$RE{list}{-pat => '\w+'}{-keep}/
0  'one, two, three, four, five'
1  ', '
```

Smart Matching

Perl 5.10 introduce el operador de smart matching. El siguiente texto es tomado casi verbatim del site de la compañía Perl Training Australia[2]:

[2] This Perl tip and associated text is copyright Perl Training Australia

Perl 5.10 introduces a new-operator, called smart-match, written `~~`*. As the name suggests, smart-match tries to compare its arguments in an intelligent fashion. Using smart-match effectively allows many complex operations to be reduces to very simple statements.*

Unlike many of the other features introduced in Perl 5.10, there's no need to use the feature pragma to enable smart-match, as long as you're using 5.10 it's available.

The smart-match operator is always commutative. That means that `$x ~~ $y` *works the same way as* `$y ~~ $x`*. You'll never have to remember which order to place to your operands with smart-match. Smart-match in action.*

As a simple introduction, we can use smart-match to do a simple string comparison between simple scalars. For example:

```
use feature qw(say);

my $x = "foo";
my $y = "bar";
my $z = "foo";

say '$x and $y are identical strings' if $x ~~ $y;
say '$x and $z are identical strings' if $x ~~ $z;    # Printed
```

If one of our arguments is a number, then a numeric comparison is performed:

```
my $num   = 100;
my $input = <STDIN>;

say 'You entered 100' if $num ~~ $input;
```

This will print our message if our user enters 100, 100.00, +100, 1e2, or any other string that looks like the number 100.

We can also smart-match against a regexp:

```
my $input  = <STDIN>;

say 'You said the secret word!' if $input ~~ /xyzzy/;
```

Smart-matching with a regexp also works with saved regexps created with qr.

So we can use smart-match to act like eq, `==` *and* `=~`*, so what? Well, it does much more than that.*

We can use smart-match to search a list:

```
casiano@millo:~/Lperltesting$ perl5.10.1 -wdE 0
main::(-e:1):   0
  DB<1> @friends = qw(Frodo Meriadoc Pippin Samwise Gandalf)
  DB<2> print "You're a friend" if 'Pippin' ~~ @friends
You're a friend
  DB<3> print "You're a friend" if 'Mordok' ~~ @friends
```

It's important to note that searching an array with smart-match is extremely fast. It's faster than using grep, it's faster than using `first` *from* `Scalar::Util`*, and it's faster than walking through the loop with* `foreach`*, even if you do know all the clever optimisations.*

Esta es la forma típica de buscar un elemento en un array en versiones anteriores a la 5.10:

```
casiano@millo:~$ perl -wde 0
main::(-e:1):   0
  DB<1> use List::Util qw{first}
  DB<2> @friends = qw(Frodo Meriadoc Pippin Samwise Gandalf)
  DB<3> x first { $_ eq 'Pippin'} @friends
0  'Pippin'
  DB<4> x first { $_ eq 'Mordok'} @friends
0  undef
```

We can also use smart-match to compare arrays:

```
  DB<4> @foo = qw(x y z xyzzy ninja)
  DB<5> @bar = qw(x y z xyzzy ninja)
  DB<7> print "Identical arrays" if @foo ~~ @bar
Identical arrays
  DB<8> @bar = qw(x y z xyzzy nOnjA)
  DB<9> print "Identical arrays" if @foo ~~ @bar
  DB<10>
```

And even search inside an array using a string:

```
 DB<11> x @foo = qw(x y z xyzzy ninja)
0  'x'
1  'y'
2  'z'
3  'xyzzy'
4  'ninja'
  DB<12> print "Array contains a ninja " if @foo ~~ 'ninja'
```

or using a regexp:

```
  DB<13> print "Array contains magic pattern" if @foo ~~ /xyz/
Array contains magic pattern
  DB<14> print "Array contains magic pattern" if @foo ~~ /\d+/
```

Smart-match works with array references, too[3]*:*

```
  DB<16> $array_ref = [ 1..10 ]
  DB<17> print "Array contains 10" if 10 ~~ $array_ref
Array contains 10
  DB<18> print "Array contains 10" if $array_ref ~~ 10
  DB<19>
```

En el caso de un número y un array devuelve cierto si el escalar aparece en un array anidado:

```
casiano@millo:~/Lperltesting$ perl5.10.1 -E 'say "ok" if 42 ~~  [23, 17, [40..50], 70];'
ok
casiano@millo:~/Lperltesting$ perl5.10.1 -E 'say "ok" if 42 ~~  [23, 17, [50..60], 70];'
casiano@millo:~/Lperltesting$
```

Of course, we can use smart-match with more than just arrays and scalars, it works with searching for the key in a hash, too!

[3] En este caso la conmutatividad no funciona

```
  DB<19> %colour = ( sky   => 'blue', grass => 'green', apple => 'red',)
  DB<20> print "I know the colour" if 'grass' ~~ %colour
I know the colour
  DB<21> print "I know the colour" if 'cloud' ~~ %colour
  DB<22>
  DB<23> print "A key starts with 'gr'" if %colour ~~ /^gr/
A key starts with 'gr'
  DB<24> print "A key starts with 'clou'" if %colour ~~ /^clou/
  DB<25>
```

You can even use it to see if the two hashes have identical keys:

```
  DB<26> print 'Hashes have identical keys' if %taste ~~ %colour;
Hashes have identical keys
```

La conducta del operador de smart matching viene dada por la siguiente tabla tomada de la sección 'Smart-matching-in-detail' en `perlsyn`:

> *The behaviour of a smart match depends on what type of thing its arguments are. The behaviour is determined by the following table: the first row that applies determines the match behaviour (which is thus mostly determined by the type of the right operand). Note that the smart match implicitly dereferences any non-blessed hash or array ref, so the "Hash.ªnd .ᴬrray.ᵉⁿtries apply in those cases. (For blessed references, the .ᴼbject.ᵉⁿtries apply.)*
>
> *Note that the "Matching Codeçolumn is not always an exact rendition. For example, the smart match operator short-circuits whenever possible, but grep does not.*

```
 $a      $b         Type of Match Implied    Matching Code
 ======  =====      =====================    =============
 Any     undef      undefined                !defined $a

 Any     Object     invokes ~~ overloading on $object, or dies

 Hash    CodeRef    sub truth for each key[1] !grep { !$b->($_) } keys %$a
 Array   CodeRef    sub truth for each elt[1] !grep { !$b->($_) } @$a
 Any     CodeRef    scalar sub truth          $b->($a)

 Hash    Hash       hash keys identical (every key is found in both hashes)
 Array   Hash       hash slice existence     grep { exists $b->{$_} } @$a
 Regex   Hash       hash key grep            grep /$a/, keys %$b
 undef   Hash       always false (undef can't be a key)
 Any     Hash       hash entry existence     exists $b->{$a}

 Hash    Array      hash slice existence     grep { exists $a->{$_} } @$b
 Array   Array      arrays are comparable[2]
 Regex   Array      array grep               grep /$a/, @$b
 undef   Array      array contains undef     grep !defined, @$b
 Any     Array      match against an array element[3]
                                             grep $a ~~ $_, @$b

 Hash    Regex      hash key grep            grep /$b/, keys %$a
 Array   Regex      array grep               grep /$b/, @$a
 Any     Regex      pattern match            $a =~ /$b/
```

```
Object  Any       invokes ~~ overloading on $object, or falls back:
Any     Num       numeric equality        $a == $b
Num     numish[4] numeric equality        $a == $b
undef   Any       undefined               !defined($b)
Any     Any       string equality         $a eq $b
```

Ejercicios

Ejercicio 3.1.4.

- *Indique la salida del siguiente programa:*

```
pl@nereida:~/Lperltesting$ cat twonumbers.pl
$_ = "I have 2 numbers: 53147";
@pats = qw{
  (.*)(\d*)
  (.*)(\d+)
  (.*?)(\d*)
  (.*?)(\d+)
  (.*)(\d+)$
  (.*?)(\d+)$
  (.*)\b(\d+)$
  (.*\D)(\d+)$
};

print "$_\n";
for $pat (@pats) {
  printf "%-12s ", $pat;
  <>;
  if ( /$pat/ ) {
    print "<$1> <$2>\n";
  } else {
    print "FAIL\n";
  }
}
```

3.1.2. Depuración de Expresiones Regulares

Para obtener información sobre la forma en que es compilada una expresión regular y como se produce el proceso de matching podemos usar la opción `'debug'` del módulo `re`. La versión de Perl 5.10 da una información algo mas legible que la de las versiones anteriores:

```
pl@nereida:~/Lperltesting$ perl5_10_1 -wde 0

Loading DB routines from perl5db.pl version 1.32
Editor support available.

Enter h or `h h' for help, or `man perldebug' for more help.

main::(-e:1):   0
  DB<1> use re 'debug'; 'astr' =~ m{[sf].r}
Compiling REx "[sf].r"
Final program:
   1: ANYOF[fs][] (12)
  12: REG_ANY (13)
  13: EXACT <r> (15)
```

```
  15: END (0)
anchored "r" at 2 (checking anchored) stclass ANYOF[fs][] minlen 3
Guessing start of match in sv for REx "[sf].r" against "astr"
Found anchored substr "r" at offset 3...
Starting position does not contradict /^/m...
start_shift: 2 check_at: 3 s: 1 endpos: 2
Does not contradict STCLASS...
Guessed: match at offset 1
Matching REx "[sf].r" against "str"
   1 <a> <str>              |  1:ANYOF[fs][](12)
   2 <as> <tr>              | 12:REG_ANY(13)
   3 <ast> <r>              | 13:EXACT <r>(15)
   4 <astr> <>              | 15:END(0)
Match successful!
Freeing REx: "[sf].r"
```

Si se usa la opción `debug` de `re` con objetos expresión regular, se obtendrá información durante el proceso de matching:

```
 DB<3> use re 'debug'; $re = qr{[sf].r}
Compiling REx "[sf].r"
Final program:
   1: ANYOF[fs][] (12)
  12: REG_ANY (13)
  13: EXACT <r> (15)
  15: END (0)
anchored "r" at 2 (checking anchored) stclass ANYOF[fs][] minlen 3

  DB<4> 'astr' =~ $re
Guessing start of match in sv for REx "[sf].r" against "astr"
Found anchored substr "r" at offset 3...
Starting position does not contradict /^/m...
start_shift: 2 check_at: 3 s: 1 endpos: 2
Does not contradict STCLASS...
Guessed: match at offset 1
Matching REx "[sf].r" against "str"
   1 <a> <str>              |  1:ANYOF[fs][](12)
   2 <as> <tr>              | 12:REG_ANY(13)
   3 <ast> <r>              | 13:EXACT <r>(15)
   4 <astr> <>              | 15:END(0)
Match successful!
```

3.1.3. Tablas de Escapes, Metacarácteres, Cuantificadores, Clases

Sigue una sección de tablas con notaciones tomada de `perlre`:

Metacharacters

The following metacharacters have their standard egrep-ish meanings:

```
1. \ Quote the next metacharacter
2. ^ Match the beginning of the line
3. . Match any character (except newline)
4. $ Match the end of the line (or before newline at the end)
5. | Alternation
```

6. `()` Grouping
7. `[]` Character class

Standard greedy quantifiers

The following standard greedy quantifiers are recognized:

1. `*` Match 0 or more times
2. `+` Match 1 or more times
3. `?` Match 1 or 0 times
4. `{n}` Match exactly n times
5. `{n,}` Match at least n times
6. `{n,m}` Match at least n but not more than m times

Non greedy quantifiers

The following non greedy quantifiers are recognized:

1. `*?` Match 0 or more times, not greedily
2. `+?` Match 1 or more times, not greedily
3. `??` Match 0 or 1 time, not greedily
4. `{n}?` Match exactly n times, not greedily
5. `{n,}?` Match at least n times, not greedily
6. `{n,m}?` Match at least n but not more than m times, not greedily

Possesive quantifiers

The following possesive quantifiers are recognized:

1. `*+` Match 0 or more times and give nothing back
2. `++` Match 1 or more times and give nothing back
3. `?+` Match 0 or 1 time and give nothing back
4. `{n}+` Match exactly n times and give nothing back (redundant)
5. `{n,}+` Match at least n times and give nothing back
6. `{n,m}+` Match at least n but not more than m times and give nothing back

Escape sequences

1. `\t` tab (HT, TAB)
2. `\n` newline (LF, NL)
3. `\r` return (CR)
4. `\f` form feed (FF)
5. `\a` alarm (bell) (BEL)
6. `\e` escape (think troff) (ESC)
7. `\033` octal char (example: ESC)
8. `\x1B` hex char (example: ESC)
9. `\x{263a}` long hex char (example: Unicode SMILEY)
10. `\cK` control char (example: VT)
11. `\N{name}` named Unicode character
12. `\l` lowercase next char (think vi)
13. `\u` uppercase next char (think vi)
14. `\L` lowercase till `\E` (think vi)
15. `\U` uppercase till `\E` (think vi)
16. `\E` end case modification (think vi)
17. `\Q` quote (disable) pattern metacharacters till `\E`

Ejercicio 3.1.5. *Explique la salida:*

```
casiano@tonga:~$ perl -wde 0
main::(-e:1):   0
  DB<1> $x = '([a-z]+)'
  DB<2> x 'hola' =~ /$x/
0  'hola'
  DB<3> x 'hola' =~ /\Q$x/
  empty array
  DB<4> x '([a-z]+)' =~ /\Q$x/
0  1
```

Character Classes and other Special Escapes

```
\w Match a "word" character (alphanumeric plus "_")
\W Match a non-"word" character
\s Match a whitespace character
\S Match a non-whitespace character
\d Match a digit character
\D Match a non-digit character
\pP Match P, named property. Use \p{Prop} for longer names.
\PP Match non-P
\X Match eXtended Unicode "combining character sequence",
   equivalent to (?>\PM\pM*)
\C Match a single C char (octet) even under Unicode.
   NOTE: breaks up characters into their UTF-8 bytes,
   so you may end up with malformed pieces of UTF-8.
   Unsupported in lookbehind.
\1 Backreference to a specific group.
'1' may actually be any positive integer.
\g1 Backreference to a specific or previous group,
\g{-1} number may be negative indicating a previous buffer and may
       optionally be wrapped in curly brackets for safer parsing.
\g{name} Named backreference
\k<name> Named backreference
\K Keep the stuff left of the \K, don't include it in $&
\v Vertical whitespace
\V Not vertical whitespace
\h Horizontal whitespace
\H Not horizontal whitespace
\R Linebreak
```

Zero width assertions

Perl defines the following zero-width assertions:

```
\b Match a word boundary
\B Match except at a word boundary
\A Match only at beginning of string
\Z Match only at end of string, or before newline at the end
\z Match only at end of string
\G Match only at pos() (e.g. at the end-of-match position
of prior m//g)
```

The POSIX character class syntax

The POSIX character class syntax:

```
[:class:]
```

is also available. Note that the `[` and `]` brackets are literal; they must always be used within a character class expression.

```
# this is correct:
$string =~ /[[:alpha:]]/;

# this is not, and will generate a warning:
$string =~ /[:alpha:]/;
```

Available classes

The available classes and their backslash equivalents (if available) are as follows:

```
alpha
alnum
ascii
blank
cntrl
digit \d
graph
lower
print
punct
space \s
upper
word \w
xdigit
```

For example use `[:upper:]` to match all the uppercase characters. Note that the `[]` are part of the `[::]` construct, not part of the whole character class. For example:

```
[01[:alpha:]%]
```

matches zero, one, any alphabetic character, and the percent sign.

Equivalences to Unicode

The following equivalences to Unicode `\p{}` constructs and equivalent backslash character classes (if available), will hold:

```
[[:...:]] \p{...} backslash

alpha IsAlpha
alnum IsAlnum
ascii IsASCII
blank
cntrl IsCntrl
digit IsDigit \d
graph IsGraph
lower IsLower
print IsPrint
punct IsPunct
space IsSpace
IsSpacePerl \s
upper IsUpper
word IsWord \w
xdigit IsXDigit
```

Negated character classes

You can negate the `[::]` character classes by prefixing the class name with a `'^'`. This is a Perl extension. For example:

```
POSIX traditional Unicode

[[:^digit:]] \D \P{IsDigit}
[[:^space:]] \S \P{IsSpace}
[[:^word:]] \W \P{IsWord}
```

3.1.4. Variables especiales después de un emparejamiento

Despues de un emparejamiento con éxito, las siguientes variables especiales quedan definidas:

`$&`	El texto que casó
`` $` ``	El texto que está a la izquierda de lo que casó
`$'`	El texto que está a la derecha de lo que casó
`$1, $2, $3`, etc.	Los textos capturados por los paréntesis
`$+`	Una copia del `$1, $2,` ... con número mas alto
`@-`	Desplazamientos de las subcadenas que casan en `$1` ...
`@+`	Desplazamientos de los finales de las subcadenas en `$1` ...
`$#-`	El índice del último paréntesis que casó
`$#+`	El índice del último paréntesis en la última expresión regular

Las Variables de match, pre-match y post-mach

Ejemplo:

```
#!/usr/bin/perl -w
if ("Hello there, neighbor" =~ /\s(\w+),/) {
  print "That was: ($`)($&)($').\n",
}
```

```
> matchvariables.pl
That was: (Hello)( there,)( neighbor).
```

El uso de estas variables tenía un efecto negativo en el rendimiento de la regexp. Véase `perlfaq6` la sección `` Why does using $&, $`, or $' slow my program down? ``.

> *Once Perl sees that you need one of these variables anywhere in the program, it provides them on each and every pattern match. That means that on every pattern match the entire string will be copied, part of it to* `` $` ``*, part to* `$&`*, and part to* `$'`*. Thus the penalty is most severe with long strings and patterns that match often. Avoid* `$&`*,* `$'`*, and* `` $` `` *if you can, but if you can't, once you've used them at all, use them at will because you've already paid the price. Remember that some algorithms really appreciate them. As of the 5.005 release, the* `$&` *variable is no longer .expensive"the way the other two are.*
>
> *Since Perl 5.6.1 the special variables* `@-` *and* `@+` *can functionally replace* `` $` ``*,* `$&` *and* `$'`*. These arrays contain pointers to the beginning and end of each match (see* `perlvar` *for the full story), so they give you essentially the same information, but without the risk of excessive string copying.*
>
> *Perl 5.10 added three specials,* `${^MATCH}`*,* `${^PREMATCH}`*, and* `${^POSTMATCH}` *to do the same job but without the global performance penalty. Perl 5.10 only sets these variables if you compile or execute the regular expression with the* `/p` *modifier.*

```
pl@nereida:~/Lperltesting$ cat ampersandoldway.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1 -w
use strict;
use Benchmark qw(cmpthese timethese);

'hola juan' =~ /ju/;
my ($a, $b, $c) = ($`, $&, $');


cmpthese( -1, {
    oldway => sub { 'hola juan' =~ /ju/  },
});
pl@nereida:~/Lperltesting$ cat ampersandnewway.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1 -w
use strict;
use Benchmark qw(cmpthese timethese);

'hola juan' =~ /ju/p;
my ($a, $b, $c) = (${^PREMATCH}, ${^MATCH}, ${^POSTMATCH});


cmpthese( -1, {
    newway => sub { 'hola juan' =~ /ju/  },
});

pl@nereida:~/Lperltesting$ time ./ampersandoldway.pl
            Rate oldway
oldway 2991861/s     --

real    0m3.761s
user    0m3.740s
sys     0m0.020s
pl@nereida:~/Lperltesting$ time ./ampersandnewway.pl
            Rate newway
newway 8191999/s     --

real    0m6.721s
user    0m6.704s
sys     0m0.016s
```

Véase

- `perlvar` (busque por `$MATCH`)

Texto Asociado con el Último Paréntesis

La variable `$+` contiene el texto que casó con el último paréntesis en el patrón. Esto es útil en situaciones en las cuáles una de un conjunto de alternativas casa, pero no sabemos cuál:

```
  DB<9> "Revision: 4.5" =~ /Version: (.*)|Revision: (.*)/ && ($rev = $+);
  DB<10> x $rev
0  4.5
  DB<11> "Version: 4.5" =~ /Version: (.*)|Revision: (.*)/ && ($rev = $+);
  DB<12> x $rev
0  4.5
```

Los Offsets de los Inicios de los Casamientos: `@-`

El vector `@-` contiene los *offsets* o desplazamientos de los casamientos en la última expresión regular. La entrada `$-[0]` es el desplazamiento del último casamiento con éxito y `$-[n]` es el desplazamiento de la subcadena que casa con el `n`-ésimo paréntesis (o `undef` si el párentesis no casó). Por ejemplo:

```
#           012345678
DB<1> $z = "hola13.47"
DB<2> if ($z =~ m{a(\d+)(\.(\d+))?}) { print "@-\n"; }
3 4 6 7
```

El resultado se interpreta como sigue:

- 3 = desplazamiento de comienzo de `$& = a13.47`
- 4 = desplazamiento de comienzo de `$1 = 13`
- 6 = desplazamiento de comienzo de `$2 = .`
- 7 = desplazamiento de comienzo de `$3 = 47`

Esto es lo que dice `perlvar` sobre `@-`:

> *This array holds the offsets of the beginnings of the last successful submatches in the currently active dynamic scope.* `$-[0]` *is the offset into the string of the beginning of the entire match. The nth element of this array holds the offset of the nth submatch, so* `$-[1]` *is the offset where* `$1` *begins,* `$-[2]` *the offset where* `$2` *begins, and so on.*
>
> *After a match against some variable* `$var`*:*
>
> ```
> $` is the same as substr($var, 0, $-[0])
> $& is the same as substr($var, $-[0], $+[0] - $-[0])
> $' is the same as substr($var, $+[0])
> $1 is the same as substr($var, $-[1], $+[1] - $-[1])
> $2 is the same as substr($var, $-[2], $+[2] - $-[2])
> $3 is the same as substr($var, $-[3], $+[3] - $-[3])
> ```

Desplazamientos de los Finales de los Emparejamientos: `@+`

El array `@+` contiene los desplazamientos de los finales de los emparejamientos. La entrada `$+[0]` contiene el desplazamiento del final de la cadena del emparejamiento completo. Siguiendo con el ejemplo anterior:

```
#            0123456789
DB<17> $z = "hola13.47x"
DB<18> if ($z =~ m{a(\d+)(\.)(\d+)?}) { print "@+\n"; }
9 6 7 9
```

El resultado se interpreta como sigue:

- 9 = desplazamiento final de `$& = a13.47x`
- 6 = desplazamiento final de `$1 = 13`
- 7 = desplazamiento final de `$2 = .`
- 9 = desplazamiento final de `$3 = 47`

Número de paréntesis en la última regexp con éxito

Se puede usar `$#+` para determinar cuantos parentesis había en el último emparejamiento que tuvo éxito.

```
  DB<29> $z = "h"
  DB<30> print "$#+\n" if ($z =~ m{(a)(b)}) || ($z =~ m{(h)(.)?(.)?})
3
  DB<31> $z = "ab"
  DB<32> print "$#+\n" if ($z =~ m{(a)(b)}) || ($z =~ m{(h)(.)?(.)?})
2
```

Indice del Ultimo Paréntesis

La variable `$#-` contiene el índice del último paréntesis que casó. Observe la siguiente ejecución con el depurador:

```
  DB<1> $x = '13.47'; $y = '125'
  DB<2> if ($y =~ m{(\d+)(\.(\d+))?}) { print "last par = $#-, content = $+\n"; }
last par = 1, content = 125
  DB<3> if ($x =~ m{(\d+)(\.(\d+))?}) { print "last par = $#-, content = $+\n"; }
last par = 3, content = 47
```

`@-` y `@+` no tienen que tener el mismo tamaño

En general no puede asumirse que `@-` y `@+` sean del mismo tamaño.

```
  DB<1> "a" =~ /(a)|(b)/; @a = @-; @b = @+
  DB<2> x @a
0  0
1  0
  DB<3> x @b
0  1
1  1
2  undef
```

Véase También

Para saber más sobre las variables especiales disponibles consulte

- `perldoc perlretut`
- `perldoc perlvar`.

3.1.5. Ambito Automático

Como sabemos, ciertas variables (como `$1`, `$&` ...) reciben automáticamente un valor con cada operación de "matching".

Considere el siguiente código:

```
if (m/(...)/) {
  &do_something();
  print "the matched variable was $1.\n";
}
```

Puesto que `$1` es automáticamente declarada `local` a la entrada de cada bloque, no importa lo que se haya hecho en la función `&do_something()`, el valor de `$1` en la sentencia `print` es el correspondiente al "matching" realizado en el `if`.

3.1.6. Opciones

Modificador	Significado
e	evaluar: evaluar el lado derecho de una sustitución como una expresión
g	global: Encontrar todas las ocurrencias
i	ignorar: no distinguir entre mayúsculas y minúsculas
m	multilínea (`^` y `$` casan con `\n` internos)
o	optimizar: compilar una sola vez
s	`^` y `$` ignoran `\n` pero el punto `.` "casa" con `\n`
x	extendida: permitir comentarios

El Modificador `/g` La conducta de este modificador depende del contexto. En un contexto de listas devuelve una lista con todas las subcadenas casadas por todos los paréntesis en la expresión regular. Si no hubieran paréntesis devuelve una lista con todas las cadenas casadas (como si hubiera paréntesis alrededor del patrón global).

```
#!/usr/bin/perl -w
($one, $five, $fifteen) = (`uptime` =~ /(\d+\.\d+)/g);
print "$one, $five, $fifteen\n";
```

Observe la salida:

```
> uptime
  1:35pm  up 19:22,  0 users,  load average: 0.01, 0.03, 0.00
> glist.pl
0.01, 0.03, 0.00
```

En un contexto escalar `m//g` itera sobre la cadena, devolviendo cierto cada vez que casa, y falso cuando deja de casar. En otras palabras, recuerda donde se quedo la última vez y se recomienza la búsqueda desde ese punto. Se puede averiguar la posicion del emparejamiento utilizando la función `pos`. Si por alguna razón modificas la cadena en cuestión, la posición de emparejamiento se reestablece al comienzo de la cadena.

```
#!/usr/bin/perl -w
# count sentences in a document
#defined as ending in [.!?] perhaps with
# quotes or parens on either side.
$/ = ""; # paragraph mode
while ($paragraph = <>) {
  print $paragraph;
  while ($paragraph =~ /[a-z]['")]*[.!?]+['")]*\s/g) {
    $sentences++;
  }
}
print "$sentences\n";
```

Observe el uso de la variable especial `$/`. Esta variable contiene el separador de registros en el fichero de entrada. Si se iguala a la cadena vacía usará las líneas en blanco como separadores. Se le puede dar el valor de una cadena multicarácter para usarla como delimitador. Nótese que establecerla a `\n\n` es diferente de asignarla a `""`. Si se deja `undef`, la siguiente lectura leerá todo el fichero.

Sigue un ejemplo de ejecución. El programa se llama `gscalar.pl`. Introducimos el texto desde `STDIN`. El programa escribe el número de párrafos:

```
> gscalar.pl
este primer parrafo. Sera seguido de un
segundo parrafo.
```

"Cita de Seneca".

3

La opción `e`: Evaluación del remplazo La opción `/e` permite la evaluación como expresión perl de la cadena de reemplazo (En vez de considerarla como una cadena delimitada por doble comilla).

```
#!/usr/bin/perl -w
$_ = "abc123xyz\n";
s/\d+/$&*2/e;
print;
s/\d+/sprintf("%5d",$&)/e;
print;
s/\w/$& x 2/eg;
print;
```

El resultado de la ejecución es:

```
> replacement.pl
abc246xyz
abc  246xyz
aabbcc  224466xxyyzz
```

Véase un ejemplo con anidamiento de `/e`:

```
#!/usr/bin/perl
$a ="one";
$b = "two";
$_ = '$a $b';
print "_ = $_\n\n";
s/(\$\w+)/$1/ge;
print "After 's/(\$\w+)/$1/ge' _ = $_\n\n";
s/(\$\w+)/$1/gee;
print "After 's/(\$\w+)/$1/gee' _ = $_\n\n";
```

El resultado de la ejecución es:

```
> enested.pl
_ = $a $b

After 's/($w+)/$b/ge' _ = $a $b

After 's/($w+)/$b/gee' _ = one two
```

He aqui una solución que hace uso de `e` al siguiente ejercicio (véase 'Regex to add space after punctuation sign' en PerlMonks) Se quiere poner un espacio en blanco después de la aparición de cada coma:

```
s/,/, /g;
```

pero se quiere que la sustitución no tenga lugar si la coma esta incrustada entre dos dígitos. Además se pide que si hay ya un espacio después de la coma, no se duplique

```
s/(\d[,.]\d)|(,(?!\s))/$1 || "$2 "/ge;
```

Se hace uso de un lookahead negativo `(?!\s)`. Véase la sección 3.2.3 para entender como funciona un lookahead negativo.

3.2. Algunas Extensiones

3.2.1. Comentarios

`(?#text)` Un comentario. Se ignora `text`. Si se usa la opción `x` basta con poner `#`.

3.2.2. Modificadores locales

Los modificadores de la conducta de una expresión regular pueden ser empotrados en una subexpresión usando el formato `(?pimsx-imsx)`.

Véase el correspondiente texto *Extended Patterns* de la sección 'Extended-Patterns' en `perlre`:

> *One or more embedded pattern-match modifiers, to be turned on (or turned off, if preceded by '`-`') for the remainder of the pattern or the remainder of the enclosing pattern group (if any). This is particularly useful for dynamic patterns, such as those read in from a configuration file, taken from an argument, or specified in a table somewhere. Consider the case where some patterns want to be case sensitive and some do not: The case insensitive ones merely need to include (?i) at the front of the pattern. For example:*
>
> ```
> $pattern = "foobar";
> if ( /$pattern/i ) { }
>
> # more flexible:
>
> $pattern = "(?i)foobar";
> if ( /$pattern/ ) { }
> ```
>
> *These modifiers are restored at the end of the enclosing group. For example,*
>
> ```
> ( (?i) blah ) \s+ \1
> ```
>
> *will match* `blah` *in any case, some spaces, and an exact (including the case!) repetition of the previous word, assuming the* `/x` *modifier, and no* `/i` *modifier outside this group.*

El siguiente ejemplo extiende el ejemplo visto en la sección 3.1.1 eliminando los comentarios `/* ... */` y `// ...` de un programa C. En dicho ejemplo se usaba el modificador `s` para hacer que el punto casara con cualquier carácter:

```
casiano@tonga:~/Lperltesting$ cat -n extendedcomments.pl
#!/usr/bin/perl -w
use strict;

my $progname = shift @ARGV or die "Usage:\n$0 prog.c\n";
open(my $PROGRAM,"<$progname") || die "can't find $progname\n";
my $program = '';
{
  local $/ = undef;
  $program = <$PROGRAM>;
}
$program =~ s{(?xs)
  /\*  # Match the opening delimiter
  .*?  # Match a minimal number of characters
  \*/  # Match the closing delimiter
  |
    (?-s)//.* # C++ // comments. No s modifier
}[]g;

print $program;
```

Sigue un ejemplo de ejecución. Usaremos como entrada el programa C:

```
casiano@tonga:~/Lperltesting$ cat -n ehello.c
     1  #include <stdio.h>
     2  /* first
     3  comment
     4  */
     5  main() { // A C++ comment
     6    printf("hello world!\n"); /* second comment */
     7  } // final comment
```

Al ejecutar el programa eliminamos los comentarios:

```
casiano@tonga:~/Lperltesting$ extendedcomments.pl ehello.c | cat -n
     1  #include <stdio.h>
     2
     3  main() {
     4    printf("hello world!\n");
     5  }
```

3.2.3. Mirando hacia adetrás y hacia adelante

El siguiente fragmento esta 'casi' literalmente tomado de la sección 'Looking-ahead-and-looking-behind' en `perlretut`:

Las zero-width assertions como caso particular de mirar atrás-adelante

In Perl regular expressions, most regexp elements 'eat up' a certain amount of string when they match. For instance, the regexp element `[abc}]` *eats up one character of the string when it matches, in the sense that Perl moves to the next character position in the string after the match. There are some elements, however, that don't eat up characters (advance the character position) if they match.*

The examples we have seen so far are the anchors. The anchor `^` *matches the beginning of the line, but doesn't eat any characters.*

Similarly, the word boundary anchor `\b` *matches wherever a character matching* `\w` *is next to a character that doesn't, but it doesn't eat up any characters itself.*

Anchors are examples of zero-width assertions. *Zero-width, because they consume no characters, and assertions, because they test some property of the string.*

In the context of our walk in the woods analogy to regexp matching, most regexp elements move us along a trail, but anchors have us stop a moment and check our surroundings. If the local environment checks out, we can proceed forward. But if the local environment doesn't satisfy us, we must backtrack.

Checking the environment entails either looking ahead on the trail, looking behind, or both.

- `^` *looks behind, to see that there are no characters before.*
- `$` *looks ahead, to see that there are no characters after.*
- `\b` *looks both ahead and behind, to see if the characters on either side differ in their "word-ness".*

The lookahead and lookbehind assertions are generalizations of the anchor concept. Lookahead and lookbehind are zero-width assertions that let us specify which characters we want to test for.

Lookahead assertion

The lookahead assertion is denoted by `(?=regexp)` *and the lookbehind assertion is denoted by* `(?<=fixed-regexp)`.

En español, operador de "trailing" o "mirar-adelante" positivo. Por ejemplo, `/\w+(?=\t)/` *solo casa una palabra si va seguida de un tabulador, pero el tabulador no formará parte de* `$&`. *Ejemplo:*

```
> cat -n lookahead.pl
   1  #!/usr/bin/perl
   2
   3   $a = "bugs the rabbit";
   4   $b = "bugs the frog";
   5   if ($a =~ m{bugs(?= the cat| the rabbit)}i) { print "$a matches. \$& = $&\n"; }
   6   else { print "$a does not match\n"; }
   7   if ($b =~ m{bugs(?= the cat| the rabbit)}i) { print "$b matches. \$& = $&\n"; }
   8   else { print "$b does not match\n"; }
```

Al ejecutar el programa obtenemos:

```
> lookahead.pl
bugs the rabbit matches. $& = bugs
bugs the frog does not match
>
```

Some examples using the debugger[4]*:*

```
  DB<1>      #012345678901234567890
  DB<2> $x = "I catch the housecat 'Tom-cat' with catnip"
  DB<3> print "($&) (".pos($x).")\n" if $x  =~ /cat(?=\s)/g
(cat) (20)                    # matches 'cat' in 'housecat'

  DB<5> $x = "I catch the housecat 'Tom-cat' with catnip" # To reset pos
  DB<6> x @catwords = ($x =~ /(?<=\s)cat\w+/g)
0  'catch'
1  'catnip'

  DB<7>      #012345678901234567890123456789
  DB<8> $x = "I catch the housecat 'Tom-cat' with catnip"
  DB<9> print "($&) (".pos($x).")\n" if $x =~ /\bcat\b/g
(cat) (29) # matches 'cat' in 'Tom-cat'

  DB<10> $x = "I catch the housecat 'Tom-cat' with catnip"
  DB<11> x  $x =~ /(?<=\s)cat(?=\s)/
  empty array
  DB<12> # doesn't match; no isolated 'cat' in middle of $x
```

A hard RegEx problem

Véase el nodo A hard RegEx problem en PerlMonks. Un monje solicita:

> *Hi Monks,*
> *I wanna to match this issues:*
>
> 1. *The string length is between 3 and 10*
> 2. *The string ONLY contains [0-9] or [a-z] or [A-Z], but*
> 3. *The string must contain a number AND a letter at least*
>
> *Pls help me check. Thanks*

Solución:

[4]catnip: La nepeta cataria, también llamada menta de los gatos, de la familia del tomillo y la lavanda. Su perfume desencadena un comportamiento en el animal, similar al del celo

```
casiano@millo:~$ perl -wde 0
main::(-e:1):   0
  DB<1> x 'aaa2a1' =~  /\A(?=.*[a-z])(?=.*\d)\w{3,10}\z/i
0  1
  DB<2> x 'aaaaaa' =~  /\A(?=.*[a-z])(?=.*\d)\w{3,10}\z/i
  empty array
  DB<3> x '1111111' =~  /\A(?=.*[a-z])(?=.*\d)\w{3,10}\z/i
  empty array
  DB<4> x '1111111bbbbb' =~  /\A(?=.*[a-z])(?=.*\d)\w{3,10}\z/i
  empty array
  DB<5> x '111bbbbb' =~  /\A(?=.*[a-z])(?=.*\d)\w{3,10}\z/i
0  1
```

Los paréntesis looakehaed and lookbehind no capturan

Note that the parentheses in `(?=regexp)` *and* `(?<=regexp)` *are non-capturing, since these are zero-width assertions.*

Limitaciones del lookbehind

Lookahead `(?=regexp)` *can match arbitrary regexps, but lookbehind* `(?<=fixed-regexp)` *only works for regexps of fixed width, i.e., a fixed number of characters long.*

Thus `(?<=(ab|bc))` *is fine, but* `(?<=(ab)*)` *is not.*

Negación de los operadores de lookahead y lookbehind

The negated versions of the lookahead and lookbehind assertions are denoted by `(?!regexp)` *and* `(?<!fixed-regexp)` *respectively. They evaluate true if the regexps do not match:*

```
$x = "foobar";
$x =~ /foo(?!bar)/;  # doesn't match, 'bar' follows 'foo'
$x =~ /foo(?!baz)/;  # matches, 'baz' doesn't follow 'foo'
$x =~ /(?<!\s)foo/;  # matches, there is no \s before 'foo'
```

Ejemplo: split con lookahead y lookbehind

Here is an example where a string containing blank-separated words, numbers and single dashes is to be split into its components.

Using `/\s+/` *alone won't work, because spaces are not required between dashes, or a word or a dash. Additional places for a split are established by looking ahead and behind:*

```
casiano@tonga:~$ perl5.10.1 -wdE 0
main::(-e:1):   0
  DB<1> $str = "one two - --6-8"
  DB<2> x @toks = split / \s+ | (?<=\S) (?=-) | (?<=-)  (?=\S)/x, $str
0  'one'
1  'two'
2  '-'
3  '-'
4  '-'
5  6
6  '-'
7  8
```

Look Around en perlre

El siguiente párrafo ha sido extraído la sección 'Look-Around-Assertions' en `pelre`. Usémoslo como texto de repaso:

Look-around assertions are zero width patterns which match a specific pattern without including it in `$&`*. Positive assertions match when their subpattern matches, negative assertions match when their subpattern fails. Look-behind matches text up to the current match position, look-ahead matches text following the current match position.*

- `(?=pattern)`

  A zero-width positive look-ahead assertion. For example, `/\w+(?=\t)/` *matches a word followed by a tab, without including the tab in* `$&`*.*

- `(?!pattern)`

  A zero-width negative look-ahead assertion. For example `/foo(?!bar)/` *matches any occurrence of* `foo` *that isn't followed by* `bar`*.*

  Note however that look-ahead and look-behind are NOT the same thing. You cannot use this for look-behind.

  If you are looking for a `bar` *that isn't preceded by a* `foo`*,* `/(?!foo)bar/` *will not do what you want.*

  That's because the `(?!foo)` *is just saying that the next thing cannot be* `foo` *–and it's not, it's a* `bar`*, so* `foobar` *will match.*

  You would have to do something like `/(?!foo)...bar/` *for that.*

  We say "like"because there's the case of your `bar` *not having three characters before it.*

  You could cover that this way: `/(?:(?!foo)...|^.{0,2})bar/`*. Sometimes it's still easier just to say:*

  ```
  if (/bar/ && $` !~ /foo$/)
  ```

  For look-behind see below.

- `(?<=pattern)`

  A zero-width positive look-behind assertion.

  For example, `/(?<=\t)\w+/` *matches a word that follows a tab, without including the tab in* `$&`*. Works only for fixed-width look-behind.*

- `\K`

  There is a special form of this construct, called `\K` *, which causes the regex engine to 'keep' everything it had matched prior to the* `\K` *and not include it in* `$&`*. This effectively provides variable length look-behind. The use of* `\K` *inside of another look-around assertion is allowed, but the behaviour is currently not well defined.*

  For various reasons `\K` *may be significantly more efficient than the equivalent* `(?<=...)` *construct, and it is especially useful in situations where you want to efficiently remove something following something else in a string. For instance*

  ```
  s/(foo)bar/$1/g;
  ```

  can be rewritten as the much more efficient

  ```
  s/foo\Kbar//g;
  ```

  Sigue una sesión con el depurador que ilustra la semántica del operador:

  ```
  casiano@millo:~$ perl5.10.1 -wdE 0
  main::(-e:1):   0
    DB<1> print "& = <$&> 1 = <$1>\n" if "alphabet" =~ /([^aeiou][a-z][aeiou])[a-z]/
  & = <phab> 1 = <pha>
    DB<2> print "& = <$&> 1 = <$1>\n" if "alphabet" =~ /\K([^aeiou][a-z][aeiou])[a-z]/
  & = <phab> 1 = <pha>
    DB<3> print "& = <$&> 1 = <$1>\n" if "alphabet" =~ /([^aeiou]\K[a-z][aeiou])[a-z]/
  & = <hab> 1 = <pha>
  ```

```
  DB<4> print "& = <$&> 1 = <$1>\n" if "alphabet" =~ /([^aeiou][a-z]\K[aeiou])[a-z]/
& = <ab> 1 = <pha>
  DB<5> print "& = <$&> 1 = <$1>\n" if "alphabet" =~ /([^aeiou][a-z][aeiou])\K[a-z]/
& = <b> 1 = <pha>
  DB<6> print "& = <$&> 1 = <$1>\n" if "alphabet" =~ /([^aeiou][a-z][aeiou])[a-z]\K/
& = <> 1 = <pha>
  DB<7> @a = "alphabet" =~ /([aeiou]\K[^aeiou])/g; print "$&\n"
t
  DB<8> x @a
0  'al'
1  'ab'
2  'et'
```

Otro ejemplo: eliminamos los blancos del final en una cadena:

```
  DB<23> $x = '  cadena entre blancos  '
  DB<24> ($y = $x) =~ s/.*\b\K.*//g
  DB<25> p "<$y>"
<  cadena entre blancos>
```

- `(?<!pattern)`
  A zero-width negative look-behind assertion.
  For example `/(?<!bar)foo/` *matches any occurrence of* `foo` *that does not follow* `bar`*.*
  Works only for fixed-width look-behind.

Veamos un ejemplo de uso. Se quiere sustituir las extensiones `.something` por `.txt` en cadenas que contienen una ruta a un fichero:

```
casiano@millo:~$ perl5.10.1 -wdE 0
main::(-e:1):   0
  DB<1> ($b = $a = 'abc/xyz.something') =~ s{\.[^.]*$}{.txt}
  DB<2> p $b
abc/xyz.txt
  DB<3> ($b = $a = 'abc/xyz.something') =~ s/\.\K[^.]*$/txt/;
  DB<4> p $b
abc/xyz.txt
  DB<5> p $a
abc/xyz.something
```

Véase también:

- `Regexp::Keep` por Jeff Pinyan
- El nodo *positive look behind regexp mystery* en PerlMonks

Operador de predicción negativo: Última ocurrencia

Escriba una expresión regular que encuentre la última aparición de la cadena `foo` en una cadena dada.

```
  DB<6> x ($a = 'foo foo bar bar foo bar bar') =~ /foo(?!.*foo)/g; print pos($a)."\n"
19
  DB<7> x ($a = 'foo foo bar bar foo bar bar') =~ s/foo(?!.*foo)/\U$&/
0  1
  DB<8> x $a
0  'foo foo bar bar FOO bar bar'
```

Diferencias entre mirar adelante negativo y mirar adelante con clase negada

Aparentemente el operador “mirar-adelante” negativo es parecido a usar el operador “mirar-adelante” positivo con la negación de una clase.

`/regexp(?![abc])/`	`/regexp(?=[^abc])/`

Sin embargo existen al menos dos diferencias:

- Una negación de una clase debe casar algo para tener éxito. Un ‘mirar-adelante” negativo tiene éxito si, en particular no logra casar con algo. Por ejemplo:

  `\d+(?!\.)` casa con `$a = '452'`, mientras que `\d+(?=[^.])` lo hace, pero porque `452` es `45` seguido de un carácter que no es el punto:

```
> cat lookaheadneg.pl
#!/usr/bin/perl

 $a = "452";
 if ($a =~ m{\d+(?=[^.])}i) { print "$a casa clase negada. \$& = $&\n"; }
 else { print "$a no casa\n"; }
 if ($a =~ m{\d+(?!\.)}i) { print "$a casa predicción negativa. \$& = $&\n"; }
 else { print "$b no casa\n"; }
nereida:~/perl/src> lookaheadneg.pl
452 casa clase negada. $& = 45
452 casa predicción negativa. $& = 452
```

- Una clase negada casa un único carácter. Un ‘mirar-adelante” negativo puede tener longitud arbitraria.

AND y AND NOT

Otros dos ejemplos:

- `^(?![A-Z]*$)[a-zA-Z]*$`

  casa con líneas formadas por secuencias de letras tales que no todas son mayúsculas. (Obsérvese el uso de las anclas).

- `^(?=.*?esto)(?=.*?eso)`

  casan con cualquier línea en la que aparezcan `esto` y `eso`. Ejemplo:

```
> cat estoyeso.pl
#!/usr/bin/perl

 my $a = shift;

 if ($a =~ m{^(?=.*?esto)(?=.*?eso)}i) { print "$a matches.\n"; }
 else { print "$a does not match\n"; }

>estoyeso.pl 'hola eso y esto'
hola eso y esto matches.
> estoyeso.pl 'hola esto y eso'
hola esto y eso matches.
> estoyeso.pl 'hola aquello y eso'
```

```
hola aquello y eso does not match
> estoyeso.pl 'hola esto y aquello'
hola esto y aquello does not match
```

El ejemplo muestra que la interpretación es que cada operador mirar-adelante se interpreta siempre a partir de la posición actual de búsqueda. La expresión regular anterior es básicamente equivalente a `(/esto/ && /eso/)`.

- `(?!000)(\d\d\d)`

  casa con cualquier cadena de tres dígitos que no sea la cadena `000`.

Lookahead negativo versus lookbehind

Nótese que el "mirar-adelante" negativo no puede usarse fácilmente para imitar un "mirar-atrás", esto es, que no se puede imitar la conducta de `(?<!foo)bar` mediante algo como `(/?!foo)bar`. Tenga en cuenta que:

- Lo que dice `(?!foo)` es que los tres caracteres que siguen no puede ser `foo`.
- Así, `foo` no pertenece a `/(?!foo)bar/`, pero `foobar` pertenece a `(?!foo)bar/` porque `bar` es una cadena cuyos tres siguientes caracteres son `bar` y no son `foo`.
- Si quisieramos conseguir algo parecido a `(?<!foo)bar` usando un lookahead negativo tendríamos que escribir algo asi como `/(?!foo)...bar/` que casa con una cadena de tres caracteres que no sea `foo` seguida de `bar` (pero que tampoco es exactamente equivalente):

```
pl@nereida:~/Lperltesting$ cat -n foobar.pl
use v5.10;
use strict;

my $a = shift;

for my $r (q{(?<!foo)bar}, q{(?!foo)bar}, q{(?!foo)...bar}) {
  if ($a =~ /$r/) {
    say "$a casa con $r"
  }
  else {
    say "$a no casa con $r"
  }
}
```

- Al ejecutar con diferentes entradas el programa anterior vemos que la solución `q{(?!foo)...bar}` se apróxima mas a `(q{(?<!foo)bar}`:

```
pl@nereida:~/Lperltesting$ perl5.10.1 foobar.pl foobar
foobar no casa con (?<!foo)bar
foobar casa con (?!foo)bar
foobar no casa con (?!foo)...bar

pl@nereida:~/Lperltesting$ perl5.10.1 foobar.pl bar
bar casa con (?<!foo)bar
bar casa con (?!foo)bar
bar no casa con (?!foo)...bar
```

Ejercicio 3.2.1. *Explique porqué* `bar` *casa con* `(?<!foo)bar` *pero no con* `(?!foo)...bar`. *¿Sabría encontrar una expresión regular mas apropiada usando lookahead negativo?*

- En realidad, posiblemente sea mas legible una solución como:

```
if (/bar/ and $‘ !~ /foo$/)
```

o aún mejor (véase 3.1.4):

```
if (/bar/p && ${^PREMATCH} =~ /foo$/)
```

El siguiente programa puede ser utilizado para ilustrar la equivalencia:

```
pl@nereida:~/Lperltesting$ cat -n foobarprematch.pl
use v5.10;
use strict;

$_ = shift;

if (/bar/p && ${^PREMATCH} =~ /foo$/) {
  say "$_ no cumple ".q{/bar/p && ${^PREMATCH} =~ /foo$/};
}
else {
  say "$_ cumple ".q{/bar/p && ${^PREMATCH} =~ /foo$/};
}
if (/(?<!foo)bar/) {
  say "$_ casa con (?<!foo)bar"
}
else {
  say "$_ no casa con (?<!foo)bar"
}
```

Siguen dos ejecuciones:

```
pl@nereida:~/Lperltesting$ perl5.10.1 foobarprematch.pl bar
bar cumple /bar/p && ${^PREMATCH} =~ /foo$/
bar casa con (?<!foo)bar
pl@nereida:~/Lperltesting$ perl5.10.1 foobarprematch.pl foobar
foobar no cumple /bar/p && ${^PREMATCH} =~ /foo$/
foobar no casa con (?<!foo)bar
```

Ejercicios

Ejercicio 3.2.2.

- *Escriba una sustitución que reemplaze todas las apariciones de* `foo` *por* `foo,` *usando* `\K` *o lookbehind*
- *Escriba una sustitución que reemplaze todas las apariciones de* `lookahead` *por* `look-ahead` *usando lookaheads y lookbehinds*
- *Escriba una expresión regular que capture todo lo que hay entre las cadenas* `foo` *y* `bar` *siempre que no se incluya la palabra* `baz`
- *¿Cuál es la salida?*

  ```
  DB<1> x 'abc' =~ /(?=(.)(.)(.))a(b)/
  ```

- *Se quiere poner un espacio en blanco después de la aparición de cada coma:*

  ```
  s/,/, /g;
  ```

pero se quiere que la sustitución no tenga lugar si la coma esta incrustada entre dos dígitos.

- *Se quiere poner un espacio en blanco después de la aparición de cada coma:*

  ```
  s/,/, /g;
  ```

  pero se quiere que la sustitución no tenga lugar si la coma esta incrustada entre dos dígitos. Además se pide que si hay ya un espacio después de la coma, no se duplique

- *¿Cuál es la salida?*

  ```
  pl@nereida:~/Lperltesting$ cat -n ABC123.pl
  use warnings;
  use strict;

  my $c = 0;
  my @p = ('^(ABC)(?!123)', '^(\D*)(?!123)',);

  for my $r (@p) {
    for my $s (qw{ABC123 ABC445}) {
      $c++;
      print "$c: '$s' =~ /$r/ : ";
      <>;
      if ($s =~ /$r/) {
        print " YES ($1)\n";
      }
      else {
        print " NO\n";
      }
    }
  }
  ```

3.2.4. Definición de Nombres de Patrones

Perl 5.10 introduce la posibilidad de definir subpatrones en una sección del patrón.

Lo que dice `perlretut` sobre la definición de nombres de patrones
Citando la sección *Defining named patterns* en el documento la sección 'Defining-named-patterns' en `perlretut` para perl5.10:

> *Some regular expressions use identical subpatterns in several places. Starting with Perl 5.10, it is possible to define named subpatterns in a section of the pattern so that they can be called up by name anywhere in the pattern. This syntactic pattern for this definition group is* `"(?(DEFINE)(?<name>pattern)...)"` *An insertion of a named pattern is written as* `(?&name)`.

Veamos un ejemplo que define el lenguaje de los números en punto flotante:

```
pl@nereida:~/Lperltesting$ cat -n definingnamedpatterns.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1 -w
use v5.10;

my $regexp = qr{
    ^ (?<num>
              (?&osg)[\t\ ]* (?: (?&int)(?&dec)? | (?&dec) )
```

```
      )
      (?: [eE]
      (?<exp> (?&osg)(?&int)) )?
    $
       (?(DEFINE)
         (?<osg>[-+]?)         # optional sign
         (?<int>\d++)          # integer
         (?<dec>\.(?&int))     # decimal fraction
       )
}x;

my $input = <>;
chomp($input);
my @r;
if (@r = $input =~ $regexp) {
  my $exp = $+{exp} || '';
  say "$input matches: (num => '$+{num}', exp => '$exp')";
}
else {
  say "does not match";
}
```

`perlretut` comenta sobre este ejemplo:

> *The example above illustrates this feature. The three subpatterns that are used more than once are the optional sign, the digit sequence for an integer and the decimal fraction. The* `DEFINE` *group at the end of the pattern contains their definition. Notice that the decimal fraction pattern is the first place where we can reuse the integer pattern.*

Lo que dice `perlre` sobre la definición de patrones

Curiosamente, `(DEFINE)` se considera un caso particular de las expresiones regulares condicionales de la forma `(?(condition)yes-pattern)` (véase la sección 3.2.10). Esto es lo que dice la sección 'Extended-Patterns' en `perlre` al respecto:

> *A special form is the* `(DEFINE)` *predicate, which never executes directly its yes-pattern, and does not allow a no-pattern. This allows to define subpatterns which will be executed only by using the recursion mechanism. This way, you can define a set of regular expression rules that can be bundled into any pattern you choose.*
>
> *It is recommended that for this usage you put the* `DEFINE` *block at the end of the pattern, and that you name any subpatterns defined within it.*
>
> *Also, it's worth noting that patterns defined this way probably will not be as efficient, as the optimiser is not very clever about handling them.*
>
> *An example of how this might be used is as follows:*
>
> ```
> /(?<NAME>(?&NAME_PAT))(?<ADDR>(?&ADDRESS_PAT))
>        (?(DEFINE)
>          (?<NAME_PAT>....)
>          (?<ADRESS_PAT>....)
> )/x
> ```
>
> *Note that* **capture buffers matched inside of recursion are not accessible after the recursion returns**, *so the extra layer of capturing buffers is necessary. Thus* `$+{NAME_PAT}` *would not be defined even though* `$+{NAME}` *would be.*

Lo que dice perlvar sobre patrones con nombre Esto es lo que dice `perlvar` respecto a las variables implicadas `%+` y `%-`. Con respecto a el hash `%+`:

- `%LAST_PAREN_MATCH`, `%+`

  Similar to `@+` *, the* `%+` *hash allows access to the named capture buffers, should they exist, in the last successful match in the currently active dynamic scope.*

  For example, `$+{foo}` *is equivalent to* `$1` *after the following match:*

  ```
  'foo' =~ /(?<foo>foo)/;
  ```

  The keys of the `%+` *hash list only the names of buffers that have captured (and that are thus associated to defined values).*

  The underlying behaviour of `%+` *is provided by the* `Tie::Hash::NamedCapture` *module.*

  Note: `%-` *and* `%+` *are tied views into a common internal hash associated with the last successful regular expression. Therefore mixing iterative access to them via* `each` *may have unpredictable results. Likewise, if the last successful match changes, then the results may be surprising.*

- `%-`

  Similar to `%+` *, this variable allows access to the named capture buffers in the last successful match in the currently active dynamic scope.* **To each capture buffer name found in the regular expression, it associates a reference to an array containing the list of values captured by all buffers with that name (should there be several of them), in the order where they appear**.

  Here's an example:

  ```
  if ('1234' =~ /(?<A>1)(?<B>2)(?<A>3)(?<B>4)/) {
    foreach my $bufname (sort keys %-) {
      my $ary = $-{$bufname};
      foreach my $idx (0..$#$ary) {
        print "\$-{$bufname}[$idx] : ",
              (defined($ary->[$idx]) ? "'$ary->[$idx]'" : "undef"),
              "\n";
      }
    }
  }
  ```

  would print out:

  ```
  $-{A}[0] : '1'
  $-{A}[1] : '3'
  $-{B}[0] : '2'
  $-{B}[1] : '4'
  ```

  The keys of the `%-` *hash correspond to all buffer names found in the regular expression.*

3.2.5. Patrones Recursivos

Perl 5.10 introduce la posibilidad de definir subpatrones en una sección del patrón. Citando la versión del documento `perlretut` para perl5.10:

> *This feature (introduced in Perl 5.10) significantly extends the power of Perl's pattern matching. By referring to some other capture group anywhere in the pattern with the construct* `(?group-ref)`*, the pattern within the referenced group is used as an independent subpattern in place of the group reference itself. Because the group reference may be contained within the group it refers to, it is now possible to apply pattern matching to tasks that hitherto required a recursive parser.*

...

In `(?...)` *both absolute and relative backreferences may be used. The entire pattern can be reinserted with* `(?R)` *or* `(?0)`*. If you prefer to name your buffers, you can use* `(?&name)` *to recurse into that buffer.*

Palíndromos

Véase un ejemplo que reconoce los palabra-palíndromos (esto es, la lectura directa y la inversa de la cadena pueden diferir en los signos de puntuación):

```
casiano@millo:~/Lperltesting$ cat -n palindromos.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1 -w
use v5.10;

my $regexp = qr/^(\W*
                   (?:
                         (\w) (?1) \g{-1}  # palindromo estricto
                       |
                         \w?               # no recursiva
                   )
                \W*)$/ix;

my $input = <>;
chomp($input);
if ($input =~ $regexp) {
  say "$input is a palindrome";
}
else {
  say "does not match";
}
```

Ejercicio 3.2.3. *¿Cuál es el efecto del modificador* `i` *en la regexp* `qr/^(\W* (?: (\w) (?1) \g{-1} | \w? ) \W*`

Siguen algunos ejemplos de ejecución[5]

```
pl@nereida:~/Lperltesting$ ./palindromos.pl
A man, a plan, a canal: Panama!
A man, a plan, a canal: Panama! is a palindrome
pl@nereida:~/Lperltesting$ ./palindromos.pl
A man, a plan, a cam, a yak, a yam, a canal { Panama!
A man, a plan, a cam, a yak, a yam, a canal { Panama! is a palindrome
pl@nereida:~/Lperltesting$ ./palindromos.pl
A man, a plan, a cat, a ham, a yak, a yam, a hat, a canal { Panama!
A man, a plan, a cat, a ham, a yak, a yam, a hat, a canal { Panama! is a palindrome
pl@nereida:~/Lperltesting$ ./palindromos.pl
saippuakauppias
saippuakauppias is a palindrome
pl@nereida:~/Lperltesting$ ./palindromos.pl
dfghjgfd
does not match
```

[5]

- saippuakauppias: Vendedor de jabón (suomi)
- yam: batata (inglés)
- cam: leva

```
pl@nereida:~/Lperltesting$ ./palindromos.pl
...,;;;;
...,;;;; is a palindrome
```

Lo que dice `perlre` sobre recursividad

> `(?PARNO) (?-PARNO) (?+PARNO) (?R) (?0)`
>
> *Similar to* `(??{ code })` *(véase la sección 3.2.9) except it does not involve compiling any code, instead it treats the contents of a capture buffer as an independent pattern that must match at the current position. Capture buffers contained by the pattern will have the value as determined by the outermost recursion.*
>
> `PARNO` *is a sequence of digits (not starting with 0) whose value reflects the paren-number of the capture buffer to recurse to.*
>
> `(?R)` *recurses to the beginning of the whole pattern.* `(?0)` *is an alternate syntax for* `(?R)`.
>
> *If* `PARNO` *is preceded by a plus or minus sign then it is assumed to be relative, with negative numbers indicating preceding capture buffers and positive ones following. Thus (?-1) refers to the most recently declared buffer, and (?+1) indicates the next buffer to be declared.*
>
> *Note that the counting for relative recursion differs from that of relative backreferences, in that with recursion unclosed buffers are included.*

Hay una diferencia fundamental entre `\g{-1}` y `(?-1)`. El primero significa *lo que casó con el último paréntesis*. El segundo significa que se debe *llamar a la expresión regular que define el último paréntesis*. Véase un ejemplo:

```
pl@nereida:~/Lperltesting$ perl5.10.1 -wdE 0
main::(-e:1):   0
  DB<1>  x ($a = "12 aAbB 34") =~ s/([aA])(?-1)(?+1)([bB])/-\1\2-/g
0  1
  DB<2> p $a
12 -aB- 34
```

En `perlre` también se comenta sobre este punto:

> *If there is no corresponding capture buffer defined, then it is a fatal error. Recursing deeper than 50 times without consuming any input string will also result in a fatal error. The maximum depth is compiled into perl, so changing it requires a custom build.*

Paréntesis Equilibrados

El siguiente programa (inspirado en uno que aparece en `perlre`) reconoce una llamada a una función `foo()` que puede contener una secuencia de expresiones con paréntesis equilibrados como argumento:

```
pl@nereida:~/Lperltesting$ cat perlrebalancedpar.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1  -w
use v5.10;
use strict;

my $regexp = qr{ (                      # paren group 1 (full function)
                foo
                   (                    # paren group 2 (parens)
                     \(
                        (               # paren group 3 (contents of parens)
                           (?:
```

```
                                  [^()]+  # Non-parens
                                |
                                  (?2) # Recurse to start of paren group 2
                             )*
                          )             # 3
                       \)
                      )                 # 2
                )                       # 1
     }x;

my $input = <>;
chomp($input);
my @res = ($input =~ /$regexp/);
if (@res) {
  say "<$&> is balanced\nParen: (@res)";
}
else {
  say "does not match";
}
```

Al ejecutar obtenemos:

```
pl@nereida:~/Lperltesting$ ./perlrebalancedpar.pl
foo(bar(baz)+baz(bop))
<foo(bar(baz)+baz(bop))> is balanced
Paren: (foo(bar(baz)+baz(bop)) (bar(baz)+baz(bop)) bar(baz)+baz(bop))
```

Como se comenta en `perlre` es conveniente usar índices relativos si se quiere tener una expresión regular reciclable:

> *The following shows how using negative indexing can make it easier to embed recursive patterns inside of a* `qr//` *construct for later use:*
>
> ```
> my $parens = qr/(\((?:[^()]++|(?-1))*+\))/;
> if (/foo $parens \s+ + \s+ bar $parens/x) {
>   # do something here...
> }
> ```

Véase la sección 3.2.6 para comprender el uso de los operadores posesivos como `++`.

Capturando los bloques de un programa

El siguiente programa presenta una heurística para determinar los bloques de un programa:

```
pl@nereida:~/Lperltesting$ cat blocks.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1 -w
use v5.10;
use strict;
#use re 'debug';

my $rb = qr{(?x)
    (
      \{               # llave abrir
         (?:
             [^{}]++   # no llaves
```

```
            |
                 [^{}]*+  # no llaves
                 (?1)     # recursivo
                 [^{}]*+  # no llaves
            )*+
          \}              # llave cerrar
       )
     };

  local $/ = undef;
  my $input = <>;
  my@blocks = $input =~ m{$rb}g;
  my $i = 0;
  say($i++.":\n$_\n===") for @blocks;
```

Veamos una ejecución. Le daremos como entrada el siguiente programa: Al ejecutar el programa con esta entrada obtenemos:

```
pl@nereida:~/Lperltesting$ cat -n blocks.c
     1  main() { /* 1 */
     2    { /* 2 */ }
     3    { /* 3 */ }
     4  }
     5
     6  f(){  /* 4 */
     7    {   /* 5 */
     8      { /* 6 */ }
     9    }
    10    {   /* 7 */
    11      { /* 8 */ }
    12    }
    13  }
    14
    15  g(){ /* 9 */
    16  }
    17
    18  h() {
    19  {{{}}}
    20  }
    21  /* end h */
```

```
pl@nereida:~/Lperltesting$ perl5.10.1 blocks.pl b
0:
{ /* 1 */
  { /* 2 */ }
  { /* 3 */ }
}
===
1:
{  /* 4 */
  {   /* 5 */
    { /* 6 */ }
  }
  {   /* 7 */
    { /* 8 */ }
  }
}
===
2:
{ /* 9 */
}
===
3:
{
{{{}}}
}
===
```

Reconocimiento de Lenguajes Recursivos: Un subconjunto de LaTeX

La posibilidad de combinar en las expresiones regulares Perl 5.10 la recursividad con los constructos `(?<name>...)` y `?&name)` así como las secciones `(?(DEFINE) ...)` permiten la escritura de expresiones regulares que reconocen lenguajes recursivos. El siguiente ejemplo muestra un reconocedor de un subconjunto del lenguaje LaTeX (véase la entrada LaTeX en la wikipedia):

```
pl@nereida:~/Lperltesting$ cat latex5_10.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1 -w
```

```
use strict;
use v5.10;

my $regexp = qr{
    \A(?&File)\z

    (?(DEFINE)
        (?<File>     (?&Element)*+\s*
        )

        (?<Element>  \s* (?&Command)
                  |  \s* (?&Literal)
        )

        (?<Command>  \\ \s* (?<L>(?&Literal)) \s* (?<Op>(?&Options)?) \s* (?<A>(?&Args))
           (?{
              say "command: <$+{L}> options: <$+{Op}> args: <$+{A}>"
           })
        )

        (?<Options>  \[ \s* (?:(?&Option) (?:\s*,\s* (?&Option) )*)? \s* \]
        )

        (?<Args>     (?: \{ \s* (?&Element)* \s* \} )*
        )

        (?<Option>   \s* [^][\$&%#_{}~^\s,]+
        )

        (?<Literal>  \s* ([^][\$&%#_{}~^\s]+)
        )
    )
}xms;

my $input = do{ local $/; <>};
if ($input =~ $regexp) {
  say "$@: matches:\n$&";
}
else {
  say "does not match";
}
```

Añadimos una acción semántica al final de la aceptación de un <Command>.

```
(?<Command>  \\ \s* (?<L>(?&Literal)) \s* (?<Op>(?&Options)?) \s* (?<A>(?&Args)?)
   (?{
      say "command: <$+{L}> options: <$+{Op}> args: <$+{A}>"
   })
)
```

Esta acción es ejecutada pero no afecta al proceso de análisis. (véase la sección 3.2.8 para mas información sobre las acciones semánticas en medio de una regexp). La acción se limita a mostrar que ha casado con cada una de las tres componentes: el comando, las opciones y los argumentos.

Los paréntesis adicionales, como en (?<L>(?&Literal)) son necesarios para guardar lo que casó.

Cuando se ejecuta produce la siguiente salida[6]:

```
pl@nereida:~/Lperltesting$ cat prueba.tex
\documentclass[a4paper,11pt]{article}
\usepackage{latexsym}
\author{D. Conway}
\title{Parsing \LaTeX{}}
\begin{document}
\maketitle
\tableofcontents
\section{Description}
...is easy \footnote{But not\\ \emph{necessarily} simple}.
In fact it's easy peasy to do.
\end{document}

pl@nereida:~/Lperltesting$ ./latex5_10.pl prueba.tex
command: <documentclass> options: <[a4paper,11pt]> args: <{article}>
command: <usepackage> options: <> args: <{latexsym}>
command: <author> options: <> args: <{D. Conway}>
command: <LaTeX> options: <> args: <{}>
command: <title> options: <> args: <{Parsing \LaTeX{}}>
command: <begin> options: <> args: <{document}>
command: <maketitle> options: <> args: <>
command: <tableofcontents> options: <> args: <>
command: <section> options: <> args: <{Description}>
command: <emph> options: <> args: <{necessarily}>
command: <footnote> options: <> args: <{But not\\ \emph{necessarily} simple}>
command: <end> options: <> args: <{document}>
: matches:
\documentclass[a4paper,11pt]{article}
\usepackage{latexsym}
\author{D. Conway}
\title{Parsing \LaTeX{}}
\begin{document}
\maketitle
\tableofcontents
\section{Description}
...is easy \footnote{But not\\ \emph{necessarily} simple}.
In fact it's easy peasy to do.
```

[6]

- peasy:A disagreeable taste of very fresh green peas
- easy peasy:
  1. (uk) very easy (short for easy-peasy-lemon-squeezy)
  2. the first half of a rhyming phrase with several alternate second halves, all of which connote an activity or a result that is, respectively, simple to perform or achieve.

     Tie your shoes? Why that's easy peasy lemon squeezy!
     Beat your meat? Why that's easy peasy Japanesey!
     As a red-stater, condemn books and films without having read or seen them? Why that's easy peasy puddin'n'pie!
  3. It comes from a 1970's british TV commercial for Lemon Squeezy detergent. They were with a little girl who points out dirty greasy dishes to an adult (mom or relative) and then this adult produces Lemon Squeezy and they clean the dishes quickly. At the end of the commercial the girl says *Easy Peasy Lemon Squeezy.*

     Today it is a silly way to state something was or will be very easy.

```
\end{document}
```

La siguiente entrada `prueba3.tex` no pertenece al lenguaje definido por el patrón regular, debido a la presencia de la cadena `$In$` en la última línea:

```
pl@nereida:~/Lperltesting$ cat prueba3.tex
\documentclass[a4paper,11pt]{article}
\usepackage{latexsym}
\author{D. Conway}
\title{Parsing \LaTeX{}}
\begin{document}
\maketitle
\tableofcontents
\section{Description}
\comm{a}{b}
...is easy \footnote{But not\\ \emph{necessarily} simple}.
$In$ fact it's easy peasy to do.
\end{document}

pl@nereida:~/Lperltesting$ ./latex5_10.pl prueba3.tex
command: <documentclass> options: <[a4paper,11pt]> args: <{article}>
command: <usepackage> options: <> args: <{latexsym}>
command: <author> options: <> args: <{D. Conway}>
command: <LaTeX> options: <> args: <{}>
command: <title> options: <> args: <{Parsing \LaTeX{}}>
command: <begin> options: <> args: <{document}>
command: <maketitle> options: <> args: <>
command: <tableofcontents> options: <> args: <>
command: <section> options: <> args: <{Description}>
command: <comm> options: <> args: <{a}{b}>
command: <emph> options: <> args: <{necessarily}>
command: <footnote> options: <> args: <{But not\\ \emph{necessarily} simple}>
does not match
```

Ejercicio 3.2.4. *Obsérvese el uso del cuantificador posesivo en:*

```
        (?<File>     (?&Element)*+\s*
        )
```

¿Que ocurrre si se quita el posesivo y se vuelve a ejecutar `$ ./latex5_10.pl prueba3.tex`*?*

Reconocimiento de Expresiones Aritméticas

Véase el nodo Complex regex for maths formulas en perlmonks para la formulación del problema. Un monje pregunta:

> *Hiya monks,*
>
> *Im having trouble getting my head around a regular expression to match sequences. I need to catch all exceptions where a mathematical expression is illegal...*
>
> *There must be either a letter or a digit either side of an operator parenthesis must open and close next to letters or digits, not next to operators, and do not have to exist variables must not be more than one letter Nothing other than* `a-z,A-Z,0-9,+,-,*,/,(,)` *can be used*
>
> *Can anyone offer a hand on how best to tackle this problem?*
>
> *many thanks*

La solución parte de que una *expresión* es o bien un *término* o bien un *término* seguido de una operador y un *término*, esto es:

- `termino`
- `termino op termino op termino` ...

que puede ser unificado como `termino (op termino)*`.

Un *término* es un número o un identificador o una *expresión* entre paréntesis, esto es:

- `numero`
- `identificador`
- `( expresión )`

La siguiente expresión regular recursiva sigue esta idea:

```
pl@nereida:~/Lperltesting$ cat -n simpleexpressionsna.pl
    #!/usr/local/lib/perl/5.10.1/bin//perl5.10.1
    use v5.10;
    use strict;
    use warnings;

    local our ($skip, $term, $expr);
    $skip = qr/\s*/;
    $expr = qr{ (?<EXPR>
                       (?<TERM>              # An expression is a TERM  ...
                              $skip (?<ID>[a-zA-Z]+)
                            | $skip (?<INT>[1-9]\d*)
                            | $skip \(
                              $skip  (?&EXPR)
                              $skip \)
                       ) (?: $skip           # possibly followed by a sequence of ...
                             (?<OP>[-+*/])
                             (?&TERM)        # ... operand TERM pairs
                         )*
                )
              }x;
    my $re = qr/^ $expr $skip \z/x;
    sub is_valid { shift =~ /$re/o }

    my @test = ( '(a + 3)', '(3 * 4)+(b + x)', '(5 - a)*z',
                 '((5 - a))*((((z)))+2)', '3 + 2', '!3 + 2', '3 + 2!',
                 '3 a', '3 3', '3 * * 3',
                 '2 - 3 * 4',  '2 - 3 + 4',
               );
    foreach (@test) {
      say("$_:");
      say(is_valid($_) ? "\n<$_> is valid" : "\n<$_> is not valid")
    }
```

Podemos usar acciones semánticas empotradas para ver la forma en la que trabaja la expresión regular (véase la sección 3.2.8):

```
pl@nereida:~/Lperltesting$ cat -n simpleexpressions.pl
    #!/usr/local/lib/perl/5.10.1/bin//perl5.10.1
    use v5.10;
    use strict;
    use warnings;

    use re ’eval’; # to allow Eval-group at runtime

    local our ($skip, $term, $expr);
    $skip = qr/\s*/;
    $expr = qr{ (?<EXPR>
                       (?<TERM>              # An expression is a TERM  ...
                              $skip (?<ID>[a-zA-Z]+)  (?{ print "[ID $+{ID}] "  })
                            | $skip (?<INT>[1-9]\d*)  (?{ print "[INT $+{INT}] " })
                            | $skip \(                (?{ print "[(] " })
                              $skip  (?&EXPR)
                              $skip \)                (?{ print "[)] " })
                       ) (?: $skip           # possibly followed by a sequence of ...
                             (?<OP>[-+*/])            (?{ print "[OP $+{OP}] " })
                             (?&TERM)        # ... operand TERM pairs
                         )*
                )
              }x;
    my $re = qr/^ $expr $skip \z/x;
    sub is_valid { shift =~ /$re/o }

    my @test = ( ’(a + 3)’, ’(3 * 4)+(b + x)’, ’(5 - a)*z’,
                 ’((5 - a))*((((z)))+2)’, ’3 + 2’, ’!3 + 2’, ’3 + 2!’,
                 ’3 a’, ’3 3’, ’3 * * 3’,
                 ’2 - 3 * 4’,  ’2 - 3 + 4’,
               );
    foreach (@test) {
      say("$_:");
      say(is_valid($_) ? "\n<$_> is valid" : "\n<$_> is not valid")
    }
```

Ejecución:

```
pl@nereida:~/Lperltesting$ ./simpleexpressions.pl
(a + 3):
[(] [ID a] [OP +] [INT 3] [)]
<(a + 3)> is valid
(3 * 4)+(b + x):
[(] [INT 3] [OP *] [INT 4] [)] [OP +] [(] [ID b] [OP +] [ID x] [)]
<(3 * 4)+(b + x)> is valid
(5 - a)*z:
[(] [INT 5] [OP -] [ID a] [)] [OP *] [ID z]
<(5 - a)*z> is valid
((5 - a))*((((z)))+2):
[(] [(] [INT 5] [OP -] [ID a] [)] [)] [OP *] [(] [(] [(] [(] [ID z] [)] [)] [)] [OP +] [INT 2]
<((5 - a))*((((z)))+2)> is valid
3 + 2:
[INT 3] [OP +] [INT 2]
<3 + 2> is valid
```

```
!3 + 2:

<!3 + 2> is not valid
3 + 2!:
[INT 3] [OP +] [INT 2]
<3 + 2!> is not valid
3 a:
[INT 3]
<3 a> is not valid
3 3:
[INT 3]
<3 3> is not valid
3 * * 3:
[INT 3] [OP *]
<3 * * 3> is not valid
2 - 3 * 4:
[INT 2] [OP -] [INT 3] [OP *] [INT 4]
<2 - 3 * 4> is valid
2 - 3 + 4:
[INT 2] [OP -] [INT 3] [OP +] [INT 4]
<2 - 3 + 4> is valid
```

3.2.6. Cuantificadores Posesivos

Por defecto, cuando un subpatrón con un cuantificador impide que el patrón global tenga éxito, se produce un backtrack. Hay ocasiones en las que esta conducta da lugar a ineficiencia.

Perl 5.10 provee los cuantificadores posesivos: Un cuantificador posesivo actúa como un cuantificador greedy pero no se produce backtracking.

`*+`	Casar 0 o mas veces y no retroceder
`++`	Casar 1 o mas veces y no retroceder
`?+`	Casar 0 o 1 veces y no retroceder
`{n}+`	Casar exactamente n veces y no retroceder (redundante)
`{n,}+`	Casar al menos n veces y no retroceder
`{n,m}+`	Casar al menos n veces y no mas de m veces y no retroceder

Por ejemplo, la cadena `'aaaa'` no casa con `/(a++a)/` porque no hay retroceso después de leer las 4 aes:

```
pl@nereida:~/Lperltesting$ perl5.10.1 -wde 0
main::(-e:1):   0
  DB<1> x 'aaaa' =~ /(a+a)/
0  'aaaa'
  DB<2> x 'aaaa' =~ /(a++a)/
  empty array
```

Cadenas Delimitadas por Comillas Dobles

Los operadores posesivos sirven para poder escribir expresiones regulares mas eficientes en aquellos casos en los que sabemos que el retroceso no conducirá a nuevas soluciones, como es el caso del reconocimiento de las cadenas delimitadas por comillas dobles:

```
pl@nereida:~/Lperltesting$ cat -n ./quotedstrings.pl
     1  #!/usr/local/lib/perl/5.10.1/bin//perl5.10.1
     2  use v5.10;
     3
     4  my $regexp = qr/
     5    "             # double quote
```

```
  (?:           # no memory
      [^"\\]++  # no " or escape: Don't backtrack
    | \\.       # escaped character
  )*+
  "             # end double quote
/x;

my $input = <>;
chomp($input);
if ($input =~ $regexp) {
  say "$& is a string";
}
else {
  say "does not match";
}
```

Paréntesis Posesivos

Los paréntesis posesivos `(?> ...)` dan lugar a un reconocedor que rechaza las demandas de retroceso. De hecho, los operadores posesivos pueden ser reescritos en términos de los paréntesis posesivos: La notación `X++` es equivalente a `(?>X+)`.

Paréntesis Balanceados

El siguiente ejemplo reconoce el lenguaje de los paréntesis balanceados:

```
pl@nereida:~/Lperltesting$ cat -n ./balancedparenthesis.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1
use v5.10;

my $regexp =
    qr/^(
           [^()]*+ # no hay paréntesis, no backtrack
          \(
              (?>        # subgrupo posesivo
                 [^()]++ # no hay paréntesis, + posesivo, no backtrack
                |(?1)    # o es un paréntesis equilibrado
              )*
           \)
           [^()]*+ # no hay paréntesis
         )$/x;

my $input = <>;
chomp($input);
if ($input =~ $regexp) {
  say "$& is a balanced parenthesis";
}
else {
  say "does not match";
}
```

Cuando se ejecuta produce una salida como:

```
pl@nereida:~/Lperltesting$ ./balancedparenthesis.pl
(2*(3+4)-5)*2
(2*(3+4)-5)*2 is a balanced parenthesis
```

```
pl@nereida:~/Lperltesting$ ./balancedparenthesis.pl
(2*(3+4)-5))*2
does not match
pl@nereida:~/Lperltesting$ ./balancedparenthesis.pl
2*(3+4
does not match
pl@nereida:~/Lperltesting$ ./balancedparenthesis.pl
4*(2*(3+4)-5)*2
4*(2*(3+4)-5)*2 is a balanced parenthesis
```

Encontrando los bloques de un programa

El uso de los operadores posesivos nos permite reescribir la solución al problema de encontrar los bloques maximales de un código dada en la sección 3.2.5 de la siguiente manera:

```
pl@nereida:~/Lperltesting$ cat blocksopti.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1 -w
use v5.10;
use strict;
#use re 'debug';

my $rb = qr{(?x)
    (
      \{               # llave abrir
         (?:
             [^{}]++   # no llaves
         |
             (?1)      # recursivo
             [^{}]*+   # no llaves
         )*+
       \}              # llave cerrar
    )
  };

local $/ = undef;
my $input = <>;
my@blocks = $input =~ m{$rb}g;
my $i = 0;
say($i++.":\n$_\n===") for @blocks;
```

Véase también

- Possessive Quantifiers en http://www.regular-expressions.info/
- Nodo *Possessive Quantifiers in Perl 5.10 regexps* en PerlMonks
- perldoc `perlre`

3.2.7. Perl 5.10: Numeración de los Grupos en Alternativas

A veces conviene tener una forma de acceso uniforme a la lista proporcionada por los paréntesis con memoria. Por ejemplo, la siguiente expresión regular reconoce el lenguaje de las horas en notaciones civil y militar:

```
pl@nereida:~/Lperltesting$ perl5.10.1 -wde 0
main::(-e:1):   0
```

```
  DB<1> '23:12' =~ /(\d\d|\d):(\d\d)|(\d\d)(\d\d)/; print "1->$1 2->$2\n"
1->23 2->12

  DB<2> '2312' =~ /(\d\d|\d):(\d\d)|(\d\d)(\d\d)/; print "3->$3 4->$4\n"
3->23 4->12
```

Parece inconveniente tener los resultados en variables distintas. El constructo `(?| ...)` hace que los paréntesis se enumeren relativos a las alternativas:

```
  DB<3> '2312' =~ /(?|(\d\d|\d):(\d\d)|(\d\d)(\d\d))/; print "1->$1 2->$2\n"
1->23 2->12

  DB<4> '23:12' =~ /(?|(\d\d|\d):(\d\d)|(\d\d)(\d\d))/; print "1->$1 2->$2\n"
1->23 2->12
```

Ahora en ambos casos `$1` y `$2` contienen las horas y minutos.

3.2.8. Ejecución de Código dentro de una Expresión Regular

Es posible introducir código Perl dentro de una expresión regular. Para ello se usa la notación `(?{code})`.

El siguiente texto esta tomado de la sección 'A-bit-of-magic:-executing-Perl-code-in-a-regular-expression' en `perlretut`:

> *Normally, regexps are a part of Perl expressions. Code evaluation expressions turn that around by allowing arbitrary Perl code to be a part of a regexp. A code evaluation expression is denoted (?code), with code a string of Perl statements.*
>
> *Be warned that this feature is considered experimental, and may be changed without notice.*
>
> *Code expressions are zero-width assertions, and the value they return depends on their environment.*
>
> *There are two possibilities: either the code expression is used as a conditional in a conditional expression* `?(condition)...)`*, or it is not.*
>
> - *If the code expression is a conditional, the code is evaluated and the result (i.e., the result of the last statement) is used to determine truth or falsehood.*
> - *If the code expression is not used as a conditional, the assertion always evaluates true and the result is put into the special variable* `$^R` *. The variable* `$^R` *can then be used in code expressions later in the regexp*

Resultado de la última ejecución

Las expresiones de código son *zero-width assertions*: no consumen entrada. El resultado de la ejecución se salva en la variable especial `$^R`.

Veamos un ejemplo:

```
pl@nereida:~/Lperltesting$ perl5.10.1 -wde 0
main::(-e:1):   0
  DB<1> $x = "abcdef"
  DB<2> $x =~ /abc(?{ "Hi mom\n" })def(?{ print $^R })$/
Hi mom
  DB<3> $x =~ /abc(?{ print "Hi mom\n"; 4 })def(?{ print "$^R\n" })/
Hi mom
4
  DB<4> $x =~ /abc(?{ print "Hi mom\n"; 4 })ddd(?{ print "$^R\n" })/ # does not match
  DB<5>
```

En el último ejemplo (línea `DB<4>`) ninguno de los `print` se ejecuta dado que no hay matching.

El Código empotrado no es interpolado
Tomado de la sección 'Extended-Patterns' en `perlre`:

> *This zero-width assertion evaluates any embedded Perl code. It always succeeds, and its code is not interpolated. Currently, the rules to determine where the code ends are somewhat convoluted.*

Contenido del último paréntesis y la variable por defecto en acciones empotradas
Tomado de la sección 'Extended-Patterns' en `perlre`:

> *... can be used with the special variable* `$^N` *to capture the results of submatches in variables without having to keep track of the number of nested parentheses. For example:*

```
pl@nereida:~/Lperltesting$ perl5.10.1 -wdE 0
main::(-e:1):   0
  DB<1> $x = "The brown fox jumps over the lazy dog"
  DB<2> x $x =~ /the (\S+)(?{ $color = $^N }) (\S+)(?{ $animal = $^N })/i
0  'brown'
1  'fox'
  DB<3> p "color=$color animal=$animal\n"
color=brown animal=fox
  DB<4> $x =~ /the (\S+)(?{ print (substr($_,0,pos($_)))."\n" }) (\S+)/i
The brown
```

> *Inside the* `(?{...})` *block,* `$_` *refers to the string the regular expression is matching against. You can also use* `pos()` *to know what is the current position of matching within this string.*

Los cuantificadores y el código empotrado
Si se usa un cuantificador sobre un código empotrado, actúa como un bucle:

```
pl@nereida:~/Lperltesting$ perl5.10.1 -wde 0
main::(-e:1):   0
  DB<1> $x = "aaaa"
  DB<2>  $x =~ /(a(?{ $c++ }))*/
  DB<3> p $c
4
  DB<4> $y = "abcd"
  DB<5> $y =~ /(?:(.)(?{ print "-$1-\n" }))*/
-a-
-b-
-c-
-d-
```

Ámbito
Tomado (y modificado el ejemplo) de la sección 'Extended-Patterns' en `perlre`:

> *... The code is properly scoped in the following sense: If the assertion is backtracked (compare la sección 'Backtracking' en* `perlre`*), all changes introduced after localization are undone, so that*

```
pl@nereida:~/Lperltesting$ cat embededcodescope.pl
  use strict;

  our ($cnt, $res);
```

```
 sub echo {
local our $pre = substr($_,0,pos($_));
local our $post = (pos($_) < length)? (substr($_,1+pos($_))) : '';

print("$pre(count = $cnt)$post\n");
 }

 $_ = 'a' x 8;
 m<
(?{ $cnt = 0 }) # Initialize $cnt.
(
  a
  (?{
    local $cnt = $cnt + 1; # Update $cnt, backtracking-safe.
    echo();
  })
)*
aaaa
(?{ $res = $cnt }) # On success copy to non-localized
# location.
 >x;

 print "FINAL RESULT: cnt = $cnt res =$res\n";
```

will set `$res = 4` *. Note that after the match,* `$cnt` *returns to the globally introduced value, because the scopes that restrict local operators are unwound.*

```
pl@nereida:~/Lperltesting$ perl5.8.8 -w embededcodescope.pl
a(count = 1)aaaaaa
aa(count = 2)aaaaa
aaa(count = 3)aaaa
aaaa(count = 4)aaa
aaaaa(count = 5)aa
aaaaaa(count = 6)a
aaaaaaa(count = 7)
aaaaaaaa(count = 8)
FINAL RESULT: cnt = 0 res =4
```

Caveats

- *Due to an unfortunate implementation issue, the Perl code contained in these blocks is treated as a compile time closure that can have seemingly bizarre consequences when used with lexically scoped variables inside of subroutines or loops. There are various workarounds for this, including simply using global variables instead. If you are using this construct and strange results occur then check for the use of lexically scoped variables.*
- *For reasons of security, this construct is forbidden if the regular expression involves run-time interpolation of variables, unless the perilous* `use re 'eval'` *pragma has been used (see* `re`*), or the variables contain results of* `qr//` *operator (see* `"qr/STRING/imosx"` *in* `perlop`*).*

  This restriction is due to the wide-spread and remarkably convenient custom of using run-time determined strings as patterns. For example:

1. `$re = <>;`
2. `chomp $re;`
3. `$string =~ /$re/;`

Before Perl knew how to execute interpolated code within a pattern, this operation was completely safe from a security point of view, although it could raise an exception from an illegal pattern. If you turn on the `use re 'eval'` *, though, it is no longer secure, so you should only do so if you are also using taint checking. Better yet, use the carefully constrained evaluation within a* `Safe` *compartment. See perlsec for details about both these mechanisms. (Véase la sección 'Taint-mode' en* `perlsec`*)*

- *Because Perl's regex engine is currently* not re-entrant, *interpolated code may not invoke the regex engine either directly with* `m//` *or* `s///`*, or indirectly with functions such as* `split`*.*

Depurando con código empotrado Colisiones en los Nombres de las Subexpresiones Regulares

Las acciones empotradas pueden utilizarse como mecanismo de depuración y de descubrimiento del comportamiento de nuestras expresiones regulares.

En el siguiente programa se produce una colisión entre los nombres `<i>` y `<j>` de los patrones que ocurren en el patrón `<expr>` y en el patrón principal:

```
pl@nereida:~/Lperltesting$ cat -n clashofnamedofssets.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1
use v5.10;

my $input;

local $" = ", ";

my $parser = qr{
    ^ (?<i> (?&expr)) (?<j> (?&expr)) \z
      (?{
           say "main \$+ hash:";
           say " ($_ => $+{$_}) " for sort keys %+;
       })

    (?(DEFINE)
        (?<expr>
            (?<i> . )
            (?<j> . )
              (?{
                  say "expr \$+ hash:";
                  say " ($_ => $+{$_}) " for sort keys %+;
              })
        )
    )
}x;

$input = <>;
chomp($input);
if ($input =~ $parser) {
  say "matches: ($&)";
}
```

La colisión hace que la salida sea esta:

```
pl@nereida:~/Lperltesting$ ./clashofnamedofssets.pl
abab
expr $+ hash:
 (i => a)
 (j => b)
expr $+ hash:
 (i => ab)
 (j => b)
main $+ hash:
 (i => ab)
 (j => ab)
matches: (abab)
```

Si se evitan las colisiones, se evita la pérdida de información:

```
pl@nereida:~/Lperltesting$ cat -n namedoffsets.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1
use v5.10;

my $input;

local $" = ", ";

my $parser = qr{
    ^ (?<i> (?&expr)) (?<j> (?&expr)) \z
      (?{
           say "main \$+ hash:";
           say " ($_ => $+{$_}) " for sort keys %+;
       })

    (?(DEFINE)
        (?<expr>
            (?<i_e> . )
            (?<j_e> . )
              (?{
                  say "expr \$+ hash:";
                  say " ($_ => $+{$_}) " for sort keys %+;
              })
        )
    )
}x;

$input = <>;
chomp($input);
if ($input =~ $parser) {
  say "matches: ($&)";
}
```

que al ejecutarse produce:

```
pl@nereida:~/Lperltesting$ ./namedoffsets.pl
abab
expr $+ hash:
```

```
 (i_e => a)
 (j_e => b)
expr $+ hash:
 (i => ab)
 (i_e => a)
 (j_e => b)
main $+ hash:
 (i => ab)
 (j => ab)
matches: (abab)
```

3.2.9. Expresiones Regulares en tiempo de matching

Los paréntesis especiales:

```
(??{ Código Perl })
```

hacen que el `Código Perl` sea evaluado durante el tiempo de matching. El resultado de la evaluación se trata como una expresión regular. El match continuará intentando casar con la expresión regular retornada.

Paréntesis con memoria dentro de una *pattern code expression*

Los paréntesis en la expresión regular retornada no cuentan en el patrón exterior. Véase el siguiente ejemplo:

```
pl@nereida:~/Lperltesting$ cat -n postponedregexp.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1 -w
use v5.10;
use strict;

my $r = qr{(?x)                # ignore spaces
            ([ab])             # save 'a' or 'b' in \$1
            (??{ "($^N)"x3 })  # 3 more of the same as in \$1
          };
say "<$&> lastpar = $#-" if 'bbbb' =~ $r;
say "<$&> lastpar = $#-" if 'aaaa' =~ $r;
say "<abab> didn't match" unless 'abab' =~ $r;
say "<aaab> didn't match" unless 'aaab' =~ $r;
```

Como se ve, hemos accedido desde el código interior al último paréntesis usando `$^N`. Sigue una ejecución:

```
pl@nereida:~/Lperltesting$ ./postponedregexp.pl
<bbbb> lastpar = 1
<aaaa> lastpar = 1
<abab> didn't match
<aaab> didn't match
```

Ejemplo: Secuencias de dígitos de longitud especificada por el primer dígito

Consideremos el problema de escribir una expresión regular que reconoce secuencias no vacías de dígitos tales que la longitud de la secuencia restante viene determinada por el primer dígito. Esta es una solución:

```
pl@nereida:~/Lperltesting$ cat -n intints.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1 -w
```

```
use v5.10;
use strict;

my $r = qr{(?x)                # ignore spaces
           (\d)                # a digit
           ( (??{
               "\\d{$^N}"      # as many as the former
             })                # digit says
           )
          };
say "<$&> <$1> <$2>" if '3428' =~ $r;
say "<$&> <$1> <$2>" if '228' =~ $r;
say "<$&> <$1> <$2>" if '14' =~ $r;
say "24 does not match" unless '24' =~ $r;
say "4324 does not match" unless '4324' =~ $r;
```

Cuando se ejecuta se obtiene:

```
pl@nereida:~/Lperltesting$ ./intints.pl
<3428> <3> <428>
<228> <2> <28>
<14> <1> <4>
24 does not match
4324 does not match
```

Ejemplo: Secuencias de dígitos no repetidos

Otro ejemplo: queremos escribir una expresión regular que reconozca secuencias de `$n` dígitos en las que no todos los dígitos se repiten. Donde quizá `$n` es capturado de un paréntesis anterior en la expresión regular. Para simplificar la ilustración de la técnica supongamos que `$n = 7`:

```
pl@nereida:~$ perl5.10.1 -wdE 0
main::(-e:1):   0
  DB<1> x join '', map { "(?!".$_."{7})" } 0..9
0  '(?!0{7})(?!1{7})(?!2{7})(?!3{7})(?!4{7})(?!5{7})(?!6{7})(?!7{7})(?!8{7})(?!9{7})'
  DB<2> x '7777777' =~ /(??{join '', map { "(?!".$_."{7})" } 0..9})(\d{7})/
  empty array
  DB<3> x '7777778' =~ /(??{join '', map { "(?!".$_."{7})" } 0..9})(\d{7})/
0  7777778
  DB<4> x '4444444' =~ /(??{join '', map { "(?!".$_."{7})" } 0..9})(\d{7})/
  empty array
  DB<5> x '4422444' =~ /(??{join '', map { "(?!".$_."{7})" } 0..9})(\d{7})/
0  4422444
```

Palíndromos con independencia del acento

Se trata en este ejercicio de generalizar la expresión regular introducida en la sección 3.2.5 para reconocer los palabra-palíndromos.

Se trata de encontrar una regexp que acepte que la lectura derecha e inversa de una frase en Español pueda diferir en la acentuación (como es el caso del clásico palíndromo *dábale arroz a la zorra el abad*). Una solución trivial es preprocesar la cadena eliminando los acentos. Supondremos sin embargo que se quiere trabajar sobre la cadena original. He aquí una solucion:

```
pl@nereida:~/Lperltesting$ cat actionspanishpalin.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1 -w -CIOEioA
use v5.10;
```

```
use strict;
use utf8;
use re 'eval';
use Switch;

sub f {
  my $char = shift;

  switch($char) {
    case [ qw{a á} ] { return '[aá]' }
    case [ qw{e é} ] { return '[eé]' }
    case [ qw{i í} ] { return '[ií]' }
    case [ qw{o ó} ] { return '[oó]' }
    case [ qw{u ú} ] { return '[uú]' }
    else             { return $char  };
  }
}

my $regexp = qr/^(\W* (?:
                            (\w) (?-2)(??{ f($^N) })
                          | \w?
                      ) \W*
                  )
                $
               /ix;

my $input = <>; # Try: 'dábale arroz a la zorra el abad';
chomp($input);
if ($input =~ $regexp) {
  say "$input is a palindrome";
}
else {
  say "$input does not match";
}
```

Sigue un ejemplo de ejecución:

```
pl@nereida:~/Lperltesting$ ./actionspanishpalin.pl
dábale arroz a la zorra el abad
dábale arroz a la zorra el abad is a palindrome
pl@nereida:~/Lperltesting$ ./actionspanishpalin.pl
éoíúaáuioé
éoíúaáuioé is a palindrome
pl@nereida:~/Lperltesting$ ./actionspanishpalin.pl
dáed
dáed does not match
```

Postponiendo para conseguir recursividad

Véase el nodo Complex regex for maths formulas para la formulación del problema:

> *Hiya monks,*
>
> *Im having trouble getting my head around a regular expression to match sequences. I need to catch all exceptions where a mathematical expression is illegal...*

There must be either a letter or a digit either side of an operator parenthesis must open and close next to letters or digits, not next to operators, and do not have to exist variables must not be more than one letter Nothing other than `a-z,A-Z,0-9,+,-,*,/,(,)` *can be used*

Can anyone offer a hand on how best to tackle this problem?

many thanks

La respuesta dada por ikegami usa `(?{{ ... })` para conseguir una conducta recursiva en versiones de perl anteriores a la 5.10:

```
pl@nereida:~/Lperltesting$ cat -n complexformula.pl
#!/usr/bin/perl
use strict;
use warnings;

sub is_valid_expr {
   use re 'eval'; # to allow Eval-group at runtime

   local our ($skip, $term, $expr);
   $skip = qr! \s* !x;
   $term = qr! $skip [a-zA-Z]+              # A term is an identifier
             | $skip [1-9][0-9]*            # or a number
             | $skip \( (??{ $expr }) $skip # or an expression
                     \)
             !x;
   $expr = qr! $term                         # A expr is a term
               (?: $skip [-+*/] $term )*     # or a term + a term ...
             !x;

   return $_[0] =~ / ^ $expr $skip \z /x;
}

print(is_valid_expr($_) ? "$_ is valid\n" : "$_ is not valid\n") foreach (
 '(a + 3)',
 '(3 * 4)+(b + x)',
 '(5 - a)*z',
 '3 + 2',

 '!3 + 2',
 '3 + 2!',

 '3 a',
 '3 3',
 '3 * * 3',

 '2 - 3 * 4',
 '2 - 3 + 4',
);
```

Sigue el resultado de la ejecución:

```
pl@nereida:~/Lperltesting$ perl complexformula.pl
(a + 3) is valid
(3 * 4)+(b + x) is valid
```

```
(5 - a)*z is valid
3 + 2 is valid
!3 + 2 is not valid
3 + 2! is not valid
3 a is not valid
3 3 is not valid
3 * * 3 is not valid
2 - 3 * 4 is valid
2 - 3 + 4 is valid
```

Caveats

Estos son algunos puntos a tener en cuenta cuando se usan patrones postpuestos. Véase la entrada `(??{ code })` en la sección 'Extended-Patterns' en `perlre`:

> *WARNING: This extended regular expression feature is considered experimental, and may be changed without notice. Code executed that has side effects may not perform identically from version to version due to the effect of future optimisations in the regex engine.*
>
> *This is a* postponed regular subexpression. *The code is evaluated at run time, at the moment this subexpression may match. The result of evaluation is considered as a regular expression and matched as if it were inserted instead of this construct.*
>
> *The code is not interpolated.*
>
> *As before, the rules to determine where the code ends are currently somewhat convoluted.*
>
> *Because perl's regex engine is not currently re-entrant, delayed code may not invoke the regex engine either directly with* `m//` *or* `s///`*), or indirectly with functions such as* `split`*.*
>
> *Recursing deeper than 50 times without consuming any input string will result in a fatal error. The maximum depth is compiled into perl, so changing it requires a custom build.*

3.2.10. Expresiones Condicionales

Citando a `perlre`:

> *A conditional expression is a form of if-then-else statement that allows one to choose which patterns are to be matched, based on some condition.*
>
> *There are two types of conditional expression:* `(?(condition)yes-regexp)` *and* `(?(condition)yes-rege`
>
> `(?(condition)yes-regexp)` *is like an* `if () {}` *statement in Perl. If the condition is true, the yes-regexp will be matched. If the condition is false, the yes-regexp will be skipped and Perl will move onto the next regexp element.*
>
> *The second form is like an* `if () {} else {}` *statement in Perl. If the condition is true, the yes-regexp will be matched, otherwise the no-regexp will be matched.*
>
> *The condition can have several forms.*
>
> - *The first form is simply an integer in parentheses (integer). It is true if the corresponding backreference* `\integer` *matched earlier in the regexp. The same thing can be done with a name associated with a capture buffer, written as* `(<name>)` *or* `('name')`*.*
> - *The second form is a bare zero width assertion* `(?...)`*, either a lookahead, a lookbehind, or a code assertion.*
> - *The third set of forms provides tests that return true if the expression is executed within a recursion* `(R)` *or is being called from some capturing group, referenced either by number* `(R1`*, or by name* `(R&name)`*.*

Condiciones: número de paréntesis

Una expresión condicional puede adoptar diversas formas. La mas simple es un entero en paréntesis. Es cierta si la correspondiente referencia `\integer` casó (también se puede usar un nombre si se trata de un paréntesis con nombre).

En la expresión regular `/^(.)(..)?(?(2)a|b)/` si el segundo paréntesis casa, la cadena debe ir seguida de una `a`, si no casa deberá ir seguida de una `b`:

```
  DB<1> x 'hola' =~ /^(.)(..)?(?(2)a|b)/
0  'h'
1  'ol'
  DB<2> x 'ha' =~ /^(.)(..)?(?(2)a|b)/
  empty array
  DB<3> x 'hb' =~ /^(.)(..)?(?(2)a|b)/
0  'h'
1  undef
```

Ejemplo: cadenas de la forma ***una*-otra-otra-*una***

La siguiente búsqueda casa con patrones de la forma `$x$x` o `$x$y$y$x`:

```
pl@nereida:~/Lperltesting$ perl5.10.1 -wde 0
main::(-e:1):   0
  DB<1> x 'aa' =~ m{^(\w+)(\w+)?(?(2)\2\1|\1)$}
0  'a'
1  undef
  DB<2> x 'abba' =~ m{^(\w+)(\w+)?(?(2)\2\1|\1)$}
0  'a'
1  'b'
  DB<3> x 'abbc' =~ m{^(\w+)(\w+)?(?(2)\2\1|\1)$}
  empty array
  DB<4> x 'juanpedropedrojuan' =~ m{^(\w+)(\w+)?(?(2)\2\1|\1)$}
0  'juan'
1  'pedro'
```

Condiciones: Código

Una expresión condicional también puede ser un código:

```
  DB<1> $a = 0; print "$&" if 'hola' =~ m{(?(?{$a})hola|adios)} # No hay matching

  DB<2> $a = 1; print "$&" if 'hola' =~ m{(?(?{$a})hola|adios)}
hola
```

Ejemplo: Cadenas con posible paréntesis inicial (no anidados)

La siguiente expresión regular utiliza un condicional para forzar a que si una cadena comienza por un paréntesis abrir termina con un paréntesis cerrar. Si la cadena no comienza por paréntesis abrir no debe existir un paréntesis final de cierre:

```
pl@nereida:~/Lperltesting$ cat -n conditionalregexp.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1 -w
use v5.10;
use strict;

my $r = qr{(?x)                # ignore spaces
            ^
            ( \( )?            # may be it comes an open par
```

```
               [^()]+            # no parenthesis
               (?(1)             # did we sart with par?
                 \)              # if yes then close par
               )
               $
             };
say "<$&>" if '(abcd)' =~ $r;
say "<$&>" if 'abc' =~ $r;
say "<(abc> does not match" unless '(abc' =~ $r;
say "<abc)> does not match" unless 'abc)' =~ $r;
```

Al ejecutar este programa se obtiene:

```
pl@nereida:~/Lperltesting$ ./conditionalregexp.pl
<(abcd)>
<abc>
<(abc> does not match
<abc)> does not match
```

Expresiones Condicionales con `(R)`

El siguiente ejemplo muestra el uso de la condición `(R)`, la cual comprueba si la expresión ha sido evaluada dentro de una recursión:

```
pl@nereida:~/Lperltesting$ perl5.10.1 -wdE 0
main::(-e:1):   0
  DB<1> x 'bbaaaabb' =~ /(b(?(R)a+|(?0))b)/
0  'bbaaaabb'
  DB<2> x 'bb' =~ /(b(?(R)a+|(?0))b)/
  empty array
  DB<3> x 'bab' =~ /(b(?(R)a+|(?0))b)/
  empty array
  DB<4> x 'bbabb' =~ /(b(?(R)a+|(?0))b)/
0  'bbabb'
```

La sub-expresión regular `(?(R)a+|(?0))` dice: si esta siendo evaluada recursivamente admite `a+` si no, evalúa la regexp completa recursivamente.

Ejemplo: Palíndromos con Equivalencia de Acentos Españoles

Se trata en este ejercicio de generalizar la expresión regular introducida en la sección 3.2.5 para reconocer los palabra-palíndromos[7]. Se trata de encontrar una regexp que acepte que la lectura derecha e inversa de una frase en Español pueda diferir en la acentuación (como es el caso del clásico palíndromo *dábale arroz a la zorra el abad*). Una solución trivial es preprocesar la cadena eliminando los acentos. Supondremos sin embargo que se quiere trabajar sobre la cadena original. He aquí una solucion parcial (por consideraciones de legibilidad sólo se consideran las vocales `a` y `o`:

```
pl@nereida:~/Lperltesting$ cat spanishpalin.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1 -w -CIOEioA
use v5.10;
use strict;
use utf8;

my $regexp = qr/^(?<pal>\W* (?:
                            (?<L>(?<a>[áa])|(?<e>[ée])|\w) # letter
```

[7] No sé si existe el término. Significa que la lectura directa y la inversa pueden diferir en los signos de puntuación

```
                              (?&pal)                        # nested palindrome
                              (?(<a>)[áa]                    # if is an "a" group
                                    |(?:((?<e>)[ée]          # if is an "e" group
                                              |\g{L}         # exact match
                                        )                    # end if [ée]
                                     )                       # end group
                              )                              # end if [áa]
                            | \w?                            # non rec. case
                        ) \W*                                # punctuation symbols
                    )
                  $
                 /ix;

my $input = <>; # Try: 'dábale arroz a la zorra el abad';
chomp($input);
if ($input =~ $regexp) {
  say "$input is a palindrome";
}
else {
  say "$input does not match";
}
```

Ejecución:

```
pl@nereida:~/Lperltesting$ ./spanishpalin.pl
dábale arroz a la zorra el abad
dábale arroz a la zorra el abad is a palindrome
pl@nereida:~/Lperltesting$ ./spanishpalin.pl
óuuo
óuuo does not match
pl@nereida:~/Lperltesting$ ./spanishpalin.pl
éaáe
éaáe is a palindrome
```

Hemos usado la opción `-CIOEioA` para asegurarnos que los ficheros de entrada/saldia y error y la línea de comandos estan en modo UTF-8. (Véase la sección **??**)

Esto es lo que dice la documentación de `perlrun` al respecto:

> *The* `-C` *flag controls some of the Perl Unicode features.*
>
> *As of 5.8.1, the* `-C` *can be followed either by a number or a list of option letters. The letters, their numeric values, and effects are as follows; listing the letters is equal to summing the numbers.*
>
> ```
> I 1 STDIN is assumed to be in UTF-8
> O 2 STDOUT will be in UTF-8
> E 4 STDERR will be in UTF-8
> S 7 I + O + E
> i 8 UTF-8 is the default PerlIO layer for input streams
> o 16 UTF-8 is the default PerlIO layer for output streams
> D 24 i + o
> A 32 the @ARGV elements are expected to be strings encoded
> in UTF-8
> L 64 normally the "IOEioA" are unconditional,
> the L makes them conditional on the locale environment
> ```

```
variables (the LC_ALL, LC_TYPE, and LANG, in the order
of decreasing precedence) -- if the variables indicate
UTF-8, then the selected "IOEioA" are in effect
a 256 Set ${^UTF8CACHE} to -1, to run the UTF-8 caching code in
debugging mode.
```

> *For example,* `-COE` *and* `-C6` *will both turn on UTF-8-ness on both* `STDOUT` *and* `STDERR`*. Repeating letters is just redundant, not cumulative nor toggling.*
>
> *The io options mean that any subsequent* `open()` *(or similar I/O operations) will have the* `:utf8` *PerlIO layer implicitly applied to them, in other words,* `UTF-8` *is expected from any input stream, and* `UTF-8` *is produced to any output stream. This is just the default, with explicit layers in* `open()` *and with* `binmode()` *one can manipulate streams as usual.*
>
> `-C` *on its own (not followed by any number or option list), or the empty string* `""` *for the* `PERL_UNICODE` *environment variable, has the same effect as* `-CSDL` *. In other words, the standard I/O handles and the default*`open()` *layer are UTF-8-fied but only if the locale environment variables indicate a UTF-8 locale. This behaviour follows the implicit (and problematic) UTF-8 behaviour of Perl 5.8.0.*
>
> *You can use* `-C0` *(or* `0` *for* `PERL_UNICODE` *) to explicitly disable all the above Unicode features.*

El pragma `use utf8` hace que se utilice una semántica de carácteres (por ejemplo, la regexp `/./` casará con un carácter unicode), el pragma `use bytes` cambia de semántica de caracteres a semántica de bytes (la regexp `.` casará con un byte).

3.2.11. Verbos que controlan el retroceso

El verbo de control `(*FAIL)`

Tomado de la sección 'Backtracking-control-verbs' en `perlretut`:

> *The control verb* `(*FAIL)` *may be abbreviated as* `(*F)`*. If this is inserted in a regexp it will cause to fail, just like at some mismatch between the pattern and the string. Processing of the regexp continues like after any "normal"failure, so that the next position in the string or another alternative will be tried. As failing to match doesn't preserve capture buffers or produce results, it may be necessary to use this in combination with embedded code.*

```
pl@nereida:~/Lperltesting$ cat -n vowelcount.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1  -w
use strict;

my $input = shift() || <STDIN>;
my %count = ();
$input =~ /([aeiou])(?{ $count{$1}++; })(*FAIL)/i;
printf("'%s' => %3d\n", $_, $count{$_})  for (sort keys %count);
```

Al ejecutarse con entrada `supercalifragilistico` produce la salida:

```
pl@nereida:~/Lperltesting$ ./vowelcount.pl
supercalifragilistico
'a' =>   2
'e' =>   1
'i' =>   4
'o' =>   1
'u' =>   1
```

Ejercicio 3.2.5. *¿Que queda en* `$1` *depués de ejecutado el matching* `$input =~ /([aeiou])(?{ $count{$1}++;`

Véase también:

- El nodo en PerlMonks *The Oldest Plays the Piano*
- Véase el ejercicio *Las tres hijas* en la sección 3.4.4

El verbo de control `(*ACCEPT)`

Tomado de `perlretut`:

> *This pattern matches nothing and causes the end of successful matching at the point at which the* `(*ACCEPT)` *pattern was encountered, regardless of whether there is actually more to match in the string. When inside of a nested pattern, such as recursion, or in a subpattern dynamically generated via* `(??{})`*, only the innermost pattern is ended immediately.*
>
> *If the* `(*ACCEPT)` *is inside of capturing buffers then the buffers are marked as ended at the point at which the* `(*ACCEPT)` *was encountered. For instance:*

```
  DB<1> x 'AB' =~ /(A (A|B(*ACCEPT)|C) D)(E)/x
0  'AB'
1  'B'
2  undef
  DB<2> x 'ACDE'  =~ /(A (A|B(*ACCEPT)|C) D)(E)/x
0  'ACD'
1  'C'
2  'E'
```

El verbo `SKIP`

> *This zero-width pattern prunes the backtracking tree at the current point when backtracked into on failure. Consider the pattern* `A (*SKIP) B`*, where* `A` *and* `B` *are complex patterns. Until the* `(*SKIP)` *verb is reached,* `A` *may backtrack as necessary to match. Once it is reached, matching continues in* `B`*, which may also backtrack as necessary; however, should* `B` *not match, then no further backtracking will take place, and the pattern will fail outright at the current starting position.*
>
> *It also signifies that whatever text that was matched leading up to the* `(*SKIP)` *pattern being executed cannot be part of any match of this pattern. This effectively means that the regex engine* `skips` *forward to this position on failure and tries to match again, (assuming that there is sufficient room to match).*
>
> *The name of the* `(*SKIP:NAME)` *pattern has special significance. If a* `(*MARK:NAME)` *was encountered while matching, then it is that position which is used as the "skip point". If no* `(*MARK)` *of that name was encountered, then the* `(*SKIP)` *operator has no effect. When used without a name the "skip pointïs where the match point was when executing the* `(*SKIP)` *pattern.*

Ejemplo:

```
pl@nereida:~/Lperltesting$ cat -n SKIP.pl
#!/soft/perl5lib/bin/perl5.10.1 -w
use strict;
use v5.10;

say "NO SKIP: /a+b?(*FAIL)/";
our $count = 0;
'aaab' =~ /a+b?(?{print "$&\n"; $count++})(*FAIL)/;
say "Count=$count\n";

```

```
say "WITH SKIP: a+b?(*SKIP)(*FAIL)/";
$count = 0;
’aaab’ =~ /a+b?(*SKIP)(?{print "$&\n"; $count++})(*FAIL)/;
say "WITH SKIP: Count=$count\n";

say "WITH SKIP /a+(*SKIP)b?(*FAIL)/:";
$count = 0;
’aaab’ =~ /a+(*SKIP)b?(?{print "$&\n"; $count++})(*FAIL)/;
say "Count=$count\n";

say "WITH SKIP /(*SKIP)a+b?(*FAIL): ";
$count = 0;
’aaab’ =~ /(*SKIP)a+b?(?{print "$&\n"; $count++})(*FAIL)/;
say "Count=$count\n";
```

Ejecución:

```
pl@nereida:~/Lperltesting$ perl5.10.1 SKIP.pl
NO SKIP: /a+b?(*FAIL)/
aaab
aaa
aa
a
aab
aa
a
ab
a
Count=9

WITH SKIP: a+b?(*SKIP)(*FAIL)/
aaab
WITH SKIP: Count=1

WITH SKIP /a+(*SKIP)b?(*FAIL)/:
aaab
aaa
Count=2

WITH SKIP /(*SKIP)a+b?(*FAIL):
aaab
aaa
aa
a
aab
aa
a
ab
a
Count=9
```

Marcas

Tomado de la sección ’Backtracking-control-verbs’ en `perlretut`:

`(*MARK:NAME) (*:NAME)`

> *This zero-width pattern can be used to mark the point reached in a string when a certain part of the pattern has been successfully matched. This mark may be given a name. A later* `(*SKIP)` *pattern will then skip forward to that point if backtracked into on failure. Any number of* `(*MARK)` *patterns are allowed, and the* `NAME` *portion is optional and may be duplicated.*
>
> *In addition to interacting with the* `(*SKIP)` *pattern,* `(*MARK:NAME)` *can be used to* `label` *a pattern branch, so that after matching, the program can determine which branches of the pattern were involved in the match.*
>
> *When a match is successful, the* `$REGMARK` *variable will be set to the name of the most recently executed* `(*MARK:NAME)` *that was involved in the match.*
>
> *This can be used to determine which branch of a pattern was matched without using a separate capture buffer for each branch, which in turn can result in a performance improvement.*
>
> *When a match has failed, and unless another verb has been involved in failing the match and has provided its own name to use, the* `$REGERROR` *variable will be set to the name of the most recently executed* `(*MARK:NAME)`*.*

```
pl@nereida:~/Lperltesting$ cat -n mark.pl
use v5.10;
use strict;

our $REGMARK;

$_ = shift;
say $REGMARK if /(?:x(*MARK:mx)|y(*MARK:my)|z(*MARK:mz))/;
say $REGMARK if /(?:x(*:xx)|y(*:yy)|z(*:zz))/;
```

Cuando se ejecuta produce:

```
pl@nereida:~/Lperltesting$ perl5.10.1 mark.pl y
my
yy
pl@nereida:~/Lperltesting$ perl5.10.1 mark.pl z
mz
zz
```

Poniendo un espacio después de cada signo de puntuación

Se quiere poner un espacio en blanco después de la aparición de cada coma:

```
s/,/, /g;
```

pero se quiere que la sustitución no tenga lugar si la coma esta incrustada entre dos dígitos. Además se pide que si hay ya un espacio después de la coma, no se duplique. Sigue una solución que usa marcas:

```
pl@nereida:~/Lperltesting$ perl5.10.1 -wdE 0
main::(-e:1):   0
  DB<1> $a = 'ab,cd, ef,12,34,efg,56,78,df, ef,'
  DB<2> x ($b = $a) =~ s/\d,\d(*:d)|,(?!\s)/($REGMARK eq 'd')? $& : ', '/ge
0  8
  DB<3> p "<$b>"
<ab, cd, ef, 12,34, efg, 56,78, df, ef, >
```

3.3. Expresiones Regulares en Otros Lenguajes

Vim

- Learn vi/vim in 50 lines and 15 minutes
- VIM Regular Expressions
- Editing features for advanced users
- Vim documentation: pattern
- Vim Regular Expressions Chart

Java

El siguiente ejemplo muestra un programa estilo `grep`: solicita una expresión regular para aplicarla luego a una serie de entradas leídas desde la entrada estandar.

```
casiano@nereida:~/projects/PA/regexp$ cat -n Application.java
/**
 * javac Application.java
 * java Application
 */

import java.io.*;
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class Application {

    public static void main(String[] args){
        String regexp = "";
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        try {
            System.out.print("Enter your regex: ");
            regexp = br.readLine();
        } catch (IOException e) { System.exit(1); };
        while (true) {

            String input = "";
            try {
              System.out.print("Enter input string to search: ");
              input = br.readLine();
            } catch (IOException e) { System.exit(1); };

            Pattern pattern = Pattern.compile(regexp);
            Matcher matcher = pattern.matcher(input);

            boolean found = false;
            while (matcher.find()) {
                System.out.println("I found the text "
                                   + matcher.group()
                                   + " starting at index "
                                   + matcher.start()
                                   + " and ending at index "
```

```
                                      +matcher.end()
                    );
                    found = true;
                }
                if(!found){
                    System.out.println("No match found.");
                }
            }
        }
    }
```

Ejecución:

```
casiano@nereida:~/Ljavatesting$ java Application
Enter your regex: (\d+).(\d+)
Enter input string to search: a4b5d6c7efg
I found the text 4b5 starting at index 1 and ending at index 4
I found the text 6c7 starting at index 5 and ending at index 8
Enter input string to search: abc
No match found.
Enter input string to search:
```

Véase también Java Regular Expressions

bash

Esta es una versión en `bash` del conversor de temperaturas visto en las secciones anteriores:

```
pl@nereida:~/src/bash$ cat -n f2c
#!/bin/bash
echo "Enter a temperature (i.e. 32F, 100C):";
read input;

if [ -z "$(echo $input | grep -i '^[-+]\?[0-9]\+\(\.[0-9]*\)\?\ *[CF]$')" ]
then
  echo "Expecting a temperature, so don't understand \"$input\"." 1>&2;
else
  input=$(echo $input | tr -d ' ');
  InputNum=${input:0:${#input}-1};
  Type=${input: -1}

  if [ $Type = "c" -o $Type = "C" ]
  then
   celsius=$InputNum;
   fahrenheit=$(echo "scale=2; ($celsius * 9/5)+32" | bc -l);
  else
   fahrenheit=$InputNum;
   celsius=$(echo "scale=2; ($fahrenheit -32)*5/9" | bc -l);
  fi

  echo "$celsius C = $fahrenheit F";
fi
```

C

```
pl@nereida:~/src/regexpr$ cat -n pcregrep.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <pcre.h>

char enter_reverse_mode[] = "\33[7m";
char exit_reverse_mode[] = "\33[0m";

int main(int argc, char **argv)
{
  const char *pattern;
  const char *errstr;
  int erroffset;
  pcre *expr;
  char line[512];
  assert(argc == 2); /* XXX fixme */
  pattern = argv[1];
  if (!(expr = pcre_compile(pattern, 0, &errstr, &erroffset, 0))) {
    fprintf(stderr, "%s: %s\n", pattern, errstr);
    return EXIT_FAILURE;
  }
  while (fgets(line, sizeof line, stdin)) {
    size_t len = strcspn(line, "\n");
    int matches[2];
    int offset = 0;
    int flags = 0;
    line[len] = '\0';
    while (0 < pcre_exec(expr, 0, line, len, offset, flags, matches, 2)) {
      printf("%.*s%s%.*s%s",
        matches[0] - offset, line + offset,
        enter_reverse_mode,
        matches[1] - matches[0], line + matches[0],
        exit_reverse_mode);
      offset = matches[1];
      flags |= PCRE_NOTBOL;
    }
    printf("%s\n", line + offset);
  }
  return EXIT_SUCCESS;
}
```

Compilación:

```
pl@nereida:~/src/regexpr$ gcc -lpcre pcregrep.c -o pcregrep
```

Cuando se ejecuta espera un patrón en la línea de comandos y pasa a leer desde la entrada estandar. Las cadenas que casan se muestran resaltadas:

```
pl@nereida:~/src/regexpr$ ./pcregrep '\d+'
435 otro 23
```

<u>435</u> otro <u>23</u>

```
hola
hola
```

Python

```
pl@nereida:~/src/python$ cat -n c2f.py
#!/usr/local/bin/python
import re

temp = raw_input( ’ Introduzca una temperatura (i.e. 32F, 100C): ’ )
pattern = re.compile( "^([-+]?[0-9]+(\.[0-9]*)?)\s*([CF])$", re.IGNORECASE )
mo = pattern.match( temp )

if mo:
  inputNum = float(mo.group( 1 ))
  type = mo.group( 3 )
  celsius = 0.0
  fahrenheit = 0.0
  if ( type == "C" or type == "c" ) :
    celsius = inputNum
    fahrenheit = ( celsius * 9/5 ) + 32
  else :
    fahrenheit = inputNum
    celsius = ( fahrenheit - 32 ) * 5/9
  print " ", ’%.2f’%(celsius), " C = ", ’%.2f’%(fahrenheit), " F\n"
else :
  print " Se experaba una temperatura, no se entiende", temp, "\n"
```

Ruby

```
pl@nereida:~/src/ruby$ cat -n f2c_b
#!/usr/bin/ruby

# Primero leemos una temperatura
class Temperature_calculator
  def initialize temp
  comp = Regexp.new(’^([-+]?\d+(\.\d*)?)\s*([CFcf])$’)
  if temp =~ comp
  begin
    cifra = Float($1)
    @C,@F = ( $3 == "F" or $3 == "f")? [(cifra -32) * 5/9, cifra] : [cifra , cifra * 9/5 +
  end
  else
    raise("Entrada incorrecta")
  end
end

  def show
    puts "Temperatura en Celsius: #{@C}, temperatura en Fahrenheit: #{@F}"
  end
end

temperatura = Temperature_calculator.new(readline.chop)
temperatura.show
```

Javascript

```
<SCRIPT LANGUAGE="JavaScript"><!--
function demoMatchClick() {
  var re = new RegExp(document.demoMatch.regex.value);
  if (document.demoMatch.subject.value.match(re)) {
    alert("Successful match");
  } else {
    alert("No match");
  }
}

function demoShowMatchClick() {
  var re = new RegExp(document.demoMatch.regex.value);
  var m = re.exec(document.demoMatch.subject.value);
  if (m == null) {
    alert("No match");
  } else {
    var s = "Match at position " + m.index + ":\n";
    for (i = 0; i < m.length; i++) {
      s = s + m[i] + "\n";
    }
    alert(s);
  }
}

function demoReplaceClick() {
  var re = new RegExp(document.demoMatch.regex.value, "g");
  document.demoMatch.result.value =
    document.demoMatch.subject.value.replace(re,
      document.demoMatch.replacement.value);
}
// -->
</SCRIPT>

<FORM ID="demoMatch" NAME="demoMatch" METHOD=POST ACTION="javascript:void(0)">
<P>Regexp: <INPUT TYPE=TEXT NAME="regex" VALUE="\bt[a-z]+\b" SIZE=50></P>
<P>Subject string: <INPUT TYPE=TEXT NAME="subject"
   VALUE="This is a test of the JavaScript RegExp object" SIZE=50></P>
<P><INPUT TYPE=SUBMIT VALUE="Test Match" ONCLICK="demoMatchClick()">
<INPUT TYPE=SUBMIT VALUE="Show Match" ONCLICK="demoShowMatchClick()"></P>

<P>Replacement text: <INPUT TYPE=TEXT NAME="replacement" VALUE="replaced" SIZE=50></P>
<P>Result: <INPUT TYPE=TEXT NAME="result"
   VALUE="click the button to see the result" SIZE=50></P>
<P><INPUT TYPE=SUBMIT VALUE="Replace" ONCLICK="demoReplaceClick()"></P>
</FORM>
```

3.4. Casos de Estudio

3.4.1. Secuencias de números de tamaño fijo

El siguiente problema y sus soluciones se describen en el libro de J.E.F. Friedl [2]. Supongamos que tenemos un texto conteniendo códigos que son números de tamaño fijo, digamos seis dígitos, todos pegados, sin separadores entre ellos, como sigue:

012345678901**123334**234567890123**125934**890123345126

El problema es encontrar los códigos que comienzan por 12. En negrita se han resaltado las soluciones. Son soluciones sólo aquellas que, comienzan por 12 en una posición múltiplo de seis. Una solución es:

```
@nums = grep {m/^12/} m/\d{6}/g;
```

que genera una lista con los números y luego selecciona los que comienzan por 12. Otra solución es:

```
@nums = grep { defined } m/(12\d{4})|\d{6}/g;
```

que aprovecha que la expresión regular devolverá una lista vacía cuando el número no empieza por 12:

```
DB<1> $x = '012345678901123334234567890123125934890123345126'
DB<2> x  ($x =~ m/(12\d{4})|\d{6}/g)
0  undef
1  undef
2  123334
3  undef
4  undef
5  125934
6  undef
7  undef
```

Obsérvese que se esta utilizando también que el operador | no es *greedy*.

¿Se puede resolver el problema usando sólamente una expresión regular? Obsérvese que esta solución "casi funciona":

```
 DB<3> x @nums = $x =~ m/(?:\d{6})*?(12\d{4})/g;
0  123334
1  125934
2  123345
```

recoge la secuencia mas corta de grupos de seis dígitos que no casan, seguida de una secuencia que casa. El problema que tiene esta solución es al final, cuando se han casado todas las soluciones, entonces la búsqueda exhaustiva hará que nos muestre soluciones que no comienzan en posiciones múltiplo de seis. Por eso encuentra 123345:

012345678901**123334**234567890123**125934**890**123345**126

Por eso, Friedl propone esta solución:

```
@nums = m/(?:\d{6})*?(12\d{4})(?:(?!12)\d{6})*/g;
```

Se asume que existe al menos un éxito en la entrada inicial. Que es un extraordinario ejemplo de como el uso de paréntesis de agrupamiento simplifica y mejora la legibilidad de la solución. Es fantástico también el uso del operador de predicción negativo.

Solución usando el ancla \ `G`

El ancla `\G` ha sido concebida para su uso con la opción `/g`. Casa con el punto en la cadena en el que terminó el último emparejamiento. Cuando se trata del primer intento o no se está usando `/g`, usar `\G` es lo mismo que usar `\A`.

Mediante el uso de este ancla es posible formular la siguiente solución al problema planteado:

```
pl@nereida:~/Lperltesting$ perl -wde 0
main::(-e:1):   0
DB<1> $_ = '012345678901123334234567890123125934890123345126'
DB<2> x m/\G(?:\d{6})*?(12\d{4})/g
0  123334
1  125934
```

Sustitución

Si lo que se quiere es sustituir las secuencias deseadas es poisble hacerlo con la siguiente expresión regular:

```
casiano@nereida:~/docs/curriculums/CV_MEC$ perl -wde 0
DB<1> x $x = '012345678901123334234567890123125934890123345126'
0  012345678901123334234567890123125934890123345126
DB<2> x  ($y = $x) =~ s/(12\d{4})|\d{6}/$1? "-$1-":$& /ge
0  8
DB<3> p $y
012345678901-123334-234567890123-125934-890123345126
```

3.4.2. Palabras Repetidas

Su jefe le pide una herramienta que compruebe la aparición de duplicaciones consecutivas en un texto texto (como esta esta y la anterior anterior). La solución debe cumplir las siguientes especificaciones:

- Aceptar cualquier número de ficheros. Resaltar las apariciones de duplicaciones. Cada línea del informe debe estar precedida del nombre del fichero.
- Funcionar no sólo cuando la duplicación ocurre en la misma línea.
- Funcionar independientemente del *case* y de los blancos usados en medio de ambas palabras.
- Las palabras en cuestión pueden estar separadas por *tags* `HTML`.

```
#!/usr/bin/perl -w
use strict;
use Term::ANSIScreen qw/:constants/;

my $bold = BOLD();
my $clear = CLEAR();
my $line = 1;

# read paragraph
local $/ = ".\n";
while (my $par = <>) {
  next unless $par =~ s{
        \b                 # start word ...
        ([a-z]+)           # grab word in $1 and \1
        (                  # save the tags and spaces in $2
        (\s|<[^>]+>)+      # spaces or HTML tags
```

```
        )
        (\1\b)            # repeated word in $4
  }!$bold$1$clear$2$bold$4$clear!igx;

  $par =~ s/^/"$ARGV(".$line++."): "/meg;   # insert filename and line number

  print $par;
}
```

3.4.3. Análisis de cadenas con datos separados por comas

Supongamos que tenemos cierto texto en `$text` proveniente de un fichero `CSV` (*Comma Separated Values*). Esto es el fichero contiene líneas con el formato:

```
"earth",1,"moon",9.374
```

Esta línea representa cinco campos. Es razonable querer guardar esta información en un *array*, digamos `@field`, de manera que `$field[0] == 'earth'`, `$field[1] == '1'`, etc. Esto no sólo implica descomponer la cadena en campos sino también quitar las comillas de los campos entrecomillados. La primera solución que se nos ocurre es hacer uso de la función `split`:

```
@fields = split(/,/,$text);
```

Pero esta solución deja las comillas dobles en los campos entrecomillados. Peor aún, los campos entrecomillados pueden contener comas, en cuyo caso la división proporcionada por `split` sería errónea.

```
#!/usr/bin/perl -w
use Text::ParseWords;

sub parse_csv {
  my $text = shift;
  my @fields = (); # initialize @fields to be empty

  while ($text =~
    m/"(([^"\\]|\\.)*)",? # quoted fields
      |
      ([^,]+),?           # $3 = non quoted fields
      |
      ,                   # allows empty fields
    /gx
    )
  {
    push(@fields, defined($1)? $1:$3); # add the just matched field
  }
  push(@fields, undef) if $text =~ m/,$/; #account for an empty last field
  return @fields;
}

$test = '"earth",1,"a1, a2","moon",9.374';
print "string = \'$test\'\n";
print "Using parse_csv\n:";
@fields = parse_csv($test);
foreach $i (@fields) {
  print "$i\n";
```

```
29 }
30
31 print "Using Text::ParseWords\n:";
32 #  @words = &quotewords($delim, $keep, @lines);
33 #The $keep argument is a boolean flag.  If true, then the
34 #tokens are split on the specified delimiter, but all other
35 #characters (quotes, backslashes, etc.) are kept in the
36 #tokens.  If $keep is false then the &*quotewords()
37 #functions remove all quotes and backslashes that are not
38 #themselves backslash-escaped or inside of single quotes
39 #(i.e., &quotewords() tries to interpret these characters
40 #just like the Bourne shell).
41
42 @fields = quotewords(',',0,$test);
43 foreach $i (@fields) {
44   print "$i\n";
45 }
```

Las subrutinas en Perl reciben sus argumentos en el *array* `@_`. Si la lista de argumentos contiene listas, estas son "aplanadas" en una única lista. Si, como es el caso, la subrutina ha sido declarada antes de la llamada, los argumentos pueden escribirse sin paréntesis que les rodeen:

```
@fields = parse_csv $test;
```

Otro modo de llamar una subrutina es usando el prefijo `&`, pero sin proporcionar lista de argumentos.

```
@fields = &parse_csv;
```

En este caso se le pasa a la rutina el valor actual del *array* `@_`.

Los operadores `push` (usado en la línea 17) y `pop` trabajan sobre el final del *array*. De manera análoga los operadores `shift` y `unshift` lo hacen sobre el comienzo. El operador ternario `?` trabaja de manera análoga como lo hace en C.

El código del `push` podría sustituirse por este otro:

```
push(@fields, $+);
```

Puesto que la variable `$+` contiene la cadena que ha casado con el último paréntesis que haya casado en el ultimo "matching".

La segunda parte del código muestra que existe un módulo en Perl, el módulo `Text::Parsewords` que proporciona la rutina `quotewords` que hace la misma función que nuestra subrutina.

Sigue un ejemplo de ejecución:

```
> csv.pl
string = '"earth",1,"a1, a2","moon",9.374'
Using parse_csv
:earth
1
a1, a2
moon
9.374
Using Text::ParseWords
:earth
1
a1, a2
moon
9.374
```

3.4.4. Las Expresiones Regulares como Exploradores de un Árbol de Soluciones

Números Primos

El siguiente programa evalúa si un número es primo o no:

```
pl@nereida:~/Lperltesting$ cat -n isprime.pl
#!/usr/bin/perl -w
use strict;

my $num = shift;
die "Usage: $0 integer\n" unless (defined($num) && $num =~ /^\d+$/);

if (("1" x $num) =~ /^(11+)\1+$/) {
  my $factor = length($1);
  print "$num is $factor x ".$num/$factor."\n";
}
else {
  print "$num is prime\n";
}
```

Siguen varias ejecuciones:

```
pl@nereida:~/Lperltesting$ ./isprime.pl 35.32
Usage: ./isprime.pl integer
pl@nereida:~/Lperltesting$ ./isprime.pl 47
47 is prime
pl@nereida:~/Lperltesting$ ./isprime.pl 137
137 is prime
pl@nereida:~/Lperltesting$ ./isprime.pl 147
147 is 49 x 3
pl@nereida:~/Lperltesting$ ./isprime.pl 137
137 is prime
pl@nereida:~/Lperltesting$ ./isprime.pl 49
49 is 7 x 7
pl@nereida:~/Lperltesting$ ./isprime.pl 47
47 is prime
```

Ecuaciones Diofánticas: Una solución

Según dice la entrada Diophantine_equationen la wikipedia:

> *In mathematics, a Diophantine equation is an indeterminate polynomial equation that allows the variables to be integers only.*

La siguiente sesión con el depurador muestra como se puede resolver una ecuación lineal diofántica con coeficientes positivos usando una expresión regular:

```
  DB<1> # Resolvamos 3x + 2y + 5z = 40
DB<2> x ('a'x40) =~  /^((?:...)+)((?:..)+)((?:.....)+)$/
0  'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
1  'aa'
2  'aaaaa'
DB<3> x map { length }  ('a'x40) =~  /^((?:...)+)((?:..)+)((?:.....)+)$/
0  33
1  2
2  5
```

```
DB<4> @c = (3, 2, 5)
DB<5> x map { length($_) / $c[$i++] }  ('a'x40) =~  /^((?:...)+)((?:..)+)((?:.....)+)$/
0  11
1  1
2  1
DB<6> p 3*11+2*1+5*1
40
```

Ecuaciones Diofánticas: Todas las soluciones

Usando el verbo `(*FAIL)` es posible obtener todas las soluciones:

```
main::(-e:1):   0
DB<1>  sub equ { my @c = @_; print "\t3*$c[0]+2*$c[1]+5*$c[2] = ",3*$c[0]+2*$c[1]+5*$c[2],"\n"
DB<2> sub f { my @c = ((length($1)/3), (length($2)/2), (length($3)/5)); equ(@c); }
DB<3> x ('a'x40) =~  /^((?:...)+)((?:..)+)((?:.....)+)$(?{ f() })(*FAIL)/x
        3*11+2*1+5*1 = 40
        3*9+2*4+5*1 = 40
        3*8+2*3+5*2 = 40
        3*7+2*7+5*1 = 40
        3*7+2*2+5*3 = 40
        3*6+2*6+5*2 = 40
        3*6+2*1+5*4 = 40
        3*5+2*10+5*1 = 40
        3*5+2*5+5*3 = 40
        3*4+2*9+5*2 = 40
        3*4+2*4+5*4 = 40
        3*3+2*13+5*1 = 40
        3*3+2*8+5*3 = 40
        3*3+2*3+5*5 = 40
        3*2+2*12+5*2 = 40
        3*2+2*7+5*4 = 40
        3*2+2*2+5*6 = 40
        3*1+2*16+5*1 = 40
        3*1+2*11+5*3 = 40
        3*1+2*6+5*5 = 40
        3*1+2*1+5*7 = 40
  empty array
DB<4>
```

Ecuaciones Diofánticas: Resolutor general

El siguiente programa recibe en línea de comandos los coeficientes y término inependeinte de una ecuación lineal diofántica con coeficientes positivos y muestra todas las soluciones. El algoritmo primero crea una cadena conteniendo el código Perl que contiene la expresión regular adecuada para pasar luego a evaluarlo:

```
pl@nereida:~/Lperltesting$ cat -n diophantinesolvergen.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1 -w
use v5.10;
use strict;

# Writes a Perl solver for
# a1 x1 + a2 x2 + ... + an xn = b
# a_i and b integers > 0
#
```

```perl
my $b = pop;
my @a = @ARGV;
my $debug = 1;

my $b1 = '1'x$b;
my @a1 = map { '1'x$_ } @a;
my @index = map { 'length($'.$_.")/".$a[$_-1] } 1..(@a);
my $aux = join ",", @index;

my $regexp = '^';
$regexp .= "((?:$_)+)" for @a1;

$regexp .= '$(?{ f() })(*FAIL)';

my $solver = <<"SOLVER";
my \@stack;
sub f {
  my \@s = ($aux);
  push \@stack, [ \@s ];
}

q{$b1} =~ m{$regexp}x;

return \@stack;
SOLVER

print "Solver:\n--------\n$solver\n--------\n" if $debug;

my @stack = eval $solver;

say("@$_") for @stack
```

Sigue un ejemplo de ejecución:

```
pl@nereida:~/Lperltesting$ ./diophantinesolvergen.pl 3 2 5 40
Solver:
--------
my @stack;
sub f {
  my @s = (length($1)/3,length($2)/2,length($3)/5);
  push @stack, [ @s ];
}

q{1111111111111111111111111111111111111111} =~ m{^((?:111)+)((?:11)+)((?:11111)+)$(?{ f() })(*

return @stack;

--------
11 1 1
9 4 1
8 3 2
7 7 1
7 2 3
```

```
6 6 2
6 1 4
5 10 1
5 5 3
4 9 2
4 4 4
3 13 1
3 8 3
3 3 5
2 12 2
2 7 4
2 2 6
1 16 1
1 11 3
1 6 5
1 1 7
```

Las Tres Hijas

En la páginas de Retos Matemáticos de
DIVULGAMAT
puede encontrarse el siguiente problema:

Ejercicio 3.4.1. *Dos matemáticos se vieron en la calle después de muchos años sin coincidir.*

- *¡Hola!, ¿qué tal?, ¿te casaste?, y... ¿cuántos hijos tienes?*
- *Pues tengo tres hijas.*
- *¿y qué años tienen?*
- *¡A ver si lo adivinas!: el producto de las edades de las tres es 36, y su suma es el número del portal que ves enfrente...*
- *¡Me falta un dato!*
- *¡Ah, sí!, ¡la mayor toca el piano!*

¿Qué edad tendrán las tres hijas?

¿Podemos ayudarnos de una expresión regular para resolver el problema? Al ejecutar el siguiente programa:

```
pl@nereida:~/Lperltesting$ cat -n playspiano.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1  -w
use v5.10;
use strict;
use List::Util qw{sum};

local our %u;
sub f {
  my @a = @_;
  @a = sort { $b <=> $a } (length($a[1]), length($a[0])/length($a[1]), 36/length($a[0]) );

  local $" = ", ";
  say "(@a)\t ".sum(@a) unless exists($u{"@a"});
  $u{"@a"} = undef;
}
```

```

say "SOL\t\tNUMBER";
my @a =  ('1'x36) =~
         /^((1+)\2+)(\1+)$
                    (?{ f($1, $2, $3)
                     })
            (*FAIL)
         /x;
```

obtenemos la salida:

```
pl@nereida:~/Lperltesting$ ./playspiano.pl
SOL             NUMBER
(9, 2, 2)        13
(6, 3, 2)        11
(4, 3, 3)        10
(18, 2, 1)       21
(12, 3, 1)       16
(9, 4, 1)        14
(6, 6, 1)        13
```

Explique el funcionamiento del programa. A la vista de la salida ¿Cuáles eran las edades de las hijas?

Mochila 0-1

Para una definición del problema vea la sección El Problema de la Mochila 0-1 en los apuntes de LHP

Ejercicio 3.4.2. *¿Sería capaz de resolver usando expresiones regulares el problema de la mochila 0-1? ¡Si lo logra merece el premio a la solución mas freak que se haya encontrado para dicho problema!*

Véase también

Véase también:

- Véase el nodo en PerlMonks *The Oldest Plays the Piano*
- Solving Algebraic Equations Using Regular Expressions

3.4.5. Número de substituciones realizadas

El operador de substitución devuelve el número de substituciones realizadas, que puede ser mayor que uno si se usa la opción `/g`. En cualquier otro caso retorna el valor falso.

```
#!/usr/bin/perl -w
undef($/);
$paragraph = <STDIN>;
$count = 0;
$count = ($paragraph =~ s/Mister\b/Mr./ig);
print "$paragraph";
print "\n$count\n";
```

El resultado de la ejecución es el siguiente:

```
> numsust.pl
Dear Mister Bean,
Is a pleasure for me and Mister Pluto
```

```
to invite you to the Opening Session
Official dinner that will be chaired by
Mister Goofy.

Yours sincerely
  Mister Mickey Mouse
Dear Mr. Bean,
Is a pleasure for me and Mr. Pluto
to invite you to the Opening Session
Official dinner that will be chaired by
Mr. Goofy.

Yours sincerely
  Mr. Mickey Mouse

4
```

3.4.6. Expandiendo y comprimiendo tabs

Este programa convierte los tabs en el número apropiado de blancos.

```
pl@nereida:~/Lperltesting$ cat -n expandtabs.pl
   1  #!/usr/bin/perl -w
   2  use strict;
   3
   4  my @string = <>;
   5
   6  for (@string) {
   7    while (s/\t+/' ' x (length($&)*8 - length($`)%8)/e) {}
   8    print $_;
   9  }
```

Sigue un ejemplo de ejecución:

```
pl@nereida:~/Lperltesting$ cat -nt tabs.in
     1  01234567012345670123456701234567O
     2  one^Itwo^I^Ithree
     3  four^I^I^I^Ifive
     4  ^I^Itwo
pl@nereida:~/Lperltesting$ ./expandtabs.pl tabs.in | cat -tn
     1  01234567012345670123456701234567O
     2  one     two             three
     3  four                            five
     4                  two
```

Ejercicio 3.4.3. *¿Funciona igual si se cambia el bucle* `while` *por una opción* `/g`*?*

```
pl@nereida:~/Lperltesting$ cat -n ./expandtabs2.pl
   1    #!/usr/bin/perl -w
   2    use strict;
   3
   4    my @string = <>;
   5
   6    for (@string) {
   7      s/\t+/' ' x (length($&)*8 - length($`)%8)/ge;
   8      print $_;
   9    }
```

¿Porqué?

3.4.7. Modificación de Múltiples Ficheros: one liner

Aunque no es la forma de uso habitual, Perl puede ser utilizado en “modo sed” para modificar el texto en múltiples ficheros:

`perl  -e 's/nereida\.deioc\.ull\.es/miranda.deioc.ull.es/gi'  -p  -i.bak  *.html`

Este programa sustituye la palabra original (g)lobalmente e i)gnorando el “case”) en todos los ficheros `*.html` y para cada uno de ellos crea una copia de seguridad `*.html.bak`.

Otro ejemplo: la sustitución que sigue ocurre en todos los ficheros `info.txt` en todos los subdirectorios de los subdirectorios que comiencen por `alu`:

```
perl -e 's/\|hyperpage//gi' -p -i.bak  alu*/*/info.txt
```

Las *opciones de línea* de comandos significan lo siguiente:

-e puede usarse para definir el script en la línea de comandos. Multiples -e te permiten escribir un multi-script. Cuando se usa -e, perl no busca por un fichero de script entre la lista de argumentos.

-p La opción `-p` hace que perl incluya un bucle alrededor de tu “script” al estilo sed:

```
while (<>) {
        ...             # your script goes here
} continue {
        print;
}
```

-n Nótese que las líneas se imprimen automáticamente. Para suprimir la impresión usa la opción `-n`

-i[ext] La opción `-i` Expresa que los ficheros procesados serán modificados. Se renombra el fichero de entrada `file.in` a `file.in.ext`, abriendo el de salida con el mismo nombre del fichero de entrada `file.in`. Se selecciona dicho fichero como de salida por defecto para las sentencias `print`. Si se proporciona una extensión se hace una copia de seguridad. Si no, no se hace copia de seguridad.

En general las opciones pueden ponerse en la primera línea del “script”, donde se indica el intérprete. Asi pues, decir

`perl -p -i.bak -e "s/foo/bar/;"`

es equivalente a usar el “script”:

```
#!/usr/bin/perl -pi.bak
s/foo/bar/;
```

3.5. tr y split

El operador de traducción permite la conversión de unos caracteres por otros. Tiene la sintáxis:

`tr/SEARCHLIST/REPLACEMENTLIST/cds`
`y/SEARCHLIST/REPLACEMENTLIST/cds`

El operador permite el reemplazo carácter a carácter, por ejemplo:

```
$ perl -de 0
  DB<1> $a = 'fiboncacci'
  DB<2> $a =~ tr/aeiou/AEIOU/
  DB<3> print $a
fIbOncAccI
```

```
  DB<4> $a =~ y/fbnc/FBNC/
  DB<5> print $a
FIBONCACCI
```

El operador devuelve el número de carácteres reeemplazados o suprimidos.

```
$cnt = $sky =~ tr/*/*/; # count the stars in $sky
```

Si se especifica el modificador `/d`, cualquier carácter en `SEARCHLIST` que no figure en `REPLACEMENTLIST` es eliminado.

```
  DB<6> print $a
FIBONCACCI
  DB<7> $a =~ y/OA//d
  DB<8> print $a
FIBNCCCI
```

Si se especifica el modificador `/s`, las secuencias de carácteres consecutivos que serían traducidas al mismo carácter son comprimidas a una sola:

```
  DB<1> $b = 'aaghhh!'
  DB<2> $b =~ tr/ah//s
  DB<3> p $b
agh!
```

Observa que si la cadena `REPLACEMENTLIST` es vacía, no se introduce ninguna modificación.

Si se especifica el modificador `/c`, se complementa `SEARCHLIST`; esto es, se buscan los caracteres que no están en `SEARCHLIST`.

```
tr/a-zA-Z/ /cs; # change non-alphas to single space
```

Cuando se dan múltiples traducciones para un mismo carácter, solo la primera es utilizada:

```
tr/AAA/XYZ/
```

traducirá `A` por `X`.

El siguiente *script* busca una expresión regular en el fichero de `passwords` e imprime los *login* de los usuarios que casan con dicha cadena. Para evitar posibles confusiones con las vocales acentuadas se usa el operador `tr`.

```
#!/usr/bin/perl -w
$search = shift(@ARGV) or die("you must provide a regexpr\n");
$search =~ y/ÁÉÍÓÚáéíóú/AEIOUaeiou/;
open(FILE,"/etc/passwd");
while ($line = <FILE>) {
  $line =~ y/ÁÉÍÓÚáéíóú/AEIOUaeiou/;
  if ($line =~ /$search/io) {
    @fields = split(":",$line);
    $login = $fields[0];
    if ($line !~ /^#/) {
      print "$login\n";
    }
    else {
      print "#$login\n";
    }
  }
}

```

Ejecución (suponemos que el nombre del fichero anterior es `split.pl`):

```
> split.pl Rodriguez
##direccion
call
casiano
alu5
alu6
##doctorado
paco
falmeida
##ihiu07
```

Para familiarizarte con este operador, codifica y prueba el siguiente código:

```
 1 #!/usr/bin/perl -w
 2 $searchlist = shift @ARGV;
 3 $replacelist = shift @ARGV;
 4 $option = "";
 5 $option = shift @ARGV if @ARGV;
 6
 7 while (<>) {
 8   $num = eval "tr/$searchlist/$replacelist/$option";
 9   die $@ if $@;
10   print "$num: $_";
11 }
```

Perl construye la tabla de traducción en "tiempo de compilación". Por ello ni `SEARCHLIST` ni `REPLACEMENTLIST` son susceptibles de ser interpolados. Esto significa que si queremos usar variables tenemos que recurrir a la función `eval`.

La expresión pasada como parámetro a `eval` en la línea 8 es analizada y ejecutada como si se tratara de un pequeño programa Perl. Cualquier asignación a variables permanece después del `eval`, asi como cualquier definición de subrutina. El código dentro de `eval` se trata como si fuera un bloque, de manera que cualesquiera variables locales (declaradas con `my`) desaparecen al final del bloque.

La variable `$@` contiene el mensaje de error asociado con la última ejecución del comando `eval`. Si es nula es que el último comando se ejecuto correctamente. Aqui tienes un ejemplo de llamada:

```
> tr.pl 'a-z' 'A-Z' s
jose hernandez
13: JOSE HERNANDEZ
joosee hernnandez
16: JOSE HERNANDEZ
```

3.6. Pack y Unpack

El operador `pack` trabaja de forma parecida a `sprintf`. Su primer argumento es una cadena, seguida de una lista de valores a formatear y devuelve una cadena:

```
pack("CCC", 65, 66, 67, 68) # empaquetamos A B C D
```

el inverso es el operador `unpack`

```
unpack("CCC", "ABCD")
```

La cadena de formato es una lista de especificadores que indican el tipo del dato que se va a empaquetar/desempaquetar. Cada especificador puede opcionalmente seguirse de un contador de repetición que indica el número de elementos a formatear. Si se pone un asterisco (*) se indica que la especificación se aplica a todos los elementos restantes de la lista.

Formato	Descripción
A	Una cadena completada con blancos
a	Una cadena completada con ceros
B	Una cadena binaria en orden descendente
b	Una cadena binaria en orden ascendente
H	Una cadena hexadecimal, los nibble altos primero
h	Una cadena hexadecimal, los nibble bajos primero

Ejemplo de uso del formato `A`:

```
DB<1> $a = pack "A2A3", "Pea","rl"
DB<2> p $a
Perl
DB<3> @b = unpack "A2A3", "Perl"
DB<4> p "@b"
Pe rl
```

La variable `@b` tiene ahora dos cadenas. Una es `Pe` la otra es `rl`. Veamos un ejemplo con el formato `B`:

```
p ord('A')
65
DB<22> $x = pack "B8", "01000001"
DB<23> p $x
A
DB<24> @y = unpack "B8", "A"
DB<25> p "@y"
01000001
DB<26> $x = pack "b8", "10000010"
DB<27> p $x
```

3.7. Práctica: Un lenguaje para Componer Invitaciones

En el capítulo 6 (sección 6.4.2.2) del libro The LaTex Web Companion se define un lenguaje para componer textos para enviar invitaciones.

Para escribir una invitación en ese lenguaje escribiríamos algo así:

```
pl@nereida:~/Lpl0910/Practicas/161009/src$ cat -n invitation.xml
<?xml version="1.0"?>
<!DOCTYPE invitation SYSTEM "invitation.dtd">
<invitation>
<!-- ++++ The header part of the document ++++ -->
<front>
<to>Anna, Bernard, Didier, Johanna</to>
<date>Next Friday Evening at 8 pm</date>
<where>The Web Cafe</where>
<why>My first XML baby</why>
</front>
<!-- +++++ The main part of the document +++++ -->
<body>
<par>
I would like to invite you all to celebrate
the birth of <emph>Invitation</emph>, my
```

```
first XML document child.
</par>
<par>
Please do your best to come and join me next Friday
evening. And, do not forget to bring your friends.
</par>
<par>
I <emph>really</emph> look forward to see you soon!
</par>
</body>
<!-- +++ The closing part of the document ++++ -->
<back>
<signature>Michel</signature>
</back>
</invitation>
```

La sintáxis del lenguaje queda reflejada en la siguiente *Document Type Definition* (*DTD*) que aparece en la sección 6.4.3 del libro de Goosens:

```
pl@nereida:~/Lpl0910/Practicas/161009/src$ cat -n invitation.dtd
<!-- invitation DTD   -->
<!-- May 26th 1998 mg -->
<!ELEMENT invitation (front, body, back) >
<!ELEMENT front      (to, date, where, why?) >
<!ELEMENT date       (#PCDATA) >
<!ELEMENT to         (#PCDATA) >
<!ELEMENT where      (#PCDATA) >
<!ELEMENT why        (#PCDATA) >
<!ELEMENT body       (par+) >
<!ELEMENT par        (#PCDATA|emph)* >
<!ELEMENT emph       (#PCDATA) >
<!ELEMENT back       (signature) >
<!ELEMENT signature  (#PCDATA) >
```

El objetivo de esta práctica es escribir un programa Perl que usando las extensiones para expresiones regulares presentes en la versión 5.10 reconozca el lenguaje anterior.

Véase también:

- The LaTex Web Companion
- Examples from The LaTeX Web Companion (véanse los subdirectorios correspondietnes a los capítulos 6 y 7)

3.8. Analisis Sintáctico con Expresiones Regulares Perl

3.8.1. Introducción al Analísis Sintáctico con Expresiones Regulares

Como se ha comentado en la sección 3.2.5 Perl 5.10 permite el reconocimiento de expresiones definidas mediante gramáticas recursivas, siempre que estas puedan ser analizadas por un analizador recursivo descendente. Sin embargo, las expresiones regulares Perl 5.10 hace difícil construir una representación del árbol de análisis sintáctico abstracto. Además, la necesidad de explicitar en la regexp los blancos existentes entre los símbolos hace que la descripción sea menos robusta y menos legible.

Ejemplo: Traducción de expresiones aritméticas en infijo a postfijo

El siguiente ejemplo muestra una expresión regular que traduce expresiones de diferencias en infijo a postfijo.

Se usa una variable `$tran` para calcular la traducción de la subexpresión vista hasta el momento.

La gramática original que consideramos es recursiva a izquierdas:

```
 exp ->   exp '-' digits
        | digits
```

aplicando las técnicas explicadas en 6.8.1 y en el nodo de perlmonks 553889 transformamos la gramática en:

```
exp ->   digits rest
rest ->   '-' rest
        | # empty
```

Sigue el código:

```
pl@nereida:~/Lperltesting$ cat -n infixtopostfix.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1
use v5.10;

# Infix to postfix translator using 5.10 regexp
# original grammar:
#  exp ->   exp '-' digits
#         | digits
#
# Applying left-recursion elimination we have:
#  exp ->   digits rest
#  rest ->   '-' rest
#          | # empty
#
my $input;
local our $tran = '';

my $regexp = qr{
    (?&exp)

    (?(DEFINE)
        (?<exp>    ((?&digits)) \s* (?{ $tran .= "$^N "; say "tran=$tran"; }) (?&rest)
                      (?{
                         say "exp -> digits($^N) rest";
                      })
        )

        (?<rest>     \s* - ((?&digits)) (?{ $tran .= "$^N - "; say "tran=$tran"; }) (?&
                        (?{
                           say "rest -> - digits($^N) rest";
                        })
                  |  # empty
                        (?{
                           say "rest -> empty";
                        })
        )

```

```
            (?<digits>  \s* (\d+)
            )
        )
    }xms;

    $input = <>;
    chomp($input);
    if ($input =~ $regexp) {
      say "matches: $&\ntran=$tran";
    }
    else {
      say "does not match";
    }
```

La variable `$^N` contiene el valor que casó con el último paréntesis. Al ejecutar el código anterior obtenemos:

Véase la ejecución:

```
pl@nereida:~/Lperltesting$ ./infixtopostfix.pl
ab 5 - 3 -2 cd;
tran= 5
tran= 5  3 -
tran= 5  3 - 2 -
rest -> empty
rest -> - digits(2) rest
rest -> - digits( 3) rest
exp -> digits( 5) rest
matches:  5 - 3 -2
tran= 5  3 - 2 -
```

Como se ve, el recorrido primero profundo se traduce en la reconstrucción de una derivación a derechas.

Accediendo a los atributos de paréntesis anteriores mediante acciones intermedias

Es difícil extender el ejemplo anterior a lenguajes mas complejos debido a la limitación de que sólo se dispone de acceso al último paréntesis vía `$^N`. En muchos casos es necesario poder acceder a paréntesis/atributos anteriores.

El siguiente código considera el caso de expresiones con sumas, restas, multiplicaciones y divisiones. Utiliza la variable `op` y una acción intermedia (líneas 51-53) para almacenar el segundo paréntesis necesitado:

```
pl@nereida:~/Lperltesting$ cat -n ./calc510withactions3.pl
 1    #!/usr/local/lib/perl/5.10.1/bin//perl5.10.1
 2    use v5.10;
 3
 4    # Infix to postfix translator using 5.10 regexp
 5    # Original grammar:
 6
 7    #  exp ->   exp [-+] term
 8    #         | term
 9    #  term ->   term [*/] digits
10    #          | digits
11
12    # Applying left-recursion elimination we have:
13
```

```
#  exp  ->   term re
#  re   ->   [+-] term re
#          | # empty
#  term ->   digits rt
#  rt   ->   [*/] rt
#          | # empty


my $input;
my @stack;

local our $op = '';
my $regexp = qr{
    (?&exp)

    (?(DEFINE)
        (?<exp>    (?&term) (?&re)
                     (?{ say "exp -> term re" })
        )

        (?<re>     \s* ([+-]) (?&term) \s* (?{ push @stack, $^N }) (?&re)
                     (?{ say "re -> [+-] term re" })
                 | # empty
                     (?{ say "re -> empty" })
        )

        (?<term>   ((?&digits))
                      (?{ # intermediate action
                          push @stack, $^N
                      })
                   (?&rt)
                      (?{
                          say "term-> digits($^N) rt";
                      })
        )

        (?<rt>     \s*([*/])
                           (?{ # intermediate action
                              local $op = $^N;
                           })
                   ((?&digits)) \s*
                           (?{  # intermediate action
                                push @stack, $^N, $op
                            })
                   (?&rt) # end of <rt> definition
                           (?{
                                say "rt -> [*/] digits($^N) rt"
                            })
                  |  # empty
                     (?{ say "rt -> empty" })
        )

        (?<digits>  \s* \d+
```

```
            )
        )
    }xms;

    $input = <>;
    chomp($input);
    if ($input =~ $regexp) {
      say "matches: $&\nStack=(@stack)";
    }
    else {
      say "does not match";
    }
```

Sigue una ejecución:

```
pl@nereida:~/Lperltesting$ ./calc510withactions3.pl
5-8/4/2-1
rt -> empty
term-> digits(5) rt
rt -> empty
rt -> [*/] digits(2) rt
rt -> [*/] digits(4) rt
term-> digits(8) rt
rt -> empty
term-> digits(1) rt
re -> empty
re -> [+-] term re
re -> [+-] term re
exp -> term re
matches: 5-8/4/2-1
Stack=(5 8 4 / 2 / - 1 -)
```

Accediendo a los atributos de paréntesis anteriores mediante `@-`

Sigue una solución alternativa que obvia la necesidad de introducir incómodas acciones intermedias. Utilizamos las variables `@-` y `@+`:

> *Since Perl 5.6.1 the special variables* `@-` *and* `@+` *can functionally replace* `` $` ``, `$&` *and* `$'`*. These arrays contain pointers to the beginning and end of each match (see* `perlvar` *for the full story), so they give you essentially the same information, but without the risk of excessive string copying.*

Véanse los párrafos en las páginas 121, 121) y 122 para mas información sobre `@-` y `@+`.

Nótese la función `rc` en las líneas 21-28. `rc(1)` nos retorna lo que casó con el último paréntesis, `rc(2)` lo que casó con el penúltimo, etc.

```
pl@nereida:~/Lperltesting$ cat -n calc510withactions4.pl
 1    #!/usr/local/lib/perl/5.10.1/bin//perl5.10.1
 2    use v5.10;
 3
 4    # Infix to postfix translator using 5.10 regexp
 5    # Original grammar:
 6
 7    #  exp ->   exp [-+] term
 8    #         | term
 9    #  term ->   term [*/] digits
```

```
#          | digits

# Applying left-recursion elimination we have:

#  exp  ->   term re
#  re   ->   [+-] term re
#          | # empty
#  term ->   digits rt
#  rt   ->   [*/] rt
#          | # empty

sub rc {
  my $ofs = - shift;

  # Number of parenthesis that matched
  my $np = @-;
  #      string, ofsset, length
  substr($_, $-[$ofs], $+[$np+$ofs] - $-[$ofs])
}

my $input;
my @stack;

my $regexp = qr{
    (?&exp)

    (?(DEFINE)
        (?<exp>    (?&term) (?&re)
                     (?{ say "exp -> term re" })
        )

        (?<re>     \s* ([+-]) (?&term) \s* (?{ push @stack, rc(1) }) (?&re)
                     (?{ say "re -> [+-] term re" })
                 | # empty
                     (?{ say "re -> empty" })
        )

        (?<term>   ((?&digits))
                      (?{ # intermediate action
                          push @stack, rc(1)
                      })
                   (?&rt)
                      (?{
                          say "term-> digits(".rc(1).") rt";
                      })
        )

        (?<rt>     \s*([*/]) ((?&digits)) \s*
                           (?{  # intermediate action
                                push @stack, rc(1), rc(2)
                            })
                   (?&rt) # end of <rt> definition
                           (?{
```

```
                                say "rt -> [*/] digits(".rc(1).") rt"
                            })
                    |  # empty
                       (?{ say "rt -> empty" })
          )

          (?<digits>  \s* \d+
          )
      )
}xms;

$input = <>;
chomp($input);
if ($input =~ $regexp) {
  say "matches: $&\nStack=(@stack)";
}
else {
  say "does not match";
}
```

Ahora accedemos a los atributos asociados con los dos paréntesis, en la regla de `<rt>` usando la función `rc`:

```
         (?<rt>     \s*([*/]) ((?&digits)) \s*
                            (?{  # intermediate action
                                 push @stack, rc(1), rc(2)
                             })
```

Sigue una ejecución del programa:

```
pl@nereida:~/Lperltesting$ ./calc510withactions4.pl
5-8/4/2-1
rt -> empty
term-> digits(5) rt
rt -> empty
rt -> [*/] digits(2) rt
rt -> [*/] digits(4) rt
term-> digits(8) rt
rt -> empty
term-> digits(1) rt
re -> empty
re -> [+-] term re
re -> [+-] term re
exp -> term re
matches: 5-8/4/2-1
Stack=(5 8 4 / 2 / - 1 -)
pl@nereida:~/Lperltesting$
```

Accediendo a los atributos de paréntesis anteriores mediante paréntesis con nombre

Una nueva solución: dar nombre a los paréntesis y acceder a los mismos:

```
          (?<rt>     \s*(?<op>[*/]) (?<num>(?&digits)) \s*
                            (?{  # intermediate action
                                 push @stack, $+{num}, $+{op}
                             })
```

Sigue el código completo:

```
pl@nereida:~/Lperltesting$ cat -n ./calc510withnamedpar.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1
use v5.10;

# Infix to postfix translator using 5.10 regexp
# Original grammar:

#  exp ->   exp [-+] term
#         | term
#  term ->   term [*/] digits
#          | digits

# Applying left-recursion elimination we have:

#  exp  ->   term re
#  re   ->   [+-] term re
#          | # empty
#  term ->   digits rt
#  rt   ->   [*/] rt
#          | # empty

my @stack;

my $regexp = qr{
    (?&exp)

    (?(DEFINE)
        (?<exp>    (?&term) (?&re)
                     (?{ say "exp -> term re" })
        )

        (?<re>     \s* ([+-]) (?&term) \s* (?{ push @stack, $^N }) (?&re)
                     (?{ say "re -> [+-] term re" })
                 | # empty
                     (?{ say "re -> empty" })
        )

        (?<term>   ((?&digits))
                      (?{ # intermediate action
                          push @stack, $^N
                      })
                   (?&rt)
                      (?{
                          say "term-> digits($^N) rt";
                      })
        )

        (?<rt>     \s*(?<op>[*/]) (?<num>(?&digits)) \s*
                           (?{  # intermediate action
                                push @stack, $+{num}, $+{op}
                            })
                   (?&rt) # end of <rt> definition
```

```
                           (?{
                                say "rt -> [*/] digits($^N) rt"
                            })
                  |  # empty
                     (?{ say "rt -> empty" })
        )

        (?<digits>  \s* \d+
        )
    )
}xms;

my $input = <>;
chomp($input);
if ($input =~ $regexp) {
  say "matches: $&\nStack=(@stack)";
}
else {
  say "does not match";
}
```

Ejecución:

```
pl@nereida:~/Lperltesting$ ./calc510withnamedpar.pl
5-8/4/2-1
rt -> empty
term-> digits(5) rt
rt -> empty
rt -> [*/] digits(2) rt
rt -> [*/] digits(4) rt
term-> digits(8) rt
rt -> empty
term-> digits(1) rt
re -> empty
re -> [+-] term re
re -> [+-] term re
exp -> term re
matches: 5-8/4/2-1
Stack=(5 8 4 / 2 / - 1 -)
```

Véase También

- El nodo *Backreference variables in code embedded inside Perl 5.10 regexps* en PerlMonks
- El nodo *Strange behavior of @- and @+ in perl5.10 regexps* en PerlMonks

3.8.2. Construyendo el AST con Expresiones Regulares 5.10

Construiremos en esta sección un traductor de infijo a postfijo utilizando una aproximación general: construiremos una representación del Abstract Syntax Tree o AST (véase la sección **??** Árbol de Análisis Abstracto para una definición detallada de que es un árbol sintáctico).

Como la aplicación es un poco mas compleja la hemos dividido en varios ficheros. Esta es la estructura:

.

```
|-- ASTandtrans3.pl    # programa principal
|-- BinaryOp.pm        # clases para el manejo de los nodos del AST
|-- testreegxpparen.pl # prueba para Regexp::Paren
`-- Regexp
    `-- Paren.pm       # módulo de extensión de $^N
```

La salida del programa puede ser dividida en tres partes. La primera muestra una antiderivación a derechas inversa:

```
pl@nereida:~/Lperltesting$ ./ASTandtrans3.pl
2*(3-4)
factor -> NUM(2)
factor -> NUM(3)
rt -> empty
term-> factor rt
factor -> NUM(4)
rt -> empty
term-> factor rt
re -> empty
re -> [+-] term re
exp -> term re
factor -> ( exp )
rt -> empty
rt -> [*/] factor rt
term-> factor rt
re -> empty
exp -> term re
matches: 2*(3-4)
```

Que leída de abajo a arriba nos da una derivación a derechas de la cadena `2*(3-4)`:

```
exp => term re => term => factor rt =>
factor [*/](*) factor rt => factor [*/](*) factor =>
factor [*/](*) ( exp ) => factor [*/](*) ( term re ) =>
factor [*/](*) ( term [+-](-) term re ) =>
factor [*/](*) ( term [+-](-) term ) =>
factor [*/](*) ( term [+-](-) factor rt ) =>
factor [*/](*) ( term [+-](-) factor ) =>
factor [*/](*) ( term [+-](-) NUM(4) ) =>
factor [*/](*) ( factor rt [+-](-) NUM(4) ) =>
factor [*/](*) ( factor [+-](-) NUM(4) ) =>
factor [*/](*) ( NUM(3) [+-](-) NUM(4) )  =>
NUM(2) [*/](*) ( NUM(3) [+-](-) NUM(4) )
```

La segunda parte nos muestra la representación del AST para la entrada dada (`2*(3-4)`):

```
AST:
$VAR1 = bless( {
  'left' => bless( {
  'right' => bless(
    'left' => bless(
    'right' => bless
    'op' => '-'
  }, 'ADD' ),
  'op' => '*'
}, 'MULT' );
```

La última parte de la salida nos muestra la traducción a postfijo de la expresión en infijo suministrada en la entrada (`2*(3-4)`):

```
2 3 4 - *
```

Programa Principal: usando la pila de atributos

La gramática original que consideramos es recursiva a izquierdas:

```
exp    ->   exp [-+] term
          | term
term   ->   term [*/] factor
          | factor
factor ->  \( exp \)
          | \d+
```

aplicando las técnicas explicadas en 6.8.2 es posible transformar la gramática en una no recursiva por la izquierda:

```
exp       ->   term restoexp
restoexp  ->   [-+] term restoexp
             | # vacío
term      ->   term restoterm
restoterm ->   [*/] factor restoterm
             | # vacío
factor    ->   \( exp \)
             | \d+
```

Ahora bien, no basta con transformar la gramática en una equivalente. Lo que tenemos como punto de partida no es una gramática sino un *esquema de traducción* (véase la sección 4.4) que construye el AST asociado con la expresión. Nuestro esquema de traducción conceptual es algo así:

```
exp    ->   exp ([-+]) term       { ADD->new(left => $exp, right => $term, op => $1) }
          | term                  { $term }
term   ->   term ([*/]) factor    { MULT->new(left => $exp, right => $term, op => $1) }
          | factor                { $factor }
factor ->  \( exp \)              { $exp }
          | (\d+)                 { NUM->new(val => $1) }
```

Lo que queremos conseguir un conjunto de acciones semánticas asociadas para gramática no recursiva que sea equivalente a este.

Este es el programa resultante una vez aplicadas las transformaciones. La implementación de la asociación entre símbolos y atributos la realizamos manualmente mediante una pila de atributos:

```
pl@nereida:~/Lperltesting$ cat -n ./ASTandtrans3.pl
#!/usr/local/lib/perl/5.10.1/bin//perl5.10.1
use v5.10;
use strict;
use Regexp::Paren qw{g};
use BinaryOp;

use Data::Dumper;
$Data::Dumper::Indent = 1;

# Builds AST
my @stack;
my $regexp = qr{
```

```
    (?&exp)

    (?(DEFINE)
        (?<exp>    (?&term) (?&re)
                     (?{ say "exp -> term re" })
        )

        (?<re>     \s* ([+-]) (?&term)
                      (?{  # intermediate action
                          local our ($ch1, $term) = splice @stack, -2;

                          push @stack, ADD->new( {left => $ch1, right => $term, op => g(1)
                      })
                   (?&re)
                     (?{ say "re -> [+-] term re" })
                 | # empty
                     (?{ say "re -> empty" })
        )

        (?<term>   ((?&factor)) (?&rt)
                      (?{
                          say "term-> factor rt";
                      })
        )

        (?<rt>     \s*([*/]) (?&factor)
                       (?{  # intermediate action
                            local our ($ch1, $ch2) = splice @stack, -2;

                            push @stack, MULT->new({left => $ch1, right => $ch2, op => g(1
                        })
                   (?&rt) # end of <rt> definition
                       (?{
                            say "rt -> [*/] factor rt"
                        })
                 | # empty
                       (?{ say "rt -> empty" })
        )

        (?<factor> \s* (\d+)
                        (?{
                           say "factor -> NUM($^N)";
                           push @stack, bless { 'val' => g(1) }, 'NUM';
                        })
                   | \s* \( (?&exp) \s* \)
                        (?{ say "factor -> ( exp )" })
        )
    )
}xms;

my $input = <>;
chomp($input);
if ($input =~ $regexp) {
```

```
  say "matches: $&";
  my $ast = pop @stack;
  say "AST:\n", Dumper $ast;

  say $ast->translate;
}
else {
  say "does not match";
}
```

Las Clases representando a los AST

Cada nodo del AST es un objeto. La clase del nodo nos dice que tipo de nodo es. Así los nodos de la clase `MULT` agrupan a los nódos de multiplicación y división. Los nodos de la clase `ADD` agrupan a los nódos de suma y resta. El procedimiento general es asociar un método `translate` con cada clase de nodo. De esta forma se logra el polimorfismo necesario: cada clase de nodo sabe como traducirse y el método `translate` de cada clase puede escribirse como

- Obtener los resultados de llamar a `$child->translate` para cada uno de los nodos hijos `$child`. Por ejemplo, si el nodo fuera un nodo `IF_ELSE` de un hipotético lenguaje de programación, se llamaría a los métodos `translate` sobre sus tres hijos `boolexpr`, `ifstatement` y `elsestatement`.
- Combinar los resultados para producir la traducción adecuada del nodo actual.

Es esta combinación la que mas puede cambiar según el tipo de nodo. Así, en el caso de el nodo `IF_ELSE` el seudocódigo para la traducción sería algo parecido a esto:

```
my $self = shift;
my $etiqueta1 = generar_nueva_etiqueta;
my $etiqueta2 = generar_nueva_etiqueta;

my $boolexpr      = $self->boolexpr->translate;
my $ifstatement   = $self->ifstatement->translate,
my $elsestatement = $self->elsestatement->translate,
return << "ENDTRANS";
    $boolexpr
    JUMPZERO $etiqueta1:
    $ifstatement
    JUMP     $etiqueta2:
  $etiqueta1:
    $elsestatement
  $etiqueta2:
ENDTRANS
```

Siguiendo estas observaciones el código de `BinaryOp.pm` queda así:

```
pl@nereida:~/Lperltesting$ cat -n BinaryOp.pm
package BinaryOp;
use strict;
use base qw(Class::Accessor);

BinaryOp->mk_accessors(qw{left right op});

sub translate {
  my $self = shift;

```

```
  return $self->left->translate." ".$self->right->translate." ".$self->op;
}

package ADD;
use base qw{BinaryOp};

package MULT;
use base qw{BinaryOp};

package NUM;

sub translate {
  my $self = shift;

  return $self->{val};
}

1;
```

Véase también:

- `Class::Accessor`

Accediendo a los paréntesis lejanos: El módulo `Regexp::Paren`

En esta solución utilizamos las variables `@-` y `@+` para construir una función que nos permite acceder a lo que casó con los últimos paréntesis con memoria:

> *Since Perl 5.6.1 the special variables* `@-` *and* `@+` *can functionally replace* `` $` ``, `$&` *and* `$'`*. These arrays contain pointers to the beginning and end of each match (see* `perlvar` *for the full story), so they give you essentially the same information, but without the risk of excessive string copying.*

Véanse los párrafos en las páginas 121, 121) y 122 para mas información sobre `@-` y `@+`.

`g(1)` nos retorna lo que casó con el último paréntesis, `g(2)` lo que casó con el penúltimo, etc.

```
pl@nereida:~/Lperltesting$ cat -n Regexp/Paren.pm
package Regexp::Paren;
use strict;

use base qw{Exporter};

our @EXPORT_OK = qw{g};

sub g {
  die "Error in 'Regexp::Paren::g'. Not used inside (?{ code }) construct\n" unless define
  my $ofs = - shift;

  # Number of parenthesis that matched
  my $np = @-;
  die "Error. Illegal 'Regexp::Paren::g' ref inside (?{ code }) construct\n" unless ($np >
  # $_ contains the string being matched
  substr($_, $-[$ofs], $+[$np+$ofs] - $-[$ofs])
}

1;
```

```

=head1 NAME

Regexp::Paren - Extends $^N inside (?{ ... }) constructs

=head1 SYNOPSIS

  use Regexp::Paren qw{g};

  'abcde' =~ qr{(.)(.)(.)
                     (?{ print g(1)." ".g(2)." ".g(3)."\n" })                  # c b a
               (.)     (?{ print g(1)." ".g(2)." ".g(3)." ".g(4)."\n" })         # d c b
               (.)     (?{ print g(1)." ".g(2)." ".g(3)." ".g(4)." ".g(5)."\n" }) # e d c
              }x;

  print g(1)." ".g(2)." ".g(3)." ".g(4)." ".g(5)."\n"; # error!

=head1 DESCRIPTION

Inside a C<(?{ ... })> construct, C<g(1)> refers to what matched the last parenthesis
(like C<$^N>), C<g(2)> refers to the string that matched with the parenthesis before
the last, C<g(3)> refers to the string that matched with the parenthesis at distance 3,
etc.

=head1 SEE ALSO

=over 2

=item * L<perlre>

=item * L<perlretut>

=item * PerlMonks node I<Strange behavior o> C<@-> I<and> C<@+> I<in perl5.10 regexps> L<h

=item * PerlMonks node I<Backreference variables in code embedded inside Perl 5.10 regexps

=back

=head1 AUTHOR

Casiano Rodriguez-Leon (casiano@ull.es)

=head1 ACKNOWLEDGMENTS

This work has been supported by CEE (FEDER) and the Spanish Ministry of
I<Educacion y Ciencia> through I<Plan Nacional I+D+I> number TIN2005-08818-C04-04
(ULL::OPLINK project L<http://www.oplink.ull.es/>).
Support from Gobierno de Canarias was through GC02210601
(I<Grupos Consolidados>).
The University of La Laguna has also supported my work in many ways
and for many years.

=head1 LICENCE AND COPYRIGHT
```

```

Copyright (c) 2009- Casiano Rodriguez-Leon (casiano@ull.es). All rights reserved.

These modules are free software; you can redistribute it and/or
modify it under the same terms as Perl itself. See L<perlartistic>.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Al ejecutar `perldoc Regexp::Paren` podemos ver la documentación incluida (véase la documentación en `perlpod` y `perlpodspec` así como la sección La Documentación en Perl para mas detalles):

```
NAME
    Regexp::Paren - Extends $^N inside (?{ ... }) constructs

SYNOPSIS
      use Regexp::Paren qw{g};

      'abcde' =~ qr{(.)(.)(.)
                     (?{ print g(1)." ".g(2)." ".g(3)."\n" })                 # c
                (.)  (?{ print g(1)." ".g(2)." ".g(3)." ".g(4)."\n" })        # d
                (.)  (?{ print g(1)." ".g(2)." ".g(3)." ".g(4)." ".g(5)."\n" }) # e
               }x;

      print g(1)." ".g(2)." ".g(3)." ".g(4)." ".g(5)."\n"; # error!

DESCRIPTION
    Inside a "(?{ ... })" construct, g(1) refers to what matched the last
    parenthesis (like $^N), g(2) refers to the string that matched with the
    parenthesis before the last, g(3) refers to the string that matched with
    the parenthesis at distance 3, etc.

SEE ALSO
    * perlre
    * perlretut
    * PerlMonks node *Strange behavior o* "@-" *and* "@+" *in perl5.10
      regexps* <http://www.perlmonks.org/?node_id=794736>
    * PerlMonks node *Backreference variables in code embedded inside Perl
      5.10 regexps* <http://www.perlmonks.org/?node_id=794424>

AUTHOR
    Casiano Rodriguez-Leon (casiano@ull.es)

ACKNOWLEDGMENTS
    This work has been supported by CEE (FEDER) and the Spanish Ministry of
    *Educacion y Ciencia* through *Plan Nacional I+D+I* number
    TIN2005-08818-C04-04 (ULL::OPLINK project <http://www.oplink.ull.es/>).
    Support from Gobierno de Canarias was through GC02210601 (*Grupos
    Consolidados*). The University of La Laguna has also supported my work
    in many ways and for many years.

LICENCE AND COPYRIGHT
    Copyright (c) 2009- Casiano Rodriguez-Leon (casiano@ull.es). All rights
```

3.9. Práctica: Traducción de invitation a HTML

Esta práctica es continuación de la práctica *un lenguaje para componer invitaciones* especificada en la sección 3.7.

El objetivo es traducir la entrada escrita en el lenguaje de invitaciones a HTML. La traducción del ejemplo anterior debería ser parecida a esta:

```
pl@nereida:~/Lpl0910/Practicas/161009/src$ cat -n invitat
<?xml version="1.0"?>
<!DOCTYPE invitation SYSTEM "invitation.dtd">
<invitation>
<!-- ++++ The header part of the document ++++ -->
<front>
<to>Anna, Bernard, Didier, Johanna</to>
<date>Next Friday Evening at 8 pm</date>
<where>The Web Cafe</where>
<why>My first XML baby</why>
</front>
<!-- +++++ The main part of the document +++++ -->
<body>
<par>
I would like to invite you all to celebrate
the birth of <emph>Invitation</emph>, my
first XML document child.
</par>
<par>
Please do your best to come and join me next Friday
evening. And, do not forget to bring your friends.
</par>
<par>
I <emph>really</emph> look forward to see you soon!
</par>
</body>
<!-- +++ The closing part of the document ++++ -->
<back>
<signature>Michel</signature>
</back>
</invitation>
```

Para ver el resultado en su navegador visite el fichero invitation.html

Su programa deberá producir un Abstract Syntax Tree. Los nodos serán objetos. Cada clase (`FRONT`, `TO`, etc.) deberá de disponer de un método `translate`.

Para simplificar el proceso de traducción a `HTML` se sugiere utilizar una hoja de estilo parecida a la siguiente (tomada de la seción 7.4.4 del citado libro de Goosens):

```
pl@nereida:~/Lpl0910/Practicas/161009/src$ cat -n invit.css
/* CSS stylesheet for invitation1 in HTML */
BODY {margin-top: 1em;     /* global page parameters */
      margin-bottom: 1em;
      margin-left: 1em;
      margin-right: 1em;
      font-family: serif;
      line-height: 1.1;
      color: black;
}
```

```
H1   {text-align: center;  /* for global title   */
       font-size: x-large;
}
P    {text-align: justify; /* paragraphs in body */
       margin-top: 1em;
}
TABLE { border-width: 0pt }
TBODY { border-width: 0pt }
TD[class="front"] {        /* table data in front matter */
       text-align: left;
       font-weight: bold;
}
TD.front {         /* table data in front matter */
       text-align: left;
       font-weight: bold;
}
EM   {font-style: italic;  /* emphasis in body   */
}
P.signature {     /* signature          */
       text-align: right;
       font-weight: bold;
}
```

Véase también:

- The LaTex Web Companion
- Examples from The LaTeX Web Companion (véanse los subdirectorios correspondietnes a los capítulos 6 y 7)
- CSS Tutorial
- Edición extremadamente simple de HTML
- Perl-XML Frequently Asked Questions

3.10. Análisis Sintáctico con `Regexp::Grammars`

El módulo `Regexp::Grammars` escrito por Damian Conway extiende las expresiones regulares Perl con la capacidad de generar representaciones del árbol de análisis sintáctico abstracto y obviando la necesidad de explicitar los blancos. El módulo necesita para funcionar una versión de Perl superior o igual a la 5.10.

3.10.1. Introducción

El Problema

La documentación de `Regexp::Grammars` establece cual es el problema que aborda el módulo:

> *. . . Perl5.10 makes possible to use regexes to recognize complex, hierarchical–and even recursive–textual structures. The problem is that Perl 5.10 doesn't provide any support for extracting that hierarchical data into nested data structures. In other words, using Perl 5.10 you can match complex data, but not parse it into an internally useful form.*
>
> *An additional problem when using Perl 5.10 regexes to match complex data formats is that you have to make sure you remember to insert whitespace- matching constructs (such as `\s*`) at every possible position where the data might contain ignorable whitespace. This reduces the readability of such patterns, and increases the chance of errors (typically caused by overlooking a location where whitespace might appear).*

Una solución: `Regexp::Grammars`

The `Regexp::Grammars` *module solves both those problems.*

If you import the module into a particular lexical scope, it preprocesses any regex in that scope, so as to implement a number of extensions to the standard Perl 5.10 regex syntax. These extensions simplify the task of defining and calling subrules within a grammar, and allow those subrule calls to capture and retain the components of they match in a proper hierarchical manner.

La sintaxis de una expresión regular `Regexp::Grammars`

Las expresiones regulares `Regexp::Grammars` aumentan las regexp Perl 5.10. La sintáxis se expande y se modifica:

> *A* `Regexp::Grammars` *specification consists of a pattern (which may include both standard Perl 5.10 regex syntax, as well as special* `Regexp::Grammars` *directives), followed by one or more rule or token definitions.*

Sigue un ejemplo:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n balanced_brackets.pl
use strict;
use warnings;
use 5.010;
use Data::Dumper;

my $rbb = do {
    use Regexp::Grammars;
    qr{
      (<pp>)

      <rule: pp>   \( (?: [^()]*+ | <escape> | <pp> )* \)

      <token: escape> \\.

    }xs;
};

while (my $input = <>) {
    while ($input =~ m{$rbb}g) {
        say("matches: <$&>");
        say Dumper \%/;
    }
}
```

Note that there is no need to explicitly place `\s*` *subpatterns throughout the rules; that is taken care of automatically.*

...

The initial pattern (`(<pp>)`*) acts like the* top *rule of the grammar, and must be matched completely for the grammar to match.*

The rules and tokens are declarations only and they are not directly matched. Instead, they act like subroutines, and are invoked by name from the initial pattern (or from within a rule or token).

Each rule or token extends from the directive that introduces it up to either the next rule or token directive, or (in the case of the final rule or token) to the end of the grammar.

El hash %/: Una representación del AST Al ejecutar el programa anterior con entrada `(2*(3+5))*4+(2-3)` produce:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 balanced_brackets.pl
(2*(3+5))*4+(2-3)
matches: <(2*(3+5))>
$VAR1 = {
          '' => '(2*(3+5))',
          'pp' => {
                    '' => '(2*(3+5))',
                    'pp' => '(3+5)'
                  }
        };

matches: <(2-3)>
$VAR1 = {
          '' => '(2-3)',
          'pp' => '(2-3)'
        };
```

> *Each rule calls the subrules specified within it, and then return a hash containing whatever result each of those subrules returned, with each result indexed by the subrule's name.*
>
> *In this way, each level of the hierarchical regex can generate hashes recording everything its own subrules matched, so when the entire pattern matches, it produces a tree of nested hashes that represent the structured data the pattern matched.*
>
> ...
>
> *In addition each result-hash has one extra key: the empty string. The value for this key is whatever string the entire subrule call matched.*

Diferencias entre `token` y `rule`

> *The difference between a token and a rule is that a token treats any whitespace within it exactly as a normal Perl regular expression would. That is, a sequence of whitespace in a token is ignored if the `/x` modifier is in effect, or else matches the same literal sequence of whitespace characters (if `/x` is not in effect).*

En el ejemplo anterior el comportamiento es el mismo si se reescribe la regla para el token `escape` como:

```
    <rule: escape> \\.
```

En este otro ejemplo mostramos que la diferencia entre token y rule es significativa:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n tokenvsrule.pl
use strict;
use warnings;
use 5.010;
use Data::Dumper;

my $rbb = do {
    use Regexp::Grammars;
    qr{
      <s>

      <rule: s> <a> <c>
```

```

        <rule: c>  c d

        <token: a>  a b

    }xs;
};

while (my $input = <>) {
    if ($input =~ m{$rbb}) {
        say("matches: <$&>");
        say Dumper \%/;
    }
    else {
        say "Does not match";
    }
}
```

Al ejecutar este programa vemos la diferencia en la interpretación de los blancos:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 tokenvsrule.pl
ab c d
matches: <ab c d>
$VAR1 = {
          '' => 'ab c d',
          's' => {
                   '' => 'ab c d',
                   'c' => 'c d',
                   'a' => 'ab'
                 }
        };

a b c d
Does not match
ab cd
matches: <ab cd>
$VAR1 = {
          '' => 'ab cd',
          's' => {
                   '' => 'ab cd',
                   'c' => 'cd',
                   'a' => 'ab'
                 }
        };
```

Obsérvese como la entrada `a b c d` es rechazada mientras que la entrada `ab c d` es aceptada.

Redefinición de los espacios en blanco

> *In a rule, any sequence of whitespace (except those at the very start and the very end of the rule) is treated as matching the implicit subrule* `<.ws>`*, which is automatically predefined to match optional whitespace (i.e.* `\s*`*).*
>
> *You can explicitly define a* `<ws>` *token to change that default behaviour. For example, you could alter the definition of whitespace to include Perlish comments, by adding an explicit* `<token: ws>`*:*

```
<token: ws>
   (?: \s+ | #[^\n]* )*
```

> *But be careful not to define* `<ws>` *as a rule, as this will lead to all kinds of infinitely recursive unpleasantness.*

El siguiente ejemplo ilustra como redefinir `<ws>`:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n tokenvsruleandws.pl
use strict;
use warnings;
use 5.010;
use Data::Dumper;

my $rbb = do {
    use Regexp::Grammars;
    no warnings 'uninitialized';
    qr{
      <s>

      <token: ws> (?: \s+ | /\* .*? \*/)*+

      <rule: s> <a> <c>

      <rule: c>  c d

      <token: a>  a b

    }xs;
};

while (my $input = <>) {
    if ($input =~ m{$rbb}) {
        say Dumper \%/;
    }
    else {
        say "Does not match";
    }
}
```

Ahora podemos introducir comentarios en la entrada:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 -w tokenvsruleandws.pl
ab /* 1 */ c d
$VAR1 = {
          '' => 'ab /* 1 */ c d',
          's' => {
                   '' => 'ab /* 1 */ c d',
                   'c' => 'c d',
                   'a' => 'ab'
                 }
        };
```

Llamando a las subreglas

> *To invoke a rule to match at any point, just enclose the rule’s name in angle brackets (like in Perl 6). There must be no space between the opening bracket and the rulename. For example:*
>
> ```
> qr{
>     file:             # Match literal sequence ’f’ ’i’ ’l’ ’e’ ’:’
>     <name>            # Call <rule: name>
>     <options>?        # Call <rule: options> (it’s okay if it fails)
>
>     <rule: name>
>         # etc.
> }x;
> ```
>
> *If you need to match a literal pattern that would otherwise look like a subrule call, just backslash-escape the leading angle:*
>
> ```
> qr{
>     file:             # Match literal sequence ’f’ ’i’ ’l’ ’e’ ’:’
>     \<name>           # Match literal sequence ’<’ ’n’ ’a’ ’m’ ’e’ ’>’
>     <options>?        # Call <rule: options> (it’s okay if it fails)
>
>     <rule: name>
>         # etc.
> }x;
> ```

El siguiente programa ilustra algunos puntos discutidos en la cita anterior:

```
casiano@millo:~/src/perl/regexp-grammar-examples$ cat -n badbracket.pl
use strict;
use warnings;
use 5.010;
use Data::Dumper;

my $rbb = do {
    use Regexp::Grammars;
    qr{
      (<pp>)

      <rule: pp>   \( (?: <b  > | \< | < escape> | <pp> )* \)

      <token: b  > b

      <token: escape> \\.

    }xs;
};

while (my $input = <>) {
    while ($input =~ m{$rbb}g) {
        say("matches: <$&>");
        say Dumper \%/;
    }
}
```

Obsérvense los blancos en `< escape>` y en `<token: b  > b`. Pese a ello el programa funciona:

```
casiano@millo:~/src/perl/regexp-grammar-examples$ perl5.10.1 badbracket.pl
(\(\))
matches: <(\(\))>
$VAR1 = {
          '' => '(\\(\\))',
          'pp' => {
                    '' => '(\\(\\))',
                    'escape' => '\\)'
                  }
        };

(b)
matches: <(b)>
$VAR1 = {
          '' => '(b)',
          'pp' => {
                    '' => '(b)',
                    'b' => 'b'
                  }
        };

(<)
matches: <(<)>
$VAR1 = {
          '' => '(<)',
          'pp' => '(<)'
        };

(c)

casiano@millo:
```

Eliminación del anidamiento de ramas unarias en %/

> *... Note, however, that if the result-hash at any level contains only the empty-string key (i.e. the subrule did not call any sub-subrules or save any of their nested result-hashes), then the hash is unpacked and just the matched substring itself if returned.*
>
> *For example, if* `<rule: sentence>` *had been defined:*
>
> ```
> <rule: sentence>
>     I see dead people
> ```
>
> *then a successful call to the rule would only add:*
>
> ```
> sentence => 'I see dead people'
> ```
>
> *to the current result-hash.*
>
> *This is a useful feature because it prevents a series of nested subrule calls from producing very unwieldy data structures. For example, without this automatic unpacking, even the simple earlier example:*
>
> ```
> <rule: sentence>
>     <noun> <verb> <object>
> ```

would produce something needlessly complex, such as:

```
sentence => {
    ""     => 'I saw a dog',
    noun   => {
        "" => 'I',
    },
    verb   => {
        "" => 'saw',
    },
    object => {
        ""      => 'a dog',
        article => {
            "" => 'a',
        },
        noun    => {
            "" => 'dog',
        },
    },
}
```

El siguiente ejemplo ilustra este punto:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n unaryproductions.pl
use strict;
use warnings;
use 5.010;
use Data::Dumper;

my $rbb = do {
    use Regexp::Grammars;
    qr{
      <s>

      <rule: s> <noun> <verb> <object>

      <token: noun> he | she | Peter | Jane

      <token: verb> saw | sees

      <token: object> a\s+dog | a\s+cat

    }x;
};

while (my $input = <>) {
    while ($input =~ m{$rbb}g) {
        say("matches: <$&>");
        say Dumper \%/;
    }
}
```

Sigue una ejecución del programa anterior:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 unaryproductions.pl
```

```
he saw a dog
matches: <he saw a dog>
$VAR1 = {
          ’’ => ’he saw a dog’,
          ’s’ => {
                   ’’ => ’he saw a dog’,
                   ’object’ => ’a dog’,
                   ’verb’ => ’saw’,
                   ’noun’ => ’he’
                 }
        };

Jane sees a cat
matches: <Jane sees a cat>
$VAR1 = {
          ’’ => ’Jane sees a cat’,
          ’s’ => {
                   ’’ => ’Jane sees a cat’,
                   ’object’ => ’a cat’,
                   ’verb’ => ’sees’,
                   ’noun’ => ’Jane’
                 }
        };
```

Ámbito de uso de `Regexp::Grammars`

Cuando se usa `Regexp::Grammars` como parte de un programa que utiliza otras regexes hay que evitar que `Regexp::Grammars` procese las mismas. `Regexp::Grammars` reescribe las expresiones regulares durante la fase de preproceso. Esta por ello presenta las mismas limitaciones que cualquier otra forma de 'source filtering' (véase `perlfilter`). Por ello es una buena idea declarar la gramática en un bloque `do` restringiendo de esta forma el ámbito de acción del módulo.

```
my $calculator = do{
    use Regexp::Grammars;
    qr{
        ........
    }xms
};
```

3.10.2. Objetos

> *When a grammar has parsed successfully, the* `%/` *variable will contain a series of nested hashes (and possibly arrays) representing the hierarchical structure of the parsed data.*
>
> *Typically, the next step is to walk that tree, extracting or converting or otherwise processing that information. If the tree has nodes of many different types, it can be difficult to build a recursive subroutine that can navigate it easily.*
>
> *A much cleaner solution is possible if the nodes of the tree are proper objects. In that case, you just define a* `trasnlate()` *method for each of the classes, and have every node call that method on each of its children. The chain of* `translate()` *calls would cascade down the nodes of the tree, each one invoking the appropriate* `translate()` *method according to the type of node encountered.*
>
> *The only problem is that, by default,* `Regexp::Grammars` *returns a tree of plain-old hashes, not* `Class::Whatever` *objects. Fortunately, it's easy to request that the result has-*

hes be automatically blessed into the appropriate classes, using the `<objrule:...>` *and* `<objtoken:...>` *directives.*

These directives are identical to the `<rule:...>` *and* `<token:...>` *directives (respectively), except that the rule or token they create will also bless the hash it normally returns, converting it to an object of a class whose name is the same as the rule or token itself.*

For example:

```
<objrule: Element>
    # ...Defines a rule that can be called as <Element>
    # ...and which returns a hash-based Element object
```

The `IDENTIFIER` *of the rule or token may also be fully qualified. In such cases, the rule or token is defined using only the final* short name, *but the result object is blessed using the fully qualified* long name. *For example:*

```
<objrule: LaTeX::Element>
    # ...Defines a rule that can be called as <Element>
    # ...and which returns a hash-based LaTeX::Element object
```

This can be useful to ensure that returned objects don't collide with other namespaces in your program.

Note that you can freely mix object-returning and plain-old-hash-returning rules and tokens within a single grammar, though you have to be careful not to subsequently try to call a method on any of the unblessed nodes.

3.10.3. Renombrando los resultados de una subregla

Nombre de la regla versus Nombre del Resultado

No siempre el nombre de la regla es el mas apropiado para ser el nombre del resultado:

> *It is not always convenient to have subrule results stored under the same name as the rule itself. Rule names should be optimized for understanding the behaviour of the parser, whereas result names should be optimized for understanding the structure of the data. Often those two goals are identical, but not always; sometimes rule names need to describe what the data looks like, while result names need to describe what the data means.*

Colisión de nombres de reglas

For example, sometimes you need to call the same rule twice, to match two syntactically identical components whose positions give then semantically distinct meanings:

```
<rule: copy_cmd>
    copy <file> <file>
```

The problem here is that, if the second call to `<file>` *succeeds, its result-hash will be stored under the key* `file`*, clobbering the data that was returned from the first call to* `<file>`*.*

Aliasing

To avoid such problems, `Regexp::Grammars` *allows you to alias any subrule call, so that it is still invoked by the original name, but its result-hash is stored under a different key. The syntax for that is:* `<alias=rulename>`*. For example:*

```
<rule: copy_cmd>
    copy <from=file> <to=file>
```

Here, `<rule: file>` *is called twice, with the first result-hash being stored under the key* `from`*, and the second result-hash being stored under the key* `to`*.*

Note, however, that the alias before the `=` *must be a proper identifier (i.e. a letter or underscore, followed by letters, digits, and/or underscores). Aliases that start with an underscore and aliases named* `MATCH` *have special meaning.*

Normalización de los resultados mediante aliasing

Aliases can also be useful for normalizing data that may appear in different formats and sequences. For example:

```
<rule: copy_cmd>
    copy <from=file>        <to=file>
  | dup    <to=file>  as  <from=file>
  |      <from=file>  ->    <to=file>
  |        <to=file>  <-  <from=file>
```

Here, regardless of which order the old and new files are specified, the result-hash always gets:

```
copy_cmd => {
    from => 'oldfile',
      to => 'newfile',
}
```

Ejemplo

El siguiente programa ilustra los comentarios de la documentación:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n copygrammar.pl
use strict;
use warnings;
use 5.010;
use Data::Dumper;

my $rbb = do {
    use Regexp::Grammars;
    qr{
      <copy_cmd>

      <rule: copy_cmd>
            copy <from=file> <to=file>
        |   <from=file> ->   <to=file>
        |   <to=file>   <- <from=file>

      <token: file> [\w./\\]+
    }x;
};

while (my $input = <>) {
    while ($input =~ m{$rbb}g) {
        say("matches: <$&>");
        say Dumper \%/;
    }
}
```

Cuando lo ejecutamos obtenemos:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 copygrammar.pl
copy a b
matches: <copy a b>
$VAR1 = {
          '' => 'copy a b',
          'copy_cmd' => {
                          '' => 'copy a b',
                          'to' => 'b',
                          'from' => 'a'
                        }
        };

b <- a
matches: <b <- a>
$VAR1 = {
          '' => 'b <- a',
          'copy_cmd' => {
                          '' => 'b <- a',
                          'to' => 'b',
                          'from' => 'a'
                        }
        };

a -> b
matches: <a -> b>
$VAR1 = {
          '' => 'a -> b',
          'copy_cmd' => {
                          '' => 'a -> b',
                          'to' => 'b',
                          'from' => 'a'
                        }
        };
```

3.10.4. Listas

El operador de cierre positivo

If a subrule call is quantified with a repetition specifier:

```
<rule: file_sequence>
    <file>+
```

then each repeated match overwrites the corresponding entry in the surrounding rule's result-hash, so only the result of the final repetition will be retained. That is, if the above example matched the string `foo.pl bar.py baz.php`*, then the result-hash would contain:*

```
file_sequence {
    ""   => 'foo.pl bar.py baz.php',
    file => 'baz.php',
}
```

Operadores de listas y espacios en blanco

Existe un caveat con el uso de los operadores de repetición y el manejo de los blancos. Véase el siguiente programa:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n numbers3.pl
use strict;
use warnings;
use 5.010;
use Data::Dumper;

my $rbb = do {
    use Regexp::Grammars;

    qr{
      <numbers>

      <rule: numbers>
        (<number>)+

      <token: number> \s*\d+
    }xms;
};

while (my $input = <>) {
    if ($input =~ m{$rbb}) {
        say("matches: <$&>");
        say Dumper \%/;
    }
}
```

Obsérvese el uso explícito de espacios `\s*\d+` en la definición de `number`.

Sigue un ejemplo de ejecución:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5_10_1 numbers3.pl
1 2 3 4
matches: <1 2 3 4>
$VAR1 = {
          '' => '1 2 3 4',
          'numbers' => {
                         '' => '1 2 3 4',
                         'number' => ' 4'
                       }
        };
```

Si se eliminan los blancos de la definición de `number`:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n numbers.pl
use strict;
use warnings;
use 5.010;
use Data::Dumper;

my $rbb = do {
    use Regexp::Grammars;

```

```
    qr{
      <numbers>

      <rule: numbers>
        (<number>)+

      <token: number> \d+
    }xms;
};

while (my $input = <>) {
    if ($input =~ m{$rbb}) {
        say("matches: <$&>");
        say Dumper \%/;
    }
}
```

se obtiene una conducta que puede sorprender:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 numbers.pl
12 34 56
matches: <12>
$VAR1 = {
          '' => '12',
          'numbers' => {
                         '' => '12',
                         'number' => '12'
                       }
        };
```

La explicación está en la documentación: véase la sección Grammar Syntax:

> <rule: IDENTIFIER>
> *Define a rule whose name is specified by the supplied identifier.*
>
> *Everything following the* <rule:...> *directive (up to the next* <rule:...> *or* <token:...> *directive) is treated as part of the rule being defined.*
>
> Any whitespace in the rule is replaced by a call to the <.ws> *subrule (which defaults to matching* \s*, *but may be explicitly redefined).*

También podríamos haber resuelto el problema introduciendo un blanco explícito dentro del cierre positivo:

```
<rule: numbers>
  (<number> )+

<token: number> \d+
```

Una Solución al problema de recordar los resultados de una lista: El uso de brackets

> *Usually, that's not the desired outcome, so* Regexp::Grammars *provides another mechanism by which to call a subrule; one that saves all repetitions of its results.*
>
> *A regular subrule call consists of the rule's name surrounded by angle brackets. If, instead, you surround the rule's name with* <[...]> *(angle and square brackets) like so:*

```
<rule: file_sequence>
    <[file]>+
```

then the rule is invoked in exactly the same way, but the result of that submatch is pushed onto an array nested inside the appropriate result-hash entry. In other words, if the above example matched the same `foo.pl bar.py baz.php` *string, the result-hash would contain:*

```
file_sequence {
    ""   => 'foo.pl bar.py baz.php',
    file => [ 'foo.pl', 'bar.py', 'baz.php' ],
}
```

Teniendo en cuenta lo dicho anteriormente sobre los blancos dentro de los cuantificadores, es necesario introducir blancos dentro del operador de repetición:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n numbers4.pl
use strict;
use warnings;
use 5.010;
use Data::Dumper;

my $rbb = do {
    use Regexp::Grammars;

    qr{
      <numbers>

      <rule: numbers>
        (?:  <[number]> )+

      <token: number> \d+
    }xms;
};

while (my $input = <>) {
    if ($input =~ m{$rbb}) {
        say("matches: <$&>");
        say Dumper \%/;
    }
}
```

Al ejecutar este programa obtenemos:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5_10_1 numbers4.pl
1 2 3 4
matches: <1 2 3 4
>
$VAR1 = {
          '' => '1 2 3 4
',
          'numbers' => {
                         '' => '1 2 3 4
',
                         'number' => [ '1', '2', '3', '4' ]
                       }
        };
```

Otra forma de resolver las colisiones de nombres: salvarlos en una lista

> *This* listifying subrule call *can also be useful for non-repeated subrule calls, if the same subrule is invoked in several places in a grammar. For example if a cmdline option could be given either one or two values, you might parse it:*
>
> ```
> <rule: size_option>
>     -size <[size]> (?: x <[size]> )?
> ```
>
> *The result-hash entry for* `size` *would then always contain an array, with either one or two elements, depending on the input being parsed.*

Sigue un ejemplo:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n sizes.pl
use strict;
use warnings;
use 5.010;
use Data::Dumper;

my $rbb = do {
    use Regexp::Grammars;

    qr{
      <command>

      <rule: command> ls <size_option>

      <rule: size_option>
          -size <[size]> (?: x <[size]> )?

      <token: size> \d+
    }x;
};

while (my $input = <>) {
    while ($input =~ m{$rbb}g) {
        say("matches: <$&>");
        say Dumper \%/;
    }
}
```

Veamos su comportamiento con diferentes entradas:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 sizes.pl
ls -size 4
matches: <ls -size 4
>
$VAR1 = {
          '' => 'ls -size 4
',
          'command' => {
                         'size_option' => {
                                            '' => '-size 4
',
```

```
                                            'size' => [ '4' ]
                                          },
                         '' => 'ls -size 4
'
                       }
        };

ls -size 2x8
matches: <ls -size 2x8
>
$VAR1 = {
          '' => 'ls -size 2x8
',
          'command' => {
                         'size_option' => {
                                            '' => '-size 2x8
',
                                            'size' => [ '2', '8' ]
                                          },
                         '' => 'ls -size 2x8
'
                       }
        };
```

Aliasing de listas

> *Listifying subrules can also be given aliases, just like ordinary subrules. The alias is always specified inside the square brackets:*
>
> ```
> <rule: size_option>
>     -size <[size=pos_integer]> (?: x <[size=pos_integer]> )?
> ```
>
> *Here, the sizes are parsed using the* `pos_integer` *rule, but saved in the result-hash in an array under the key* `size`*.*

Sigue un ejemplo:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n aliasedsizes.pl
use strict;
use warnings;
use 5.010;
use Data::Dumper;

my $rbb = do {
    use Regexp::Grammars;

    qr{
      <command>

      <rule: command> ls <size_option>

      <rule: size_option>
          -size <[size=int]> (?: x <[size=int]> )?

```

```
      <token: int> \d+
    }x;
};

while (my $input = <>) {
    while ($input =~ m{$rbb}g) {
        say("matches: <$&>");
        say Dumper \%/;
    }
}
```

Veamos el resultado de una ejecución:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 aliasedsizes.pl
ls -size 2x4
matches: <ls -size 2x4
>
$VAR1 = {
          '' => 'ls -size 2x4
',
          'command' => {
                         'size_option' => {
                                            '' => '-size 2x4
',
                                            'size' => [
                                                        '2',
                                                        '4'
                                                      ]
                                          },
                         '' => 'ls -size 2x4
'
                       }
        };
```

Caveat: Cierres y Warnings

En este ejemplo aparece `<number>+` sin corchetes ni paréntesis:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n numbers5.pl
use strict;
use warnings;
use 5.010;
use Data::Dumper;

my $rbb = do {
    use Regexp::Grammars;

    qr{
      <numbers>

      <rule: numbers>
        <number>+

      <token: number> \d+
    }xms;
```

```
};

while (my $input = <>) {
    if ($input =~ m{$rbb}) {
        say("matches: <$&>");
        say Dumper \%/;
    }
}
```

Este programa produce un mensaje de advertencia:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 numbers5.pl
  warn | Repeated subrule <number>+ will only capture its final match
       | (Did you mean <[number]>+ instead?)
       |
```

Si se quiere evitar el mensaje y se está dispuesto a asumir la pérdida de los valores asociados con los elementos de la lista se deberán poner el operando entre paréntesis (con o sin memoria).

Esto es lo que dice la documentación sobre este warning:

> `Repeated subrule <rule> will only capture its final match`
>
> *You specified a subrule call with a repetition qualifier, such as:*
>
> ```
> <ListElem>*
> ```
>
> *or:*
>
> ```
> <ListElem>+
> ```
>
> *Because each subrule call saves its result in a hash entry of the same name, each repeated match will overwrite the previous ones, so only the last match will ultimately be saved. If you want to save all the matches, you need to tell* `Regexp::Grammars` *to save the sequence of results as a nested array within the hash entry, like so:*
>
> ```
> <[ListElem]>*
> ```
>
> *or:*
>
> ```
> <[ListElem]>+
> ```
>
> *If you really did intend to throw away every result but the final one, you can silence the warning by placing the subrule call inside any kind of parentheses. For example:*
>
> ```
> (<ListElem>)*
> ```
>
> *or:*
>
> ```
> (?: <ListElem> )+
> ```

3.10.5. Pseudo sub-reglas

Subpatrones

> *Aliases can also be given to standard Perl subpatterns, as well as to code blocks within a regex. The syntax for subpatterns is:*
>
> ```
> <ALIAS= (SUBPATTERN) >
> ```
>
> *In other words, the syntax is exactly like an aliased subrule call, except that the rule name is replaced with a set of parentheses containing the subpattern. Any parentheses–capturing or non-capturing–will do.*
>
> *The effect of aliasing a standard subpattern is to cause whatever that subpattern matches to be saved in the result-hash, using the alias as its key. For example:*
>
> ```
> <rule: file_command>
>
>     <cmd=(mv|cp|ln)>  <from=file>  <to=file>
> ```
>
> *Here, the* `<cmd=(mv|cp|ln)>` *is treated exactly like a regular* `(mv|cp|ln)`*, but whatever substring it matches is saved in the result-hash under the key* `'cmd'`*.*

Sigue un ejemplo:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n subpattern.pl
use strict;
use warnings;
use 5.010;
use Data::Dumper;

my $rbb = do {
    use Regexp::Grammars;

    qr{
        <file_command>

        <rule: file_command>

        <cmd=(mv|cp|ln)>  <from=([\w./]+)>  <to=([\w./]+)>

    }x;
};

while (my $input = <>) {
    while ($input =~ m{$rbb}g) {
        say("matches: <$&>");
        say Dumper \%/;
    }
}
```

y una ejecución:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 subpattern.pl
mv a b
matches: <mv a b>
$VAR1 = {
```

```
          '' => 'mv a b',
          'file_command' => {
                              '' => 'mv a b',
                              'to' => 'b',
                              'cmd' => 'mv',
                              'from' => 'a'
                            }
        };

cp c d
matches: <cp c d>
$VAR1 = {
          '' => 'cp c d',
          'file_command' => {
                              '' => 'cp c d',
                              'to' => 'd',
                              'cmd' => 'cp',
                              'from' => 'c'
                            }
        }
```

Bloques de código

The syntax for aliasing code blocks is:

```
<ALIAS= (?{ your($code->here) }) >
```

Note, however, that the code block must be specified in the standard Perl 5.10 regex notation: `(?{...})`. *A common mistake is to write:*

```
<ALIAS= { your($code->here } >
```

instead, which will attempt to interpolate `$code` *before the regex is even compiled, as such variables are only* protected *from interpolation inside a* `(?{...})`.

When correctly specified, this construct executes the code in the block and saves the result of that execution in the result-hash, using the alias as its key. Aliased code blocks are useful for adding semantic information based on which branch of a rule is executed. For example, consider the `copy_cmd` *alternatives shown earlier:*

```
<rule: copy_cmd>
    copy <from=file>        <to=file>
  | dup    <to=file>  as  <from=file>
  |      <from=file>  ->    <to=file>
  |        <to=file>  <-  <from=file>
```

Using aliased code blocks, you could add an extra field to the result- hash to describe which form of the command was detected, like so:

```
<rule: copy_cmd>
    copy <from=file>        <to=file>  <type=(?{ 'std' })>
  | dup    <to=file>  as  <from=file>  <type=(?{ 'rev' })>
  |      <from=file>  ->    <to=file>  <type=(?{ 'fwd' })>
  |        <to=file>  <-  <from=file>  <type=(?{ 'bwd' })>
```

Now, if the rule matched, the result-hash would contain something like:

```
        copy_cmd => {
            from => 'oldfile',
              to => 'newfile',
            type => 'fwd',
        }
```

El siguiente ejemplo ilustra lo dicho en la documentación. En la línea 15 hemos introducido una regla para el control de errores[8]:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n aliasedcodeblock2.pl
 1  use strict;
 2  use warnings;
 3  use 5.010;
 4  use Data::Dumper;
 5
 6  my $rbb = do {
 7      use Regexp::Grammars;
 8      qr{
 9        <copy_cmd>
10
11        <rule: copy_cmd>
12              copy (<from=file>) (<to=file>) <type=(?{ 'std' })>
13          |   <from=file> ->   <to=file> <type=(?{ 'fwd' })>
14          |   <to=file>   <- <from=file> <type=(?{ 'bwd' })>
15          |   .+ (?{ die "Syntax error!\n" })
16
17        <token: file> [\w./\\]+
18      }x;
19  };
20
21  while (my $input = <>) {
22      while ($input =~ m{$rbb}g) {
23          say("matches: <$&>");
24          say Dumper \%/;
25      }
26  }
```

La ejecución muestra el comportamiento del programa con tres entradas válidas y una errónea:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 aliasedcodeblock2.pl
copy a b
matches: <copy a b
>
$VAR1 = {
          '' => 'copy a b
',
          'copy_cmd' => {
                          '' => 'copy a b
',
                          'to' => 'b',
                          'from' => 'a',
                          'type' => 'std'
                        }
```

[8]Versión de `Grammar.pm` obtenida por email con las correcciones de Damian

```
        };

b <- a
matches: <b <- a
>
$VAR1 = {
          '' => 'b <- a
',
          'copy_cmd' => {
                          '' => 'b <- a
',
                          'to' => 'b',
                          'from' => 'a',
                          'type' => 'bwd'
                        }
        };

a -> b
matches: <a -> b
>
$VAR1 = {
          '' => 'a -> b
',
          'copy_cmd' => {
                          '' => 'a -> b
',
                          'to' => 'b',
                          'from' => 'a',
                          'type' => 'fwd'
                        }
        };

cp a b
Syntax error!
```

Pseudo subreglas y depuración

Note that, in addition to the semantics described above, aliased subpatterns and code blocks also become visible to `Regexp::Grammars` *integrated debugger (see Debugging).*

3.10.6. Llamadas a subreglas desmemoriadas

By default, every subrule call saves its result into the result-hash, either under its own name, or under an alias.

However, sometimes you may want to refactor some literal part of a rule into one or more subrules, without having those submatches added to the result-hash. The syntax for calling a subrule, but ignoring its return value is:

```
<.SUBRULE>
```

(which is stolen directly from Perl 6).

For example, you may prefer to rewrite a rule such as:

```
<rule: paren_pair>
```

```
\(
    (?: <escape> | <paren_pair> | <brace_pair> | [^()] )*
\)
```

without any literal matching, like so:

```
<rule: paren_pair>

    <.left_paren>
        (?: <escape> | <paren_pair> | <brace_pair> | <.non_paren> )*
    <.right_paren>

<token: left_paren>   \(
<token: right_paren>  \)
<token: non_paren>    [^()]
```

Moreover, as the individual components inside the parentheses probably aren't being captured for any useful purpose either, you could further optimize that to:

```
<rule: paren_pair>

    <.left_paren>
        (?: <.escape> | <.paren_pair> | <.brace_pair> | <.non_paren> )*
    <.right_paren>
```

Note that you can also use the dot modifier on an aliased subpattern:

```
<.Alias= (SUBPATTERN) >
```

This seemingly contradictory behaviour (of giving a subpattern a name, then deliberately ignoring that name) actually does make sense in one situation. Providing the alias makes the subpattern visible to the debugger, while using the dot stops it from affecting the result-hash. See Debugging non-grammars *for an example of this usage.*

Ejemplo: Números entre comas

Por ejemplo, queremos reconocer listas de números separados por comas. Supongamos también que queremos darle un nombre a la expresión regular de separación. Quizá, aunque no es el caso, porque la expresión regular de separación sea suficientemente compleja. Si no usamos la notación *punto* la coma aparecerá en la estructura:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n numberscomma.pl
use strict;
use warnings;
use 5.010;
use Data::Dumper;
$Data::Dumper::Indent = 1;

my $rbb = do {
    use Regexp::Grammars;

    qr{
      <numbers>

      <objrule: numbers>
```

```
        <[number]> (<comma> <[number]>)*

      <objtoken: number> \s*\d+
      <token: comma>  \s*,
    }xms;
};

while (my $input = <>) {
    if ($input =~ m{$rbb}) {
        say("matches: <$&>");
        say Dumper \%/;
    }
}
```

En efecto, aparece la clave `comma`:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 numberscomma.pl
2, 3, 4
matches: <2, 3, 4>
$VAR1 = {
  '' => '2, 3, 4',
  'numbers' => bless( {
    '' => '2, 3, 4',
    'number' => [
      bless( { '' => '2' }, 'number' ),
      bless( { '' => '3' }, 'number' ),
      bless( { '' => '4' }, 'number' )
    ],
    'comma' => ','
  }, 'numbers' )
};
```

Si cambiamos la llamada a la regla `<comma>` por `<.comma>`

```
pl@nereida:~/Lregexpgrammars/demo$ diff numberscomma.pl numberscomma2.pl
14c14
<         <[number]> (<comma> <[number]>)*
---
>         <[number]> (<.comma> <[number]>)*
```

eliminamos la aparición de la innecesaria clave:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 numberscomma2.pl
2, 3, 4
matches: <2, 3, 4>
$VAR1 = {
  '' => '2, 3, 4',
  'numbers' => bless( {
    '' => '2, 3, 4',
    'number' => [
      bless( { '' => '2' }, 'number' ),
      bless( { '' => '3' }, 'number' ),
      bless( { '' => '4' }, 'number' )
    ]
  }, 'numbers' )
};
```

3.10.7. Destilación del resultado

Destilación manual

Regexp::Grammars *also offers full manual control over the distillation process. If you use the reserved word* MATCH *as the alias for a subrule call:*

```
<MATCH=filename>
```

or a subpattern match:

```
<MATCH=( \w+ )>
```

or a code block:

```
<MATCH=(?{ 42 })>
```

then the current rule will treat the return value of that subrule, pattern, or code block as its complete result, and return that value instead of the usual result-hash it constructs. This is the case even if the result has other entries that would normally also be returned.

For example, in a rule like:

```
<rule: term>
      <MATCH=literal>
    | <left_paren> <MATCH=expr> <right_paren>
```

The use of MATCH *aliases causes the rule to return either whatever* <literal> *returns, or whatever* <expr> *returns (provided it's between left and right parentheses).*

Note that, in this second case, even though <left_paren> *and* <right_paren> *are captured to the result-hash, they are not returned, because the* MATCH *alias overrides the normal* return the result-hash *semantics and returns only what its associated subrule (i.e.* <expr>*) produces.*

El siguiente ejemplo ilustra el uso del alias MATCH:

```
$ cat -n demo_calc.pl
#!/usr/local/lib/perl/5.10.1/bin/perl5.10.1
use v5.10;
use warnings;

my $calculator = do{
    use Regexp::Grammars;
    qr{
        <Answer>

        <rule: Answer>
            <X=Mult> <Op=([+-])> <Y=Answer>
          | <MATCH=Mult>

        <rule: Mult>
            <X=Pow> <Op=([*/%])> <Y=Mult>
          | <MATCH=Pow>

        <rule: Pow>
            <X=Term> <Op=(\^)> <Y=Pow>
          | <MATCH=Term>
```

```

        <rule: Term>
               <MATCH=Literal>
          | \( <MATCH=Answer> \)

        <token: Literal>
            <MATCH=( [+-]? \d++ (?: \. \d++ )?+ )>
    }xms
};

while (my $input = <>) {
    if ($input =~ $calculator) {
        use Data::Dumper 'Dumper';
        warn Dumper \%/;
    }
}
```

Veamos una ejecución:

```
$ ./demo_calc.pl
2+3*5
$VAR1 = {
          '' => '2+3*5',
          'Answer' => {
                        '' => '2+3*5',
                        'Op' => '+',
                        'X' => '2',
                        'Y' => {
                                 '' => '3*5',
                                 'Op' => '*',
                                 'X' => '3',
                                 'Y' => '5'
                               }
                      }
        };
4-5-2
$VAR1 = {
          '' => '4-5-2',
          'Answer' => {
                        '' => '4-5-2',
                        'Op' => '-',
                        'X' => '4',
                        'Y' => {
                                 '' => '5-2',
                                 'Op' => '-',
                                 'X' => '5',
                                 'Y' => '2'
                               }
                      }
        };
```

Obsérvese como el árbol construido para la expresión `4-5-2` se hunde a derechas dando lugar a una jerarquía errónea. Para arreglar el problema sería necesario eliminar la recursividad por la izquierda en las reglas correspondientes.

Destilación en el programa

It's also possible to control what a rule returns from within a code block. `Regexp::Grammars` *provides a set of reserved variables that give direct access to the result-hash.*

The result-hash itself can be accessed as `%MATCH` *within any code block inside a rule. For example:*

```
<rule: sum>
    <X=product> \+ <Y=product>
        <MATCH=(?{ $MATCH{X} + $MATCH{Y} })>
```

Here, the rule matches a product (aliased `'X'` *in the result-hash), then a literal* `'+'`*, then another product (aliased to* `'Y'` *in the result-hash). The rule then executes the code block, which accesses the two saved values (as* `$MATCH{X}` *and* `$MATCH{Y}`*), adding them together. Because the block is itself aliased to* `MATCH`*, the sum produced by the block becomes the (only) result of the rule.*

It is also possible to set the rule result from within a code block (instead of aliasing it). The special override *return value is represented by the special variable* `$MATCH`*. So the previous example could be rewritten:*

```
<rule: sum>
    <X=product> \+ <Y=product>
        (?{ $MATCH = $MATCH{X} + $MATCH{Y} })
```

Both forms are identical in effect. Any assignment to `$MATCH` *overrides the normal* return all subrule results *behaviour.*

Assigning to `$MATCH` *directly is particularly handy if the result may not always be* distillable*, for example:*

```
<rule: sum>
    <X=product> \+ <Y=product>
        (?{ if (!ref $MATCH{X} && !ref $MATCH{Y}) {
                # Reduce to sum, if both terms are simple scalars...
                $MATCH = $MATCH{X} + $MATCH{Y};
            }
            else {
                # Return full syntax tree for non-simple case...
                $MATCH{op} = '+';
            }
        })
```

Note that you can also partially override the subrule return behaviour. Normally, the subrule returns the complete text it matched under the empty key *of its result-hash. That is, of course,* `$MATCH{""}`*, so you can override just that behaviour by directly assigning to that entry.*

For example, if you have a rule that matches key/value pairs from a configuration file, you might prefer that any trailing comments not be included in the matched text entry of the rule's result-hash. You could hide such comments like so:

```
<rule: config_line>
    <key> : <value>  <comment>?
        (?{
            # Edit trailing comments out of "matched text" entry...
            $MATCH = "$MATCH{key} : $MATCH{value}";
        })
```

Some more examples of the uses of `$MATCH`*:*

```
<rule: FuncDecl>
  # Keyword  Name               Keep return the name (as a string)...
    func     <Identifier> ;     (?{ $MATCH = $MATCH{'Identifier'} })


<rule: NumList>
  # Numbers in square brackets...
    \[
        ( \d+ (?: , \d+)* )
    \]

  # Return only the numbers...
    (?{ $MATCH = $CAPTURE })


<token: Cmd>
  # Match standard variants then standardize the keyword...
    (?: mv | move | rename )      (?{ $MATCH = 'mv'; })
```

> `$CAPTURE` *and* `$CONTEXT` *are both aliases for the built-in read-only* `$^N` *variable, which always contains the substring matched by the nearest preceding* `(...)` *capture.* `$^N` *still works perfectly well, but these are provided to improve the readability of code blocks and error messages respectively.*

El siguiente código implementa una calculadora usando destilación en el código:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n demo_calc_inline.pl
use v5.10;
use warnings;

my $calculator = do{
    use Regexp::Grammars;
    qr{
        <Answer>

        <rule: Answer>
            <X=Mult> \+ <Y=Answer>
                (?{ $MATCH = $MATCH{X} + $MATCH{Y}; })
          | <X=Mult> - <Y=Answer>
                (?{ $MATCH = $MATCH{X} - $MATCH{Y}; })
          | <MATCH=Mult>

        <rule: Mult>
            <X=Pow> \* <Y=Mult>
                (?{ $MATCH = $MATCH{X} * $MATCH{Y}; })
          | <X=Pow>  / <Y=Mult>
                (?{ $MATCH = $MATCH{X} / $MATCH{Y}; })
          | <X=Pow>  % <Y=Mult>
                (?{ $MATCH = $MATCH{X} % $MATCH{Y}; })
          | <MATCH=Pow>

        <rule: Pow>
```

```
            <X=Term> \^ <Y=Pow>
                (?{ $MATCH = $MATCH{X} ** $MATCH{Y}; })
          | <MATCH=Term>

        <rule: Term>
               <MATCH=Literal>
          | \( <MATCH=Answer> \)

        <token: Literal>
            <MATCH=( [+-]? \d++ (?: \. \d++ )?+ )>
    }xms
};

while (my $input = <>) {
    if ($input =~ $calculator) {
        say '--> ', $/{Answer};
    }
}
```

Ejercicio 3.10.1. *Cual es la salida del programa anterior para las entradas:*

- `4-2-2`
- `8/4/2`
- `2^2^3`

3.10.8. Llamadas privadas a subreglas y subreglas privadas

If a rule name (or an alias) begins with an underscore:

```
  <_RULENAME>       <_ALIAS=RULENAME>
 <[_RULENAME]>     <[_ALIAS=RULENAME]>
```

then matching proceeds as normal, and any result that is returned is stored in the current result-hash in the usual way.

However, when any rule finishes (and just before it returns) it first filters its result-hash, removing any entries whose keys begin with an underscore. This means that any subrule with an underscored name (or with an underscored alias) remembers its result, but only until the end of the current rule. Its results are effectively private to the current rule.

This is especially useful in conjunction with result distillation.

3.10.9. Mas sobre listas

Reconocimiento manual de listas

Analizando listas manualmente

El siguiente ejemplo muestra como construir un reconocedor de listas (posiblemente vacías) de números:

```
casiano@millo:~/Lregexp-grammar-examples$ cat -n simple_list.pl
#!/soft/perl5lib/bin/perl5.10.1
use v5.10;

use Regexp::Grammars;

```

```
my $list = qr{
    <List>

    <rule: List>
         <digit> <List>
       | # empty

    <rule: digit>
        <MATCH=(\d+)>

}xms;

while (my $input = <>) {
    chomp $input;
    if ($input =~ $list) {
        use Data::Dumper 'Dumper';
        warn Dumper \%/;
    }
    else {
      warn "Does not match\n"
    }
}
```

Sigue una ejecución:

```
casiano@millo:~/Lregexp-grammar-examples$ ./simple_list.pl
2 3 4
$VAR1 = {
          '' => '2 3 4',
          'List' => {
                      '' => '2 3 4',
                      'digit' => '2'
                      'List' => {
                                  '' => '3 4',
                                  'digit' => '3'
                                  'List' => {
                                              '' => '4',
                                              'digit' => '4'
                                              'List' => '',
                                            },
                                },
                    }
        };
```

Influencia del orden en el lenguaje reconocido

Tenga en cuenta que el orden de las reglas influye en el lenguaje reconocido. Véase lo que ocurre si cambiamos en el ejemplo anterior el orden de las reglas:

```
casiano@millo:~/Lregexp-grammar-examples$ cat -n simple_list_empty_first.pl
     1  #!/soft/perl5lib/bin/perl5.10.1
     2  use v5.10;
     3
     4  use Regexp::Grammars;
     5
     6  my $list = qr{
```

```
    <List>

    <rule: List>
         # empty
       | <digit> <List>

    <rule: digit>
        <MATCH=(\d+)>

}xms;

while (my $input = <>) {
    chomp $input;
    if ($input =~ $list) {
        use Data::Dumper 'Dumper';
        warn Dumper \%/;
    }
    else {
      warn "Does not match\n"
    }
}
```

Al ejecutar se obtiene:

```
casiano@millo:~/Lregexp-grammar-examples$ ./simple_list_empty_first.pl
2 3 4
$VAR1 = {
          '' => '',
          'List' => ''
        };
```

Por supuesto basta poner anclas en el patrón a buscar para forzar a que se reconozca la lista completa:

```
pl@nereida:~/Lregexpgrammars/demo$ diff simple_list_empty_first.pl simple_list_empty_first_wit
7c7
<     <List>
---
>     ^<List>$
```

En efecto, la nueva versión reconoce la lista:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 simple_list_empty_first_with_anchors.pl
2 3 4
$VAR1 = {
          '' => '2 3 4',
          'List' => {
                        'List' => {
                                      'List' => {
                                                    'List' => '',
                                                    '' => '4',
                                                    'digit' => '4'
                                                  },
                                      '' => '3 4',
                                      'digit' => '3'
                                    },
```

```
                        ’’ => ’2 3 4’,
                        ’digit’ => ’2’
                      }
        };
```

Si se quiere mantener la producción vacía en primer lugar pero forzar el reconocimiento de la lista completa, se puede hacer uso de un lookahead negativo:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n simple_list_empty_first_with_lookahead.pl
#!/soft/perl5lib/bin/perl5.10.1
use v5.10;

use strict;
use Regexp::Grammars;

my $list = qr{
    <List>

    <rule: List>
         (?! <digit> ) # still empty production
       | <digit> <List>

    <rule: digit>
        <MATCH=(\d+)>

}xms;

while (my $input = <>) {
    chomp $input;
    if ($input =~ $list) {
        use Data::Dumper ’Dumper’;
        warn Dumper \%/;
    }
    else {
      warn "Does not match\n"
    }
}
```

Así, sólo se reducirá por la regla vacía si el siguiente token no es un número. Sigue un ejemplo de ejecución:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 simple_list_empty_first_with_lookahead.pl
2 3 4
$VAR1 = {
          ’’ => ’2 3 4’,
          ’List’ => {
                      ’List’ => {
                                  ’List’ => {
                                              ’List’ => ’’,
                                              ’’ => ’4’,
                                              ’digit’ => ’4’
                                            },
                                  ’’ => ’3 4’,
                                  ’digit’ => ’3’
                                },
```

```
                   '' => '2 3 4',
                   'digit' => '2'
                 }
        };
```

Aplanamiento manual de listas

¿Cómo podemos hacer que la estructura retornada por el reconocedor sea una lista?. Podemos añadir acciones como sigue:

```
casiano@millo:~/Lregexp-grammar-examples$ cat -n simple_list_action.pl
#!/soft/perl5lib/bin/perl5.10.1
use v5.10;

use Regexp::Grammars;

my $list = qr{
    <List>

    <rule: List>
         <digit> <X=List> <MATCH= (?{ unshift @{$MATCH{X}}, $MATCH{digit}; $MATCH{X} }
       | # empty
         <MATCH= (?{ [] })>

    <rule: digit>
        <MATCH=(\d+)>

}xms;

while (my $input = <>) {
    chomp $input;
    if ($input =~ $list) {
        use Data::Dumper 'Dumper';
        warn Dumper \%/;
    }
    else {
      warn "Does not match\n"
    }
}
```

Al ejecutarse este programa produce una salida como:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 simple_list_action.pl
2 3 4
$VAR1 = {
          '' => '2 3 4',
          'List' => [ '2', '3', '4' ]
        };
```

Los operadores de repetición

Los operadores de repetición como `*`, `+`, etc. permiten simplificar el análisis de lenguajes de listas:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n simple_list_star.pl
#!/soft/perl5lib/bin/perl5.10.1
```

```
use v5.10;

use Regexp::Grammars;

my $list = qr{
    <List>

    <rule: List>
        (?: <[digit]>)*

    <rule: digit>
        <MATCH=(\d+)>

}xms;

while (my $input = <>) {
    chomp $input;
    if ($input =~ $list) {
        use Data::Dumper 'Dumper';
        warn Dumper \%/;
    }
    else {
      warn "Does not match\n"
    }
}
```

Los corchetes alrededor de `digit` hacen que el valor asociado con el patrón sea la lista de números. Si no los ponemos el valor asociado sería el último valor de la lista.

Listas separadas por Algo

> *One of the commonest tasks in text parsing is to match a list of unspecified length, in which items are separated by a fixed token. Things like:*
>
> ```
> 1, 2, 3 , 4 ,13, 91        # Numbers separated by commas and spaces
>
> g-c-a-g-t-t-a-c-a          # Bases separated by dashes
>
> /usr/local/bin             # Names separated by directory markers
>
> /usr:/usr/local:bin        # Directories separated by colons
> ```
>
> *The usual construct required to parse these kinds of structures is either:*
>
> ```
> <rule: list>
>
>     <item> <separator> <list               # recursive definition
>   | <item>                                 # base case
> ```
>
> *Or, more efficiently, but less prettily:*
>
> ```
> <rule: list>
>
>     <[item]> (?: <separator> <[item]> )*   # iterative definition
> ```

Because this is such a common requirement, `Regexp::Grammars` *provides a cleaner way to specify the iterative version. The syntax is taken from Perl 6:*

```
<rule: list>

    <[item]> ** <separator>              # iterative definition
```

This is a repetition specifier on the first subrule (hence the use of `**` *as the marker, to reflect the repetitive behaviour of* `*`*). However, the number of repetitions is controlled by the second subrule: the first subrule will be repeatedly matched for as long as the second subrule matches immediately after it.*

So, for example, you can match a sequence of numbers separated by commas with:

```
<[number]> ** <comma>

<token: number>  \d+
<token: comma>   \s* , \s*
```

Note that it's important to use the `<[...]>` *form for the items being matched, so that all of them are saved in the result hash. You can also save all the separators (if that's important):*

```
<[number]> ** <[comma]>
```

The repeated item must be specified as a subrule call fo some kind, but the separators may be specified either as a subrule or a bracketed pattern. For example:

```
<[number]> ** ( , )
```

The separator must always be specified in matched delimiters of some kind: either matching `<...>` *or matching* `(...)`*. A common error is to write:*

```
<[number]> ** ,
```

You can also use a pattern as the item matcher, but it must be aliased into a subrule:

```
<[item=(\d+)]> ** ( , )
```

Ejemplo: Listas de números separados por comas

Veamos un ejemplo sencillo:

```
casiano@millo:~/src/perl/regexp-grammar-examples$ cat -n demo_list.pl
#!/soft/perl5lib/bin/perl5.10.1
use v5.10;

use Regexp::Grammars;

my $list_nonempty = qr{
    <List>

    <rule: List>
        \(  <[Value]> ** (,)  \)

    <token: Value>
        \d+
```

```
}xms;

my $list_empty = qr{
    <List>

    <rule: List>
        \(  (?: <[Value]> ** <_Sep=(,)> )?  \)

    <token: Value>
        \d+
}xms;

use Smart::Comments;


while (my $input = <>) {
    my $input2 = $input;
    if ($input =~ $list_nonempty) {
        ### nonempty: $/{List}
    }
    if ($input2 =~ $list_empty) {
        ### empty: $/{List}
    }
}
```

Sigue un ejemplo de ejecución:

```
casiano@millo:~/src/perl/regexp-grammar-examples$ ./demo_list.pl
(3,4,5)

### nonempty: {
###            '' => '(3,4,5)',
###            Value => [
###                       '3',
###                       '4',
###                       '5'
###                     ]
###          }

### empty: {
###         '' => '(3,4,5)',
###         Value => [
###                    '3',
###                    '4',
###                    '5'
###                  ]
###       }
()

### empty: '()'
```

Ejemplo: AST para las expresiones aritméticas

Las expresiones aritméticas puede definirse como una jerarquía de listas como sigue:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n calcaslist.pl
use strict;
use warnings;
use 5.010;
use Data::Dumper;
$Data::Dumper::Indent = 1;

my $rbb = do {
    use Regexp::Grammars;

    qr{
      \A<expr>\z

      <objrule: expr>      <[operands=term]> ** <[operators=addop]>

      <objrule: term>      <[operands=uneg]> ** <[operators=mulop]>

      <objrule: uneg>      <[operators=minus]>* <[operands=power]>

      <objrule: power>     <[operands=factorial]> ** <[operators=powerop]>

      <objrule: factorial> <[operands=factor]>  <[operators=(!)]>*

      <objrule: factor>    <val=([+-]?\d+(?:\.\d*)?)>
                         | \( <MATCH=expr> \)

      <token: addop>        [+-]

      <token: mulop>        [*/]

      <token: powerop>      \*\*|\^

      <token: minus>        - <MATCH=(?{ 'NEG' })>

    }x;
};

while (my $input = <>) {
    chomp($input);
    if ($input =~ m{$rbb}) {
        my $tree = $/{expr};
        say Dumper $tree;

    }
    else {
        say("does not match");
    }
}
```

Obsérvese el árbol generado para la expresión 4-2-2:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 calcaslist.pl
4-2-2
$VAR1 = bless( {
```

```
'operands' => [
  bless( {
    'operands' => [
      bless( {
        'operands' => [
          bless( {
            'operands' => [
              bless( {
                'operands' => [
                  bless( { '' => '4', 'val' => '4' }, 'factor' )
                ],
                '' => '4'
              }, 'factorial' )
            ],
            '' => '4'
          }, 'power' )
        ],
        '' => '4'
      }, 'uneg' )
    ],
    '' => '4'
  }, 'term' ),
  bless( {
    'operands' => [
      bless( {
        'operands' => [
          bless( {
            'operands' => [
              bless( {
                'operands' => [
                  bless( { '' => '2', 'val' => '2' }, 'factor' )
                ],
                '' => '2'
              }, 'factorial' )
            ],
            '' => '2'
          }, 'power' )
        ],
        '' => '2'
      }, 'uneg' )
    ],
    '' => '2'
  }, 'term' ),
  bless( {
    'operands' => [
      bless( {
        'operands' => [
          bless( {
            'operands' => [
              bless( {
                'operands' => [
                  bless( { '' => '2', 'val' => '2' }, 'factor' )
                ],
```

```
                  '' => '2'
                }, 'factorial' )
              ],
              '' => '2'
            }, 'power' )
          ],
          '' => '2'
        }, 'uneg' )
      ],
      '' => '2'
    }, 'term' )
  ],
  '' => '4-2-2',
  'operators' => [
    '-',
    '-'
  ]
}, 'expr' );
```

3.10.10. La directiva `require`

La directiva `require` es similar en su funcionamiento al paréntesis 5.10 `(??{ Código Perl })` el cuál hace que el `Código Perl` sea evaluado durante el tiempo de matching. El resultado de la evaluación se trata como una expresión regular con la que deberá casarse. (véase la sección 3.2.9 para mas detalles).

La sintáxis de la directiva `<require:>` es

```
<require: (?{ CODE }) >
```

> *The code block is executed and if its final value is true, matching continues from the same position. If the block's final value is false, the match fails at that point and starts backtracking.*
>
> *The* `<require:...>` *directive is useful for testing conditions that it's not easy (or even possible) to check within the syntax of the the regex itself. For example:*
>
> ```
> <rule: IPV4_Octet_Decimal>
>     # Up three digits...
>     <MATCH= ( \d{1,3}+ )>
>
>     # ...but less that 256...
>     <require: (?{ $MATCH <= 255 })>
> ```
>
> *A require expects a regex codeblock as its argument and succeeds if the final value of that codeblock is true. If the final value is false, the directive fails and the rule starts backtracking.*
>
> *Note, in this example that the digits are matched with* `\d{1,3}+` . The trailing `+` *prevents the* `{1,3}` *repetition from backtracking to a smaller number of digits if the* `<require:...>` *fails.*

El programa `demo_IP4.pl` ilustra el uso de la directiva:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n ./demo_IP4.pl
 1  #!/usr//bin/env perl5.10.1
 2  use v5.10;
 3  use warnings;
```

```

use Regexp::Grammars;

my $grammar = qr{
    \A <IP4_addr> \Z

    <token: quad>
        <MATCH=(\d{1,3})>
        <require: (?{ $MATCH < 256 })>

    <token: IP4_addr>
        <[MATCH=quad]>**(\.)
        <require: (?{ @$MATCH == 4 })>
}xms;

while (my $line = <>) {
    if ($line =~ $grammar) {
        use Data::Dumper 'Dumper';
        say Dumper \%/;
    }
    else {
        say 'Does not match'
    }
}
```

Las condiciones usadas en el `require` obligan a que cada quad[9] sea menor que 256 y a que existan sólo cuatro quads.

Sigue un ejemplo de ejecución:

```
pl@nereida:~/Lregexpgrammars/demo$ ./demo_IP4.pl
123 . 145 . 105 . 252
Does not match
pl@nereida:~/Lregexpgrammars/demo$ ./demo_IP4.pl
123.145.105.252
$VAR1 = {
          '' => '123.145.105.252',
          'IP4_addr' => [
                          123,
                          145,
                          105,
                          252
                        ]
        };
pl@nereida:~/Lregexpgrammars/demo$ ./demo_IP4.pl
148.257.128.128
Does not match
0.0.0.299
Does not match
pl@nereida:~/Lregexpgrammars/demo$  ./demo_IP4.pl
123.145.105.242.193
```

[9] A quad (pronounced KWAHD) is a unit in a set of something that comes in four units. The term is sometimes used to describe each of the four numbers that constitute an Internet Protocol (IP) address. Thus, an Internet address in its numeric form (which is also sometimes called a dot address) consists of four quads separated by "dots"(periods).

A quad also means *a quarter* in some usages. (A quarter as a U.S. coin or monetary unit means *a quarter of a dollar,* and in slang is sometimes called *two bits.* However, this usage does not mean two binary bits as used in computers.)

```
Does not match
```

Obsérvese como no se aceptan blancos entre los puntos en esta versión. ¿Sabría explicar la causa?

3.10.11. Casando con las claves de un hash

In some situations a grammar may need a rule that matches dozens, hundreds, or even thousands of one-word alternatives. For example, when matching command names, or valid userids, or English words. In such cases it is often impractical (and always inefficient) to list all the alternatives between `|` *alterators:*

```
<rule: shell_cmd>
    a2p | ac | apply | ar | automake | awk | ...
    # ...and 400 lines later
    ... | zdiff | zgrep | zip | zmore | zsh

<rule: valid_word>
    a | aa | aal | aalii | aam | aardvark | aardwolf | aba | ...
    # ...and 40,000 lines later...
    ... | zymotize | zymotoxic | zymurgy | zythem | zythum
```

To simplify such cases, `Regexp::Grammars` *provides a special construct that allows you to specify all the alternatives as the keys of a normal hash. The syntax for that construct is simply to put the hash name inside angle brackets (with no space between the angles and the hash name).*

Which means that the rules in the previous example could also be written:

```
<rule: shell_cmd>
    <%cmds>

<rule: valid_word>
    <%dict>
```

provided that the two hashes (`%cmds` *and* `%dict`*) are visible in the scope where the grammar is created.*

Internally, the construct is converted to something equivalent to:

```
<rule: shell_cmd>
    (<.hk>)  <require: exists $cmds{$CAPTURE}>

<rule: valid_word>
    (<.hk>)  <require: exists $dict{$CAPTURE}>
```

The special `<hk>` *rule is created automatically, and defaults to* `\S+`*, but you can also define it explicitly to handle other kinds of keys. For example:*

```
<rule: hk>
    .+            # Key may be any number of chars on a single line

<rule: hk>
    [ACGT]{10,}   # Key is a base sequence of at least 10 pairs
```

Matching a hash key in this way is typically significantly faster than matching a full set of alternations. Specifically, it is O(length of longest potential key), instead of O(number of keys).

Ejemplo de uso de la directiva hash

Sigue un ejemplo:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n hash.pl
#!/usr/bin/env perl5.10.1
use strict;
use warnings;
use 5.010;
use Data::Dumper;
$Data::Dumper::Deparse = 1;

my %cmd = map { ($_ => undef ) } qw( uname pwd date );

my $rbb = do {
    use Regexp::Grammars;

    qr{
      ^<command>$

      <rule: command>
        <cmd=%cmd> (?: <[arg]> )*

      <token: arg> [^\s<>`&]+
    }xms;
};

while (my $input = <>) {
    chomp($input);
    if ($input =~ m{$rbb}) {
        say("matches: <$&>");
        say Dumper \%/;
        system $/{''}
    }
    else {
        say("does not match");
    }
}
```

Sigue un ejemplo de ejecución:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 hash.pl
a2p f1 f2
matches: <a2p f1 f2>
$VAR1 = {
          '' => 'a2p f1 f2',
          'command' => {
                         '' => 'a2p f1 f2',
                         'cmd' => 'a2p',
                         'arg' => [
                                    'f1',
                                    'f2'
                                  ]
                       }
        };
```

```
pocho 2 5
does not match
```

3.10.12. Depuración

Regexp::Grammars *provides a number of features specifically designed to help debug both grammars and the data they parse.*

All debugging messages are written to a log file (which, by default, is just STDERR*). However, you can specify a disk file explicitly by placing a* "<logfile:...>" *directive* at the start of your grammar[10]*:*

```
$grammar = qr{

    <logfile: LaTeX_parser_log >

    \A <LaTeX_file> \Z    # Pattern to match

    <rule: LaTeX_file>
        # etc.
}x;
```

You can also explicitly specify that messages go to the terminal:

```
<logfile: - >
```

Debugging grammar creation

Whenever a log file has been directly specified, Regexp::Grammars *automatically does verbose static analysis of your grammar. That is, whenever it compiles a grammar containing an explicit* "<logfile:...>" *directive it logs a series of messages explaining how it has interpreted the various components of that grammar. For example, the following grammar:*

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n log.pl
#!/usr/bin/env perl5.10.1
use strict;
use warnings;
use 5.010;
use Data::Dumper;

my $rbb = do {
    use Regexp::Grammars;

    qr{
      <logfile: ->

      <numbers>

      <rule: numbers>
        <number> ** <.comma>

      <token: number> \d+

```

[10]no funcionará si no se pone al principio de la gramática

```
        <token: comma>   ,
    }xms;
};

while (my $input = <>) {
    if ($input =~ m{$rbb}) {
        say("matches: <$&>");
        say Dumper \%/;
    }
}
```

would produce the following analysis in the terminal:

```
pl@nereida:~/Lregexpgrammars/demo$ ./log.pl
  warn | Repeated subrule <number>* will only capture its final match
       | (Did you mean <[number]>* instead?)
       |
  info | Processing the main regex before any rule definitions
       |    |
       |    |...Treating <numbers> as:
       |    |      |  match the subrule <numbers>
       |    |       \ saving the match in $MATCH{'numbers'}
       |    |
       |     \___End of main regex
       |
       | Defining a rule: <numbers>
       |    |...Returns: a hash
       |    |
       |    |...Treating <number> as:
       |    |      |  match the subrule <number>
       |    |       \ saving the match in $MATCH{'number'}
       |    |
       |    |...Treating <.comma> as:
       |    |      |  match the subrule <comma>
       |    |       \ but don't save anything
       |    |
       |    |...Treating <number> ** <.comma> as:
       |    |      |  repeatedly match the subrule <number>
       |    |       \ as long as the matches are separated by matches of <.comma>
       |    |
       |     \___End of rule definition
       |
       | Defining a rule: <number>
       |    |...Returns: a hash
       |    |
       |    |...Treating '\d' as:
       |    |       \ normal Perl regex syntax
       |    |
       |    |...Treating '+ ' as:
       |    |       \ normal Perl regex syntax
       |    |
       |     \___End of rule definition
       |
       | Defining a rule: <comma>
```

```
        |     |...Returns: a hash
        |     |
        |     |...Treating ', ' as:
        |     |       \ normal Perl regex syntax
        |     |
        |      \___End of rule definition
        |
2, 3, 4
matches: <2, 3, 4>
$VAR1 = {
          '' => '2, 3, 4',
          'numbers' => {
                         '' => '2, 3, 4',
                         'number' => '4'
                       }
        };
```

This kind of static analysis is a useful starting point in debugging a miscreant grammar[11]*, because it enables you to see what you actually specified (as opposed to what you* thought *you'd specified).*

Debugging grammar execution

`Regexp::Grammars` *also provides a simple interactive debugger, with which you can observe the process of parsing and the data being collected in any result-hash.*

To initiate debugging, place a `<debug:...>` *directive anywhere in your grammar. When parsing reaches that directive the debugger will be activated, and the command specified in the directive immediately executed. The available commands are:*

```
<debug: on>    - Enable debugging, stop when entire grammar matches
<debug: match> - Enable debugging, stope when a rule matches
<debug: try>   - Enable debugging, stope when a rule is tried
<debug: off>   - Disable debugging and continue parsing silently

<debug: continue> - Synonym for <debug: on>
<debug: run>      - Synonym for <debug: on>
<debug: step>     - Synonym for <debug: try>
```

These directives can be placed anywhere within a grammar and take effect when that point is reached in the parsing. Hence, adding a `<debug:step>` *directive is very much like setting a breakpoint at that point in the grammar. Indeed, a common debugging strategy is to turn debugging on and off only around a suspect part of the grammar:*

```
<rule: tricky>   # This is where we think the problem is...
    <debug:step>
    <preamble> <text> <postscript>
    <debug:off>
```

Once the debugger is active, it steps through the parse, reporting rules that are tried, matches and failures, backtracking and restarts, and the parser's location within both the grammar and the text being matched. That report looks like this:

[11] miscreant - *One who has behaved badly, or illegally; One not restrained by moral principles; an unscrupulous villain; One who holds an incorrect religious belief; an unbeliever; Lacking in conscience or moral principles; unscrupulous; Holding an incorrect religious belief.*

```
===============> Trying <grammar> from position 0
> cp file1 file2 |...Trying <cmd>
                 |   |...Trying <cmd=(cp)>
                 |   |    \FAIL <cmd=(cp)>
                 |    \FAIL <cmd>
                  \FAIL <grammar>
===============> Trying <grammar> from position 1
 cp file1 file2  |...Trying <cmd>
                 |   |...Trying <cmd=(cp)>
 file1 file2     |   |    \_____<cmd=(cp)> matched 'cp'
file1 file2      |   |...Trying <[file]>+
 file2           |   |    \_____<[file]>+ matched 'file1'
                 |   |...Trying <[file]>+
[eos]            |   |    \_____<[file]>+ matched ' file2'
                 |   |...Trying <[file]>+
                 |   |    \FAIL <[file]>+
                 |   |...Trying <target>
                 |   |   |...Trying <file>
                 |   |   |    \FAIL <file>
                 |   |    \FAIL <target>
 <~~~~~~~~~~~~~~ |   |...Backtracking 5 chars and trying new match
file2            |   |...Trying <target>
                 |   |   |...Trying <file>
                 |   |   |    \____ <file> matched 'file2'
[eos]            |   |    \_____<target> matched 'file2'
                 |    \_____<cmd> matched ' cp file1 file2'
                  \_____<grammar> matched ' cp file1 file2'
```

The first column indicates the point in the input at which the parser is trying to match, as well as any backtracking or forward searching it may need to do. The remainder of the columns track the parser's hierarchical traversal of the grammar, indicating which rules are tried, which succeed, and what they match.

Provided the logfile is a terminal (as it is by default), the debugger also pauses at various points in the parsing process–before trying a rule, after a rule succeeds, or at the end of the parse–according to the most recent command issued. When it pauses, you can issue a new command by entering a single letter:

```
m       - to continue until the next subrule matches
t or s  - to continue until the next subrule is tried
r or c  - to continue to the end of the grammar
o       - to switch off debugging
```

Note that these are the first letters of the corresponding `<debug:...>` *commands, listed earlier. Just hitting ENTER while the debugger is paused repeats the previous command.*

While the debugger is paused you can also type a `d`*, which will display the result-hash for the current rule. This can be useful for detecting which rule isn't returning the data you expected.*

Veamos un ejemplo. El siguiente programa activa el depurador:

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n demo_debug.pl
#!/usr/bin/env perl5.10.1
use 5.010;
use warnings;

```

```
use Regexp::Grammars;

my $balanced_brackets = qr{
    <debug:on>

    <left_delim=(  \( )>
    (?:
        <[escape=(  \\ )]>
    |   <recurse=( (?R) )>
    |   <[simple=(  .  )]>
    )*
    <right_delim=( \) )>
}xms;

while (<>) {
    if (/$balanced_brackets/) {
        say 'matched:';
        use Data::Dumper 'Dumper';
        warn Dumper \%/;
    }
}
```

Al ejecutar obtenemos

```
pl@nereida:~/Lregexpgrammars/demo$ ./demo_debug.pl
(a)
=====> Trying <grammar> from position 0
(a)\n  |...Trying <left_delim=(  \( )>
a)\n   |    \_____<left_delim=(  \( )> matched '('      c
       |...Trying <[escape=(  \ )]>
       |    \FAIL <[escape=(  \ )]>
       |...Trying <recurse=( (?R) )>
=====> Trying <grammar> from position 1
a)\n   |   |...Trying <left_delim=(  \( )>
       |   |    \FAIL <left_delim=(  \( )>
        \FAIL <grammar>
       |...Trying <[simple=(  .  )]>
)\n    |    \_____<[simple=(  .  )]> matched 'a'
       |...Trying <[escape=(  \ )]>
       |    \FAIL <[escape=(  \ )]>
       |...Trying <recurse=( (?R) )>
=====> Trying <grammar> from position 2
)\n    |   |...Trying <left_delim=(  \( )>
       |   |    \FAIL <left_delim=(  \( )>
        \FAIL <grammar>
       |...Trying <[simple=(  .  )]>
\n     |    \_____<[simple=(  .  )]> matched ')'
       |...Trying <[escape=(  \ )]>
       |    \FAIL <[escape=(  \ )]>
       |...Trying <recurse=( (?R) )>
=====> Trying <grammar> from position 3
\n     |   |...Trying <left_delim=(  \( )>
       |   |    \FAIL <left_delim=(  \( )>
        \FAIL <grammar>
```

```
       |...Trying <[simple=(  .  )]>
[eos]  |    \_____<[simple=(  .  )]> matched ''
       |...Trying <[escape=(  \ )]>
       |    \FAIL <[escape=(  \ )]>
       |...Trying <recurse=( (?R) )>
=====> Trying <grammar> from position 4
[eos]  |   |...Trying <left_delim=(  \( )>
       |   |    \FAIL <left_delim=(  \( )>
        \FAIL <grammar>
       |...Trying <[simple=(  .  )]>
       |    \FAIL <[simple=(  .  )]>
       |...Trying <right_delim=( \) )>
       |    \FAIL <right_delim=( \) )>
 <~~~~ |...Backtracking 1 char and trying new match
\n     |...Trying <right_delim=( \) )>
       |    \FAIL <right_delim=( \) )>
 <~~~~ |...Backtracking 1 char and trying new match
)\n    |...Trying <right_delim=( \) )>
\n     |    \_____<right_delim=( \) )> matched ')'
        \_____<grammar> matched '(a)'   d
              :         {
              :           '' => '(a)',
              :           'left_delim' => '(',
              :           'simple' => [
              :                         'a'
              :                       ],
              :           'right_delim' => ')'
              :         };      o
matched:
$VAR1 = {
          '' => '(a)',
          'left_delim' => '(',
          'simple' => [
                        'a'
                      ],
          'right_delim' => ')'
        };
```

3.10.13. Mensajes de `log` del usuario

Both static and interactive debugging send a series of predefined log messages to whatever log file you have specified. It is also possible to send additional, user-defined messages to the log, using the `"<log:...>"` *directive.*

This directive expects either a simple text or a codeblock as its single argument. If the argument is a code block, that code is expected to return the text of the message; if the argument is anything else, that something else is *the literal message. For example:*

```
        <rule: ListElem>

            <Elem=   ( [a-z]\d+) >
                <log: Checking for a suffix, too...>

            <Suffix= ( : \d+   ) >?
                <log: (?{ "ListElem: $MATCH{Elem} and $MATCH{Suffix}" })>
```

User-defined log messages implemented using a codeblock can also specify a severity level. If the codeblock of a `<log:...>` *directive returns two or more values, the first is treated as a log message severity indicator, and the remaining values as separate lines of text to be logged. For example:*

```
<rule: ListElem>
    <Elem=   ( [a-z]\d+) >
    <Suffix= ( : \d+   ) >?

        <log: (?{
            warn => "Elem was: $MATCH{Elem}",
                    "Suffix was $MATCH{Suffix}",
        })>
```

When they are encountered, user-defined log messages are interspersed between any automatic log messages (i.e. from the debugger), at the correct level of nesting for the current rule.

3.10.14. Depuración de Regexps

It is possible to use `Regexp::Grammars` *without creating* any *subrule definitions, simply to debug a recalcitrant regex. For example, if the following regex wasn't working as expected:*

```
my $balanced_brackets = qr{
    \(             # left delim
    (?:
        \\         # escape or
    |   (?R)       # recurse or
    |   .          # whatever
    )*
    \)             # right delim
}xms;
```

you could instrument it with aliased subpatterns and then debug it step-by-step, using `Regexp::Grammars`*:*

```
use Regexp::Grammars;

my $balanced_brackets = qr{
    <debug:step>

    <.left_delim=  (  \(  )>
    (?:
        <.escape=  (  \\  )>
    |   <.recurse= ( (?R) )>
    |   <.whatever=(  .   )>
    )*
    <.right_delim= (  \)  )>
}xms;

while (<>) {
    say 'matched' if /$balanced_brackets/;
}
```

Note the use of amnesiac aliased subpatterns to avoid needlessly building a result-hash. Alternatively, you could use listifying aliases to preserve the matching structure as an additional debugging aid:

```
use Regexp::Grammars;

my $balanced_brackets = qr{
    <debug:step>

    <[left_delim=  (  \(  )]>
    (?:
        <[escape=  (  \\  )]>
    |   <[recurse= ( (?R) )]>
    |   <[whatever=(  .   )]>
    )*
    <[right_delim= (  \)  )]>
}xms;

if ( '(a(bc)d)' =~ /$balanced_brackets/) {
    use Data::Dumper 'Dumper';
    warn Dumper \%/;
}
```

3.10.15. Manejo y recuperación de errores

En este punto debo decir que no he podido reproducir el comportamiento de las directivas `<error:>` y `<warning:>` tal y como las describe Conway en el manual de `Regexp::Grammars`.

El siguiente ejemplo ilustra un conjunto de técnicas de gestión de errores que son independientes del soprote dado por `Regexp::Grammars`.

Se trata de la misma calculadora explicada en la sección 3.10.18.

```
pl@nereida:~/Lregexpgrammars/demo/calculator$ cat -n calculatorwitherrmanagement.pl
#!/usr/bin/env perl5.10.1
use strict;
use warnings;
use 5.010;
use Lingua::EN::Inflect qw(PL);
use Scalar::Util qw{blessed};

my $rbb = do {
    my ($warnings, $errors);    # closure
    sub warnings { $warnings }  # accessor
    sub errors { $errors }      # accessor

    use Regexp::Grammars;
    qr{
      (?{
          $warnings = 0;
          $errors = 0;
      })
      \A<expr>
      (?:   \z
           |
             (.*) (?{
```

```
                      # Accept the string but emit a warning
                      $warnings++;
                      local our $expr = \$MATCH{expr}{''};
                      local our $endlegal = length($$expr) > 4? "... ".substr($$expr, -4)
                      warn "Warning: Unexpected '". substr($^N, 0, 10)."' after '$endlegal
                   })
      )

      <objrule: expr>      <[operands=term]> ** <[operators=addop]>

      <objrule: term>      <[operands=uneg]> ** <[operators=mulop]>

      <objrule: uneg>      <[operators=minus]>* <[operands=power]>

      <objrule: power>     <[operands=factorial]> ** <[operators=powerop]>

      <objrule: factorial> <[operands=factor]>  <[operators=(!)]>*

      <objrule: factor>    (<val=([+-]?\d+(?:\.\d*)?)>)
                         | \( <MATCH=expr> \)
                         | ([^-+(0-9]+) (?{
                                          # is + and not * to avoid infinite recursion
                                          warn "Error: expecting a number or a open parent
                                          $warnings++;
                                          $errors++;
                                      }) <MATCH=factor>

      <token: addop>        [+-]

      <token: mulop>        [*/]

      <token: powerop>      \*\*|\^

      <token: minus>        - <MATCH=(?{ 'NEG' })>

    }x;
};

sub test_calc {
  my $prompt = shift;

  print $prompt;
  while (my $input = <>) {
      chomp($input);

      local %/;
      $input =~ m{$rbb};

      say warnings." ".PL('warning',warnings) if warnings;
      say errors." ".PL('error',errors)       if errors;

      my $tree = $/{expr};
      if (blessed($tree)) {
```

```
          do "PostfixCalc.pm";
          say "postfix: ".$tree->ceval;

          do "EvalCalc.pm";
          say "result: ".$tree->ceval;
      }
      print $prompt;
  }
  say "Bye!"
}

########## main
test_calc(
  'Parsing infix arithmetic expressions (CTRL-D to end in unix) ',
);
```

Veamos algunas ejecuciones que incluyen entradas erróneas:

```
pl@nereida:~/Lregexpgrammars/demo/calculator$ ./calculatorwitherrmanagement.pl
Parsing infix arithmetic expressions (CTRL-D to end in unix) 2+3
postfix: 2 3 +
result: 5
Parsing infix arithmetic expressions (CTRL-D to end in unix) 2*(3+#)
Error: expecting a number or a open parenthesis, found: '#)'
Error: expecting a number or a open parenthesis, found: '#'
Error: expecting a number or a open parenthesis, found: ')'
Warning: Unexpected '*(3+#)' after '2'
4 warnings
3 errors
postfix: 2
result: 2
Parsing infix arithmetic expressions (CTRL-D to end in unix) 2+#*4
Error: expecting a number or a open parenthesis, found: '#*'
1 warning
1 error
postfix: 2 4 +
result: 6
Parsing infix arithmetic expressions (CTRL-D to end in unix) Bye!
```

Obsérvese los mensajes de error repetidos para la entrada `2*(3+#)`. Ellos son debidos a los reiterados intentos de casar `<factor>` en la regla de recuperación de errores:

```
      <objrule: factor>    (<val=([+-]?\d+(?:\.\d*)?)>)
                         | \( <MATCH=expr> \)
                         | ([^-+(0-9]+) (?{
                                         # is + and not * to avoid infinite recursion
                                         warn "Error: expecting a number or a open parent
                                         $warnings++;
                                         $errors++;
                                     }) <MATCH=factor>
```

en este caso resulta imposible encontrar un factor. Se puede cambiar la conducta indicando un `(* COMMIT)` antes de la llamada a `<MATCH=factor>`:

```
      <objrule: factor>    (<val=([+-]?\d+(?:\.\d*)?)>)
                         | \( <MATCH=expr> \)
```

```
                        | ([^-+(0-9]+) (?{
                                         # is + and not * to avoid infinite recursion
                                         warn "Error: expecting a number or a open parent
                                         $warnings++;
                                         $errors++;
                                     }) (*COMMIT) <MATCH=factor>
```

en este caso la conducta es abandonar en el caso de que no se pueda encontrar un `<factor>`:

```
pl@nereida:~/Lregexpgrammars/demo/calculator$ ./calculatorwitherrmanagement.pl
Parsing infix arithmetic expressions (CTRL-D to end in unix) 2*(3+#)
Error: expecting a number or a open parenthesis, found: '#)'
1 warning
1 error
Parsing infix arithmetic expressions (CTRL-D to end in unix) 2*3
postfix: 2 3 *
result: 6
Parsing infix arithmetic expressions (CTRL-D to end in unix) @
Error: expecting a number or a open parenthesis, found: '@'
1 warning
1 error
Parsing infix arithmetic expressions (CTRL-D to end in unix) Bye!
```

3.10.16. Mensajes de Warning

> *Sometimes, you want to detect problems, but not invalidate the entire parse as a result. For those occasions, the module provides a* less stringent *form of error reporting: the* `<warning:...>` *directive.*
>
> *This directive is exactly the same as an* `<error:...>` *in every respect except that it does not induce a failure to match at the point it appears.*
>
> *The directive is, therefore, useful for reporting* non-fatal *problems in a parse. For example:*
>
> ```
> qr{ \A            # ...Match only at start of input
>     <ArithExpr>   # ...Match a valid arithmetic expression
>
>     (?:
>         # Should be at end of input...
>         \s* \Z
>       |
>         # If not, report the fact but don't fail...
>         <warning: Expected end-of-input>
>         <warning: (?{ "Extra junk at index $INDEX: $CONTEXT" })>
>     )
>
>     # Rule definitions here...
> }xms;
> ```
>
> *Note that, because they do not induce failure, two or more* `<warning:...>` *directives can be "stackedïn sequence, as in the previous example.*

3.10.17. Simplificando el AST

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n exprdamian.pl
     1  use strict;
```

```perl
use warnings;
use 5.010;
use Data::Dumper;
$Data::Dumper::Indent = 1;

my $rbb = do {
    use Regexp::Grammars;

    qr{
      \A<expr>\z

      <objrule: expr>    <MATCH=term> (?! <addop> )                  # bypass
                       | <[operands=term]> ** <[operators=addop]>

      <objrule: term>    <MATCH=factor> (?! <mulop> )                # bypass
                       | <[operands=factor]> ** <[operators=mulop]>

      <objrule: factor>    <val=([+-]?\d+(?:\.\d*)?)>
                       | \( <MATCH=expr> \)

      <token: addop> [+-]

      <token: mulop> [*/]

    }x;
};

while (my $input = <>) {
    chomp($input);
    if ($input =~ m{$rbb}) {
        my $tree = $/{expr};
        say Dumper $tree;
        say $tree->ceval;

    }
    else {
        say("does not match");
    }
}

BEGIN {

  package LeftBinaryOp;
  use strict;
  use base qw(Class::Accessor);

  LeftBinaryOp->mk_accessors(qw{operators operands});

  my %f = (
    '+' => sub { shift() + shift() },
    '-' => sub { shift() - shift() },
    '*' => sub { shift() * shift() },
    '/' => sub { shift() / shift() },
```

```
  );

  sub ceval {
    my $self = shift;

    # recursively evaluate the children first
    my @operands = map { $_->ceval } @{$self->operands};

    # then combine them
    my $s = shift @operands;
    for (@{$self->operators}) {
      $s = $f{$_}->($s, shift @operands);
    }
    return $s;
  }

  package term;
  use base qw{LeftBinaryOp};

  package expr;
  use base qw{LeftBinaryOp};

  package factor;

  sub ceval {
    my $self = shift;

    return $self->{val};
  }

  1;
}
```

Ejecuciones:

```
pl@nereida:~/Lregexpgrammars/demo$ perl5.10.1 exprdamian.pl
4-2-2
$VAR1 = bless( {
  'operands' => [
    bless( {
      '' => '4',
      'val' => '4'
    }, 'factor' ),
    bless( {
      '' => '2',
      'val' => '2'
    }, 'factor' ),
    bless( {
      '' => '2',
      'val' => '2'
    }, 'factor' )
  ],
  '' => '4-2-2',
  'operators' => [
```

```
    '-',
    '-'
  ]
}, 'expr' );

0
8/4/2
$VAR1 = bless( {
  'operands' => [
    bless( {
      '' => '8',
      'val' => '8'
    }, 'factor' ),
    bless( {
      '' => '4',
      'val' => '4'
    }, 'factor' ),
    bless( {
      '' => '2',
      'val' => '2'
    }, 'factor' )
  ],
  '' => '8/4/2',
  'operators' => [
    '/',
    '/'
  ]
}, 'term' );

1
3
$VAR1 = bless( {
  '' => '3',
  'val' => '3'
}, 'factor' );

3
2*(3+4)
$VAR1 = bless( {
  'operands' => [
    bless( {
      '' => '2',
      'val' => '2'
    }, 'factor' ),
    bless( {
      'operands' => [
        bless( {
          '' => '3',
          'val' => '3'
        }, 'factor' ),
        bless( {
          '' => '4',
          'val' => '4'
```

```
        }, 'factor' )
      ],
      '' => '3+4',
      'operators' => [
        '+'
      ]
    }, 'expr' )
  ],
  '' => '2*(3+4)',
  'operators' => [
    '*'
  ]
}, 'term' );

14
```

3.10.18. Reciclando una `Regexp::Grammar`

Ejecución

El siguiente programa `calculator.pl` recibe como entrada una expresión en infijo.

La ejecución consta de dos bucles. En la primera parte se inyecta a la jerarquía de clases de los AST generados para las expresiones en infijo una semántica que permite evaluar la expresión:

```
require EvalCalc;

test_calc(
  'Evaluating infix arithmetic expressions (CTRL-D to end in unix) ',
  sub { print &Data::Dumper::Dumper(shift()) },
);
```

En esta primera parte mostraremos además el AST construido para la expresión infija de entrada.

```
pl@nereida:~/Lregexpgrammars/demo$ ./calculator.pl
Evaluating infix arithmetic expressions (CTRL-D to end in unix)
8-4-2
$VAR1 = bless( {
  'operands' => [
    bless( {
      'operands' => [
        bless( {
          'operands' => [
            bless( {
              'operands' => [
                bless( {
                  'operands' => [
                    bless( { '' => '8', 'val' => '8' }, 'factor' )
                  ],
                  '' => '8'
                }, 'factorial' )
              ],
              '' => '8'
            }, 'power' )
          ],
          '' => '8'
        }, 'uneg' )
```

```
      ],
      '' => '8'
    }, 'term' ),
    bless( {
      'operands' => [
        bless( {
          'operands' => [
            bless( {
              'operands' => [
                bless( {
                  'operands' => [
                    bless( { '' => '4', 'val' => '4' }, 'factor' )
                  ],
                  '' => '4'
                }, 'factorial' )
              ],
              '' => '4'
            }, 'power' )
          ],
          '' => '4'
        }, 'uneg' )
      ],
      '' => '4'
    }, 'term' ),
    bless( {
      'operands' => [
        bless( {
          'operands' => [
            bless( {
              'operands' => [
                bless( {
                  'operands' => [
                    bless( { '' => '2', 'val' => '2' }, 'factor' )
                  ],
                  '' => '2'
                }, 'factorial' )
              ],
              '' => '2'
            }, 'power' )
          ],
          '' => '2'
        }, 'uneg' )
      ],
      '' => '2'
    }, 'term' )
  ],
  '' => '8-4-2',
  'operators' => [
    '-',
    '-'
  ]
}, 'expr' );
2
```

Observamos que la asociatividad es la correcta. El 2 final es el resultado de la evaluación de 8-4-2.

La estructura del árbol se corresponde con la de la gramática:

```
my $rbb = do {
    use Regexp::Grammars;

    qr{
      \A<expr>\z

      <objrule: expr>      <[operands=term]> ** <[operators=addop]>

      <objrule: term>      <[operands=uneg]> ** <[operators=mulop]>

      <objrule: uneg>      <[operators=minus]>* <[operands=power]>

      <objrule: power>     <[operands=factorial]> ** <[operators=powerop]>

      <objrule: factorial> <[operands=factor]>  <[operators=(!)]>*

      <objrule: factor>    <val=([+-]?\d+(?:\.\d*)?)>
                         | \( <MATCH=expr> \)

      <token: addop>        [+-]

      <token: mulop>        [*/]

      <token: powerop>      \*\*|\^

      <token: minus>        - <MATCH=(?{ 'NEG' })>

    }x;
};
```

Ahora, en una segunda parte sobreescribimos los métodos `sem` que describen la semántica para producir una traducción de infijo a postfijo:

```
require PostfixCalc;
test_calc('Translating expressions to postfix (CTRL-D to end in unix) ');
```

Ahora al proporcionar la entrada `6--3!` obtenemos:

```
Translating expressions to postfix (CTRL-D to end in unix)
6--3!
6 3 ! ~ -
```

Aquí `~` es el operador de negación unaria y `!` es el operador factorial.

Estructura de la aplicación

Estos son los ficheros que integran la aplicación:

```
pl@nereida:~/Lregexpgrammars/demo/calculator$ tree
.
|-- EvalCalc.pm            # Soporte para la evaluación de la expresión: sem
|-- Operator.pm            # Soporte a las clases nodo: recorridos
|-- PostfixCalc.pm         # Soporte para la traducción a postfijo: sem
`-- calculator.pl          # programa principal
```

Programa principal

En el programa principal definimos la gramática y escribimos una subrutina `test_calc` que realiza el parsing.

```
pl@nereida:~/Lregexpgrammars/demo/calculator$ cat -n calculator.pl
#!/usr/bin/env perl5.10.1
use strict;
use warnings;
use 5.010;
use Data::Dumper;
$Data::Dumper::Indent = 1;

my $rbb = do {
    use Regexp::Grammars;

    qr{
      \A<expr>\z

      <objrule: expr>      <[operands=term]> ** <[operators=addop]>

      <objrule: term>      <[operands=uneg]> ** <[operators=mulop]>

      <objrule: uneg>      <[operators=minus]>* <[operands=power]>

      <objrule: power>     <[operands=factorial]> ** <[operators=powerop]>

      <objrule: factorial> <[operands=factor]>  <[operators=(!)]>*

      <objrule: factor>    <val=([+-]?\d+(?:\.\d*)?)>
                         | \( <MATCH=expr> \)

      <token: addop>        [+-]

      <token: mulop>        [*/]

      <token: powerop>      \*\*|\^

      <token: minus>        - <MATCH=(?{ 'NEG' })>

    }x;
};

sub test_calc {
  my $prompt = shift;
  my $handler = shift;

  say $prompt;
  while (my $input = <>) {
      chomp($input);
      if ($input =~ m{$rbb}) {
          my $tree = $/{expr};
          $handler->($tree) if $handler;

          say $tree->ceval;
```

```

      }
      else {
          say("does not match");
      }
  }
}

require EvalCalc;

test_calc(
  'Evaluating infix arithmetic expressions (CTRL-D to end in unix) ',
  sub { print &Data::Dumper::Dumper(shift()) },
);


require PostfixCalc;
test_calc('Translating expressions to postfix (CTRL-D to end in unix) ');
```

Los nodos del AST poseen un método `ceval` que se encarga de realizar la traducción del nodo.

Las Clases de nodos del AST

```
pl@nereida:~/Lregexpgrammars/demo/calculator$ cat -n Operator.pm
#   Class hierarchy diagram:
#  $ vgg -t 'Operator(LeftBinaryOp(expr,term),RightBinaryOp(power),PreUnaryOp(uneg),Post
#                           +--------+
#                           |Operator|
#                           +--------+
#          .---------------.----^--------.-------------.
#    +------------+ +-------------+ +----------+ +-----------+
#    |LeftBinaryOp| |RightBinaryOp| |PreUnaryOp| |PostUnaryOp|
#    +------------+ +-------------+ +----------+ +-----------+
#      .---^--.        |              |               |
#    +----+ +----+  +-----+         +----+       +---------+
#    |expr| |term|  |power|         |uneg|       |factorial|
#    +----+ +----+  +-----+         +----+       +---------+
#
#
# NOTE: package "factor" actually implements numbers and is
#       outside this hierarchy
#
package Operator;
use strict;
use Carp;

sub Operands {
  my $self = shift;

  return () unless exists $self->{operands};
  return @{$self->{operands}};
}

sub Operators {
```

```
  my $self = shift;

  return () unless exists $self->{operators};
  return @{$self->{operators}};
}

sub sem {
  confess "not defined sem";
}

sub make_sem {
  my $class = shift;
  my %semdesc = @_;

  for my $class (keys %semdesc) {
    my %sem = %{$semdesc{$class}};

    # Install 'sem' method in $class
    no strict 'refs';
    no warnings 'redefine';
    *{$class."::sem"} = sub {
      my ($self, $op) = @_;
      $sem{$op}
    };
  }
}

package LeftBinaryOp;
use base qw{Operator};

sub ceval {
  my $self = shift;

  # recursively evaluate the children first
  my @operands = map { $_->ceval } $self->Operands;

  # then combine them
  my $s = shift @operands;
  for ($self->Operators) {
    $s = $self->sem($_)->($s, shift @operands);
  }
  return $s;
}

package RightBinaryOp;
use base qw{Operator};

sub ceval {
  my $self = shift;

  # recursively evaluate the children first
  my @operands = map { $_->ceval } $self->Operands;

```

```
  # then combine them
  my $s = pop @operands;
  for (reverse $self->Operators) {
    $s = $self->sem($_)->(pop @operands, $s);
  }
  return $s;
}

package PreUnaryOp;
use base qw{Operator};

sub ceval {
  my $self = shift;

  # recursively evaluate the children first
  my @operands = map { $_->ceval } $self->Operands;

  # then combine them
  my $s = shift @operands;
  for (reverse $self->Operators) {
    $s = $self->sem($_)->($s);
  }
  return $s;
}

package PostUnaryOp;
use base qw{Operator};

sub ceval {
  my $self = shift;

  # recursively evaluate the children first
  my @operands = map { $_->ceval } $self->Operands;

  # then combine them
  my $s = shift @operands;
  for ($self->Operators) {
    $s = $self->sem($_)->($s);
  }
  return $s;
}

package term;
use base qw{LeftBinaryOp};

package expr;
use base qw{LeftBinaryOp};

package power;
use base qw{RightBinaryOp};

package uneg;
use base qw{PreUnaryOp};
```

```

package factorial;
use base qw{PostUnaryOp};

package factor;

sub ceval {
  my $self = shift;

  return $self->{val};
}

1;
```

Definiendo `sem` para la evaluación de la expresión

```
pl@nereida:~/Lregexpgrammars/demo/calculator$ cat -n EvalCalc.pm
package EvalCalc;
use strict;
use Carp;

use Operator;

####
sub f {
  $_[0]>1?$_[0]*f($_[0]-1):1;
}

sub fac {
  my $n = shift;

  confess "Not valid number" unless $n =~ /^\d+$/;
  f($n);
};

my $s = sub { shift() **  shift() };

Operator->make_sem(
   expr => {
      '+' => sub { shift()  +  shift() },
      '-' => sub { shift()  -  shift() },
   },
   term => {
     '*' => sub { shift()  *  shift() },
     '/' => sub { shift()  /  shift() },
   },
   power => {
      '^'  => $s,
      '**' => $s,
   },
   uneg => {
      'NEG' => sub { -shift() },
   },
   factorial => {
```

```
      '!' => \&fac,
   },
);

1;
```

Definiendo `sem` **para la traducción a postfijo**

```
pl@nereida:~/Lregexpgrammars/demo/calculator$ cat -n PostfixCalc.pm
package PostfixCalc;
use strict;

use Operator;

# Modify semantics: now translate to postfix
my $powers = sub { shift().' '.shift().' **' };

Operator->make_sem(
   expr => {
      '+' => sub { shift().' '.shift().' +'  },
      '-' => sub { shift().' '.shift().' -' },
    },
    term => {
      '*' => sub { shift().' '.shift().' *'  },
      '/' => sub { shift().' '.shift().' /' },
    },
    power => {
      '^'  => $powers,
      '**' => $powers,
    },
    uneg => {
       # use ~ for unary minus
      'NEG' => sub { shift().' ~' },
    },
    factorial => {
      '!' => sub { shift().' !'},
    },
);

1;
```

Ejercicio 3.10.2.

- *Explique el significado de la primera línea del programa principal*

```
pl@nereida:~/Lregexpgrammars/demo$ cat -n calculator.pl
#!/usr/bin/env perl5.10.1
```

- *Explique el significado de* `$handler` *en* `test_calc`*:*

```
sub test_calc {
  my $prompt = shift;
  my $handler = shift;

  say $prompt;
  while (my $input = <>) {
```

```
        chomp($input);
        if ($input =~ m{$rbb}) {
            my $tree = $/{expr};
            $handler->($tree) if $handler;

            say $tree->ceval;

        }
        else {
            say("does not match");
        }
    }
}
```

- *Aisle las funciones relacionadas con la creación de semántica como* `make_sem`*,* `fac` *y las llamadas a* `make_sem` *en un módulo* `Calculator::Semantics` *aparte.*
- *Añada un traductor de infijo a prefijo al código presentado en esta sección. Una expresión como* `2*3+4` *se traducirá como* `+ * 2 3 4`

3.10.19. Práctica: Calculadora con `Regexp::Grammars`

- Reforme la estructura del ejemplo para que tenga una jerarquía de desarrollo de acuerdo a los estándares de Perl. Use `h2xs` o bien `Module::Starter`. Use el espacio de nombres `Calculator`. Mueva el módulo `Operator` a `Calculator::Operator`. Lea el capítulo Modulos de los apuntes de LHP.
- Defina el conjunto de pruebas que deberá pasar su traductor. Añádalas como pruebas `TODO`. Cuando la funcionalidad a comprobar esté operativa cambie su estatus.
- Añada variables y la expresión de asignación:

  ```
  b = a = 4*2
  ```

  que será traducida a postfijo como:

  ```
  4 2 * a = b =
  ```

  El operador de asignación es asociativo a derechas. El valor devuelto por una expresión de asignación es el valor asignado.

  Use un hash para implantar la relación nombre-valor en el caso de la evaluación

- Introduzca la expresión bloque:

  ```
  c = { a = 4; b = 2*a }
  ```

  Los bloques son listas entre llaves de expresiones separadas por punto y coma. El valor retornado por una expresión bloque es el último evaluado en el bloque.

  El símbolo de arranque de la gramática (esto es, el patrón regular contra el que hay que casar) será la expresión bloque.

- Introduzca las expresiones de comparación `<`, `>`, `<=`, `>=`, `==` y `!=` con la prioridad adecuada. Tenga en cuenta que una expresión como:

```
a = b+2 > c*4
```

deberá entenderse como

```
a = ((b+2) > (c*4))
```

Esto es, se traducirá como:

```
b 2 + c 4 * > a =
```

- Introduzca la expresión `if ... then ... else`. La parte del `else` será opcional:

  ```
  c = if a > 0 then { a = a -1; 2*a } else { b + 2 };
  d = if a > 0 then { a = b -1; 2*b };
  ```

  un `else` casa con el `if` mas cercano. La sentencia:

  ```
  if (a > 0) then if (b > 0) then {5} else {6}
  ```

  se interpreta como:

  ```
  if (a > 0) then (if (b > 0) then {5} else {6})
  ```

  y no como:

  ```
  if (a > 0) then (if (b > 0) then {5}) else {6}
  ```

  Se traducirá como:

  ```
          a
          0
          >
          jz endif124
          b
          0
          >
          jz else125
          5
          j endif126
  :else125
          6
  :endif124
  :endif125
          ...
  ```

- Escriba un intérprete de la máquina orientada a pila definida en los apartados anteriores. El código generado debería poder ejecutarse correctamente en el intérprete.

Capítulo 4

Analizadores Descendentes Predictivos en JavaScript

4.1. Conceptos Básicos para el Análisis Sintáctico

Suponemos que el lector de esta sección ha realizado con éxito un curso en teoría de autómatas y lenguajes formales. Las siguientes definiciones repasan los conceptos mas importantes.

Definición 4.1.1. *Dado un conjunto A, se define A^* el cierre de Kleene de A como: $A^* = \cup_{n=0}^{\infty} A^n$*

Se admite que $A^0 = \{\epsilon\}$, donde ϵ denota la palabra vacía, esto es la palabra que tiene longitud cero, formada por cero símbolos del conjunto base A.

Definición 4.1.2. *Una gramática G es una cuaterna $G = (\Sigma, V, P, S)$. Σ es el conjunto de terminales. V es un conjunto (disjunto de Σ) que se denomina conjunto de* variables sintácticas *o* categorías gramáticales*, P es un conjunto de pares de $V \times (V \cup \Sigma)^*$. En vez de escribir un par usando la notación $(A, \alpha) \in P$ se escribe $A \to \alpha$. Un elemento de P se denomina producción. Por último, S es un símbolo del conjunto V que se denomina símbolo de arranque.*

Definición 4.1.3. *Dada una gramática $G = (\Sigma, V, P, S)$ y $\mu = \alpha A \beta \in (V \cup \Sigma)^*$ una frase formada por variables y terminales y $A \to \gamma$ una producción de P, decimos que μ deriva en un paso en $\alpha\gamma\beta$. Esto es, derivar una cadena $\alpha A \beta$ es sustituir una variable sintáctica A de V por la parte derecha γ de una de sus reglas de producción. Se dice que μ deriva en n pasos en δ si deriva en $n-1$ pasos en una cadena $\alpha A \beta$ la cual deriva en un paso en δ. Se escribe entonces que $\mu \stackrel{*}{\Longrightarrow} \delta$. Una cadena deriva en 0 pasos en si misma.*

Definición 4.1.4. *Dada una gramática $G = (\Sigma, V, P, S)$ se denota por $L(G)$ o lenguaje generado por G al lenguaje:*

$$L(G) = \{x \in \Sigma^* : S \stackrel{*}{\Longrightarrow} x\}$$

Esto es, el lenguaje generado por la gramática G esta formado por las cadenas de terminales que pueden ser derivados desde el símbolo de arranque.

Definición 4.1.5. *Una derivación que comienza en el símbolo de arranque y termina en una secuencia formada por sólo terminales de Σ se dice* completa.

Una derivación $\mu \stackrel{*}{\Longrightarrow} \delta$ en la cual en cada paso $\alpha A x$ la regla de producción aplicada $A \to \gamma$ se aplica en la variable sintáctica mas a la derecha se dice una derivación a derechas

Una derivación $\mu \stackrel{*}{\Longrightarrow} \delta$ en la cual en cada paso $x A \alpha$ la regla de producción aplicada $A \to \gamma$ se aplica en la variable sintáctica mas a la izquierda se dice una derivación a izquierdas

Definición 4.1.6. *Observe que una derivación puede ser representada como un árbol cuyos nodos están etiquetados en $V \cup \Sigma$. La aplicación de la regla de producción $A \to \gamma$ se traduce en asignar como hijos del nodo etiquetado con A a los nodos etiquetados con los símbolos $X_1 \ldots X_n$ que constituyen la frase $\gamma = X_1 \ldots X_n$. Este árbol se llama* árbol sintáctico concreto *asociado con la derivación.*

Definición 4.1.7. *Observe que, dada una frase $x \in L(G)$ una derivación desde el símbolo de arranque da lugar a un árbol. Ese árbol tiene como raíz el símbolo de arranque y como hojas los terminales $x_1 \ldots x_n$ que forman x. Dicho árbol se denomina* árbol de análisis sintáctico concreto *de x. Una derivación determina una forma de recorrido del árbol de análisis sintáctico concreto.*

Definición 4.1.8. *Una gramática G se dice ambigua si existe alguna frase $x \in L(G)$ con al menos dos árboles sintácticos. Es claro que esta definición es equivalente a afirmar que existe alguna frase $x \in L(G)$ para la cual existen dos derivaciones a izquierda (derecha) distintas.*

4.1.1. Ejercicio

Dada la gramática con producciones:

program → declarations statements | statements
declarations → declaration ';' declarations | declaration ';'
declaration → INT idlist | STRING idlist
statements → statement ';' statements | statement
statement → ID '=' expression | P expression
expression → term '+' expression | term
term → factor '*' term | factor
factor → '(' expression ')' | ID | NUM | STR
idlist → ID ',' idlist | ID

En esta gramática, Σ esta formado por los caracteres entre comillas simples y los símbolos cuyos identificadores están en mayúsculas. Los restantes identificadores corresponden a elementos de V. El símbolo de arranque es $S =$ program.

Conteste a las siguientes cuestiones:

1. Describa con palabras el lenguaje generado.
2. Construya el árbol de análisis sintáctico concreto para cuatro frases del lenguaje.
3. Señale a que recorridos del árbol corresponden las respectivas derivaciones a izquierda y a derecha en el apartado 2.
4. ¿Es ambigua esta gramática?. Justifique su respuesta.

4.2. Análisis Sintáctico Predictivo Recursivo

La siguiente fase en la construcción del analizador es la fase de análisis sintáctico. Esta toma como entrada el flujo de terminales y construye como salida el árbol de análisis sintáctico abstracto.

El árbol de análisis sintáctico abstracto es una representación compactada del árbol de análisis sintáctico concreto que contiene la misma información que éste.

Existen diferentes métodos de análisis sintáctico. La mayoría caen en una de dos categorías: ascendentes y descendentes. Los ascendentes construyen el árbol desde las hojas hacia la raíz. Los descendentes lo hacen en modo inverso. El que describiremos aqui es uno de los mas sencillos: se denomina método de análisis predictivo descendente recursivo.

4.2.1. Introducción

En este método se asocia una subrutina con cada variable sintáctica $A \in V$. Dicha subrutina (que llamaremos `A`) reconocerá el lenguaje generado desde la variable A:

$$L_A(G) = \{x \in \Sigma^* : A \stackrel{*}{\Longrightarrow} x\}$$

statements → statement ';' statements | statement
statement → ID '=' expression | P expression
expression → term '+' expression | term
term → factor '*' term | factor
factor → '(' expression ')' | ID | NUM

Cuadro 4.1: Una Gramática Simple

En este método se escribe una rutina `A` por variable sintáctica $A \in V$. Se le da a la rutina asociada el mismo nombre que a la variable sintáctica asociada.

La función de la rutina `A` asociada con la variable $A \in V$ es reconocer el lenguaje $L(A)$ generado por A.

La estrategia general que sigue la rutina `A` para reconocer $L(A)$ es decidir en términos del terminal a en la entrada que regla de producción concreta $A \to \alpha$ se aplica para a continuación comprobar que la entrada que sigue pertenece al lenguaje generado por α.

En un analizador predictivo descendente recursivo (APDR) se asume que el símbolo que actualmente esta siendo observado (denotado habitualmente como `lookahead`) permite determinar unívocamente que producción de A hay que aplicar.

Una vez que se ha determinado que la regla por la que continuar la derivación es $A \to \alpha$ se procede a reconocer $L_\alpha(G)$, el lenguaje generado por α. Si $\alpha = X_1 \dots X_n$, las apariciones de terminales X_i en α son emparejadas con los terminales en la entrada mientras que las apariciones de variables $X_i = B$ en α se traducen en llamadas a la correspondiente subrutina asociada con `B`.

Ejemplo

Para ilustrar el método, simplificaremos la gramática presentada en el ejercicio 7.1.1 eliminando las declaraciones:

La secuencia de llamadas cuando se procesa la entrada mediante el siguiente programa construye implícitamente el *árbol de análisis sintáctico concreto*.

```
parse = (input) ->
  tokens = input.tokens()
  lookahead = tokens.shift()
  match = (t) ->
    if lookahead.type is t
      lookahead = tokens.shift()
      lookahead = null  if typeof lookahead is "undefined"
    else # Error. Throw exception
      throw "Syntax Error. Expected #{t} found '" +
            lookahead.value + "' near '" +
            input.substr(lookahead.from) + "'"
    return

  statements = ->
    result = [statement()]
    while lookahead and lookahead.type is ";"
      match ";"
      result.push statement()
    (if result.length is 1 then result[0] else result)

  statement = ->
    result = null
    if lookahead and lookahead.type is "ID"
```

```
    left =
      type: "ID"
      value: lookahead.value

    match "ID"
    match "="
    right = expression()
    result =
      type: "="
      left: left
      right: right
  else if lookahead and lookahead.type is "P"
    match "P"
    right = expression()
    result =
      type: "P"
      value: right
  else # Error!
    throw "Syntax Error. Expected identifier but found " +
      (if lookahead then lookahead.value else "end of input") +
      " near '#{input.substr(lookahead.from)}'"
  result

expression = ->
  result = term()
  if lookahead and lookahead.type is "+"
    match "+"
    right = expression()
    result =
      type: "+"
      left: result
      right: right
  result

term = ->
  result = factor()
  if lookahead and lookahead.type is "*"
    match "*"
    right = term()
    result =
      type: "*"
      left: result
      right: right
  result

factor = ->
  result = null
  if lookahead.type is "NUM"
    result =
      type: "NUM"
      value: lookahead.value

    match "NUM"
```

```
  else if lookahead.type is "ID"
    result =
      type: "ID"
      value: lookahead.value

    match "ID"
  else if lookahead.type is "("
    match "("
    result = expression()
    match ")"
  else # Throw exception
    throw "Syntax Error. Expected number or identifier or '(' but found " +
      (if lookahead then lookahead.value else "end of input") +
      " near '" + input.substr(lookahead.from) + "'"
  result

tree = statements(input)
if lookahead?
  throw "Syntax Error parsing statements. " +
    "Expected 'end of input' and found '" +
    input.substr(lookahead.from) + "'"
tree
```

```
var parse = function(input) {
  var tokens = input.tokens();
  var lookahead = tokens.shift();

  var match = function(t) {
    if (lookahead.type === t) {
      lookahead = tokens.shift();
      if (typeof lookahead === 'undefined') {
       lookahead = null; // end of input
      }
    } else { // Error. Throw exception
        throw "Syntax Error. Expected "+t+" found '"+lookahead.value+
              "' near '"+input.substr(lookahead.from)+"'";
    }
  };

  var statements = function() {
    var result = [ statement() ];
    while (lookahead && lookahead.type === ';') {
      match(';');
      result.push(statement());
    }
    return result.length === 1? result[0] : result;
  };

  var statement = function() {
    var result = null;

    if (lookahead && lookahead.type === 'ID') {
      var left = { type: 'ID', value: lookahead.value };
      match('ID');
```

```
    match('=');
    right = expression();
    result = { type: '=', left: left, right: right };
  } else if (lookahead && lookahead.type === 'P') {
    match('P');
    right = expression();
    result = { type: 'P', value: right };
  } else { // Error!
    throw "Syntax Error. Expected identifier but found "+
          (lookahead? lookahead.value : "end of input")+
          " near '"+input.substr(lookahead.from)+"'";
  }
  return result;
};

var expression = function() {
  var result = term();
  if (lookahead && lookahead.type === '+') {
    match('+');
    var right = expression();
    result = {type: '+', left: result, right: right};
  }
  return result;
};

var term = function() {
  var result = factor();
  if (lookahead && lookahead.type === '*') {
    match('*');
    var right = term();
    result = {type: '*', left: result, right: right};
  }
  return result;
};

var factor = function() {
  var result = null;

  if (lookahead.type === 'NUM') {
    result = {type: 'NUM', value: lookahead.value};
    match('NUM');
  }
  else if (lookahead.type === 'ID') {
    result = {type: 'ID', value: lookahead.value};
    match('ID');
  }
  else if (lookahead.type === '(') {
    match('(');
    result = expression();
    match(')');
  } else { // Throw exception
    throw "Syntax Error. Expected number or identifier or '(' but found "+
          (lookahead? lookahead.value : "end of input")+
```

```
          " near '"+input.substr(lookahead.from)+"'";
      }
      return result;
    };
    var tree = statements(input);
    if (lookahead != null) {
        throw "Syntax Error parsing statements. Expected end of input and found '"+
              input.substr(lookahead.from)+"'";
    }
    return tree;
  }
```

Caracterización de las Gramáticas Analizables Como vemos en el ejemplo, el análisis predictivo confía en que, si estamos ejecutando la entrada del procedimiento `A`, el cuál está asociado con la variable $A \in V$, el símbolo terminal que esta en la entrada a determine de manera unívoca la regla de producción $A \to a\alpha$ que debe ser procesada.

Si se piensa, esta condición requiere que todas las partes derechas α de las reglas $A \to \alpha$ de A comiencen por diferentes símbolos. Para formalizar esta idea, introduciremos el concepto de conjunto $FIRST(\alpha)$:

Definición 4.2.1. *Dada una gramática* $G = (\Sigma, V, P, S)$ *y un símbolo* $\alpha \in (V \cup \Sigma)^*$ *se define el conjunto* $FIRST(\alpha)$ *como:*

$$FIRST(\alpha) = \left\{ b \in \Sigma : \alpha \stackrel{*}{\Longrightarrow} b\beta \right\} \cup N(\alpha)$$

donde:

$$N(\alpha) = \begin{cases} \{\epsilon\} & si\ \alpha \stackrel{*}{\Longrightarrow} \epsilon \\ \emptyset & en\ otro\ caso \end{cases}$$

Podemos reformular ahora nuestra afirmación anterior en estos términos: Si $A \to \gamma_1 \mid \ldots \mid \gamma_n$ y los conjuntos $FIRST(\gamma_i)$ son disjuntos podemos construir el procedimiento para la variable A siguiendo este seudocódigo:

```
A = function() {
  if (lookahead in FIRST(gamma_1)) { imitar gamma_1 }
  else if (lookahead in FIRST(gamma_2)) { imitar gamma_2 }
  ...
  else (lookahead in FIRST(gamma_n)) { imitar gamma_n }
}
```

Donde si γ_j es $X_1 \ldots X_k$ el código `gamma_j` consiste en una secuencia $i = 1 \ldots k$ de llamadas de uno de estos dos tipos:

- Llamar a la subrutina `X_i` si X_i es una variable sintáctica
- Hacer una llamada a `match(X_i)` si X_i es un terminal

4.2.2. Ejercicio: Recorrido del árbol en un ADPR

¿En que forma es recorrido el árbol de análisis sintáctico concreto en un analizador descendente predictivo recursivo? ¿En que orden son visitados los nodos?

4.3. Recursión por la Izquierda

Definición 4.3.1. *Una* gramática es recursiva por la izquierda *cuando existe una derivación* $A \stackrel{*}{\Longrightarrow} A\alpha$.

En particular, es recursiva por la izquierda si contiene una regla de producción de la forma $A \to A\alpha$. *En este caso se dice que la recursión por la izquierda es directa.*

Cuando la gramática es *recursiva por la izquierda*, el método de análisis recursivo descendente predictivo no funciona. En ese caso, el procedimiento `A` asociado con A ciclaría para siempre sin llegar a consumir ningún terminal.

4.4. Esquemas de Traducción

Definición 4.4.1. *Un* esquema de traducción *es una gramática independiente del contexto en la cual se han insertado fragmentos de código en las partes derechas de sus reglas de producción. Los fragmentos de código asi insertados se denominan* acciones semánticas. *Dichos fragmentos actúan, calculan y modifican los atributos asociados con los nodos del árbol sintáctico. El orden en que se evalúan los fragmentos es el de un recorrido primero-profundo del árbol de análisis sintáctico.*

Obsérvese que, en general, para poder aplicar un esquema de traducción hay que construir el árbol sintáctico y después aplicar las acciones empotradas en las reglas en el orden de recorrido primero-profundo. Por supuesto, si la gramática es ambigua una frase podría tener dos árboles y la ejecución de las acciones para ellos podría dar lugar a diferentes resultados. Si se quiere evitar la multiplicidad de resultados (interpretaciones semánticas) es necesario precisar de que árbol sintáctico concreto se esta hablando.

Por ejemplo, si en la regla $A \rightarrow \alpha\beta$ insertamos un fragmento de código:

$$A \rightarrow \alpha\{action\}\beta$$

La acción $\{action\}$ se ejecutará después de todas las acciones asociadas con el recorrido del subárbol de α y antes que todas las acciones asociadas con el recorrido del subárbol β.

El siguiente esquema de traducción recibe como entrada una expresión en infijo y produce como salida su traducción a postfijo para expresiones aritmeticas con sólo restas de números:

$expr \rightarrow expr_1 - NUM$	`{ expr.TRA = expr[1].TRA+" "+NUM.VAL+" - "}`
$expr \rightarrow NUM$	`{ expr.TRA = NUM.VAL }`

Las apariciones de variables sintácticas en una regla de producción se indexan como se ve en el ejemplo, para distinguir de que nodo del árbol de análisis estamos hablando. Cuando hablemos del atributo de un nodo utilizaremos el punto (.). Aquí `VAL` es un atributo de los nodos de tipo NUM denotando su valor numérico y para accederlo escribiremos `NUM.VAL`. Análogamente `expr.TRA` denota el atributo traducción de los nodos de tipo $expr$.

Ejercicio 4.4.1. *Muestre la secuencia de acciones a la que da lugar el esquema de traducción anterior para la frase 7 -5 -4.*

En este ejemplo, el cómputo del atributo `expr.TRA` depende de los atributos en los nodos hijos, o lo que es lo mismo, depende de los atributos de los símbolos en la parte derecha de la regla de producción. Esto ocurre a menudo y motiva la siguiente definición:

Definición 4.4.2. *Un atributo tal que su valor en todo nodo del árbol sintáctico puede ser computado en términos de los atributos de los hijos del nodo se dice que es un* atributo sintetizado.

4.5. Eliminación de la Recursión por la Izquierda en un Esquema de Traducción

La eliminación de la recursión por la izquierda es sólo un paso: debe ser extendida a esquemas de traducción, de manera que no sólo se preserve el lenguaje sino la secuencia de acciones. Supongamos que tenemos un esquema de traducción de la forma:

$A \rightarrow A\alpha$	`{ alpha_action }`
$A \rightarrow A\beta$	`{ beta_action }`
$A \rightarrow \gamma$	`{ gamma_action }`

para una sentencia como $\gamma\beta\alpha$ la secuencia de acciones será:

```
gamma_action  beta_action alpha_action
```

¿Cómo construir un esquema de traducción para la gramática resultante de eliminar la recursión por la izquierda que ejecute las acciones asociadas en el mismo orden?. Supongamos para simplificar, que las acciones no dependen de atributos ni computan atributos, sino que actúan sobre variables globales. En tal caso, la siguiente ubicación de las acciones da lugar a que se ejecuten en el mismo orden:

$A \to \gamma$ `{ gamma_action }` R
$R \to \beta$ `{ beta_action }` R
$R \to \alpha$ `{ alpha_action }` R
$R \to \epsilon$

Si hay atributos en juego, la estrategia para construir un esquema de traducción equivalente para la gramática resultante de eliminar la recursividad por la izquierda se complica.

4.6. Práctica: Analizador Descendente Predictivo Recursivo

Partiendo del analizador sintáctico descendente predictivo recursivo para la gramática descrita en la sección 4.2.1

Donde Puede encontrar la versión de la que partir en

- Despliegue en Heroku: http://predictiveparser.herokuapp.com/
- Repositorio en GitHub: https://github.com/crguezl/prdcalc
- ```
  [~/javascript/PLgrado/predictiveRD/prdcalc(develop)]$ pwd -P
  /Users/casiano/local/src/javascript/PLgrado/predictiveRD/prdcalc
  [~/javascript/PLgrado/predictiveRD/prdcalc(develop)]$ git remote -v
  heroku  git@heroku.com:predictiveparser.git (fetch)
  heroku  git@heroku.com:predictiveparser.git (push)
  origin  git@github.com:crguezl/prdcalc.git (fetch)
  origin  git@github.com:crguezl/prdcalc.git (push)
  ```

Tareas Añada:

- Extienda y modifique el analizador para que acepte el lenguaje descrito por la gramática EBNF del lenguaje PL/0 que se describe en la entrada de la Wikipedia Recursive descent parser. Procure que el arbol generado refleje la asociatividad correcta para las diferencias y las divisiones. No es necesario que el lenguaje sea exactamente igual pero debería ser parecido. Tener los mismos constructos.
-
- Use CoffeeScript para escribir el código (fichero `views/main.coffee`)
- Use slim para las vistas
- Usa Sass para las hojas de estilo
- Despliegue la aplicación en Heroku
- Añada pruebas
  ```

Sinatra
Véase el fichero main.rb.

1. Filters

2. Helpers

   a) El helper `css` es usado en views/layout.slim
   b) El helper `current?` es usado en views/nav.slim para añadir la clase `current` a la página que esta siendo visitada.
   c) El estilo de la entrada de la página actual es modificado en el fichero de estilo views/styles.scss

   ```
   nav a.current {
     background: lighten($black, 50%);
   }
   ```

   El método lighten es proveído por Sass .

3. Views / Templates

Sass
Véase el fichero views/styles.scss.

1. Sass

2. Sass Basics

3. css2sass en GitHub (https://github.com/jpablobr/css2sass) y despliegue en Heroku (http://css2sass.heroku.c

Slim Véanse los ficheros `views/*.slim`:

- views/layout.slim
- views/home.slim
- views/nav.slim

1. slim

2. Slim docs

3. html2slim

4. 2011.12.16 Tech Talk: Slim Templates de Big Nerd Ranch (Vimeo)

CoffeeScript

1. CoffeeScript

2. CoffeeScript Cookbook

3. js2coffee.org

Construyendo Árboles con la Asociatividad Correcta Añadamos el operador `-` al código de nuestra práctica. Para ello, podemos extender nuestro gramática con una regla de producción:

| expression → term '+' expression |term '-' expression | term |

que da lugar a un código como el que sigue:

```
expression = ->
  result = term()
  if lookahead and lookahead.type is "+"
    ....
  if lookahead and lookahead.type is "-"
    match "-"
    right = expression()
    result =
      type: "-"
      left: result
      right: right
```

Cuando le damos como entrada `a = 4-2-1` produce el siguiente AST:

```
{
  "type": "=",
  "left": {
    "type": "ID",
    "value": "a"
  },
  "right": {
    "type": "-",
    "left": {
      "type": "NUM",
      "value": 4
    },
    "right": {
      "type": "-",
      "left": {
        "type": "NUM",
        "value": 2
      },
      "right": {
        "type": "NUM",
        "value": 1
      }
    }
  }
}
```

que se corresponde con esta parentización: `a = (4 - (2 - 1))`

Este árbol no se corresponde con la asociatividad a izquierdas del operador `-`. Es un árbol que refleja una asociación a derechas (`a = 3`).

Ahora bien, el lenguaje generado por dos reglas de la forma: $\begin{array}{ll} A \to A\alpha & \texttt{\{ alpha\_action \}} \\ A \to \gamma & \texttt{\{ gamma\_action \}} \end{array}$ Es $\gamma\alpha*$. Por tanto el método asociado con A podría reescribirse como sigue:

```
A = () ->
  gamma() # imitar gamma
  gamma_action() # acción semántica asociada con gamma
  while lookahead and lookahead.type belongs to  FIRST(alpha)
    alpha() # imitar alpha
    alpha_action()
```

Capítulo 5

Análisis Sintáctico Mediante Precedencia de Operadores en JavaScript

5.1. Ejemplo Simple de Intérprete: Una Calculadora

1. How to write a simple interpreter in JavaScript

5.2. Análisis Top Down Usando Precedencia de Operadores

1. Véase el libro [3] Beautiful Code: Leading Programmers Explain How They Think, Capítulo 9.
2. Top Down Operator Precedence por Douglas Crockford
3. Top Down Operator Precedence demo por Douglas Crockford
4. jslint
5. David Majda - Easy parsing with PEG.js

5.2.1. Gramática de JavaScript

1. Especificación de JavaScript 1997
2. NQLL(1) grammar (Not Quite LL(1)) for JavaScrip 1997
3. Postscript con la especificación de JavaScript 1997
4. Mozilla JavaScript Language Resources
5. JavaScript 1.4 LR(1) Grammar 1999.
6. Apple JavaScript Core Specifications
7. Creating a JavaScript Parser Una implementación de ECAMScript 5.1 usando Jison disponible en GitHub en https://github.com/cjihrig/jsparser.

Capítulo 6

Análisis Descendente mediante Parsing Expresion Grammars en JavaScript

6.1. Introducción a los PEGs

In computer science, a *parsing expression grammar*, or *PEG*, is a type of analytic formal grammar, i.e. it describes a formal language in terms of a set of rules for recognizing strings in the language.

The formalism was introduced by Bryan Ford in 2004 and is closely related to the family of top-down parsing languages introduced in the early 1970s.

Syntactically, PEGs also look similar to context-free grammars (CFGs), but they have a different interpretation:

- the choice operator selects the first match in PEG, while it is ambiguous in CFG.
- This is closer to how string recognition tends to be done in practice, e.g. by a recursive descent parser.

Unlike CFGs, PEGs cannot be ambiguous; *if a string parses, it has exactly one valid parse tree.*

It is conjectured that there exist context-free languages that cannot be parsed by a PEG, but this is not yet proven.

6.1.1. Syntax

Formally, a parsing expression grammar consists of:

- A finite set N of nonterminal symbols.
- A finite set Σ of terminal symbols that is disjoint from N.
- A finite set P of parsing rules.
- An expression e_S termed the starting expression.

Each parsing rule in P has the form $A \leftarrow e$, where A is a nonterminal symbol and e is a *parsing expression*.

A parsing expression is a hierarchical expression similar to a regular expression, which is constructed in the following fashion:

1. An atomic parsing expression consists of:
    a) any terminal symbol,
    b) any nonterminal symbol, or
    c) the empty string ϵ.

2. Given any existing parsing expressions e, e_1, and e_2, a new parsing expression can be constructed using the following operators:

   a) Sequence: $e1e2$

   b) Ordered choice: $e1/e2$

   c) Zero-or-more: $e*$

   d) One-or-more: $e+$

   e) Optional: $e?$

   f) And-predicate: $\&e$

   g) Not-predicate: $!e$

6.1.2. Semantics

The fundamental difference between context-free grammars and parsing expression grammars is that the PEG's choice operator is ordered:

1. If the first alternative succeeds, the second alternative is ignored.

2. Thus ordered choice is not commutative, unlike unordered choice as in context-free grammars.

3. The consequence is that if a CFG is transliterated directly to a PEG, any ambiguity in the former is resolved by deterministically picking one parse tree from the possible parses.

4. By carefully choosing the order in which the grammar alternatives are specified, a programmer has a great deal of control over which parse tree is selected.

5. PEGs can look ahead into the input string without actually consuming it

6. The and-predicate expression $\&e$ invokes the sub-expression e, and then succeeds if e succeeds and fails if e fails, *but in either case never consumes any input.*

7. The not-predicate expression $!e$ succeeds if e fails and fails if e succeeds, *again consuming no input in either case.*

6.1.3. Implementing parsers from parsing expression grammars

Any parsing expression grammar can be converted directly into a *recursive descent parser*.

Due to the unlimited lookahead capability that the grammar formalism provides, however, the resulting parser could exhibit exponential time performance in the worst case.

It is possible to obtain better performance for any parsing expression grammar by converting its recursive descent parser into a packrat parser, which always runs in linear time, at the cost of substantially greater storage space requirements.

A packrat parser is a form of parser similar to a recursive descent parser in construction, except that during the parsing process [redacted] *the intermediate results of all invocations of the mutually recursive parsing functions*, ensuring that each parsing function is only invoked at most once at a given input position.

Because of this memoization, a packrat parser has the ability to parse many context-free grammars and any parsing expression grammar (including some that do not represent context-free languages) in linear time.

Examples of memoized recursive descent parsers are known from at least as early as 1993.

Note that this analysis of the performance of a packrat parser **assumes that enough memory is available to hold all of the memoized results**; in practice, if there were not enough memory, some parsing functions might have to be invoked more than once at the same input position, and consequently the parser could take more than linear time.

It is also possible to build LL parsers and LR parsers from parsing expression grammars, with better worst-case performance than a recursive descent parser, but the unlimited lookahead capability of the grammar formalism is then lost. Therefore, not all languages that can be expressed using parsing expression grammars can be parsed by LL or LR parsers.

6.1.4. Lexical Analysis

Parsers for languages expressed as a CFG, such as LR parsers, require a separate tokenization step to be done first, which breaks up the input based on the location of spaces, punctuation, etc.

The tokenization is necessary because of the way these parsers use lookahead to parse CFGs that meet certain requirements in linear time.

PEGs do not require tokenization to be a separate step, and tokenization rules can be written in the same way as any other grammar rule.

6.1.5. Left recursion

PEGs cannot express left-recursive rules where a rule refers to itself without moving forward in the string. For example, the following left-recursive CFG rule:

```
string-of-a -> string-of-a 'a' | 'a'
```

can be rewritten in a PEG using the plus operator:

```
string-of-a <- 'a'+
```

The process of rewriting indirectly left-recursive rules is complex in some packrat parsers, especially when semantic actions are involved.

6.1.6. Referencias y Documentación

- Véase Parsing Expression Grammar
- PEG.js documentation
- Testing PEG.js Online
- Michael's Blog: JavaScript Parser Generators. The PEG.js Tutorial
- The Packrat Parsing and Parsing Expression Grammars Page
- PL101: Create Your Own Programming Language. Véanse [4] y [5]
- PL101: Create Your Own Programming Language: Parsing

6.2. PEGJS

What is

PEG.js is a parser generator for JavaScript that produces parsers.

PEG.js generates a parser from a Parsing Expression Grammar describing a language.

We can specify what the parser returns (using semantic actions on matched parts of the input).

Installation

To use the pegjs command, install PEG.js globally:

```
$ npm install -g pegjs
```

To use the JavaScript API, install PEG.js locally:

```
$ npm install pegjs
```

To use it from the browser, download the PEG.js library (regular or minified version).

El compilador de línea de comandos

```
[~/srcPLgrado/pegjs/examples(master)]$ pegjs --help
Usage: pegjs [options] [--] [<input_file>] [<output_file>]

Generates a parser from the PEG grammar specified in the <input_file> and writes
it to the <output_file>.

If the <output_file> is omitted, its name is generated by changing the
<input_file> extension to ".js". If both <input_file> and <output_file> are
omitted, standard input and output are used.

Options:
  -e, --export-var <variable>        name of the variable where the parser
                                     object will be stored (default:
                                     "module.exports")
      --cache                        make generated parser cache results
      --allowed-start-rules <rules>  comma-separated list of rules the generated
                                     parser will be allowed to start parsing
                                     from (default: the first rule in the
                                     grammar)
  -o, --optimize <goal>              select optimization for speed or size
                                     (default: speed)
      --plugin <plugin>              use a specified plugin (can be specified
                                     multiple times)
      --extra-options <options>      additional options (in JSON format) to pass
                                     to PEG.buildParser
      --extra-options-file <file>    file with additional options (in JSON
                                     format) to pass to PEG.buildParser
  -v, --version                      print version information and exit
  -h, --help                         print help and exit
```

Using it

```
[~/srcPLgrado/pegjs/examples(master)]$ node
> PEG = require("pegjs")
{ VERSION: '0.8.0',
  GrammarError: [Function],
  parser:
   { SyntaxError: [Function: SyntaxError],
     parse: [Function: parse] },
  compiler:
   { passes:
      { check: [Object],
        transform: [Object],
        generate: [Object] },
     compile: [Function] },
  buildParser: [Function] }

> parser = PEG.buildParser("start = ('a' / 'b')+")
{ SyntaxError: [Function: SyntaxError],
  parse: [Function: parse] }
```

Using the generated parser is simple — just call its `parse` method and pass an input string as a parameter.

The method will return

- a parse result or
- throw an exception if the input is invalid.

You can tweak parser behavior by passing a second parameter with an `options` object to the `parse` method.

Only one option is currently supported:

`startRule` which is the name of the rule to start parsing from.

```
> parser.parse("abba");
[ 'a', 'b', 'b', 'a' ]
>
```

Opciones: allowedStartRules Specifying allowedStartRules we can set the rules the parser will be allowed to start parsing from (default: the first rule in the grammar).

```
[~/srcPLgrado/pegjs/examples(master)]$ cat allowedstartrules.js
var PEG = require("pegjs");
var grammar = "a = 'hello' b\nb = 'world'"; //"a = 'hello' b\nb='world';
console.log(grammar);

var parser = PEG.buildParser(grammar,{ allowedStartRules: ['a', 'b'] });
var r = parser.parse("helloworld", { startRule: 'a' });
console.log(r); // [ 'hello', 'world' ]
r = parser.parse("helloworld")
console.log(r); // [ 'hello', 'world' ]

r =  parser.parse("world", { startRule: 'b' })
console.log(r); // 'world'

try {
  r = parser.parse("world"); // Throws an exception
}
catch(e) {
  console.log("Error!!!!");
  console.log(e);
}

[~/srcPLgrado/pegjs/examples(master)]$ node allowedstartrules.js
a = 'hello' b
b = 'world'
[ 'hello', 'world' ]
[ 'hello', 'world' ]
world
Error!!!!
{ message: 'Expected "hello" but "w" found.',
  expected: [ { type: 'literal', value: 'hello', description: '"hello"' } ],
  found: 'w',
  offset: 0,
  line: 1,
  column: 1,
  name: 'SyntaxError' }
```

The exception contains

- message
- expected,
- found
- offset,
- line,
- column,
- name

and properties with more details about the error.

Opciones: output

When `output` is set to parser, the method will return generated parser object; if set to source, it will return parser source code as a string (default: `parser`).

```
> PEG = require("pegjs")
> grammar = "a = 'hello' b\nb='world'"
'a = \'hello\' b\nb=\'world\''
> console.log(grammar)
a = 'hello' b
b='world'
undefined
> parser = PEG.buildParser(grammar,{ output: "parse"})
undefined
> parser = PEG.buildParser(grammar,{ output: "source"})
> typeof parser
'string'
> console.log(parser.substring(0,100))
(function() {
  /*
   * Generated by PEG.js 0.8.0.
   *
   * http://pegjs.majda.cz/
   */
```

Opciones: plugin La opción plugins indica que plugin se van a usar.

```
[~/srcPLgrado/pegjs/examples(master)]$ cat plugin.coffee
#!/usr/bin/env coffee
PEG = require 'pegjs'
coffee = require 'pegjs-coffee-plugin'
grammar = """
a = 'hello' _ b { console.log 1 }
b = 'world'     { console.log 2 }
_ = [ \t]+      { console.log 3 }
"""
parser = PEG.buildParser grammar, plugins: [coffee]
r = parser.parse "hello world"
```

```
[~/srcPLgrado/pegjs/examples(master)]$ ./plugin.coffee
3
2
1
```

cache

If `true`, makes the parser cache results, avoiding exponential parsing time in pathological cases but making the parser slower (default: `false`).

optimize

Selects between optimizing the generated parser for parsing speed (`speed`) or code size (`size`) (default: `speed`).

6.3. Un Ejemplo Sencillo

Donde

```
[~/srcPLgrado/pegjs/examples(master)]$ pwd -P
/Users/casiano/local/src/javascript/PLgrado/pegjs/examples
[~/srcPLgrado/pegjs/examples(master)]$ git remote -v
dmajda  https://github.com/dmajda/pegjs.git (fetch)
dmajda  https://github.com/dmajda/pegjs.git (push)
origin  git@github.com:crguezl/pegjs.git (fetch)
origin  git@github.com:crguezl/pegjs.git (push)
```

https://github.com/crguezl/pegjs/blob/master/examples/arithmetics.pegjs

arithmetics.pegjs

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat arithmetics.pegjs
/*
 * Classic example grammar, which recognizes simple arithmetic expressions like
 * "2*(3+4)". The parser generated from this grammar then computes their value.
 */

start
  = additive

additive
  = left:multiplicative PLUS right:additive { return left + right; }
  / left:multiplicative MINUS right:additive { return left - right; }
  / multiplicative

multiplicative
  = left:primary MULT right:multiplicative { return left * right; }
  / left:primary DIV right:multiplicative { return left / right; }
  / primary

primary
  = integer
  / LEFTPAR additive:additive RIGHTPAR { return additive; }

integer "integer"
  = NUMBER

_ = $[ \t\n\r]*

PLUS = _"+"_
MINUS = _"-"_
```

```
MULT = _"*"_
DIV = _"/"_
LEFTPAR = _"("_
RIGHTPAR = _")"_
NUMBER = _ digits:$[0-9]+ _ { return parseInt(digits, 10); }
```

There are several types of parsing expressions, some of them containing subexpressions and thus forming a recursive structure:

- `expression *`

  Match zero or more repetitions of the expression and return their match results in an array. The matching is greedy, i.e. the parser tries to match the expression as many times as possible.

- `expression +`

  Match one or more repetitions of the expression and return their match results in an array. The matching is greedy, i.e. the parser tries to match the expression as many times as possible.

- `$ expression`

  Try to match the expression. If the match succeeds, return the matched string instead of the match result.

main.js

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat main.js
var PEG = require("./arithmetics.js");
var r = PEG.parse("(2+9-1)/2");
console.log(r);
```

Rakefile

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat Rakefile
PEGJS = "../bin/pegjs"
task :default => :run

desc "Compile arithmetics.pegjs"
task :compile do
  sh "#{PEGJS} arithmetics.pegjs"
end

desc "Run and use the parser generated from arithmetics.pegjs"
task :run => :compile do
  sh "node main.js"
end
```

Compilación

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ rake
../bin/pegjs arithmetics.pegjs
node main.js
5
```

6.3.1. Asociación Incorrecta para la Resta y la División

Definición 6.3.1. *Una* gramática es recursiva por la izquierda *cuando existe una derivación* $A \stackrel{*}{\Longrightarrow} A\alpha$.

En particular, es recursiva por la izquierda si contiene una regla de producción de la forma $A \rightarrow A\alpha$. *En este caso se dice que la recursión por la izquierda es directa.*

Cuando la gramática es *recursiva por la izquierda*, el método de análisis recursivo descendente predictivo no funciona. En ese caso, el procedimiento `A` asociado con A ciclaría para siempre sin llegar a consumir ningún terminal.

Es por eso que hemos escrito las reglas de la calculadora con recursividad a derechas,

```
additive
  = left:multiplicative PLUS right:additive { return left + right; }
  / left:multiplicative MINUS right:additive { return left - right; }
  / multiplicative

multiplicative
  = left:primary MULT right:multiplicative { return left * right; }
  / left:primary DIV right:multiplicative { return left / right; }
  / primary
```

pero eso da lugar a árboles hundidos hacia la derecha y a una aplicación de las reglas semánticas errónea:

```
[~/pegjs/examples(master)]$ cat main.js
var PEG = require("./arithmetics.js");
var r = PEG.parse("5-3-2");
console.log(r);

[~/pegjs/examples(master)]$ node main.js
4
```

6.4. Acciones Intermedias

Supongamos que queremos poner una acción semántica intermedia en un programa PEG.js :

```
[~/srcPLgrado/pegjs/examples(master)]$ cat direct_intermedia.pegjs
a = 'a'+ { console.log("acción intermedia"); } 'b'+ {
                      console.log("acción final");
                      return "hello world!";
                    }
```

Al compilar nos da un mensjae de error:

```
[~/srcPLgrado/pegjs/examples(master)]$ pegjs direct_intermedia.pegjs
1:48: Expected "/", ";", end of input or identifier but "'" found.
```

La solución consiste en introducir una variable sintáctica en medio que derive a la palabra vacía y que tenga asociada la correspondiente acción semántica:

```
[~/srcPLgrado/pegjs/examples(master)]$ cat intermedia.pegjs
a = 'a'+ temp 'b'+ {
                      console.log("acción final");
                      return "hello world!";
                    }
temp = { console.log("acción intermedia"); }
```

Este es el progrma que usa el parser generado:

```
[~/srcPLgrado/pegjs/examples(master)]$ cat main_intermedia.js
var parser = require("intermedia");
var input = process.argv[2] || 'aabb';
var result = parser.parse(input);
console.log(result);
```

al ejecutar tenemos:

```
[~/srcPLgrado/pegjs/examples(master)]$ pegjs intermedia.pegjs
[~/srcPLgrado/pegjs/examples(master)]$ node main_intermedia.js
acción intermedia
acción final
hello world!
```

6.5. PegJS en los Browser

Donde

- ```
  ~/srcPLgrado/pegjs/examples(master)]$ pwd -P
  /Users/casiano/local/src/javascript/PLgrado/pegjs/examples
  ```
- ```
  [~/srcPLgrado/pegjs/examples(master)]$ git remote -v
  dmajda  https://github.com/dmajda/pegjs.git (fetch)
  dmajda  https://github.com/dmajda/pegjs.git (push)
  origin  git@github.com:crguezl/pegjs.git (fetch)
  origin  git@github.com:crguezl/pegjs.git (push)
  ```
- https://github.com/crguezl/pegjs/tree/master/examples
-

Versiones para Browser Podemos usar directamente las versiones para los browser:

- PEG.js — minified
- PEG.js — development

La opción -e de pegjs

```
[~/Dropbox/src/javascript/PLgrado/jison]$ pegjs --help
Usage: pegjs [options] [--] [<input_file>] [<output_file>]

Generates a parser from the PEG grammar specified in the <input_file> and
writes it to the <output_file>.

If the <output_file> is omitted, its name is generated by changing the
<input_file> extension to ".js". If both <input_file> and <output_file> are
omitted, standard input and output are used.

Options:
  -e, --export-var <variable>  name of the variable where the parser object
                               will be stored (default: "module.exports")
      --cache                  make generated parser cache results
      --track-line-and-column  make generated parser track line and column
  -v, --version                print version information and exit
  -h, --help                   print help and exit
```

Compilación
Le indicamos que el parser se guarde en `calculator`:

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ rake web
../bin/pegjs -e calculator arithmetics.pegjs
```

```
[~/srcPLgrado/pegjs/examples(master)]$ head -5 arithmetics.js
calculator = (function() {
  /*
   * Generated by PEG.js 0.7.0.
   *
   * http://pegjs.majda.cz/
```

calc.js Ahora, desde el JavaScript que llama al parser accedemos al objeto mediante la variable `calculator`:

```
[~/srcPLgrado/pegjs/examples(master)]$ cat calc.js
$(document).ready(function() {
  $('#eval').click(function() {
    try {
      var result = calculator.parse($('#input').val());
      $('#output').html(result);
    } catch (e) {
      $('#output').html('<div class="error"><pre>\n' + String(e) + '\n</pre></div>');
    }
  });

  $("#examples").change(function(ev) {
    var f = ev.target.files[0];
    var r = new FileReader();
    r.onload = function(e) {
      var contents = e.target.result;

      input.innerHTML = contents;
    }
    r.readAsText(f);
  });

});
```

arithmetic.pegjs El PEG describe una calculadora:

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat arithmetics.pegjs
/*
 * Classic example grammar, which recognizes simple arithmetic expressions like
 * "2*(3+4)". The parser generated from this grammar then computes their value.
 */

start
  = additive

additive
  = left:multiplicative PLUS right:additive { return left + right; }
  / left:multiplicative MINUS right:additive { return left - right; }
  / multiplicative
```

```
multiplicative
  = left:primary MULT right:multiplicative { return left * right; }
  / left:primary DIV right:multiplicative { return left / right; }
  / primary

primary
  = integer
  / LEFTPAR additive:additive RIGHTPAR { return additive; }

integer "integer"
  = NUMBER

_ = $[ \t\n\r]*

PLUS = _"+"_
MINUS = _"-"_
MULT = _"*"_
DIV = _"/"_
LEFTPAR = _"("_
RIGHTPAR = _")"_
NUMBER = _ digits:$[0-9]+ _ { return parseInt(digits, 10); }
```

calculator.html

```
[~/srcPLgrado/pegjs/examples(master)]$ cat calculator.html
<!DOCTYPE HTML>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>pegjs</title>
    <link rel="stylesheet" href="global.css" type="text/css" media="screen" charset="utf-8" />
  </head>
  <body>
    <h1>pegjs</h1>
    <div id="content">
      <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script
      <script src="arithmetics.js"></script>
      <script src="calc.js"></script>

      <p>
      Load an example:
      <input type="file" id="examples" />
      </p>

      <p>
      <table>
        <tr>
          <td>
            <textarea id="input" autofocus cols = "40" rows = "4">2+3*4</textarea>
          </td>
          <td class="output">
            <pre>
              <span id="output"></span> <!-- Output goes here! -->
```

```
          </pre>
        </td>
        <td><button id="eval" type="button">eval</button></td>
      </tr>
    </table>
    </p>
  </div>
</body>
</html>
```

Figura 6.1: pegjs en la web

6.6. Eliminación de la Recursividad por la Izquierda en PEGs

Donde

- `[~/srcPLgrado/pegjs-coffee-plugin/examples(master)]$ pwd -P`
  `/Users/casiano/local/src/javascript/PLgrado/pegjs-coffee-plugin/examples`

- `[~/srcPLgrado/pegjs-coffee-plugin/examples(master)]$ git remote -v`
  ```
  dignifiedquire  git@github.com:Dignifiedquire/pegjs-coffee-plugin.git (fetch)
  dignifiedquire  git@github.com:Dignifiedquire/pegjs-coffee-plugin.git (push)
  origin  git@github.com:crguezl/pegjs-coffee-plugin.git (fetch)
  origin  git@github.com:crguezl/pegjs-coffee-plugin.git (push)
  ```

- https://github.com/crguezl/pegjs-coffee-plugin/tree/master/examples

PEGjs Coffee Plugin PEGjs Coffee Plugin is a plugin for PEG.js to use CoffeeScript in actions.

Veamos un ejemplo de uso via la API:

```
[~/srcPLgrado/pegjs/examples(master)]$ cat plugin.coffee
#!/usr/bin/env coffee
PEG = require 'pegjs'
coffee = require 'pegjs-coffee-plugin'
grammar = """
a = 'hello' _ b { console.log 1; "hello world!" }
b = 'world'     { console.log 2 }
_ = [ \t]+      { console.log 3 }
```

```
"""
parser = PEG.buildParser grammar, plugins: [coffee]
r = parser.parse "hello world"
console.log(r)

[~/srcPLgrado/pegjs/examples(master)]$ coffee plugin.coffee
3
2
1
hello world!
```

Un Esquema de Traducción Recursivo por la Izquierda Consideremos el siguiente esquema de traducción implementado en Jison :

```
[~/srcPLgrado/pegjs-coffee-plugin/examples(master)]$ cat leftrec.jison
/*
Exercise: Find a PEG equivalent to the following left-recursive
grammar:
*/
%lex
%%

\s+                { /* skip whitespace */ }
y                  { return 'y';}
.                  { return 'x';}

/lex

%{
  do_y = function(y)   { console.log("A -> 'y'   do_y("+y+")"); return y; }
  do_x = function(a, x){ console.log("A -> A 'x' do_x("+a+", "+x+")"); return a+x; }
%}

%%
A : A 'x' { $$ = do_x($1, $2); }
  | 'y'   { $$ = do_y($1); }
;

[~/srcPLgrado/pegjs-coffee-plugin/examples(master)]$ jison leftrec.jison
[~/srcPLgrado/pegjs-coffee-plugin/examples(master)]$ ls -ltr leftrec.j*
-rw-r--r--  1 casiano  staff    441 18 mar 20:22 leftrec.jison
-rw-r--r--  1 casiano  staff  20464 18 mar 20:34 leftrec.js

[~/srcPLgrado/pegjs-coffee-plugin/examples(master)]$ cat main_leftrec.js
var parser = require('./leftrec');
input = "y x x x";
var r = parser.parse(input);

[~/srcPLgrado/pegjs-coffee-plugin/examples(master)]$ node main_leftrec.js
A -> 'y'   do_y(y)
A -> A 'x' do_x(y, x)
A -> A 'x' do_x(yx, x)
A -> A 'x' do_x(yxx, x)
```

Métodología

Es posible modificar la gramática para eliminar la recursión por la izquierda. En este apartado nos limitaremos al caso de recursión por la izquierda directa. La generalización al caso de recursión por la izquierda no-directa se reduce a la iteración de la solución propuesta para el caso directo.

Consideremos una variable A con dos producciones:

$$A \to A\alpha |\ \beta$$

donde $\alpha, \beta \in (V \cup \Sigma)^*$ no comienzan por A. Estas dos producciones pueden ser sustituidas por:

$$A \to \beta\alpha*$$

eliminando así la recursión por la izquierda.

Solución

```
[~/pegjs-coffee-remove-left(master)]$ cat -n remove_left_recursive.pegjs
     1  /*
     2
     3  Exercise: Find a PEG equivalent to the following left-recursive
     4  grammar:
     5
     6  A : A 'x' { $$ = do_x($1, $2); } | 'y' { $$ = do_y($1); }
     7
     8  */
     9
    10  {
    11    @do_y = (y)   -> console.log("do_y(#{y})"); y
    12    @do_x = (a, x)-> console.log("do_x(#{a}, #{x})"); a+x
    13  }
    14
    15  A = y:'y' xs:('x'*)
    16       {
    17           a = @do_y(y)
    18           for x in xs
    19             a = @do_x(a, x)
    20           a
    21       }
```

```
[~/pegjs-coffee-remove-left(master)]$ pegjs --plugin pegjs-coffee-plugin remove_left_recursive
[~/pegjs-coffee-remove-left(master)]$ ls -ltr | tail -1
-rw-rw-r--  1 casiano  staff   8919  3 jun 10:42 remove_left_recursive.js
```

```
[~/pegjs-coffee-remove-left(master)]$ cat use_remove_left.coffee
PEG = require("./remove_left_recursive.js")
inputs = [
           "yxx"
           "y"
           "yxxx"
         ]

for input in inputs
  console.log("input = #{input}")
  r = PEG.parse input
  console.log("result = #{r}\n")
```

```
[~/pegjs-coffee-remove-left(master)]$ coffee use_remove_left.coffee
input = yxx
do_y(y)
do_x(y, x)
do_x(yx, x)
result = yxx

input = y
do_y(y)
result = y

input = yxxx
do_y(y)
do_x(y, x)
do_x(yx, x)
do_x(yxx, x)
result = yxxx
```

6.7. Eliminando la Recursividad por la Izquierda en la Calculadora

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat simple.pegjs
/* From the Wikipedia
Value   ← [0-9]+ / '(' Expr ')'
Product ← Value (('*' / '/') Value)*
Sum     ← Product (('+' / '-') Product)*
Expr    ← Sum
*/
{
  function reduce(left, right) {
    var sum = left;
    // console.log("sum = "+sum);
    for(var i = 0; i < right.length;i++) {
      var t = right[i];
      var op = t[0];
      var num = t[1];
      switch(op) {
        case '+' : sum += num; break;
        case '-' : sum -= num; break;
        case '*' : sum *= num; break;
        case '/' : sum /= num; break;
        default : console.log("Error! "+op);
      }
      // console.log("sum = "+sum);
    }
    return sum;
  }
}

sum    = left:product right:($[+-] product)* { return reduce(left, right); }
product = left:value right:($[*/] value)*   { return reduce(left, right); }
value   = number:$[0-9]+                    { return parseInt(number,10); }
        / '(' sum:sum ')'                   { return sum; }
```

Es posible especificar mediante llaves un código que este disponible dentro de las acciones semánticas.

Ejecución:

```
[~/pegjs/examples(master)]$ cat use_simple.js
var PEG = require("./simple.js");
var r = PEG.parse("2-3-4");
console.log(r);

[~/pegjs/examples(master)]$ node use_simple.js
-5
```

Veamos otra ejecución:

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat use_simple.js
var PEG = require("./simple.js");
var r = PEG.parse("2+3*(2+1)-10/2");
console.log(r);

[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ ../bin/pegjs simple.pegjs
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ node use_simple.js
6
```

6.8. Eliminación de la Recursividad por la Izquierda y Atributos Heredados

La sección anterior da una forma sencilla de resolver el problema respetando la semántica. Si no se dispone de operadores de repetición la cosa se vuelve mas complicada. Las siguientes secciones muestran una solución para transformar un esquema de traducción recursivo por la izquierda en otro no recursivo por la izquierda respetando el orden en el que se ejecutan las acciones semánticas. Por último se ilustra como se puede aplicar esta técnica en `pegjs` (aunque obviamente es mucho mejor usar la ilustrada anteriormente).

6.8.1. Eliminación de la Recursión por la Izquierda en la Gramática

Es posible modificar la gramática para eliminar la recursión por la izquierda. En este apartado nos limitaremos al caso de recursión por la izquierda directa. La generalización al caso de recursión por la izquierda no-directa se reduce a la iteración de la solución propuesta para el caso directo.

Consideremos una variable A con dos producciones:

$$A \to A\alpha \mid \beta$$

donde $\alpha, \beta \in (V \cup \Sigma)^*$ no comienzan por A. Estas dos producciones pueden ser sustituidas por:

$$\begin{array}{l} A \to \beta R \\ R \to \alpha R \mid \epsilon \end{array}$$

eliminando así la recursión por la izquierda.

Definición 6.8.1. *La producción $R \to \alpha R$ se dice* recursiva por la derecha*.*

Las producciones recursivas por la derecha dan lugar a árboles que se hunden hacia la derecha. Es mas difícil traducir desde esta clase de árboles operadores como el menos, que son asociativos a izquierdas.

Ejercicio 6.8.1. *Elimine la recursión por la izquierda de la gramática*

$expr \rightarrow expr - NUM$
$expr \rightarrow NUM$

6.8.2. Eliminación de la Recursión por la Izquierda en un Esquema de Traducción

La eliminación de la recursión por la izquierda es sólo un paso: debe ser extendida a esquemas de traducción, de manera que no sólo se preserve el lenguaje sino la secuencia de acciones. Supongamos que tenemos un esquema de traducción de la forma:

$A \rightarrow A\alpha$ `{ alpha_action }`
$A \rightarrow A\beta$ `{ beta_action }`
$A \rightarrow \gamma$ `{ gamma_action }`

para una sentencia como $\gamma\beta\alpha$ la secuencia de acciones será:

```
gamma_action  beta_action alpha_action
```

¿Cómo construir un esquema de traducción para la gramática resultante de eliminar la recursión por la izquierda que ejecute las acciones asociadas en el mismo orden?. Supongamos para simplificar, que las acciones no dependen de atributos ni computan atributos, sino que actúan sobre variables globales. En tal caso, la siguiente ubicación de las acciones da lugar a que se ejecuten en el mismo orden:

$A \rightarrow \gamma$ `{ gamma_action }` R
$R \rightarrow \beta$ `{ beta_action }` R
$R \rightarrow \alpha$ `{ alpha_action }` R
$R \rightarrow \epsilon$

Si hay atributos en juego, la estrategia para construir un esquema de traducción equivalente para la gramática resultante de eliminar la recursividad por la izquierda se complica. Consideremos de nuevo el esquema de traducción de infijo a postfijo de expresiones aritméticas de restas:

$expr \rightarrow expr_1 - NUM$ `{ $expr{T} = $expr[1]{T}." ".$NUM{VAL}." - "}`
$expr \rightarrow NUM$ `{ $expr{T} = $NUM{VAL} }`

En este caso introducimos un atributo `H` para los nodos de la clase r el cuál acumula la traducción a postfijo hasta el momento. Observe como este atributo se computa en un nodo r a partir del correspondiente atributo del el padre y/o de los hermanos del nodo:

$expr \rightarrow NUM$ `{ $r{H} = $NUM{VAL} }` r `{ $expr{T} = $r{T} }`
$r \rightarrow -NUM$ `{ $r_1{H} = $r{H}." ".$NUM{VAL}." - " }` r_1 `{ $r{T} = $r_1{T} }`
$r \rightarrow \epsilon$ `{ $r{T} = $r{H} }`

El atributo `H` es un ejemplo de atributo heredado.

6.8.3. Eliminación de la Recursividad por la Izquierda en PEGJS

PegJS no permite acciones intermedias. Tampoco se puede acceder al atributo de la parte izquierda. Por eso, a la hora de implantar la solución anterior debemos introducir variables sintácticas temporales que produzcan la palabra vacía y que vayan acompañadas de la acción semántica correspondiente.

Además nos obliga a usar variables visibles por todas las reglas semánticas para emular el acceso a los atributos de la parte izquierda de una regla de proudcción.

El siguiente ejemplo ilustra como eliminar la reucrusión por la izquierda respetando la asociatividad de la oepración de diferencia:

```
[~/pegjs/examples(master)]$ cat inherited.pegjs
{
  var h = 0, number = 0;
}
e = NUMBER aux1 r          { return h; }
aux1 = /* empty */         { h = number; }

r =   '-' NUMBER  aux2 r  { return h; }
    / /* empty */
aux2 = /* empty */         { h -= number; }

NUMBER = _ digits:$[0-9]+ _  { number = parseInt(digits, 10); return number; }

_ = $[ \t\n\r]*

[~/pegjs/examples(master)]$ cat use_inherited.js
var PEG = require("./inherited.js");
var r = PEG.parse("2-1-1");
console.log(r);

var r = PEG.parse("4-2-1");
console.log(r);

var r = PEG.parse("2-3-1");
console.log(r);


[~/pegjs/examples(master)]$ pegjs inherited.pegjs
Referenced rule "$" does not exist.
[~/pegjs/examples(master)]$ ../bin/pegjs inherited.pegjs
[~/pegjs/examples(master)]$ node use_inherited.js
0
1
-2
```

6.9. Dangling else: Asociando un else con su if mas cercano

The dangling else is a problem in computer programming in which an optional `else clause` in an `If{then({else)` statement results in nested conditionals being ambiguous.

Formally, the reference context-free grammar of the language is ambiguous, meaning there is more than one correct parse tree.

In many programming languages one may write conditionally executed code in two forms:

the `if-then` form, and the `if-then-else` form – the `else` clause is optional:

```
if a then s
if a then s1 else s2
```

This gives rise to an ambiguity in interpretation when there are nested statements, specifically whenever an `if-then` form appears as `s1` in an `if-then-else` form:

```
if a then if b then s else s2
```

In this example, `s` is unambiguously executed when `a` is `true` and `b` is `true`, but one may interpret `s2` as being executed when `a` is `false`

- (thus attaching the `else` to the first `if`) or when

- `a` is `true` and `b` is `false` (thus attaching the `else` to the second `if`).

In other words, one may see the previous statement as either of the following expressions:

```
if a then (if b then s) else s2
```

or

```
if a then (if b then s else s2)
```

This is a problem that often comes up in compiler construction, especially scannerless parsing.

The convention when dealing with the dangling `else` is to attach the `else` to the nearby `if` statement.

Programming languages like Pascal and C follow this convention, so there is no ambiguity in the semantics of the language, though the use of a parser generator may lead to ambiguous grammars. In these cases alternative grouping is accomplished by explicit blocks, such as `begin...end` in Pascal and `{...}` in C.

Here follows a solution in PEG.js:

danglingelse.pegjs

```
$ cat danglingelse.pegjs
/*
S ← 'if' C 'then' S 'else' S / 'if' C 'then' S
*/

S =   if C:C then S1:S else S2:S { return [ 'ifthenelse', C, S1, S2 ]; }
    / if C:C then S:S            { return [ 'ifthen', C, S]; }
    / O                          { return 'O'; }
_ = ' '*
C = _'c'_                        { return 'c'; }
O = _'o'_                        { return 'o'; }
else = _'else'_
if = _'if'_
then = _'then'_
```

use_danglingelse.js

```
$ cat use_danglingelse.js
var PEG = require("./danglingelse.js");
var r = PEG.parse("if c then if c then o else o");
console.log(r);
```

Ejecución

```
$ ../bin/pegjs danglingelse.pegjs
$ node use_danglingelse.js
[ 'ifthen', 'c', [ 'ifthenelse', 'c', 'O', 'O' ] ]
```

Donde

- `[~/srcPLgrado/pegjs/examples(master)]$ pwd -P`
  `/Users/casiano/local/src/javascript/PLgrado/pegjs/examples`

- ```
  [~/srcPLgrado/pegjs/examples(master)]$ git remote -v
  dmajda  https://github.com/dmajda/pegjs.git (fetch)
  dmajda  https://github.com/dmajda/pegjs.git (push)
  origin  git@github.com:crguezl/pegjs.git (fetch)
  origin  git@github.com:crguezl/pegjs.git (push)
  ```
- https://github.com/crguezl/pegjs/tree/master/examples

6.10. Not Predicate: Comentarios Anidados

The following recursive PEG.js program matches Pascal-style nested comment syntax:

```
(* which can (* nest *) like this *)
```

Pascal_comments.pegjs

```
$ cat pascal_comments.pegjs
/* Pascal nested comments */

P     =   prog:N+                          { return prog; }
N     =   chars:$(!Begin ANY)+             { return chars;}
        / C
C     = Begin chars:T* End                 { return chars.join(''); }
T     =   C
        / (!Begin !End char:ANY)           { return char;}
Begin = '(*'
End   = '*)'
ANY   =   'z'    /* any character */       { return 'z';  }
        / char:[^z]                        { return char; }
```

use_pascal_comments.js

```
$ cat use_pascal_comments.js
var PEG = require("./pascal_comments.js");
var r = PEG.parse(
  "not bla bla (* pascal (* nested *) comment *)"+
  " pum pum (* another comment *)");
console.log(r);
```

Ejecución

```
$ ../bin/pegjs pascal_comments.pegjs
$ node use_pascal_comments.js
[ 'not bla bla ',
  ' pascal  nested  comment ',
  ' pum pum ',
  ' another comment ' ]
```

Donde

- ```
  [~/srcPLgrado/pegjs/examples(master)]$ pwd -P
  /Users/casiano/local/src/javascript/PLgrado/pegjs/examples
  ```

- ```
  [~/srcPLgrado/pegjs/examples(master)]$ git remote -v
  dmajda  https://github.com/dmajda/pegjs.git (fetch)
  dmajda  https://github.com/dmajda/pegjs.git (push)
  origin  git@github.com:crguezl/pegjs.git (fetch)
  origin  git@github.com:crguezl/pegjs.git (push)
  ```

- https://github.com/crguezl/pegjs/tree/master/examples

6.11. Un Lenguaje Dependiente del Contexto

El lenguaje $\{a^n b^n c^n / n \in \mathcal{N}\}$ no puede ser expresado mediante una gramática independiente del contexto.

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat anbncn.pegjs
/*
  The following parsing expression grammar describes the classic
  non-context-free language :
               { a^nb^nc^n / n >= 1 }

            S ← &(A 'c') 'a'+ B !('a'/'b'/'c')
            A ← 'a' A? 'b'
            B ← 'b' B? 'c'
*/

S = &(A 'c') 'a'+ B !('a'/'b'/'c')
A = 'a' A? 'b'
B = 'b' B? 'c'
```

Este ejemplo puede ser obtenido desde GitHub:

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ git remote -v
dmajda  https://github.com/dmajda/pegjs.git (fetch)
dmajda  https://github.com/dmajda/pegjs.git (push)
origin  git@github.com:crguezl/pegjs.git (fetch)
origin  git@github.com:crguezl/pegjs.git (push)
```

Veamos un ejemplo de uso:

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat use_anbncn.js
var PEG = require("./anbncn.js");
var r = PEG.parse("aabbcc");
console.log(r);

try {
  r = PEG.parse("aabbc");
  console.log(r);
}
catch (e) {
  console.log("Grr...."+e);
}
```

Ejecución:
```

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ ../bin/pegjs anbncn.pegjs

[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ node use_anbncn.js
[ '', [ 'a', 'a' ], [ 'b', [ 'b', '', 'c' ], 'c' ], '' ]
Grr....SyntaxError: Expected "c" but end of input found.
```

6.12. Usando Pegjs con CoffeeScript

Instalación de pegjs-coffee-plugin

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ sudo  npm install -g pegjs-coffee-p
```

Ejemplo Sencillo

```
[~/Dropbox/src/javascript/PLgrado/pegjs-coffee-plugin/examples(master)]$ cat simple.pegjs
{
  @reduce = (left, right)->
    sum = left
    for t in right
      op = t[0]
      num = t[1]
      switch op
        when '+' then sum += num; break
        when '-' then sum -= num; break
        when '*' then sum *= num; break
        when '/' then sum /= num; break
        else console.log("Error! "+op)
    sum

}
sum   = left:product right:([+-] product)* { @reduce(left, right); }
product = left:value right:([*/] value)*   { @reduce(left, right); }
value   = number:[0-9]+                    { parseInt(number.join(''),10) }
        / '(' sum:sum ')'                  { sum }

[~/Dropbox/src/javascript/PLgrado/pegjs-coffee-plugin/examples(master)]$  cat use_simple.coffe
PEG = require("./simple.js")
r = PEG.parse("2+3*(2+1)-10/2")
console.log(r)

[~/Dropbox/src/javascript/PLgrado/pegjs-coffee-plugin/examples(master)]$ cat Rakefile

task :default do
  sh "pegcoffee simple.pegjs"
end

task :run do
  sh "coffee use_simple.coffee"
end

[~/Dropbox/src/javascript/PLgrado/pegjs-coffee-plugin/examples(master)]$ rake
pegcoffee simple.pegjs
[~/Dropbox/src/javascript/PLgrado/pegjs-coffee-plugin/examples(master)]$ rake run
coffee use_simple.coffee
6
```

Véase También

- pegjs-coffee-plugin en GitHub

6.13. Práctica: Analizador de PL0 Ampliado Usando PEG.js

Reescriba el analizador sintáctico del lenguaje PL0 realizado en la práctica 4.6 usando PEG.js .

Donde

- Repositorio en GitHub
- Despliegue en Heroku
- `[~/srcPLgrado/pegjscalc(master)]$ pwd -P`
  `/Users/casiano/local/src/javascript/PLgrado/pegjscalc`
- ```
  [~/srcPLgrado/pegjscalc(master)]$ git remote -v
  heroku  git@heroku.com:pegjspl0.git (fetch)
  heroku  git@heroku.com:pegjspl0.git (push)
  origin  git@github.com:crguezl/pegjscalc.git (fetch)
  origin  git@github.com:crguezl/pegjscalc.git (push)
  ```

Tareas

- Modifique `block` y `statement` para que los `procedure` reciban argumentos y las llamadas a procedimiento puedan pasar argumentos. Añada `if ... then ... else ...`.
- Actualice la documentación de la gramática para que refleje la gramática ampliada
- Limite el número de programas que se pueden salvar a un número prefijado, por ejemplo 10. Si se intenta salvar uno se suprime uno al azar y se guarda el nuevo.
- Las pruebas deben comprobar que la asociatividad a izquierdas funciona bien y probar todos los constructos del lenguaje así como alguna situación de error

Referencias para esta Práctica

- Véase el capítulo *Heroku* 31
- Heroku Postgres
- Véase el capítulo *DataMapper* 32

6.14. Práctica: Ambiguedad en C++

This lab illustrates a problem that arises in C++. The C++ syntax does not disambiguate between expression statements (`stmt`) and declaration statements (`decl`). The ambiguity arises when an expression statement has a function-style cast as its left-most subexpression. Since C does not support function-style casts, this ambiguity does not occur in C programs. For example, the phrase
`int (x) = y+z;`
parses as either a `decl` or a `stmt`.

The disambiguation rule used in C++ is that *if the statement can be interpreted both as a declaration and as an expression, the statement is interpreted as a declaration statement.*

The following examples disambiguate into *expression* statements when the potential *declarator* is followed by an operator different from equal or semicolon (`type_spec` stands for a type specifier):
  ```

expr	dec

`type_spec(i)++;`
`type_spec(i,3)<<d;`
`type_spec(i)->l=24;`
	`type_spec(*i)(int);`
	`type_spec(j)[5];`
	`type_spec(m) = { 1, 2 };`
	`type_spec(a);`
	`type_spec(*b)();`
	`type_spec(c)=23;`
	`type_spec(d),e,f,g=0;`
	`type_spec(h)(e,3);`

Regarding to this problem, Bjarne Stroustrup remarks:

> *Consider analyzing a statement consisting of a sequence of tokens as follows:*
>
> `type_spec (dec_or_exp) tail`
>
> *Here* `dec_or_exp` *must be a declarator, an expression, or both for the statement to be legal. This implies that* `tail` *must be a semicolon, something that can follow a parenthesized declarator or something that can follow a parenthesized expression, that is, an initializer,* `const`*,* `volatile`*, (, [, or a postfix or infix operator. The general cases cannot be resolved without backtracking, nested grammars or similar advanced parsing strategies. In particular, the lookahead needed to disambiguate this case is not limited.*

The following grammar depicts an oversimplified version of the C++ ambiguity:

```
$ cat CplusplusNested.y
%token ID INT NUM

%right '='
%left '+'

%%
prog:
    /* empty */
  | prog stmt
;

stmt:
    expr ';'
  | decl
;

expr:
    ID
  | NUM
  | INT '(' expr ')' /* typecast */
  | expr '+' expr
  | expr '=' expr
;

decl:
    INT declarator ';'
```

```
  | INT declarator '=' expr ';'
;

declarator:
    ID
  | '(' declarator ')'
;

%%
```

Escriba un programa PegJS en CoffeeScript que distinga correctamente entre declaraciones y sentencias. Este es un ejemplo de un programa que usa una solución al problema:

```
[~/Dropbox/src/javascript/PLgrado/pegjs-coffee-plugin/examples(master)]$ cat use_cplusplus.cof
PEG = require("./cplusplus.js")
input = "int (a); int c = int (b);"

r = PEG.parse(input)
console.log("input = '#{input}'\noutput="+JSON.stringify r)

input = "int b = 4+2  ;  "
r = PEG.parse(input)
console.log("input = '#{input}'\noutput="+JSON.stringify r)

input = "bum = caf = 4-1;\n"
r = PEG.parse(input)
console.log("input = '#{input}'\noutput="+JSON.stringify r)

input = "b2 = int(4);"
r = PEG.parse(input)
console.log("input = '#{input}'\noutput="+JSON.stringify r)

input = "int(4);"
r = PEG.parse(input)
console.log("input = '#{input}'\noutput="+JSON.stringify r)
```

Y este un ejemplo de salida:

```
$ pegcoffee cplusplus.pegjs
$ coffee use_cplusplus.coffee
input = 'int (a); int c = int (b);'
output=["decl","decl"]
input = 'int b = 4+2  ;  '
output=["decl"]
input = 'bum = caf = 4-1;
'
output=["stmt"]
input = 'b2 = int(4);'
output=["stmt"]
input = 'int(4);'
output=["stmt"]
```

6.15. Práctica: Inventando un Lenguaje: Tortoise

El objetivo de esta práctica es crear un lenguaje de programación imperativa sencillo de estilo LOGO. Para ello lea el capítulo Inventing a Language - Tortoise del curso PL101: Create Your Own Programming de Nathan Whitehead. Haga todos los ejercicios e implemente el lenguaje descrito.

Puede encontrar una solución a la práctica en GitHub en el repositorio pl101 de Dave Ingram. Usela como guía cuando se sienta desorientado.

Recursos

- Inventing a Language - Tortoise por Nathan Whitehead
- Repositorio dingram / pl101 en GitHub con las soluciones a esta práctica.
  - Blog de dingram (Dave Ingram)
- Repositorio PatrixCR / PL101 en GitHub con las soluciones a esta práctica.
- Repositorio Clinton N. Dreisbach/ PL101 en GitHub con contenidos del curso PL101
- Foro
- Sobre Nathan Whitehead
  - Nathan's Lessons
  - Nathan Whitehead en GitHub
  - Nathan in YouTube

Capítulo 7

Análisis Sintáctico Ascendente en JavaScript

7.1. Conceptos Básicos para el Análisis Sintáctico

Suponemos que el lector de esta sección ha realizado con éxito un curso en teoría de autómatas y lenguajes formales. Las siguientes definiciones repasan los conceptos mas importantes.

Definición 7.1.1. *Dado un conjunto A, se define A^* el cierre de Kleene de A como: $A^* = \cup_{n=0}^{\infty} A^n$*

Se admite que $A^0 = \{\epsilon\}$, donde ϵ denota la palabra vacía, esto es la palabra que tiene longitud cero, formada por cero símbolos del conjunto base A.

Definición 7.1.2. *Una gramática G es una cuaterna $G = (\Sigma, V, P, S)$. Σ es el conjunto de terminales. V es un conjunto (disjunto de Σ) que se denomina conjunto de* variables sintácticas *o* categorías gramáticales, *P es un conjunto de pares de $V \times (V \cup \Sigma)^*$. En vez de escribir un par usando la notación $(A, \alpha) \in P$ se escribe $A \to \alpha$. Un elemento de P se denomina producción. Por último, S es un símbolo del conjunto V que se denomina símbolo de arranque.*

Definición 7.1.3. *Dada una gramática $G = (\Sigma, V, P, S)$ y $\mu = \alpha A \beta \in (V \cup \Sigma)^*$ una frase formada por variables y terminales y $A \to \gamma$ una producción de P, decimos que μ deriva en un paso en $\alpha\gamma\beta$. Esto es, derivar una cadena $\alpha A \beta$ es sustituir una variable sintáctica A de V por la parte derecha γ de una de sus reglas de producción. Se dice que μ deriva en n pasos en δ si deriva en $n - 1$ pasos en una cadena $\alpha A \beta$ la cual deriva en un paso en δ. Se escribe entonces que $\mu \stackrel{*}{\Longrightarrow} \delta$. Una cadena deriva en 0 pasos en si misma.*

Definición 7.1.4. *Dada una gramática $G = (\Sigma, V, P, S)$ se denota por $L(G)$ o lenguaje generado por G al lenguaje:*

$$L(G) = \{x \in \Sigma^* : S \stackrel{*}{\Longrightarrow} x\}$$

Esto es, el lenguaje generado por la gramática G esta formado por las cadenas de terminales que pueden ser derivados desde el símbolo de arranque.

Definición 7.1.5. *Una derivación que comienza en el símbolo de arranque y termina en una secuencia formada por sólo terminales de Σ se dice* completa.

Una derivación $\mu \stackrel{*}{\Longrightarrow} \delta$ en la cual en cada paso $\alpha A x$ la regla de producción aplicada $A \to \gamma$ se aplica en la variable sintáctica mas a la derecha se dice una derivación a derechas

Una derivación $\mu \stackrel{*}{\Longrightarrow} \delta$ en la cual en cada paso $x A \alpha$ la regla de producción aplicada $A \to \gamma$ se aplica en la variable sintáctica mas a la izquierda se dice una derivación a izquierdas

Definición 7.1.6. *Observe que una derivación puede ser representada como un árbol cuyos nodos están etiquetados en $V \cup \Sigma$. La aplicación de la regla de producción $A \to \gamma$ se traduce en asignar como hijos del nodo etiquetado con A a los nodos etiquetados con los símbolos $X_1 \ldots X_n$ que constituyen la frase $\gamma = X_1 \ldots X_n$. Este árbol se llama* árbol sintáctico concreto *asociado con la derivación.*

Definición 7.1.7. *Observe que, dada una frase $x \in L(G)$ una derivación desde el símbolo de arranque da lugar a un árbol. Ese árbol tiene como raíz el símbolo de arranque y como hojas los terminales $x_1 \dots x_n$ que forman x. Dicho árbol se denomina* árbol de análisis sintáctico concreto *de x. Una derivación determina una forma de recorrido del árbol de análisis sintáctico concreto.*

Definición 7.1.8. *Una gramática G se dice ambigua si existe alguna frase $x \in L(G)$ con al menos dos árboles sintácticos. Es claro que esta definición es equivalente a afirmar que existe alguna frase $x \in L(G)$ para la cual existen dos derivaciones a izquierda (derecha) distintas.*

7.1.1. Ejercicio

Dada la gramática con producciones:

program → declarations statements | statements
declarations → declaration ';' declarations | declaration ';'
declaration → INT idlist | STRING idlist
statements → statement ';' statements | statement
statement → ID '=' expression | P expression
expression → term '+' expression | term
term → factor '*' term | factor
factor → '(' expression ')' | ID | NUM | STR
idlist → ID ',' idlist | ID

En esta gramática, Σ esta formado por los caracteres entre comillas simples y los símbolos cuyos identificadores están en mayúsculas. Los restantes identificadores corresponden a elementos de V. El símbolo de arranque es $S = program$.

Conteste a las siguientes cuestiones:

1. Describa con palabras el lenguaje generado.
2. Construya el árbol de análisis sintáctico concreto para cuatro frases del lenguaje.
3. Señale a que recorridos del árbol corresponden las respectivas derivaciones a izquierda y a derecha en el apartado 2.
4. ¿Es ambigua esta gramática?. Justifique su respuesta.

7.2. Ejemplo Simple en Jison

Jison es un generador de analizadores sintácticos LALR. Otro analizador LALR es JS/CC.

Gramática

```
%%

S   : A
    ;
A   : /* empty */
    | A x
    ;
```

basic2_lex.jison

```
[~/jison/examples/basic2_lex(develop)]$ cat basic2_lex.jison
/* description: Basic grammar that contains a nullable A nonterminal. */
```

```
%lex
%%

\s+               {/* skip whitespace */}
[a-zA-Z_]\w*      {return 'x';}

/lex

%%

S   : A
           { return $1+" identifiers"; }
    ;
A   : /* empty */
           {
              console.log("starting");
              $$ = 0;
           }
    | A x  {
              $$ = $1 + 1;
              console.log($$)
           }
    ;
```

index.html

```
$ cat basic2_lex.html
<!DOCTYPE HTML>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Jison</title>
    <link rel="stylesheet" href="global.css" type="text/css" media="screen" charset="utf-8" />
  </head>
  <body>
    <h1>basic2_lex demo</h1>
    <div id="content">
      <script src="jquery/jquery.js"></script>
      <script src="basic2_lex.js"></script>
      <script src="main.js"></script>
      <p>
        <input type="text" value="x x x x" /> <button>parse</button>
        <span id="output"></span> <!-- Output goes here! -->
      </p>
    </div>
  </body>
</html>
```

Rakefile

```
$ cat Rakefile
# install package:
#     sudo npm install beautifier
#
```

```
#   more about beautifier:
#       https://github.com/rickeyski/node-beautifier

dec "compile the grammar basic2_lex_ugly.jison"
task :default => %w{basic2_lex_ugly.js} do
  sh "mv basic2_lex.js basic2_lex_ugly.js"
  sh "jsbeautify basic2_lex_ugly.js > basic2_lex.js"
  sh "rm -f basic2_lex_ugly.js"
end

file "basic2_lex_ugly.js" => %w{basic2_lex.jison} do
  sh "jison basic2_lex.jison -o basic2_lex.js"
end
```

1. node-beautifier

Véase También

1. JISON
2. Try Jison Examples
3. JavaScript 1.4 LR(1) Grammar 1999.
4. Creating a JavaScript Parser Una implementación de ECMAScript 5.1 usando Jison disponible en GitHub en https://github.com/cjihrig/jsparser. Puede probarse en: http://www.cjihrig.com/development
5. Bison on JavaScript por Rolando Perez
6. Slogo a language written using Jison
7. List of languages that compile to JS
8. Prototype of a Scannerless, Generalized Left-to-right Rightmost (SGLR) derivation parser for JavaScript

global.css

```
[~/jison/examples/basic2_lex(develop)]$ cat global.css
html *
{
   font-size: large;
   /* The !important ensures that nothing can override what you've set in this style (unless i
   font-family: Arial;
}

.thumb {
    height: 75px;
    border: 1px solid #000;
    margin: 10px 5px 0 0;
  }

h1            { text-align: center; font-size: x-large; }
th, td        { vertical-align: top; text-align: left; }
/* #finaltable  * { color: white; background-color: black; }   */

/* #finaltable table { border-collapse:collapse; } */
```

```
/* #finaltable table, td { border:1px solid white; } */
#finaltable:hover td { background-color: blue; }
tr:nth-child(odd)    { background-color:#eee; }
tr:nth-child(even)    { background-color:#00FF66; }
input       { text-align: right;  border: none;       }     /* Align input to the right  */
textarea     { border: outset; border-color: white;       }
table        { border: inset; border-color: white; }
.hidden      { display: none; }
.unhidden    { display: block; }
table.center { margin-left:auto; margin-right:auto; }
#result      { border-color: red; }
tr.error       { background-color: red; }
pre.output   { background-color: white; }
span.repeated { background-color: red }
span.header { background-color: blue }
span.comments { background-color: orange }
span.blanks { background-color: green }
span.nameEqualValue { background-color: cyan }
span.error { background-color: red }

body
{
 background-color:#b0c4de;  /* blue */
}
```

7.2.1. Véase También

1. JISON
2. Try Jison Examples
3. JavaScript 1.4 LR(1) Grammar 1999.
4. Creating a JavaScript Parser Una implementación de ECAMScript 5.1 usando Jison disponible en GitHub en https://github.com/cjihrig/jsparser. Puede probarse en: http://www.cjihrig.com/development/
5. Slogo a language written using Jison
6. List of languages that compile to JS
7. Prototype of a Scannerless, Generalized Left-to-right Rightmost (SGLR) derivation parser for JavaScript

7.2.2. Práctica: Secuencia de Asignaciones Simples

Modifique este ejemplo para que el lenguaje acepte una secuencia de sentencias de asignación de la forma `ID = NUM` separadas por puntos y comas, por ejemplo `a = 4; b = 4.56; c = -8.57e34`. El analizador retorna un hash/objeto cuyas claves son los identificadores y cuyos valores son los números. Clone el repositorio en https://github.com/crguezl/jison-basic2.

Modifique los analizadores léxico y sintáctico de forma conveniente.

Añada acciones semánticas para que el analizador devuelva una tabla de símbolos con los identificadores y sus valores.

7.3. Ejemplo en Jison: Calculadora Simple

1. Enlace al fork del proyecto jison de crguezl (GitHub)

calculator.jison

```
[~/jison/examples/html_calc_example(develop)]$ cat calculator.jison

/* description: Parses end executes mathematical expressions. */

/* lexical grammar */
%lex
%%

\s+                   /* skip whitespace */
[0-9]+("."[0-9]+)?\b  return 'NUMBER'
"*"                   return '*'
"/"                   return '/'
"-"                   return '-'
"+"                   return '+'
"^"                   return '^'
"!"                   return '!'
"%"                   return '%'
"("                   return '('
")"                   return ')'
"PI"                  return 'PI'
"E"                   return 'E'
<<EOF>>               return 'EOF'
.                     return 'INVALID'

/lex

/* operator associations and precedence */

%left '+' '-'
%left '*' '/'
%left '^'
%right '!'
%right '%'
%left UMINUS

%start expressions

%% /* language grammar */

expressions
    : e EOF
        { typeof console !== 'undefined' ? console.log($1) : print($1);
          return $1; }
    ;

e
    : e '+' e
        {$$ = $1+$3;}
    | e '-' e
        {$$ = $1-$3;}
    | e '*' e
        {$$ = $1*$3;}
```

```
    | e '/' e
        {$$ = $1/$3;}
    | e '^' e
        {$$ = Math.pow($1, $3);}
    | e '!'
        {{
          $$ = (function fact (n) { return n==0 ? 1 : fact(n-1) * n })($1);
        }}
    | e '%'
        {$$ = $1/100;}
    | '-' e %prec UMINUS
        {$$ = -$2;}
    | '(' e ')'
        {$$ = $2;}
    | NUMBER
        {$$ = Number(yytext);}
    | E
        {$$ = Math.E;}
    | PI
        {$$ = Math.PI;}
    ;
```

main.js

```
[~/jison/examples/html_calc_example(develop)]$ cat main.js
$(document).ready(function () {
  $("button").click(function () {
    try {
      var result = calculator.parse($("input").val())
      $("span").html(result);
    } catch (e) {
      $("span").html(String(e));
    }
  });
});
```

calculator.html

```
[~/jison/examples/html_calc_example(develop)]$ cat calculator.html
<!DOCTYPE HTML>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Calc</title>
    <link rel="stylesheet" href="global.css" type="text/css" media="screen" charset="utf-8" />
  </head>
  <body>
    <h1>Calculator demo</h1>
    <div id="content">
      <script src="jquery/jquery.js"></script>
      <script src="calculator.js"></script>
      <script src="main.js"></script>
      <p>
        <input type="text" value="PI*4^2 + 5" /> <button>equals</button>
```

```
        <span></span> <!-- Output goes here! -->
      </p>
    </div>
  </body>
</html>
```

Rakefile

```
[~/jisoncalc(clase)]$ cat Rakefile
task :default => %w{calcugly.js} do
  sh "jsbeautify calcugly.js > calculator.js"
  sh "rm -f calcugly.js"
end

file "calcugly.js" => %w{calculator.jison} do
  sh "jison calculator.jison calculator.l -o calculator.js; mv calculator.js calcugly.js"
end

task :testf do
  sh "open -a firefox test/test.html"
end

task :tests do
  sh "open -a safari test/test.html"
end
```

global.css

```
[~/jison/examples/html_calc_example(develop)]$ cat global.css
html *
{
   font-size: large;
   /* The !important ensures that nothing can override what you've set in this style (unless i
   font-family: Arial;
}

.thumb {
    height: 75px;
    border: 1px solid #000;
    margin: 10px 5px 0 0;
  }

h1            { text-align: center; font-size: x-large; }
th, td        { vertical-align: top; text-align: left; }
/* #finaltable  * { color: white; background-color: black; }   */

/* #finaltable table { border-collapse:collapse; } */
/* #finaltable table, td { border:1px solid white; } */
#finaltable:hover td { background-color: blue; }
tr:nth-child(odd)    { background-color:#eee; }
tr:nth-child(even)    { background-color:#00FF66; }
input        { text-align: right;  border: none;       }     /* Align input to the right  */
textarea     { border: outset; border-color: white;       }
table        { border: inset; border-color: white; }
```

```
.hidden      { display: none; }
.unhidden    { display: block; }
table.center { margin-left:auto; margin-right:auto; }
#result      { border-color: red; }
tr.error       { background-color: red; }
pre.output   { background-color: white; }
span.repeated { background-color: red }
span.header { background-color: blue }
span.comments { background-color: orange }
span.blanks { background-color: green }
span.nameEqualValue { background-color: cyan }
span.error { background-color: red }

body
{
 background-color:#b0c4de;  /* blue */
}
```

test/assert.html

```
$ cat test/assert.js
var output = document.getElementById('output');

function assert( outcome, description) {
  var li = document.createElement('li');
  li.className = outcome ? 'pass' : 'fail';
  li.appendChild(document.createTextNode(description));

  output.appendChild(li);
};
```

test/test.css

```
~/jisoncalc(clase)]$ cat test/test.css
.pass:before {
  content: 'PASS: ';
  color:  blue;
  font-weight: bold;
}

.fail:before {
  content: 'FAIL: ';
  color: red;
  font-weight: bold;

}
```

test/test.html

```
[~/jisoncalc(clase)]$ cat test/test.html
<!DOCTYPE HTML>
<html lang="en">
  <head>
    <meta charset="UTF-8">
```

```
    <title>Testing Our Simple Calculator</title>
    <link rel="stylesheet" href="test.css" />
    <script type="text/javascript" src="../calculator.js"></script>

  </head>
  <body>
    <h1>Testing Our Simple Calculator
    </h1>

    <ul id="output"></ul>
    <script type="text/javascript" src="______.js"></script>

    <script type="text/javascript">
      var r = __________.parse("a = 4*8");
      assert(_______________, "a is 4*8");
      assert(_____________, "32 == 4*8");
      r = calculator.parse("a = 4;\nb=a+1;\nc=b*2");
      assert(____________, "4 is the first computed result ");
      assert(______________, "a is 4");
      assert(______________, "b is 5");
      assert(______________, "c is 10");
    </script>
      See the NetTuts+ tutorial at <a href="http://net.tutsplus.com/tutorials/javascript-ajax/
  </body>
</html>
```

7.3.1. Práctica: Calculadora con Listas de Expresiones y Variables

Modifique la calculadora vista en la sección anterior 7.3 para que el lenguaje cumpla los siguientes requisitos:

- Extienda el lenguaje de la calculadora para que admita expresiones de asignación `a = 2*3`
- Extienda el lenguaje de la calculadora para que admita listas de sentencias `a = 2; b = a +1`
- El analizador devuelve la lista de expresiones evaluadas y la tabla de símbolos (con las parejas variable-valor).
- Emita un mensaje de error específico si se intentan modificar las constantes `PI` y `e`.
- Emita un mensaje de error específico si se intenta una división por cero
- Emita un mensaje de error específico si se intenta acceder para lectura a una variable no inicializada `a = c`
- El lenguaje debería admitir expresiones vacías, estos es secuencias consecutivas de puntos y comas sin producir error (`a = 4;;; b = 5`)
- Introduzca pruebas unitarias como las descritas en la sección **??** (*Quick Tip: Quick and Easy JavaScript Test*

7.4. Conceptos Básicos del Análisis LR

Los analizadores generados por `jison` entran en la categoría de analizadores *LR*. Estos analizadores construyen una derivación a derechas inversa (o *antiderivación*). De ahí la R en LR (del inglés *rightmost derivation*). El árbol sintáctico es construido de las hojas hacia la raíz, siendo el último paso en la antiderivación la construcción de la primera derivación desde el símbolo de arranque.

Empezaremos entonces considerando las frases que pueden aparecer en una derivación a derechas. Tales frases consituyen el *lenguaje de las formas sentenciales a rderechas* FSD:

Definición 7.4.1. *Dada una gramática* $G = (\Sigma, V, P, S)$ *no ambigua, se denota por* FSD *(lenguaje de las formas Sentenciales a Derechas) al lenguaje de las sentencias que aparecen en una derivación a derechas desde el símbolo de arranque.*

$$FSD = \left\{ \alpha \in (\Sigma \cup V)* : \exists S \underset{RM}{\overset{*}{\Longrightarrow}} \alpha \right\}$$

*Donde la notacion RM indica una derivación a derechas (*rightmost*). Los elementos de* FSD *se llaman "formas sentenciales derechas".*

Dada una gramática no ambigua $G = (\Sigma, V, P, S)$ y una frase $x \in L(G)$ el proceso de antiderivación consiste en encontrar la última derivación a derechas que dió lugar a x. Esto es, si $x \in L(G)$ es porque existe una derivación a derechas de la forma

$$S \overset{*}{\Longrightarrow} yAz \Longrightarrow ywz = x.$$

El problema es averiguar que regla $A \to w$ se aplicó y en que lugar de la cadena x se aplicó. En general, si queremos antiderivar una forma sentencial derecha $\beta\alpha w$ debemos averiguar por que regla $A \to \alpha$ seguir y en que lugar de la forma (después de β en el ejemplo) aplicarla.

$$S \overset{*}{\Longrightarrow} \beta A w \Longrightarrow \beta\alpha w.$$

La pareja formada por la regla y la posición se denomina *handle*, *mango* o *manecilla* de la forma. Esta denominación viene de la visualización gráfica de la regla de producción como una mano que nos permite escalar hacia arriba en el árbol. Los "dedos" serían los símbolos en la parte derecha de la regla de producción.

Definición 7.4.2. *Dada una gramática* $G = (\Sigma, V, P, S)$ *no ambigua, y dada una forma sentencial derecha* $\alpha = \beta\gamma x$*, con* $x \in \Sigma^*$*, el* mango *o* handle *de* α *es la última producción/posición que dió lugar a* α*:*

$$S \underset{RM}{\overset{*}{\Longrightarrow}} \beta B x \Longrightarrow \beta\gamma x = \alpha$$

Escribiremos: $handle(\alpha) = (B \to \gamma, \beta\gamma)$. La función *handle* tiene dos componentes: $handle_1(\alpha) = B \to \gamma$ y $handle_2(\alpha) = \beta\gamma$

Si dispusieramos de un procedimiento que fuera capaz de identificar el mango, esto es, de detectar la regla y el lugar en el que se posiciona, tendríamos un mecanismo para construir un analizador. Lo curioso es que, a menudo es posible encontrar un autómata finito que reconoce el lenguaje de los prefijos $\beta\gamma$ que terminan en el mango. Con mas precisión, del lenguaje:

Definición 7.4.3. *El conjunto de* prefijos viables *de una gramática* G *se define como el conjunto:*

$$PV = \left\{ \delta \in (\Sigma \cup V)* : \exists S \underset{RM}{\overset{*}{\Longrightarrow}} \alpha = \beta\gamma x \ y\ \delta\ es\ un\ prefijo\ de\ handle_2(\alpha) = \beta\gamma \right\}$$

Esto es, es el lenguaje de los prefijos viables es el conjunto de frases que son prefijos de $handle_2(\alpha)) = \beta\gamma$, siendo α una forma sentencial derecha ($\alpha \in FSD$). Los elementos de PV se denominan prefijos viables.

Obsérvese que si se dispone de un autómata que reconoce PV entonces se dispone de un mecanismo para investigar el lugar y el aspecto que pueda tener el mango. Si damos como entrada la sentencia $\alpha = \beta\gamma x$ a dicho autómata, el autómata aceptará la cadena $\beta\gamma$ pero rechazará cualquier extensión del prefijo. Ahora sabemos que el mango será alguna regla de producción de G cuya parte derecha sea un sufijo de $\beta\gamma$.

Definición 7.4.4. *El siguiente autómata finito no determinista puede ser utilizado para reconocer el lenguaje de los prefijos viables PV:*

- *Alfabeto* $= V \cup \Sigma$
- *Los estados del autómata se denominan LR(0) items. Son parejas formadas por una regla de producción de la gramática y una posición en la parte derecha de la regla de producción. Por ejemplo,* $(E \to E + E, 2)$ *sería un LR(0) item para la gramática de las expresiones.*

  Conjunto de Estados:

  $Q = \{(A \to \alpha, n) : A \to \alpha \in P,\ n \leq |\alpha|\}$

  La notación $\mid \alpha \mid$ *denota la longitud de la cadena* $\mid \alpha \mid$. *En vez de la notación* $(A \to \alpha, n)$ *escribiremos:* $A \to \beta_{\uparrow}\gamma = \alpha$, *donde la flecha ocupa el lugar indicado por el número* $n =\mid \beta \mid$ *:*
- *La función de transición intenta conjeturar que partes derechas de reglas de producción son viables. El conjunto de estados actual del NFA representa el conjunto de pares (regla de producción, posición en la parte derecha) que tienen alguna posibilidad de ser aplicadas de acuerdo con la entrada procesada hasta el momento:*

$$\delta(A \to \alpha_{\uparrow}X\beta, X) = A \to \alpha X_{\uparrow}\beta \ \forall X \in V \cup \Sigma$$
$$\delta(A \to \alpha_{\uparrow}B\beta, \epsilon) = B \to_{\uparrow} \gamma \ \forall B \to \gamma \in P$$

- *Estado de arranque: Se añade la "superregla"* $S' \to S$ *a la gramática* $G = (\Sigma, V, P, S)$. *El LR(0) item* $S' \to_{\uparrow} S$ *es el estado de arranque.*
- *Todos los estados definidos (salvo el de muerte) son de aceptación.*

Denotaremos por $LR(0)$ a este autómata. Sus estados se denominan $LR(0) - items$. La idea es que este autómata nos ayuda a reconocer los prefijos viables PV.

Una vez que se tiene un autómata que reconoce los prefijos viables es posible construir un analizador sintáctico que construye una antiderivación a derechas. La estrategia consiste en "alimentar" el autómata con la forma sentencial derecha. El lugar en el que el autómata se detiene, rechazando indica el lugar exacto en el que termina el *handle* de dicha forma.

Ejemplo 7.4.1. *Consideremos la gramática:*

$$\begin{array}{l} S \to a\ S\ b \\ S \to \epsilon \end{array}$$

El lenguaje generado por esta gramática es $L(G) = \{a^n b^n\ :\ n \geq 0\}$ *Es bien sabido que el lenguaje* $L(G)$ *no es regular. La figura 7.1 muestra el* autómata finito no determinista con ϵ-transiciones *(*NFA*) que reconoce los prefijos viables de esta gramática, construido de acuerdo con el algoritmo 7.4.4.*

Véase https://github.com/crguezl/jison-aSb para una implementación en Jison de una variante de esta gramática.

Ejercicio 7.4.1. *Simule el comportamiento del autómata sobre la entrada aabb. ¿Donde rechaza? ¿En que estados está el autómata en el momento del rechazo?. ¿Qué etiquetas tienen? Haga también las trazas del autómata para las entradas aaSbb y aSb. ¿Que antiderivación ha construido el autómata con sus sucesivos rechazos? ¿Que terminales se puede esperar que hayan en la entrada cuando se produce el rechazo del autómata?*

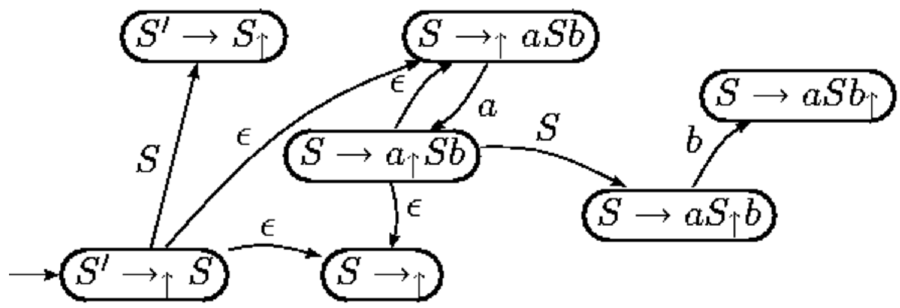


Figura 7.1: NFA que reconoce los prefijos viables

7.5. Construcción de las Tablas para el Análisis SLR

7.5.1. Los conjuntos de Primeros y Siguientes

Repasemos las nociones de conjuntos de *Primeros* y *siguientes*:

Definición 7.5.1. *Dada una gramática $G = (\Sigma, V, P, S)$ y una frase $\alpha \in (V \cup \Sigma)^*$ se define el conjunto $FIRST(\alpha)$ como:*

$$FIRST(\alpha) = \left\{ b \in \Sigma : \alpha \stackrel{*}{\Longrightarrow} b\beta \right\} \cup N(\alpha)$$

donde:

$$N(\alpha) = \begin{cases} \{\epsilon\} & si\ \alpha \stackrel{*}{\Longrightarrow} \epsilon \\ \emptyset & en\ otro\ caso \end{cases}$$

Definición 7.5.2. *Dada una gramática $G = (\Sigma, V, P, S)$ y una variable $A \in V$ se define el conjunto $FOLLOW(A)$ como:*

$$FOLLOW(A) = \left\{ b \in \Sigma : \exists\ S \stackrel{*}{\Longrightarrow} \alpha A b \beta \right\} \cup E(A)$$

donde

$$E(A) = \begin{cases} \{\$\} & si\ S \stackrel{*}{\Longrightarrow} \alpha A \\ \emptyset & en\ otro\ caso \end{cases}$$

Algoritmo 7.5.1. *Construcción de los conjuntos $FIRST(X)$*

1. *Si $X \in \Sigma$ entonces $FIRST(X) = X$*

2. *Si $X \to \epsilon$ entonces $FIRST(X) = FIRST(X) \cup \{\epsilon\}$*

3. *Si $X \in V$ y $X \to Y_1 Y_2 \cdots Y_k \in P$ entonces*

$$\begin{array}{l} i = 1; \\ do \\ \quad FIRST(X) = FIRST(X) \cup FIRST(Y_i) - \{\epsilon\}; \\ \quad i++; \\ mientras\ (\epsilon \in FIRST(Y_i)\ and\ (i \leq k)) \\ si\ (\epsilon \in FIRST(Y_k)\ and\ i > k)\ FIRST(X) = FIRST(X) \cup \{\epsilon\} \end{array}$$

Este algoritmo puede ser extendido para calcular $FIRST(\alpha)$ para $\alpha = X_1 X_2 \cdots X_n \in (V \cup \Sigma)^*$.

Algoritmo 7.5.2. *Construcción del conjunto $FIRST(\alpha)$*

$$
\begin{array}{l}
i = 1; \\
FIRST(\alpha) = \emptyset; \\
do \\
\quad FIRST(\alpha) = FIRST(\alpha) \cup FIRST(X_i) - \{\epsilon\}; \\
\quad i++; \\
mientras\ (\epsilon \in FIRST(X_i)\ and\ (i \leq n)) \\
si\ (\epsilon \in FIRST(X_n)\ and\ i > n)\ FIRST(\alpha) = FIRST(X) \cup \{\epsilon\}
\end{array}
$$

Algoritmo 7.5.3. *Construcción de los conjuntos $FOLLOW(A)$ para las variables sintácticas $A \in V$: Repetir los siguientes pasos hasta que ninguno de los conjuntos $FOLLOW$ cambie:*

1. *$FOLLOW(S) = \{\$\}$ (\$ representa el final de la entrada)*
2. *Si $A \to \alpha B \beta$ entonces*

$$FOLLOW(B) = FOLLOW(B) \cup (FIRST(\beta) - \{\epsilon\})$$

3. *Si $A \to \alpha B$ o bien $A \to \alpha B \beta$ y $\epsilon \in FIRST(\beta)$ entonces*

$$FOLLOW(B) = FOLLOW(B) \cup FOLLOW(A)$$

7.5.2. Construcción de las Tablas

Para la construcción de las tablas de un analizador SLR se construye el *autómata finito determinista* (*DFA*) (Q, Σ, δ, q_0) equivalente al NFA presentado en la sección 7.4 usando el *algoritmo de construcción del subconjunto*.

Como recordará, en la construcción del subconjunto, partiendo del estado de arranque q_0 del NFA con ϵ-transiciones se calcula su *clausura* $\overline{\{q_0\}}$ y las clausuras de los conjuntos de estados $\overline{\delta(\overline{\{q_0\}}, a)}$ a los que transita. Se repite el proceso con los conjuntos resultantes hasta que no se introducen nuevos conjuntos-estado.

La clausura $\overline{A}$ de un subconjunto de estados del autómata A esta formada por todos los estados que pueden ser alcanzados mediante transiciones etiquetadas con la palabra vacía (denominadas ϵ transiciones) desde los estados de A. Se incluyen en $\overline{A}$, naturalmente los estados de A.

$$\overline{A} = \{q \in Q \ / \ \exists q' \in Q \ : \ \hat{\delta}(q', \epsilon) = q\}$$

Aquí $\hat{\delta}$ denota la *función de transición del autómata* extendida a cadenas de Σ^*.

$$\hat{\delta}(q, x) = \begin{cases} \delta(\hat{\delta}(q, y), a) & \text{si } x = ya \\ q & \text{si } x = \epsilon \end{cases} \tag{7.1}$$

En la práctica, y a partir de ahora así lo haremos, se prescinde de diferenciar entre δ y $\hat{\delta}$ usándose indistintamente la notación δ para ambas funciones.

La clausura puede ser computada usando una estructura de pila o aplicando la expresión recursiva dada en la ecuación 7.1.

Para el NFA mostrado en el ejemplo 7.4.1 el DFA construído mediante esta técnica es el que se muestra en la figura 7.3. Se ha utilizado el símbolo `#` como marcador. Se ha omitido el número 3 para que los estados coincidan en numeración con los generados por `jison` (véase el cuadro **??**).

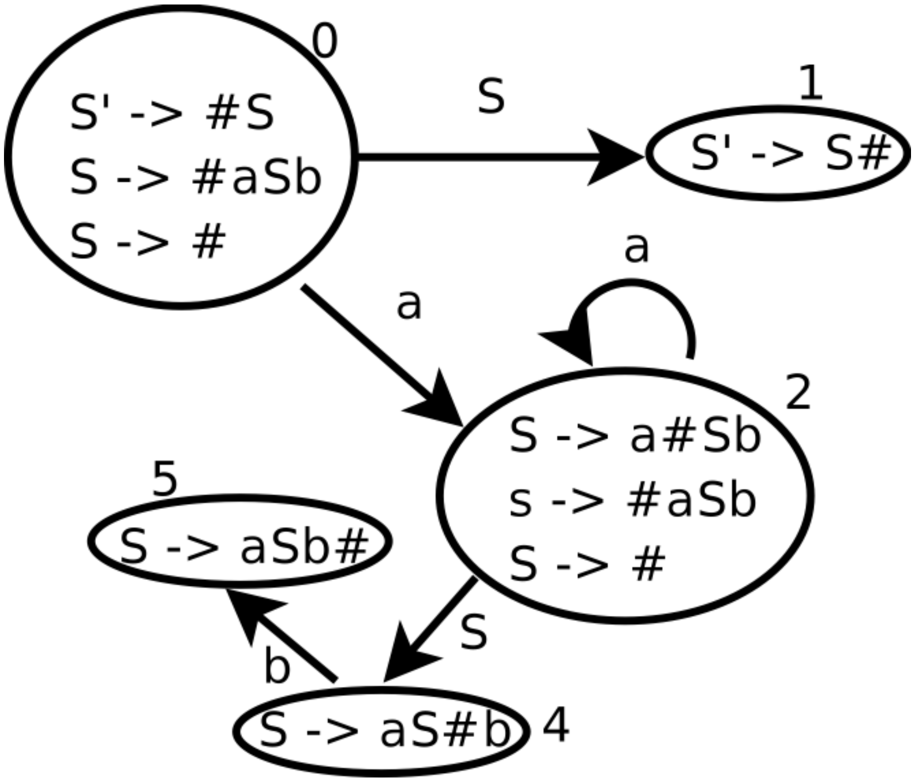


Figura 7.2: DFA equivalente al NFA de la figura 7.1

Un analizador sintáctico LR utiliza una tabla para su análisis. Esa tabla se construye a partir de la tabla de transiciones del DFA. De hecho, la tabla se divide en dos tablas, una llamada *tabla de saltos* o *tabla de gotos* y la otra *tabla de acciones*.

La tabla *goto* de un analizador *SLR* no es más que la tabla de transiciones del autómata DFA obtenido aplicando la construcción del subconjunto al NFA definido en 7.4.4. De hecho es la tabla de transiciones restringida a V (recuerde que el alfabeto del autómata es $V \cup \Sigma$, i denota al i-ésimo estado resultante de aplicar la construcción del subconjunto y que I_i denota al conjunto de LR(0) item asociado con dicho estado):

$$\delta_{|V \times Q} : V \times Q \to Q.$$
donde se define $goto(i, A) = \delta(A, I_i)$

La parte de la función de transiciones del DFA que corresponde a los terminales que no producen rechazo, esto es, $\delta_{|\Sigma \times Q} : \Sigma \times Q \to Q$ se adjunta a una tabla que se denomina *tabla de acciones*. La tabla de acciones es una tabla de doble entrada en los estados y en los símbolos de Σ. Las acciones de transición ante terminales se denominan *acciones de desplazamiento* o (*acciones shift*):

$$\delta_{|\Sigma \times Q} : \Sigma \times Q \to Q$$
donde se define $action(i, a) = shift\ \delta(a, I_i)$

Cuando un estado s contiene un LR(0)-item de la forma $A \to \alpha_\uparrow$, esto es, el estado corresponde a un posible rechazo, ello indica que hemos llegado a un final del prefijo viable, que hemos visto α y que, por tanto, es probable que $A \to \alpha$ sea el *handle* de la forma sentencial derecha actual. Por tanto, añadiremos en entradas de la forma (s, a) de la tabla de acciones una acción que indique que hemos encontrado el mango en la posición actual y que la regla asociada es $A \to \alpha$. A una acción de este tipo se la denomina *acción de reducción*.

La cuestión es, ¿para que valores de $a \in \Sigma$ debemos disponer que la acción para (s, a) es de reducción?

Se define $action(i, a) = reduce\ A \to \alpha$ ¿Pero, para que $a \in \Sigma$?

Podríamos decidir que ante cualquier terminal $a \in \Sigma$ que produzca un rechazo del autómata, pero podemos ser un poco mas selectivos. No cualquier terminal puede estar en la entrada en el momento en el que se produce la antiderivación o reducción. Observemos que si $A \to \alpha$ es el *handle* de γ es porque:

$$\exists S \overset{*}{\underset{RM}{\Longrightarrow}} \beta A b x \overset{*}{\underset{RM}{\Longrightarrow}} \beta \alpha b x = \gamma$$

Por tanto, cuando estamos reduciendo por $A \to \alpha$ los únicos terminales legales que cabe esperar en una reducción por $A \to \alpha$ son los terminales $b \in FOLLOW(A)$.

Se define $action(i, b) = reduce\ A \to \alpha$ Para $b \in FOLLOW(A)$

Dada una gramática $G = (\Sigma, V, P, S)$, podemos construir las tablas de acciones (*action table*) y transiciones (*gotos table*) mediante el siguiente algoritmo:

Algoritmo 7.5.4. *Construcción de Tablas* SLR

1. *Utilizando el Algoritmo de Construcción del Subconjunto, se construye el Autómata Finito Determinista (DFA) $(Q, V \cup \Sigma, \delta, I_0, F)$ equivalente al Autómata Finito No Determinista (NFA) definido en 7.4.4. Sea $C = \{I_1, I_2, \cdots I_n\}$ el conjunto de estados del DFA. Cada estado I_i es un conjunto de LR(0)-items o estados del NFA. Asociemos un índice i con cada conjunto I_i.*

2. *La tabla de* gotos *no es más que la función de transición del autómata restringida a las variables de la gramática:*

$$goto(i, A) = \delta(I_i, A) \text{ para todo } A \in V$$

3. *Las acciones para el estado I_i se determinan como sigue:*

   a) *Si $A \to \alpha_{\uparrow} a \beta \in I_i$, $\delta(I_i, a) = I_j$, $a \in \Sigma$ entonces:*

   $$action[i][a] = shift\ j$$

   b) *Si $S' \to S_{\uparrow} \in I_i$ entonces*

   $$action[i][\$] = accept$$

   c) *Para cualquier otro caso de la forma $A \to \alpha_{\uparrow} \in I_i$ distinto del anterior hacer*

   $$\forall a \in\ FOLLOW(A) :\ action[i][a] = reduce\ A \to \alpha$$

4. *Las entradas de la tabla de acción que queden indefinidas después de aplicado el proceso anterior corresponden a acciones de "error".*

Definición 7.5.3. *Si alguna de las entradas de la tabla resulta multievaluada, decimos que existe un conflicto y que la gramática no es* SLR.

1. *En tal caso, si una de las acciones es de 'reducción" y la otra es de 'desplazamiento", decimos que hay un* conflicto shift-reduce *o* conflicto de desplazamiento-reducción.

2. *Si las dos reglas indican una acción de reducción, decimos que tenemos un* conflicto reduce-reduce *o de* reducción-reducción.

Ejemplo 7.5.1. *Al aplicar el algoritmo 7.5.4 a la gramática 7.4.1*

1	$S \rightarrow a\ S\ b$
2	$S \rightarrow \epsilon$

partiendo del autómata finito determinista que se construyó en la figura 7.3 y calculando los conjuntos de primeros y siguientes

	FIRST	*FOLLOW*
S	a, ϵ	$b, \$$

obtenemos la siguiente tabla de acciones SLR:

	a	*b*	*$*
0	*s2*	*r2*	*r2*
1			*aceptar*
2	*s2*	*r2*	*r2*
4		*s5*	
5		*r1*	*r1*

Las entradas denotadas con s n (s por shift) indican un desplazamiento al estado n, las denotadas con r n (r por reduce o reducción) indican una operación de reducción o antiderivación por la regla n. Las entradas vacías corresponden a acciones de error.

El método de análisis *LALR* usado por `jison` es una extensión del método SLR esbozado aqui. Supone un compromiso entre potencia (conjunto de gramáticas englobadas) y eficiencia (cantidad de memoria utilizada, tiempo de proceso). Veamos como `jison` aplica la construcción del subconjunto a la gramática del ejemplo 7.4.1. Para ello construimos el siguiente programa `jison`:

```
[~/srcPLgrado/aSb(develop)]$ cat -n aSb.jison
     1  %lex
     2  %%
     3  .               { return yytext; }
     4  /lex
     5  %%
     6  P: S            { return $1; }
     7  ;
     8  S: /* empty */  { console.log("empty");    $$ = ''; }
     9     | 'a' S 'b'  { console.log("S -> aSb"); $$ = $1+$2+$3; }
    10  ;
    11  %%
```

y lo compilamos con `jison`. Estas son las opciones disponibles:

```
nereida:[~/PLgradoBOOK(eps)]$ jison --help

Usage: jison [file] [lexfile] [options]

file        file containing a grammar
lexfile     file containing a lexical grammar

Options:
   -o FILE, --outfile FILE       Filename and base module name of the generated parser
   -t, --debug                   Debug mode
   -t TYPE, --module-type TYPE   The type of module to generate (commonjs, amd, js)
   -V, --version                 print version and exit
```

Desafortunadamente carece de la típica opción `-v` que permite generar las tablas de análisis. Podemos intentar usar `bison`, pero, obviamente, `bison` protesta ante la entrada:

```
[~/srcPLgrado/aSb(develop)]$ bison -v aSb.jison
aSb.jison:1.1-4: invalid directive: ‘%lex’
aSb.jison:3.1: syntax error, unexpected identifier
aSb.jison:4.1: invalid character: ‘/’
```

El error es causado por la presencia del analizador léxico empotrado en el fichero `aSb.jison`. Si suprimimos provisionalmente las líneas del analizador léxico empotrado, `bison` es capaz de analizar la gramática:

```
[~/srcPLgrado/aSb(develop)]$ bison -v aSb.jison
[~/srcPLgrado/aSb(develop)]$ ls -ltr | tail -1
-rw-rw-r--  1 casiano  staff    926 19 mar 13:29 aSb.output
```

Que tiene los siguientes contenidos:

```
[~/srcPLgrado/aSb(develop)]$ cat -n aSb.output
     1  Grammar
     2
     3      0 $accept: P $end
     4
     5      1 P: S
     6
     7      2 S: /* empty */
     8      3  | 'a' S 'b'
     9
    10
    11  Terminals, with rules where they appear
    12
    13  $end (0) 0
    14  'a' (97) 3
    15  'b' (98) 3
    16  error (256)
    17
    18
    19  Nonterminals, with rules where they appear
    20
    21  $accept (5)
    22      on left: 0
    23  P (6)
    24      on left: 1, on right: 0
    25  S (7)
    26      on left: 2 3, on right: 1 3
    27
    28
    29  state 0
    30
    31      0 $accept: . P $end
    32
    33      'a'  shift, and go to state 1
    34
    35      $default  reduce using rule 2 (S)
    36
    37      P  go to state 2
    38      S  go to state 3
    39
```

```
state 1

    3 S: 'a' . S 'b'

    'a'  shift, and go to state 1

    $default  reduce using rule 2 (S)

    S  go to state 4


state 2

    0 $accept: P . $end

    $end  shift, and go to state 5


state 3

    1 P: S .

    $default  reduce using rule 1 (P)


state 4

    3 S: 'a' S . 'b'

    'b'  shift, and go to state 6


state 5

    0 $accept: P $end .

    $default  accept


state 6

    3 S: 'a' S 'b' .

    $default  reduce using rule 3 (S)
```

Observe que el final de la entrada se denota por `$end` y el marcador en un LR-item por un punto. Fíjese en el estado 1: En ese estado están también los items

`S -> . 'a' S 'b'` y `S -> .`

sin embargo no se explicitan por que se entiende que su pertenencia es consecuencia directa de aplicar la operación de clausura. Los LR items cuyo marcador no está al principio se denominan *items núcleo*.

7.6. Práctica: Analizador de PL0 Usando Jison

Reescriba el analizador sintáctico del lenguaje PL0 realizado en las prácticas 4.6 y 6.13 usando Jison .

Donde

- Repositorio en GitHub
- Despliegue en Heroku
- `[~/jison/jisoncalc(develop)]$ pwd -P`
  `/Users/casiano/local/src/javascript/PLgrado/jison/jisoncalc`
- ```
  [~/jison/jisoncalc(develop)]$ git remote -v
  heroku  git@heroku.com:jisoncalc.git (fetch)
  heroku  git@heroku.com:jisoncalc.git (push)
  origin  git@github.com:crguezl/ull-etsii-grado-pl-jisoncalc.git (fetch)
  origin  git@github.com:crguezl/ull-etsii-grado-pl-jisoncalc.git (push)
  ```

Tareas

- La salida debe ser el AST del programa de entrada
- Modifique `block` y `statement` para que los `procedure` reciban argumentos y las llamadas a procedimiento puedan pasar argumentos.
- Añada `if ... then ... else ....`
- Actualice la documentación de la gramática para que refleje la gramática ampliada
- Limite el número de programas que se pueden salvar a un número prefijado, por ejemplo 10. Si se intenta salvar uno se suprime uno al azar y se guarda el nuevo.
- Las pruebas deben comprobar que los AST generados reflejan la semántica del lenguaje así como alguna situación de error
- Sólo usuarios autenticados pueden salvar sus programas en la base de datos.
- Extienda la autenticación OAuth para que además de Google pueda hacerse con Twitter ó GitHub ó Facebook ó ... Sólo debe implementar una.
- Método de Entrega:
    - Use un repositorio privado en BitBucket o bien solicite al administrador del Centro de Cálculo un repositorio privado en GitHub.
    - Comparta dicho repositorio con sus colaboradores y con el profesor.
    - Suba la práctica al workshop/taller antes de la fecha límite
    - Cuando el taller pase a la fase de evaluación haga público su repositorio

Referencias para esta Práctica

- Véase el capítulo *Oauth: Google, Twitter, GitHub, Facebook* 34
- Véase Intridea Omniauth y omniauth en GitHub
- La gema omniauth-google-oauth2
- Google Developers Console
  ```

- Revoking Access to an App in Google
- La gema sinatra-flash
- Véase el capítulo *Heroku* 31
- Heroku Postgres
- Véase el capítulo *DataMapper* 32

7.7. Práctica: Análisis de Ámbito en PL0

Objetivos

- Modifique la práctica anterior para que cada nodo del tipo `PROCEDURE` disponga de una tabla de símbolos en la que se almacenan todos las constantes, variables y procedimientos declarados en el mismo.
- Existirá ademas una tabla de símbolos asociada con el nodo raíz que representa al programa principal.
- Las declaraciones de constantes y variables no crean nodo, sino que se incorporan como información a la tabla de símbolos del procedimiento actual
- Para una entrada de la tabla de símbolos `sym["a"]` se guarda que clase de objeto es: constante, variable, procedimiento, etc.
- Si es un procedimiento se guarda el número de argumentos
- Si es una constante se guarda su valor
- Cada uso de un identificador (constante, variable, procedimiento) tiene un atributo `declared_in` que referencia en que nodo se declaró
- Si un identificador es usado y no fué declarado es un error
- Si se trata de una llamada a procedimiento (se ha usado `CALL` y el identificador corresponde a un `PROCEDURE`) se comprobará que el número de argumentos coincide con el número de parámetros declarados en su definición
- Si es un identificador de una constante, es un error que sea usado en la parte izquierda de una asignación (que no sea la de su declaración)
- Base de Datos
  1. Guarde en una tabla el nombre de usuario que guardó un programa. Provea una ruta para ver los programas de un usuario.
  2. Un programa `belongs_to` un usuario. Un usuario `has n` programas. Vea la sección DataMapper Associat
- Use la sección `issues` de su repositorio en GitHub para coordinarse así como para llevar un histórico de las incidencias y la forma en la que se resolvieron. Repase el tutorial Mastering Issues

7.8. Práctica: Traducción de Infijo a Postfijo

Modifique el programa Jison realizado en la práctica 7.3.1 para traducir de infijo a postfijo. Añada los operadores de comparación e igualdad. Por ejemplo

Infijo	Postfijo
`a = 3+2*4`	`3 2 4 * + &a =`
`b = a == 11`	`a 11 == &b =`

En estas traducciones la notación `&a` indica la dirección de la variable `a` y `a` indica el valor almacenado en la variable `a`.

Añada sentencias `if` ... `then` e `if` ... `then` ... `else`

Para realizar la traducción de estas sentencias añada instrucciones `jmp label` y `jmpz label` (por *jump if zero*) y etiquetas:

Infijo

```
a = (2+5)*3;
if a == 0 then b = 5 else b = 3;
c = b + 1;
```

Postfijo

```
        2
        5
        +
        3
        *
        &a
        =
        a
        0
        ==
        jmpz else1
        5
        &b
        =
        jmp endif0
:else1
        3
        &b
        =
:endif0
        b
        1
        +
        &c
        =
```

Parta del repositorio https://github.com/crguezl/jison-simple-html-calc.

7.9. Práctica: Calculadora con Funciones

Añada funciones y sentencias de llamada a función a la práctica de traducción de infijo a postfijo 7.8. Sigue un ejemplo de traducción:

<pre>def f(x) { x + 1 }
def g(a, b) { a * f(b) }
c = 3;
f(1+c);
g(3, 4)</pre>
	<pre>:f args :x
	        $x
	        1
	        +
	        return
	:g args :a,:b
	        $a
	        $b
	        call :f
	        *
	        return
	:main:
	        3
	        &c
	        =
	        1
	        c
	        +
	        call :f
	        3
	        4
	        call :g</pre>

- Las funciones retornan la última expresión evaluada
- Es un error llamar a una función con un número de argumentos distinto que el número de parámetros con el que fué declarada
- En la llamada, los argumentos se empujan en la pila. Después la instrucción `call :etiqueta` llama a la función con el nombre dado por la `etiqueta`
- Dentro de la función los argumentos se sitúan por encima del puntero base. La pseudo-instrucción `args, p1, p2, ...` da nombre a los parámetros empujados. Dentro del cuerpo de la función nos referimos a ellos prefijándolos con `$`.
- La instrucción `return` limpia la pila dejándola en su estado anterior y retorna la última expresión evaluada

7.10. Práctica: Calculadora con Análisis de Ámbito

Extienda la práctica anterior para que haga un análisis completo del ámbito de las variables.

- Añada declaraciones de variable con `var x, y = 1, z`. Las variables podrán opcionalmente ser inicializadas. Se considerará un error usar una variable no declarada.
- Modifique la gramática para que permita el anidamiento de funciones: funciones dentro de funciones.

```
var c = 4, d = 1, e;
def g(a, b) {
  var d, e;
  def f(u, v) { a + u + v + d }
  a * f(b, 2) + d + c
}
```

- Una declaración de variable en un ámbito anidado tapa a una declaración con el mismo nombre en el ámbito exterior.

<pre>var c = 4, d = 1, e;
def g(a, b) {
  var d, e; # esta "d" tapa la d anterior
  def f(u, v) { a + u + v + d }
  a * f(b, 2) + d + c
}</pre>
	<pre> # global: var c,d,e
	:g.f
	        $a, 1
	        $u, 0
	        +
	        $v, 0
	        +
	        d, 1
	        +
	        return
	:g
	        $a, 0
	        $b, 0
	        2
	        call :g.f
	        *
	        d, 0 # acceder a la d en el ámbito actual
	        +
	        c, 1
	        +
	        return</pre>

- Los nombres de funciones se traducen por una secuencia anidada de nombres que indican su ámbito. Así la función `f` anidada en `g` es traducida a la función con nombre `g.f`. Una función `h` anidada en una función `f` anidada en `g` es traducida a la función con nombre `g.f.h`

- Las variables ademas de su nombre (dirección/offset) reciben un entero adicional 0,1,2, ... que indica su nivel de anidamiento. El número de stack frames que hay que recorrer para llegar a la variable

```
$a, 1
$u, 0
+
$v, 0
+
d, 1
+
```

  Asi `$a, 1` significa acceder al parámetro `a` que está a distancia `1` del stack frame/ámbito actual y `$v, 0` es el parámetro `v` en el ámbito/stack frame actual

- El frame pointer o base pointer BP indica el nivel de anidamiento estático (en el fuente) de la rutina. Así cuando se va a buscar una variable local declarada en la rutina que anida la actual se recorre la lista de frames via BP o frame pointer tantas veces como el nivel de anidamiento indique.

- 1. Esto es lo que dice la Wikipedia sobre la implementación de llamadas a subrutinas anidadas:

  > *Programming languages that support nested subroutines also have a field in the call frame that points to the stack frame of the latest activation of the procedure that most closely encapsulates the callee, i.e. the immediate scope of the callee. This is*

> *called an* access link *or* static link *(as it keeps track of static nesting during dynamic and recursive calls) and provides the routine (as well as any other routines it may invoke) access to the local data of its encapsulating routines at every nesting level.*

2. Esto es lo que dice sobre las ventajas de tener una pila y de almacenar la dirección de retorno y las variables locales:

   > When a subroutine is called, the location (address) of the instruction at which it can later resume needs to be saved somewhere. Using a stack to save the return address has important advantages over alternatives. One is that each task has its own stack, and thus the subroutine can be *reentrant*, that is, can be active simultaneously for different tasks doing different things. Another benefit is that *recursion* is automatically supported. When a function calls itself recursively, a return address needs to be stored for each activation of the function so that it can later be used to return from the function activation. This capability is automatic with a stack.

3. Almacenamiento local:

   > A subroutine frequently needs memory space for storing the values of local variables, the variables that are known only within the active subroutine and do not retain values after it returns. It is often convenient to allocate space for this use by simply moving the top of the stack by enough to provide the space. This is very fast compared to heap allocation. Note that each separate activation of a subroutine gets its own separate space in the stack for locals.

4. Parámetros:

   > Subroutines often require that values for parameters be supplied to them by the code which calls them, and it is not uncommon that space for these parameters may be laid out in the call stack.
   > The call stack works well as a place for these parameters, especially since each call to a subroutine, which will have differing values for parameters, will be given separate space on the call stack for those values.

5. Pila de Evaluación

   > Operands for arithmetic or logical operations are most often placed into registers and operated on there. However, in some situations the operands may be stacked up to an arbitrary depth, which means something more than registers must be used (this is the case of *register spilling*). The stack of such operands, rather like that in an RPN calculator, is called an *evaluation stack*, and may occupy space in the call stack.

6. Puntero a la instancia actual

   > Some object-oriented languages (e.g., C++), store the `this` pointer along with function arguments in the call stack when invoking methods. The `this pointer` points to the object instance associated with the method to be invoked.

- Los parámetros se siguen prefijando de `$` como en la práctica anterior

- Sigue un ejemplo de traducción:

<pre>var c = 4, d = 1, e;
def f(x) {
  var y = 1;
  x + y
}
def g(a, b) {
  var d, e;
  def f(u, v) { a + u + v + d }
  a * f(b, 2) + d + c
}
c = 3;
f(1+c);
g(3, 4)</pre>
	<pre> # global: var c,
	        # f: args x
	        # f: var y
	:f
	        1
	        &y, 0
	        =
	        $x, 0
	        y, 0
	        +
	        return
	        # g: args a,b
	        # g: var d,e
	        # g.f: args u,v
	:g.f
	        $a, 1
	        $u, 0
	        +
	        $v, 0
	        +
	        d, 1
	        +
	        return
	:g
	        $a, 0
	        $b, 0
	        2
	        call :g.f
	        *
	        d, 0
	        +
	        c, 1
	        +
	        return
	:main:
	        4
	        &c, 0
	        =
	        1
	        &d, 0
	        =
	        3
	        &c, 0
	        =
	        1
	        c, 0
	        +
	        call :f
	        3
	        4
	        call :g</pre>

- Puede comenzar haciendo un fork del proyecto ull-etsii-grado-pl-infix2postfix en GitHub. Esta incompleto. Rellene las acciones semánticas que faltan; la mayoría relacionadas con el análisis de ámbito.

-
    - Una solución completa se encuentra en el proyecto crguezl/jisoninfix2postfix.
    - `[~/jison/jisoninfix2postfix(gh-pages)]$ pwd -P`
      `/Users/casiano/local/src/javascript/PLgrado/jison/jisoninfix2postfix`
    - ```
      [~/jison/jisoninfix2postfix(gh-pages)]$ git remote -v
      bitbucket       ssh://git@bitbucket.org/casiano/jisoninfix2postfix.git (fetch)
      bitbucket       ssh://git@bitbucket.org/casiano/jisoninfix2postfix.git (push)
      origin  git@github.com:crguezl/jisoninfix2postfix.git (fetch)
      origin  git@github.com:crguezl/jisoninfix2postfix.git (push)
      ```

- Veanse:
    - Véase COMP 3290 Compiler Construction Fall 2008 Notes/Symbol Tables
    - El capítulo Symbol Table Structure del libro de Muchnick Advanced Compiler Design Implementation [6]
    - El capítulo Symbol Table Structure del libro de Basics of Compiler Design de Torben Ægidius Mogensen [7]

7.11. Algoritmo de Análisis LR

Asi pues la tabla de transiciones del autómata nos genera dos tablas: la tabla de acciones y la de saltos. El algoritmo de análisis sintáctico *LR* en el que se basa *jison* utiliza una pila y dos tablas para analizar la entrada. Como se ha visto, la tabla de acciones contiene cuatro tipo de acciones:

1. Desplazar (*shift*)
2. Reducir (*reduce*)
3. Aceptar
4. Error

El algoritmo utiliza una pila en la que se guardan los estados del autómata. De este modo se evita tener que "comenzar" el procesado de la forma sentencial derecha resultante después de una reducción (antiderivación).

Algoritmo 7.11.1. *Análizador LR*

```
push(s0);
b = yylex();
for( ; ; ;) {
  s = top(0); a = b;
  switch (action[s][a]) {
    case "shift t" :
      t.attr = a.attr;
      push(t);
      b = yylex();
      break;
    case "reduce A ->alpha" :
      eval(Sem{A -> alpha}(top(|alpha|-1).attr, ... , top(0).attr));
      pop(|alpha|);
      push(goto[top(0)][A]);
      break;
```
```

```
      case "accept" : return (1);
      default : yyerror("syntax error");
    }
 }
```

- Como es habitual, $|x|$ denota la longitud de la cadena x.
- La función `top(k)` devuelve el elemento que ocupa la posición `k` desde el *top* de la pila (esto es, está a profundidad `k`).
- La función `pop(k)` extrae `k` elementos de la pila.
- La notación `state.attr` hace referencia al atributo asociado con cada estado, el cual desde el punto de vista del programador esta asociado con el correspondiente símbolo de la parte derecha de la regla. Nótese que cada estado que está en la pila es el resultado de una transición con un símbolo. El atributo de ese símbolo es guardado en el objeto estado cada vez que ocurre una transición.
- Denotamos por `Sem {reduce A -> alpha}` el código de la acción semántica asociada con la regla $A \to \alpha$.

Todos los analizadores LR comparten, salvo pequeñas excepciones, el mismo algoritmo de análisis. Lo que más los diferencia es la forma en la que construyen las tablas. En `jison` la construcción de las tablas de *acciones* y *gotos* se realiza por defecto mediante el algoritmo *LALR*.

7.12. El módulo Generado por `jison`

7.12.1. Version

En esta sección estudiamos el analizador generado por Jison:

```
[~/Dropbox/src/javascript/PLgrado/jison-aSb(develop)]$ jison --version
0.4.2
```

7.12.2. Gramática Inicial

Veamos el módulo generado por jison para esta gramática:

```
[~/srcPLgrado/aSb(develop)]$ cat aSb.jison
%lex
%%
.               { return yytext; }
/lex
%%
S: /* empty */  { console.log("empty"); }
   | 'a' S 'b'  { console.log("S -> aSb"); }
;
%%
```

7.12.3. Tablas

Esta es la primera parte del parser generado:

```
/* parser generated by jison 0.4.2 */
var aSb = (function() {
    var parser = {
        trace: function trace() {},
```

```
        yy: {},
        symbols_: {
            "$accept": 0, /* super-arranque $accept -> S */
            "$end": 1     /* end of input */
            "error": 2, /* numero para el símbolo 'error' */
            "S": 3,     /* numero para el símbolo 'S' */
            "a": 4,
            "b": 5,
        },
        /* array inverso de terminales */
        terminals_: {   /* numero -> terminal */
            2: "error",
            4: "a",
            5: "b"
        },
        productions_:
        [0,
/* 1 */     [3, 0], /* S : vacio        simbolo,longitud de la parte derecha */
/* 2 */     [3, 3]  /* S : a S b        simbolo,longitud */
        ],
```

7.12.4. Acciones Semánticas

Cada vez que se produce una acción de reducción esta función es llamada:

```
performAction: function anonymous(yytext, yyleng, yylineno, yy, yystate, $$, _$) {

    var $0 = $$.length - 1;
    switch (yystate) { /* yystate: numero de regla de producción */
        case 1:
            console.log("empty");
            break;
        case 2:
            console.log("S -> aSb");
            break;
    }
},
```

- Parece que cuando se llama a este método `this` refiere a un objeto `yyval`. Este es el punto de llamada a la acción semántica dentro del parser generado por Jison. Puede encontrarse dentro del parser en el caso de un `switch` que corresponde a la acción de reducción:

  ```
  r = this.performAction.call(yyval, yytext, yyleng, yylineno, this.yy, action[1], vstack, l
  ```

  El método `call` nos permite invocar una función como si fuera un método de algún otro objeto. Véase la sección ??.

  Este objeto `yyval` tiene dos atributos: `$` y `_$`.

  - El atributo `$` se corresponde con `$$` de la gramática (atributo de la variable sintactica en la parte izquierda)
  - El atributo `_$` guarda información sobre la posición del último token leído.

- `yytext` parece contener el texto asociado con el token actual

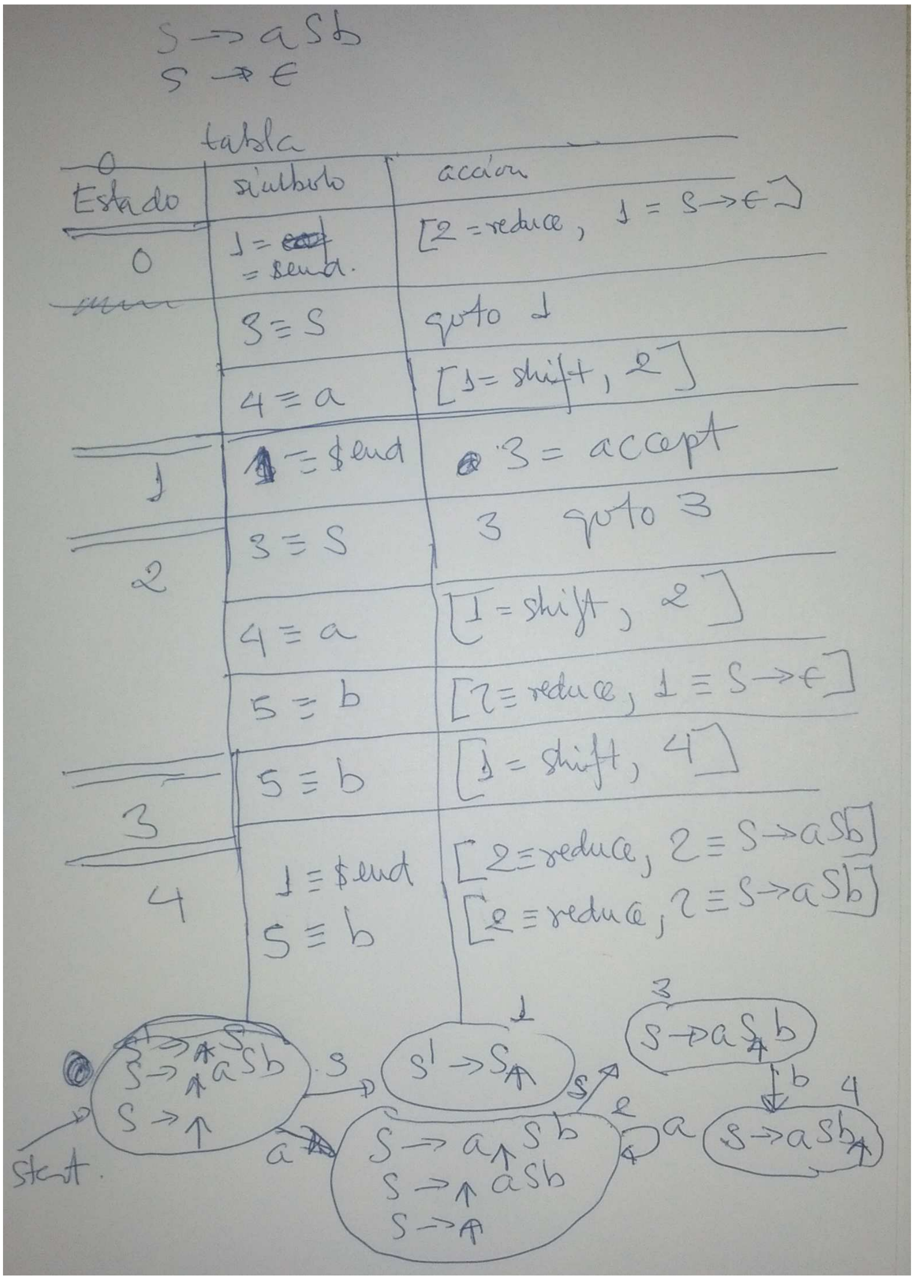


Figura 7.3: DFA construido por Jison

- `yyleng` es la longitud del token actual
- `yylineno` es la línea actual (empezando en 0)
- `yy` es un objeto con dos atributos `lexer` y `parser`
- `yystate` es el estado actual
- `$$` parece ser un array/pila conteniendo los valores de los atributos asociados con los estados de la pila (`vstack` ¿Por value stack?)
- Asi pues `$0` es el índice en `$0` del último elemento de `$$`. Por ejemplo, una acción semántica asociada con una regla `A : B C D` con tres elementos como:

  ```
  $$ = $1 + $2 + $3;
  ```

  Se traduce por:

```
this.$ = $$[$0 - 2] + $$[$0 - 1] + $$[$0];
```

- `_$` Es un array con la información sobre la localización de los simbolos (`lstack` ¿Por location stack?)

7.12.5. Tabla de Acciones y GOTOs

```
        table: [{
/* 0 */     1: [2, 1],    /* En estado 0 viendo $end(1) reducir por S : vacio */
            3: 1,         /* En el estado 0 viendo S(3) ir al estado 1 */
            4: [1, 2]     /* Estado 0 viendo a(4) shift(1) al estado 2 */
        }, {
/* 1 */     1: [3]        /* En 1 viendo $end(1) aceptar */
        }, {
/* 2 */     3: 3,         /* En 2 viendo S ir a 3 */
            4: [1, 2],    /* En 2 viendo a(4) shift a 2 */
            5: [2, 1]     /* En 2 viendo b(5) reducir por regla 1: S -> vacio */
        }, {
/* 3 */     5: [1, 4]     /* En 3 viendo b(5) shift a 4 */
        }, {
/* 4 */     1: [2, 2],    /* En 4  viendo $end(1) reducir(2) por la 2: S -> aSb */
            5: [2, 2]     /* En 4 viendo b(5) reducir por la 2: S-> aSb */
        }],
```

- La tabla es un array de objetos
- El índice de la tabla es el estado. En el ejemplo tenemos 5 estados
- El objeto/hash que es el valor contiene las acciones ante los símbolos.
  1. Los atributos/claves son los símbolos, los valores las acciones
  2. Las acciones son de dos tipos:
     a) El número del estado al que se transita mediante la tabla `goto` cuando el símbolo es una variable sintactica
     b) Un par `[tipo de acción, estado o regla]`. Si el `tipo de acción` es 1 indica un shift al `estado` con ese número. Si el `tipo de acción` es 2 indica una reducción por la `regla` con ese número.
  3. Por ejemplo `table[0]` es

```
{
    1: [2, 1],    /* En estado 0 viendo $end(1) reducir(2) por S : vacio */
    3: 1,         /* En el estado 0 viendo S(3) ir (goto) al estado 1 */
    4: [1, 2]     /* Estado 0 viendo a(4) shift(1) al estado 2 */
}
```

7.12.6. defaultActions

```
defaultActions: {},
```

- `defaultActions` contiene las acciones por defecto.
- Después de la construcción de la tabla, Jison identifica para cada estado la reducción que tiene el conjunto de lookaheads mas grande. Para reducir el tamaño del parser, Jison puede decidir suprimir dicho conjunto y asiganr esa reducción como acción del parser por defecto. Tal reducción se conoce como *reducción por defecto.*

- Esto puede verse en este segmento del código del parser:

```
while (true) {
      state = stack[stack.length - 1];
      if (this.defaultActions[state]) {
          action = this.defaultActions[state];
      } else {
          if (symbol === null || typeof symbol == "undefined") {
              symbol = lex();
          }
          action = table[state] && table[state][symbol];
      }
      ...
}
```

7.12.7. Reducciones

```
parse: function parse(input) {
    ...
    while (true) {
        state = stack[stack.length - 1];
        if (this.defaultActions[state]) {
            action = this.defaultActions[state];
        } else {
            if (symbol === null || typeof symbol == "undefined") {
                symbol = lex(); /* obtener siguiente token */
            }
            action = table[state] && table[state][symbol];
        }
        if (typeof action === "undefined" || !action.length || !action[0]) {
          ... // error
        }
        if (action[0] instanceof Array && action.length > 1) {
            throw new Error("Parse Error: multiple actions possible at state: ..."
        }
        switch (action[0]) {
            case 1:                                    // shift
                ...
                break;
            case 2:                                    // reduce
                len = this.productions_[action[1]][1]; // longitud de la producción
                yyval.$ = vstack[vstack.length - len];
                yyval._$ = {                           // datos de la posición
                    first_line: lstack[lstack.length - (len || 1)].first_line,
                    last_line: lstack[lstack.length - 1].last_line,
                    first_column: lstack[lstack.length - (len || 1)].first_column,
                    last_column: lstack[lstack.length - 1].last_column
                };
                ...
                r = this.performAction.call(yyval, yytext, yyleng, yylineno, this.yy, action[1
                if (typeof r !== "undefined") {
                    return r; /* un return de algo distinto de undefined nos saca del parser *
                }
                if (len) {                                     /* retirar de las pilas */
```

```
                    stack = stack.slice(0, - 1 * len * 2);  /* simbolo, estado, simbolo, estad
                    vstack = vstack.slice(0, - 1 * len);    /* retirar atributos */
                    lstack = lstack.slice(0, - 1 * len);    /* retirar localizaciones */
                }
                stack.push(this.productions_[action[1]][0]); /* empujemos el símbolo */
                vstack.push(yyval.$);                        /* empujemos valor semantico */
                lstack.push(yyval._$);                       /* empujemos localización */
                newState = table[stack[stack.length - 2]][stack[stack.length - 1]];
                stack.push(newState);                        /* empujemos goto[top][A]*/
                break;
            case 3: // accept
                return true;
        }
    }
    return true;
}
```

7.12.8. Desplazamientos/Shifts

```
parse: function parse(input) {
    ...
    while (true) {
        state = stack[stack.length - 1];     /* estado en el top de la pila */
        if (this.defaultActions[state]) {    /* definida la acción por defecto? */
            action = this.defaultActions[state];
        } else {
            if (symbol === null || typeof symbol == "undefined") {
                symbol = lex();              /* obtener token */
            }
            action = table[state] && table[state][symbol]; /* obtener la acción para el estado
        }
        if (typeof action === "undefined" || !action.length || !action[0]) {
            ... /* error */
        }
        if (action[0] instanceof Array && action.length > 1) {
            throw new Error("Parse Error: multiple actions possible at state: " + state + ", t
        }
        switch (action[0]) {
            case 1:
                stack.push(symbol);                 /* empujamos token */
                vstack.push(this.lexer.yytext);     /* empujamos el atributo del token */
                lstack.push(this.lexer.yylloc);     /* salvamos la localización del token */
                stack.push(action[1]);              /* salvamos el estado */
                symbol = null;
                if (!preErrorSymbol) {              /* si no hay errores ... */
                    yyleng = this.lexer.yyleng;     /* actualizamos los atributos */
                    yytext = this.lexer.yytext;     /* del objeto */
                    yylineno = this.lexer.yylineno;
                    yyloc = this.lexer.yylloc;
                    if (recovering > 0) recovering--; /* las cosas van mejor si hubieron error
                } else {
                    symbol = preErrorSymbol;
                    preErrorSymbol = null;
                }
```

```
                break;
            case 2:
                ...
                break;
            case 3:
                return true;
        }
    }
    return true;
}
```

7.12.9. Manejo de Errores

```
while (true) {
    state = stack[stack.length - 1];
    if (this.defaultActions[state]) { action = this.defaultActions[state]; }
    else {
        if (symbol === null || typeof symbol == "undefined") { symbol = lex(); }
        action = table[state] && table[state][symbol];
    }
    if (typeof action === "undefined" || !action.length || !action[0]) {
        var errStr = "";
        if (!recovering) { /* recovering = en estado de recuperación de un error */
            expected = [];                       /* computemos los tokens esperados */
            for (p in table[state])              /* si el estado "state" transita con p */
              if (this.terminals_[p] && p > 2) { /* y "p" es un terminal no especial */
                  expected.push("'" + this.terminals_[p] + "'"); /* entonces es esperado */
              }
            if (this.lexer.showPosition) { /* si esta definida la función showPosition */
                errStr = "Parse error on line " + (yylineno + 1) +
                         ":\n" + this.lexer.showPosition() +
                         "\nExpecting " + expected.join(", ") +
                         ", got '" +
                         (this.terminals_[symbol] || symbol) + /* terminals_ es el array inver
                         "'";                                  /* numero -> terminal
            } else { /* ¡monta la cadena como puedas! */
                errStr = "Parse error on line " + (yylineno + 1) +
                         ": Unexpected " +
                         (symbol == 1 ? "end of input" : "'" +
                         (this.terminals_[symbol] || symbol) + "'");
            }
            this.parseError(errStr, {    /* genera la excepción */
                text: this.lexer.match,  /* hash/objeto conteniendo los detalles del */
                token: this.terminals_[symbol] || symbol,                   /* error */
                line: this.lexer.yylineno,
                loc: yyloc,
                expected: expected
            });
        }
    }
    if (action[0] instanceof Array && action.length > 1) {
        throw new Error("Parse Error: multiple actions possible at state: " + state + ", token
    }
    ...
```

```
}
```

La función `parseError` genera una excepción:

```
parseError: function parseError(str, hash) {
    throw new Error(str); /* El hash contiene info sobre el error: token, linea, etc.
},
```

- `parseError` es llamada cada vez que ocurre un error sintáctico. `str` contiene la cadena con el mensaje de error del tipo: `Expecting something, got other thing'`. `hash` contiene atributos como `expected`: el array de tokens esperados; `line` la línea implicada, `loc` una descripción de la localización detallada del punto/terminal en el que ocurre el error; etc.

7.12.10. Analizador Léxico

El analizador léxico:

```
/* generated by jison-lex 0.1.0 */
var lexer = (function() {
    var lexer = {
        EOF: 1,
        parseError: function parseError(str, hash) { /* manejo de errores léxicos */ },
        setInput: function(input) { /* inicializar la entrada para el analizadorléxico */},
        input: function() { /* ... */ },
        unput: function(ch) { /* devolver al flujo de entrada */ },
        more: function() { /* ... */ },
        less: function(n) { /* ... */ },
        pastInput: function() { /* ... */ },
        upcomingInput: function() { /* ... */ },
        showPosition: function() { /* ... */ },
        next: function() {
                if (this.done) { return this.EOF; }
                if (!this._input) this.done = true;

                var token, match, tempMatch, index, col, lines;
                if (!this._more) { this.yytext = ''; this.match = ''; }
                var rules = this._currentRules();
                for (var i = 0; i < rules.length; i++) {
                    tempMatch = this._input.match(this.rules[rules[i]]);
                    if (tempMatch && (!match || tempMatch[0].length > match[0].length)) {
                        match = tempMatch;
                        index = i;
                        if (!this.options.flex) break;
                    }
                }
                if (match) {
                    lines = match[0].match(/(?:\r\n?|\n).*/g);
                    if (lines) this.yylineno += lines.length;
                    this.yylloc = {
                        first_line: this.yylloc.last_line,
                        last_line: this.yylineno + 1,
                        first_column: this.yylloc.last_column,
                        last_column:
                          lines ? lines[lines.length - 1].length -
                                  lines[lines.length - 1].match(/\r?\n?/)[0].length
```

```
                            :
                            this.yylloc.last_column + match[0].length
            };
            this.yytext += match[0];
            this.match += match[0];
            this.matches = match;
            this.yyleng = this.yytext.length;
            if (this.options.ranges) {
                this.yylloc.range = [this.offset, this.offset += this.yyleng];
            }
            this._more = false;
            this._input = this._input.slice(match[0].length);
            this.matched += match[0];
            token = this.performAction.call(
                         this,
                         this.yy,
                         this,
                         rules[index],
                         this.conditionStack[this.conditionStack.length - 1]
                    );
            if (this.done && this._input) this.done = false;
            if (token) return token;
            else return;
        }
        if (this._input === "") { return this.EOF; }
        else {
            return this.parseError(
                     'Lexical error on line ' + (this.yylineno + 1) +
                      '. Unrecognized text.\n' + this.showPosition(),
                      { text: "", token: null, line: this.yylineno }
                   );
        }
    },
lex: function lex() {
    var r = this.next();
    if (typeof r !== 'undefined') {
        return r;
    } else {
        return this.lex();
    }
},
begin: function begin(condition) { },
popState: function popState() { },
_currentRules: function _currentRules() { },
topState: function() { },
pushState: function begin(condition) { },
options: {},
performAction: function anonymous(yy, yy_, $avoiding_name_collisions, YY_START)
{
    var YYSTATE = YY_START;
    switch ($avoiding_name_collisions) {
        case 0:
            return yy_.yytext;
```

```
                    break;
                }
        },
        rules: [/^(?:.)/], /* lista de expresiones regulares */
        conditions: { /* ... */ }
  }
};
```

7.12.11. Exportación

Si no ha sido exportado ya ...

```
if (typeof require !== 'undefined' && typeof exports !== 'undefined') {
    exports.parser = aSb;         /* hacemos accesible el objeto aSb  */
    exports.Parser = aSb.Parser;
```

El objeto `aSb.Parser` representa al parser. Este es el código que lo crea.

```
      function Parser() {
          this.yy = {};
      }
      Parser.prototype = parser;
      parser.Parser = Parser;
      return new Parser;
  })();
```

También se exporta una función `parse`:

```
    exports.parse = function() {
        return aSb.parse.apply(aSb, arguments);
    };
```

y una función `main`:

```
    exports.main = function commonjsMain(args) {
        if (!args[1]) {
            console.log('Usage: ' + args[0] + ' FILE');
            process.exit(1);
        }
        var source = require('fs').readFileSync(require('path').normalize(args[1]), "utf8");
        return exports.parser.parse(source);
    };
    if (typeof module !== 'undefined' && require.main === module) {
        exports.main(process.argv.slice(1));
    }
}
```

Esto permite ejecutar el módulo directamente:

```
[~/Dropbox/src/javascript/PLgrado/jison-aSb(develop)]$ node aSb.js input.ab
empty
S -> aSb
S -> aSb
[~/Dropbox/src/javascript/PLgrado/jison-aSb(develop)]$ cat input.ab
aabb
```

```
~/Dropbox/src/javascript/PLgrado/jison-aSb(develop)]$ node debug aSb.js input.ab
< debugger listening on port 5858
connecting... ok
break in aSb.js:2
  1 /* parser generated by jison 0.4.2 */
  2 var aSb = (function() {
  3     var parser = {
  4         trace: function trace() {},
debug> n
break in aSb.js:390
 388     return new Parser;
 389 })();
 390 if (typeof require !== 'undefined' && typeof exports !== 'undefined') {
 391     exports.parser = aSb;
 392     exports.Parser = aSb.Parser;

debug> repl
Press Ctrl + C to leave debug repl
>
> typeof require
'function'
> typeof exports
'object'
> aSb
{ yy: {} }
> aSb.Parser
[Function]
^C
debug> sb(396)
 395     };
debug> c
break in aSb.js:396
 394         return aSb.parse.apply(aSb, arguments);
 395     };
*396     exports.main = function commonjsMain(args) {
 397         if (!args[1]) {
 398             console.log('Usage: ' + args[0] + ' FILE');
debug> n
break in aSb.js:404
 402         return exports.parser.parse(source);
 403     };
 404     if (typeof module !== 'undefined' && require.main === module) {
 405         exports.main(process.argv.slice(1));
 406     }
debug> repl
Press Ctrl + C to leave debug repl
> process.argv.slice(1)
[ '/Users/casiano/Dropbox/src/javascript/PLgrado/jison-aSb/aSb.js',
  'input.ab' ]
> typeof module
'object'
> require.main
{ id: '.',
  exports:
```

```
  { parser: { yy: {} },
    Parser: [Function],
    parse: [Function],
    main: [Function] },
 parent: null,
 filename: '/Users/casiano/Dropbox/src/javascript/PLgrado/jison-aSb/aSb.js',
 loaded: false,
 children: [],
 paths:
  [ '/Users/casiano/Dropbox/src/javascript/PLgrado/jison-aSb/node_modules',
    '/Users/casiano/Dropbox/src/javascript/PLgrado/node_modules',
    '/Users/casiano/Dropbox/src/javascript/node_modules',
    '/Users/casiano/Dropbox/src/node_modules',
    '/Users/casiano/Dropbox/node_modules',
    '/Users/casiano/node_modules',
    '/Users/node_modules',
    '/node_modules' ] }
^C
debug> n
break in aSb.js:405
 403     };
 404     if (typeof module !== 'undefined' && require.main === module) {
 405         exports.main(process.argv.slice(1));
 406     }
 407 }
debug> n
< empty
< S -> aSb
< S -> aSb
break in aSb.js:409
 407 }
 408
 409 });
debug> c
program terminated
debug>
```

7.13. Precedencia y Asociatividad

Recordemos que si al construir la tabla LALR, alguna de las entradas de la tabla resulta multievaluada, decimos que existe un conflicto. Si una de las acciones es de 'reducción" y la otra es de 'desplazamiento", se dice que hay un *conflicto shift-reduce* o *conflicto de desplazamiento-reducción*. Si las dos reglas indican una acción de reducción, decimos que tenemos un *conflicto reduce-reduce* o de *reducción-reducción*. En caso de que no existan indicaciones específicas *jison* resuelve los conflictos que aparecen en la construcción de la tabla utilizando las siguientes reglas:

1. Un conflicto *reduce-reduce* se resuelve eligiendo la producción que se listó primero en la especificación de la gramática.

2. Un conflicto *shift-reduce* se resuelve siempre en favor del *shift*

Las declaraciones de precedencia y asociatividad mediante las palabras reservadas `%left` , `%right` , `%nonassoc` se utilizan para modificar estos criterios por defecto. La declaración de `token` s mediante la palabra reservada `%token` no modifica la precedencia. Si lo hacen las declaraciones realizadas usando las palabras `left` , `right` y `nonassoc` .

1. Los *tokens* declarados en la misma línea tienen igual precedencia e igual asociatividad. La precedencia es mayor cuanto mas abajo su posición en el texto. Así, en el ejemplo de la calculadora en la sección **??**, el *token* `*` tiene mayor precedencia que `+` pero la misma que `/`.

2. La precedencia de una regla $A \to \alpha$ se define como la del terminal mas a la derecha que aparece en α. En el ejemplo, la producción

   ```
   expr : expr '+' expr
   ```

   tiene la precedencia del *token* `+`.

3. Para decidir en un conflicto *shift-reduce* se comparan la precedencia de la regla con la del terminal que va a ser desplazado. Si la de la regla es mayor se reduce si la del *token* es mayor, se desplaza.

4. Si en un conflicto *shift-reduce* ambos la regla y el terminal que va a ser desplazado tiene la misma precedencia *jison* considera la asociatividad, si es asociativa a izquierdas, reduce y si es asociativa a derechas desplaza. Si no es asociativa, genera un mensaje de error.
   Obsérvese que, en esta situación, la asociatividad de la regla y la del *token* han de ser por fuerza, las mismas. Ello es así, porque en *jison* los *tokens* con la misma precedencia se declaran en la misma línea y sólo se permite una declaración por línea.

5. *Por tanto es imposible declarar dos* tokens *con diferente asociatividad y la misma precedencia.*

6. Es posible modificar la precedencia "natural" de una regla, calificándola con un *token* específico. para ello se escribe a la derecha de la regla `prec token`, donde `token` es un *token* con la precedencia que deseamos. Vea el uso del *token* `dummy` en el siguiente ejercicio.

Para ilustrar las reglas anteriores usaremos el siguiente programa `jison`:

```
[~/jison/jison-prec(ast)]$ cat -n precedencia.jison
%token NUMBER
%left '@'
%right '&'  dummy
%%
s
    : list      { console.log($list); }
    ;

list
    :
                {
                  $$ = [];
                }
    | list '\n'
                {
                  $$ = $1;
                }
    | list e
                {
                  $$ = $1;
                  $$.push($e);
                }
```

```
    ;

e : NUMBER
                {
                  $$ = "NUMBER ("+yytext+")";
                }
  | e '&' e
                {
                  $$ = [ "&", $e1, $e2];
                }
  | e '@' e %prec dummy
                {
                  $$ = ["@", $e1, $e2];
                }
  ;

%%
```

Obsérvese la siguiente ejecución:

```
[~/jison/jison-prec(ast)]$ cat input.txt
2@3@4
2&3&4
[~/jison/jison-prec(ast)]$ node precedencia.js input.txt
[ [ '@', [ '@', 'NUMBER (2)', 'NUMBER (3)' ], 'NUMBER (4)' ],
  [ '&', 'NUMBER (2)', [ '&', 'NUMBER (3)', 'NUMBER (4)' ] ] ]
```

Compilamos a continuación con `bison` usando la opción `-v` para producir información sobre los conflictos y las tablas de salto y de acciones:

```
[~/jison/jison-prec(ast)]$ bison -v precedencia.jison
precedencia.jison:6.31: warning: stray `$'
precedencia.jison:21.27: warning: stray `$'
precedencia.jison:31.31: warning: stray `$'
precedencia.jison:31.36: warning: stray `$'
precedencia.jison:35.30: warning: stray `$'
precedencia.jison:35.35: warning: stray `$'
```

La opción `-v` genera el fichero `Precedencia.output` el cual contiene información detallada sobre el autómata:

```
[~/jison/jison-prec(ast)]$ cat precedencia.output
Grammar

    0 $accept: s $end

    1 s: list

    2 list: /* empty */
    3     | list '\n'
    4     | list e

    5 e: NUMBER
    6  | e '&' e
    7  | e '@' e
```

```
Terminals, with rules where they appear

$end (0) 0
'\n' (10) 3
'&' (38) 6
'@' (64) 7
error (256)
NUMBER (258) 5
dummy (259)


Nonterminals, with rules where they appear

$accept (8)
    on left: 0
s (9)
    on left: 1, on right: 0
list (10)
    on left: 2 3 4, on right: 1 3 4
e (11)
    on left: 5 6 7, on right: 4 6 7


state 0

    0 $accept: . s $end

    $default  reduce using rule 2 (list)

    s     go to state 1
    list  go to state 2


state 1

    0 $accept: s . $end

    $end  shift, and go to state 3


state 2

    1 s: list .
    3 list: list . '\n'
    4     | list . e

    NUMBER  shift, and go to state 4
    '\n'    shift, and go to state 5

    $default  reduce using rule 1 (s)
```

```
    e  go to state 6


state 3

    0 $accept: s $end .

    $default  accept


state 4

    5 e: NUMBER .

    $default  reduce using rule 5 (e)


state 5

    3 list: list '\n' .

    $default  reduce using rule 3 (list)


state 6

    4 list: list e .
    6 e: e . '&' e
    7  | e . '@' e

    '@'  shift, and go to state 7
    '&'  shift, and go to state 8

    $default  reduce using rule 4 (list)


state 7

    7 e: e '@' . e

    NUMBER  shift, and go to state 4

    e  go to state 9


state 8

    6 e: e '&' . e

    NUMBER  shift, and go to state 4

    e  go to state 10
```

```
state 9

    6 e: e . '&' e
    7  | e . '@' e
    7  | e '@' e .

    '&'  shift, and go to state 8

    $default  reduce using rule 7 (e)


state 10

    6 e: e . '&' e
    6  | e '&' e .
    7  | e . '@' e

    '&'  shift, and go to state 8

    $default  reduce using rule 6 (e)
```

La presencia de conflictos, aunque no siempre, en muchos casos es debida a la introducción de ambiguedad en la gramática. Si el conflicto es de desplazamiento-reducción se puede resolver explicitando alguna regla que rompa la ambiguedad. Los conflictos de reducción-reducción suelen producirse por un diseño erróneo de la gramática. En tales casos, suele ser mas adecuado modificar la gramática.

7.14. Esquemas de Traducción

Un *esquema de traducción* es una gramática independiente del contexto en la cuál se han asociado atributos a los símbolos de la gramática. Un atributo queda caracterizado por un identificador o nombre y un tipo o clase. Además se han insertado acciones, esto es, código JavaScript/Perl/Python/C, ... en medio de las partes derechas. En ese código es posible referenciar los atributos de los símbolos de la gramática como variables del lenguaje subyacente.

Recuerde que el orden en que se evalúan los fragmentos de código es el de un recorrido primero-profundo del árbol de análisis sintáctico. Mas específicamente, considerando a las acciones como hijos-hoja del nodo, el recorrido que realiza un esquema de traducción es:

```
function esquema_de_traduccion(node) {

  for(c in node.children) { # de izquierda a derecha
    child = node.children[i];
    if (child.instanceof('SemanticAction') { # si es una acción semántica
      child.execute;
    }
    else { esquema_de_traduccion(child) }
  }
}
```

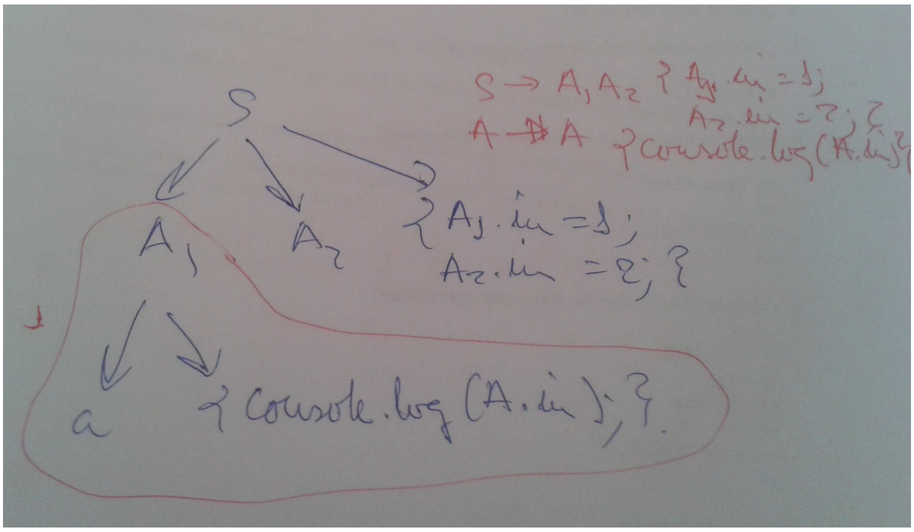
Obsérvese que, como el bucle recorre a los hijos de izquierda a derecha, se debe dar la siguiente condición para que un esquema de traducción funcione:

Para cualquier regla de producción aumentada con acciones, de la forma

$$A \to X_1 \dots X_j \texttt{\{ action(A\{b\}, X}_1\texttt{\{c\}... X}_n\texttt{\{d\})\}} X_{j+1} \dots X_n$$

debe ocurrir que los atributos evaluados en la acción insertada después de X_j dependan de atributos y variables que fueron computadas durante la visita de los hermanos izquierdos o de sus ancestros. En particular no deberían depender de atributos asociados con las variables $X_{j+1} \ldots X_n$. Ello no significa que no sea correcto evaluar atributos de $X_{j+1} \ldots X_n$ en esa acción.

Por ejemplo, el siguiente esquema no satisface el requisito:



porque cuando vas a ejecutar la acción `{ console.log(A.in) }` el atributo `A.in` no ha sido computado.

Los atributos de cada símbolo de la gramática $X \in V \cup \Sigma$ se dividen en dos grupos disjuntos: *atributos sintetizados* y *atributos heredados*:

- Un atributo de X es un *atributo heredado* si depende de atributos de su padre y hermános en el árbol.
- Un *atributo sintetizado* es aquél tal que el valor del atributo depende de los valores de los atributos de los hijos, es decir en tal caso X ha de ser una variable sintáctica y los atributos en la parte derecha de la regla semántica deben ser atributos de símbolos en la parte derecha de la regla de producción asociada.

7.15. Manejo en `jison` de Atributos Heredados

Supongamos que `jison` esta inmerso en la construcción de la antiderivación a derechas y que la forma sentencial derecha en ese momento es:

$$X_m \ldots X_1 X_0 Y_1 \ldots Y_n a_1 \ldots a_0$$

y que el mango es $B \to Y_1 \ldots Y_n$ y en la entrada quedan por procesar $a_1 \ldots a_0$.

No es posible acceder en `jison` a los valores de los atributos de los estados en la pila del analizador que se encuentran "por debajo" o si se quiere "a la izquierda" de los estados asociados con la regla por la que se reduce.

Vamos a usar un pequeño hack para acceder a los atributos asociados con símbolos vistos en el pasado remoto":

```
[~/jison/jison-inherited(grammar)]$ cat inherited.jison
%lex
%%
```

```
\s+                         {}
(global|local|integer|float)  { return yytext; }
[a-zA-Z_]\w*                { return 'id'; }
.                           { return yytext; }
/lex
%%
D
  : C T L
  ;

C
  : global
  | local
  ;

T
  : integer
  | float
;

L
  : L ',' id    {
                  console.log("L -> L ',' id ("+yytext+")");
                  var s =  eval('$$');
                  console.log(s);
                }
  | id          {
                  console.log("L -> id ("+yytext+")");
                  var s =  eval('$$');
                  console.log(s);
                }
  ;
%%
```

Veamos un ejemplo de ejecución:

```
[~/jison/jison-inherited(grammar)]$ cat input.txt
global integer a, b, c
[~/jison/jison-inherited(grammar)]$ node inherited.js input.txt
L -> id (a)
[ null, 'global', 'integer', 'a' ]
L -> L ',' id (b)
[ null, 'global', 'integer', 'a', ',', 'b' ]
L -> L ',' id (c)
[ null, 'global', 'integer', 'a', ',', 'c' ]
```

Esta forma de acceder a los atributos es especialmente útil cuando se trabaja con *atributos heredados*. Esto es, cuando un atributo de un nodo del árbol sintáctico se computa en términos de valores de atributos de su padre y/o sus hermanos. Ejemplos de atributos heredados son la clase y tipo en la declaración de variables.

Es importante darse cuenta que en cualquier derivación a derechas desde`D`, cuando se reduce por una de las reglas

$$\mathrm{L} \rightarrow \mathrm{id} \mid \mathrm{L}_1 \text{ ','} \mathrm{ id}$$

el símbolo a la izquierda de L es T y el que esta a la izquierda de T es C. Considere, por ejemplo la derivación a derechas:

$$D \Longrightarrow C\ T\ L \Longrightarrow C\ T\ L,\ id \Longrightarrow C\ T\ L,\ id,\ id \Longrightarrow C\ T\ id,\ id,\ id \Longrightarrow$$
$$\Longrightarrow C\ \text{float}\ id,\ id,\ id \Longrightarrow \text{local float}\ id,\ id,\ id$$

Observe que el orden de recorrido de `jison` es:

$$\text{local float}\ id,\ id,\ id \Longleftarrow C\ \text{float}\ id,\ id \Longleftarrow C\ T\ id,\ id,\ id \Longleftarrow$$
$$\Longleftarrow C\ T\ L,\ id,\ id \Longleftarrow C\ T\ L,\ id \Longleftarrow C\ T\ L \Longleftarrow D$$

en la antiderivación, cuando el mango es una de las dos reglas para listas de identificadores, $L \rightarrow id$ y $L \rightarrow L,\ id$ es decir durante las tres ultimas antiderivaciones:

$$C\ T\ L,\ id,\ id \Longleftarrow C\ T\ L,\ id \Longleftarrow C\ T\ L \Longleftarrow D$$

las variables a la izquierda del mango son T y C. Esto ocurre siempre. Estas observaciones nos conducen al siguiente programa `jison`:

```
[~/jison/jison-inherited(deepstack)]$ cat inherited.jison
%lex
%%
\s+                           {}
(global|local|integer|float)  { return yytext; }
[a-zA-Z_]\w*                  { return 'id'; }
.                             { return yytext; }
/lex
%%
D
  : C T L
  ;

C
  : global
  | local
  ;

T
  : integer
  | float
  ;

L
  : L ',' id    {
                  var s =  eval('$$');
                  var b0 = s.length - 3;

                  console.log("L -> L ',' id ("+yytext+")");
                  console.log($id + ' is of type ' + s[b0-1]);
                  console.log(s[b0] + ' is of class ' + s[b0-2]);
                }
  | id          {
                  var s =  eval('$$');
                  var b0 = s.length - 1;
```

```
                console.log("L -> id ("+yytext+")");
                console.log($id + ' is of type ' + s[b0-1]);
                console.log(s[b0] + ' is of class ' + s[b0-2]);
              }
  ;
%%
```

A continuación sigue un ejemplo de ejecución:

```
[~/jison/jison-inherited(deepstack)]$ node inherited.js input.txt
L -> id (a)
a is of type integer
a is of class global
L -> L ',' id (b)
b is of type integer
a is of class global
L -> L ',' id (c)
c is of type integer
a is of class global
```

En este caso, existen varias alternativas simples a esta solución:

- Montar la lista de identificadores en un array y ponerles el tipo y la clase de "golpe.[en] la regla de producción superior `D : C T L ;`
- usar variables visibles (globales o atributos del objeto parser, por ejemplo) como `current_type`, `current_class`

  ```
  C
    : global { current_class = 'global'; }
    | local  { current_class = 'local'; }
  ```

  y depués accederlas en las reglas de `L`
- La que posiblemente sea la mejor estrategia general: construir el árbol de análisis sintáctico. Posteriormente podemos recorrer el árbol como queramos y tantas veces como queramos.

7.16. Definición Dirigida por la Sintáxis

Una *definición dirigida por la sintáxis* es un pariente cercano de los esquemas de traducción. En una definición dirigida por la sintáxis una gramática $G = (V, \Sigma, P, S)$ se aumenta con nuevas características:

- A cada símbolo $S \in V \cup \Sigma$ de la gramática se le asocian cero o mas atributos. Un atributo queda caracterizado por un identificador o nombre y un tipo o clase. A este nivel son *atributos formales*, como los parámetros formales, en el sentido de que su realización se produce cuando el nodo del árbol es creado.
- A cada regla de producción $A \to X_1 X_2 \ldots X_n \in P$ se le asocian un conjunto de *reglas de evaluación de los atributos* o *reglas semánticas* que indican que el atributo en la parte izquierda de la regla semántica depende de los atributos que aparecen en la parte derecha de la regla. El atributo que aparece en la parte izquierda de la regla semántica puede estar asociado con un símbolo en la parte derecha de la regla de producción.

- Los atributos de cada símbolo de la gramática $X \in V \cup \Sigma$ se dividen en dos grupos disjuntos: *atributos sintetizados* y *atributos heredados*. Un atributo de X es un *atributo heredado* si depende de atributos de su padre y hermános en el árbol. Un *atributo sintetizado* es aquél tal que el valor del atributo depende de los valores de los atributos de los hijos, es decir en tal caso X ha de ser una variable sintáctica y los atributos en la parte derecha de la regla semántica deben ser atributos de símbolos en la parte derecha de la regla de producción asociada.

- Los atributos predefinidos se denominán *atributos intrínsecos*. Ejemplos de atributos intrínsecos son los atributos sintetizados de los terminales, los cuáles se han computado durante la fase de análisis léxico. También son atributos intrínsecos los atributos heredados del símbolo de arranque, los cuales son pasados como parámetros al comienzo de la computación.

La diferencia principal con un esquema de traducción está en que no se especifica el orden de ejecución de las reglas semánticas. Se asume que, bien de forma manual o automática, se resolverán las dependencias existentes entre los atributos determinadas por la aplicación de las reglas semánticas, de manera que serán evaluados primero aquellos atributos que no dependen de ningún otro, despues los que dependen de estos, etc. siguiendo un esquema de ejecución que viene guiado por las dependencias existentes entre los datos.

Aunque hay muchas formas de realizar un evaluador de una definición dirigida por la sintáxis, conceptualmente, tal evaluador debe:

1. Construir el árbol de análisis sintáctico para la gramática y la entrada dadas.

2. Analizar las reglas semánticas para determinar los atributos, su clase y las dependencias entre los mismos.

3. Construir el *grafo de dependencias* de los atributos, el cual tiene un nodo para cada ocurrencia de un atributo en el árbol de análisis sintáctico etiquetado con dicho atributo. El grafo tiene una arista entre dos nodos si existe una dependencia entre los dos atributos a través de alguna regla semántica.

4. Supuesto que el grafo de dependencias determina un *orden parcial* (esto es cumple las propiedades reflexiva, antisimétrica y transitiva) construir un *orden topológico* compatible con el orden parcial.

5. Evaluar las reglas semánticas de acuerdo con el orden topológico.

Una definición dirigida por la sintáxis en la que las reglas semánticas no tienen efectos laterales se denomina una *gramática atribuída*.

Si la definición dirigida por la sintáxis puede ser realizada mediante un esquema de traducción se dice que es *L-atribuída*. Para que una definición dirigida por la sintáxis sea L-atribuída deben cumplirse que cualquiera que sea la regla de producción $A \to X_1 \dots X_n$, los atributos heredados de X_j pueden depender únicamente de:

1. Los atributos de los símbolos a la izquierda de X_j

2. Los atributos heredados de A

Nótese que las restricciones se refieren a los atributos heredados. El cálculo de los atributos sintetizados no supone problema para un esquema de traducción. Si la gramática es LL(1), resulta fácil realizar una definición L-atribuída en un analizador descendente recursivo predictivo.

Si la definición dirigida por la sintáxis sólo utiliza atributos sintetizados se denomina *S-atribuída*. Una definición S-atribuída puede ser fácilmente trasladada a un programa `jison`.

Ejercicio 7.16.1. *El siguiente es un ejemplo de definición dirigida por la sintáxis:*

$S \to a\ A\ C$	`$C{i} = $A{s}`
$S \to b\ A\ B\ C$	`$C{i} = $A{s}`
$C \to c$	`$C{s} = $C{i}`
$A \to a$	`$A{s} = "a"`
$B \to b$	`$B{s} = "b"`

Determine un orden correcto de evaluación de la anterior definición dirigida por la sintáxis para la entrada `b a b c`*.*

Ejercicio 7.16.2.

Lea el artículo Are Attribute Grammars used in Industry? por Josef Grosch

> *I am observing a lack of education and knowledge about compiler construction in industry. When I am asking the participants of our trainings or the employees we met in our projects then only few people have learned about compiler construction during their education. For many of them compiler construction has a bad reputation because of what and how they have learned about this topic. Even fewer people have a usable knowledge about compilers. Even fewer people know about the theory of attribute grammars. And even fewer people know how to use attribute grammars for solving practical problems. Nevertheless, attribute grammars are used in industry. However, in many cases the people in industry do not know about this fact. They are running prefabricated subsystems constructed by external companies such as ours. These subsystems are for example parsers which use attribute grammar technology.*

7.17. Ejercicios: Casos de Estudio

Véase nuestro proyecto Grammar Repository en GoogleCode.

7.17.1. Un mal diseño

Ejercicio 7.17.1. *This grammar*

```
%token D S

%%
p: ds ';' ss  | ss ;
ds: D ';' ds
  | D
;
ss: S ';' ss  | S  ;
%%
```

illustrates a typical LALR conflict due to a bad grammar design.

- *Reescriba la gramática para que no existan conflictos*
- *Escriba las acciones semánticas necesarias para imprimir la lista de* `D`*s seguida de la lista de* `S`*s*

Donde

- ```
  [~/jison/jison-decs-sts(master)]$ pwd -P
  /Users/casiano/local/src/javascript/PLgrado/jison/jison-decs-sts
  ```
- ```
  [~/jison/jison-decs-sts(master)]$ git remote -v
  origin  git@github.com:crguezl/jison-decs-sts.git (fetch)
  origin  git@github.com:crguezl/jison-decs-sts.git (push)
  ```
- https://github.com/crguezl/jison-decs-sts

7.17.2. Gramática no LR(1)

La siguiente gramática no es LR(1).

```
[~/srcPLgrado/jison/jison-nolr]$ cat confusingsolvedppcr.y
%%
A:
    B 'c' 'd'
  | E 'c' 'f'
;
B:
    'x' 'y'
;
E:
    'x' 'y'
;

%%
```

Encuentre una gramática sin conflictos equivalente a la anterior.

Donde

- `[~/jison/jison-nolr(master)]$ pwd -P`
  `/Users/casiano/local/src/javascript/PLgrado/jison/jison-nolr`
- `[~/jison/jison-nolr(master)]$ git remote -v`
  `origin  git@github.com:crguezl/jison-nolr.git (fetch)`
  `origin  git@github.com:crguezl/jison-nolr.git (push)`
- https://github.com/crguezl/jison-nolr

7.17.3. Un Lenguaje Intrínsecamente Ambiguo

Ejercicio 7.17.2. *A context-free language is inherently ambiguous if all context-free grammars generating that language are ambiguous. While some context-free languages have both ambiguous and unambiguous grammars, there are context-free languages for which no unambiguous context-free grammar can exist. An example of an inherently ambiguous language is the set*

`{ a^n b^n c^m : n >= 0, m >= 0 } U { a^n b^m c^m : n >= 0, m >= 0 }`

Esto es: Concatenaciones de repeticiones de `a`*s seguidas de repeticiones de* `b`*s y seguidas de repeticiones de* `c`*s donde el número de* `a`*s es igual al número de* `b`*s o bien el número de* `b`*s es igual al número de* `c`*s.*

- *Escriba una gramática que genere dicho lenguaje*

  ```
  s: aeqb | beqc ;
  aeqb: ab cs ;
  ab: /* empty */ | 'a' ab 'b' ;
  cs: /* empty */ | cs 'c' ;
  beqc: as bc ;
  bc: /* empty */ | 'b' bc 'c' ;
  as: /* empty */ | as 'a' ;
  ```

  The symbol `aeqb` *correspond to guess that there are the same number of* `a`*s than* `b`*s. In the same way,* `beqc` *starts the subgrammar for those phrases where the number of* `b`*s is equal to the number of* `c`*s. The usual approach to eliminate the ambiguity by changing the grammar to an unambiguous one does not work.*

- *Escriba un programa Jison que reconozca este lenguaje.*

Donde

- `[~/jison/jison-ambiguouslanguage(master)]$ pwd -P`
  `/Users/casiano/local/src/javascript/PLgrado/jison/jison-ambiguouslanguage`
- ```
  [~/jison/jison-ambiguouslanguage(master)]$ git remote -v
  origin  git@github.com:crguezl/jison-ambiguouslanguage.git (fetch)
  origin  git@github.com:crguezl/jison-ambiguouslanguage.git (push)
  ```
- https://github.com/crguezl/jison-ambiguouslanguage

7.17.4. Conflicto reduce-reduce

La siguiente gramática presenta conflictos reduce-reduce:

Ejercicio 7.17.3. `[~/srcPLgrado/jison/jison-reducereduceconflict]$ cat reducereduceconflictPPCR2`

```
%token ID

%%

def:    param_spec return_spec ','
        ;
param_spec:
             type
        |    name_list ':' type
        ;
return_spec:
             type
        |    name ':' type
        ;
type:
             ID
        ;
name:
             ID
        ;
name_list:
             name
        |    name ',' name_list
        ;
%%
```

Este es el diagnóstico de Jison:

```
~/srcPLgrado/jison/jison-reducereduceconflict]$ jison reducereduceconflictPPCR2.y
Conflict in grammar: multiple actions possible when lookahead token is ID in state 5
- reduce by rule: name -> ID
- reduce by rule: type -> ID
Conflict in grammar: multiple actions possible when lookahead token is : in state 5
- reduce by rule: name -> ID
- reduce by rule: type -> ID
Conflict in grammar: multiple actions possible when lookahead token is , in state 5
- reduce by rule: name -> ID
- reduce by rule: type -> ID
```
```

```
States with conflicts:
State 5
  type -> ID . #lookaheads= ID : ,
  name -> ID . #lookaheads= ID : ,
```

Este es el resultado de compilar con `bison -v`*:*

```
[~/jison/jison-reducereduceconflict(master)]$ bison -v reducereduceconflictPPCR2.y
reducereduceconflictPPCR2.y: conflicts: 1 reduce/reduce
```

El fichero `reducereduceconflictPPCR2.output` *queda así:*

```
[~/jison/jison-reducereduceconflict(master)]$ cat reducereduceconflictPPCR2.output
State 1 conflicts: 1 reduce/reduce


Grammar

    0 $accept: def $end

    1 def: param_spec return_spec ','

    2 param_spec: type
    3           | name_list ':' type

    4 return_spec: type
    5            | name ':' type

    6 type: ID

    7 name: ID

    8 name_list: name
    9          | name ',' name_list


Terminals, with rules where they appear

$end (0) 0
',' (44) 1 9
':' (58) 3 5
error (256)
ID (258) 6 7


Nonterminals, with rules where they appear

$accept (6)
    on left: 0
def (7)
    on left: 1, on right: 0
param_spec (8)
    on left: 2 3, on right: 1
return_spec (9)
```

```
    on left: 4 5, on right: 1
type (10)
    on left: 6, on right: 2 3 4 5
name (11)
    on left: 7, on right: 5 8 9
name_list (12)
    on left: 8 9, on right: 3 9


state 0

    0 $accept: . def $end

    ID  shift, and go to state 1

    def         go to state 2
    param_spec  go to state 3
    type        go to state 4
    name        go to state 5
    name_list   go to state 6


state 1

    6 type: ID .
    7 name: ID .

    ','       reduce using rule 6 (type)
    ','       [reduce using rule 7 (name)]
    ':'       reduce using rule 7 (name)
    $default  reduce using rule 6 (type)


state 2

    0 $accept: def . $end

    $end  shift, and go to state 7


state 3

    1 def: param_spec . return_spec ','

    ID  shift, and go to state 1

    return_spec  go to state 8
    type         go to state 9
    name         go to state 10


state 4
```

```
    2 param_spec: type .

    $default  reduce using rule 2 (param_spec)


state 5

    8 name_list: name .
    9          | name . ',' name_list

    ','  shift, and go to state 11

    $default  reduce using rule 8 (name_list)


state 6

    3 param_spec: name_list . ':' type

    ':'  shift, and go to state 12


state 7

    0 $accept: def $end .

    $default  accept


state 8

    1 def: param_spec return_spec . ','

    ','  shift, and go to state 13


state 9

    4 return_spec: type .

    $default  reduce using rule 4 (return_spec)


state 10

    5 return_spec: name . ':' type

    ':'  shift, and go to state 14


state 11

    9 name_list: name ',' . name_list
```

```
    ID  shift, and go to state 15

    name       go to state 5
    name_list  go to state 16


state 12

    3 param_spec: name_list ':' . type

    ID  shift, and go to state 17

    type  go to state 18


state 13

    1 def: param_spec return_spec ',' .

    $default  reduce using rule 1 (def)


state 14

    5 return_spec: name ':' . type

    ID  shift, and go to state 17

    type  go to state 19


state 15

    7 name: ID .

    $default  reduce using rule 7 (name)


state 16

    9 name_list: name ',' name_list .

    $default  reduce using rule 9 (name_list)


state 17

    6 type: ID .

    $default  reduce using rule 6 (type)
```

```
state 18

    3 param_spec: name_list ':' type .

    $default  reduce using rule 3 (param_spec)


state 19

    5 return_spec: name ':' type .

    $default  reduce using rule 5 (return_spec)
```

Encuentre una gramática equivalente a la anterior sin conflictos.

Solución When the problem arises, you can often fix it by identifying the two parser states that are being confused, and adding something to make them look distinct. In the above example, adding one rule to `return_spec` as follows makes the problem go away:

```
%token BOGUS
...
%%
...
return_spec:
             type
        |    name ':' type
        /* This rule is never used.  */
        |    ID BOGUS
        ;
```

This corrects the problem because it introduces the possibility of an additional active rule in the context after the ID at the beginning of `return_spec`. This rule is not active in the corresponding context in a `param_spec`, so the two contexts receive distinct parser states. As long as the token BOGUS is never generated by yylex, the added rule cannot alter the way actual input is parsed.

In this particular example, there is another way to solve the problem: rewrite the rule for `return_spec` to use ID directly instead of via name. This also causes the two confusing contexts to have different sets of active rules, because the one for `return_spec` activates the altered rule for `return_spec` rather than the one for name.

```
param_spec:
             type
        |    name_list ':' type
        ;
return_spec:
             type
        |    ID ':' type
        ;
```

Donde

- ```
  [~/jison/jison-reducereduceconflict(master)]$ pwd -P
  /Users/casiano/local/src/javascript/PLgrado/jison/jison-reducereduceconflict
  ```
- ```
  [~/jison/jison-reducereduceconflict(master)]$ git remote -v
  origin  git@github.com:crguezl/ull-etsii-grado-PL-reducereduce.git (fetch)
  origin  git@github.com:crguezl/ull-etsii-grado-PL-reducereduce.git (push)
  ```

- https://github.com/crguezl/ull-etsii-grado-PL-reducereduce

Véase Véase Mysterious Reduce/Reduce Conflicts

7.18. Recuperación de Errores

La recuperación de errores no parece estar implementada en Jison. véase

- la sección Error Recovery de la documentación
- Pullreq 5 - parser built-in grammar error recovery was completely broken

```
[~/srcPLgrado/jison/jison-aSb(error)]$ cat aSb.jison
%lex
%%
\s+              {}
[ab]             { return yytext; }
.                { return "INVALID"; }
/lex
%%
S: /* empty */  { $$ = ''; console.log("empty"); }
   | 'a' S 'b'  { $$ = $1 + $2 + $3; console.log("S -> aSb"); }
   | 'a' S error
;
%%
```

```
        parse: function parse(input) {
              var self = this,
                  stack = [0],
                  vstack = [null], // semantic value stack
                  lstack = [], // location stack
                  ...
                  recovering = 0,
                  TERROR = 2,
                  EOF = 1;
            while (true) {
                  // retreive state number from top of stack
                  state = stack[stack.length - 1];

                  ...

                  // handle parse error
>>                _handle_error: if (typeof action === 'undefined' || !action.length || !  ...

                      var errStr = '';
                      if (!recovering) {
```

7.19. Depuración en `jison`

7.20. Construcción del Árbol Sintáctico

El siguiente ejemplo usa `jison` para construir el árbol sintáctico de una expresión en infijo:

7.21. Consejos a seguir al escribir un programa `jison`

Cuando escriba un programa `jison` asegurese de seguir los siguientes consejos:

1. Coloque el punto y coma de separación de reglas en una línea aparte. Un punto y coma "pegado" al final de una regla puede confundirse con un terminal de la regla.

2. Si hay una regla que produce vacío, coloquela en primer lugar y acompáñela de un comentario resaltando ese hecho.

3. Nunca escriba dos reglas de producción en la misma línea.

4. Sangre convenientemente todas las partes derechas de las reglas de producción de una variable, de modo que queden alineadas.

5. Ponga nombres representativos a sus variables sintácticas. No llame `Z` a una variable que representa el concepto "lista de parámetros", llámela `ListaDeParametros`.

6. Es conveniente que declare los terminales simbólicos, esto es, aquellos que llevan un identificador asociado. Si no llevan prioridad asociada o no es necesaria, use una declaración `%token`. De esta manera el lector de su programa se dará cuenta rápidamente que dichos identificadores no se corresponden con variables sintácticas. Por la misma razón, si se trata de terminales asociados con caracteres o cadenas no es tan necesario que los declare, a menos que, como en el ejemplo de la calculadora para `'+'` y `'*'`, sea necesario asociarles una precedencia.

7. Es importante que use la opción `-v` para producir el fichero `.output` conteniendo información detallada sobre los conflictos y el autómata. Cuando haya un conflicto shift-reduce no resuelto busque en el fichero el estado implicado y vea que LR(0) items $A \to \alpha_\uparrow$ y $B \to \beta_\uparrow\gamma$ entran en conflicto.

8. Si según el informe de `jison` el conflicto se produce ante un terminal a, es porque $a \in FOLLOW(A)$ y $a \in FIRST(\gamma)$. Busque las causas por las que esto ocurre y modifique su gramática con vistas a eliminar la presencia del terminal a en uno de los dos conjuntos implicados o bien establezca reglas de prioridad entre los terminales implicados que resuelvan el conflicto.

9. Nótese que cuando existe un conflicto de desplazamiento reducción entre $A \to \alpha_\uparrow$ y $B \to \beta_\uparrow\gamma$, el programa `jison` contabiliza un error por cada terminal $a \in FOLLOW(A) \cap FIRST(\gamma)$. Por esta razón, si hay 16 elementos en $FOLLOW(A) \cap FIRST(\gamma)$, el analizador `jison` informará de la existencia de 16 conflictos *shift-reduce*, cuando en realidad se trata de uno sólo. No desespere, los conflictos "auténticos" suelen ser menos de los que `jison` anuncia.

10. Si necesita declarar variables globales, inicializaciones, etc. que afectan la conducta global del analizador, escriba el código correspondiente en la cabecera del analizador, protegido por los delimitadores `%{` y `%}`. Estos delimitadores deberán aparecer en una línea aparte. Por ejemplo:

```
%{
our contador = 0;
%}

%token NUM
...
%%
```

Capítulo 8

Análisis Sintáctico Ascendente en Ruby

8.1. La Calculadora

8.1.1. Uso desde Línea de Comandos

```
[~/src/PL/rexical/sample(master)]$ racc --help
Usage: racc [options] <input>
    -o, --output-file=PATH           output file name [<input>.tab.rb]
    -t, --debug                      Outputs debugging parser.
    -g                               Equivalent to -t (obsolete).
    -v, --verbose                    Creates <filename>.output log file.
    -O, --log-file=PATH              Log file name [<input>.output]
    -e, --executable [RUBYPATH]      Makes executable parser.
    -E, --embedded                   Embeds Racc runtime in output.
        --line-convert-all           Converts line numbers of user codes.
    -l, --no-line-convert            Never convert line numbers.
    -a, --no-omit-actions            Never omit actions.
        --superclass=CLASSNAME       Uses CLASSNAME instead of Racc::Parser.
        --runtime=FEATURE            Uses FEATURE instead of 'racc/parser'
    -C, --check-only                 Checks syntax and quit immediately.
    -S, --output-status              Outputs internal status time to time.
    -P                               Enables generator profile
    -D flags                         Flags for Racc debugging (do not use).
        --version                    Prints version and quit.
        --runtime-version            Prints runtime version and quit.
        --copyright                  Prints copyright and quit.
        --help                       Prints this message and quit.
```

```
[~/Dropbox/src/PL/rexical/sample(master)]$ cat -n Rakefile
     1  task :default => %W{racc rex} do
     2    sh "ruby calc3.tab.rb"
     3  end
     4
     5  task :racc do
     6    sh "racc calc3.racc"
     7  end
     8
     9  task :rex do
    10    sh "rex calc3.rex"
    11  end
```

8.1.2. Análisis Léxico con rexical

```
[~/Dropbox/src/PL/rexical/sample(master)]$ cat -n calc3.rex
#
# calc3.rex
# lexical scanner definition for rex
#

class Calculator3
macro
  BLANK         \s+
  DIGIT         \d+
rule
  {BLANK}
  {DIGIT}       { [:NUMBER, text.to_i] }
  .|\n          { [text, text] }
inner
end
```

8.1.3. Análisis Sintáctico

```
[~/Dropbox/src/PL/rexical/sample(master)]$ cat -n calc3.racc
#
# A simple calculator, version 3.
#

class Calculator3
  prechigh
    nonassoc UMINUS
    left '*' '/'
    left '+' '-'
  preclow
  options no_result_var
rule
  target  : exp
          | /* none */ { 0 }

  exp     : exp '+' exp { val[0] + val[2] }
          | exp '-' exp { val[0] - val[2] }
          | exp '*' exp { val[0] * val[2] }
          | exp '/' exp { val[0] / val[2] }
          | '(' exp ')' { val[1] }
          | '-' NUMBER  =UMINUS { -(val[1]) }
          | NUMBER
end

---- header ----
#
# generated by racc
#
require 'calc3.rex'

---- inner ----

```

```
---- footer ----

puts 'sample calc'
puts '"q" to quit.'
calc = Calculator3.new
while true
  print '>>> '; $stdout.flush
  str = $stdin.gets.strip
  break if /q/i === str
  begin
    p calc.scan_str(str)
  rescue ParseError
    puts 'parse error'
  end
end
```

Precedencias `right` is yacc's `%right`, `left` is yacc's `%left`.
`= SYMBOL` means yacc's `%prec SYMBOL`:

```
prechigh
  nonassoc '++'
  left     '*' '/'
  left     '+' '-'
  right    '='
preclow

rule
  exp: exp '*' exp
     | exp '-' exp
     | '-' exp       =UMINUS   # equals to "%prec UMINUS"
         :
         :
```

Atributos You can use following special local variables in action.

1. `result ($$)`

   The value of left-hand side (lhs). A default value is `val[0]`.

2. `val ($1,$2,$3...)`

   An array of value of right-hand side (rhs).

3. `_values (...$-2,$-1,$0)`

   A stack of values. DO NOT MODIFY this stack unless you know what you are doing.

Declaring Tokens By declaring tokens, you can avoid bugs.

```
token NUM ID IF
```

Opciones
You can write options for racc command in your racc file.

```
options OPTION OPTION ...
```

Options are:

1. `omit_action_call`

   omit empty action call or not.

2. `result_var`

   use/does not use local variable result”

You can use `no_` prefix to invert its meanings.

User Code Block

Ü̈ser Code Blocks̈ a Ruby source code which is copied to output. There are three user code block, ”headerinner.ªnd ”footer”.

Format of user code is like this:

```
---- header
  ruby statement
  ruby statement
  ruby statement

---- inner
  ruby statement
      :
      :
```

If four `-` exist on line head, racc treat it as beginning of user code block. A name of user code must be one word.

8.2. Véase También

- Racc en GitHub
-
- Racc User’s Manual
- Martin Fowler Hello Racc
- Rexical en GitHub

Capítulo 9

Transformaciones Árbol

9.1. Árbol de Análisis Abstracto

Un *árbol de análisis abstracto* (denotado AAA, en inglés *abstract syntax tree* o AST) porta la misma información que el árbol de análisis sintáctico pero de forma mas condensada, eliminándose terminales y producciones que no aportan información.

Alfabeto con Aridad o Alfabeto Árbol

Definición 9.1.1. *Un* alfabeto con función de aridad *es un par* (Σ, ρ) *donde* Σ *es un conjunto finito y* ρ *es una función* $\rho : \Sigma \to \mathbb{N}_0$, *denominada* función de aridad. *Denotamos por* $\Sigma_k = \{a \in \Sigma : \ \rho(a) = k\}$.

Lenguaje de los Arboles *Definimos el* lenguaje árbol homogéneo $B(\Sigma)$ *sobre* Σ *inductivamente:*

- *Todos los elementos de aridad 0 están en* $B(\Sigma)$*:* $a \in \Sigma_0$ *implica* $a \in B(\Sigma)$
- *Si* $b_1, \ldots, b_k \in B(\Sigma)$ *y* $f \in \Sigma_k$ *es un elemento k-ario, entonces* $f(b_1, \ldots, b_k) \in B(\Sigma)$

Los elementos de $B(\Sigma)$ *se llaman* árboles *o* términos.

Ejemplo 9.1.1. *Sea* $\Sigma = \{A, CONS, NIL\}$ *con* $\rho(A) = \rho(NIL) = 0, \ \rho(CONS) = 2$. *Entonces*
$B(\Sigma) = \{A, NIL, CONS(A, NIL), CONS(NIL, A), CONS(A, A), CONS(NIL, NIL), \ldots\}$

Ejemplo 9.1.2. *Una versión simplificada del alfabeto con aridad en el que estan basados los árboles construidos por el compilador de Tutu es:*
$\Sigma = \{ID, NUM, LEFTVALUE, STR, PLUS, TIMES, ASSIGN, PRINT\}$
$\rho(ID) = \rho(NUM) = \rho(LEFTVALUE) = \rho(STR) = 0$
$\rho(PRINT) = 1$
$\rho(PLUS) = \rho(TIMES) = \rho(ASSIGN) = 2$.
Observe que los elementos en $B(\Sigma)$ *no necesariamente son árboles "correctos". Por ejemplo, el árbol* $ASSIGN(NUM, PRINT(ID))$ *es un elemento de* $B(\Sigma)$.

Gramática Árbol

Definición 9.1.2. *Una* gramática árbol regular *es una cuadrupla* $((\Sigma, \rho), N, P, S)$*, donde:*

- (Σ, ρ) *es un alfabeto con aricidad* $\rho : \Sigma \to \mathbb{N}$
- N *es un conjunto finito de variables sintácticas o no terminales*
- P *es un conjunto finito de reglas de producción de la forma* $X \to s$ *con* $X \in N$ *y* $s \in B(\Sigma \cup N)$
- $S \in N$ *es la variable o símbolo de arranque*

Lenguaje Generado por una Gramática Árbol

Definición 9.1.3. *Dada una gramática* (Σ, N, P, S)*, se dice que un árbol* $t \in B(\Sigma \cup N)$ *es del tipo* $(X_1, \ldots X_k)$ *si el* j*-ésimo noterminal, contando desde la izquierda, que aparece en* t *es* $X_j \in N$*.*

Si $p = X \rightarrow s$ *es una producción y* s *es de tipo* $(X_1, \ldots X_n)$*, diremos que la producción* p *es de tipo* $(X_1, \ldots X_n) \rightarrow X$*.*

Definición 9.1.4. *Consideremos un árbol* $t \in B(\Sigma \cup N)$ *que sea del tipo* $(X_1, \ldots X_n)$*, esto es las variables sintácticas en el árbol leídas de izquierda a derecha son* $(X_1, \ldots X_n)$*.*

- *Si* $X_i \rightarrow s_i \in P$ *para algún* i*, entonces decimos que el árbol* t deriva en un paso en el árbol t' *resultante de sustituir el nodo* X_i *por el árbol* s_i *y escribiremos* $t \Longrightarrow t'$*. Esto es,* $t' = t\{X_i/s_i\}$
- *Todo árbol deriva en cero pasos en si mismo* $t \stackrel{0}{\Longrightarrow} t$*.*
- *Decimos que un árbol* t *deriva en* n *pasos en el árbol* t' *y escribimos* $t \stackrel{n}{\Longrightarrow} t'$ *si* t *deriva en un paso en un árbol* t'' *el cuál deriva en* $n-1$ *pasos en* t'*. En general, si* t *deriva en un cierto número de pasos en* t' *escribiremos* $t \stackrel{*}{\Longrightarrow} t'$*.*

Definición 9.1.5. *Se define el* lenguaje árbol generado por una gramática $G = (\Sigma, N, P, S)$ *como el lenguaje* $L(G) = \{t \in B(\Sigma) : \exists S \stackrel{*}{\Longrightarrow} t\}$*.*

Ejemplo 9.1.3. *Sea* $G = (\Sigma, V, P, S)$ *con* $\Sigma = \{A, CONS, NIL\}$ *y* $\rho(A) = \rho(NIL) = 0$*,* $\rho(CONS) = 2$ *y sea* $V = \{exp, list\}$*. El conjunto de producciones* P *es:*

$$P_1 = \{list \rightarrow NIL,\ list \rightarrow CONS(exp, list),\ exp \rightarrow A\}$$

La producción $list \rightarrow CONS(exp, list)$ *es del tipo* $(exp, list) \rightarrow list$*.*

El lenguaje generado por G *se obtiene realizando sustituciones sucesivas (derivando) desde el símbolo de arranque hasta producir un árbol cuyos nodos estén etiquetados con elementos de* Σ*. En este ejemplo,* $L(G)$ *es el conjunto de arboles de la forma:*

$$L(G) = \{NIL, CONS(A, NIL), CONS(A, CONS(A, NIL)), \ldots\}$$

Ejercicio 9.1.1. *Construya una derivación para el árbol* $CONS(A, CONS(A, NIL))$*. ¿De que tipo es el árbol* $CONS(exp, CONS(A, CONS(exp, L)))$*?.*

Cuando hablamos del AAA producido por un analizador sintáctico, estamos en realidad hablando de un lenguaje árbol cuya definición precisa debe hacerse a través de una gramática árbol regular. Mediante las gramáticas árbol regulares disponemos de un mecanismo para describir formalmente el lenguaje de los AAA que producirá el analizador sintáctico para las sentencias Tutu.

Ejemplo 9.1.4. *Sea* $G = (\Sigma, V, P, S)$ *con*

$$\begin{array}{l}
\Sigma = \{ID, NUM, LEFTVALUE, STR, PLUS, TIMES, ASSIGN, PRINT\} \\
\rho(ID) = \rho(NUM) = \rho(LEFTVALUE) = \rho(STR) = 0 \\
\rho(PRINT) = 1 \\
\rho(PLUS) = \rho(TIMES) = \rho(ASSIGN) = 2 \\
V = \{st, expr\}
\end{array}$$

y las producciones:

$$\begin{array}{lll}
P = & \{ & \\
 & st & \rightarrow ASSIGN(LEFTVALUE, expr) \\
 & st & \rightarrow PRINT(expr) \\
 & expr & \rightarrow PLUS(expr, expr) \\
 & expr & \rightarrow TIMES(expr, expr) \\
 & expr & \rightarrow NUM \\
 & expr & \rightarrow ID \\
 & expr & \rightarrow STR \\
 & \} &
\end{array}$$

Entonces el lenguaje $L(G)$ contiene árboles como el siguiente:

$$
\begin{array}{llll}
ASSIGN & (& & \\
 & LEFTVALUE, & & \\
 & PLUS & (& \\
 & & ID, & \\
 & & TIMES & (\\
 & & & NUM, \\
 & & & ID \\
 & & &) \\
 & &) & \\
 &) & &
\end{array}
$$

El cual podría corresponderse con una sentencia como `a = b + 4 * c`.

El lenguaje de árboles descrito por esta gramática árbol es el lenguaje de los AAA de las sentencias de Tutu.

Ejercicio 9.1.2. *Redefina el concepto de árbol de análisis concreto dado en la definición 7.1.7 utilizando el concepto de gramática árbol. Con mas precisión, dada una gramática $G = (\Sigma, V, P, S)$ defina una gramática árbol $T = (\Omega, N, R, U)$ tal que $L(T)$ sea el lenguaje de los árboles concretos de G. Puesto que las partes derechas de las reglas de producción de P pueden ser de distinta longitud, existe un problema con la aricidad de los elementos de Ω. Discuta posibles soluciones.*

Ejercicio 9.1.3. *¿Cómo son los árboles sintácticos en las derivaciones árbol? Dibuje varios árboles sintácticos para las gramáticas introducidas en los ejemplos* **??** *y* **??**.

Intente dar una definición formal del concepto de árbol de análisis sintáctico asociado con una derivación en una gramática árbol

Notación de Dewey o Coordenadas de un Árbol

Definición 9.1.6. *La notación de Dewey es una forma de especificar los subárboles de un árbol $t \in B(\Sigma)$. La notación sigue el mismo esquema que la numeración de secciones en un texto: es una palabra formada por números separados por puntos. Así t/2.1.3 denota al tercer hijo del primer hijo del segundo hijo del árbol t. La definición formal sería:*

- $t/\epsilon = t$
- *Si $t = a(t_1, \ldots t_k)$ y $j \in \{1 \ldots k\}$ y n es una cadena de números y puntos, se define inductivamente el subárbol $t/j.n$ como el subárbol n-ésimo del j-ésimo subárbol de t. Esto es: $t/j.n = t_j/n$*

Ejercicio 9.1.4. *Sea el árbol:*

$$
\begin{array}{llll}
t = ASSIGN & (& & \\
 & LEFTVALUE, & & \\
 & PLUS & (& \\
 & & ID, & \\
 & & TIMES & (\\
 & & & NUM, \\
 & & & ID \\
 & & &) \\
 & &) & \\
 &) & &
\end{array}
$$

Calcule los subárboles t/ϵ, $t/2.2.1$, $t/2.1$ y $t/2.1.2$.

9.2. Selección de Código y Gramáticas Árbol

La generación de código es la fase en la que a partir de la *Representación intermedia* o *IR* se genera una secuencia de instrucciones para la máquina objeto. Esta tarea conlleva diversas subtareas, entre ellas destacan tres:

- La selección de instrucciones o selección de código,
- La asignación de registros y
- La planificación de las instrucciones.

El problema de la *selección de código* surge de que la mayoría de las máquinas suelen tener una gran variedad de instrucciones, habitualmente cientos y muchas instrucciones admiten mas de una decena de modos de direccionamiento. En consecuencia,

There Is More Than One Way To Do It (The Translation)

Es posible asociar una gramática árbol con el juego de instrucciones de una máquina. Las partes derechas de las reglas de producción de esta gramática vienen determinadas por el conjunto de árboles sintácticos de las instrucciones. La gramática tiene dos variables sintácticas que denotan dos tipos de recursos de la máquina: los registros representados por la variable sintáctica R y las direcciones de memoria representadas por M. Una instrucción deja su resultado en cierto lugar, normalmente un registro o memoria. La idea es que las variables sintácticas en los lados izquierdos de las reglas representan los lugares en los cuales las instrucciones dejan sus resultados.

Ademas, a cada instrucción le asociamos un coste:

Gramática Arbol Para un Juego de Instrucciones Simple		
Producción	Instrucción	Coste
R → NUM	`LOADC R, NUM`	1
R → M	`LOADM R, M`	3
M → R	`STOREM M, R`	3
R → PLUS(R,M)	`PLUSM R, M`	3
R → PLUS(R,R)	`PLUSR R, R`	1
R → TIMES(R,M)	`TIMESM R, M`	6
R → TIMES(R,R)	`TIMESR R, R`	4
R → PLUS(R,TIMES(NUM,R))	`PLUSCR R, NUM, R`	4
R → TIMES(R,TIMES(NUM,R))	`TIMESCR R, NUM, R`	5

Consideremos la IR consistente en el AST generado por el front-end del compilador para la expresión `x+3*(7*y)`:

```
PLUS(M[x],TIMES(N[3],TIMES(N[7],M[y])
```

Construyamos una derivación a izquierdas para el árbol anterior:

Una derivación árbol a izquierdas para $P(M,T(N,T(N,M)))$			
Derivación	Producción	Instrucción	Coste
$R \Longrightarrow$	R → PLUS(R,R)	`PLUSR R, R`	1
$P(R,R) \Longrightarrow$	R → M	`LOADM R, M`	3
$P(M,R) \Longrightarrow$	R → TIMES(R,R)	`TIMESR R, R`	4
$P(M,T(R,R)) \Longrightarrow$	R → NUM	`LOADC R, NUM`	1
$P(M,T(N,R)) \Longrightarrow$	R → TIMES(R,R)	`TIMESR R, R`	4
$P(M,T(N,T(R,R))) \Longrightarrow$	R → NUM	`LOADC R, NUM`	1
$P(M,T(N,T(N,R))) \Longrightarrow$	R → M	`LOADM R, M`	3
$P(M,T(N,T(N,M)))$			Total: 17

Obsérvese que, si asumimos por ahora que hay suficientes registros, la secuencia de instrucciones resultante en la tercera columna de la tabla si se lee en orden inverso (esto es, si se sigue el orden de instrucciones asociadas a las reglas de producción en orden de anti-derivación) y se hace una asignación correcta de registros nos da una traducción correcta de la expresión `x+3*(7*y)`:

```
LOADM R, M          # y
LOADC R, NUM        # 7
TIMESR R, R         # 7*y
LOADC R, NUM        # 3
TIMESR R, R         # 3*(7*y)
LOADM R, M          # x
PLUSR R, R          # x+3*(7*y)
```

La gramática anterior es ambigua. El árbol de `x+3*(7*y)` puede ser generado también mediante la siguiente derivación a izquierdas:

Otra derivación árbol a izquierdas para $P(M,T(N,T(N,M)))$			
Derivación	Producción	Instrucción	Coste
$R \Longrightarrow$	R → PLUS(R,TIMES(NUM,R))	`PLUSCR R, NUM, R`	4
$P(R,T(N,R)) \Longrightarrow$	R → M	`LOADM R, M`	3
$P(M,T(N,R)) \Longrightarrow$	R → TIMES(R,M)	`TIMESM R, M`	6
$P(M,T(N,T(R,M)))$	R → NUM	`LOADC R, NUM`	1
$P(M,T(N,T(N,M)))$			Total: 14

La nueva secuencia de instrucciones para `x+3*(7*y)` es:

```
LOADC R, NUM         # 7
TIMESM R, M          # 7*y
LOADM R, M           # x
PLUSCR R, NUM, R     # x+3*(7*y)
```

Cada antiderivación a izquierdas produce una secuencia de instrucciones que es una traducción legal del AST de `x+3*(7*y)`.

El problema de la selección de código óptima puede aproximarse resolviendo el problema de encontrar la derivación árbol óptima que produce el árbol de entrada (en representación intermedia IR)

Definición 9.2.1. *Un* generador de generadores de código *es una componente software que toma como entrada una especificación de la plataforma objeto -por ejemplo mediante una gramática árbol- y genera un módulo que es utilizado por el compilador. Este módulo lee la representación intermedia (habitualmente un árbol) y retorna código máquina como resultado.*

Un ejemplo de generador de generadores de código es iburg [8].

Véase también el libro Automatic Code Generation Using Dynamic Programming Techniques y la página http://www.bytelabs.org/hburg.html

Ejercicio 9.2.1. *Responda a las siguientes preguntas:*

- *Sea G_M la gramática árbol asociada segun la descripción anterior con el juego de instrucciones de la máquina M. Especifique formalmente las cuatro componentes de la gramática $G_M = (\Sigma_M, V_M, P_M, S_M)$*
- *¿Cual es el lenguaje árbol generado por G_M?*
- *¿A que lenguaje debe pertenecer la representación intermedia IR para que se pueda aplicar la aproximación presentada en esta sección?*

9.3. Patrones Árbol y Transformaciones Árbol

Una transformación de un programa puede ser descrita como un conjunto de *reglas de transformación* o *esquema de traducción árbol* sobre el árbol abstracto que representa el programa.

En su forma mas sencilla, estas reglas de transformación vienen definidas por ternas $(p, e, action)$, donde la primera componente de la terna p es un *patrón árbol* que dice que árboles deben ser seleccionados. La segunda componente e dice cómo debe transformarse el árbol que casa con el patrón p. La acción *action* indica como deben computarse los atributos del árbol transformado a partir de los atributos del árbol que casa con el patrón p. Una forma de representar este esquema sería:

$$p \Longrightarrow e \texttt{ \{ action \}}$$

Por ejemplo:

$PLUS(NUM_1, NUM_2) \Longrightarrow NUM_3$ `{ $NUM_3{VAL} = $NUM_1{VAL} + $NUM_2{VAL} }`

cuyo significado es que dondequiera que haya un nódo del AAA que case con el *patrón de entrada* $PLUS(NUM, NUM)$ deberá sustituirse el subárbol $PLUS(NUM, NUM)$ por el subárbol NUM. Al igual que en los esquemas de traducción, enumeramos las apariciones de los símbolos, para distinguirlos en la parte semántica. La acción indica como deben recomputarse los atributos para el nuevo árbol: El atributo `VAL` del árbol resultante es la suma de los atributos `VAL` de los operandos en el árbol que ha casado. La transformación se repite hasta que se produce la *normalización del árbol.*

Las reglas de "casamiento" de árboles pueden ser mas complejas, haciendo alusión a propiedades de los atributos, por ejemplo

$$ASSIGN(LEFTVALUE, x) \ and \texttt{ \{ notlive(\$LEFTVALUE\{VAL\}) \} } \Longrightarrow NIL$$

indica que se pueden eliminar aquellos árboles de tipo asignación en los cuáles la variable asociada con el nodo $LEFTVALUE$ no se usa posteriormente.

Otros ejemplos con variables S_1 y S_2:

$$IFELSE(NUM, S_1, S_2) \text{ and } \texttt{\{ \$NUM\{VAL\} != 0 \}} \Longrightarrow S_1$$
$$IFELSE(NUM, S_1, S_2) \text{ and } \texttt{\{ \$NUM\{VAL\} == 0 \}} \Longrightarrow S_2$$

Observe que en el patrón de entrada $ASSIGN(LEFTVALUE, x)$ aparece un "comodín": la variable-árbol x, que hace que el árbol patrón $ASSIGN(LEFTVALUE, x)$ case con cualquier árbol de asignación, independientemente de la forma que tenga su subárbol derecho.

Las siguientes definiciones formalizan una aproximación simplificada al significado de los conceptos *patrones árbol* y *casamiento de árboles.*

Patrón Árbol

Definición 9.3.1. *Sea (Σ, ρ) un alfabeto con función de aridad y un conjunto (puede ser infinito) de variables $V = \{x_1, x_2, \ldots\}$. Las variables tienen aridad cero:*

$$\rho(x) = 0 \ \forall x \in V.$$

Un elemento de $B(V \cup \Sigma)$ se denomina patrón *sobre Σ.*

Patrón Lineal

Definición 9.3.2. *Se dice que un patrón es un* patrón lineal *si ninguna variable se repite.*

Definición 9.3.3. *Se dice que un patrón es de tipo $(x_1, \ldots x_k)$ si las variables que aparecen en el patrón leidas de izquierda a derecha en el árbol son $x_1, \ldots x_k$.*

Ejemplo 9.3.1. *Sea* $\Sigma = \{A, CONS, NIL\}$ *con* $\rho(A) = \rho(NIL) = 0, \rho(CONS) = 2$ *y sea* $V = \{x\}$. *Los siguientes árboles son ejemplos de patrones sobre* Σ*:*

$$\{\ x, CONS(A, x), CONS(A, CONS(x, NIL)), \ldots\}$$

El patrón $CONS(x, CONS(x, NIL))$ *es un ejemplo de patrón no lineal. La idea es que un patrón lineal como éste "fuerza" a que los árboles* t *que casen con el patrón deben tener iguales los dos correspondientes subárboles* $t/1$ *y* $t/2.1$ *situados en las posiciones de las variables* [1]

Ejercicio 9.3.1. *Dado la gramática árbol:*

$$\begin{array}{c} S \rightarrow S_1(a, S, b) \\ S \rightarrow S_2(NIL) \end{array}$$

la cuál genera los árboles concretos para la gramática

$$S \rightarrow aSb \mid \epsilon$$

¿Es $S_1(a, X(NIL), b)$ *un patrón árbol sobre el conjunto de variables* $\{X, Y\}$*? ¿Lo es* $S_1(X, Y, a)$*? ¿Es* $S_1(X, Y, Y)$ *un patrón árbol?*

Ejemplo 9.3.2. *Ejemplos de patrones para el AAA definido en el ejemplo* **??** *para el lenguaje Tutu son:*

$$x, y, PLUS(x, y), ASSIGN(x, TIMES(y, ID)), PRINT(y) \ldots$$

considerando el conjunto de variables $V = \{x, y\}$*. El patrón* $ASSIGN(x, TIMES(y, ID))$ *es del tipo* (x, y)*.*

Sustitución

Definición 9.3.4. *Una* sustitución árbol *es una aplicación* θ *que asigna variables a patrones* $\theta : V \rightarrow B(V \cup \Sigma)$.

Tal función puede ser naturalmente extendida de las variables a los árboles: los nodos (hoja) etiquetados con dichas variables son sustituidos por los correspondientes subárboles.

$$\begin{array}{l} \theta : B(V \cup \Sigma) \rightarrow B(V \cup \Sigma) \\ t\theta = \left\{ \begin{array}{ll} x\theta & si\ t = x \in V \\ a(t_1\theta, \ldots, t_k\theta) & si\ t = a(t_1, \ldots, t_k) \end{array} \right. \end{array}$$

Obsérvese que, al revés de lo que es costumbre, la aplicación de la sustitución θ *al patrón se escribe por detrás:* $t\theta$*.*

También se escribe $t\theta = t\{x_1/x_1\theta, \ldots x_k/x_k\theta\}$ *si las variables que aparecen en* t *de izquierda a derecha son* $x_1, \ldots x_k$*.*

Ejemplo 9.3.3. *Si aplicamos la sustitución* $\theta = \{x/A, y/CONS(A, NIL)\}$ *al patrón* $CONS(x, y)$ *obtenemos el árbol* $CONS(A, CONS(A, NIL))$*. En efecto:*

$$CONS(x, y)\theta = CONS(x\theta, y\theta) = CONS(A, CONS(A, NIL))$$

Ejemplo 9.3.4. *Si aplicamos la sustitución* $\theta = \{x/PLUS(NUM, x), y/TIMES(ID, NUM)\}$ *al patrón* $PLUS(x, y)$ *obtenemos el árbol* $PLUS(PLUS(NUM, x), TIMES(ID, NUM))$*:*

$$PLUS(x, y)\theta = PLUS(x\theta, y\theta) = PLUS(PLUS(NUM, x), TIMES(ID, NUM))$$

[1] Repase la notación de Dewey introducida en la definición **??**

Casamiento Árbol

Definición 9.3.5. *Se dice que un patrón* $\tau \in B(V \cup \Sigma)$ *con variables* $x_1, \ldots x_k$ casa con un árbol $t \in B(\Sigma)$ *si existe una sustitución de* τ *que produce* t*, esto es, si existen* $t_1, \ldots t_k \in B(\Sigma)$ *tales que* $t = \tau\{x_1/t_1, \ldots x_k/t_k\}$*. También se dice que* τ casa con la sustitución $\{x_1/t_1, \ldots x_k/t_k\}$.

Ejemplo 9.3.5. *El patrón* $\tau = CONS(x, NIL)$ *casa con el árbol* $t = CONS(CONS(A, NIL), NIL)$ *y con el subárbol* $t.\,1$*. Las respectivas sustituciones son* $t\{x/CONS(A, NIL)\}$ *y* $t.\,1\{x/A\}$.

$$t = \tau\{x/CONS(A, NIL)\}$$
$$t.\,1 = \tau\{x/A\}$$

Ejercicio 9.3.2. *Sea* $\tau = PLUS(x, y)$ *y* $t = TIMES(PLUS(NUM, NUM), TIMES(ID, ID))$*. Calcule los subárboles* t' *de* t *y las sustituciones* $\{x/t_1, y/t_2\}$ *que hacen que* τ *case con* t'*.*

Por ejemplo es obvio que para el árbol raíz t/ϵ *no existe sustitución posible:*
$t = TIMES(PLUS(NUM, NUM), TIMES(ID, ID)) = \tau\{x/t_1, y/t_2\} = PLUS(x, y)\{x/t_1, y/t_2\}$
ya que un término con raíz $TIMES$ *nunca podrá ser igual a un término con raíz* $PLUS$*.*

El problema aquí es equivalente al de las expresiones regulares en el caso de los lenguajes lineales. En aquellos, los autómatas finitos nos proveen con un mecanismo para reconocer si una determinada cadena “casa”’ o no con la expresión regular. Existe un concepto análogo, el de *autómata árbol* que resuelve el problema del “casamiento” de patrones árbol. Al igual que el concepto de autómata permite la construcción de software para la búsqueda de cadenas y su posterior modificación, el concepto de autómata árbol permite la construcción de software para la búsqueda de los subárboles que casan con un patrón árbol dado.

9.4. Ejemplo de Transformaciones Árbol: Parse::Eyapp::TreeRegexp

Instalación

```
[~/jison/jison-aSb(master)]$ sudo cpan Parse::Eyapp
```

Donde

- `[~/src/perl/parse-eyapp/examples/MatchingTrees]$ pwd -P`
  `/Users/casiano/local/src/perl/parse-eyapp/examples/MatchingTrees`
- Parse::Eyapp
- Ejemplo de uso de Parse::Eyapp::Treeregexp
- Tree Matching and Tree Substitution
- Node.pm (Véase el método `s`)

La gramática: Expresiones

```
my $grammar = q{
  %lexer {
      m{\G\s+}gc;
      m{\G([0-9]+(?:\.[0-9]+)?)}gc and return('NUM',$1);
      m{\G([A-Za-z][A-Za-z0-9_]*)}gc and return('VAR',$1);
      m{\G(.)}gcs and return($1,$1);
  }

  %right  '='     # Lowest precedence
```

```
  %left   '-' '+' # + and - have more precedence than = Disambiguate a-b-c as (a-b)-c
  %left   '*' '/' # * and / have more precedence than + Disambiguate a/b/c as (a/b)/c
  %left   NEG     # Disambiguate -a-b as (-a)-b and not as -(a-b)
  %tree           # Let us build an abstract syntax tree ...

  %%
  line:
      exp <%name EXPRESSION_LIST + ';'>
        { $_[1] } /* list of expressions separated by ';' */
  ;

  /* The %name directive defines the name of the
     class to which the node being built belongs */
  exp:
      %name NUM
      NUM
    | %name VAR
      VAR
    | %name ASSIGN
      VAR '=' exp
    | %name PLUS
      exp '+' exp
    | %name MINUS
      exp '-' exp
    | %name TIMES
      exp '*' exp
    | %name DIV
      exp '/' exp
    | %name UMINUS
      '-' exp %prec NEG
    | '(' exp ')'
        { $_[2] }  /* Let us simplify a bit the tree */
  ;

  %%
}; # end grammar
```

Ejecución

```
El trozo de código:
\begin{verbatim}
$parser->input(\"2*-3+b*0;--2\n");       # Set the input
my $t = $parser->YYParse;
```

da lugar a este árbol:

```
[~/src/perl/parse-eyapp/examples/MatchingTrees]$ ./synopsis.pl
Syntax Tree:
EXPRESSION_LIST(
  PLUS(
    TIMES(
      NUM( TERMINAL[2]),
      UMINUS( NUM( TERMINAL[3])) # UMINUS
    ) # TIMES,
    TIMES( VAR( TERMINAL[b]), NUM( TERMINAL[0])) # TIMES
```

```
  ) # PLUS,
  UMINUS(
    UMINUS( NUM( TERMINAL[2])) # UMINUS
  ) # UMINUS
) # EXPRESSION_LIST
```

Al aplicar las transformaciones:

```
# Let us transform the tree. Define the tree-regular expressions ..
my $p = Parse::Eyapp::Treeregexp->new( STRING => q{
    { #  Example of support code
      my %Op = (PLUS=>'+', MINUS => '-', TIMES=>'*', DIV => '/');
    }
    constantfold: /TIMES|PLUS|DIV|MINUS/:bin(NUM($x), NUM($y))
      => {
        my $op = $Op{ref($bin)};
        $x->{attr} = eval  "$x->{attr} $op $y->{attr}";
        $_[0] = $NUM[0];
      }
    uminus: UMINUS(NUM($x)) => { $x->{attr} = -$x->{attr}; $_[0] = $NUM }
    zero_times_whatever: TIMES(NUM($x), .) and { $x->{attr} == 0 } => { $_[0] = $NUM }
    whatever_times_zero: TIMES(., NUM($x)) and { $x->{attr} == 0 } => { $_[0] = $NUM }
  },
  OUTPUTFILE=> 'main.pm'
);
$p->generate(); # Create the tranformations

$t->s($uminus); # Transform UMINUS nodes
$t->s(@all);    # constant folding and mult. by zero
```

Obtenemos el árbol:

```
Syntax Tree after transformations:
EXPRESSION_LIST(NUM(TERMINAL[-6]),NUM(TERMINAL[2]))
```

synopsis.pl

```
[~/src/perl/parse-eyapp/examples/MatchingTrees]$ cat synopsis.pl
#!/usr/bin/perl -w
use strict;
use Parse::Eyapp;
use Parse::Eyapp::Treeregexp;

sub TERMINAL::info {
  $_[0]{attr}
}

my $grammar = q{
  %lexer {
      m{\G\s+}gc;
      m{\G([0-9]+(?:\.[0-9]+)?)}gc and return('NUM',$1);
      m{\G([A-Za-z][A-Za-z0-9_]*)}gc and return('VAR',$1);
      m{\G(.)}gcs and return($1,$1);
  }
```

```
  %right  '='     # Lowest precedence
  %left   '-' '+' # + and - have more precedence than = Disambiguate a-b-c as (a-b)-c
  %left   '*' '/' # * and / have more precedence than + Disambiguate a/b/c as (a/b)/c
  %left   NEG     # Disambiguate -a-b as (-a)-b and not as -(a-b)
  %tree           # Let us build an abstract syntax tree ...

  %%
  line:
      exp <%name EXPRESSION_LIST + ';'>
        { $_[1] } /* list of expressions separated by ';' */
  ;

  /* The %name directive defines the name of the
     class to which the node being built belongs */
  exp:
      %name NUM
      NUM
    | %name VAR
      VAR
    | %name ASSIGN
      VAR '=' exp
    | %name PLUS
      exp '+' exp
    | %name MINUS
      exp '-' exp
    | %name TIMES
      exp '*' exp
    | %name DIV
      exp '/' exp
    | %name UMINUS
      '-' exp %prec NEG
    | '(' exp ')'
        { $_[2] }  /* Let us simplify a bit the tree */
  ;

  %%
}; # end grammar

our (@all, $uminus);

Parse::Eyapp->new_grammar( # Create the parser package/class
  input=>$grammar,
  classname=>'Calc', # The name of the package containing the parser
);
my $parser = Calc->new();                # Create a parser
$parser->input(\"2*-3+b*0;--2\n");       # Set the input
my $t = $parser->YYParse;                # Parse it!
local $Parse::Eyapp::Node::INDENT=2;
print "Syntax Tree:",$t->str;

# Let us transform the tree. Define the tree-regular expressions ..
my $p = Parse::Eyapp::Treeregexp->new( STRING => q{
    { #  Example of support code
```

```
    my %Op = (PLUS=>'+', MINUS => '-', TIMES=>'*', DIV => '/');
  }
  constantfold: /TIMES|PLUS|DIV|MINUS/:bin(NUM($x), NUM($y))
    => {
      my $op = $Op{ref($bin)};
      $x->{attr} = eval  "$x->{attr} $op $y->{attr}";
      $_[0] = $NUM[0];
    }
  uminus: UMINUS(NUM($x)) => { $x->{attr} = -$x->{attr}; $_[0] = $NUM }
  zero_times_whatever: TIMES(NUM($x), .) and { $x->{attr} == 0 } => { $_[0] = $NUM }
  whatever_times_zero: TIMES(., NUM($x)) and { $x->{attr} == 0 } => { $_[0] = $NUM }
 },
 OUTPUTFILE=> 'main.pm'
);
$p->generate(); # Create the tranformations

$t->s($uminus); # Transform UMINUS nodes
$t->s(@all);    # constant folding and mult. by zero

local $Parse::Eyapp::Node::INDENT=0;
print "\nSyntax Tree after transformations:\n",$t->str,"\n";
```

El método s

El código de s está en lib/Parse/Eyapp/Node.pm:

```
sub s {
  my @patterns = @_[1..$#_];

  # Make them Parse::Eyapp:YATW objects if they are CODE references
  @patterns = map { ref($_) eq 'CODE'?
                      Parse::Eyapp::YATW->new(
                        PATTERN => $_,
                        #PATTERN_ARGS => [],
                      )
                      :
                      $_
                  }
                  @patterns;
  my $changes;
  do {
    $changes = 0;
    foreach (@patterns) {
      $_->{CHANGES} = 0;
      $_->s($_[0]);
      $changes += $_->{CHANGES};
    }
  } while ($changes);
}
```

Véase

- Parse::Eyapp
- Ejemplo de uso de Parse::Eyapp::Treeregexp

- Tree Matching and Tree Substitution
- Node.pm (Véase el método `s`)

9.5. Treehugger

Donde

- `[~/srcPLgrado/treehugger(master)]$ pwd -P`
  `/Users/casiano/local/src/javascript/PLgrado/treehugger`
- `[~/srcPLgrado/treehugger(master)]$ git remote -v`
  `origin  git@github.com:crguezl/treehugger.git (fetch)`
  `origin  git@github.com:crguezl/treehugger.git (push)`
- https://github.com/crguezl/treehugger

learning.html

```
[~/srcPLgrado/treehugger(master)]$ cat learning.html
<!DOCTYPE html>
<html>
  <head>
    <title>treehugger.js demo</title>
    <script data-main="lib/demo" src="lib/require.js"></script>
    <link rel="stylesheet" href="examples/style.css" type="text/css" />
  </head>
  <body>
  <h1>Treehugger.js playground</h1>
    <table>
    <tr>
      <th>Javascript</th>
      <th>AST</th>
    </tr>
    <tr>
      <td><textarea id="code" rows="15" cols="42">var a = 10, b;
console.log(a, b, c);</textarea></td>
      <td><textarea id="ast" rows="15" cols="42" readonly style="background-color: #eee;"></te
    </tr>
    <tr>
      <th>Analysis code <button id="runbutton">Run</button></th>
      <th>Output</th>
    </tr>
    <tr>
      <td><textarea id="analysis" rows="15" cols="42">var declared = {console: true};

ast.traverseTopDown(
   'VarDecl(x)', function(b) {
      declared[b.x.value] = true;
   },
   'VarDeclInit(x, _)', function(b) {
      declared[b.x.value] = true;
   },
   'Var(x)', function(b) {
      if(!declared[b.x.value])
```

```
        log("Variable " + b.x.value + " is not declared.");
   }
);
</textarea></td>
      <td><textarea id="output" rows="15" cols="42" readonly style="background-color: #eee;"><
    </tr>
    </table>
  </body>
</html>
```

lib/demo.js

```
[~/srcPLgrado/treehugger(master)]$ cat lib/demo.js
require({ baseUrl: "lib" },
    ["treehugger/tree",
     "treehugger/traverse",
     "treehugger/js/parse",
     "jquery",
     "treehugger/js/acorn", // Acorn is a JavaScript parser
     "treehugger/js/acorn_loose" // This module provides an alternative
                                 // parser with the same interface as
                                 // `parse`, but will try to parse
                                 // anything as JavaScript, repairing
                                 // syntax error the best it can.
    ], function(tree, traverse, parsejs, jq, acorn, acorn_loose) {
          window.acorn_loose = acorn_loose

          function log(message) {
            $("#output").val($("#output").val() + message + "\n");
          }

          function exec() {
            var js = $("#code").val();
            var analysisJs = $("#analysis").val();
            $("#output").val("");

            // https://developer.mozilla.org/en-US/docs/Web/API/Performance.now()
            var t = performance.now();
            var ast = parsejs.parse(js);
            t -= performance.now();
            $("#ast").val(t + "\n" + ast.toPrettyString());
            try {
              eval(analysisJs);
            } catch(e) {
              $("#output").val("JS Error");
              console.log(e.message)
            }
          }

          tree.Node.prototype.log = function() {
            $("#output").val(this.toPrettyString());
          }

          require.ready(function() {
```

```
            $("#code").keyup(exec);
            $("#runbutton").click(exec);
            exec();
        });
    }
);
```

Véase

- treehugger.js is a Javascript library for program processing. It has generic means to represent and manipula
- You can see treehugger.js in action in this simple demo.
- Avoiding JavaScript Pitfalls Through Tree Hugging YouTube. Slides.
- AST traversal javascript libraries
- RequireJS

9.6. Práctica: Transformaciones en Los Árboles del Analizador PL0

Partimos del código realizado en la práctica *Análisis de Ámbito en PL0* 7.7.

Modifique el árbol generado por el código de esa práctica usando las transformaciones de *constant folding* o *plegado de las constantes*:

$PLUS(NUM_1, NUM_2) \Longrightarrow NUM_3$ `{ $NUM_3{VAL} = $NUM_1{VAL} + $NUM_2{VAL} }` $MINUS(NUM_1, NU$

NUM_3 `{ $NUM_3{VAL} = $NUM_1{VAL} - $NUM_2{VAL} }` $TIMES(NUM_1, NUM_2) \Longrightarrow NUM_3$ `{ $NUM_3{VAL`

$DIV(NUM_1, NUM_2) \Longrightarrow NUM_3$ `{ $NUM_3{VAL} = $NUM_1{VAL} / $NUM_2{VAL} }`

etc.

Opcionalmente si lo desea puede considerar otras transformaciones: $TIMES(X, NUM_2)$ and

`{ $NUM_2{VAL} = ` 2^s para algún s `}` $\Longrightarrow SHIFTLEFT(X; NUM_3)$ `{ $NUM_3{VAL} = s }`

Parte II

PARTE: CREATE YOUR OWN PROGRAMMING LANGUAGE

A course by Nathan Whitehead.

- Nathan Whitehead en YouTube

Repositorios relacionados:

- https://github.com/crguezl/nathanuniversityexercisesPL

Capítulo 10

JavaScript Review

http://nathansuniversity.com/jsreview.html

10.1. Closures

http://nathansjslessons.appspot.com/

Capítulo 11

Your First Compiler

http://nathansuniversity.com/music.html

Capítulo 12

Parsing

http://nathansuniversity.com/pegs.html

Capítulo 13

Scheem Interpreter

http://nathansuniversity.com/scheem.html

13.1. Scheem Interpreter

13.2. Variables

13.3. Setting Values

13.4. Putting Things Together

13.4.1. Unit Testing: Mocha

Introducción

Mocha is a feature-rich JavaScript test framework running on node.js and the browser, making asynchronous testing simple and fun. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases.

- http://visionmedia.github.io/mocha/
- https://github.com/visionmedia/mocha
- - An example setup for unit testing JavaScript in the browser with the Mocha testing framework and Chai assertions: https://github.com/ludovicofischer/mocha-chai-browser-demo
  - Karma - a test runner

mocha init

```
[~/srcPLgrado/mocha-chai-browser-demo(master)]$ mocha --help

  Usage: _mocha [debug] [options] [files]

  Commands:

    init <path>            initialize a client-side mocha setup at <path>

  Options:

    -h, --help                      output usage information
    -V, --version                   output the version number
    -r, --require <name>            require the given module
    -R, --reporter <name>           specify the reporter to use
```

```
    -u, --ui <name>                  specify user-interface (bdd|tdd|exports)
    -g, --grep <pattern>             only run tests matching <pattern>
    -i, --invert                     inverts --grep matches
    -t, --timeout <ms>               set test-case timeout in milliseconds [2000]
    -s, --slow <ms>                  "slow" test threshold in milliseconds [75]
    -w, --watch                      watch files for changes
    -c, --colors                     force enabling of colors
    -C, --no-colors                  force disabling of colors
    -G, --growl                      enable growl notification support
    -d, --debug                      enable node's debugger, synonym for node --debug
    -b, --bail                       bail after first test failure
    -A, --async-only                 force all tests to take a callback (async)
    -S, --sort                       sort test files
    --recursive                      include sub directories
    --debug-brk                      enable node's debugger breaking on the first line
    --globals <names>                allow the given comma-delimited global [names]
    --check-leaks                    check for global variable leaks
    --interfaces                     display available interfaces
    --reporters                      display available reporters
    --compilers <ext>:<module>,...   use the given module(s) to compile files
    --inline-diffs                   display actual/expected differences inline within each str
    --no-exit                        require a clean shutdown of the event loop: mocha will not
```

```
[~/srcPLgrado]$ mocha init chuchu
[~/srcPLgrado]$ ls -ltr
total 16
....
drwxr-xr-x   6 casiano  staff  204 20 ene 11:16 chuchu
[~/srcPLgrado]$ tree chuchu/
chuchu/
|-- index.html
|-- mocha.css
|-- mocha.js
`-- tests.js
```

```
[~/srcPLgrado/mocha-tutorial]$ cat test/test.js
var assert = require("assert")
describe('Array', function(){
  describe('#indexOf()', function(){
    it('should return -1 when the value is not present', function(){
      assert.equal(-1, [1,2,3].indexOf(5));
      assert.equal(-1, [1,2,3].indexOf(0));
      assert.equal( 0, [1,2,3].indexOf(99));
    })
  })
})
```

```
[~/srcPLgrado/mocha-tutorial]$ mocha
  .
  0 passing (5ms)
  1 failing

  1) Array #indexOf() should return -1 when the value is not present:
     AssertionError: 0 == -1
      at Context.<anonymous> (/Users/casiano/local/src/javascript/PLgrado/mocha-tutorial/test/
```

Mocha allows you to use any assertion library you want, if it throws an error, it will work! This means you can utilize libraries such as should.js, node's regular assert module, or others.

Browser support
Mocha runs in the browser.

- Every release of Mocha will have new builds of `./mocha.js` and `./mocha.css` for use in the browser.
- To setup Mocha for browser use all you have to do is include the script, stylesheet,
- Tell Mocha which interface you wish to use, and then
- Run the tests.

A typical setup might look something like the following, where we call `mocha.setup('bdd')` to use the BDD interface before loading the test scripts, running them onload with `mocha.run()`.

```
<html>
<head>
  <meta charset="utf-8">
  <title>Mocha Tests</title>
  <link rel="stylesheet" href="mocha.css" />
</head>
<body>
  <div id="mocha"></div>
  <script src="jquery.js"></script>
  <script src="expect.js"></script>
  <script src="mocha.js"></script>

  <script>mocha.setup('bdd')</script>

  <script src="test.array.js"></script>
  <script src="test.object.js"></script>
  <script src="test.xhr.js"></script>

  <script>
    mocha.checkLeaks();
    mocha.globals(['jQuery']);
    mocha.run();
  </script>

</body>
</html>
```

- Mocha ïnterface"system allows developers to choose their style of DSL. Shipping with BDD, TDD, and exports flavoured interfaces.
- `mocha.globals([names ...])`

  A list of accepted global variable names. For example, suppose your app deliberately exposes a global named app and YUI
- `mocha.checkLeaks()`

  By default Mocha will not check for global variables leaked while running tests

TDD

The *Mocha TDD interface* provides `suite()`, `test()`, `setup()`, and `teardown()`.

```
suite('Array', function(){
  setup(function(){
    // ...
  });

  suite('#indexOf()', function(){
    test('should return -1 when not present', function(){
      assert.equal(-1, [1,2,3].indexOf(4));
    });
  });
});
```

Véase

- https://github.com/crguezl/nathanuniversityexercisesPL/tree/master/scheem8

13.4.2. Karma

- *Karma* (See Karma installation) is essentially a tool which spawns a web server that executes source code against test code for each of the browsers connected.
- The results for each test against each browser are examined and displayed via the command line to the developer such that they can see which browsers and tests passed or failed.
- A browser can be captured either
  - manually, by visiting the URL where the Karma server is listening (typically `http://localhost:9876/`)
  - or automatically by letting Karma know which browsers to start when Karma is run
- Karma also watches all the files, specified within the configuration file, and whenever any file changes, it triggers the test run by sending a signal the testing server to inform all of the captured browsers to run the test code again.
- Each browser then loads the source files inside an `IFrame`[1], executes the tests and reports the results back to the server.
- The server collects the results from all of the captured browsers and presents them to the developer.
- JS.everywhere(Europe) 2012: Testacular, the Spectacular JavaScript Test Runner - Vojta Jína YouTube
- Google Test Automation Conference GTAC 2013: Karma - Test Runner for JavaScript Vojta Jína. YouTube

```
[~/srcPLgrado/mocha-chai-browser-demo(master)]$ karma --help
Karma - Spectacular Test Runner for JavaScript.

Usage:
  /usr/local/bin/karma <command>
```

[1]The `iframe` tag specifies an inline frame. An inline frame is used to embed another document within the current HTML document

```
Commands:
  start [<configFile>] [<options>] Start the server / do single run.
  init [<configFile>] Initialize a config file.
  run [<options>] [ -- <clientArgs>] Trigger a test run.
  completion Shell completion for karma.

Run --help with particular command to see its description and available options.

Options:
  --help     Print usage and options.
  --version  Print current version.
```

In order to serve us well, Karma needs to know about our project in order to test it and this is done via a configuration file.

The configuration file can be generated using `karma init`:

```
$ karma init my.conf.js

Which testing framework do you want to use ?
Press tab to list possible options. Enter to move to the next question.
> jasmine

Do you want to use Require.js ?
This will add Require.js plugin.
Press tab to list possible options. Enter to move to the next question.
> no
```

http://requirejs.org/

```
Do you want to capture a browser automatically ?
Press tab to list possible options. Enter empty string to move to the next question.
> Chrome

What is the location of your source and test files ?
You can use glob patterns, eg. "js/*.js" or "test/**/*Spec.js".
Enter empty string to move to the next question.
>

Should any of the files included by the previous patterns be excluded ?
You can use glob patterns, eg. "**/*.swp".
Enter empty string to move to the next question.


Do you want Karma to watch all the files and run the tests on change ?
Press tab to list possible options.
> yes

Config file generated at "/Users/casiano/local/src/javascript/PLgrado/mocha-tutorial/karma.con
```

The configuration file can be written in CoffeeScript as well. In fact, if you execute karma init with a `.coffee` filename extension, it will generate a CoffeeScript file.

Of course, you can write the config file by hand or copy paste it from another project ;-)

```
[~/srcPLgrado/mocha-tutorial]$ cat karma.conf.js
// Karma configuration
// Generated on Mon Jan 20 2014 16:21:22 GMT+0000 (WET)

module.exports = function(config) {
  config.set({

    // base path, that will be used to resolve files and exclude
    basePath: '',


    // frameworks to use
    frameworks: ['jasmine'],


    // list of files / patterns to load in the browser
    files: [

    ],


    // list of files to exclude
    exclude: [

    ],


    // test results reporter to use
    // possible values: 'dots', 'progress', 'junit', 'growl', 'coverage'
    reporters: ['progress'],


    // web server port
    port: 9876,


    // enable / disable colors in the output (reporters and logs)
    colors: true,


    // level of logging
    // possible values: config.LOG_DISABLE || config.LOG_ERROR || config.LOG_WARN || config.LO
    logLevel: config.LOG_INFO,


    // enable / disable watching file and executing tests whenever any file changes
    autoWatch: true,


    // Start these browsers, currently available:
    // - Chrome
    // - ChromeCanary
    // - Firefox
```

```
    // - Opera (has to be installed with `npm install karma-opera-launcher`)
    // - Safari (only Mac; has to be installed with `npm install karma-safari-launcher`)
    // - PhantomJS
    // - IE (only Windows; has to be installed with `npm install karma-ie-launcher`)
    browsers: ['Chrome', 'Firefox'],


    // If browser does not capture in given timeout [ms], kill it
    captureTimeout: 60000,


    // Continuous Integration mode
    // if true, it capture browsers, run tests and exit
    singleRun: false
  });
};
```

When starting Karma, the configuration file path can be passed in as the first argument. By default, Karma will look for `karma.conf.js` in the current directory.

```
# Start Karma using your configuration
$ karma start my.conf.js
```

Some configurations, which are already present within the configuration file, can be overridden by specifying the configuration as a command line argument for when Karma is executed.

```
karma start karma-conf.js --command-one --command-two
```

```
[~/srcPLgrado/mocha-tutorial]$ karma start --help
Karma - Spectacular Test Runner for JavaScript.

START - Start the server / do a single run.

Usage:
  /usr/local/bin/karma start [<configFile>] [<options>]

Options:
  --port                 <integer> Port where the server is running.
  --auto-watch           Auto watch source files and run on change.
  --no-auto-watch        Do not watch source files.
  --log-level            <disable | error | warn | info | debug> Level of logging.
  --colors               Use colors when reporting and printing logs.
  --no-colors            Do not use colors when reporting or printing logs.
  --reporters            List of reporters (available: dots, progress, junit, growl, coverage).
  --browsers             List of browsers to start (eg. --browsers Chrome,ChromeCanary,Firefox)
  --capture-timeout      <integer> Kill browser if does not capture in given time [ms].
  --single-run           Run the test when browsers captured and exit.
  --no-single-run        Disable single-run.
  --report-slower-than   <integer> Report tests that are slower than given time [ms].
  --help                 Print usage and options.
```

Using Karma with Mocha To use Karma with Mocha we need the karma-mocha adapter.

If we want to pass configuration options directly to mocha you can do this in the following way

```
// karma.conf.js
```

```
module.exports = function(config) {
  config.set({
    frameworks: ['mocha'],

    files: [
      '*.js'
    ],

    client: {
      mocha: {
        ui: 'tdd'
      }
    }
  });
};
```

(By default the `ui` is `bdd`).

Here is an example (https://github.com/crguezl/nathanuniversityexercisesPL/blob/master/scheem8/karma.co

```
[~/srcPLgrado/nathansuniversity/exercises/scheem8(master)]$ cat karma.conf.js
// Karma configuration
// Generated on Tue Jan 21 2014 12:20:45 GMT+0000 (WET)

module.exports = function(config) {

  config.set({

    // base path, that will be used to resolve files and exclude
    basePath: '',


    // frameworks to use
    frameworks: ['mocha'],


    // list of files / patterns to load in the browser
    files: [
      'js/chai.js',
      'js/jquery-1.10.2.js',
      'js/mocha.js',
      'js/scheem8.js',
      'js/simpletest.js'
    ],


    // list of files to exclude
    exclude: [

    ],


    // test results reporter to use
    // possible values: 'dots', 'progress', 'junit', 'growl', 'coverage'
    reporters: ['progress'],
```

```
    // web server port
    port: 9876,


    // enable / disable colors in the output (reporters and logs)
    colors: true,


    // level of logging
    // possible values: config.LOG_DISABLE || config.LOG_ERROR || config.LOG_WARN || config.LO
    logLevel: config.LOG_INFO,


    // enable / disable watching file and executing tests whenever any file changes
    autoWatch: true,


    // Start these browsers, currently available:
    // - Chrome
    // - ChromeCanary
    // - Firefox
    // - Opera (has to be installed with `npm install karma-opera-launcher`)
    // - Safari (only Mac; has to be installed with `npm install karma-safari-launcher`)
    // - PhantomJS
    // - IE (only Windows; has to be installed with `npm install karma-ie-launcher`)
    browsers: ['Chrome', 'Firefox'],


    // If browser does not capture in given timeout [ms], kill it
    captureTimeout: 60000,


    // Continuous Integration mode
    // if true, it capture browsers, run tests and exit
    singleRun: false,

    client: {
      mocha: {
        ui: 'tdd'
      }
    }
  });
};
```

Load HTML files with Karma

If you have one html file:

```
[~/srcPLgrado/karma/html]$ cat template.html
<div id="tpl">content of the template</div>
```

which you want to load and then get all elements from that html page in your test script, you can use the `html2js` preprocessor, which basically converts HTML files into JavaScript strings and include these files.

```
[~/srcPLgrado/karma/html]$ cat karma.conf.js
module.exports = function(karma) {
  karma.configure({
    basePath: '',
    frameworks: ['jasmine'],
    files: [ '*.js', '*.html' ],
    preprocessors: { '*.html': 'html2js' },
    ....
```

Then, you can access these strings in your test:

```
[~/srcPLgrado/karma/html]$ cat test.js
describe('template', function() {
  it('should expose the templates to __html__', function() {
    document.body.innerHTML = __html__['template.html'];
    expect(document.getElementById('tpl')).toBeDefined();
  })
})
```

See

- Load HTML files with Karma in StackOverflow.
- karma-html2js-preprocessor
- Example

13.4.3. Grunt

http://gruntjs.com/getting-started

```
npm install -g grunt-cli
```

A typical setup will involve adding two files to your project: `package.json` and the `Gruntfile`.

- `package.json`: This file is used by `npm` to store metadata for projects published as `npm` modules. You will list grunt and the Grunt plugins your project needs as *devDependencies* in this file.
- `Gruntfile`: This file is named `Gruntfile.js` or `Gruntfile.coffee` and is used to configure or define tasks and load Grunt plugins.

package.json

- The package.json file belongs in the root directory of your project, next to the Gruntfile, and should be committed with your project source.
- Running npm install in the same folder as a package.json file will install the correct version of each dependency listed therein.
- There are a few ways to create a package.json file for your project:
    - Most grunt-init templates will automatically create a project-specific package.json file.
    - The npm init command will create a basic package.json file.
    - Start with the example below, and expand as needed, following this specification.

```
{
  "name": "my-project-name",
  "version": "0.1.0",
  "devDependencies": {
    "grunt": "~0.4.2",
    "grunt-contrib-jshint": "~0.6.3",
    "grunt-contrib-nodeunit": "~0.2.0",
    "grunt-contrib-uglify": "~0.2.2"
  }
}
```

Gruntfile

The Gruntfile.js or Gruntfile.coffee file is a valid JavaScript or CoffeeScript file that belongs in the root directory of your project, next to the package.json file, and should be committed with your project source.

A Gruntfile is comprised of the following parts:

- The "wrapper"function
- Project and task configuration
- Loading Grunt plugins and tasks
- Custom tasks

An example Gruntfile

In the following Gruntfile, project metadata is imported into the Grunt config from the project's `package.json` file and the

grunt-contrib-uglify

plugin's uglify task is configured to minify a source file and generate a banner comment dynamically using that metadata.

When grunt is run on the command line, the uglify task will be run by default.

```
module.exports = function(grunt) {

  // Project configuration.
  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    uglify: {
      options: {
        banner: '/*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> */\n'
      },
      build: {
        src: 'src/<%= pkg.name %>.js',
        dest: 'build/<%= pkg.name %>.min.js'
      }
    }
  });

  // Load the plugin that provides the "uglify" task.
  grunt.loadNpmTasks('grunt-contrib-uglify');

  // Default task(s).
```

```
  grunt.registerTask('default', ['uglify']);

};
```

Now that you've seen the whole Gruntfile, let's look at its component parts.

The "wrapper"function

Every Gruntfile (and gruntplugin) uses this basic format, and all of your Grunt code must be specified inside this function:

```
module.exports = function(grunt) {
  // Do grunt-related things in here
};
```

Project and task configuration

Most Grunt tasks rely on configuration data defined in an object passed to the `grunt.initConfig` method.

In this example, `grunt.file.readJSON('package.json')` imports the JSON metadata stored in `package.json` into the grunt config. Because `<% %>` template strings may reference any config properties, configuration data like filepaths and file lists may be specified this way to reduce repetition.

You may store any arbitrary data inside of the configuration object, and as long as it doesn't conflict with properties your tasks require, it will be otherwise ignored. Also, because this is JavaScript, you're not limited to JSON; you may use any valid JS here. You can even programmatically generate the configuration if necessary.

Like most tasks, the `grunt-contrib-uglify` plugin's uglify task expects its configuration to be specified in a property of the same name. Here, the `banner` option is specified, along with a single uglify target named `build` that minifies a single source file to a single destination file.

```
// Project configuration.
grunt.initConfig({
  pkg: grunt.file.readJSON('package.json'),
  uglify: {
    options: {
      banner: '/*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> */\n'
    },
    build: {
      src: 'src/<%= pkg.name %>.js',
      dest: 'build/<%= pkg.name %>.min.js'
    }
  }
});
```

A simple Grunt.js example

https://github.com/UWMadisonUcomm/grunt-simple-example

```
[~/srcPLgrado/grunt-simple-example(master)]$ pwd
/Users/casiano/srcPLgrado/grunt-simple-example
[~/srcPLgrado/grunt-simple-example(master)]$ git remote -v
origin  git@github.com:UWMadisonUcomm/grunt-simple-example.git (fetch)
origin  git@github.com:UWMadisonUcomm/grunt-simple-example.git (push)
[~/srcPLgrado/grunt-simple-example(master)]$ ls
Gruntfile.js Readme.md    assets       index.html   node_modules package.json src
```

```
[~/srcPLgrado/grunt-simple-example(master)]$ cat Gruntfile.js
module.exports = function(grunt){
  grunt.initConfig({
    uglify: {
      main: {
        files: {
          'assets/app.min.js': [
            'src/javascripts/jquery-1.10.2.min.js',
            'src/javascripts/bootstrap.js',
            'src/javascripts/application.js'
          ]
        }
      }
    },
    less: {
      application: {
        options: {
          yuicompress: true
        },
        files: {
          "assets/app.min.css": "src/stylesheets/application.less"
        }
      }
    },
    watch: {
      javascripts: {
        files: ['src/javascripts/**/*'],
        tasks: ['uglify']
      },
      stylesheets: {
        files: ['src/stylesheets/**/*'],
        tasks: ['less']
      }
    }
  });

  // Load plugins
  grunt.loadNpmTasks('grunt-contrib-less');
  grunt.loadNpmTasks('grunt-contrib-uglify');
  grunt.loadNpmTasks('grunt-contrib-watch');

  // Register tasks
  grunt.registerTask('default', ['uglify', 'less']);
}

[~/srcPLgrado/grunt-simple-example(master)]$ cat package.json
{
  "name": "grunt-simple-example",
  "version": "0.0.1",
  "main": "index.js",
  "devDependencies": {
    "grunt": "~0.4.1",
    "grunt-contrib-cssmin": "~0.6.2",
    "grunt-contrib-less": "~0.7.0",
```

```
    "grunt-contrib-uglify": "~0.2.4",
    "grunt-contrib-watch": "~0.5.3"
  },
  "author": "Bryan Shelton",
  "license": "BSD-2-Clause"
}
```

```
[~/srcPLgrado/grunt-simple-example(master)]$ npm install
npm WARN package.json grunt-simple-example@0.0.1 No repository field.
[~/srcPLgrado/grunt-simple-example(master)]$
```

```
[~/srcPLgrado/grunt-simple-example(master)]$ grunt watch
Running "watch" task
Waiting...OK
>> File "src/javascripts/application.js" changed.

Running "uglify:main" (uglify) task
File "assets/app.min.js" created.

Done, without errors.
Completed in 3.897s at Mon Jan 20 2014 19:02:03 GMT+0000 (WET) - Waiting...
```

13.4.4. GitHub Project Pages

Project Pages are kept in the same repository as the project they are for.
These pages are similar to User and Org Pages, with a few slight differences:

- The `gh-pages` branch is used to build and publish from.
- A custom domain on user/org pages will apply the same domain redirect to all project pages hosted under that account, unless the project pages use their own custom domain.
- If no custom domain is used, the project pages are served under a subpath of the user pages:

  ```
  username.github.io/projectname
  ```

  Por ejemplo, mi usuario es `crguezl`. Si el proyecto se llama `nathanuniversityexercisesPL`, la dirección será:

  http://crguezl.github.io/nathanuniversityexercisesPL/

- Custom 404s will only work if a custom domain is used, otherwise the User Pages 404 is used.
- Creating Project Pages manually

1. Setting up Pages on a project requires a new .ºrphan"branch in your repository. The safest way to do this is to start with a fresh clone.

   ```
   git clone https://github.com/user/repository.git
   # Clone our repository
   # Cloning into 'repository'...
   remote: Counting objects: 2791, done.
   remote: Compressing objects: 100% (1225/1225), done.
   remote: Total 2791 (delta 1722), reused 2513 (delta 1493)
   Receiving objects: 100% (2791/2791), 3.77 MiB | 969 KiB/s, done.
   Resolving deltas: 100% (1722/1722), done.
   ```

2. Now that we have a clean repository, we need to create the new branch and remove all content from the working directory and index.

```
cd repository

git checkout --orphan gh-pages
# Creates our branch, without any parents (it's an orphan!)
# Switched to a new branch 'gh-pages'

git rm -rf .
# Remove all files from the old working tree
# rm '.gitignore'
```

3. Now we have an empty working directory. We can create some content in this branch and push it to GitHub. For example:

```
echo "My GitHub Page" > index.html
git add index.html
git commit -a -m "First pages commit"
git push origin gh-pages
```

Capítulo 14

Functions and all that

http://nathansuniversity.com/funcs.html

Capítulo 15

Inventing a language for turtle graphics

http://nathansuniversity.com/turtle.html

Parte III

PARTE: SINATRA

Capítulo 16

Rack, un Webserver Ruby Modular

16.1. Introducción

Que es Rack rack provides an minimal interface between webservers supporting Ruby and Ruby frameworks.

Ruby on Rails, Ramaze, Sinatra and other Ruby frameworks use it by default to talk to web servers, including Mongrel, Thin or Apache via Passenger.

Lo que hace Rack es que unifica la API de los diferentes web servers envolviendo las peticiones y respuestas HTTP en la forma mas simple posible.

1. Rack includes *handlers* that connect Rack to all these web application servers (WEBrick, Mongrel etc.).
2. Rack includes *adapters* that connect Rack to various web frameworks (Sinatra, Rails etc.).
3. Between the server and the framework, Rack can be customized to your applications needs using *middleware*.

The fundamental idea behind *Rack middleware* is – come between the calling client and the server, process the HTTP request before sending it to the server, and processing the HTTP response before returning it to the client.

Que es una Aplicación Rack Una aplicación Rack es un objeto que

1. Debe responder al método `call`.
2. El método `call` será llamado por el servidor y se le pasa como argumento `env` que es un hash que contiene información sobre el entorno CGI.
3. El método `call` debe retornar un array con tres elementos:
   a) `status`: un entero
   b) `headers`: un hash
   c) `body`: un objeto que responde al método `each` y que para cada llamada de `each` retorna una `String`.

Un Ejemplo Sencillo

Rack uses a configuration file with extension `.ru`, that instructs Rack::Builder what middleware should it use and in which order. Let's create one:

```
[~/sinatra/rackup/simple(master)]$ cat myapp.rb
# my_app.rb
#
```

```
class MyApp
  def call env
    [200, {"Content-Type" => "text/html"}, ["Hello Rack Participants"]]
  end
end
```

Esta es la aplicación Rack mas simple posible.

```
[~/sinatra/rackup/simple(master)]$ cat config.ru
require './myapp'
run MyApp.new
```

To start your newly created app, you need to use `rackup` command:

```
$ rackup config.ru
```

The application will be available by default on port 9292, so you have to visit `http://localhost:9292` to see it.

Un Ejemplo con la Consola Interactiva de Ruby

Arranquemos la consola interactiva de Ruby. Cargamos `rack`:

```
1] pry(main)> require 'rack'
=> true
```

Comprobemos que handlers tenemos instalados:

```
[2] pry(main)> Rack::Handler::constants
=> [:CGI,
 :FastCGI,
 :Mongrel,
 :EventedMongrel,
 :SwiftipliedMongrel,
 :WEBrick,
 :LSWS,
 :SCGI,
 :Thin]
```

Todos los handlers Rack tienen un método `run` por lo que podemos llamar el método `run` en cualquiera de esos handlers instalados.

Un objeto que tiene un método `call` es cualquier objeto `Proc` y por tanto podemos usar una lambda que se atenga al protocolo de rack para nuestro ejemplo:

```
[3] pry(main)> Rack::Handler::WEBrick.run lambda
                  { |env| [200,
                          {"Content-Type" => "text/plain"},
                         ["Hello. The time is #{Time.now}"]] }
[2013-09-11 17:59:07] INFO  WEBrick 1.3.1
[2013-09-11 17:59:07] INFO  ruby 1.9.3 (2013-02-22) [x86_64-darwin11.4.2]
[2013-09-11 17:59:07] INFO  WEBrick::HTTPServer#start: pid=25123 port=8080
localhost - - [11/Sep/2013:17:59:26 WEST] "GET / HTTP/1.1" 200 44
- -> /
localhost - - [11/Sep/2013:17:59:26 WEST] "GET /favicon.ico HTTP/1.1" 200 44
- -> /favicon.ico
```

El primer argumento de `run` es nuestra aplicación Rack

```
lambda { |env| [200, {"Content-Type" => "text/plain"}, ["Hello. The time is #{Time.now}"]] }
```

y el segundo es el conjunto de opciones para nuestro programa.

ahora podemos visitar la aplicación con nuestro navegador en `http://localhost:8080`

16.2. Analizando env con pry-debugger

16.2.1. Introducción

Tenemos esta sencilla aplicación:

```
~/local/src/ruby/sinatra/rack/rack-debugging]$ cat hello.rb
require 'rack'
require 'pry-debugger'

class HelloWorld
  def call env
    binding.pry
    [200, {"Content-Type" => "text/plain"}, ["Hello world"]]
  end
end
```

Arrancamos un servidor:

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ pry
[1] pry(main)> require './hello'
=> true
[6] pry(main)> Rack::Handler::WEBrick.run HelloWorld.new
[2013-09-23 12:36:21] INFO  WEBrick 1.3.1
[2013-09-23 12:36:21] INFO  ruby 1.9.3 (2013-02-22) [x86_64-darwin11.4.2]
[2013-09-23 12:36:21] INFO  WEBrick::HTTPServer#start: pid=9458 port=8080
```

En otra ventana arrancamos un cliente:

```
[~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ curl -v localhost:8080
* About to connect() to localhost port 8080 (#0)
*   Trying ::1... connected
* Connected to localhost (::1) port 8080 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:8080
> Accept: */*
>
```

En la ventana del servidor ahora aparece:

```
From: /Users/casiano/local/src/ruby/sinatra/rack/rack-debugging/hello.rb @ line 6 HelloWorld#c

    5: def call env
 => 6:   binding.pry
    7:   [200, {"Content-Type" => "text/plain"}, ["Hello world"]]
    8: end
```

Ahora podemos inspeccionar las variables:

```
[1] pry(#<HelloWorld>)> env
=> {"GATEWAY_INTERFACE"=>"CGI/1.1",
 "PATH_INFO"=>"/",
 "QUERY_STRING"=>"",
 "REMOTE_ADDR"=>"::1",
 "REMOTE_HOST"=>"localhost",
```

```
 "REQUEST_METHOD"=>"GET",
 "REQUEST_URI"=>"http://localhost:8080/",
 "SCRIPT_NAME"=>"",
 "SERVER_NAME"=>"localhost",
 "SERVER_PORT"=>"8080",
 "SERVER_PROTOCOL"=>"HTTP/1.1",
 "SERVER_SOFTWARE"=>"WEBrick/1.3.1 (Ruby/1.9.3/2013-02-22)",
 "HTTP_USER_AGENT"=>
  "curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.5",
 "HTTP_HOST"=>"localhost:8080",
 "HTTP_ACCEPT"=>"*/*",
 "rack.version"=>[1, 2],
 "rack.input"=>#<StringIO:0x007fbba40263b0>,
 "rack.errors"=>#<IO:<STDERR>>,
 "rack.multithread"=>true,
 "rack.multiprocess"=>false,
 "rack.run_once"=>false,
 "rack.url_scheme"=>"http",
 "HTTP_VERSION"=>"HTTP/1.1",
 "REQUEST_PATH"=>"/"}
[2] pry(#<HelloWorld>)>
```

Hay tres categorías de variables en *env*:

1. Variables CGI
2. Variables específicas de Rack (empiezan por `rack.`)
3. Un tercer tipo de variables son las de la aplicación y/o el servidor. En este ejemplo no aparecen

Véase la especificación Rack.

Le indicamos al servidor que continue:

```
[2] pry(#<HelloWorld>)> co<TABULADOR>
cohen-poem  continue
[2] pry(#<HelloWorld>)> continue
localhost - - [23/Sep/2013:12:36:48 WEST] "GET / HTTP/1.1" 200 11
- -> /
```

después de entregar la respuesta el servidor cierra la conexión HTTP. Esto es así porque HTTP es un protocolo sin estado, esto es, no se mantiene información de la conexión entre transacciones. En la ventana del cliente obtenemos la siguiente salida:

```
[~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ curl -v localhost:8080
* About to connect() to localhost port 8080 (#0)
*   Trying ::1... connected
* Connected to localhost (::1) port 8080 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:8080
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/plain
< Server: WEBrick/1.3.1 (Ruby/1.9.3/2013-02-22)
< Date: Mon, 23 Sep 2013 11:45:00 GMT
```

```
< Content-Length: 11
< Connection: Keep-Alive
<
* Connection #0 to host localhost left intact
* Closing connection #0
Hello world
```

16.2.2. REQUEST_METHOD, QUERY_STRING y PATH_INFO

```
[~/local/src/ruby/sinatra/rack/rack-env]$ cat app.rb
require 'rack'
require 'thin'

cgi_inspector = lambda do |env|
  [200, #status
    { 'Content-Type' => 'text/html' }, #headers
    ["<h1>
        Your request:<br>
       <ul>
         <li>http method is: #{env['REQUEST_METHOD']}
         <li>path is: #{env['PATH_INFO']}
         <li>Query string is: #{env['QUERY_STRING']}
       </ul>
      </h1>
     "
    ]
  ]
end

Rack::Handler::Thin.run cgi_inspector, :Port => 3000
```

Visite la página `localhost:3000/camino?var=4`.
Esta es la salida:

```
[~/local/src/ruby/sinatra/rack/rack-env]$ curl -v localhost:3000/camino?var=4
* About to connect() to localhost port 3000 (#0)
*   Trying ::1... Connection refused
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 3000 (#0)
> GET /camino?var=4 HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:3000
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Connection: close
< Server: thin 1.5.1 codename Straight Razor
<
<h1>
        Your request:<br>
       <ul>
         <li>http method is: GET
         <li>path is: /camino
```

```
        <li>Query string is: var=4
      </ul>
     </h1>
* Closing connection #0
```

16.3. Detectando el Proceso que está Usando un Puerto

Si intentamos ejecutar una segunda instancia del servidor mientras otra instancia esta ejecutandose obtenemos un error que indica que el puerto está en uso:

```
[~/sinatra/sinatra-simple(master)]$ rackup
Thin web server (v1.6.1 codename Death Proof)
Maximum connections set to 1024
Listening on 0.0.0.0:9292, CTRL+C to stop
/Users/casiano/.rvm/gems/ruby-2.0.0-p247/gems/eventmachine-1.0.3/lib/eventmachine.rb:526:
  in `start_tcp_server': no acceptor
  (port is in use or requires root privileges) (RuntimeError)
  from /Users/casiano/.rvm/gems/ruby-2.0.0-p247/gems/eventmachine-1.0.3/lib/eventmachine.rb:52
    in `start_server'
...
```

Si sabemos en que puerto esta corriendo - como es el caso - podemos hacer algo así para saber el PID del proceso que lo ocupa:

```
[~/sinatra/sinatra-simple(master)]$ lsof -i :9292
COMMAND   PID    USER   FD   TYPE             DEVICE SIZE/OFF NODE NAME
ruby    52870 casiano    9u  IPv4 0x9f3ffc595152af29      0t0  TCP *:armtechdaemon (LISTEN)
```

Si no lo sabemos podemos hacer:

```
[~/sinatra/sinatra-simple(master)]$ ps -fA | egrep ruby
  501 52870   565   0 11:16AM ttys003    0:00.61 ruby /Users/casiano/.rvm/gems/ruby-2.0.0-p247
  501 53230 52950   0 11:35AM ttys006    0:00.00 egrep ruby
```

Si tenemos privilegios suficientes podemos ahora eliminar el proceso:

```
[~/sinatra/sinatra-simple(master)]$ kill -9 52870

[~/sinatra/sinatra-simple(master)]$ rackup
Thin web server (v1.6.1 codename Death Proof)
Maximum connections set to 1024
Listening on 0.0.0.0:9292, CTRL+C to stop
Killed: 9
```

El comando

```
$ lsof -i | egrep -i 'tcp.*(\d+.)+'
```

Nos da una lista bastante completa de como están nuestras conexiones.

1. `-i [i]` selects the listing of files any of whose Internet address matches the address specified in i. If no address is specified, this option selects the listing of all Internet and x.25 (HP-UX) network files.

16.4. Usando PATH_INFO y erubis para construir una aplicación (Noah Gibbs)

config.ru

```
[~/local/src/ruby/sinatra/rack/hangout-framework(master)]$ cat config.ru
require "erubis"

use Rack::ContentType

def output(text, options = {})
  [ options[:status] || 200,
    {}, [ text ].flatten ]
end

def from_erb(file, vars = {})
  eruby = Erubis::Eruby.new File.read(file)
  output eruby.result vars
end

run proc { |env|
  path = env['PATH_INFO']
  if path =~ %r{^/foo}
    from_erb "template.html.erb"
  else
    output "Not found!", :status => 400
  end
}
```

Template erb

```
[~/local/src/ruby/sinatra/rack/hangout-framework(master)]$ cat template.html.erb
<p> A template! </p>
<% 10.times do -%> <p> Pretty cool! </p> <% end -%>
```

Arrancando el Servidor

```
[~/local/src/ruby/sinatra/rack/hangout-framework(master)]$ rackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop}
```

Ejecutando un cliente

```
[~/local/src/ruby/sinatra/rack/hangout-framework(master)]$ curl -v http://localhost:9292/fooch
* About to connect() to localhost port 9292 (#0)
*   Trying ::1... Connection refused
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /foochazam HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
```

```
< HTTP/1.1 200 OK
< Content-Type: text/html
< Transfer-Encoding: chunked
< Connection: close
< Server: thin 1.5.1 codename Straight Razor
<
<p> A template! </p>
 <p> Pretty cool! </p>  <p> Pretty cool! </p>  <p> Pretty cool! </p>
 <p> Pretty cool! </p>  <p> Pretty cool! </p>  <p> Pretty cool! </p>
 <p> Pretty cool! </p>  <p> Pretty cool! </p>  <p> Pretty cool! </p>
 <p> Pretty cool! </p>
* Closing connection #0
```

Logs del servidor

```
[~/local/src/ruby/sinatra/rack/hangout-framework(master)]$ rackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
 127.0.0.1 - - [20/Oct/2013 12:22:37] "GET /foochazam HTTP/1.1" 200 - 0.0014
```

Véase

1. erubis
2. Noah Gibbs Demo Rack framework for March 6th, 2013 Ruby Hangout.
3. Ruby Hangout 3-13 Noah Gibbs

16.5. HTTP

16.5.1. Introducción

1. HTTP es un protocolo sin estado: que no guarda ninguna información sobre conexiones anteriores.
2. El desarrollo de aplicaciones web necesita frecuentemente mantener estado.
3. Para esto se usan las cookies, que es información que un servidor puede almacenar en el sistema cliente.
4. Esto le permite a las aplicaciones web introducir la noción de *sesión*, y también permite rastrear usuarios ya que las cookies pueden guardarse en el cliente por tiempo indeterminado.
5. Una transacción HTTP está formada por un *encabezado* seguido, opcionalmente, por una línea en blanco y algún dato.
6. El encabezado especificará cosas como la acción requerida del servidor, o el tipo de dato retornado, o el código de estado.
7. El uso de campos de encabezados enviados en las transacciones HTTP le da flexibilidad al protocolo. Estos campos permiten que se envíe información descriptiva en la transacción, permitiendo así la autenticación, cifrado e identificación de usuario.
8. Un encabezado es un bloque de datos que precede a la información propiamente dicha, por lo que a veces se hace referencia a él como metadato, porque tiene datos sobre los datos.

9. Si se reciben líneas de encabezado del cliente, el servidor las coloca en las variables de entorno de CGI con el prefijo `HTTP_` seguido del nombre del encabezado. Cualquier carácter guion (`-`) del nombre del encabezado se convierte a caracteres `"_"`.

   Ejemplos de estos encabezados del cliente son `HTTP_ACCEPT` y `HTTP_USER_AGENT`.

   a) `HTTP_ACCEPT`. Los tipos MIME que el cliente aceptará, dados los encabezados HTTP. Los elementos de esta lista deben estar separados por comas

   b) `HTTP_USER_AGENT`. El navegador que utiliza el cliente para realizar la petición. El formato general para esta variable es: software/versión biblioteca/versión.

   El servidor envía al cliente:

   a) Un *código de estado* que indica si la petición fue correcta o no. Los códigos de error típicos indican que el archivo solicitado no se encontró, que la petición no se realizó de forma correcta o que se requiere autenticación para acceder al archivo.

   b) La información propiamente dicha. HTTP permite enviar documentos de todo tipo y formato, como gráficos, audio y video.

   c) Información sobre el objeto que se retorna.

16.5.2. Sesiones HTTP

1. Una sesión HTTP es una secuencia de transacciones de red de peticiones y respuestas

2. Un cliente HTTP inicia una petición estableciendo una conexión TCP con un puerto particular de un servidor (normalmente el puerto 80)

3. Un servidor que esté escuchando en ese puerto espera por un mensaje de petición de un cliente.

4. El servidor retorna la *línea de estatus*, por ejemplo `HTTP/1.1 200 OK`, y su propio mensaje. El cuerpo de este mensaje suele ser el recurso solicitado, aunque puede que se trate de un mensaje de error u otro tipo de información.

Veamos un ejemplo. Usemos este servidor:

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ cat hello1.rb
require 'rack'

class HelloWorld
  def call env
    [200, {"Content-Type" => "text/plain"}, ["Hello world"]]
  end
end

Rack::Handler::WEBrick::run HelloWorld.new

[~/local/src/ruby/sinatra/rack/rack-debugging]$ ruby hello1.rb
[2013-09-23 15:16:58] INFO  WEBrick 1.3.1
[2013-09-23 15:16:58] INFO  ruby 1.9.3 (2013-02-22) [x86_64-darwin11.4.2]
[2013-09-23 15:16:58] INFO  WEBrick::HTTPServer#start: pid=12113 port=8080
```

Arrancamos un cliente con telnet con la salida redirigida:

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ telnet localhost 8080 > salida
```

Escribimos esto en la entrada estandard:

```
GET /index.html HTTP/1.1
Host: localhost
Connection: close


```

con una línea en blanco al final. Este texto es enviado al servidor.

El cliente deja su salida en el fichero `salida`:

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ cat salida
Trying ::1...
Connected to localhost.
Escape character is '^]'.
HTTP/1.1 200 OK
Content-Type: text/plain
Server: WEBrick/1.3.1 (Ruby/1.9.3/2013-02-22)
Date: Mon, 23 Sep 2013 14:33:16 GMT
Content-Length: 11
Connection: close

Hello world
```

El cliente escribe en la salida estandard:

```
Connection closed by foreign host.
```

16.5.3. Métodos de Petición

1. *GET*

   Solicita una representación de un recurso especificado. Las peticiones que usen *GET* deberían limitarse a obtener los datos y no tener ningún otro efecto.

2. *HEAD*

   Pregunta por la misma respuesta que una petición *GET* pero sin el cuerpo de la respuesta

3. *POST*

   Requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, as examples,

   a) an annotation for existing resources;

   b) a message for a bulletin board, newsgroup, mailing list, or comment thread;

   c) a block of data that is the result of submitting a web form to a data-handling process;

   d) or an item to add to a database.

4. *PUT*

   Requests that the enclosed entity be stored under the supplied URI. If the URI refers to an already existing resource, it is modified; if the URI does not point to an existing resource, then the server can create the resource with that URI.

5. *DELETE*

   Deletes the specified resource.

6. *TRACE*

   Echoes back the received request so that a client can see what (if any) changes or additions have been made by intermediate servers.

7. *OPTIONS*

   Returns the HTTP methods that the server supports for the specified URL. This can be used to check the functionality of a web server by requesting * instead of a specific resource.

8. *CONNECT*

   Converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication (HTTPS) through an unencrypted HTTP proxy.

9. *PATCH*

   Is used to apply partial modifications to a resource. HTTP servers are required to implement at least the GET and HEAD methods and, whenever possible, also the OPTIONS method

16.5.4. Véase

1. ArrrrCamp #6 - Konstantin Haase - We don't know HTTP
2. Resources, For Real This Time (with Webmachine) Sean Cribbs Ruby Conference 2011

16.6. Rack::Request y Depuración con `pry-debugger`

16.6.1. Conexión sin Parámetros

Partimos del mismo código fuente que en la sección anterior:

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ cat hello.rb
require 'rack'
require 'pry-debugger'

class HelloWorld
  def call env
    binding.pry
    [200, {"Content-Type" => "text/plain"}, ["Hello world"]]
  end
end
```

Arranquemos un servidor dentro de `pry`:

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ pry
[1] pry(main)> require './hello'
=> true
[2] pry(main)> Rack::Handler::WEBrick::run HelloWorld.new
[2013-09-23 13:10:42] INFO  WEBrick 1.3.1
[2013-09-23 13:10:42] INFO  ruby 1.9.3 (2013-02-22) [x86_64-darwin11.4.2]
[2013-09-23 13:10:42] INFO  WEBrick::HTTPServer#start: pid=10395 port=8080
```

Si visitamos la página:

```
$ curl -v localhost:8080/jkdfkdjg
```

Esto hace que se alcance el break:

```
From: /Users/casiano/local/src/ruby/sinatra/rack/rack-debugging/hello.rb @ line 6 HelloWorld#c

    5: def call env
 => 6:   binding.pry
    7:   [200, {"Content-Type" => "text/plain"}, ["Hello world"]]
    8: end
```

Ahora creamos un objeto Rack::Request:

```
[3] pry(#<HelloWorld>)> req = Rack::Request.new(env)
=> #<Rack::Request:0x007fbba4ff3298
 @env=
  {"GATEWAY_INTERFACE"=>"CGI/1.1",
   "PATH_INFO"=>"/jkdfkdjg",
   "QUERY_STRING"=>"",
   "REMOTE_ADDR"=>"::1",
   "REMOTE_HOST"=>"localhost",
   "REQUEST_METHOD"=>"GET",
   "REQUEST_URI"=>"http://localhost:8080/jkdfkdjg",
   "SCRIPT_NAME"=>"",
   "SERVER_NAME"=>"localhost",
   "SERVER_PORT"=>"8080",
   "SERVER_PROTOCOL"=>"HTTP/1.1",
   "SERVER_SOFTWARE"=>"WEBrick/1.3.1 (Ruby/1.9.3/2013-02-22)",
   "HTTP_USER_AGENT"=>
    "curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.5",
   "HTTP_HOST"=>"localhost:8080",
   "HTTP_ACCEPT"=>"*/*",
   "rack.version"=>[1, 2],
   "rack.input"=>#<StringIO:0x007fbba4e74980>,
   "rack.errors"=>#<IO:<STDERR>>,
   "rack.multithread"=>true,
   "rack.multiprocess"=>false,
   "rack.run_once"=>false,
   "rack.url_scheme"=>"http",
   "HTTP_VERSION"=>"HTTP/1.1",
   "REQUEST_PATH"=>"/jkdfkdjg"}>
```

Este objeto Rack::Request tiene métodos para informarnos del Rack::Request:

```
[4] pry(#<HelloWorld>)> req.get?
=> true
[5] pry(#<HelloWorld>)> req.post?
=> false
[7] pry(#<HelloWorld>)> req.port
=> 8080
[12] pry(#<HelloWorld>)> req.host()
=> "localhost"
[13] pry(#<HelloWorld>)> req.host_with_port()
=> "localhost:8080"
[15] pry(#<HelloWorld>)> req.path()
=> "/jkdfkdjg"
[18] pry(#<HelloWorld>)> req.url()
=> "http://localhost:8080/jkdfkdjg"
[19] pry(#<HelloWorld>)> req.user_agent
=> "curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.5"
```

16.6.2. Conexión con Parámetros

Partimos del mismo código fuente que en la sección anterior:

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ cat hello.rb
```

```
require 'rack'
require 'pry-debugger'

class HelloWorld
  def call env
    binding.pry
    [200, {"Content-Type" => "text/plain"}, ["Hello world"]]
  end
end
```

Arrancamos el servidor:

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ pry
[1] pry(main)> require './hello'
=> true
[2] pry(main)> Rack::Handler::WEBrick::run HelloWorld.new
[2013-09-23 13:10:42] INFO  WEBrick 1.3.1
[2013-09-23 13:10:42] INFO  ruby 1.9.3 (2013-02-22) [x86_64-darwin11.4.2]
[2013-09-23 13:10:42] INFO  WEBrick::HTTPServer#start: pid=10395 port=8080
```

En el cliente tendríamos:

```
$ curl -v 'localhost:8080?a=1&b=2&c=3'
```

comienza produciendo esta salida:

```
* About to connect() to localhost port 8080 (#0)
*   Trying ::1... connected
* Connected to localhost (::1) port 8080 (#0)
> GET /?a=1&b=2&c=3 HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:8080
> Accept: */*
>
```

En la ventana del servidor se produce el break:

```
From: /Users/casiano/local/src/ruby/sinatra/rack/rack-debugging/hello.rb @ line 6 HelloWorld#c

    5: def call env
 => 6:   binding.pry
    7:   [200, {"Content-Type" => "text/plain"}, ["Hello world"]]
    8: end
```

Rack::Request.new

Creamos un objeto Rack::Request:

```
[1] pry(#<HelloWorld>)> req = Rack::Request.new env
=> #<Rack::Request:0x007fafd27946c0
 @env=
  {"GATEWAY_INTERFACE"=>"CGI/1.1",
   "PATH_INFO"=>"/",
   "QUERY_STRING"=>"a=1&b=2&c=3",
   "REMOTE_ADDR"=>"::1",
   "REMOTE_HOST"=>"localhost",
   "REQUEST_METHOD"=>"GET",
```

```
  "REQUEST_URI"=>"http://localhost:8080/?a=1&b=2&c=3",
  "SCRIPT_NAME"=>"",
  "SERVER_NAME"=>"localhost",
  "SERVER_PORT"=>"8080",
  "SERVER_PROTOCOL"=>"HTTP/1.1",
  "SERVER_SOFTWARE"=>"WEBrick/1.3.1 (Ruby/1.9.3/2013-02-22)",
  "HTTP_USER_AGENT"=>
   "curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.5",
  "HTTP_HOST"=>"localhost:8080",
  "HTTP_ACCEPT"=>"*/*",
  "rack.version"=>[1, 2],
  "rack.input"=>#<StringIO:0x007fafd26bbbe0>,
  "rack.errors"=>#<IO:<STDERR>>,
  "rack.multithread"=>true,
  "rack.multiprocess"=>false,
  "rack.run_once"=>false,
  "rack.url_scheme"=>"http",
  "HTTP_VERSION"=>"HTTP/1.1",
  "REQUEST_PATH"=>"/"}>
```

req.params Ahora podemos interrogarle:

```
[2] pry(#<HelloWorld>)> req.params
=> {"a"=>"1", "b"=>"2", "c"=>"3"}
```

Indexación de los objetos Rack::Request Recordemos que la URL visitada fué: `localhost:8080?a=1&b=2&`

```
[3] pry(#<HelloWorld>)> req["a"]
=> "1"
[4] pry(#<HelloWorld>)> req["b"]
=> "2"
[5] pry(#<HelloWorld>)> req["c"]
=> "3"
```

req.path

```
[6] pry(#<HelloWorld>)> req.path
=> "/"
[7] pry(#<HelloWorld>)> req.fullpath
=> "/?a=1&b=2&c=3"
[9] pry(#<HelloWorld>)> req.path_info
=> "/"
[10] pry(#<HelloWorld>)> req.query_string
=> "a=1&b=2&c=3"
```

req.url

```
[11] pry(#<HelloWorld>)> req.url
=> "http://localhost:8080/?a=1&b=2&c=3"
```

req.values

```
[12] pry(#<HelloWorld>)> req.values_at("a")
=> ["1"]
```

```
[13] pry(#<HelloWorld>)> req.values_at("a", "b")
=> ["1", "2"]
[14] pry(#<HelloWorld>)> req.values_at("a", "b", "c")
=> ["1", "2", "3"]

[16] pry(#<HelloWorld>)> continue
localhost - - [23/Sep/2013:13:10:49 WEST] "GET /?a=1&b=2&c=3 HTTP/1.1" 200 11
- -> /?a=1&b=2&c=3

$ curl -v 'localhost:8080?a=1&b=2&c=3'
* About to connect() to localhost port 8080 (#0)
*   Trying ::1... connected
* Connected to localhost (::1) port 8080 (#0)
> GET /?a=1&b=2&c=3 HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:8080
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/plain
< Server: WEBrick/1.3.1 (Ruby/1.9.3/2013-02-22)
< Date: Mon, 23 Sep 2013 12:35:37 GMT
< Content-Length: 11
< Connection: Keep-Alive
<
* Connection #0 to host localhost left intact
* Closing connection #0
Hello world
```

16.7. Rack::Response

16.7.1. Introducción

Rack::Response provides a convenient interface to create a Rack response.

It allows setting of headers and cookies, and provides useful defaults (a `OK` response containing HTML).

You can use `Response#write` to iteratively generate your response, but note that this is buffered by Rack::Response until you call `finish`.

Alternatively, the method `finish` can take a block inside which calls to write are synchronous with the Rack response.

Your application's call should end returning `Response#finish`.

16.7.2. Ejemplo Simple

```
[~/local/src/ruby/sinatra/rack/rack-debugging(master)]$ cat body_bytesize.rb
require 'rack'
require 'thin'

app = lambda do |env|
  req = Rack::Request.new env
  res = Rack::Response.new

  body = "--------------- Header ------------------\n"
```

```
  if req.path_info == '/hello'
    body << "hi "
    name = req['name']
    body << name if name
    body << "\n"
  else
    body << "Instead of #{req.url} visit something like "+
            "http://localhost:8080/hello?name=Casiano\n"
  end
  res['Content-Type'] = 'text/plain'
  res["Content-Length"] = body.bytesize.to_s
  #res["Content-Length"] = Rack::Utils.bytesize(body).to_s
  res.body = [ body ]
  res.finish
end

Rack::Handler::Thin.run app
```

16.7.3. Ejemplo con POST

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ cat hello_response.rb
# encoding: utf-8
require 'rack'
require 'pry-debugger'

class HelloWorld

  def call env
    req = Rack::Request.new(env)
    res = Rack::Response.new
    binding.pry if ARGV[0]
    res['Content-Type'] = 'text/html'
    name = (req["firstname"] && req["firstname"] != '') ? req["firstname"] :'World'
    res.write <<-"EOS"
      <!DOCTYPE HTML>
      <html>
        <title>Rack::Response</title>
        <body>
          <h1>
             Hello #{name}!
             <form action="/" method="post">
               Your name: <input type="text" name="firstname" autofocus><br>
               <input type="submit" value="Submit">
             </form>
          </h1>
        </body>
      </html>
    EOS
    res.finish
  end
end

Rack::Server.start(
  :app => HelloWorld.new,
```

```
  :Port => 9292,
  :server => 'thin'
)

[~/local/src/ruby/sinatra/rack/rack-debugging]$ ruby hello_response.rb debug
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
```

Ahora cuando visitamos la página `http://localhost:9292` el navegador queda a la espera del servidor y el servidor alcanza la línea de break.

```
From: /Users/casiano/local/src/ruby/sinatra/rack/rack-debugging/hello_response.rb @ line 10 He

      7: def call env
      8:   req = Rack::Request.new(env)
      9:   res = Rack::Response.new
 => 10:   binding.pry if ARGV[0]
     11:   res['Content-Type'] = 'text/html'
     12:   name = (req["firstname"] && req["firstname"] != '') ? req["firstname"] :'World'
     13:   res.write <<-"EOS"
     14:     <!DOCTYPE HTML>
     15:     <html>
     16:       <title>Rack::Response</title>
     17:       <body>
     18:         <h1>
     19:            Hello #{name}!
     20:            <form action="/" method="post">
     21:              Your name: <input type="text" name="firstname" autofocus><br>
     22:              <input type="submit" value="Submit">
     23:            </form>
     24:         </h1>
     25:       </body>
     26:     </html>
     27:   EOS
     28:   res.finish
     29: end

[1] pry(#<HelloWorld>)>
```

Consultemos los contenidos de `res`:

```
[1] pry(#<HelloWorld>)> res
=> #<Rack::Response:0x007fe3fb1e6180
 @block=nil,
 @body=[],
 @chunked=false,
 @header={},
 @length=0,
 @status=200,
 @writer=
  #<Proc:0x007fe3fb1e5f50@/Users/casiano/.rvm/gems/ruby-1.9.3-p392/gems/rack-1.5.2/lib/rack/re
```

Después de un par de `continue` el servidor se queda a la espera:

```
[3] pry(#<HelloWorld>)> continue
...
[1] pry(#<HelloWorld>)> continue
```

Rellenamos la entrada con un nombre (Pedro) y de nuevo el servidor alcanza el punto de ruptura:

```
[2] pry(#<HelloWorld>)> req.params
=> {"firstname"=>"Pedro"}

[7] pry(#<HelloWorld>)> break 28
Breakpoint 1: /Users/casiano/local/src/ruby/sinatra/rack/rack-debugging/hello_response.rb @ li

    26:       </html>
    27:     EOS
 => 28:     res.finish
    29:   end

[8] pry(#<HelloWorld>)> continue
Breakpoint 1. First hit.
...
[9] pry(#<HelloWorld>)> res.headers
=> {"Content-Type"=>"text/html", "Content-Length"=>"370"}
[10] pry(#<HelloWorld>)>
```

16.8. Cookies y Rack

Cookies may be used to maintain data related to the user during navigation, possibly across multiple visits.

Introducción

1. A *cookie*, is a small piece of data sent from a website and stored in a user's web browser while the user is browsing that website.

2. Every time the user loads the website, the browser sends the cookie back to the server to notify the website of the user's previous activity

3. Cookies were designed to be a reliable mechanism for websites to remember stateful information (such as items in a shopping cart) or to record the user's browsing activity (including clicking particular buttons, logging in, or recording which pages were visited by the user as far back as months or years ago).

4. A user's *session cookie* (also known as an in-memory cookie or transient cookie) for a website exists in temporary memory only while the user is reading and navigating the website.

5. When an expiry date or validity interval is not set at cookie creation time, a session cookie is created. Web browsers normally delete session cookies when the user closes the browser

6. A *persistent cookie* will outlast user sessions. If a persistent cookie has its `Max-Age` set to 1 year, then, during that year, the initial value set in that cookie would be sent back to the server every time the user visited the server. This could be used to record information such as how the user initially came to this website. For this reason, persistent cookies are also called *tracking cookies*

7. A *secure cookie* has the *secure attribute* enabled and is only used via HTTPS, ensuring that the cookie is always encrypted when transmitting from client to server. This makes the cookie less likely to be exposed to cookie theft via eavesdropping.

8. *First-party cookies* are cookies that belong to the same domain that is shown in the browser's address bar (or that belong to the sub domain of the domain in the address bar).

9. *Third-party cookies* are cookies that belong to domains different from the one shown in the address bar.

   a) Web pages can feature content from third-party domains (such as banner adverts), which opens up the potential for tracking the user's browsing history.

   b) Privacy setting options in most modern browsers allow the blocking of third-party tracking cookies.

   c) As an example, suppose a user visits `www.example1.com`.

   d) This web site contains an advert from `ad.foxytracking.com`, which, when downloaded, sets a cookie belonging to the advert's domain (`ad.foxytracking.com`).

   e) Then, the user visits another website, `www.example2.com`, which also contains an advert from `ad.foxytracking.com`, and which also sets a cookie belonging to that domain (`ad.foxytracking.com`).

   f) Eventually, both of these cookies will be sent to the advertiser when loading their ads or visiting their website.

   g) The advertiser can then use these cookies to build up a browsing history of the user across all the websites that have ads from this advertiser.

Propiedades de un cookie

Un cookie tiene los siguientes atributos:

1. nombre

2. valor

3. domain (dominio)

4. path o camino

5. secure / seguridad

Cuando ejecutamos este programa:

```
[~/local/src/ruby/sinatra/rack/rack-session-cookie(master)]$ cat study_cookie1.ru
run lambda { |e|
  [ 200,
    { 'Content-Type' => 'text/html',
      'Set-cookie'   => "id=123456\nname=jack\nphone=65452334"
    },
    [ 'hello world' ]
  ]
}
```

y hacemos `www.example.com` un alias de `127.0.0.1`:

```
[~]$ cat /etc/hosts
##
# Host Database
#
```

```
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1 localhost www.example.com
```

al visitar la página `www.example.com:9292` y abrir las herramientas para desarrolladores tenemos:
Observemos que:

1. Como no hemos establecido el tiempo de caducidad (`expires` `Max-Age`), los cookies son de sesión.

2. Como no hemos establecido el dominio, los cookies son de dominio `www.example.com`.

Estableciendo `expires`

Modifiquemos el ejemplo anterior para establecer una fecha de caducidad:

```
[~/local/src/ruby/sinatra/rack/rack-session-cookie(master)]$ cat study_cookie2.ru
run lambda { |e|
  t = Time.now.gmtime + 3*60
  [ 200,
    { 'Content-Type' => 'text/html',
      'Set-cookie'   => "chuchu=chachi;expires=#{t.strftime("%a, %d-%b-%Y %H:%M:%S GMT")}"
    },
    [ 'hello world' ]
  ]
}
```

Al ejecutar este programa vemos que hemos establecido la caducidad. Obsérvese la diferencia entre GMT y el tiempo de Canarias.

Estableciendo el atributo `domain` **de una cookie**

1. Establezcamos `domain` a `example.com`:

   ```
   ~/local/src/ruby/sinatra/rack/rack-session-cookie(master)]$ cat study_cookie3.ru
   run lambda { |e|
     t = Time.now.gmtime + 3*60
     [ 200,
       { 'Content-Type' => 'text/html',
         'Set-cookie'   => "chuchu=chachi;expires=#{t.strftime("%a, %d-%b-%Y %H:%M:%S GMT")}
                           ";domain=example.com"
       },
       [ 'hello world' ]
     ]
   }
   ```

2. Manipulamos `/etc/hosts`:

   ```
   [~]$ cat /etc/hosts
   127.0.0.1 localhost www.example.com test.example.com app.test
   ```

3. Ejecutamos el servidor y lo visitamos con el navegador en `www.example.com:9292`.

4. A continuación arrancamos este segundo servidor en el puerto 8080:

```
[~/local/src/ruby/sinatra/rack/rack-simple(master)]$ cat config.ru
require './myapp'
run MyApp.new

[~/local/src/ruby/sinatra/rack/rack-simple(master)]$ cat myapp.rb
# my_app.rb
#
class MyApp
  def call env
    [200, {"Content-Type" => "text/html"}, ["Hello Rack Participants"]]
  end
end
```

5. y visitamos `test.example.com:8080` (que de nuevo es resuelto a `localhost`)

La figura muestra que el cookie generado por `www.example.com:9292` es enviado a `test.example.com:8080`:

El atributo `path`

Si `path` es `/` entonces casa con todos las páginas en el dominio. Si `path` es `/foo` entonces casa con `foobar` y `/foo/chuchu/toto.html`.

El atributo `secure`

Si se pone `secure` el cookie solo se envía si se usa `https`

Envío de Cookies As long as the URL requested is within the same domain and path defined in the cookie (and all of the other restrictions – secure, not expired, etc) hold, the cookie will be sent for every request. The client will include a header field similar to this:

```
Cookie: name1 = value1 [;name2=value2]
```

Establecer un cookie usando Rack::Response

```
[~/local/src/ruby/sinatra/rack/rack-debugging(master)]$ cat hello_cookie.rb
require 'rack'

class HelloWorld
  def call env
    response = Rack::Response.new("Hello world!")
    response.status = 200
    response.headers['Content-type'] = "text/plain"
    response.set_cookie('asignatura', 'SYTW')
    response.finish
  end
end

Rack::Handler::WEBrick::run HelloWorld.new
```

Obtener los valores de los cookies usando Rack::Request

Es posible acceder a los cookies con el objeto Rack::Request mediante el método `cookies`. Vease la documentación de Rack::Response y Rack::Request.

```
[~/rack/rack-debugging(master)]$ cat hello_cookie.rb
require 'rack'

class HelloWorld
  def call env
    req      = Rack::Request.new(env)
    response = Rack::Response.new("Hello world! cookies = #{req.cookies.inspect}\n")
    response.write("asignatura => #{req.cookies['asignatura']}") if req.cookies['asignatura']
    response.status = 200
    response['Content-type'] = "text/plain"
    response.set_cookie('asignatura', 'SYTW')
    response.finish
  end
end

Rack::Handler::WEBrick::run HelloWorld.new
```

El código del método `cookies` El método `cookies` retorna un hash:

```
# File lib/rack/request.rb, line 290
def cookies
  hash   = @env["rack.request.cookie_hash"] ||= {}
  string = @env["HTTP_COOKIE"]

  return hash if string == @env["rack.request.cookie_string"]
  hash.clear

  # According to RFC 2109:
  #   If multiple cookies satisfy the criteria above, they are ordered in
  #   the Cookie header such that those with more specific Path attributes
  #   precede those with less specific.  Ordering with respect to other
  #   attributes (e.g., Domain) is unspecified.
  cookies = Utils.parse_query(string, ';,') { |s| Rack::Utils.unescape(s) rescue s }
  cookies.each { |k,v| hash[k] = Array === v ? v.first : v }
  @env["rack.request.cookie_string"] = string
  hash
end
```

Código de `set_cookie`

```
        # File lib/rack/response.rb, line 57
57:     def set_cookie(key, value)
58:       Utils.set_cookie_header!(header, key, value)
59:     end
```

Aquí `value` es un hash con claves `:domain`, `:path`, `:expires`, `:secure` y `:httponly`

Código de `delete_cookie`

```
    # File lib/rack/response.rb, line 61
61:     def delete_cookie(key, value={})
62:       Utils.delete_cookie_header!(header, key, value)
63:     end
```

Aquí `value` es un hash con claves `:domain`, `:path`, `:expires`, `:secure` y `:httponly`

domains, periods, cookies and localhost

1. By design domain names must have at least two dots otherwise browser will say they are invalid.
2. Only hosts within the specified domain can set a cookie for a domain
3. domains must have at least two (2) or three (3) periods in them to prevent domains of the form: `.com`, `.edu`, and `va.us`.
4. Any domain that fails within one of the seven special top level domains `COM`, `EDU`, `NET`, `ORG`, `GOV`, `MIL`, and `INT` require two periods.
5. Any other domain requires at least three.
6. On `localhost`, when we set a cookie on server side and specify the domain explicitly as `localhost` (or `.localhost`), the cookie does not seem to be accepted by some browsers.

16.9. Gestión de Sesiones

Introducción

1. Hypertext Transfer Protocol (HTTP) is stateless: a client computer running a web browser must establish a new Transmission Control Protocol (TCP) network connection to the web server with each new HTTP GET or POST request.
2. The web server, therefore, cannot rely on an established TCP network connection for longer than a single HTTP GET or POST operation.
3. *Session management* is the technique used by the web developer to make the stateless HTTP protocol support session state.
4. For example, once a user has been authenticated to the web server, the user's next HTTP request (GET or POST) should not cause the web server to ask for the user's account and password again.
5. The session information is stored on the web server using the *session identifier* generated as a result of the first (sometimes the first authenticated) request from the end user running a web browser.
6. The "storage.ºf Session IDs and the associated session data (user name, account number, etc.) on the web server is accomplished using a variety of techniques including, but not limited to, local memory, flat files, and databases.
7. A *session token* is a unique identifier that is generated and sent from a server to a client to identify the current interaction session.
8. The client usually stores and sends the token as an HTTP cookie and/or sends it as a parameter in GET or POST queries. The reason to use session tokens is that the client only has to handle the identifier—all session data is stored on the server (usually in a database, to which the client does not have direct access) linked to that identifier.

Uso de Cookies para el manejo de sesiones

1. Allowing users to log into a website is a frequent use of cookies.
2. A web server typically sends a cookie containing a unique *session identifier*. The web browser will send back that session identifier with each subsequent request and related items are stored associated with this unique session identifier.

3. Typically the web server will first send a cookie containing a unique session identifier. Users then submit their credentials and the web application authenticates the session and allows the user access to services.

4. Applications today usually store the gathered information in a database on the server side, rather than storing them in cookies

Ejemplo

Rack::Session::Cookie proporciona un sencillo sistema para gestionar sesiones basado en cookies.

1. La sesión es un cookie que contiene un hash almacenado mediante marshalling codificado en base64.

2. Por defecto el nombre del cookie es `rack.session` pero puede ser modificado mediante el atributo `:key`.

3. Dándole un valor a `secret_key` se garantiza que es comprobada la integridad de los datos de la cookie

4. Para acceder dentro de nuestro programa a la sesión accedemos al hash `env["rack.session"]` o bien `env["key-value"]` si hemos especificado el atributo `:key`

Sigue un ejemplo:

```
[~/local/src/ruby/sinatra/rack/rack-session-cookie(master)]$ cat configapp.ru
require 'pp'
require './myapp'

use Rack::Session::Cookie,
      :key => 'rack.session',
      :domain => 'example.com',
      :secret => 'some_secret'

run MyApp.new

[~/local/src/ruby/sinatra/rack/rack-session-cookie(master)]$ cat myapp.rb
class MyApp

  def set_env(env)
    @env = env
    @session = env['rack.session']
  end

  def some_key
    return @session['some_key'].to_i if @session['some_key']
    @session['some_key'] = 0
  end

  def some_key=(value)
    @session['some_key'] = value
  end

  def call(env)
    set_env(env)
    res = Rack::Response.new
    req = Rack::Request.new env
```

```
    self.some_key = self.some_key + 1 if req.path == '/'

    res.write("some_key = #{@session['some_key']}\n")

    res.finish
  end

end
```

Hagamos la prueba conectándonos a `www.example.com`. Para ello edtiamos `/etc/hosts` para que `localhost` apunte a `www.example.com`:

```
[~/local/src/ruby/sinatra/rack/rack-session-cookie(master)]$ cat /etc/hosts
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1 localhost www.example.com
...
```

Arrancamos el servidor:

```
[~/local/src/ruby/sinatra/rack/rack-session-cookie(master)]$ rackup  configapp.ru
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
```

Y visitamos `www.example.com` con nuestro navegador:

16.9.1. Ejercicio

Supongamos el siguiente programa rack en el que se incrementa la variable `@some_key`:

```
[~/local/src/ruby/sinatra/rack/rack-appvswebserver(icon)]$ cat configapp.ru
class Persistence

  def call(env)

    res = Rack::Response.new
    req = Rack::Request.new env

    @some_key ||= 0
    @some_key = @some_key + 1

    res.write("@some_key = #{@some_key}\n")

    res.finish
  end

end

run Persistence.new
```

Supongamos que arranco el servidor:

```
[~/local/src/ruby/sinatra/rack/rack-appvswebserver(master)]$ rackup configapp.ru >> Thin web s
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
```

Nótese que con `thin` arrancado desde `rack` se tienen los valores de `env` para las claves:

```
rack.multithread => false
rack.multiprocess => false
```

lo que indica que el servidor no está soportando multithreading ni multiproceso.

Responda a estas preguntas:

1. ¿Que valores de `@some_key` serán mostrados cuando me conecto a `localhost:9292`?
2. ¿Y si recargo la página varias veces?
3. ¿Y si abro un nuevo navegador o ventana de incógnito en la misma URL?
4. ¿Y si re-arranco el servidor?
5. ¿Como afectaría a la conducta que el servidor fuera multithreading?

```
[~/local/src/ruby/sinatra/rack/rack-appvswebserver(icon)]$ rvm use jruby-1.7.3
Using /Users/casiano/.rvm/gems/jruby-1.7.3
[~/local/src/ruby/sinatra/rack/rack-appvswebserver(icon)]$ rackup configapp.ru
Puma 2.6.0 starting...
* Min threads: 0, max threads: 16
* Environment: development
* Listening on tcp://0.0.0.0:9292
rack.multithread => true
rack.multiprocess => false
```

```
[~/local/src/ruby/sinatra/rack/rack-appvswebserver(icon)]$ cat Rakefile
desc "run the server"
task :default do
  sh <<-"EOS"
  #rvm use jruby-1.7.3 &&
  #ruby -v &&
  rackup -s puma configapp.ru
  EOS
end

desc "run the client"
task :client do
  pids = []
  (0...100).each do
    pids << fork do
      sh %q{curl -v 'http://localhost:9292' >> salida 2>> logs}
    end
  end
  puts pids
end

desc "remove output and logs"
task :clean do
  sh "rm -f salida logs"
end
```

De acuerdo a una respuesta en StackOverflow a la pregunta: Is Sinatra multi-threaded? I read else where that ”

The choice is mainly made by the server and middleware you use:

1. Multi-Process, non-preforking: Mongrel, Thin, WEBrick, Zbatery
2. Multi-Process, preforking: Unicorn, Rainbows, Passenger
3. Evented (suited for sinatra-synchrony): Thin, Rainbows, Zbatery
4. Threaded: Net::HTTP::Server, Threaded Mongrel, Puma, Rainbows, Zbatery, Phusion Passenger Enterprise ¿= 4
5. Since Sinatra 1.3.0, Thin will be started in threaded mode, if it is started by Sinatra (i.e. with ruby app.rb, but not with the thin command, nor with rackup).

16.10. Ejemplo Simple Combinando Rack::Request, Rack::Response y Middleware (Lobster)

Este código se encuentra en https://github.com/crguezl/rack-lobster

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ cat lobster.rb
require 'rack/request'
require 'rack/response'

module Rack
  class Lobster
    LobsterString = "a lobster"

    def call(env)
      req = Request.new(env)

      req.env.keys.sort.each { |x| puts "#{x} => #{req.env[x]}" }

      if req.GET["flip"] == "left"
        lobster = LobsterString.reverse
        href = "?flip=right"
      elsif req.GET["flip"] == "crash"
        raise "Lobster crashed"
      else
        lobster = LobsterString
        href = "?flip=left"
      end

      res = Response.new
      res.write <<-"EOS"
      <title>Lobstericious!</title>
      <pre>
      #{lobster}
      </pre>
      <p><a href='#{href}'>flip!</a></p>
      <p><a href='?flip=crash'>crash!</a></p>
      EOS
      res.finish
    end
  end
```

```
end

if $0 == __FILE__
  require 'rack'
  require 'rack/showexceptions'
  Rack::Server.start(
    :app => Rack::ShowExceptions.new(
              Rack::Lint.new(
                Rack::Lobster.new)),
    :Port => 9292,
    :server => 'thin'
  )
end
```

Véase:

1. rack/lib/rack/showexceptions.rb

   (Rack::ShowExceptions catches all exceptions raised from the app it wraps. It shows a useful backtrace with the sourcefile and clickable context, the whole Rack environment and the request data.

   Be careful when you use this on public-facing sites as it could reveal information helpful to attackers)

2. rack/lib/rack/lint.rb en GitHub

   (Rack::Lint validates your application and the requests and responses according to the Rack spec)

Tanto Rack::ShowExceptions como Rack::Lint disponen de un método `call` que recibe una variable `env` describiendo el entorno CGI. Esto es, se trata de aplicaciones que siguen el protocolo Rack. Así este código:

```
  Rack::Server.start(
    :app => Rack::ShowExceptions.new(
              Rack::Lint.new(
                Rack::Lobster.new)),
    :Port => 9292,
    :server => 'thin'
  )
```

construye una nueva objeto/aplicación Rack que es la composición de los tres Racks.

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ cat Rakefile
desc "run the server"
task :default do
  sh "ruby lobster.rb"
end

desc "run the client flip left"
task :left do
  sh %q{curl -v 'http://localhost:9292?flip=left'}
end

desc "run the client flip right"
task :right do
  sh %q{curl -v 'http://localhost:9292?flip=right'}
```

```
end

desc "run the client. Generate exception"
task :crash do
  sh %q{curl -v 'http://localhost:9292/?flip=crash'}
end

[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ rake left
curl -v 'http://localhost:9292?flip=left'
* About to connect() to localhost port 9292 (#0)
*   Trying ::1... Connection refused
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /?flip=left HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Length: 168
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
      <title>Lobstericious!</title>
      <pre>
      retsbol a
      </pre>
      <p><a href='?flip=right'>flip!</a></p>
      <p><a href='?flip=crash'>crash!</a></p>
* Connection #0 to host localhost left intact
* Closing connection #0

[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ rake right
curl -v 'http://localhost:9292?flip=right'
* About to connect() to localhost port 9292 (#0)
*   Trying ::1... Connection refused
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /?flip=right HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Length: 167
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
      <title>Lobstericious!</title>
      <pre>
      a lobster
      </pre>
      <p><a href='?flip=left'>flip!</a></p>
      <p><a href='?flip=crash'>crash!</a></p>
```

```
* Connection #0 to host localhost left intact
* Closing connection #0
```

16.11. Práctica: Accediendo a Twitter y Mostrando los últimos twitts en una página

Convierta el programa de ejemplo usado en la sección *Ejemplo en Ruby: Accediendo a Twitter* **??** en una aplicación Rack que muestre en su página los últimos twitts de una lista de usuarios obtenidos desde un formulario (puede modificar/diseñar la interfaz como crea conveniente)

16.12. Ejemplo: Basic Authentication

Rack::Auth::Basic implements HTTP Basic Authentication, as per RFC 2617.

Introducción

1. In the context of an HTTP transaction, basic access authentication is a method for an HTTP user agent to provide a user name and password when making a request.
2. *HTTP Basic authentication* (*BA*) implementation is the simplest technique for enforcing access controls to web resources because it doesn't require cookies, session identifier and login pages. Rather, HTTP Basic authentication uses static, standard HTTP headers which means that no handshakes have to be done in anticipation.
3. The BA mechanism provides no confidentiality protection for the transmitted credentials. They are merely encoded with BASE64 in transit, but not encrypted or hashed in any way. Basic Authentication is, therefore, typically used over HTTPS.
4. Because BA header has to be sent with each HTTP request, the web browser needs to cache the credentials for a reasonable period to avoid constant prompting user for the username and password. Caching policy differs between browsers.
5. While HTTP does not provide a method for web server to instruct the browser to "log out"the user (forget cached credentials), there are a number of workarounds using specific features in various browsers. One of them is redirecting the user to an URL on the same domain containing credentials that are intentionally incorrect

Protocolo

1. When the server wants the user agent to authenticate itself towards the server, it can send a request for authentication.
2. This request should be sent using the HTTP 401 Not Authorized response code containing a WWW-Authenticate HTTP header.
3. The WWW-Authenticate header for basic authentication (used most often) is constructed as following:

   ```
   WWW-Authenticate: Basic realm="insert realm"
   ```

4. When the user agent wants to send the server authentication credentials it may use the Authorization header.
5. The Authorization header is constructed as follows:

   a) Username and password are combined into a string `username:password` *b*) The resulting string literal is then encoded using Base64

c) The authorization method and a space i.e. `Basic ` is then put before the encoded string.

d) For example, if the user agent uses `Aladdin` as the username and `open sesame` as the password then the header is formed as follows:.

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Ejemplo de BA en Rack

Initialize with the Rack application that you want protecting, and a block that checks if a username and password pair are valid.

Puede encontrar el fuente en GitHub

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ cat protectedlobster.rb
require 'rack'
require './lobster'
require 'yaml'

lobster = Rack::Lobster.new

passwd = YAML.load(File.open('etc/passwd.yml').read)

protected_lobster = Rack::Auth::Basic.new(lobster) do |username, password|
  passwd[username] == password
end

protected_lobster.realm = 'Lobster 2.0'
pretty_protected_lobster = Rack::ShowStatus.new(Rack::ShowExceptions.new(protected_lobster))

Rack::Server.start :app => pretty_protected_lobster, :Port => 9292
```

lobster.rb

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ cat lobster.rb
require 'rack/request'
require 'rack/response'

module Rack
  class Lobster
    LobsterString = "a lobster"

    def call(env)
      req = Request.new(env)

      req.env.keys.sort.each { |x| puts "#{x} => #{req.env[x]}" }

      if req.GET["flip"] == "left"
        lobster = LobsterString.reverse
        href = "?flip=right"
      elsif req.GET["flip"] == "crash"
        raise "Lobster crashed"
      else
        lobster = LobsterString
        href = "?flip=left"
```

```
      end

      res = Response.new
      res.write <<-"EOS"
      <title>Lobstericious!</title>
      <pre>
      #{lobster}
      </pre>
      <p><a href='#{href}'>flip!</a></p>
      <p><a href='?flip=crash'>crash!</a></p>
      EOS
      res.finish
    end
  end
end

if $0 == __FILE__
  require 'rack'
  require 'rack/showexceptions'
  Rack::Server.start(
    :app => Rack::ShowExceptions.new(
              Rack::Lint.new(
                Rack::Lobster.new)),
    :Port => 9292,
    :server => 'thin'
  )
end
```

etc/passwd.yml

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ cat etc/passwd.yml
--- # Indented Block
   casiano: tutu
   ana: titi
```

Rakefile

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ cat Rakefile
...

desc "run the server for protectedlobster"
task :protected do
  sh "ruby protectedlobster.rb"
end

desc "run the client with user and password flip left"
task :protectedleft do
  sh %q{curl -v --basic -u casiano:tutu 'http://localhost:9292?flip=left'}
end

...

task :crash do
  sh %q{curl -v 'http://localhost:9292/?flip=crash'}
```

```
end
```

Ejecución

1. Servidor:

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ rake protected
ruby protectedlobster.rb
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
```

2. Cliente:

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ rake protectedleft
curl -v --basic -u casiano:tutu 'http://localhost:9292?flip=left'
* About to connect() to localhost port 9292 (#0)
*   Trying ::1... Connection refused
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
* Server auth using Basic with user 'casiano'
> GET /?flip=left HTTP/1.1
> Authorization: Basic Y2FzaWFubzpzZWNyZXRv
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Length: 168
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
      <title>Lobstericious!</title>
      <pre>
      retsbol a
      </pre>
      <p><a href='?flip=right'>flip!</a></p>
      <p><a href='?flip=crash'>crash!</a></p>
* Connection #0 to host localhost left intact
* Closing connection #0
```

3. Servidor después de la petición:

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ rake protected
ruby protectedlobster.rb
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
...
HTTP_AUTHORIZATION => Basic Y2FzaWFubzp0dXR1
REMOTE_USER => casiano
...
```

Véase

1. Código de rack-lobster en GitHub
2. Código fuente de Rack::Auth::Basic
3. Documentación en Rack::Auth::Basic
4. La Wikipedia Basic Access Authentication

16.13. Redirección

```
[~/local/src/ruby/sinatra/rack/rack-debugging(master)]$ cat redirect.rb
require 'rack'
require 'thin'

app = lambda do |env|
  req = Rack::Request.new env
  res = Rack::Response.new

  if req.path_info == '/redirect'
    res.redirect('https://plus.google.com/u/0/')
  else
    res.write "You did not get redirected"
  end
  res.finish
end

Rack::Server.start(
 :app => app,
 :Port => 9292,
 :server => 'thin'
)
```

16.14. La Estructura de una Aplicación Rack

```
[~/local/src/ruby/sinatra/rack/rack-debugging(master)]$ cat middlefoo.rb
require 'rack'

class MiddleFoo

  def initialize(app)
    @app = app
  end

  def call env
    # Podemos modificar el request (env) aqui
    env['chuchu'] = 'SYTW'
    status, headers, body = @app.call(env)
    # Podemos modificar la respuesta aqui
    newbody = body.map(&:upcase)
    [status, headers, newbody]
  end
end
```

```
[~/local/src/ruby/sinatra/rack/rack-debugging(master)]$ cat hello_middle.rb
require 'rack'
require './middlefoo'

class HelloWorld
  def call env
    [200, {"Content-Type" => "text/plain"}, ["Hello world\nchuchu=#{env['chuchu']}\n"]]
  end
end

Rack::Handler::WEBrick::run MiddleFoo.new(HelloWorld.new)
```

Cuando ejecutamos el programa produce la salida:

```
HELLO WORLD
CHUCHU=SYTW
```

16.15. rackup

Introducción

1. The Rack gem gives you a rackup command which lets you start your app on any supported application server.
2. rackup is a useful tool for running Rack applications, which uses the Rack::Builder DSL to configure middleware and build up applications easily.
3. rackup automatically figures out the environment it is run in, and runs your application as FastCGI, CGI, or standalone with Mongrel or WEBrick, all from the same configuration.

De hecho este es todo el código del ejecutable rackup

```
#!/usr/bin/env ruby

require "rack"
Rack::Server.start
```

El método `start` starts a new rack server (like running rackup). This will parse `ARGV` and provide standard `ARGV` rackup options, defaulting to load `config.ru`.

Providing an option hash will prevent `ARGV` parsing and will not include any default options.

This method can be used to very easily launch a CGI application, for example:

```
  Rack::Server.start(
    :app => lambda do |e|
      [200, {'Content-Type' => 'text/html'}, ['hello world']]
    end,
    :server => 'cgi'
  )
```

Further options available here are documented on Rack::Server`#initialize` (véase el código en Rack::Server):

```
    def self.start(options = nil)
      new(options).start
    end
```

como se ve, el código de Rack::Server está en Github.

The Options of `start` and `new` may include:

1. `:app` a rack application to run (overrides :config)
2. `:config` a rackup configuration file path to load (.ru)
3. `:environment` this selects the middleware that will be wrapped around your application. Default options available are:

   a) development: CommonLogger, ShowExceptions, and Lint
   b) deployment: CommonLogger
   c) none: no extra middleware

   note: when the server is a cgi server, CommonLogger is not included.
4. `:server` choose a specific Rack::Handler, e.g. cgi, fcgi, webrick
5. `:daemonize` if true, the server will daemonize itself (fork, detach, etc)
6. `:pid` path to write a pid file after daemonize
7. `:Host` the host address to bind to (used by supporting Rack::Handler)
8. `:Port` the port to bind to (used by supporting Rack::Handler)
9. `:AccessLog` webrick acess log options (or supporting Rack::Handler)
10. `:debug` turn on debug output (`$DEBUG = true`)
11. `:warn` turn on warnings (`$-w = true`)
12. `:include` add given paths to `$LOAD_PATH`
13. `:require` require the given libraries

Ejemplo de uso

Si no se especifica, `rackup` busca un fichero con nombre `config.ru`.

```
[~/local/src/ruby/sinatra/rack/rackup/simple(master)]$ cat config.ru
require './myapp'
run MyApp.new
```

Esta es la aplicación:

```
[~/local/src/ruby/sinatra/rack/rackup/simple(master)]$ cat myapp.rb
class MyApp
  def call env
    [200, {"Content-Type" => "text/html"}, ["Hello Rack Participants"]]
  end
end
```

```
[~/local/src/ruby/sinatra/rack/rackup/simple(master)]$ cat Rakefile
task :default => :server

desc "run server"
task :server do
  sh "rackup"
end

desc "run client via curl"
task :client do
  sh "curl -v localhost:9292"
end
```

Ejecución

```
[~/local/src/ruby/sinatra/rack/rackup/simple(master)]$ rackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop

[~/local/src/ruby/sinatra/rack/rackup/simple(master)]$ curl -v localhost:9292
* About to connect() to localhost port 9292 (#0)
*   Trying ::1... Connection refused
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Transfer-Encoding: chunked
< Connection: close
< Server: thin 1.5.1 codename Straight Razor
<
* Closing connection #0
Hello Rack Participants
```

Opciones del ejecutable rackup

1. Véase rackup.

```
[~]$ rackup --help
Usage: rackup [ruby options] [rack options] [rackup config]

Ruby options:
  -e, --eval LINE          evaluate a LINE of code
  -d, --debug              set debugging flags (set $DEBUG to true)
  -w, --warn               turn warnings on for your script
  -I, --include PATH       specify $LOAD_PATH (may be used more than once)
  -r, --require LIBRARY    require the library, before executing your script

Rack options:
  -s, --server SERVER      serve using SERVER (webrick/mongrel)
  -o, --host HOST          listen on HOST (default: 0.0.0.0)
  -p, --port PORT          use PORT (default: 9292)
  -O NAME[=VALUE],         pass VALUE to the server as option NAME.
                           If no VALUE, sets it to true.
                           Run
                           'rackup -s SERVER -h'
                           to get a list of options for SERVER
      --option
  -E, --env ENVIRONMENT    use ENVIRONMENT for defaults (default: development)
  -D, --daemonize          run daemonized in the background
  -P, --pid FILE           file to store PID (default: rack.pid)
```

```
Common options:
  -h, -?, --help          Show this message
      --version           Show version
```

Especificación de Opciones en la primera línea Si la primera línea de un fichero `config.ru` empieza por `\#` es tratada como una línea de opciones permitiendo así que los argumentos de `rackup` se especifiquen en el fichero de configuración:

```
#\-w -p 8765

use Rack::Reloader, 0
use Rack::ContentLength

app = proc do |env|
  [200, {'content-Type' => 'text/plain' }, ['a']]
end

run app
```

16.16. Rack::Static

Véase

1. Documentación de Rack::Static
2. Este ejemplo: rack-static-example en GitHub
3. Código fuente de Rack::Static en GitHub

Ejemplo

```
[~/local/src/ruby/sinatra/rack/rack-static(master)]$ tree
.
|--- README
|--- README.md
|--- Rakefile
|--- config.ru
|--- myapp.rb
`---- public
   `--- index.html

1 directory, 6 files
[~/local/src/ruby/sinatra/rack/rack-static(master)]$ cat public/index.html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DT
<html>
  <head>
    <title>Hello</title>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```

```
[~/local/src/ruby/sinatra/rack/rack-static(master)]$ cat config.ru
require './myapp'

use Rack::Static, :urls => ["/public"]

run MyApp.new
```

```
[~/local/src/ruby/sinatra/rack/rack-static(master)]$ cat Rakefile
task :default => :server

desc "run server"
task :server do
  sh "rackup"
end

desc "run client via curl"
task :client do
  sh "curl -v localhost:9292"
end

desc "access to static file"
task :index do
  sh "curl -v localhost:9292/public/index.html"
end
```

```
[~/local/src/ruby/sinatra/rack/rack-static(master)]$ cat myapp.rb
# my_app.rb
#
class MyApp
  def call env
    [200, {"Content-Type" => "text/html"}, ["Hello SYTW!"]]
  end
end
```

```
[~/local/src/ruby/sinatra/rack/rack-static(master)]$ rake client
curl -v localhost:9292
* About to connect() to localhost port 9292 (#0)
*   Trying ::1... Connection refused
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Transfer-Encoding: chunked
< Connection: close
< Server: thin 1.5.1 codename Straight Razor
<
* Closing connection #0
Hello SYTW!
```

```
[~/local/src/ruby/sinatra/rack/rack-static(master)]$ rake index
```

```
curl -v localhost:9292/public/index.html
* About to connect() to localhost port 9292 (#0)
*   Trying ::1... Connection refused
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /public/index.html HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Last-Modified: Thu, 03 Oct 2013 08:24:43 GMT
< Content-Type: text/html
< Content-Length: 227
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DT
<html>
  <head>
    <title>Hello</title>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>

* Connection #0 to host localhost left intact
* Closing connection #0
```

`El comando rackup`

`rackup` converts the supplied rack config file to an instance of Rack::Builder. In short, rack config files are evaluated within the context of a Rack::Builder object.

Rackup also has a *use* method that accepts a *middleware*. Let us use one of Rack's built-in middleware.

```
[~/sinatra/rackup/middleware]$ cat config.ru
require './myapp'
require './myrackmiddleware'
use Rack::Reloader
use MyRackMiddleware
run MyApp.new

[~/sinatra/rackup/middleware]$ cat myapp.rb
# myapp.rb
class MyApp
  def call(env)
    [200, {"Content-Type" => "text/html"}, ["Hello Rack Participants from across the globe"]]
  end
end

[~/sinatra/rackup/middleware]$ cat myrackmiddleware.rb
class MyRackMiddleware
  def initialize(appl)
```

```
    @appl = appl
  end
  def call(env)
    status, headers, body = @appl.call(env)
    append_s = "... greetings from RubyLearning!!"
    [status, headers, body << append_s]
  end
end
```

Véase

- RailCast #151 Rack Middleware
- Rack Middleware as a General Purpose Abstraction por Mitchell Hashimoto

16.17. Un Ejemplo Simple: Piedra, Papel, tijeras

```
[~/rack/rack-rock-paper-scissors(simple)]$ cat -n rps.rb
require 'rack/request'
require 'rack/response'

module RockPaperScissors
  class App

    def initialize(app = nil)
      @app = app
      @content_type = :html
      @defeat = {'rock' => 'scissors', 'paper' => 'rock', 'scissors' => 'paper'}
      @throws = @defeat.keys
      @choose = @throws.map { |x|
         %Q{ <li><a href="/?choice=#{x}">#{x}</a></li> }
      }.join("\n")
      @choose = "<p>\n<ul>\n#{@choose}\n</ul>"
    end

    def call(env)
      req = Rack::Request.new(env)

      req.env.keys.sort.each { |x| puts "#{x} => #{req.env[x]}" }

      computer_throw = @throws.sample
      player_throw = req.GET["choice"]
      anwser = if !@throws.include?(player_throw)
          "Choose one of the following:"
        elsif player_throw == computer_throw
          "You tied with the computer"
        elsif computer_throw == @defeat[player_throw]
          "Nicely done; #{player_throw} beats #{computer_throw}"
        else
          "Ouch; #{computer_throw} beats #{player_throw}. Better luck next time!"
        end

      res = Rack::Response.new
      res.write <<-"EOS"
```

```
      <html>
        <title>rps</title>
        <body>
          <h1>
             #{anwser}
             #{@choose}
          </h1>
        </body>
      </html>
      EOS
      res.finish
    end # call
  end   # App
end     # RockPaperScissors

if $0 == __FILE__
  require 'rack'
  require 'rack/showexceptions'
  Rack::Server.start(
    :app => Rack::ShowExceptions.new(
              Rack::Lint.new(
                RockPaperScissors::App.new)),
    :Port => 9292,
    :server => 'thin'
  )
end
```

El Objeto `req` El objeto `req` pertenece a la clase `Rack::Request`. Tiene un único atributo `env`:

```
(rdb:1)  req
#<Rack::Request:0x007f8d735b1410
@env={
"SERVER_SOFTWARE"=>"thin 1.5.1 codename Straight Razor",
"SERVER_NAME"=>"0.0.0.0",
"rack.input"=>#<Rack::Lint::InputWrapper:0x007f8d735776c0
                @input=#<StringIO:0x007f8d735426a0>>, "rack.version"=>[1, 0],
                "rack.errors"=>#<Rack::Lint::ErrorWrapper:0x007f8d73577620 @error=#<IO:<STDERR
              >,
"rack.multithread"=>false,
"rack.multiprocess"=>false,
"rack.run_once"=>false,
"REQUEST_METHOD"=>"GET",
"REQUEST_PATH"=>"/",
"PATH_INFO"=>"/",
"REQUEST_URI"=>"/",
"HTTP_VERSION"=>"HTTP/1.1",
"HTTP_HOST"=>"0.0.0.0:9292",
"HTTP_CONNECTION"=>"keep-alive",
"HTTP_ACCEPT"=>"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
"HTTP_USER_AGENT"=>"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_5) AppleWebKit/537.36 (KHTML,
"HTTP_ACCEPT_ENCODING"=>"gzip,deflate,sdch",
"HTTP_ACCEPT_LANGUAGE"=>"es-ES,es;q=0.8",
"GATEWAY_INTERFACE"=>"CGI/1.2",
"SERVER_PORT"=>"9292",
```

```
"QUERY_STRING"=>"",
"SERVER_PROTOCOL"=>"HTTP/1.1",
"rack.url_scheme"=>"http",
"SCRIPT_NAME"=>"",
"REMOTE_ADDR"=>"127.0.0.1",
"async.callback"=>#<Method: Thin::Connection#post_process>,
"async.close"=>#<EventMachine::DefaultDeferrable:0x007f8d735603f8>}>
```

Cuando llamamos a `GET` para obtener el valor del parámetro `choice`:

```
player_throw = req.GET["choice"]
```

Si visitamos la página `http://0.0.0.0:9292/` el entorno contiene algo como esto:

```
rdb:1) p @env
{"SERVER_SOFTWARE"=>"thin 1.5.1 codename Straight Razor",
 ...
 "QUERY_STRING"=>"",
 "REQUEST_URI"=>"/"
 ...
}
```

el código de `GET` nos da los datos almacenados en `QUERY_STRING`:

```
   def GET
      if @env["rack.request.query_string"] == query_string
        @env["rack.request.query_hash"]
      else
        @env["rack.request.query_string"] = query_string
        @env["rack.request.query_hash"]   = parse_query(query_string)
      end
    end

   def query_string;    @env["QUERY_STRING"].to_s                end
```

si es la primera vez, `@env["rack.request.query_string"]` está a `nil` y se ejecuta el `else` inicializando `@env["rack.request.query_string"]` y `@env["rack.request.query_hash"]`

Si por ejemplo visitamos la URL: `http://localhost:9292?choice=rock` entonces `env` contendrá:

```
rdb:1) p env
{ ...
  "QUERY_STRING"=>"choice=rock",
  "REQUEST_URI"=>"/?choice=rock",
  ...
}
```

Familiaricemonos con algunos de los métodos de `Rack::Request`:

```
(rdb:1) req.GET
{"choice"=>"paper"}
(rdb:1)  req.GET["choice"]
"paper"
(rdb:1) req.POST
{}
(rdb:1) req.params
{"choice"=>"paper"}
(rdb:1) req["choice"]
```

```
"paper"
(rdb:1) req[:choice]
"paper"
(rdb:1) req.cookies()
{}
(rdb:1) req.get?
true
(rdb:1) req.post?
false
(rdb:1) req.fullpath
"/?choice=paper"
(rdb:1) req.host
"0.0.0.0"
(rdb:1) req.host_with_port
"0.0.0.0:9292"
(rdb:1) req.body
#<Rack::Lint::InputWrapper:0x007f8d7369b5d8 @input=#<StringIO:0x007f8d73690318>>
(rdb:1) req.cookies()
{}
(rdb:1) req.get?
true
(rdb:1) req.post?
false
(rdb:1) req.fullpath
"/?choice=paper"
(rdb:1) req.host
"0.0.0.0"
(rdb:1) req.host_with_port
"0.0.0.0:9292"
(rdb:1) req.ip
"127.0.0.1"
(rdb:1) req.params
{"choice"=>"paper"}
(rdb:1) req.path
"/"
(rdb:1) req.path_info
"/"
(rdb:1) req.port
9292
(rdb:1) req.request_method
"GET"
(rdb:1) req.scheme
"http"
(rdb:1) req.url
"http://0.0.0.0:9292/?choice=paper"
(rdb:1) req.user_agent
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_5) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/29.0.1547.76 Safari/537.36"
(rdb:1) req.values_at("choice")
["paper"]
```

Rakefile

```
[~/rack/rack-rock-paper-scissors(simple)]$ cat Rakefile
```

```
desc "run the server"
task :default do
  sh "ruby rps.rb"
end

desc "run the client with rock"
task :rock do
  sh %q{curl -v 'http://localhost:9292?choice=rock'}
end

desc "run the client with paper"
task :paper do
  sh %q{curl -v 'http://localhost:9292?choice=paper'}
end

desc "run the client with scissors"
task :scissors do
  sh %q{curl -v 'http://localhost:9292?choice=scissors'}
end
```

1. curl

Ejecuciones

```
[~/rack/rack-rock-paper-scissors(simple)]$ rake
ruby rps.rb
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop

[~/rack/rack-rock-paper-scissors(simple)]$ rake rock
curl -v 'http://localhost:9292?choice=rock'
* About to connect() to localhost port 9292 (#0)
*   Trying ::1... Connection refused
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /?choice=rock HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8x zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Length: 332
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
      <html>
        <title>rps</title>
        <body>
          <h1>
             Nicely done; rock beats scissors
             <p>
<ul>
 <li><a href="/?choice=rock">rock</a></li>
```

```
 <li><a href="/?choice=paper">paper</a></li>
 <li><a href="/?choice=scissors">scissors</a></li>
</ul>
          </h1>
        </body>
      </html>
* Connection #0 to host localhost left intact
* Closing connection #0

[~/rack/rack-rock-paper-scissors(simple)]$ rake
ruby rps.rb
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
GATEWAY_INTERFACE => CGI/1.2
HTTP_ACCEPT => */*
HTTP_HOST => localhost:9292
HTTP_USER_AGENT => curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8x zlib
HTTP_VERSION => HTTP/1.1
PATH_INFO => /
QUERY_STRING => choice=rock
REMOTE_ADDR => 127.0.0.1
REQUEST_METHOD => GET
REQUEST_PATH => /
REQUEST_URI => /?choice=rock
SCRIPT_NAME =>
SERVER_NAME => localhost
SERVER_PORT => 9292
SERVER_PROTOCOL => HTTP/1.1
SERVER_SOFTWARE => thin 1.5.1 codename Straight Razor
async.callback => #<Method: Thin::Connection#post_process>
async.close => #<EventMachine::DefaultDeferrable:0x007ff4e2bf8e78>
rack.errors => #<Rack::Lint::ErrorWrapper:0x007ff4e2c04b88>
rack.input => #<Rack::Lint::InputWrapper:0x007ff4e2c04c00>
rack.multiprocess => false
rack.multithread => false
rack.run_once => false
rack.url_scheme => http
rack.version => [1, 0]
```

Véase También

Véase la documentación de las siguientes clases:

1. Rack::Request
2. Rack::Response
3. Rack::Server
4. Rack::ShowExceptions
5. Rack::Lint

16.17.1. Práctica: Rock, Paper, Scissors: Debugging

Implemente el ejemplo anterior Rock, Paper, Scissors y ejecútelo con un depurador. Lea la sección *Depurando una Ejecución con Ruby* 25.1.

Instale la gema `debugger`. Llame al método `debugger` en el punto en el que quiere detener la ejecución para inspeccionar el estado del programa. Arranque el servidor y en el navegador visite la página.

16.17.2. Práctica: Añadir Template Haml a Rock, Paper, Scissors

Use Haml para crear un template `index.haml` en un directorio `views`.

```
[~/local/src/ruby/sinatra/rack/rack-rock-paper-scissors(template)]$ tree
.
|--- README
|--- Rakefile
|--- rps.rb
`--- views
    `--- index.haml
```

El template puede ser usado así:

```
require 'rack/request'
require 'rack/response'
require 'haml'

module RockPaperScissors
  class App
     ...

    def call(env)
      ...
      engine = Haml::Engine.new File.open("views/index.haml").read
      res = Rack::Response.new
      res.write engine.render({},
        :answer => answer,
        :choose => @choose,
        :throws => @throws)
      res.finish
    end # call
  end   # App
end     # RockPaperScissors
```

Véase:

1. Haml::Engine

La sintáxis del método `render` es:

```
  (String) render(scope = Object.new, locals = {})
```

También se puede usar como `to_html`. Procesa el template y retorna el resultado como una cadena.

El parámetro `scope` es el contexto en el cual se evalúa el template.

Si es un objeto `Binding` `haml` lo usa como segundo argumento de `Kernel#eval` (Véase la sección *Bindings (encarpetados) y eval* en **??**) en otro caso, `haml` utiliza `#instance_eval`.

Nótese que Haml modifica el contexto de la evaluación (bien el objeto ámbito o el objeto `self` del ámbito del binding). Se extiende `Haml::Helpers` y se establecen diversas variables de instancia (todas ellas prefijadas con `haml_`).

Por ejemplo:

```
s = "foobar"
Haml::Engine.new("%p= upcase").render(s)
```

produce:

```
"<p>FOOBAR</p>"
```

Ahora `s` extiende `Haml::Helpers` :

```
s.respond_to?(:html_attrs) #=> true
```

`Haml::Helpers` contiene un conjunto de métodos/utilidades para facilitar distintas tareas. La idea de que estén disponibles en el contexto es para ayudarnos dentro del template. Por ejemplo el método

```
- (String) escape_once(text)
```

Escapa las entidades HTML en el texto.

`locals` es un hash de variables locales que se deja disponible dentro del template. Por ejemplo:

```
Haml::Engine.new("%p= foo").render(Object.new, :foo => "Hello, world!")
```

producirá:

```
"<p>Hello, world!</p>"
```

Si se pasa un bloque a `render` el bloque será ejecutado en aquellos puntos en los que se llama a `yield` desde el template.

Debido a algunas peculiaridades de Ruby, si el ámbito es un `Binding` y se proporciona también un bloque, el contexto de la evaluación puede no ser el que el usuario espera.

Parametros:

1. `scope (Binding, Proc, Object)` (por defecto: `Object.new`). El contexto en el que se evalúa el template
2. `locals ({Symbol => Object})` (por defecto: `{}`). Variables locales que se dejan disponibles en el template
3. `block (#to_proc)` Un bloque que será llamado desde el template.
4. Retorna una `String` con el template renderizado

16.17.3. Práctica: Añada Hojas de Estilo a Piedra Papel Tijeras

Añada hojas de estilo a la práctica anterior (sección 16.17.2).

1. Mostramos una posible estructura de ficheros en la que se incluyen hojas de estilo usando `bootstrap` :

   ```
   [~/local/src/ruby/sinatra/rack/rack-rock-paper-scissors(bootstrap)]$ tree
   .
   |--- Gemfile
   |--- Gemfile.lock
   |--- README
   |--- Rakefile
   |--- TODO
   |--- config.ru
   |--- lib
   |   `--- rps.rb
   ```

```
|--- public
|   |--- css
|   |   |--- bootstrap-responsive.css
|   |   |--- bootstrap-responsive.min.css
|   |   |--- bootstrap.css
|   |   `-- bootstrap.min.css
|   |--- img
|   |   |--- glyphicons-halflings-white.png
|   |   |--- glyphicons-halflings.png
|   |   `-- programming-languages.jpg
|   `-- js
|       |--- bootstrap.js
|       `--- bootstrap.min.js
|--- rps.rb
`-- views
    `--- index.haml

6 directories, 18 files
```

2. El middleware Rack::Static intercepta las peticiones por ficheros estáticos (javascript, imágenes, hojas de estilo, etc.) basandose en los prefijos de las urls pasadas en las opciones y los sirve utilizando un objeto Rack::File. Ejemplos:

   ```
   use Rack::Static, :urls => ["/public"]
   ```

   Servirá todas las peticiones que comiencen por `/public` desde la carpeta `public` localizada en el directorio actual (esto es `public/*`).

   En nuestro jerarquía pondremos en el programa `rps.rb`:

   ```
   builder = Rack::Builder.new do
     use Rack::Static, :urls => ['/public']
     use Rack::ShowExceptions
     use Rack::Lint

     run RockPaperScissors::App.new
   end

   Rack::Handler::Thin.run builder
   ```

   y dentro del template `haml` nos referiremos por ejemplo al fichero javascript como

   ```
   %script{:src => "/public/js/bootstrap.js"}
   ```

   Otro ejemplo:

   ```
   use Rack::Static, :urls => ["/css", "/images"], :root => "public"
   ```

   servirá las peticiones comenzando con `/css` o `/images` desde la carpeta `public` en el directorio actual (esto es `public/css/*` y `public/images/*`)

3. Véase el código en GitHub de Rack::Static

4. En el template `views/index.haml` deberá enlazar a las hojas de estilo:

```
!!!
%html{:lang => "en"}
  %head
    %meta{:charset => "utf-8"}/
    %title RPS
    %link{:href => "/public/css/bootstrap.css", :rel => "stylesheet"}
     %link{:href => "/public/css/bootstrap.css", :rel => "stylesheet"}
```

y las imágenes como:

```
%img(src="/public/img/programming-languages.jpg" width="40%")
```

5. Rack::File es un middleware que sirve los ficheros debajo del directorio dado, de acuerdo con el `path info` de la petición Rack. por ejemplo, cuando se usa `Rack::File.new("/etc")` podremos acceder al fichero `passwd` como `localhost:9292/passwd`.

6. Vease el código en github de Rack::File

7. Para saber mas de Bootstrap véase la sección **??**

16.18. Middleware y la Clase Rack::Builder

We mentioned earlier that between the server and the framework, Rack can be customized to your applications needs using middleware.

The fundamental idea behind Rack middleware is

1. come between the calling client and the server,

2. process the HTTP request before sending it to the server, and

3. processing the HTTP response before returning it to the client.

Motivación para el método `use`

Si tenemos una app Rack `rack_app` y dos middlewares con nombres `MiddleWare1` y `MiddleWare2` que queremos usar, podemos escribir esto:

```
Rack::Handler::Thin.run Middleware1.new(Middleware2.new(rack_app))
```

Si necesitamos pasar opciones en el segundo argumento la llamada quedaría mas o menos como esto:

```
Rack::Handler::Thin.run(
  Middleware1.new(
    Middleware2.new(rack_app, options2),
    options1)
)
```

Si fueran mas de dos middlewares el correspondiente código se volverá aún mas ilegible y hace mas fácil que metamos la pata cuando queramos hacer algo como - por ejemplo -modificar el orden de los middleware.

La Clase Rack::Builder

La clase Rack::Builder implementa un pequeño DSL para facilitar la construcción de aplicaciones Rack.

Rack::Builder is the thing that glues various Rack middlewares and applications together and convert them into a single entity/rack application.

A good analogy is comparing Rack::Builder object with a stack, where at the very bottom is your actual rack application and all middlewares on top of it, and the whole stack itself is a rack application too.

1. El método `use` añade un middleware a la pila
2. El método `run` ejecuta una aplicación
3. El método `map` construye un Rack::URLMap en la forma apropiada. It mounts a stack of rack application/middleware on the specified path or URI.

Conversión de una Aplicación Rack a Rack::Builder
Dada la aplicación:

```
infinity = Proc.new {|env| [200, {"Content-Type" => "text/html"}, env.[inspect]] }
Rack::Handler::Mongrel.run infinity, :Port => 9292
```

Podemos reescribirla:

```
[~/sinatra/rack/rack-builder/map]$ cat app_builder.rb
require 'rack'

infinity = Proc.new {|env| [200, {"Content-Type" => "text/html"}, [env.inspect]]}
builder = Rack::Builder.new
builder.run infinity
Rack::Handler::Thin.run builder, :Port => 9292
```

o bien:

```
[~/sinatra/rack/rack-builder/map]$ cat app_builder2.rb
require 'rack'

infinity = Proc.new {|env| [200, {"Content-Type" => "text/html"}, [ env.inspect ]] }
builder = Rack::Builder.new do
  run infinity
end
Rack::Handler::Thin.run builder, :Port => 9292
```

Ejemplo Simple de Uso de Rack::Builder

```
[~/local/src/ruby/sinatra/rack/rack-builder/simple1]$ cat app.rb
require 'rack'
require 'rack/server'

app = Rack::Builder.new do
  use Rack::CommonLogger
  use Rack::ShowExceptions
    use Rack::Lint
    map "/chuchu" do
      run lambda { |env| [ 200, {}, ["hello"]] }
    end
    map "/chachi" do
      run lambda { |env| [ 200, {}, ["world"]] }
    end
    run lambda { |env| [ 200, {}, ["everything"]] }
end

Rack::Server.start :app => app
```

Ejemplo de Uso de Rack::Builder: Dos Middlewares

```
[~/rack/rack-from-the-beginning(master)]$ cat hello_world.rb
# hello_world.rb
require 'rack'
require 'rack/server'

class EnsureJsonResponse
  def initialize(app = nil)
    @app = app
  end

  # Set the 'Accept' header to 'application/json' no matter what.
  # Hopefully the next middleware respects the accept header :)
  def call(env)
    env['HTTP_ACCEPT'] = 'application/json'
    puts "env['HTTP_ACCEPT'] = #{env['HTTP_ACCEPT']}"
    @app.call(env) if @app
  end
end

class Timer
  def initialize(app = nil)
    @app = app
  end

  def call(env)
    before = Time.now
    status, headers, body = @app.call(env) if @app

    headers['X-Timing'] = (Time.now - before).to_i.to_s

    [status, headers, body]
  end
end

class HelloWorldApp

  def initialize(app = nil)
    @app = app
  end

  def self.call(env)
    [200, {}, ['hello world!']]
  end
end

# put the timer at the top so it captures everything below it
app = Rack::Builder.new do
  use Timer # put the timer at the top so it captures everything below it
  use EnsureJsonResponse
  run HelloWorldApp
end
```

```
Rack::Server.start :app => app

~/rack/rack-from-the-beginning(master)]$  cat Rakefile
desc "run the server"
task :default do
  sh "rackup"
end

desc "run the server hello_world.rb"
task :server do
  sh "ruby hello_world.rb"
end

desc "run the client"
task :client do
  sh %q{curl -v 'http://localhost:9292'}
end

desc "run the client for hello_world"
task :client2 do
  sh %q{curl -v 'http://localhost:8080'}
end

[~/rack/rack-from-the-beginning(master)]$ rake server
ruby hello_world.rb
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:8080, CTRL+C to stop

[~/rack/rack-from-the-beginning(master)]$ rake client2
curl -v 'http://localhost:8080'
* About to connect() to localhost port 8080 (#0)
*   Trying ::1... Connection refused
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:8080
> Accept: */*
>
< HTTP/1.1 200 OK
< X-Timing: 0
< Connection: close
< Server: thin 1.5.1 codename Straight Razor
<
* Closing connection #0
hello world!

[~/rack/rack-from-the-beginning(master)]$ rake server
ruby hello_world.rb
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:8080, CTRL+C to stop
env['HTTP_ACCEPT'] = application/json
```

16.19. Ejemplo de Middleware: Rack::ETag

An `ETag` or *entity tag*, is part of HTTP, the protocol for the World Wide Web. It is one of several mechanisms that HTTP provides for *web cache validation*, and which allows a client to make conditional requests.

This allows caches to be more efficient, and saves bandwidth, as a web server does not need to send a full response if the content has not changed.

An ETag is an opaque identifier assigned by a web server to a specific version of a resource found at a URL.

If the resource content at that URL ever changes, a new and different ETag is assigned.

Used in this manner ETags are similar to fingerprints, and they can be quickly compared to determine if two versions of a resource are the same or not.

1. Rack::ETag en GitHub
2. Documentación de Rack::ETag

```
require 'digest/md5'

module Rack
  class ETag
    DEFAULT_CACHE_CONTROL = "max-age=0, private, must-revalidate".freeze

    def initialize(app, no_cache_control = nil, cache_control = DEFAULT_CACHE_CONTROL)
      @app = app
      @cache_control = cache_control
      @no_cache_control = no_cache_control
    end

    def call(env)
      status, headers, body = @app.call(env)

      if etag_status?(status) && etag_body?(body) && !skip_caching?(headers)
        digest, body = digest_body(body)
        headers['ETag'] = %("#{digest}") if digest
      end

      unless headers['Cache-Control']
        if digest
          headers['Cache-Control'] = @cache_control if @cache_control
        else
          headers['Cache-Control'] = @no_cache_control if @no_cache_control
        end
      end

      [status, headers, body]
    end

    private

      def etag_status?(status)
        status == 200 || status == 201
      end
```

```
    def etag_body?(body)
      !body.respond_to?(:to_path)
    end

    def skip_caching?(headers)
      (headers['Cache-Control'] && headers['Cache-Control'].include?('no-cache')) ||
        headers.key?('ETag') || headers.key?('Last-Modified')
    end

    def digest_body(body)
      parts = []
      digest = nil

      body.each do |part|
        parts << part
        (digest ||= Digest::MD5.new) << part unless part.empty?
      end

      [digest && digest.hexdigest, parts]
    end
  end
end
```

16.20. Construyendo Nuestro Propio Rack::Builder

Véase:

1. https://github.com/crguezl/rack-mybuilder

```
[~/local/src/ruby/sinatra/rack/rack-builder/own(master)]$ cat mybuilder.rb
module Rack
  class MyBuilder

    def initialize(&block)
      @use = []
      instance_eval(&block) if block_given?
    end

    def use(middleware, *args, &block)
      @use << proc { |app| middleware.new(app, *args, &block) }
    end

    def run(app)
      @run = app
    end

    def to_app
      @use.reverse.inject(@run) { |app, middleware| middleware[app] }
    end

    def call(env)
      to_app.call(env)
    end
```

```
  end
end

[~/local/src/ruby/sinatra/rack/rack-builder/own(master)]$ cat decorator.rb
class Decorator

  def initialize(app, *options, &block)
    @app = app
    @options = (options[0] || {})
  end

  def call(env)
    status, headers, body = @app.call(env)

    new_body = ""
    new_body << (@options[:header] || "----Header----<br/>")
    body.each {|str| new_body << str}
    new_body << (@options[:footer] || "<br/>----Footer----")

    [status, headers, [new_body]]
  end
end

[~/local/src/ruby/sinatra/rack/rack-builder/own(master)]$ cat app.rb
require 'rack'
require 'thin'

require 'mybuilder'
require 'decorator'

app = Rack::MyBuilder.new do
  use Decorator, :header => "<strong>*********** header **********</strong><br/>"

  cheer = ARGV.shift || "<h1>Hello world!</h1>"
  run lambda { |env| [200, { 'Content-Type' => 'text/html' }, [ "<h1>#{cheer}</h1>" ]]}
end

Rack::Handler::Thin.run app, :Port => 3333, :Host => 'localhost'

[~/local/src/ruby/sinatra/rack/rack-builder/own(master)]$ cat Rakefile
desc "run app server"
task :default => :server

desc "run app server"
task :server, :greet  do |t, args|
  cheer = args[:greet] || 'bye, bye!'
  sh "ruby -I. app.rb #{cheer}"
end

[~/local/src/ruby/sinatra/rack/rack-builder/own(master)]$ rake -T
rake default        # run app server
rake server[greet]  # run app server

[~/local/src/ruby/sinatra/rack/rack-builder/own(master)]$ rake server[tachaaaAAAAAANNN]
```

```
ruby -I. app.rb tachaaaAAAAAANNN
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on localhost:3333, CTRL+C to stop
```

16.21. Código de Rack::Builder

Tomado de https://github.com/rack/rack/blob/master/lib/rack/builder.rb:

```
module Rack
  # Rack::Builder implements a small DSL to iteratively construct Rack
  # applications.
  #
  # Example:
  #
  #  require 'rack/lobster'
  #  app = Rack::Builder.new do
  #    use Rack::CommonLogger
  #    use Rack::ShowExceptions
  #    map "/lobster" do
  #      use Rack::Lint
  #      run Rack::Lobster.new
  #    end
  #  end
  #
  #  run app
  #
  # Or
  #
  #  app = Rack::Builder.app do
  #    use Rack::CommonLogger
  #    run lambda { |env| [200, {'Content-Type' => 'text/plain'}, ['OK']] }
  #  end
  #
  #  run app
  #
  # +use+ adds middleware to the stack, +run+ dispatches to an application.
  # You can use +map+ to construct a Rack::URLMap in a convenient way.

  class Builder
    def self.parse_file(config, opts = Server::Options.new)
      options = {}
      if config =~ /\.ru$/
        cfgfile = ::File.read(config)
        if cfgfile[/^#\\(.*)/] && opts
          options = opts.parse! $1.split(/\s+/)
        end
        cfgfile.sub!(/^__END__\n.*\Z/m, '')
        app = new_from_string cfgfile, config
      else
        require config
        app = Object.const_get(::File.basename(config, '.rb').capitalize)
      end
      return app, options
```

```
end

def self.new_from_string(builder_script, file="(rackup)")
  eval "Rack::Builder.new {\n" + builder_script + "\n}.to_app",
    TOPLEVEL_BINDING, file, 0
end

def initialize(default_app = nil,&block)
  @use, @map, @run = [], nil, default_app
  instance_eval(&block) if block_given?
end

def self.app(default_app = nil, &block)
  self.new(default_app, &block).to_app
end

# Specifies middleware to use in a stack.
#
#   class Middleware
#     def initialize(app)
#       @app = app
#     end
#
#     def call(env)
#       env["rack.some_header"] = "setting an example"
#       @app.call(env)
#     end
#   end
#
#   use Middleware
#   run lambda { |env| [200, { "Content-Type => "text/plain" }, ["OK"]] }
#
# All requests through to this application will first be processed by the middleware class
# The +call+ method in this example sets an additional environment key which then can be
# referenced in the application if required.
def use(middleware, *args, &block)
  if @map
    mapping, @map = @map, nil
    @use << proc { |app| generate_map app, mapping }
  end
  @use << proc { |app| middleware.new(app, *args, &block) }
end

# Takes an argument that is an object that responds to #call and returns a Rack response.
# The simplest form of this is a lambda object:
#
#   run lambda { |env| [200, { "Content-Type" => "text/plain" }, ["OK"]] }
#
# However this could also be a class:
#
#   class Heartbeat
#     def self.call(env)
#      [200, { "Content-Type" => "text/plain" }, ["OK"]]
```

```
    #     end
    #   end
    #
    #   run Heartbeat
    def run(app)
      @run = app
    end

    # Creates a route within the application.
    #
    #   Rack::Builder.app do
    #     map '/' do
    #       run Heartbeat
    #     end
    #   end
    #
    # The +use+ method can also be used here to specify middleware to run under a specific pat
    #
    #   Rack::Builder.app do
    #     map '/' do
    #       use Middleware
    #       run Heartbeat
    #     end
    #   end
    #
    # This example includes a piece of middleware which will run before requests hit +Heartbea
    #
    def map(path, &block)
      @map ||= {}
      @map[path] = block
    end

    def to_app
      app = @map ? generate_map(@run, @map) : @run
      fail "missing run or map statement" unless app
      @use.reverse.inject(app) { |a,e| e[a] }
    end

    def call(env)
      to_app.call(env)
    end

    private

    def generate_map(default_app, mapping)
      mapped = default_app ? {'/' => default_app} : {}
      mapping.each { |r,b| mapped[r] = self.class.new(default_app, &b) }
      URLMap.new(mapped)
    end
  end
end
```

16.22. Rack::Cascade

Rack::Cascade tries an request on several apps, and returns the first response that is not 404 (or in a list of configurable status codes).

Ejemplo

```
[~/local/src/ruby/sinatra/rack/rack-cascade]$ cat cascade2.ru
statuses = [200,404]
apps = [
  lambda {|env|
    status = statuses.sample
    [status, {}, ["I'm the first app. Status = #{status}\n"]]
  },
  lambda {|env|
    status = statuses.sample
    [status, {}, ["I'm the second app. Status = #{status}\n"]]
  },
  lambda {|env|
    status = statuses.sample
    [status, {}, ["I'm the last app. Status = #{status}\n"]]
  }
]
use Rack::ContentLength
use Rack::ContentType
run Rack::Cascade.new(apps)

[~/local/src/ruby/sinatra/rack/rack-cascade]$ rake client
curl -v 'http://localhost:9292'
* About to connect() to localhost port 9292 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Content-Length: 33
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
I'm the second app. Status = 200
* Connection #0 to host localhost left intact
* Closing connection #0

[~/local/src/ruby/sinatra/rack/rack-cascade]$ rake client
curl -v 'http://localhost:9292'
* About to connect() to localhost port 9292 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
```

```
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Content-Length: 31
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
I'm the last app. Status = 200
* Connection #0 to host localhost left intact
* Closing connection #0

[~/local/src/ruby/sinatra/rack/rack-cascade]$ rake client
curl -v 'http://localhost:9292'
* About to connect() to localhost port 9292 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Content-Length: 32
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
I'm the first app. Status = 200
* Connection #0 to host localhost left intact
* Closing connection #0]
```

Código del Constructor

```
    # File lib/rack/cascade.rb, line 11
    def initialize(apps, catch=404)
      @apps = []; @has_app = {}
      apps.each { |app| add app }

      @catch = {}
      [*catch].each { |status| @catch[status] = true }
    end
```

Código de `call`

```
    # File lib/rack/cascade.rb, line 19
    def call(env)
      result = NotFound

      @apps.each do |app|
        result = app.call(env)
        break unless @catch.include?(result[0].to_i)
      end
```

```

    result
  end
```

16.23. Rack::Mount

1. A *router* is similar to a Rack middleware.
2. The main difference is that it doesn't wrap a single Rack endpoint, but keeps a list of endpoints, just like Rack::Cascade does.
3. Depending on some criteria, usually the requested path, the router will then decide what endpoint to hand the request to.
4. Most routers differ in the way they decide which endpoint to hand the request to.
5. All routers meant for general usage do offer routing based on the path, but how complex their path matching might be varies.
6. While Rack::URLMap only matches prefixes, most other routers allow simple wildcard matching.
7. Both Rack::Mount, which is used by Rails, and Sinatra allow arbitrary matching logic.
8. However, such flexibility comes at a price: Rack::Mount and Sinatra have a routing complexity of O(n), meaning that in the worst-case scenario an incoming request has to be matched against all the defined routes.
9. Rack::Mount is known to produce fast routing, however its API is not meant to be used directly but rather by other libraries, like the Rails routes DSL.

```
[~/local/src/ruby/sinatra/rack/rack-mount]$ cat config.ru
require 'sinatra/base'
require 'rack/mount'

class Foo < Sinatra::Base
  get('/foo') { 'foo' }
  get('/fou') { 'fou' }
end

class Bar < Sinatra::Base
  get('/bar') { 'bar' }
  get('/ba')  { 'ba' }
end

Routes = Rack::Mount::RouteSet.new do |set|
  set.add_route Foo, :path_info => %r{^/fo[ou]$}
  set.add_route Bar, :path_info => %r{^/bar?$}
end

run Routes
```

16.24. Rack::URLMap

Rack::URLMap takes a hash mapping urls or paths to apps, and dispatches accordingly. Support for HTTP/1.1 host names exists if the URLs start with http:// or https://.

Rack::URLMap modifies the SCRIPT_NAME and PATH_INFO such that the part relevant for dispatch is in the SCRIPT_NAME, and the rest in the PATH_INFO. This should be taken care of when you need to reconstruct the URL in order to create links.

Rack::URLMap dispatches in such a way that the longest paths are tried first, since they are most specific.

```
[~/local/src/ruby/sinatra/rack/rack-urlmap(master)]$ cat config.ru
app1 = lambda { |e| [200, {}, ["one\n"]]}
app2 = lambda { |e| [200, {}, ["two\n"]]}
app3 = lambda { |e| [200, {}, ["one + two = three\n"]]}

app = Rack::URLMap.new "/one" => app1, "/two" => app2, "/one/two" => app3

run app

[~/local/src/ruby/sinatra/rack/rack-urlmap(master)]$ cat Rakefile
desc "run the server"
task :default do
  sh "rackup"
end

desc "run the client with one"
task :one do
  sh %q{curl -v 'http://localhost:9292/one'}
end

desc "run the client with two"
task :two do
  sh %q{curl -v 'http://localhost:9292/two'}
end

desc "run the client with one/two"
task :onetwo do
  sh %q{curl -v 'http://localhost:9292/one/two'}
end

[~/local/src/ruby/sinatra/rack/rack-urlmap]$ rake
rackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
127.0.0.1 - - [17/Oct/2013 21:24:48] "GET /two HTTP/1.1" 200 - 0.0006

[~/local/src/ruby/sinatra/rack/rack-urlmap(master)]$ rake onetwo
curl -v 'http://localhost:9292/one/two'
* About to connect() to localhost port 9292 (#0)
*   Trying ::1... Connection refused
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /one/two HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
```

```
< HTTP/1.1 200 OK
< Transfer-Encoding: chunked
< Connection: close
< Server: thin 1.5.1 codename Straight Razor
<
one + two = three
* Closing connection #0
```

16.25. El método run de Rack::Handler::WEBrick

Véa por ejemplo una versión del código de `run` de Rack::Handler::WEBrick: (puede encontrarse una en Rack::Handler::WEBrick):

```
def self.run(app, options={})
  options[:BindAddress] = options.delete(:Host) if options[:Host]
  options[:Port] ||= 8080
  @server = ::WEBrick::HTTPServer.new(options)
  @server.mount "/", Rack::Handler::WEBrick, app
  yield @server  if block_given?
  @server.start
end
```

1. Vemos que `run` espera un objeto `app` que representa la aplicación y un hash de opciones.
2. Si arrancamos un servidor en 127.0.0.1, sólo escucha en localhost; si lo arrancamos en 0.0.0.0, escucha a cualquier IP, en particular en nuestra IP local.

   Veamos el siguiente experimento:

   ```
   [~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ cat bindaddress0000.rb
   require 'rack'

   #ENV['RACK-ENV'] = 'production'

   app = lambda { |e|
    [200, { 'content-type' => 'text/html'}, ["<h1>hello world!</h1>"]]
    }

    Rack::Handler::WEBrick.run app, { :Host => '0.0.0.0' }

   [~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ ifconfig  en0 | grep 'in
     inet 192.168.0.103

   [~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ ruby bindaddress0000.rb
   [2013-09-23 12:04:36] INFO  WEBrick 1.3.1
   [2013-09-23 12:04:36] INFO  ruby 1.9.3 (2013-02-22) [x86_64-darwin11.4.2]
   [2013-09-23 12:04:36] INFO  WEBrick::HTTPServer#start: pid=8720 port=8080

   [~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ curl -v 'http://192.168
   *   Trying 192.168.0.103... connected
   * Connected to 192.168.0.103 (192.168.0.103) port 8080 (#0)
   > GET / HTTP/1.1
   > User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/
   ```

```
> Host: 192.168.0.103:8080
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Server: WEBrick/1.3.1 (Ruby/1.9.3/2013-02-22)
< Date: Mon, 23 Sep 2013 11:11:40 GMT
< Content-Length: 21
< Connection: Keep-Alive
<
* Connection #0 to host 192.168.0.103 left intact
* Closing connection #0
<h1>hello world!</h1>

 [~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ cat bindaddress127001.r
 require 'rack'

 #ENV['RACK-ENV'] = 'production'

 app = lambda { |e|
  [200, { 'content-type' => 'text/html'}, ["<h1>hello world!</h1>"]]
  }

  Rack::Handler::WEBrick.run app, { :Host => '127.0.0.1' }

[~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ ruby bindaddress127001.r
[2013-09-23 12:13:07] INFO  WEBrick 1.3.1
[2013-09-23 12:13:07] INFO  ruby 1.9.3 (2013-02-22) [x86_64-darwin11.4.2]
[2013-09-23 12:13:07] INFO  WEBrick::HTTPServer#start: pid=8993 port=8080

[~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ curl -v 'http://192.168
* About to connect() to 192.168.0.103 port 8080 (#0)
*   Trying 192.168.0.103... Connection refused
* couldn't connect to host
* Closing connection #0
curl: (7) couldn't connect to host
```

3. Luego se crea un nuevo objeto que representa al servidor con `@server = ::WEBrick::HTTPServer.new(opti`

   Esto crea un nuevo objeto WEBrick HTTP server de acuerdo a `options`. Un servidor HTTP tiene los siguientes atributos:

   a) `AccessLog`: An array of access logs. See WEBrick::AccessLog
   b) `BindAddress`: Local address for the server to bind to
   c) `DocumentRoot`: Root path to serve files from
   d) `DocumentRootOptions`: Options for the default HTTPServlet::FileHandler
   e) `HTTPVersion`: The HTTP version of this server
   f) `Port`: Port to listen on
   g) `RequestCallback`: Called with a request and response before each request is serviced.
   h) `RequestTimeout`: Maximum time to wait between requests
   i) `ServerAlias`: Array of alternate names for this server for virtual hosting

j) `ServerName`: Name for this server for virtual hosting

4. `mount` recibe un directorio y un servlet. Un servlet es una clase que se usa para extender las capacidades de un servidor. En este caso estamos extendiendo `@server` que es un servidor `WEBrick::HTTPServer` con las capacidades definidas en la clase `Rack::Handler::WEBrick`. La sintáxis de `mount` es:

   ```
   mount(dir, servlet, *options)
   ```

   Las opciones son pasadas al servlet en el momento de la creación del servlet.

5. Observamos que `run` puede ir seguido de un bloque al que se le pasa como argumento el objeto server

   ```
   yield @server  if block_given?
   ```

   Este bloque puede ser usado como una nueva oportunidad para configurar el server

6. Se arranca el servidor con la llamada al método `start` definido en `webrick/server.rb`

16.26. Documentación

- rack documentación

16.27. Pruebas/Testing

16.27.1. Pruebas Unitarias

1. Los fuentes de este ejemplo están en https://github.com/crguezl/rack-unit-test

2. Fuentes en GitHub de Rack::Test: https://github.com/brynary/rack-test

```
[~/rack/rack-unit-test(master)]$ cat rack_hello_world.rb
# my_app.rb
#
require 'rack'

class MyApp
  def call env
    [200, {"Content-Type" => "text/html"}, ["Hello"]]
  end
end

[~/rack/rack-unit-test(master)]$ cat test_hello_world.rb
require "test/unit"
require "rack/test"
require './rack_hello_world'

class AppTest < Test::Unit::TestCase
  include Rack::Test::Methods

  def app
    Rack::Builder.new do
      run MyApp.new
    end.to_app
```

```
  end

  def test_index
    get "/"
    #puts last_response.inspect
    assert last_response.ok?
  end

  def test_body
    get "/"
    assert_equal last_response.body,  'Hello', "body must be hello"
  end
end
```

1. The Rack::Test::Methods module serves as the primary integration point for using `Rack::Test` in a testing environment.

   It depends on an `app` method being defined in the same context,

   ```
   def app
     Rack::Builder.new do
       run MyApp.new
     end.to_app
   end
   ```

   and provides the `Rack::Test` API methods (see Rack::Test::Session for their documentation).

2. The `get` method issue a `GET` request for the given URI. Stores the issues request object in `#last_request` and the app's response in `#last_response` (whose class is Rack::MockResponse) .

   Yield `#last_response` to a block if given.

   ```
   def test_index
     get "/"
     assert last_response.ok?
   end
   ```

3. Otros métodos que se pueden usar son:

   a) `(Object) basic_authorize(username, password) (also: #authorize)` Set the username and password for `HTTP` Basic authorization, to be included in subsequent requests in the `HTTP_AUTHORIZATION` header.

   b) `(Object) delete(uri, params = {}, env = {}, &block)` Issue a `DELETE` request for the given URI.

   c) `(Object) digest_authorize(username, password)` Set the username and password for `HTTP` Digest authorization, to be included in subsequent requests in the `HTTP_AUTHORIZATION` header.

   d) `(Object) env(name, value)` Set an env var to be included on all subsequent requests through the session.

   e) `(Object) follow_redirect!` Rack::Test will not follow any redirects automatically.

   f) `(Object) get(uri, params = {}, env = {}, &block)` Issue a `GET` request for the given `URI` with the given params and Rack environment.

   g) `(Object) head(uri, params = {}, env = {}, &block)` Issue a `HEAD` request for the given URI.

h) `(Object) header(name, value)` Set a header to be included on all subsequent requests through the session.

i) `(Session) initialize(mock_session) constructor` Creates a Rack::Test::Session for a given Rack app or Rack::MockSession.

j) `(Object) options(uri, params = {}, env = {}, &block)` Issue an `OPTIONS` request for the given URI.

k) `(Object) patch(uri, params = {}, env = {}, &block)` Issue a `PATCH` request for the given URI.

l) `(Object) post(uri, params = {}, env = {}, &block)` Issue a `POST` request for the given URI.

m) `(Object) put(uri, params = {}, env = {}, &block)` Issue a `PUT` request for the given URI.

n) `(Object) request(uri, env = {}, &block)` Issue a request to the Rack app for the given `URI` and optional Rack environment.

4. The `#last_response` object has methods:

```
=~(other) body() empty?() match(other)
```

and attributes:

```
errors          [RW]
original_headers[R]
Headers
```

5. Si se usan middleware adicionales es necesario especificarlo en `app`:

```
def app
  Rack::Builder.new do
    use(Rack::Session::Cookie, {:key => 'rack session',·
                                #:domain => 'localhost',
                                #:path => '/', #:expire_after => 2592000,·
                                :secret => 'change_me'})
    run RockPaperScissors::App.new
  end.to_app
end
```

6. El método `last_response.body` returns the last response received in the session. Raises an error if no requests have been sent yet.

```
[~/rack/rack-unit-test(master)]$ cat Rakefile
task :default => :test
desc "run the tests"
task :test do
  sh "ruby test_hello_world.rb"
end

[~/rack/rack-unit-test(master)]$ cat Gemfile
source 'https://rubygems.org'

gem 'rack'
gem 'rack-test'
```

```
[~/rack/rack-unit-test(master)]$ rake
ruby test_hello_world.rb
Run options:

# Running tests:

..

Finished tests in 0.015253s, 131.1217 tests/s, 131.1217 assertions/s.

2 tests, 2 assertions, 0 failures, 0 errors, 0 skips
```

16.27.2. Rspec con Rack

Véase

1. Los fuentes de este ejemplo están en: https://github.com/crguezl/rack-rspec
2. Using RSpec with Rack en Youtube por Mike Bethany
3. Documentación en rubydoc.info del módulo Rack::MockSession http://rdoc.info/github/brynary/rack-test/m
4. Código fuente en lib/rack/mock.rb
5. Documentación en rubydoc.info del módulo Rack::Test::Methods: http://rdoc.info/github/brynary/rack-test
6. Documentación de Rack::Test::Session
7. webmock gem
8. Class: Rack::MockRequest documentation
9. How to Test Sinatra-Based Web Services by Harlow Ward, March 17, 2013 Webmock Written by thoughtbot Harlow Ward March 17, 2013

Jerarquía

```
[~/rack/rack-rspec(master)]$ tree
.
|--- Gemfile
|--- Gemfile.lock
|--- README
|--- Rakefile
|--- lib
|   |--- rsack
|   |   `--- server.rb
|   `--- rsack.rb
`--- spec
    |--- rsack
    |   `--- server_spec.rb
    `--- spec_helper.rb

4 directories, 8 files
```

lib/rsack.rb

```
[~/rack/rack-rspec(master)]$ cat lib/rsack.rb
require 'rack'
require 'rsack/server'
```

lib/rsack/server.rb

```
[~/rack/rack-rspec(master)]$ cat lib/rsack/server.rb
module Rsack
  class Server
    def call(env)
      #["200", {}, "hello"]
      response = Rack::Response.new
      response.write("Hello world!")
      response.finish
    end
  end
end
```

spec/rsack/server_spec.rb

```
[~/rack/rack-rspec(master)]$ cat spec/rsack/server_spec.rb
require 'spec_helper'

describe Rsack::Server do

  #let(:server) { Rack::MockRequest.new(Rsack::Server.new) }
  def server
    Rack::MockRequest.new(Rsack::Server.new)
  end

  context '/' do
    it "should return a 200 code" do
      response = server.get('/')
      response.status.should == 200
    end
  end
end
```

Rack::MockRequest helps testing your Rack application without actually using HTTP.

```
    Rack::MockRequest.new(Rsack::Server.new)
```

After performing a request on a URL `response = server.get('/')` with `get/post/put/patch/delete`, it returns a MockResponse with useful helper methods for effective testing (Véase el código de MockResponse en Github en el fichero lib/rack/mock.rb).

Un objeto MockResponse dispone de los métodos:

```
=~   []   match   new
```

y de los atributos:

```
body   [R]   Body
errors   [RW]   Errors
headers   [R]   Headers
original_headers   [R]   Headers
status   [R]   Status
```

Si se usan middleware adicionales es necesario especificarlo en `server`. Por ejemplo:

```
 Rack::MockRequest.new(Rack::Session::Cookie.new(RockPaperScissors::App.new,
                                                 :secret =>'cookie'))
```

spec/spec_helper.rb

```
[~/rack/rack-rspec(master)]$ cat spec/spec_helper.rb
$:.unshift File.expand_path(File.dirname(__FILE__)+'../lib')
$:.unshift File.dirname(__FILE__)

#puts $:.inspect

require 'rspec'
require 'rack'

require 'rsack'
```

Rakefile

```
[~/rack/rack-rspec(master)]$ cat Rakefile
desc "run rspec tests"
task :default do
  sh "rspec spec/rsack/server_spec.rb"
end
```

Gemfile

```
[~/rack/rack-rspec(master)]$ cat Gemfile
# A sample Gemfile
source "https://rubygems.org"

gem 'rack'

group :development, :test do
  gem 'rspec'
end
```

16.28. Práctica: Añada Pruebas a Rock, Paper,Scissors

Complete la practica realizada en la sección *Añada Hojas de Estilo a Piedra Papel Tijeras* 16.17.3 con:

1. Pruebas unitarias (Vea la sección *Pruebas Unitarias* 16.27.1
2. *Desarrollo Dirigido por las Pruebas TDD* (Vea la sección *Rspec con Rack* 16.27.2)
3. Cree una sesión de manera que la aplicación disponga de contadores que lleven el número de partidas jugadas y el número de partidas ganadas por el jugador (Vea las secciones *Gestión de Sesiones* 16.9 y *Cookies* 16.8)

16.29. Prácticas: Centro de Cálculo

1. Modo de trabajo en el sistema de archivos del CC de la ETSII
2. Ubicación de las salas

16.30. Despliegue de una Aplicación Web en la ETSII

Para desplegar una aplicación web usaremos `exthost2` (en 2013).
Veamos que puertos están libres usando `netstat` :

```
casiano@exthost2:~$ netstat -an | less
```

o bien usamos `lsof` :

```
lsof -i | less
```

y después - si es necesario - terminamos el proceso que ya estuviera escuchando en el puerto

```
kill -9 PID
```

Veamos una simple aplicación usando `rack`:

```
casiano@exthost2:~/src/ruby/simplewebapp$ cat hello.rb
require 'rack'

app = lambda { |env| [200, {"Content-Type" => "text/plain"}, ["Hello. The time is #{Time.now}"
Rack::Handler::WEBrick.run app,:Port => 4567
```

La ejecutamos:

```
casiano@exthost2:~/src/ruby/simplewebapp$ ruby hello.rb
[2013-10-28 09:58:54] INFO  WEBrick 1.3.1
[2013-10-28 09:58:54] INFO  ruby 1.9.3 (2011-10-30) [i686-linux]
[2013-10-28 09:58:54] WARN  TCPServer Error: Address already in use - bind(2)
[2013-10-28 09:58:54] INFO  WEBrick::HTTPServer#start: pid=16597 port=4567
```

Ya tenemos disponible la página en `exthost2` en el puerto correspondiente.
El acceso al servidor está limitado a la red de la ULL.

Véase también

1. *Gemas instaladas en local* **??**

16.31. Práctica: Despliegue en Heroku su Aplicación Rock, Paper,Scissors

Despliegue en Heroku la practica realizada en la sección *Añada Pruebas a Rock, Paper,Scissors* 16.28. Repase la sección *Despliegue en Heroku* **??**

16.32. Faking Sinatra with Rack and Middleware

1. Faking Sinatra with Rack and Middleware por Charles Max Wood (Vimeo)
2. crguezl/rack-sinatra-in-5-minutes en GitHub
3. Noah Gibbs Ruby Hangout

16.33. Véase También

- ArrrrCamp 2013 - Web applications with Ruby (not Rails) YouTube David Padilla
- A Quick Introduction to Rack
- Writing modular web applications with Rack
- Rackup Wiki
- Rack from the Beginning por Adam Hawkins (github: https://github.com/crguezl/rack-from-the-beginning)
- Understanding Rack de Tekpub Productions (Vimeo)
- Media Test: Rack Middleware on Any Framework por Noah Gibbs (YouTube)
- Rails Online Conf: Rack in Rails 3 por Ryan Tomayko (Youtube)
- 32 Rack Resources to Get You Started por Jason Seifer
- The Little Rack Book
- Rack Developer's Notebook
- Rails Conf 2013 You've got a Sinatra on your Rails by José Valim
- The Web Server Gateway Interface is a simple and universal interface between web servers and web applications or frameworks for the Python programming language. Rack is inspired in WSGI

Capítulo 17

Primeros Pasos

17.1. Introducción

17.1.1. Referencias sobre Sinatra

- Ruby/Sinatra Class Page
- Sinatra introduction de Ben Schwarz
- How to create a Twilio app on Heroku de Morten Baga
- ArrrrCamp #6 - Aleksander Dabrowski - Sinatra autopsy Vimeo

Referencias sobre Rack:

- Rackup Wiki
- Understanding Rack de Tekpub Productions

17.1.2. Ejercicio: Instale la Documentación en sinatra.github.com

Instale la documentación de Sinatra en https://github.com/sinatra/sinatra.github.com

A la hora de empezar la jerarquía de una aplicación sinatra se puede seguir la estructura propuesta por Lee Martin en sinatra-stack

Capítulo 18

Fundamentos

18.1. Ejemplo Simple de uso de Sinatra

Código

```
[~/sinatra/sinatra-simple(master)]$ cat hi.rb
require 'sinatra'

get '/hi' do
  "Hello World!"
end
```

Opciones de Ejecución

```
[~/sinatra/sinatra-simple(master)]$ ruby hi.rb --help
Usage: hi [options]
    -p port                         set the port (default is 4567)
    -o addr                         set the host (default is localhost)
    -e env                          set the environment (default is development)
    -s server                       specify rack server/handler (default is thin)
    -x                              turn on the mutex lock (default is off)
```

Sinatra can be used in threaded environments where more than a single request is processed at a time.

However, not all applications and libraries are thread-safe and may cause intermittent errors or general weirdness.

Enabling the `-x` setting causes all requests to synchronize on a mutex lock, ensuring that only a single request is processed at a time.

The mutex lock setting is disabled by default.

18.2. Rutas/Routes

Repase la sección *HTTP* 16.5.

Vease el código de este ejemplo en GitHub

Aplicacion

```
[~/Dropbox/src/ruby/sinatra/sinatra-simple(master)]$ cat app.rb
require 'sinatra/base'

class App < Sinatra::Base
  get '/' do
    "hello get!"
```

```
  end

  post '/' do
    'hello post!'
  end

  put '/' do
    'hello put!'
  end

  delete '/' do
    'hello delete!'
  end

  get '/:name' do |name|
    "hello #{name}!"
  end

  get '/:name/?:apelido1?' do |name, apellido|
    "hello #{apellido}, #{name}!"
  end
end
```

In Sinatra, a route is an HTTP method paired with a URL-matching pattern. Each route is associated with a block

Routes are matched in the order they are defined. The first route that matches the request is invoked.

Route patterns may include named parameters, accessible via the params hash:

```
get '/hello/:name' do
  # matches "GET /hello/foo" and "GET /hello/bar"
  # params[:name] is 'foo' or 'bar'
  "Hello #{params[:name]}!"
end
```

You can also access named parameters via block parameters:

```
get '/:name' do |name|
  "hello #{name}!"
end
```

Route patterns may also include splat (or wildcard) parameters, accessible via the `params[:splat]` array:

```
get '/say/*/to/*' do
  # matches /say/hello/to/world
  params[:splat] # => ["hello", "world"]
end

get '/download/*.*' do
  # matches /download/path/to/file.xml
  params[:splat] # => ["path/to/file", "xml"]
end
```

config.ru

```
[~/sinatra/sinatra-simple(master)]$ cat config.ru
require './app'

run App
```

Rakefile

```
[~/sinatra/sinatra-simple(master)]$ cat Rakefile
task :default => :server

desc "run server"
task :server do
  sh "rackup"
end

desc "make a get / request via curl"
task :get do
  sh "curl -v localhost:9292"
end

desc "make a post / request via curl"
task :post do
  sh "curl -X POST -v -d 'ignored data' localhost:9292"
end

desc "make a put / request via curl"
task :put do
  sh "curl -X PUT -v localhost:9292"
end

desc "make a DELETE / request via curl"
task :delete do
  sh "curl -X DELETE -v localhost:9292"
end

desc "make a get /name request via curl"
task :getname, :name do |t,h|
  name = h[:name] or 'pepe'
  sh "curl -v localhost:9292/#{name}"
end

desc "make a get /name/appellido request via curl"
task :getfullname, :name, :apellido do |t,h|
  name = h[:name] or 'pepe'
  apellido = h[:apellido] or 'rodriguez'
  sh "curl -v localhost:9292/#{name}/#{apellido}"
end

task :html do
  sh "kramdown README.md > README.html"
end
```

Ejecución del servidor

```
[~/sinatra/sinatra-simple(master)]$ rake server
rackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
127.0.0.1 - - [01/Jul/2013 20:25:16] "GET /juana HTTP/1.1" 200 12 0.0689
```

Ejecución de los clientes

```
[~/Dropbox/src/ruby/sinatra/sinatra-simple(master)]$ rake getname[juana]
{:name=>"juana"}
curl -v localhost:9292/juana
* About to connect() to localhost port 9292 (#0)
*   Trying ::1... Connection refused
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /juana HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8x zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html;charset=utf-8
< Content-Length: 12
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
* Connection #0 to host localhost left intact
* Closing connection #0
hello juana!

[~/Dropbox/src/ruby/sinatra/sinatra-simple(master)]$ rake getfullname[Ana,Hernandez]
curl -v localhost:9292/Ana/Hernandez
* About to connect() to localhost port 9292 (#0)
*   Trying ::1... Connection refused
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /Ana/Hernandez HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8x zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html;charset=utf-8
< Content-Length: 21
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Connection: keep-alive
```

```
< Server: thin 1.5.1 codename Straight Razor
<
* Connection #0 to host localhost left intact
* Closing connection #0
hello Hernandez, Ana!
```

18.2.1. Verbos HTTP en Sinatra/Base

Método `get`

```
def get(path, opts = {}, &block)
  conditions = @conditions.dup
  route('GET', path, opts, &block)

  @conditions = conditions
  route('HEAD', path, opts, &block)
end
```

```
def put(path, opts = {}, &bk)     route 'PUT',     path, opts, &bk end
def post(path, opts = {}, &bk)    route 'POST',    path, opts, &bk end
def delete(path, opts = {}, &bk)  route 'DELETE',  path, opts, &bk end
def head(path, opts = {}, &bk)    route 'HEAD',    path, opts, &bk end
def options(path, opts = {}, &bk) route 'OPTIONS', path, opts, &bk end
def patch(path, opts = {}, &bk)   route 'PATCH',   path, opts, &bk end
def link(path, opts = {}, &bk)    route 'LINK',    path, opts, &bk end
def unlink(path, opts = {}, &bk)  route 'UNLINK',  path, opts, &bk end
```

Método `route`

```
def route(verb, path, options = {}, &block)
  # Because of self.options.host
  host_name(options.delete(:host)) if options.key?(:host)
  enable :empty_path_info if path == "" and empty_path_info.nil?
  signature = compile!(verb, path, block, options)
  (@routes[verb] ||= []) << signature
  invoke_hook(:route_added, verb, path, block)
  signature
end
```

18.3. Ficheros Estáticos

1. Static files are served from the `./public` directory.
2. You can specify a different location by setting the `:public_folder` option:

   ```
   set :public_folder, File.dirname(__FILE__) + '/static'
   ```

   Put this code in a `configure` block
3. Note that the public directory name is not included in the URL.
4. A file `./public/css/style.css` is made available as `http://example.com/css/style.css`.
5. Use the `:static_cache_control` setting to add Cache-Control header info. Use an explicit array when setting multiple values:

   ```
   set :static_cache_control, [:public, :max_age => 300]
   ```
6. What would be delivered in the event that a defined route conflicts with the name of the static resource?. The answer is the static resource.

18.4. Vistas

> Writing a program that spits out HTML is often more difficult than you might imagine. Although programming languages are better at creating text than they used to be (some of us remember character handling in Fortran and standard Pascal), creating and concatenating string constructs is still painful. If there isn't much to do, it isn't too bad, but a whole HTML page is a lot of text manipulation.
>
> With static HTML pages - those that don't change from request to request - you can use nice WYSIWG editors. Even those of us who like raw text editors find it easier to just type in the text and tags rather than fiddle with string concatenation in a programming language.
>
> Of course the issue is with dynamic Web pages - those that take the results of something like database queries and embed them into the HTML. The page looks different with each result, and as a result regular HTML editors aren't up to the job.
>
> The best way to work is to compose the dynamic Web page as you do a static page but put in markers that can be resolved into calls to gather dynamic information. Since the static part of the page acts as a template for the particular response, I call this a *Template View*.
>
> Martin Fowler

Views in Sinatra are *HTML templates* that can optionally contain data passed from the application. There are two ways to work with views in Sinatra: *inline templates* and *external templates*. Véase en GitHub sinatra-up-and-running/tree/master/chapter2/views.

18.4.1. Templates Inline

Templates may be defined at the end of the source file. En este ejemplo trabajamos con varios templates inline en diferentes ficheros:

```
[~/sinatra/sinatraupandrunning/chapter2/views(master)]$ cat example2-14.rb
require 'sinatra/base'

class App < Sinatra::Base
  enable :inline_templates
  get '/index' do
    puts "Visiting #{request.url}"
    erb :index
  end
end

require './another'
__END__
@@index
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Inline template</title>
  </head>
  <body>
    <h1>Worked!</h1>
  </body>
</html>
```

En este fichero tenemos un segundo template inline:

```
[~/sinatra/sinatraupandrunning/chapter2/views(master)]$ cat another.rb
class App
  enable :inline_templates
  get '/' do
    erb :another
  end
end

__END__
@@another
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Separated file</title>
  </head>
  <body>
    <h1>Inside another!</h1>
  </body>
</html>
```

Este es nuestro config.ru:

```
[~/sinatra/sinatraupandrunning/chapter2/views(master)]$ cat config.ru
require './example2-14'

run App
```

Para simplificar las cosas hemos hecho un `Rakefile`:

```
[~/sinatra/sinatraupandrunning/chapter2/views(master)]$ cat Rakefile
task :default => :server

desc "run server"
task :server do
  sh "rackup"
end

desc "make a get / request via curl"
task :root do
  sh "curl -v localhost:9292"
end

desc "make a get /index request via curl"
task :index do
  sh "curl -v localhost:9292/index"
end
```

El resultado de la ejecución es:

```
[~/sinatra/sinatra-up-and-running/chapter2/views/inline_templates(master)]$ curl http://localh
<!DOCTYPE html>
<html>
  <head>
```

```
    <meta charset="UTF-8">
    <title>Inline template</title>
  </head>
  <body>
    <h1>Worked!</h1>
  </body>
</html>
[~/sinatra/sinatra-up-and-running/chapter2/views/inline_templates(master)]$ curl http://localh
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Separated file</title>
  </head>
  <body>
    <h1>Inside another!</h1>
  </body>
</html>
```

18.4.2. Named Templates

Templates may also be defined using the top-level `template` method:

```
template :layout do
  "%html\n  =yield\n"
end

template :index do
  '%div.title Hello World!'
end

get '/' do
  haml :index
end
```

If a template named `layout` exists, it will be used each time a template is rendered.

You can individually disable layouts by passing `:layout => false` or disable them by default via `set :haml, :layout => false`:

```
get '/' do
  haml :index, :layout => !request.xhr?
end
```

18.4.3. Templates Externos

```
$ ls
Rakefile               example2-16.rb         views
config.ru

$ cat example2-16.rb
require 'sinatra/base'

class App < Sinatra::Base
  get '/index' do
    puts "Visiting #{request.url}"
```

```
    erb :index
  end
end

$ cat views/index.erb
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Inline template</title>
  </head>
  <body>
    <h1>Worked!</h1>
  </body>
</html>

$ cat config.ru
require './example2-16'

run App

$ cat Rakefile
task :default => :server

desc "run server"
task :server do
  sh "rackup"
end

desc "make a get / request via curl"
task :root do
  sh "curl -v localhost:9292"
end

desc "make a get /index request via curl"
task :index do
  sh "curl -v localhost:9292/index"
end

$ rake server
rackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
Visiting http://localhost:9292/index
127.0.0.1 - - [03/Jul/2013 22:30:16] "GET /index HTTP/1.1" 200 157 0.0774

$ rake index
curl -v localhost:9292/index
* About to connect() to localhost port 9292 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /index HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8x zlib/1.2.
```

```
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html;charset=utf-8
< Content-Length: 157
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Inline template</title>
  </head>
  <body>
    <h1>Worked!</h1>
  </body>
</html>

* Connection #0 to host localhost left intact
* Closing connection #0
```

18.4.4. Templates Externos en Subcarpetas

Véase en GitHub sinatra-up-and-running/tree/master/chapter2/views/external_view_files/external_in_subfolde

```
$ ls
Rakefile  app.rb    config.ru views

$ cat app.rb
require 'sinatra/base'

class App < Sinatra::Base
  get '/:user/profile' do |user|
    @user = user
    erb '/user/profile'.to_sym
  end

  get '/:user/help' do |user|
    @user = user
    erb :'/user/help'
  end
end

$ cat views/user/profile.erb
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Profile Template</title>
```

```
  </head>
  <body>
    <h1>Profile of <%= @user %></h1>
     <%= params %>
  </body>
</html>

$ cat views/user/help.erb
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>HELP Template</title>
  </head>
  <body>
    <h1>Help for user <%= @user %></h1>
     <pre>
       <%= params %>
     </pre>
  </body>
</html>

$ cat config.ru
require './app'

run App

$ cat Rakefile
PORT = 9292
task :default => :server

desc "run server"
task :server do
  sh "rackup"
end

desc "make a get /pepe/profile request via curl"
task :profile, :name do |t, h|
  user = h['name'] || 'pepe'
  sh "curl -v localhost:#{PORT}/#{user}/profile"
end

desc "make a get /pepe/help request via curl"
task :help do
  sh "curl -v localhost:#{PORT}/pepe/help"
end

$ rake server
rackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
127.0.0.1 - - [03/Jul/2013 21:40:04] "GET /Pedro/profile HTTP/1.1" 200 227 0.1077
```

```
$ rake profile[Pedro]
curl -v localhost:9292/Pedro/profile
* About to connect() to localhost port 9292 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /Pedro/profile HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8x zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html;charset=utf-8
< Content-Length: 227
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Profile Template</title>
  </head>
  <body>
    <h1>Profile of Pedro</h1>
     {"splat"=>[], "captures"=>["Pedro"], "user"=>"Pedro"}
  </body>
</html>


* Connection #0 to host localhost left intact
* Closing connection #0
```

18.4.5. Variables en las Vistas

Comunicación vía variables de instancia Los templates se evaluan en el mismo contexto que los manejadores de las rutas. Las variables de instancia son accesibles directamente en los templates.

```
get '/:id' do
  @foo = Foo.find(params[:id])
  haml '%h1= @foo.name'
end
```

Veamos un ejemplo de comunicación via variables de instancia entre el manejador de la ruta y el template:

```
[~/sinatra/sinatra-views/passing_data_into_views(master)]$ ls
Rakefile        config.ru       via_instance.rb

[~/sinatra/sinatra-views/passing_data_into_views(master)]$ cat via_instance.rb
require 'sinatra/base'
```

```
class App < Sinatra::Base
  get '/*' do |name|
    def some_template
       <<-'HAMLTEMP'
%ol
  - @foo.each do |item|
    %li
      %i #{item}
HAMLTEMP
    end

    puts "*---***#{name}*---****"
    @foo = name.split('/')
    haml some_template
  end
end

[~/sinatra/sinatra-views/passing_data_into_views(master)]$ cat config.ru
require './via_instance'

run App

[~/sinatra/sinatra-views/passing_data_into_views(master)]$ cat Rakefile
task :default => :server

desc "run server"
task :server do
  sh "rackup"
end

desc "make a get /juan/leon/hernandez request via curl"
task :client do
  sh "curl -v localhost:9292/juan/leon/hernandez"
end

[~/sinatra/sinatraupandrunning/chapter2/views/passing_data_into_views(master)]$ rake server
rackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
*---***juan/leon/hernandez*---****
127.0.0.1 - - [05/Jul/2013 17:06:05] "GET /juan/leon/hernandez HTTP/1.1" 200 109 0.3502

[~/sinatra/sinatra-views/passing_data_into_views(master)]$ rake client
curl -v localhost:9292/juan/leon/hernandez
* About to connect() to localhost port 9292 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /juan/leon/hernandez HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8x zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
```

```
< Content-Type: text/html;charset=utf-8
< Content-Length: 109
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
<ol>
  <li>
    <i>juan</i>
  </li>
  <li>
    <i>leon</i>
  </li>
  <li>
    <i>hernandez</i>
  </li>
</ol>
* Connection #0 to host localhost left intact
* Closing connection #0
```

18.4.6. Pasando variables a la vista explícitamente via un hash

También es posible pasar en la llamada un hash especificando las variables locales:

```
get '/:id' do
  foo = Foo.find(params[:id])
  haml '%h1= bar.name', :locals => { :bar => foo }
end
```

This is typically used when rendering templates as partials from within other templates.

Veamos un ejemplo:

```
$ ls
Rakefile    config.ru   via_hash.rb views

$ cat via_hash.rb
require 'sinatra/base'

class App < Sinatra::Base
  get '/*' do |name|
    def some_template
       <<-'ERBTEMP'
<ul><% name.each do |item| %>
      <li> <i> <%= item %> </i> </li>
    <% end %>
</ul>
ERBTEMP
    end # method some_template

    puts "*---***#{name}*---****"
    erb some_template, :locals => { :name => name.split('/')}
  end
end
```

```
$ cat views/layout.erb
<!DOCTYPE html>
<html>
  <head>
        <title>Sinatra</title>
  </head>
  <body>
    <h1>Accesing variables in templates via a parameter hash</h1>
    <%= yield %>
  </body>
</html>

$ cat config.ru
require './via_hash'

run App

$ cat Rakefile task :default => :server

desc "run server"
task :server do
  sh "rackup"
end

desc "make a get /juan/leon/hernandez request via curl"
task :client do
  sh "curl -v localhost:9292/juan/leon/hernandez"
end

$ rake serverrackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
*---***juan/leon/hernandez*---****
127.0.0.1 - - [05/Jul/2013 17:50:20] "GET /juan/leon/hernandez HTTP/1.1" 200 290 0.0352

$ rake client
curl -v localhost:9292/juan/leon/hernandez
* About to connect() to localhost port 9292 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /juan/leon/hernandez HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8x zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html;charset=utf-8
< Content-Length: 290
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
```

```
<
<!DOCTYPE html>
<html>
  <head>
        <title>Sinatra</title>
  </head>
  <body>
    <h1>Accesing variables in templates via a parameter hash</h1>
    <ul>
      <li> <i> juan </i> </li>

      <li> <i> leon </i> </li>

      <li> <i> hernandez </i> </li>

</ul>

  </body>
</html>
* Connection #0 to host localhost left intact
* Closing connection #0
```

18.4.7. Opciones pasadas a los Métodos de los Templates

Options passed to the render method override options set via `set`.
Available Options:

1. `locals`

   List of locals passed to the document. Handy with partials. Example:

   ```
   erb "<%= foo %>", :locals => {:foo => "bar"}
   ```

2. `default_encoding`

   String encoding to use if uncertain. Defaults to

   ```
   settings.default_encoding.
   ```

3. `views`

   Views folder to load templates from. Defaults to `settings.views`.

4. `layout`

   Whether to use a layout (true or false), if it's a Symbol, specifies what template to use. Example:

   ```
   erb :index, :layout => !request.xhr?
   ```

5. `content_type`

   Content-Type the template produces, default depends on template language.

6. `scope`

   Scope to render template under.

   Defaults to the application instance.

   If you change this, instance variables and helper methods will not be available.

7. `layout_engine`

   Template engine to use for rendering the layout.

   Useful for languages that do not support layouts otherwise.

   Defaults to the engine used for the template. Example:

   ```
   set :rdoc, :layout_engine => :erb
   ```

8. `layout_options`

   Special options only used for rendering the layout. Example:

   ```
   set :rdoc, :layout_options => { :views => 'views/layouts' }
   ```

9. Templates are assumed to be located directly under the `./views` directory.

   To use a different views directory:

   ```
   set :views, settings.root + '/templates'
   ```

10. One important thing to remember is that you always have to reference templates with symbols, even if they're in a subdirectory (in this case, use: `:'subdir/template'` or `'subdir/template'.to_sym`).

    You must use a symbol because otherwise rendering methods will render any strings passed to them directly.

18.5. Filtros

Before Filters *Before filters* are evaluated before each request within the same context as the routes will be and can modify the request and response.

Instance variables set in filters are accessible by routes and templates:

```
before do
  @note = 'Hi!'
  request.path_info = '/foo/bar/baz'
end

get '/foo/*' do
  @note #=> 'Hi!'
  params[:splat] #=> 'bar/baz'
end
```

After Filters *After filters* are evaluated after each request within the same context and can also modify the request and response.

Instance variables set in before filters and routes are accessible by after filters:

```
after do
  puts response.status
end
```

Note: Unless you use the `body` method rather than just returning a String from the routes, the body will not yet be available in the after filter, since it is generated later on.

Filters can take a Pattern Filters optionally take a pattern, causing them to be evaluated only if the request path matches that pattern:

```
before '/protected/*' do
  authenticate!
end

after '/create/:slug' do |slug|
  session[:last_slug] = slug
end
```

Filters can take a Condition Like routes, filters also take conditions:

```
before :agent => /Songbird/ do
  # ...
end

after '/blog/*', :host_name => 'example.com' do
  # ...
end
```

18.6. Manejo de Errores

1. The HTTP specification states that response status in the range 200-299 indicate success in processing a request
2. 500-599 is reserved for server errors
3. Sinatra offers helpers for the 404 (Not Found) and 500 (Internal Server Error) status

not_found When a `Sinatra::NotFound` exception is raised, or the response's status code is 404, the `not_found` handler is invoked:

```
not_found do
  'This is nowhere to be found.'
end
```

error The `error` handler is invoked any time an exception is raised from a route block or a filter. The exception object can be obtained from the `sinatra.error` Rack variable:

```
error do
  'Sorry there was a nasty error - ' + env['sinatra.error'].name
end
```

Custom errors:

```
error MyCustomError do
  'So what happened was...' + env['sinatra.error'].message
end
```

Then, if this happens:

```
get '/' do
  raise MyCustomError, 'something bad'
end
```

You get this:

```
So what happened was... something bad
```

Alternatively, you can install an error handler for a status code:

```
error 403 do
  'Access forbidden'
end

get '/secret' do
  403
end
```

Or a range:

```
error 400..510 do
  'Boom'
end
```

Sinatra installs special `not_found` and `error` handlers when running under the development environment to display nice stack traces and additional debugging information in your browser (esto es, en producción estos handlers son mucho mas ”parcos”).

18.7. The methods body, status and headers

1. It is possible and recommended to set the status code and response body with the return value of the route block.
2. However, in some scenarios you might want to set the body at an arbitrary point in the execution flow.
3. You can do so with the `body` helper method.
4. If you do so, you can use that method from there on to access the body:

   ```
   get '/foo' do
     body "bar"
   end

   after do
     puts body
   end
   ```

   It is also possible to pass a block to `body`, which will be executed by the Rack handler (this can be used to implement *streaming*).
5. Similar to the `body`, you can also set the `status` code and `headers`:

   ```
   get '/foo' do
     status 418
     headers \
       "Allow"   => "BREW, POST, GET, PROPFIND, WHEN",
       "Refresh" => "Refresh: 20; http://www.ietf.org/rfc/rfc2324.txt"
     body "I'm a tea pot!"
   end
   ```
6. Like `body`, `headers` and `status` with no arguments can be used to access their current values.

18.8. Acceso al Objeto Request

El objeto que representa la solicitud *the request object* es un hash con información de la solicitud: quien hizo la petición, que versión de HTTP usar, etc.

El objeto que representa la solicitud puede ser accedido desde el nivel de solicitud: filtros, rutas y manejadores de error.

Véase https://github.com/crguezl/sinatra_intro/blob/master/accesing_the_request_object.rb

18.9. Caching / Caches

Mediante el uso del helper `headers`podemos establecer los headers que queramos para influir sobre la forma en la que ocurre el caching downstream.

1. Caching Tutorial

18.10. Sesiones y Cookies en Sinatra

Introducción A session is used to keep state during requests. If activated, you have one session hash per user session:

```
enable :sessions

get '/' do
  "value = " << session[:value].inspect
end

get '/:value' do
  session[:value] = params[:value]
end
```

1. Note that `enable :sessions` actually stores all data in a cookie
2. This might not always be what you want (storing lots of data will increase your traffic, for instance)
3. You can use any Rack session middleware: in order to do so, do not call `enable :sessions`, but instead pull in your middleware of choice as you would any other middleware:

   ```
   use Rack::Session::Pool, :expire_after => 2592000

   get '/' do
     "value = " << session[:value].inspect
   end

   get '/:value' do
     session[:value] = params[:value]
   end
   ```

4. To improve security, the session data in the cookie is signed with a session secret
5. A random secret is generated for you by Sinatra
6. However, since this secret will change with every start of your application, you might want to set the secret yourself, so all your application instances share it:

```
set :session_secret, 'super secret'
```

If you want to configure it further, you may also store a hash with options in the sessions setting:

```
set :sessions, :domain => 'foo.com'
```

7. Just use `session.clear` to destroy the session.

```
get '/login' do
    session[:username] = params[:username]
    "logged in as #{session[:username]}"
  end

  get '/logout' do
    old_user = session[:username]
    session.clear
    "logged out #{old_user}"
  end
```

Cookies

1. According to the Computer Science definition, a cookie, which is also known as an HTTP cookie, a tracking cookie, or a browser cookie, is a piece of text, no bigger than 4 kilobytes, which is stored on the user's computer by a web server via a web browser

2. It is a key-value pair structure, which is designed to retain specific information such as user preferences, user authentication, shopping carts, demographics, sessions, or any other data used by a website

3. This mechanism, which was developed by Netscape in the distant 1994, provides a way to receive information from a web server and to send it back from the web browser absolutely unchanged

4. This system complements the stateless nature of the HTTP protocol as it provides enough memory to store pieces of information during HTTP transactions

5. When you try to access a web site, your web browser connects to a web server and it sends a request for the respective page

6. Then the web server replies by sending the requested content and it simultaneously stores a new cookie on your computer

7. Every time the web browser requests web pages from the web server, it always sends the respective cookies back to the web server

8. The process takes place as described, if the web browser supports cookies and the user allows their usage

9. Only the web server can modify one or more of the cookie values

10. Then it sends them to the web browser upon replying to a specific request

11. According to the RFC2965 specification, cookies are case insensitive

12. A set of defined properties is inherent to the cookie structure Those properties include: an expiration date, a path and a domain

13. The first attribute requires a date defined in `Wdy, DD-Mon-YYYY HH:MM:SS GMT` format

14. The rest of the cookie characteristics require a `path` and/or a `domain` defined as a string

15. Let's take a look at this example:

```
Cookie: key0=value0; ...; keyX=valueX; expires=Wed, 23-Sep-2009 23:59:59 GMT; path=/; doma
```

16. When the `expiration` date is defined, your cookie will be *persistent* as it will reoccur in different sessions until the set `expiration` date has been reached

17. If the `expiration` date has not been defined in the cookie, it will occur until the end of your current session or when you close your web browser

18. If the `path` and/or the `domain` attributes have been defined in your cookie, then the web server limits the scope of the cookie to that specific `domain`, `sub-domain` or `path`

Ejemplo con Sesiones

```
require 'rubygems'
require 'sinatra'
require 'haml'

enable :sessions

get '/' do
  session["user"] ||= nil
  haml :index
end

get '/introduction' do
  haml :introduction
end

post '/introduction' do
  session["user"] = params[:name]
  redirect '/'
end

get '/bye' do
  session["user"] = nil
  haml :bye
end
```

Ejemplo con Cookies

1. The last example will demonstrate how to directly manage cookies through the `request` and `response` singletons provided by Sinatra

2. You will see in the following example that the previously described process involving the use cookies is clearly implemented

3. This technique is recommended when your application requires to use persistent and/or scoped cookies

4. In this example, the application uses two persistent cookies, which expire at the same time, in order to store and manage different configuration data

```
require 'sinatra'
require 'haml'

get '/' do
  @@expiration_date = Time.now + (60 * 2) \
  unless request.cookies.key?('some_options') && request.cookies.key?('other_options')
  haml :index
end

get '/some_options' do
  @some_cookie = request.cookies["some_options"]
  haml :some_options
end

post '/some_options' do
  response.set_cookie('some_options', :value => cookie_values(params), :expires => @@expiratio
  redirect '/'
end

get '/other_options' do
  @other_cookie = request.cookies["other_options"]
  haml :other_options
end

post '/other_options' do
  response.set_cookie('other_options', :value => cookie_values(params),:expires => @@expiratio
  redirect '/'
end

helpers do
  def cookie_values(parameters)
    values = {}
    parameters.each do |key, value|
      case key
      when 'options'
        values[value] = true
      else
        values[key] = true
      end
    end
    values
  end
end
```

Problemas

1. I'm not sure why but my session gets wiped out every request?
2. To keep sessions consistent you need to set a session secret, e.g.:

   ```
   set :session_secret, 'super secret'
   ```

   When it's not set sinatra generates random one on application start and shotgun restarts application before every request.

Véanse

1. Daily Ruby Tips #60 – Simple Use of Sessions in Sinatra May 6, 2013
2. La sección *Cookies* en Rack 16.8.
3. Cookie-based Sessions in Sinatra by JULIO JAVIER CICCHELLI on SEPTEMBER 30, 2009 RubyLearning Blog. El código está en un Gist en GitHub

18.11. Downloads / Descargas / Attachments

Usando attachment

There is a built-in `attachment` method that optionally takes a filename parameter. If the filename has an extension (`.jpg`, etc.) that extension will be used to determine the `Content-Type` header for the response.

The evaluation of the route will provide the contents of the attachment.

1. Documentación de attachment
2. Código de attachment en GitHub
3. Upload and download files in Sinatra Random Ruby Thoughts

```
[~/sinatra/sinatra-download(master)]$ cat app.rb
require 'sinatra'

before do
  content_type :txt
end

get '/attachment?' do
  attachment 'file.txt'
  "Here's what will be sent downstream, in an attachment called 'file.txt'."
end
```

Cuando visitamos la página con el navegador se nos abre una ventana para la descarga de un fichero que será guardado (por defecto) como `file.txt`.

Los contenidos de ese fichero serán:

```
[~/sinatra/sinatra-download(master)]$ cat ~/Downloads/file.txt
Here's what will be sent downstream, in an attachment called 'file.txt'.
```

Usando send_file

Véase la documentación del módulo `Sinatra::Streaming`

```
[~/sinatra/sinatra-download(master)]$ cat sending_file.rb
require 'sinatra'

get '/' do
   send_file 'foo.png',
              :type => 'img/png',
              :disposition => 'attachment',
              :filename =>'tutu.png',
              :stream => false
end
```

The options are:

1. `filename` file name, in response, defaults to the real file name.
2. `last_modified` value for Last-Modified header, defaults to the file's `mtime`.
3. `type` content type to use, guessed from the file extension if missing.
4. `disposition` used for `Content-Disposition`, possible value: `nil` (default), `:attachment` and `:inline`
5. `length` `Content-Length` header, defaults to file size.
6. `status` Status code to be send.

   Useful when sending a static file as an error page. If supported by the Rack handler, other means than streaming from the Ruby process will be used. If you use this helper method, Sinatra will automatically handle range requests.

18.12. Uploads. Subida de Ficheros en Sinatra

Véase

1. El repositorio sinatra-upload en GitHub con el código de este ejemplo
2. FILE UPLOAD WITH SINATRA BY PANDAFOX POSTED IN RUBY, TUTORIALS
3. Multiple file uploads in Sinatra

Jerarquía de ficheros

```
[~/sinatra/sinatra-upload]$ tree
.
|-- app.rb
|-- uploads
|   `--- README
`--- views
    `--- upload.haml

2 directories, 3 files
```

upload.haml

The important part is not to forget to set the `enctype` in your form element, otherwise you will just get the filename instead of an object:

```
[~/sinatra/sinatra-upload(master)]$ cat views/upload.haml
%html
  %body
    %h1 File uploader!
    %form(method="post" enctype='multipart/form-data')
      %input(type='file' name='myfile')
      %br
      %input(type='submit' value='Upload!')
```

app.rb

```
[~/sinatra/sinatra-upload(master)]$ cat app.rb
require 'rubygems'
require 'sinatra'
require 'haml'
require 'pp'

# Handle GET-request (Show the upload form)
get "/upload?" do
  haml :upload
end

# Handle POST-request (Receive and save the uploaded file)
post "/upload" do
  pp params
  File.open('uploads/' + params['myfile'][:filename], "w") do |f|
    f.write(params['myfile'][:tempfile].read)
  end
  return "The file was successfully uploaded!"
end
[~/sinatra/sinatra-upload(master)]$
```

As you can see, you don't have to write much code to get this to work. The `params`-hash contains our uploaded element with data such as filename, type and the actual datafile, which can be accessed as a Tempfile-object.

We read the contents from this file and store it into a directory called uploads, which you will have to create before running this script.

Here's an example of what the `params`-hash may look like when uploading a picture of a cat:

```
{
  "myfile" => {
    :type => "image/png",
    :head =>  "Content-Disposition: form-data;
               name=\"myfile\";
               filename=\"cat.png\"\r\n
               Content-Type: image/png\r\n",
    :name => "myfile",
    :tempfile => #<File:/var/folders/3n/3asd/-Tmp-/RackMultipart201-1476-nfw2-0>,
    :filename=>"cat.png"
  }
}
```

File upload with sinatra. YouTube

BEWARE!

This script offers little to no security at all. Clients will be able to overwrite old images, fill up your harddrive and so on. So just use some common sense and do some Ruby magic to patch up the security holes yourself.

18.13. halt

Sometimes we want to stop the program: maybe a critical error has ocurred. To immediately stop a request within a filter or route use:

```
halt
```

You can also specify the status when halting:

```
halt 410
```

Or the body:

```
halt 'this will be the body'
```

Or both:

```
halt 401, 'go away!'
```

With headers:

```
halt 402, {'Content-Type' => 'text/plain'}, 'revenge'
```

It is of course possible to combine a template with halt:

```
halt erb(:error)
```

18.14. Passing a Request

When we want to pass processing to the next matching route we use `pass`:

```
get '/guess/:who' do
  pass unless params[:who] == 'Frank'
  'You got me!'
end

get '/guess/*' do
  'You missed!'
end
```

The route block is immediately exited and control continues with the next matching route.
If no matching route is found, a 404 is returned.

18.15. Triggering Another Route: calling `call`

Sometimes `pass` is not what you want, instead you would like to get the result of calling another route. Simply use `call` to achieve this:

```
get '/foo' do
  status, headers, body = call env.merge("PATH_INFO" => '/bar')
  [status, headers, body.map(&:upcase)]
end

get '/bar' do
  "bar"
end
```

Note that in the example above, you would ease testing and increase performance by simply moving `"bar"` into a helper used by both `/foo` and `/bar`.

If you want the request to be sent to the same application instance rather than a duplicate, use `call!` instead of `call`.

Check out the Rack specification if you want to learn more about `call`.

18.16. Logging

In the request scope, the `logger` helper exposes a `Logger` instance:

```
get '/' do
  logger.info "loading data"
  # ...
end
```

1. This `logger` will automatically take your Rack handler's `logging` settings into account

2. If `logging` is disabled, this method will return a dummy object, so you do not have to worry in your routes and filters about it

Note that `logging` is only enabled for `Sinatra::Application` by default, so if you inherit from , you probably want to enable it yourself:

```
class MyApp < Sinatra::Base
  configure :production, :development do
    enable :logging
  end
end
```

1. To avoid any `logging` middleware to be set up, set the `logging` setting to `nil`

2. However, keep in mind that `logger` will in that case return `nil`

3. A common use case is when you want to set your own `logger`. Sinatra will use whatever it will find in `env['rack.logger']`

Logging a stdout y a un fichero

Véase Rack::CommonLogger en Sinatra Recipes.

Sinatra has logging support, but it's nearly impossible to log to a file and to the stdout (like Rails does).

However, there is a little trick you can use to log to stdout and to a file:

```
require 'sinatra'

configure do
  # logging is enabled by default in classic style applications,
  # so `enable :logging` is not needed
  file = File.new("#{settings.root}/log/#{settings.environment}.log", 'a+')
  file.sync = true
  use Rack::CommonLogger, file
end

get '/' do
  'Hello World'
end
```

You can use the same configuration for modular style applications, but you have to `enable :logging first`:

```
require 'sinatra/base'

class SomeApp < Sinatra::Base
  configure do
```

```
    enable :logging
    file = File.new("#{settings.root}/log/#{settings.environment}.log", 'a+')
    file.sync = true
    use Rack::CommonLogger, file
  end

  get '/' do
    'Hello World'
  end

  run!
end
```

Ejecución

```
~/sinatra/sinatra-logging]$ tree
.
|-- app.rb
`-- log

1 directory, 1 file
[~/sinatra/sinatra-logging]$ ruby app.rb
== Sinatra/1.4.4 has taken the stage on 4567 for development with backup from Thin
Thin web server (v1.6.1 codename Death Proof)
Maximum connections set to 1024
Listening on localhost:4567, CTRL+C to stop
```

Consola después de visitar la página:

```
127.0.0.1 - - [19/Nov/2013 14:53:06] "GET / HTTP/1.1" 200 11 0.0041
```

Fichero después de visitar la página:

```
[~/sinatra/sinatra-logging]$ cat log/development.log
127.0.0.1 - - [19/Nov/2013 14:53:06] "GET / HTTP/1.1" 200 11 0.0038
```

Véase

1. el código en GitHub de Rack::CommonLogger
2. Logging in Sinatra. StackOverflow. Destination is set by changing `env['rack.errors']`. Konstantin Haase May 13 '11 at 21:18'

18.17. Generating URLs

For generating URLs you should use the `url` helper method, for instance, in Haml:

```
%a{:href => url('/foo')} foo
```

It takes reverse proxies and Rack routers into account, if present.
This method is also aliased to `to`.

18.18. Redireccionamientos/Browser Redirect

You can trigger a browser redirect with the `redirect` helper method:

```
get '/foo' do
  redirect to('/bar')
end
```

Any additional parameters are handled like arguments passed to `halt`:

```
redirect to('/bar'), 303
redirect 'http://google.com', 'wrong place, buddy'
```

You can also easily `redirect` back to the page the user came from with `redirect back`:

```
get '/foo' do
  "<a href='/bar'>do something</a>"
end

get '/bar' do
  do_something
  redirect back
end
```

To pass arguments with a `redirect`, either add them to the query:

```
redirect to('/bar?sum=42')
```

Or use a session:

```
enable :sessions

get '/foo' do
  session[:secret] = 'foo'
  redirect to('/bar')
end

get '/bar' do
  session[:secret]
end
```

18.19. Configuration / Configuración

Run once, at startup, in any environment:

```
configure do
  # setting one option
  set :option, 'value'

  # setting multiple options
  set :a => 1, :b => 2

  # same as `set :option, true`
  enable :option
```

```
  # same as `set :option, false`
  disable :option

  # you can also have dynamic settings with blocks
  set(:css_dir) { File.join(views, 'css') }
end
```

Run only when the environment (`RACK_ENV` environment variable) is set to `:production`:

```
configure :production do
  ...
end
```

Run when the environment is set to either `:production` or `:test`:

```
configure :production, :test do
  ...
end
```

You can access those options via settings:

```
configure do
  set :foo, 'bar'
end

get '/' do
  settings.foo? # => true
  settings.foo  # => 'bar'
  ...
end
```

18.20. Configuring attack protection

Sinatra is using Rack::Protection to defend your application against common, opportunistic attacks.

1. You can easily disable this behavior (which will open up your application to tons of common vulnerabilities):

   ```
   disable :protection
   ```

2. To skip a single defense layer, set protection to an options hash:

   ```
   set :protection, :except => :path_traversal
   ```

3. You can also hand in an array in order to disable a list of protections:

   ```
   set :protection, :except => [:path_traversal, :session_hijacking]
   ```

4. By default, Sinatra will only set up session based protection if `:sessions` has been enabled.

   Sometimes you want to set up sessions on your own, though.

   In that case you can get it to set up session based protections by passing the `:session option`:

   ```
   use Rack::Session::Pool
   set :protection, :session => true
   ```

18.21. Settings disponibles/Available Settings

1. `absolute_redirects` If disabled, Sinatra will allow relative redirects, however, Sinatra will no longer conform with RFC 2616 (HTTP 1.1), which only allows absolute redirects.

   Enable if your app is running behind a reverse proxy that has not been set up properly. Note that the `url` helper will still produce absolute URLs, unless you pass in `false` as the second parameter. Disabled by default.

2. `add_charsets` mime types the `content_type` helper will automatically add the charset info to.

   You should add to it rather than overriding this option:

   ```
   settings.add_charsets << "application/foobar"
   ```

3. `app_file` Path to the main application file, used to detect project root, views and public folder and inline templates.

4. `bind` IP address to bind to (default: 0.0.0.0 or localhost if your `environment` is set to `development`). Only used for built-in server.

5. `default_encoding` encoding to assume if unknown (defaults to ütf-8").

6. `dump_errors` display errors in the log.

7. `environment` current environment, defaults to `ENV['RACK_ENV']`, or `development` if not available.

8. `logging` use the logger.

9. `lock` Places a lock around every request, only running processing on request per Ruby process concurrently. Enabled if your app is not thread-safe. Disabled per default.

10. `method_override` use `_method magic` to allow put/delete forms in browsers that don't support it.

11. `port` Port to listen on. Only used for built-in server.

12. `prefixed_redirects` Whether or not to insert `request.script_name` into redirects if no absolute path is given. That way redirect '/foo' would behave like redirect to('/foo'). Disabled per default.

13. `protection` Whether or not to enable web attack protections. See protection section above.

14. `public_dir` Alias for `public_folder`. See below.

15. `public_folder` Path to the folder public files are served from. Only used if static file serving is enabled (see static setting below). Inferred from `app_file` setting if not set.

16. `reload_templates` Whether or not to reload templates between requests. Enabled in development mode.

17. `root` Path to project root folder. Inferred from `app_file` setting if not set.

18. `raise_errors` raise exceptions (will stop application). Enabled by default when environment is set to "test", disabled otherwise.

19. `run` if enabled, Sinatra will handle starting the web server, do not enable if using rackup or other means.

20. `running` is the built-in server running now? do not change this setting!

21. `server` Server or list of servers to use for built-in server. order indicates priority, default depends on Ruby implementation.

22. `sessions` Enable cookie-based sessions support using Rack::Session::Cookie. See 'Using Sessions' section for more information.

23. `show_exceptions` Show a stack trace in the browser when an exception happens. Enabled by default when environment is set to "development", disabled otherwise. Can also be set to `:after_handler` to trigger app-specified error handling before showing a stack trace in the browser.

24. `static` Whether Sinatra should handle serving static files. Disable when using a server able to do this on its own. Disabling will boost performance. Enabled per default in classic style, disabled for modular apps.

25. `static_cache_control` When Sinatra is serving static files, set this to add Cache-Control headers to the responses. Uses the `cache_control` helper. Disabled by default. Use an explicit array when setting multiple values:

    ```
    set :static_cache_control, [:public, :max_age => 300]
    ```

26. `threaded` If set to true, will tell Thin to use EventMachine.defer for processing the request.

27. `views` Path to the views folder. Inferred from `app_file` setting if not set.

28. `x_cascade` Whether or not to set the X-Cascade header if no route matches. Defaults to true.

18.22. Environments

1. There are three predefined environments: `development` , `production` and `test` .

2. Environments can be set through the `RACK_ENV` environment variable.

3. The default value is `development`

4. In the `development` environment all templates are reloaded between requests, and special `not_found` and `error` handlers display stack traces in your browser

5. In the `production` and `test` environments, templates are cached by default

6. To run different environments, set the `RACK_ENV` environment variable:

    ```
    RACK_ENV=production ruby my_app.rb
    ```

7. You can use predefined methods: `development?`, `test?` and `production?` to check the current environment setting:

    ```
    get '/' do
      if settings.development?
        "development!"
      else
        "not development!"
      end
    end
    ```

18.23. Correo

```
[~/srcSTW/sinatra-faq/mail(esau)]$ cat app.rb
require 'sinatra'
require 'pony'
raise "Execute:\n\t#{$0} password email_to email_from" if ARGV.length.zero?
get '/' do
    email = ARGV.shift
    pass = ARGV.shift
    Pony.mail({
      :to => email,
      :body => "Hello Casiano",
      :subject => 'Howdy, Partna!',
      :via => :smtp,
      :via_options => {
          :address              => 'smtp.gmail.com',
          :port                 => '587',
          :enable_starttls_auto => true,
          :user_name            => ARGV.shift,
          :password             => pass,
          :authentication       => :plain, # :plain, :login, :cram_md5, no auth by default
          :domain               => "localhost.localdomain" # the HELO domain provided by the c
        }
    })
    "Check your email at #{email}"
end
```

1. Getting started with Sinatra

18.24. Ambito

The scope you are currently in determines what methods and variables are available.

Ámbito de Clase/Class Scope

1. Every Sinatra application corresponds to a subclass of Sinatra::Base
2. If you are using the top-level DSL (`require 'sinatra'`), then this class is Sinatra::Application, otherwise it is the subclass you created explicitly
3. At class level you have methods like `get` or `before`, but you cannot access the `request` or `session` objects, as there is only a single application class for all requests

Options created via `set` are methods at class level:

```
class MyApp < Sinatra::Base
  # Hey, I'm in the application scope!
  set :foo, 42
  foo # => 42

  get '/foo' do
    # Hey, I'm no longer in the application scope!
  end
end
```

You have the application scope binding inside:

1. Your application class body
2. Methods defined by extensions
3. The block passed to `helpers`
4. Procs/blocks used as value for `set`
5. The block passed to `Sinatra.new`

You can reach the scope object (the class) like this:

1. Via the object passed to configure blocks (`configure { |c| ... }`)
2. `settings` from within the request scope

Ámbito de Instancia/Instance Scope

For every incoming request, a new instance of your application class is created and all handler blocks run in that scope

1. From within this scope you can access the `request` and `session` objects or
2. call rendering methods like `erb` or `haml`
3. You can access the application scope from within the `request` scope via the `settings` helper:

```
[~/sinatra/sinatra-scope]$ cat app.rb
require 'sinatra'

class MyApp < Sinatra::Base
  # Hey, I'm in the application scope!
  get '/define_route/:name' do
    # Request scope for '/define_route/:name'
    @value = 42
    puts "Inside /define_route/:name @value = #{@value}"
    puts self.class

    settings.get("/#{params[:name]}") do
      # Request scope for "/#{params[:name]}"
      puts "@value = <#{@value}>"
      "Inside defined route #{params[:name]}"
    end

    "Route #{params[:name]} defined!"
  end

  run! if __FILE__ == $0
end
```

Ejecución en el servidor

```
[~/sinatra/sinatra-scope]$ ruby app.rb
== Sinatra/1.4.4 has taken the stage on 4567 for development with backup from Thin
Thin web server (v1.6.1 codename Death Proof)
Maximum connections set to 1024
```

```
Listening on localhost:4567, CTRL+C to stop
Inside /define_route/:name @value = 42
MyApp
@value = <>
```

Ejecución en el cliente. Ruta: /define_route/juan

```
[~/sinatra/sinatra-scope]$ curl -v 'http://localhost:4567/define_route/juan'
* Adding handle: conn: 0x7fbacb004000
* Adding handle: send: 0
* Adding handle: recv: 0
* Curl_addHandleToPipeline: length: 1
* - Conn 0 (0x7fbacb004000) send_pipe: 1, recv_pipe: 0
* About to connect() to localhost port 4567 (#0)
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 4567 (#0)
> GET /define_route/juan HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost:4567
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html;charset=utf-8
< Content-Length: 19
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Connection: keep-alive
* Server thin 1.6.1 codename Death Proof is not blacklisted
< Server: thin 1.6.1 codename Death Proof
<
* Connection #0 to host localhost left intact
Route juan defined!
[~/sinatra/sinatra-scope]$
```

Ejecución en el cliente. Ruta: /juan

```
[~/sinatra/sinatra-scope]$ curl -v 'http://localhost:4567/juan'
* Adding handle: conn: 0x7fbdd1800000
* Adding handle: send: 0
* Adding handle: recv: 0
* Curl_addHandleToPipeline: length: 1
* - Conn 0 (0x7fbdd1800000) send_pipe: 1, recv_pipe: 0
* About to connect() to localhost port 4567 (#0)
*   Trying ::1...
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 4567 (#0)
> GET /juan HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost:4567
> Accept: */*
>
< HTTP/1.1 200 OK
```

```
< Content-Type: text/html;charset=utf-8
< Content-Length: 21
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Connection: keep-alive
* Server thin 1.6.1 codename Death Proof is not blacklisted
< Server: thin 1.6.1 codename Death Proof
<
* Connection #0 to host localhost left intact
Inside defined route
```

You have the request scope binding inside:

1. `get`, `head`, `post`, `put`, `delete`, `options`, `patch`, `link`, and `unlink` blocks
2. `before` and `after` filters
3. `helper` methods
4. templates/views

18.25. Sinatra Authentication

18.25.1. Referencias

1. Ejemplo de uso de sinatra-authentication

18.26. Autentificación Básica

```
[~/srcSTW/sinatra-faq/authentication/basic(esau)]$ cat app.rb
require 'rubygems'
require 'sinatra'

use Rack::Auth::Basic, "Restricted Area" do |username, password|
  [username, password] == ['admin', 'admin']
end

get '/' do
  "You're welcome"
end

get '/foo' do
  "You're also welcome"
end
```

18.27. Sinatra como Middleware

Not only is Sinatra able to use other Rack middleware, any Sinatra application can in turn be added in front of any Rack endpoint as middleware itself.

This endpoint could be another Sinatra application, or any other Rack-based application (Rails/-Ramaze/Camping/. . .):

1. When a request comes in, all `before` filters are triggered

2. Then, if a route matches, the corresponding block will be executed
3. If no route matches, the request is handed off to the wrapped application
4. The `after` filters are executed after we've got a response back from the route or the wrapped app

Thus, our Sinatra app is a middleware.

```
[~/sinatra/sinatra-as-middleware]$ cat app.rb
require 'sinatra/base'
require 'haml'
require 'pp'

class LoginScreen < Sinatra::Base
  enable :sessions
  enable :inline_templates

  get('/login') { haml :login }

  post('/login') do
    if params[:name] == 'admin' && params[:password] == 'admin'
      puts "params = "
      pp params
      session['user_name'] = params[:name]
      redirect '/'
    else
      redirect '/login'
    end
  end
end

class MyApp < Sinatra::Base
  enable :inline_templates
  # middleware will run before filters
  use LoginScreen

  before do
    unless session['user_name']
      halt haml :denied
    end
  end

  get('/') do
    haml :cheer, :locals => { :name => session['user_name'] }
  end

  run!
end

__END__

@@ layout
!!!
%html
```

```
  %head
    %title Sinatra as Middleware
  %body
    %h1 Sinatra as Middleware
    = yield

@@ login
%form{:action=>'/login', :method=>'post'}
  %label{:for=>'name'} Name
  %input#name{:type=>"text",     :name=>"name", :autofocus => true }
  %br
  %label{:for=>'password'} Password
  %input#password{:type=>"password", :name=>"password"}
  %br
  %button#go{:type=>"submit",  :name=>"submit", :value=>"submit"} Click me!

@@ cheer
%h1
  Hello #{name}
  %br

@@ denied
%h1
  Access denied, please
  %a{:href=>'/login'}login.
```

18.28. Práctica: TicTacToe

El código que sigue implanta un jugador de tres-en-raya.

1. Mejore el estilo actual usando SAAS: utilice variables, extensiones, mixins ...
2. Despliegue su versión en Heroku

Referencias

1. http://sytw-tresenraya.herokuapp.com/
2. https://github.com/crguezl/tictactoe-1
3. Sass (Syntactically Awesome StyleSheets): Sass Basics
4. Un TicTacToe Simple (No una webapp)

Estructura

```
[~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$ tree
.
|--- Gemfile
|--- Gemfile.lock
|--- Procfile
|--- Rakefile
|--- Readme.md
|--- app.rb
|--- public
```

```
|   |--- css
|   |   |--- app.css
|   |   `--- style.css
|   |--- images
|   |   |--- blackboard.jpg
|   |   |--- circle.gif
|   |   `--- cross.gif
|   `--- js
|       `--- app.js
`--- views
    |--- final.erb
    |--- final.haml
    |--- game.erb
    |--- game.haml
    |--- layout.erb
    |--- layout.haml
    `--- styles.scss

5 directories, 19 files
```

Rakefile

```
[~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$ cat Rakefile
desc "run server"
task :default do
  sh "bundle exec ruby app.rb"
end

desc "install dependencies"
task :install do
  sh "bundle install"
end

###
desc 'build css'
task :css do
  sh "sass views/styles.scss public/css/style.css"
end
```

HAML

1. game.haml en GitHub
2. layout.haml

```
[~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$  cat views/game.haml
.screen
  .gameboard
    - HORIZONTALS.each do |row|
      .gamerow
        - row.each do |p|
          %a(href=p)
            %div{:id => "#{p}", :class => "cell #{b[p]}"}
    .message
      %h1= m
```

```
[~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$ cat views/layout.haml
!!!
%html
  %head
    %title tic tac toe
    -#%link{:rel=>"stylesheet", :href=>"/css/app.css", :type=>"text/css"}
    -# dynamically accessed
    -#%link{:rel=>"stylesheet", :href=>"/styles.css", :type=>"text/css"}
    -# statically compiled
    %link{:rel=>"stylesheet", :href=>"css/style.css", :type=>"text/css"}
    %script{:type=>"text/javascript", :src=>"http://ajax.googleapis.com/ajax/libs/jquery/1.6.4
    %script{:type=>"text/javascript", :src=>"/js/app.js"}
  %body
    = yield
```

1. El fuente `styles.scss` puede compilarse *dinámicamente*. Véase el fragmento de código que empieza por `get '/styles.css' do` en `app.rb`
2. O puede compilarse estáticamente. Véase el Rakefile

HTML generado

```
<!DOCTYPE html>
<html>
  <head>
    <title>tic tac toe</title>
    <link href='css/style.css' rel='stylesheet' type='text/css'>
    <script src='http://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js' type='text/j
    <script src='/js/app.js' type='text/javascript'></script>
  </head>
  <body>
    <div class='screen'>
      <div class='gameboard'>
        <div class='gamerow'>
          <a href='a1'>
            <div class='cell ' id='a1'></div>
          </a>
          <a href='a2'>
            <div class='cell ' id='a2'></div>
          </a>
          <a href='a3'>
            <div class='cell ' id='a3'></div>
          </a>
        </div>
        <div class='gamerow'>
          <a href='b1'>
            <div class='cell ' id='b1'></div>
          </a>
          <a href='b2'>
            <div class='cell circle' id='b2'></div>
          </a>
          <a href='b3'>
            <div class='cell ' id='b3'></div>
          </a>
```

```
        </div>
        <div class='gamerow'>
          <a href='c1'>
            <div class='cell ' id='c1'></div>
          </a>
          <a href='c2'>
            <div class='cell ' id='c2'></div>
          </a>
          <a href='c3'>
            <div class='cell cross' id='c3'></div>
          </a>
        </div>
        <div class='message'>
          <h1></h1>
        </div>
      </div>
    </div>
  </body>
</html>
```

SASS

1. styles.scss
2. Sass (Syntactically Awesome StyleSheets): Sass Basics
3. SASS documentación
4. sass man page
5. *SASS (Syntactically Awesome StyleSheets)* **??**

```
~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$ cat views/styles.scss
$red:   #903;
$black: #444;
$white: #fff;
$ull:   #9900FF;
$pink:  #F9A7B0;

$main-font: Helvetica, Arial, sans-serif;
$message-font: 22px/1;

$board-left: 300px;
$board-margin: 0 auto;
$board-size: 500px;

$opacity:  0.8;

$cell-width:    $board-size/8.5;
$cell-height:   $board-size/8.5;
$cell-margin:  $cell-width/12;
$cell-padding:  $cell-width/1.3;

$background: "/images/blackboard.jpg";
$cross:      "/images/cross.gif";
```

```
$circle:     "/images/circle.gif";

body       {
             // background-color: lightgrey;
             font-family: $main-font;
             background: url($background) repeat; background-size: cover;
           }
.gameboard { //margin-left: $board-left;
             width: $board-size;
             margin: $board-margin;
             text-align:center;
           }
.gamerow   { clear: both; }
.cell      { color: blue;
             background-color: white;
             opacity: $opacity;
             width: $cell-width;
             height: $cell-height;
             margin: $cell-margin;
             padding: $cell-padding;
             &:hover {
               color: black ;
               background-color: $ull;
             }
             float: left;
           }

@mixin game-piece($image) {
  background: url($image)  no-repeat; background-size: cover;
}

.cross     { @include game-piece($cross); }
.circle    { @include game-piece($circle); }

.base-font { color: $pink; font: $message-font $main-font; }

.message   {
             @extend .base-font;
             display: inline;
             background-color:transparent;
           }
```

Procfile

Procfile en GitHub

In order to declare the processes that make our app, and scale them individually, we need to be able to tell Heroku what these processes are.

The Procfile is a simple YAML file which sits in the root of your application code and is pushed to your application when you deploy. This file contains a definition of every process you require in your application, and how that process should be started.

```
[~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$ cat Procfile
#web: bundle exec unicorn -p $PORT -E $RACK_ENV
#web: bundle exec ruby app.rb -p $PORT
web: bundle exec ruby app.rb
```

```
#web: bundle exec thin start
```

Véase The Procfile is your friend

1. Heroku, Thin and everything in between en StackOverflow
2. Process Types and the Procfile en Heroku

Gemfile

```
[~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$ cat Gemfile
source "https://rubygems.org"

gem "sinatra"
gem 'haml'
gem "sass", :require => 'sass'
gem 'thin'
```

La Aplicación

```
[~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$  cat app.rb
require 'sinatra'
require 'sass'
require 'pp'

settings.port = ENV['PORT'] || 4567
enable :sessions
#use Rack::Session::Pool, :expire_after => 2592000
#set :session_secret, 'super secret'

#configure :development, :test do
#  set :sessions, :domain => 'example.com'
#end

#configure :production do
#  set :sessions, :domain => 'herokuapp.com'
#end

module TicTacToe
  HUMAN = CIRCLE = "circle" # human
  COMPUTER = CROSS  = "cross"  # computer
  BLANK  = ""

  HORIZONTALS = [ %w{a1 a2 a3},  %w{b1 b2 b3}, %w{c1 c2 c3} ]
  COLUMNS     = [ %w{a1 b1 c1},  %w{a2 b2 c2}, %w{a3 b3 c3} ]
  DIAGONALS   = [ %w{a1 b2 c3},  %w{a3 b2 c1} ]
  ROWS = HORIZONTALS + COLUMNS + DIAGONALS
  MOVES       = %w{a1    a2   a3   b1   b2   b3   c1   c2   c3}

  def number_of(symbol, row)
    row.find_all{ |s| session["bs"][s] == symbol }.size
  end

  def inicializa
```

```ruby
  @board = {}
  MOVES.each do |k|
    @board[k] = BLANK
  end
  @board
end

def board
  session["bs"]
end

def [] key
  board[key]
end

def []= key, value
  board[key] = value
end

def each
  MOVES.each do |move|
    yield move
  end
end

def legal_moves
  m = []
  MOVES.each do |key|
    m << key if board[key] == BLANK
  end
  puts "legal_moves: Tablero:  #{board.inspect}"
  puts "legal_moves: m:  #{m}"
  m # returns the set of feasible moves [ "b3", "c2", ... ]
end

def winner
  ROWS.each do |row|
    circles = number_of(CIRCLE, row)
    puts "winner: circles=#{circles}"
    return CIRCLE if circles == 3  # "circle" wins
    crosses = number_of(CROSS, row)
    puts "winner: crosses=#{crosses}"
    return CROSS  if crosses == 3
  end
  false
end

def smart_move
  moves = legal_moves

  ROWS.each do |row|
    if (number_of(BLANK, row) == 1) then
      if (number_of(CROSS, row) == 2) then # If I have a win, take it.
```

```
        row.each do |e|
          return e if board[e] == BLANK
        end
      end
    end
  end
  ROWS.each do |row|
    if (number_of(BLANK, row) == 1) then
      if (number_of(CIRCLE,row) == 2) then # If he is threatening to win, stop it.
        row.each do |e|
          return e if board[e] == BLANK
        end
      end
    end
  end

  # Take the center if open.
  return "b2" if moves.include? "b2"

  # Defend opposite corners.
  if    self["a1"] != COMPUTER and self["a1"] != BLANK and self["c3"] == BLANK
    return "c3"
  elsif self["c3"] != COMPUTER and self["c3"] != BLANK and self["a1"] == BLANK
    return "a1"
  elsif self["a3"] != COMPUTER and self["a3"] != BLANK and self["c1"] == BLANK
    return "c1"
  elsif self["c1"] != COMPUTER and self["c3"] != BLANK and self["a3"] == BLANK
    return "a3"
  end

  # Or make a random move.
  moves[rand(moves.size)]
 end

 def human_wins?
  winner == HUMAN
 end

 def computer_wins?
  winner == COMPUTER
 end
end

helpers TicTacToe

get %r{^/([abc][123])?$} do |human|
 if human then
  puts "You played: #{human}!"
  puts "session: "
  pp session
  if legal_moves.include? human
    board[human] = TicTacToe::CIRCLE
    # computer = board.legal_moves.sample
```

```
      computer = smart_move
      redirect to ('/humanwins') if human_wins?
      redirect to('/') unless computer
      board[computer] = TicTacToe::CROSS
      puts "I played: #{computer}!"
      puts "Tablero:  #{board.inspect}"
      redirect to ('/computerwins') if computer_wins?
    end
  else
    session["bs"] = inicializa()
    puts "session = "
    pp session
  end
  haml :game, :locals => { :b => board, :m => ''  }
end

get '/humanwins' do
  puts "/humanwins session="
  pp session
  begin
    m = if human_wins? then
          'Human wins'
        else
          redirect '/'
        end
    haml :final, :locals => { :b => board, :m => m }
  rescue
    redirect '/'
  end
end

get '/computerwins' do
  puts "/computerwins"
  pp session
  begin
    m = if computer_wins? then
          'Computer wins'
        else
          redirect '/'
        end
    haml :final, :locals => { :b => board, :m => m }
  rescue
    redirect '/'
  end
end

not_found do
  puts "not found!!!!!!!!!!!"
  session["bs"] = inicializa()
  haml :game, :locals => { :b => board, :m => 'Let us start a new game'  }
end

get '/styles.css' do
```

```
  scss :styles
end
```

18.29. Práctica: TicTacToe usando DataMapper

Añada una base de datos a la práctica del TicTacToe 18.28 de manera que se lleve la cuenta de los usuarios registrados, las partidas jugadas, ganadas y perdídas. Repase la sección *DataMapper y Sinatra* 24.

Mejore las hojas de estilo usando SAAS **??**. Deberán mostrarse las celdas pares e impares en distintos colores. También deberá mostrarse una lista de jugadores con sus registros.

Despliegue la aplicación en Heroku.

18.30. Práctica: Servicio de Syntax Highlighting

Construya una aplicación que provee syntax higlighting para un código que se vuelca en un formulario. Use la gema syntaxi.

El siguiente ejemplo muestra como funciona la gema `syntaxi`:

```
[~/rubytesting/syntax_highlighting]$ cat ex_syntaxi.rb
require 'syntaxi'
text = <<"EOF"
[code lang="ruby"]
  def foo
    puts 'bar'
  end
[/code]
EOF
formatted_text = Syntaxi.new(text).process
puts formatted_text
```

Ejecución:

```
[~/rubytesting/syntax_highlighting]$ ruby ex_syntaxi.rb
<pre>
<code>
<span class="line_number">1</span> <span class="keyword">def </span><span class="method">foo</
<span class="line_number">2</span> <span class="ident">puts</span>
<span class="punct">'</span><span class="string">bar</span><span class="punct">'</span>
<span class="line_number">3</span> <span class="keyword">end</span>
</code>
</pre>
```

La gema `syntaxi` usa la gema `syntax`:

```
[~/rubytesting/syntax_highlighting]$ gem which syntaxi/Users/casiano/.rvm/gems/ruby-1.9.2-head
[~/rubytesting/syntax_highlighting]$ grep "require.*'" /Users/casiano/.rvm/gems/ruby-1.9.2-hea
require 'syntax/convertors/html'
```

Es en esta gema que se definen las hojas de estilo:

```
[~/rubytesting/syntax_highlighting]$ gem which syntax
/Users/casiano/.rvm/gems/ruby-1.9.2-head/gems/syntax-1.0.0/lib/syntax.rb
[~/rubytesting/syntax_highlighting]$ tree /Users/casiano/.rvm/gems/ruby-1.9.2-head/gems/syntax
/Users/casiano/.rvm/gems/ruby-1.9.2-head/gems/syntax-1.0.0/
|-- data
```

```
|   |-- ruby.css
|   |-- xml.css
|   `-- yaml.css
|-- lib
|   |-- syntax
|   |   |-- common.rb
|   |   |-- convertors
|   |   |   |-- abstract.rb
|   |   |   `-- html.rb
|   |   |-- lang
|   |   |   |-- ruby.rb
|   |   |   |-- xml.rb
|   |   |   `-- yaml.rb
|   |   `-- version.rb
|   `-- syntax.rb
`-- test
    |-- ALL-TESTS.rb
    |-- syntax
    |   |-- tc_ruby.rb
    |   |-- tc_xml.rb
    |   |-- tc_yaml.rb
    |   `-- tokenizer_testcase.rb
    `-- tc_syntax.rb

7 directories, 17 files
```

En el esquema incompleto que sigue se ha hecho para el lenguaje Ruby. Añada que se pueda elegir el lenguaje a colorear (xml, yaml).

```
$ tree -A
.
|-- Gemfile
|-- Gemfile.lock
|-- toopaste.rb
`-- views
    |-- layout.erb
    |-- new.erb
    `-- show.erb

$ cat Gemfile
source 'https://rubygems.org'

# Specify your gem's dependencies in my-gem.gemspec
# gemspec
# gem 'guard'
# gem 'guard-rspec'
# gem 'guard-bundler'
# gem 'rb-fsevent', '~> 0.9.1''

gem 'syntaxi'
```

Este es un fragmento de la aplicación:

```
[~/srcSTW/syntax_highlighting(withoutdm)]$ cat  toopaste.rb
require 'sinatra'
require 'syntaxi'

class String
  def formatted_body
    source = "[code lang='ruby']
                #{self}
              [/code]"
    html = Syntaxi.new(source).process
    %Q{
      <div class="syntax syntax_ruby">
        #{html}
      </div>
    }
  end
end

get '/' do
  erb :new
end

post '/' do
  .....
end
```

Una versión simple de lo que puede ser new.erb:

```
[~/srcSTW/syntax_highlighting(withoutdm)]$ cat views/new.erb
<div class="snippet">
  <form action="/" method="POST">
    <textarea name="body" id="body" rows="20"></textarea>
    <br/><input type="submit" value="Save"/>
  </form>
</div>
```

Véase la página HTML generada por el programa para la entrada `a = 5`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DT
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Toopaste!</title>
  <style>
    html {
      background-color: #eee;
    }
    .snippet {
      margin: 5px;
    }
    .snippet textarea, .snippet .sbody {
      border: 5px dotted #eee;
      padding: 5px;
      width: 700px;
      color: #fff;
```

```
      background-color: #333;
    }
    .snippet textarea {
      padding: 20px;
    }
    .snippet input, .snippet .sdate {
      margin-top: 5px;
    }

    /* Syntax highlighting */
    #content .syntax_ruby .normal {}
    #content .syntax_ruby .comment { color: #CCC; font-style: italic; border: none; margin: no
    #content .syntax_ruby .keyword { color: #C60; font-weight: bold; }
    #content .syntax_ruby .method { color: #9FF; }
    #content .syntax_ruby .class { color: #074; }
    #content .syntax_ruby .module { color: #050; }
    #content .syntax_ruby .punct { color: #0D0; font-weight: bold; }
    #content .syntax_ruby .symbol { color: #099; }
    #content .syntax_ruby .string { color: #C03; }
    #content .syntax_ruby .char { color: #F07; }
    #content .syntax_ruby .ident { color: #0D0; }
    #content .syntax_ruby .constant { color: #07F; }
    #content .syntax_ruby .regex { color: #B66; }
    #content .syntax_ruby .number { color: #FF0; }
    #content .syntax_ruby .attribute { color: #7BB; }
    #content .syntax_ruby .global { color: #7FB; }
    #content .syntax_ruby .expr { color: #909; }
    #content .syntax_ruby .escape { color: #277; }
    #content .syntax {
      background-color: #333;
      padding: 2px;
      margin: 5px;
      margin-left: 1em;
      margin-bottom: 1em;
    }
    #content .syntax .line_number {
      text-align: right;
      font-family: monospace;
      padding-right: 1em;
      color: #999;
    }
  </style>
</head>
<body>
  <div class="snippet">
  <div class="snippet">
  <div class="sbody" id="content">
      <div class="syntax syntax_ruby">
        <pre>
           <code>
              <span class="line_number">1</span>
              <span class="ident">a</span>
              <span class="punct">=</span>
```

```
              <span class="number">5</span>
            </code>
          </pre>
      </div>
  </div>
  <br/><a href="/">New Paste!</a>
</div>
</body>
</html>
```

La gema

Una versión resumida de `layout.erb`:

```
[~/srcSTW/syntax_highlighting(withoutdm)]$ cat views/layout.erb
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DT
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title><%= @title || 'Toopaste!' %></title>
  <style>
    ..............
  </style>
</head>
<body>
  <%= yield %>
</body>
</html>
```

Capítulo 19

Sinatra desde Dentro

19.1. tux

- tux en GitHub
- Tux: a Sinatra Console
- Tux documentación

19.2. Aplicación y Delegación

19.3. Helpers y Extensiones

19.4. Petición y Respuesta

Capítulo 20

Aplicaciones Modulares

Las aplicaciones normales Sinatra se denominan *alicaciones clásicas sinatra* y viven en Sinatra::Application, que es una subclase de Sinatra::Base.

En las aplicaciones clásicas Sinatra extiende la clase `Object` en el momento de cargarse lo que, en cierto modo, contamina el espacio de nombres global. Eso dificulta que nuestra aplicación pueda ser distribuída como una gema y que se puedan tener varias aplicaciones clásicas en un único proceso.

Una aplicación Sinatra se dice una *aplicación modular sinatra* si no hace uso de Sinatra::Application, renunciando al DSL de alto nivel proveído por Sinatra, sino que hereda de Sinatra::Base.

Podemos combinar una aplicación clásica con una modular, pero sólo puede haber una aplicación clásica por proceso.

Capítulo 21

Testing en Sinatra

1. Build a Sinatra API Using TDD, Heroku, and Continuous Integration with Travis by Darren Jones. Publis SitePoint
2. Código del artículo anterior
3. Mini MiniTest Tutorial by Tim Millwood

   We'll use MiniTest::Spec, which is a functionally complete spec engine, to create a spec for a Hello World! Sinatra application. Firstly we create a folder 'spec' within the application directory. Within this, two files.

   ```
   ENV['RACK_ENV'] = 'test'

   require 'minitest/autorun'
   require 'rack/test'
   require_relative '../app'

   include Rack::Test::Methods

   def app
     Sinatra::Application
   end
   ```

   The above code is to go into the file `spec_helper.rb`. It sets up the initial setting for the test. We set the `RACK_ENV` environment variable to 'test' then require '`minitest/autorun`' for MiniTest, 'rack/test' because we are testing a rack based application and our application, in this case app.rb. The Rack::Test::Methods module is included to add a number of helper methods to allow the testing of rack applications. Finally we define the application as a Sinatra Application. The next file `app_spec.rb` will be our spec.

   ```
   require_relative 'spec_helper'

   describe 'Hello World' do

     it 'should have hello world' do
       get '/'
       last_response.must_be :ok?
       last_response.body.must_include "Hello world!"
     end
   end
   ```

Firstly the `spec_helper` file is required, we then create a describe block to describe the 'Hello World' application. Within this a single behaviour. We run a get method against the root route and check the response is ok, and that the page includes the text 'Hello World!'.

Capítulo 22

CoffeeScript y Sinatra

```
[~/Dropbox/src/ruby/sinatra/sinatra-coffeescript]$ tree
.
|-- app.rb
`-- views
    `-- application.coffee

1 directory, 2 files
[~/Dropbox/src/ruby/sinatra/sinatra-coffeescript]$ cat app.rb
## You'll need to require coffee-script in your app
require 'sinatra'
require 'coffee-script'

get '/application.js' do
  x = coffee :application
  "<script>#{x}</script>"
end

[~/Dropbox/src/ruby/sinatra/sinatra-coffeescript]$ cat views/application.coffee
alert "hello world!"
```

Capítulo 23

Openid y Sinatra

OpenID provides sites and services with a decentralized protocol for authenticating users through a wide variety of providers. What this means is that a site integrating OpenID can allow its users to log in using, for example, their Yahoo!, Google, or AOL accounts. Not only can the consuming site avoid having to create a login system itself, but it can also take advantage of the accounts that its users already have, thereby in- creasing user registration and login rates.

In addition to simple authentication, OpenID also offers a series of extensions through which an OpenID provider can allow sites to obtain a user's profile information or integrate additional layers of security for the login procedure.

What makes OpenID so intriguing is the fact that it offers a standard that is fully decentralized from the providers and consumers. This aspect is what allows a single consuming site to allow its users to log in via Yahoo! and Google, while another site may want to allow logins via Blogger or WordPress. Ultimately, it is up to the OpenID consumer (your site or service) to choose what login methods it would like to offer its user base.

23.1. Referencias. Véase Tambien

- GitHub ahx/sinatra-openid-consumer-example
- Google Offers Named OpenIDs por Jeff Atwood
- How do I log in with OpenID?
- Programming Social Applications por Jonathan Leblanc. O'Reilly. 2011.

Capítulo 24

DataMapper y Sinatra

24.1. Introducción a Los Object Relational Mappers (ORM)

What is a Object Relational Mapper?

A simple answer is that you wrap your tables or stored procedures in classes in your programming language, so that instead of writing SQL statements to interact with your database, you use methods and properties of objects.

In other words, instead of something like this:

```
String sql = "SELECT ... FROM persons WHERE id = 10"
DbCommand cmd = new DbCommand(connection, sql);
Result res = cmd.Execute();
String name = res[0]["FIRST_NAME"];
```

you do something like this:

```
Person p = Person.Get(10);
```

or similar code (lots of variations here). The framework is what makes this code possible.

Now, benefits:

1. First of all, you hide the SQL away from your logic code
2. This has the benefit of allowing you to more easily support more database engines
3. For instance, MS SQL Server and Oracle have different names on typical functions, and different ways to do calculations with dates. This difference can be put away from your logic code.
4. Additionally, you can focus on writing the logic, instead of getting all the SQL right.
5. The code will typically be more readable as well, since it doesn't contain all the plumbing necessary to talk to the database.

24.2. Introducción al Patrón DataMapper

Martin Fowler (Catalog of Patterns of Enterprise Application Architecture):

1. Objects and relational databases have different mechanisms for structuring data.
2. Many parts of an object, such as collections and inheritance, aren't present in relational databases.
3. When you build an object model with a lot of business logic it's valuable to use these mechanisms (creo que se refiere a la herencia, etc.) to better organize the data and the behavior that goes with it.

4. Doing so leads to variant schemas; that is, the object schema and the relational schema don't match up.
5. You still need to transfer data between the two schemas, and this data transfer becomes a complexity in its own right.
6. If the in-memory objects know about the relational database structure, changes in one tend to ripple to the other.
7. The *Data Mapper* is a layer of software that separates the in-memory objects from the database.
8. *Its responsibility is to transfer data between the two and also to isolate them from each other*
9. With Data Mapper the in-memory objects needn't know even that there's a database present; they need no SQL interface code, and certainly no knowledge of the database schema.
10. (The database schema is always ignorant of the objects that use it.)

- DataMapper en la Wikipedia
- Martin Fowler: DataMapper
- Proyecto sinatra-datamapper-sample en GitHub
- Documentación de DataMapper
- Sinatra Recipes: DataMapper
- Sinatra Book: DataMapper

24.3. Ejemplo de Uso de DataMapper

Donde

- ```
  [~/sinatra/sinatra-datamapper-jump-start(master)]$ pwd -P
  /Users/casiano/local/src/ruby/sinatra/sinatra-datamapper-jump-start
  ```
- ```
  [~/sinatra/sinatra-datamapper-jump-start(master)]$ git remote -v
  origin  git@github.com:crguezl/sinatra-datamapper-jump-start.git (fetch)
  origin  git@github.com:crguezl/sinatra-datamapper-jump-start.git (push)
  ```
- Este ejemplo en GitHub
- http://sinadm.herokuapp.com/ (Puede que este caída)

Enlaces

1. Documentación del módulo DataMapper en RubyDoc
2. https://github.com/crguezl/datamapper_example
3. https://github.com/crguezl/datamapper-intro

La Clase Song

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ cat song.rb
require 'dm-core'
require 'dm-migrations'

class Song
  include DataMapper::Resource
  property :id, Serial
  property :title, String
  property :lyrics, Text
  property :length, Integer

end
```

The `Song` model is going to need to be persistent, so we'll include `DataMapper::Resource`.

The convention with model names is to use the singular, not plural version... but that's just the convention, we can do whatever we want.

```
configure do
  enable :sessions
  set :username, 'frank'
  set :password, 'sinatra'
end
```

DataMapper.finalize

```
DataMapper.finalize
```

This method performs the necessary steps to finalize `DataMapper` for the current repository. It should be called after loading all models and plugins. It ensures foreign key properties and anonymous join models are created. These are otherwise lazily declared, which can lead to unexpected errors. It also performs basic validity checking of the `DataMapper` models.

Mas código de Song.rb

```
get '/songs' do
  @songs = Song.all
  slim :songs
end

get '/songs/new' do
  halt(401,'Not Authorized') unless session[:admin]
  @song = Song.new
  slim :new_song
end

get '/songs/:id' do
  @song = Song.get(params[:id])
  slim :show_song
end

get '/songs/:id/edit' do
  @song = Song.get(params[:id])
  slim :edit_song
end
```

Song.create

If you want to create a new resource with some given attributes and then save it all in one go, you can use the `#create` method:

```
post '/songs' do
  song = Song.create(params[:song])
  redirect to("/songs/#{song.id}")
end

put '/songs/:id' do
  song = Song.get(params[:id])
  song.update(params[:song])
  redirect to("/songs/#{song.id}")
end

delete '/songs/:id' do
  Song.get(params[:id]).destroy
  redirect to('/songs')
end
```

Una sesión con pry probando DataMapper

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ pry
[1] pry(main)> require 'sinatra'
=> true
[2] pry(main)> require './song'
=> true
```

DataMapper.setup

We must specify our database connection.

We need to make sure to do this before you use our models, i.e. before we actually start accessing the database.

```
  # If you want the logs displayed you have to do this before the call to setup
  DataMapper::Logger.new($stdout, :debug)

  # An in-memory Sqlite3 connection:
  DataMapper.setup(:default, 'sqlite::memory:')

  # A Sqlite3 connection to a persistent database
  DataMapper.setup(:default, 'sqlite:///path/to/project.db')

  # A MySQL connection:
  DataMapper.setup(:default, 'mysql://user:password@hostname/database')

  # A Postgres connection:
  DataMapper.setup(:default, 'postgres://user:password@hostname/database')
Note: that currently you must setup a :default repository to work with DataMapper (and to be a
```

In our case:

```
[4] pry(main)> pry(main)> DataMapper.setup(:default,'sqlite:development.db')
```

Multiple Data-Store Connections

DataMapper sports a concept called a context which encapsulates the data-store context in which you want operations to occur. For example, when you setup a connection you are defining a context known as `:default`

```
DataMapper.setup(:default, 'mysql://localhost/dm_core_test')
```

If you supply another context name, you will now have 2 database contexts with their own unique loggers, connection pool, identity map....one *default context* and one *named context*.

```
DataMapper.setup(:external, 'mysql://someother_host/dm_core_test')
```

To use one context rather than another, simply wrap your code block inside a `repository` call. It will return whatever your block of code returns.

```
DataMapper.repository(:external) { Person.first }
# hits up your :external database and retrieves the first Person
```

This will use your connection to the `:external` data-store and the first `Person` it finds. Later, when you call `.save` on that person, it'll get saved back to the `:external` data-store; An object is aware of what context it came from and should be saved back to.

El Objeto DataMapper::Adapters

```
=> #<DataMapper::Adapters::SqliteAdapter:0x007fad2c0f6a50
 @field_naming_convention=DataMapper::NamingConventions::Field::Underscored,
 @name=:default,
 @normalized_uri=
  #<DataObjects::URI:0x007fad2c0f62a8
   @fragment="{Dir.pwd}/development.db",
   @host="",
   @password=nil,
   @path=nil,
   @port=nil,
   @query=
    {"scheme"=>"sqlite3",
     "user"=>nil,
     "password"=>nil,
     "host"=>"",
     "port"=>nil,
     "query"=>nil,
     "fragment"=>"{Dir.pwd}/development.db",
     "adapter"=>"sqlite3",
     "path"=>nil},
   @relative=nil,
   @scheme="sqlite3",
   @subscheme=nil,
   @user=nil>,
 @options=
  {"scheme"=>"sqlite3",
   "user"=>nil,
   "password"=>nil,
   "host"=>"",
   "port"=>nil,
   "query"=>nil,
```

```
   "fragment"=>"{Dir.pwd}/development.db",
   "adapter"=>"sqlite3",
   "path"=>nil},
 @resource_naming_convention=
  DataMapper::NamingConventions::Resource::UnderscoredAndPluralized>
```

Creando las tablas con DataMapper.auto_migrate! We can create the table by issuing the following command:

```
[4] pry(main)> DataMapper.auto_migrate!
```

1. This will issue the necessary `CREATE` statements (`DROP`ing the table first, if it exists) to define each storage according to their properties.
2. After `auto_migrate!` has been run, the database should be in a pristine state.
3. All the tables will be empty and match the model definitions.

DataMapper.auto_upgrade! This wipes out existing data, so you could also do:

```
DataMapper.auto_upgrade!
```

1. This tries to make the schema match the model.
2. It will `CREATE` new tables, and add columns to existing tables.
3. It won't change any existing columns though (say, to add a `NOT NULL` constraint) and it doesn't drop any columns.
4. Both these commands also can be used on an individual model (e.g. `Song.auto_migrate!`)

Métodos de la Clase Mapeada

```
[5] pry(main)> song = Song.new
=> #<Song @id=nil @title=nil @lyrics=nil @length=nil @released_on=nil>
[6] pry(main)> song.save
=> true
[7] pry(main)> song
=> #<Song @id=1 @title=<not loaded> @lyrics=<not loaded> @length=<not loaded> @released_on=<no
[8] pry(main)> song.title = "My Way"
=> "My Way"
[9] pry(main)> song.lyrics
=> nil
[10] pry(main)> song.lyrics = "And now, the end is near ..."
=> "And now, the end is near ..."
[11] pry(main)> song.length = 435
=> 435
[42] pry(main)> song.save
=> true
[43] pry(main)> song
=> #<Song @id=1 @title="My Way" @lyrics="And now, the end is near ..." @length=435 @released_o
```

El método create If you want to create a new resource with some given attributes and then save it all in one go, you can use the #create method.

```
[28] pry(main)> Song.create(title: "Come fly with me", lyrics: "Come fly with me, let's fly, l
=> #<Song @id=2 @title="Come fly with me" @lyrics="Come fly with me, let's fly, let's fly away
```

1. If the creation was successful, `#create` will return the newly created `DataMapper::Resource`
2. If it failed, it will return a new resource that is initialized with the given attributes and possible default values declared for that resource, but that's not yet saved
3. To find out wether the creation was successful or not, you can call `#saved?` on the returned resource
4. It will return `true` if the resource was successfully persisted, or `false` otherwise

first_or_create If you want to either find the first resource matching some given criteria or just create that resource if it can't be found, you can use `#first_or_create`.

```
s = Song.first_or_create(:title => 'New York, New York')
```

This will first try to find a `Song` instance with the given `title`, and if it fails to do so, it will return a newly created `Song` with that `title`.

If the criteria you want to use to query for the resource differ from the attributes you need for creating a new resource, you can pass the attributes for creating a new resource as the second parameter to `#first_or_create`, also in the form of a `#Hash`.

```
s = Song.first_or_create({ :title => 'My Way' }, { :lyrics => '... the end is not near' })
```

This will search for a `Song` named 'My Way' and if it can't find one, it will return a new `Song` instance with its name set to 'My Way' and the `lyrics` set to `.. the end is not near`

1. You can see that for creating a new resource, both hash arguments will be merged so you don't need to specify the query criteria again in the second argument Hash that lists the attributes for creating a new resource
2. However, if you really need to create the new resource with different values from those used to query for it, the second Hash argument will overwrite the first one.

```
s = Song.first_or_create({ :title => 'My Way' }, {
  :title  => 'My Way Home',
  :lyrics => '... the end is not near'
})
```

This will search for a `Song` named `'My Way'` but if it fails to find one, it will return a `Song` instance with its title set to `'My Way Home'` and its `lyrics` set to `'... the end is not near'`.

Comprobando con sqlite3

Podemos abrir la base de datos con el gestor de base de datos y comprobar que las tablas y los datos están allí:

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ sqlite3 development.db
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema
CREATE TABLE "songs" ("id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
```

```
      "title" VARCHAR(50), "lyrics" TEXT, "length"
      INTEGER, "released_on" TIMESTAMP);
sqlite> select * from songs;
1|My Way|And now, the end is near ...|435|
2|Come fly with me|Come fly with me, let's fly, let's fly away ...|199|
sqlite>
```

Búsquedas y Consultas `DataMapper` has methods which allow you to grab a single record by key, the first match to a set of conditions, or a collection of records matching conditions.

```
song  = Song.get(1)                     # get the song with primary key of 1.
song  = Song.get!(1)                    # Or get! if you want an ObjectNotFoundError on failur
song  = Song.first(:title => 'Girl')    # first matching record with the title 'Girl'
song  = Song.last(:title => 'Girl')     # last matching record with the title 'Girl'
songs = Song.all                        # all songs
```

```
[29] pry(main)> Song.count
=> 2
[30] pry(main)> Song.all
=> [#<Song @id=1 @title=nil @lyrics=<not loaded> @length=nil @released_on=nil>, #<Song @id=2 @
[31] pry(main)> Song.get(1)
=> #<Song @id=1 @title=nil @lyrics=<not loaded> @length=nil @released_on=nil>
[32] pry(main)> Song.first
=> #<Song @id=1 @title=nil @lyrics=<not loaded> @length=nil @released_on=nil>
[33] pry(main)> Song.last
=> #<Song @id=2 @title="Come fly with me" @lyrics=<not loaded> @length=199 @released_on=nil>
[35] pry(main)> x = Song.first(title: 'Come fly with me')
=> #<Song @id=2 @title="Come fly with me" @lyrics=<not loaded> @length=199 @released_on=nil>
```

```
[44] pry(main)> y = Song.first(title: 'My Way')
=> #<Song @id=1 @title="My Way" @lyrics=<not loaded> @length=435 @released_on=nil>
[45] pry(main)> y.length
=> 435
[46] pry(main)> y.update(length: 275)
=> true
```

En Sqlite3:

```
sqlite> select * from songs;
1|My Way|And now, the end is near ...|275|
2|Come fly with me|Come fly with me, let's fly, let's fly away ...|199|
```

Borrando

```
[47] pry(main)> Song.create(title: "One less lonely girl")
=> #<Song @id=3 @title="One less lonely girl" @lyrics=<not loaded> @length=<not loaded> @relea
[48] pry(main)> Song.last.destroy
=> true
[49] pry(main)> Song.all
=> [#<Song @id=1 @title="My Way" @lyrics=<not loaded> @length=275 @released_on=nil>, #<Song @i
```

Búsqueda con Condiciones Rather than defining conditions using SQL fragments, we can actually specify conditions using a hash.

The examples above are pretty simple, but you might be wondering how we can specify conditions beyond equality without resorting to SQL. Well, thanks to some clever additions to the `Symbol` class, it's easy!

```
exhibitions = Exhibition.all(:run_time.gt => 2, :run_time.lt => 5)
# => SQL conditions: 'run_time > 1 AND run_time < 5'
```

Valid symbol operators for the conditions are:

```
gt    # greater than
lt    # less than
gte   # greater than or equal
lte   # less than or equal
not   # not equal
eql   # equal
like  # like
```

Veamos un ejemplo de uso con nuestra clase `Song`:

```
[31] pry(main)> Song.all.each do |s|
[31] pry(main)*   s.update(length: rand(400))
[31] pry(main)* end
=> [#<Song @id=1 @title="My Way" @lyrics=<not loaded> @length=122 @released_on=nil>,
   #<Song @id=2 @title="Come fly with me" @lyrics=<not loaded> @length=105 @released_on=nil>,
   #<Song @id=4 @title="Girl from Ipanema" @lyrics=<not loaded> @length=389 @released_on=nil>]
[32] pry(main)> long = Song.all(:length.gt => 120)
=> [#<Song @id=1 @title="My Way" @lyrics=<not loaded> @length=122 @released_on=nil>,
   #<Song @id=4 @title="Girl from Ipanema" @lyrics=<not loaded> @length=389 @released_on=nil>]
```

Insertando SQL Sometimes you may find that you need to tweak a query manually:

```
[40] pry(main)> songs = repository(:default).adapter.select('SELECT title FROM songs WHERE len
=> ["My Way", "Girl from Ipanema"]
```

Note that this will not return `Song` objects, rather the raw data straight from the database

main.rb

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ cat main.rb
require 'sinatra'
require 'slim'
require 'sass'
require './song'

configure do
  enable :sessions
  set :username, 'frank'
  set :password, 'sinatra'
end

configure :development do
  DataMapper.setup(:default, "sqlite3://#{Dir.pwd}/development.db")
end

configure :production do
  DataMapper.setup(:default, ENV['DATABASE_URL'])
end

get('/styles.css'){ scss :styles }
```

```
get '/' do
  slim :home
end

get '/about' do
  @title = "All About This Website"
  slim :about
end

get '/contact' do
  slim :contact
end

not_found do
  slim :not_found
end

get '/login' do
  slim :login
end

post '/login' do
  if params[:username] == settings.username && params[:password] == settings.password
    session[:admin] = true
    redirect to('/songs')
  else
    slim :login
  end
end

get '/logout' do
  session.clear
  redirect to('/login')
end
```

24.4. Configurando la Base de Datos en Heroku con DataMapper. Despliegue

Heroku utiliza la base de datos PostgreSQL y una URL en una variable de entorno `ENV['DATABASE_URL']`.

```
configure :development do
  DataMapper.setup(:default, "sqlite3://#{Dir.pwd}/development.db")
end

configure :production do
  DataMapper.setup(:default, ENV['DATABASE_URL'])
end
```

Estas líneas especifican que se usa SQLite en desarrollo y PostgreSQL en producción. Obsérvese que el `Gemfile` debe estar coherente:

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ cat Gemfile
source 'https://rubygems.org'
```

```
gem "sinatra"
gem "slim"
gem "sass"
gem "dm-core"
gem "dm-migrations"
gem "thin"
gem "pg", :group => :production
gem "dm-postgres-adapter", :group => :production
gem "dm-sqlite-adapter", :group => :development
```

o mejor:

```
group :production do
    gem "pg"
    gem "dm-postgres-adapter"
end
```

```
heroku create ...
```

```
git push heroku master
```

```
heroku open
```

```
heroku logs --source app
```

Ahora ejecutamos la consola de heroku:

```
heroku run console
```

lo que nos abre una sesión `irb`.

Ahora creamos la base de datos en Heroku:

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku run console
Running `console` attached to terminal... up, run.8011
irb(main):001:0> require './main'
=> true
irb(main):002:0> DataMapper.auto_migrate!
=> #<DataMapper::DescendantSet:0x007fb89c878230 @descendants=#<DataMapper::SubjectSet:0x007fb8
irb(main):003:0>
```

Véase también la practica TicTacToe 18.28 y el capítulo *Despliegue en Heroku* **??**. .

Capítulo 25

Depuración en Sinatra

25.1. Depurando una Ejecución con Ruby

```
[~/sinatra/sinatra-debug/example1]$ ls
Gemfile            Rakefile           my_sinatra.rb
Gemfile.lock       config.ru          rackmiddleware.rb

[~/sinatra/sinatra-debug/example1]$ cat Gemfile
source 'http://rubygems.org'

group :development, :test do
  gem 'awesome_print'
  gem 'racksh'
  gem 'debugger'
  gem 'pry'
  gem 'pry-debugger'
end

[~/sinatra/sinatra-debug/example1]$ cat my_sinatra.rb
# my_sinatra.rb
require 'debugger'
require 'sinatra'
require './rackmiddleware'
use RackMiddleware
get '/:p' do |x|
   # debugger
  "Welcome to #{x}"
end

[~/sinatra/sinatra-debug/example1]$ cat rackmiddleware.rb
class RackMiddleware
  def initialize(appl)
    @appl = appl
  end

  def call(env)
    debugger
    start = Time.now
    status, headers, body = @appl.call(env) # call our Sinatra app
    stop = Time.now
    puts "Response Time: #{stop-start}" # display on console
    [status, headers, body]
```

```
  end
end

[~/sinatra/sinatra-debug/example1]$ ruby my_sinatra.rb
== Sinatra/1.4.3 has taken the stage on 4567 for development with backup from Thin
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on localhost:4567, CTRL+C to stop
```

Al conectar al servidor queda en espera:

```
[~/sinatra/sinatra-debug]$ curl 'http://localhost:4567/canarias'
```

En la otra terminal el servidor se detiene en el primer breakpoint señalado:

```
[~/sinatra/sinatra-debug/example1]$ ruby my_sinatra.rb
== Sinatra/1.4.3 has taken the stage on 4567 for development with backup from Thin
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on localhost:4567, CTRL+C to stop
/Users/casiano/Dropbox/src/ruby/sinatra/sinatra-debug/example1/rackmiddleware.rb:8
start = Time.now

[3, 12] in /Users/casiano/Dropbox/src/ruby/sinatra/sinatra-debug/example1/rackmiddleware.rb
   3      @appl = appl
   4    end
   5
   6    def call(env)
   7      debugger
=> 8      start = Time.now
   9      status, headers, body = @appl.call(env) # call our Sinatra app
   10      stop = Time.now
   11      puts "Response Time: #{stop-start}" # display on console
   12      [status, headers, body]
```

Ahora podemos ir paso a paso e inspeccionar variables:

```
(rdb:1) p start
2013-07-04 15:40:11 +0100
(rdb:1) n
/Users/casiano/Dropbox/src/ruby/sinatra/sinatra-debug/example1/rackmiddleware.rb:10
stop = Time.now

[5, 14] in /Users/casiano/Dropbox/src/ruby/sinatra/sinatra-debug/example1/rackmiddleware.rb
   5
   6    def call(env)
   7      debugger
   8      start = Time.now
   9      status, headers, body = @appl.call(env) # call our Sinatra app
=> 10      stop = Time.now
   11      puts "Response Time: #{stop-start}" # display on console
   12      [status, headers, body]
   13    end
   14  end
(rdb:1) p body
["Welcome to canarias"]
```

```
(rdb:1) p status
200
(rdb:1) p headers
{"Content-Type"=>"text/html;charset=utf-8", "Content-Length"=>"19"}
```

Capítulo 26

Envío de SMSs y Mensajes: Twilio y Clockworks

1. How to create a Twilio app on Heroku de Morten Baga
2. Twilio HackPack for Heroku and Sinatra
3. heroku-twilio A fork of the Twilio node library that is Heroku friendly
4. Twilio HackPack for Sinatra and Heroku Posted by Oscar Sanchez on July 05, 2012 in Tips, Tricks and Sa

1. Clockworks SMS
2. Documentación de Clockworks SMS
3. Ruby gem para Clockworks. En Github

```
require 'clockwork'

api = Clockwork::API.new( 'API_KEY_GOES_HERE' )
message = api.messages.build( :to => '441234123456', :content => 'This is a test message.' )
response = message.deliver

if response.success
    puts response.message_id
else
    puts response.error_code
    puts response.error_description
end
```

Capítulo 27

Rest

Uno! Use Sinatra to Implement a REST API. by Dan Schaefer. Published March 10, 2014

Capítulo 28

Sinatra + Sprockets

- Sinatra + Sprockets
- Sprockets
- https://github.com/crguezl/sinatra-sprockets-slim-backbone-example

Capítulo 29

Sinatra::Flash

Sinatra::Flash is an extension that lets you store information between requests.

Often, when an application processes a request, it will redirect to another URL upon finishing, which generates another request.

This means that any information from the previous request is lost (due to the stateless nature of HTTP).

Sinatra::Flash overcomes this by providing access to the flash—a hash-like object that stores temporary values such as error messages so that they can be retrieved later—usually on the next request.

It also removes the information once it's been used.

All this can be achieved via sessions (and that's exactly how Sinatra::Flash does it), but Sinatra::Flash is easy to implement and provides a number of helper methods.

Ejemplo

```
[~/sinatra/sinatra-flash]$ cat app.rb
require 'sinatra'
require 'sinatra/flash'

enable :sessions

get '/blah' do
  # This message won't be seen until the NEXT Web request that accesses the flash collection
  flash[:blah] = "You were feeling blah at #{Time.now}."

  # Accessing the flash displays messages set from the LAST request
  "Feeling blah again? That's too bad. #{flash[:blah]}"
end

get '/pum' do
  # This message won't be seen until the NEXT Web request that accesses the flash collection
  flash[:pum] = "You were feeling pum at #{Time.now}."

  # Accessing the flash displays messages set from the LAST request
  "Feeling pum again? That's too bad. #{flash[:pum]}"
end
```

Gemfile

```
[~/sinatra/sinatra-flash]$ cat Gemfile
source 'https://rubygems.org'

gem 'sinatra'
```

```
gem 'sinatra-flash'
```

Capítulo 30

Pruebas

1. NetTuts+ tutorial: Testing Web Apps with Capybara and Cucumber Andrew Burgess on Aug 22nd 2011 with 22 Comments

2.

Parte IV

PARTE: HERRAMIENTAS

Capítulo 31

Heroku

31.1. Introducción

Prerequisitos Estos son los prerequisitos (Octubre 2013)

1. Basic Ruby knowledge, including an installed version of Ruby 2.0.0, Rubygems, and Bundler.
2. Basic Git knowledge
3. Your application must run on Ruby (MRI) 2.0.0.
4. Your application must use Bundler.
5. A Heroku user account.

Instala el Heroku Toolbelt

1. Crea una cuenta en Heroku
2. El Heroku Toolbelt se compone de:
   a) Heroku client - CLI tool for creating and managing Heroku apps
   b) Foreman - an easy option for running your apps locally
   c) Git - revision control and pushing to Heroku

La primera vez te pedirá las credenciales:

```
$ heroku login
Enter your Heroku credentials.
Email: adam@example.com
Password:
Could not find an existing public key.
Would you like to generate one? [Yn]
Generating new SSH public key.
Uploading ssh public key /Users/adam/.ssh/id_rsa.pub
```

La clave la cargas en la sección `SSH keys add key` de https://dashboard.heroku.com/account

```
[~/rack/rack-rock-paper-scissors(test)]$ heroku --version
heroku-gem/2.39.4 (x86_64-darwin11.4.2) ruby/1.9.3
```

```
[~/local/src/ruby/sinatra/rack/rack-rock-paper-scissors(test)]$  which heroku
/Users/casiano/.rvm/gems/ruby-1.9.3-p392/bin/heroku
[~/local/src/ruby/sinatra/rack/rack-rock-paper-scissors(test)]$ ruby -v
ruby 1.9.3p392 (2013-02-22 revision 39386) [x86_64-darwin11.4.2]
```

Seguramente tienes que instalar una versión del toolbet por cada versión de Ruby con la que quieras usarlo.

Para desinstalarlo:

```
$ gem uninstall heroku --all
```

Actualizaciones The Heroku Toolbelt will automatically keep itself up to date.

1. When you run a heroku command, a background process will be spawned that checks a URL for the latest available version of the CLI.

2. If a new version is found, it will be downloaded and stored in `~/.heroku/client`.

3. This background check will happen at most once every 5 minutes.

4. The heroku binary will check for updated clients in `~/.heroku/client` before loading the system-installed version.

Ayuda

```
[~/local/src/ruby/sinatra/rack/rack-rock-paper-scissors(master)]$ heroku --help
Usage: heroku COMMAND [--app APP] [command-specific-options]

Primary help topics, type "heroku help TOPIC" for more details:

  addons    #  manage addon resources
  apps      #  manage apps (create, destroy)
  auth      #  authentication (login, logout)
  config    #  manage app config vars
  domains   #  manage custom domains
  logs      #  display logs for an app
  ps        #  manage dynos (dynos, workers)
  releases  #  manage app releases
  run       #  run one-off commands (console, rake)
  sharing   #  manage collaborators on an app

Additional topics:

  account      #  manage heroku account options
  certs        #  manage ssl endpoints for an app
  db           #  manage the database for an app
  drains       #  display syslog drains for an app
  fork         #  clone an existing app
  git          #  manage git for apps
  help         #  list commands and display help
  keys         #  manage authentication keys
  labs         #  manage optional features
  maintenance  #  manage maintenance mode for an app
  pg           #  manage heroku-postgresql databases
  pgbackups    #  manage backups of heroku postgresql databases
  plugins      #  manage plugins to the heroku gem
  regions      #  list available regions
  stack        #  manage the stack for an app
  status       #  check status of heroku platform
  update       #  update the heroku client
  version      #  display version
```

Specify Ruby Version and Declare dependencies with a Gemfile

Heroku recognizes an app as Ruby by the existence of a `Gemfile`.

Even if your app has no gem dependencies, you should still create an empty `Gemfile` in order that it appear as a Ruby app.

In local testing, you should be sure to run your app in an isolated environment (via `bundle exec` or an empty RVM gemset), to make sure that all the gems your app depends on are in the `Gemfile`.

In addition to specifying dependencies, you'll want to specify your Ruby Version using the ruby DSL provided by Bundler.

Here's an example `Gemfile` for a Sinatra app:

```
source "https://rubygems.org"
ruby "2.0.0"
gem 'sinatra', '1.1.0'
```

```
[~/sinatra/rockpaperscissors(master)]$ cat Gemfile
source 'https://rubygems.org'
gem 'sinatra'
gem 'haml'
gem 'puma'
```

Run `bundle install` to set up your bundle locally.

1. Run:

   ```
   $ bundle install
   ```

2. This ensures that all gems specified in Gemfile, together with their dependencies, are available for your application.

3. Running bundle install also generates a `Gemfile.lock` file, *which should be added to your git repository.*

4. `Gemfile.lock` ensures that your deployed versions of gems on Heroku match the version installed locally on your development machine.

Declare process types with `Procfile`

Process types are declared via a file named `Procfile` placed in the root of your app.

Its format is one process type per line, with each line containing:

```
<process type>: <command>
```

The syntax is defined as:

1. `<process type>` – an alphanumeric string, is a name for your command, such as
   a) `web`,
   b) `worker`,
   c) `urgentworker`,
   d) `clock`, etc.

2. `<command>` – a command line to launch the process, such as `rake jobs:work`.

The `web` process type is special as it's the only process type that will receive HTTP traffic from Heroku's routers.

1. Use a `Procfile`, a text file in the root directory of your application, to explicitly declare what command should be executed to start a *web dyno*.

2. Assume for instance, that we wanto to execute `web.rb` using Ruby. Here's a `Procfile`:

   ```
   web: bundle exec ruby web.rb -p $PORT
   ```

3. If we are instead deploying a straight Rack app, here's a Procfile that can execute our config.ru:

   ```
   web: bundle exec rackup config.ru -p $PORT
   ```

   ```
   [~/sinatra/rockpaperscissors(spec)]$ cat config.ru
   #\ -s puma
   require './rps'
   run RockPaperScissors::App
   ```

1. This declares a single process type, `web`, and the command needed to run it.

2. The name `web` is important here. It declares that this process type will be attached to the HTTP routing stack of Heroku, and receive web traffic when deployed.

Foreman

1. It's important when developing and debugging an application that the local development environment is executed in the same manner as the remote environments.

2. This ensures that incompatibilities and hard to find bugs are caught before deploying to production and treats the application as a holistic unit instead of a series of individual commands working independently.

3. Foreman is a command-line tool for running Procfile-backed apps. It's installed automatically by the Heroku Toolbelt.

4. If you had a Procfile with both web and worker process types, Foreman will start one of each process type, with the output interleaved on your terminal

5. We can now start our application locally using Foreman (installed as part of the Toolbelt):

   ```
   $ foreman start
   16:39:04 web.1     | started with pid 30728
   18:49:43 web.1     | [2013-03-12 18:49:43] INFO  WEBrick 1.3.1
   18:49:43 web.1     | [2013-03-12 18:49:43] INFO  ruby 2.0.0p247 (2013-06-27 revision 4167
   18:49:43 web.1     | [2013-03-12 18:49:43] INFO  WEBrick::HTTPServer#start: pid=30728 por
   ```

6. Our app will come up on port 5000. Test that it's working with `curl` or a web browser, then `Ctrl-C` to exit.

Setting local environment variables

Config vars saved in the `.env` file of a project directory will be added to the environment when run by Foreman.

For example we can set the `RACK_ENV` to `development` in your environment.

```
$ echo "RACK_ENV=development" >>.env
$ foreman run irb
> puts ENV["RACK_ENV"]
> development
```

Do not commit the `.env` file to source control. It should only be used for local configuration.

Procfile y Despliegue

Véase la descripción de los contenidos del Procfile en 31.1.

1. A Procfile is not necessary to deploy apps written in most languages supported by Heroku.
2. The platform automatically detects the language, and creates a default web process type to boot the application server.
3. Creating an explicit `Procfile` is recommended for greater control and flexibility over your app.
4. For Heroku to use your Procfile, add the Procfile to the root of your application, then push to Heroku:

```
$ git add .
$ git commit -m "Procfile"
$ git push heroku
...
-----> Procfile declares process types: web, worker
       Compiled slug size is 10.4MB
-----> Launching... done
       http://strong-stone-297.herokuapp.com deployed to Heroku

To git@heroku.com:strong-stone-297.git
 * [new branch]      master -> master
```

Store your app in Git

```
$ git init
$ git add .
$ git commit -m "init"

[~/sinatra/rockpaperscissors(master)]$ git remote -v
origin  git@github.com:crguezl/sinatra-rock-paper-scissors.git (fetch)
origin  git@github.com:crguezl/sinatra-rock-paper-scissors.git (push)
```

Deploy your application to Heroku

Create the app on Heroku:

```
[~/sinatra/rockpaperscissors(master)]$ heroku create
Creating mysterious-falls-4594... done, stack is cedar
http://mysterious-falls-4594.herokuapp.com/ | git@heroku.com:mysterious-falls-4594.git
Git remote heroku added

[~/sinatra/rockpaperscissors(spec)]$ cat Rakefile
desc "start server using rackup ..."
task :default do
  sh "rackup"
end

require 'rspec/core/rake_task'

RSpec::Core::RakeTask.new do |task|
  task.rspec_opts = ["-c", "-f progress"]
  task.pattern    = 'spec/**/*_spec.rb'
end
```

```
[~/sinatra/rockpaperscissors(master)]$ git remote -v
heroku  git@heroku.com:mysterious-falls-4594.git (fetch)
heroku  git@heroku.com:mysterious-falls-4594.git (push)
origin  git@github.com:crguezl/sinatra-rock-paper-scissors.git (fetch)
origin  git@github.com:crguezl/sinatra-rock-paper-scissors.git (push)
```

Deploy your code:

```
[~/sinatra/rockpaperscissors(master)]$ git push heroku master
Counting objects: 31, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (29/29), done.
Writing objects: 100% (31/31), 9.09 KiB, done.
Total 31 (delta 11), reused 0 (delta 0)

-----> Ruby/Rack app detected
-----> Installing dependencies using Bundler version 1.3.2
       Running: bundle install --without development:test --path vendor/bundle --binstubs vend
       Fetching gem metadata from https://rubygems.org/..........
       Fetching gem metadata from https://rubygems.org/..
       Installing tilt (1.4.1)
       Installing haml (4.0.3)
       Installing rack (1.5.2)
       Installing puma (2.0.1)
       Installing rack-protection (1.5.0)
       Installing sinatra (1.4.2)
       Using bundler (1.3.2)
       Your bundle is complete! It was installed into ./vendor/bundle
       Post-install message from haml:
       HEADS UP! Haml 4.0 has many improvements, but also has changes that may break
       your application:
       * Support for Ruby 1.8.6 dropped
       * Support for Rails 2 dropped
       * Sass filter now always outputs <style> tags
       * Data attributes are now hyphenated, not underscored
       * html2haml utility moved to the html2haml gem
       * Textile and Maruku filters moved to the haml-contrib gem
       For more info see:
       http://rubydoc.info/github/haml/haml/file/CHANGELOG.md
       Cleaning up the bundler cache.
-----> Discovering process types
       Procfile declares types     -> (none)
       Default types for Ruby/Rack -> console, rake, web

-----> Compiled slug size: 1.3MB
-----> Launching... done, v4
       http://mysterious-falls-4594.herokuapp.com deployed to Heroku

To git@heroku.com:mysterious-falls-4594.git
 * [new branch]      master -> master
[~/sinatra/rockpaperscissors(master)]$
```

Visit your application

You've deployed your code to Heroku, and specified the process types in a Procfile.

You can now instruct Heroku to execute a process type.

Heroku does this by running the associated command in a dyno - a lightweight container which is the basic unit of composition on Heroku.

Let's ensure we have one dyno running the web process type:

```
$ heroku ps:scale web=1
```

Veamos que dice la ayuda:

```
$ heroku help ps
Usage: heroku ps

 list processes for an app

Additional commands, type "heroku help COMMAND" for more details:

  ps:restart [PROCESS]           #  ps:restart [PROCESS]
  ps:scale PROCESS1=AMOUNT1 ...  #  ps:scale PROCESS1=AMOUNT1 ...
  ps:stop PROCESS                #  ps:stop PROCESS

$ heroku help ps:scale
Usage: heroku ps:scale PROCESS1=AMOUNT1 ...

 scale processes by the given amount

 Example: heroku ps:scale web=3 worker+1
```

You can check the state of the app's dynos. The heroku `ps` command lists the running dynos of your application:

```
$ heroku ps
=== web: `bundle exec ruby web.rb -p $PORT`
web.1: up for 9m
```

Here, one dyno is running.

```
[~/sinatra/sinatra-rock-paper-scissors/sinatra-rockpaperscissors(master)]$ heroku ps
Process  State        Command
-------  -----------  ------------------------------------
web.1    idle for 8h  bundle exec rackup config.ru -p $P..
```

We can now visit the app in our browser with `heroku open`.

```
[~/sinatra/rockpaperscissors(master)]$ heroku open
Opening http://mysterious-falls-4594.herokuapp.com/
[~/sinatra/rockpaperscissors(master)]$
```

Dyno sleeping and scaling

1. Having only a single web dyno running will result in the dyno going to sleep after one hour of inactivity.

2. This causes a delay of a few seconds for the first request upon waking.

3. Subsequent requests will perform normally.

4. To avoid this, you can scale to more than one web dyno. For example:

```
$ heroku ps:scale web=2
```

5. For each application, Heroku provides 750 free dyno-hours.

6. Running your app at 2 dynos would exceed this free, monthly allowance, so let's scale back:

```
$ heroku ps:scale web=1
```

View the logs

Heroku treats logs as streams of time-ordered events aggregated from the output streams of all the dynos running the components of your application.

Heroku's Logplex provides a single channel for all of these events.

View information about your running app using one of the logging commands, heroku logs:

```
$ heroku logs
2013-03-13T04:10:49+00:00 heroku[web.1]: Starting process with command `bundle exec ruby web.r
2013-03-13T04:10:50+00:00 app[web.1]: [2013-03-13 04:10:50] INFO  WEBrick 1.3.1
2013-03-13T04:10:50+00:00 app[web.1]: [2013-03-13 04:10:50] INFO  ruby 2.0.0p247 (2013-06-27 r
2013-03-13T04:10:50+00:00 app[web.1]: [2013-03-13 04:10:50] INFO  WEBrick::HTTPServer#start: p
```

heroku run bash Heroku allows you to run commands in a *one-off dyno* - scripts and applications that only need to be executed when needed - using the `heroku run` command.

Since your app is - in general - spread across many dynos by the dyno manager, there is no single place to SSH into.

You deploy and manage apps, not servers.

You can invoke a shell as a one-off dyno.

While the *web dyno* would be defined in the `Procfile` and managed by the platform, the console and script would only be executed when needed. These are *one-off dynos*.

There are differences between one-off dynos (run with heroku run) and formation dynos

1. One-off dynos run attached to your terminal, with a character-by-character TCP connection for `STDIN` and `STDOUT`. This allows you to use interactive processes like a console.

2. Since `STDOUT` is going to your terminal, the only thing recorded in the app's logs is the startup and shutdown of the dyno.

3. One-off dynos terminate as soon as you press `Ctrl-C` or otherwise disconnect in your local terminal.

4. One-off dynos never automatically restart, whether the process ends on its own or whether you manually disconnect.

5. One-off dynos are named in the scheme `run.N` rather than the scheme `<process-type>.N`.

6. One-off dynos can never receive HTTP traffic, since the routers only routes traffic to dynos named `web.N`.

```
[~/srcPLgrado/pegjscalc(master)]$ heroku run bash
Running `bash` attached to terminal... up, run.2966
~ $ uname -a
Linux 8f9f0a0c-b10d-4cd5-9c1e-8e87067b6be2 3.8.11-ec2 #1 SMP Fri May 3 09:11:15 UTC 2013 x86_6
[~/srcPLgrado/pegjscalc(master)]$ heroku run bash
Running `bash` attached to terminal... up, run.2966
~ $ ls -l
total 48
drwx------ 2 u20508 20508 4096 2014-03-24 11:23 bin
```

```
-rw------- 1 u20508 20508   42 2014-03-24 11:23 config.ru
-rw------- 1 u20508 20508  258 2014-03-24 11:23 Gemfile
-rw------- 1 u20508 20508 2399 2014-03-24 11:23 Gemfile.lock
-rw------- 1 u20508 20508 1152 2014-03-24 11:23 main.rb
-rw------- 1 u20508 20508   43 2014-03-24 11:23 Procfile
drwx------ 2 u20508 20508 4096 2014-03-24 11:23 public
-rw------- 1 u20508 20508  492 2014-03-24 11:23 Rakefile
-rw------- 1 u20508 20508  421 2014-03-24 11:23 README.md
drwx------ 2 u20508 20508 4096 2014-03-24 11:23 tmp
drwx------ 5 u20508 20508 4096 2014-03-24 11:23 vendor
drwx------ 2 u20508 20508 4096 2014-03-24 11:23 views

~ $ ls -l tmp/
total 4
-rw------- 1 u20508 20508 242 2014-03-24 11:23 heroku-buildpack-release-step.yml
~ $ ls -l vendor
total 12
drwx------ 4 u20508 20508 4096 2014-03-20 23:33 bundle
drwx------ 2 u20508 20508 4096 2014-03-20 23:33 heroku
drwx------ 6 u20508 20508 4096 2014-03-24 11:23 ruby-2.0.0

~ $ ls -l bin
total 0
lrwxrwxrwx 1 u20508 20508 28 2014-03-24 15:05 erb -> ../vendor/ruby-2.0.0/bin/erb
lrwxrwxrwx 1 u20508 20508 28 2014-03-24 15:05 gem -> ../vendor/ruby-2.0.0/bin/gem
lrwxrwxrwx 1 u20508 20508 28 2014-03-24 15:05 irb -> ../vendor/ruby-2.0.0/bin/irb
lrwxrwxrwx 1 u20508 20508 29 2014-03-24 15:05 rake -> ../vendor/ruby-2.0.0/bin/rake
lrwxrwxrwx 1 u20508 20508 29 2014-03-24 15:05 rdoc -> ../vendor/ruby-2.0.0/bin/rdoc
lrwxrwxrwx 1 u20508 20508 27 2014-03-24 15:05 ri -> ../vendor/ruby-2.0.0/bin/ri
lrwxrwxrwx 1 u20508 20508 29 2014-03-24 15:05 ruby -> ../vendor/ruby-2.0.0/bin/ruby
lrwxrwxrwx 1 u20508 20508 33 2014-03-24 15:05 ruby.exe -> ../vendor/ruby-2.0.0/bin/ruby.exe
lrwxrwxrwx 1 u20508 20508 31 2014-03-24 15:05 testrb -> ../vendor/ruby-2.0.0/bin/testrb
```

- The filesystem is ephemeral, and the dyno itself will only live as long as your console session.
- When running multiple dynos, apps are distributed across several nodes by the dyno manager.
- Access to your app always goes through the routers. As a result, dynos don't have static IP addresses.
- While you can never connect to a dyno directly, it is possible to originate outgoing requests from a dyno. However, you can count on the dyno's IP address changing as it gets restarted in different places.

heroku run console

1. Heroku allows you to run commands in a one-off dyno - scripts and applications that only need to be executed when needed - using the `heroku run` command.
2. You can use this to launch an interactive Ruby shell (`bundle exec irb`) attached to your local terminal for experimenting in your app's environment:

```
$ heroku run console
Running ‘console‘ attached to terminal... up, ps.1
irb(main):001:0>
```

3. By default, irb has nothing loaded other than the Ruby standard library. From here you can require some of your application files. Or you can do it on the command line:

   ```
   $ heroku run console -r ./web
   ```

```
[~/srcPLgrado/pegjscalc(master)]$ heroku run irb
Running ‘irb‘ attached to terminal... up, run.1081
irb(main):001:0> ENV.keys
=> ["DATABASE_URL", "SHLVL", "PORT", "HOME", "HEROKU_POSTGRESQL_BROWN_URL", "PS1", "_", "COLUM
irb(main):002:0> ENV["DATABASE_URL"]
=> "postgres://moiwgreelvvujc:GL3shXGOpURyWOPrS2G8qaxzUe@ec2-23-21-101-129.compute-1.amazonaws
irb(main):003:0> ENV["HEROKU_POSTGRESQL_BROWN_URL"]
=> "postgres://moiwgreelvvujc:GL3shXGOpURyWOPrS2G8qaxzUe@ec2-23-21-101-129.compute-1.amazonaws
irb(main):004:0>
```

Podemos cargar librerías de nuestra aplicación (véase pegjscalc) y usarlas.

```
[~/srcPLgrado/pegjscalc(master)]$ heroku run console
Running ‘console‘ attached to terminal... up, run.9013
irb(main):002:0> require './main'
=> true
irb(main):003:0> p = PL0Program.all
=> [#<PL0Program @name="3p2m1" @source="                    3-2-1\r\n          ">, #<PL0Progra
irb(main):005:0> chuchu = PL0Program.first(:name => "apbtc")
=> #<PL0Program @name="apbtc" @source="a+b*c">
irb(main):006:0> chuchu.source
=> "a+b*c"
irb(main):007:0> prog = PL0Program.create(:name => "tata", :source => "3*a-c")
=> #<PL0Program @name="tata" @source="3*a-c">
irb(main):008:0>
```

Rake

Rake can be run in an attached dyno exactly like the console:

```
[~/srcPLgrado/pegjscalc(master)]$ heroku run rake -T
Running ‘rake -T‘ attached to terminal... up, run.2124
rake clean  # Remove pl0.pegjs
rake sass   # Compile public/styles.scss into public/styles.css using sass
rake test   # tests
rake web    # Compile pl0.pegjs browser version
[~/srcPLgrado/pegjscalc(master)]$ heroku run rake test
Running ‘rake test‘ attached to terminal... up, run.2082
Not implemented (yet)
```

Using a SQL database

By default, non-Rails apps aren't given a SQL database.

This is because you might want to use a NoSQL database like Redis or CouchDB, or you don't need any database at all.

If you need a SQL database for your app, do this:

1. `$ heroku addons:add heroku-postgresql:dev`

2. You must also add the Postgres gem to your app in order to use your database. Add a line to your Gemfile like this:

   ```
   gem 'pg'
   ```

3. You'll also want to setup a local PostgreSQL database.

Webserver

By default your app (Rack) will use `Webrick`.
This is fine for testing, but for production apps you'll want to switch to a more robust webserver.
On Cedar, they recommend `Unicorn` as the webserver.

31.2. Logging

Heroku aggregates three categories of logs for your app:

1. App logs - Output from your application.

   This will include logs generated from

   a) within your application,
   b) application server and
   c) libraries.

   (Filter: `--source app`)

2. System logs -

   Messages about actions taken by the Heroku platform infrastructure on behalf of your app, such as:

   a) restarting a crashed process,
   b) sleeping or waking a web dyno, or
   c) serving an error page due to a problem in your app.

   (Filter: `--source heroku`)

3. API logs -

   Messages about administrative actions taken by you and other developers working on your app, such as:

   a) deploying new code,
   b) scaling the process formation, or
   c) toggling maintenance mode.

   (Filter: `--source heroku --ps api`)

```
[~/rack/rack-rock-paper-scissors(master)]$ heroku logs --source heroku --ps api
2013-10-23T21:33:41.105090+00:00 heroku[api]: Deploy 5ec1351 by chuchu.chachi.leon@gmail.c
2013-10-23T21:33:41.154690+00:00 heroku[api]: Release v7 created by chuchu.chachi.leon@gma
```

Logplex is designed for collating and routing log messages, not for storage. It keeps the last 1,500 lines of consolidated logs.

Heroku recommends using a separate service for long-term log storage; see Syslog drains for more information.

Writing to your log

Anything written to standard out (stdout) or standard error (stderr) is captured into your logs. This means that you can log from anywhere in your application code with a simple output statement:

```
puts "Hello, logs!"
```

To take advantage of the realtime logging, you may need to disable any log buffering your application may be carrying out. For example, in Ruby add this to your config.ru:

```
$stdout.sync = true
```

Some frameworks send log output somewhere other than stdout by default.

To fetch your logs

```
$ heroku logs
2010-09-16T15:13:46.677020+00:00 app[web.1]: Processing PostController#list (for 208.39.138.12
2010-09-16T15:13:46.677023+00:00 app[web.1]: Rendering template within layouts/application
2010-09-16T15:13:46.677902+00:00 app[web.1]: Rendering post/list
2010-09-16T15:13:46.678990+00:00 app[web.1]: Rendered includes/_header (0.1ms)
2010-09-16T15:13:46.698234+00:00 app[web.1]: Completed in 74ms (View: 31, DB: 40) | 200 OK [ht
2010-09-16T15:13:46.723498+00:00 heroku[router]: at=info method=GET path=/posts host=myapp.her
2010-09-16T15:13:47.893472+00:00 app[worker.1]: 2 jobs processed at 16.6761 j/s, 0 failed ...
```

In this example, the output includes log lines from one of the app's web dynos, the Heroku HTTP router, and one of the app's workers.

The logs command retrieves 100 log lines by default.

Log message ordering

When retrieving logs, you may notice that the logs are not always in order, especially when multiple components are involved.

This is likely an artifact of distributed computing.

Logs originate from many sources (router nodes, dynos, etc) and are assembled into a single log stream by logplex.

It is up to the logplex user to sort the logs and provide the ordering required by their application, if any

Log history limits

You can fetch up to 1500 lines using the –num (or -n) option:

```
$ heroku logs -n 200
```

Heroku only stores the last 1500 lines of log history. If you'd like to persist more than 1500 lines, use a logging add-on or create your own syslog drain[1].

Log format

Each line is formatted as follows:

1. timestamp source[dyno]: message
2. Timestamp - The date and time recorded at the time the log line was produced by the dyno or component. The timestamp is in the format specified by RFC5424, and includes microsecond precision.
3. Source -
   a) All of your app's dynos (web dynos, background workers, cron) have a source of app.
   b) All of Heroku's system components (HTTP router, dyno manager) have a source of heroku.
4. Dyno - The name of the dyno or component that wrote this log line. For example, `worker #3` appears as `worker.3`, and the Heroku HTTP router appears as router.
5. Message - The content of the log line. Dynos can generate messages up to approximately 1024 bytes in length and longer messages will be truncated.

[1]Logplex drains allow you to forward your Heroku logs to an external syslog server for long-term archiving. You must configure the service or your server to be able to receive syslog packets from Heroku, and then add its syslog URL (which contains the host and port) as a syslog drain.

Realtime tail

1. Similar to `tail -f`, realtime tail displays recent logs and leaves the session open for realtime logs to stream in.

2. By viewing a live stream of logs from your app, you can gain insight into the behavior of your live application and debug current problems.

3. You may tail your logs using `--tail` (or `-t`).

   ```
   $ heroku logs --tail
   ```

   When you are done, press Ctrl-C to close the session.

Filtering
If you only want to fetch logs with a certain source, a certain dyno, or both, you can use the `--source` (or `-s`) and `--ps` (or `-p`) filtering arguments:

```
$ heroku logs --ps router
2012-02-07T09:43:06.123456+00:00 heroku[router]: at=info method=GET path=/stylesheets/dev-cent
2012-02-07T09:43:06.123456+00:00 heroku[router]: at=info method=GET path=/articles/bundler hos

$ heroku logs --source app
2012-02-07T09:45:47.123456+00:00 app[web.1]: Rendered shared/_search.html.erb (1.0ms)
2012-02-07T09:45:47.123456+00:00 app[web.1]: Completed 200 OK in 83ms (Views: 48.7ms | ActiveR
2012-02-07T09:45:47.123456+00:00 app[worker.1]: [Worker(host:465cf64e-61c8-46d3-b480-362bfd4ec
2012-02-07T09:46:01.123456+00:00 app[web.6]: Started GET "/articles/buildpacks" for 4.1.81.209

$ heroku logs --source app --ps worker
2012-02-07T09:47:59.123456+00:00 app[worker.1]: [Worker(host:260cf64e-61c8-46d3-b480-362bfd4ec
2012-02-07T09:47:59.123456+00:00 app[worker.1]: [Worker(host:260cf64e-61c8-46d3-b480-362bfd4ec
```

When filtering by dyno, either the base name, `--ps web`, or the full name, `--ps web.1`, may be used.

You can also combine the filtering switches with `--tail` to get a realtime stream of filtered output.

```
$ heroku logs --source app --tail
```

31.3. Heroku Postgress

Véase Heroku Postgress.

Heroku Postgres is the SQL database service run by Heroku that is provisioned and managed as an add-on.

Heroku Postgres is accessible from any language with a PostgreSQL driver including all languages and frameworks supported by Heroku: Java, Ruby, Python, Scala, Play, Node.js and Clojure.

```
[~/srcPLgrado/pegjscalc(master)]$ heroku addons
=== pegjspl0 Configured Add-ons
heroku-postgresql:hobby-dev  HEROKU_POSTGRESQL_BROWN
```

In addition to a variety of management commands available via the Heroku CLI, Heroku Postgres features a web dashboard, the ability to create dataclips and several additional services on top of a fully managed database service.

Provisioning the add-on Many buildpacks (what compiles your application into a runnable entity on Heroku) automatically provision a Heroku Postgres instance for you.

Your language's buildpack documentation will specify if any add-ons are automatically provisioned.

Additionally, you can use `heroku addons` to see if your application already has a database provisioned and what plan it is[2].

```
[~/srcPLgrado/pegjscalc(master)]$ heroku addons
=== pegjspl0 Configured Add-ons
heroku-postgresql:hobby-dev  HEROKU_POSTGRESQL_BROWN
```

If your application doesn't yet have a database provisioned, or you wish to upgrade your existing database or create a master/slave setup, you can create a new database using the CLI.

Create new db Heroku Postgres can be attached to a Heroku application via the CLI[3]:

```
$ heroku addons:add heroku-postgresql:dev
Adding heroku-postgresql:dev to sushi... done, v69 (free)
Attached as HEROKU_POSTGRESQL_RED
Database has been created and is available
```

Once Heroku Postgres has been added a `HEROKU_POSTGRESQL_COLOR_URL` setting will be available in the app configuration and will contain the URL used to access the newly provisioned Heroku Postgres service.

This can be confirmed using the `heroku config` command.

```
[~/srcPLgrado/pegjscalc(master)]$ heroku config
=== pegjspl0 Config Vars
DATABASE_URL:                postgres://moiwgreelvvujc:GL3shXGOpURyWOPrS2G8qaxzUe@ec2-23-21-10
HEROKU_POSTGRESQL_BROWN_URL: postgres://moiwgreelvvujc:GL3shXGOpURyWOPrS2G8qaxzUe@ec2-23-21-10
LANG:                        en_US.UTF-8
PGBACKUPS_URL:               https://453643:cqz59jrxbbfcxj0fanhjfg0vz@pgbackups.herokuapp.com/
RACK_ENV:                    production
```

Establish primary DB

Heroku recommends using the `DATABASE_URL` config var to store the location of your primary database.

```
[~/srcPLgrado/pegjscalc(master)]$ head main.rb
require 'sinatra'
require "sinatra/reloader" if development?
require 'sinatra/flash'
require 'data_mapper'
require 'pp'

# full path!
DataMapper.setup(:default,
                 ENV['DATABASE_URL'] || "sqlite3://#{Dir.pwd}/database.db" )
```

In single-database setups your new database will have already been assigned a `HEROKU_POSTGRESQL_COLOR_URL` config with the accompanying `DATABASE_URL`.

You may verify this via heroku config and verifying the value of both `HEROKU_POSTGRESQL_COLOR_URL` and `DATABASE_URL` which should match.

[2] In order for Heroku to manage this add-on for you and respond to a variety of operational situations, the value of this config var may change at any time. Relying on it outside your Heroku app may prove problematic as you will have to re-copy the value on change.

[3] Heroku Postgres has a variety of plans spread across two general tiers of service – starter and production. Please understand the different levels of service provided by database tiers when provisioning the service. You can always upgrade databases should you outgrow your initial plan.

pg:info

To see all PostgreSQL databases provisioned by your application and the identifying characteristics of each (db size, status, number of tables, PG version, creation date etc. . .) use the `heroku pg:info` command.

```
[~/srcPLgrado/pegjscalc(master)]$ heroku pg:info
=== HEROKU_POSTGRESQL_BROWN_URL (DATABASE_URL)
Plan:        Hobby-dev
Status:      available
Connections: 0
PG Version:  9.3.3
Created:     2014-03-20 23:33 UTC
Data Size:   6.5 MB
Tables:      1
Rows:        4/10000 (In compliance)
Fork/Follow: Unsupported
Rollback:    Unsupported
```

To continuously monitor the status of your database, pass pg:info through the unix watch command:

```
[~/srcPLgrado/pegjscalc(master)]$ watch heroku pg:info
-bash: watch: no se encontró la orden
[~/srcPLgrado/pegjscalc(master)]$ brew install watch
[~/srcPLgrado/pegjscalc(master)]$ watch heroku pg:info
...
```

pg:psql `psql` is the native PostgreSQL interactive terminal and is used to execute queries and issue commands to the connected database.

To establish a `psql` session with your remote database use `heroku pg:psql`. You must have PostgreSQL installed on your system to use heroku `pg:psql`.

```
[~/srcPLgrado/pegjscalc(master)]$ heroku pg:psql
---> Connecting to HEROKU_POSTGRESQL_BROWN_URL (DATABASE_URL)
psql (9.2.6, server 9.3.3)
WARNING: psql version 9.2, server version 9.3.
         Some psql features might not work.
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

pegjspl0::BROWN=> \dt
               List of relations
 Schema |     Name     | Type  |     Owner
--------+--------------+-------+----------------
 public | pl0_programs | table | moiwgreelvvujc
(1 row)

pegjspl0::BROWN=>
pegjspl0::BROWN=> SELECT * FROM pl0_programs;
  name  |           source
--------+-----------------------------
 3m2m1  |                      3-2-1\r+
        |
 ap1tb  | a+1*b\r                     +
```

```
       |
 test  |                    a+1*b\r+
       |          \r              +
       |
 lolwut |                   3-2-1\r+
       |
(4 rows)
```

If you have more than one database, specify the database to connect to as the first argument to the command (the database located at `DATABASE_URL` is used by default).

```
$ heroku pg:psql HEROKU_POSTGRESQL_GRAY
Connecting to HEROKU_POSTGRESQL_GRAY... done
...
```

pg:reset To drop and recreate your database use `pg:reset`:

```
[~/srcPLgrado/pegjscalc(master)]$ heroku pg:reset DATABASE

 !    WARNING: Destructive Action
 !    This command will affect the app: pegjspl0
 !    To proceed, type "pegjspl0" or re-run this command with --confirm pegjspl0

> pegjspl0
Resetting HEROKU_POSTGRESQL_BROWN_URL (DATABASE_URL)... done
```

Es necesario a continuación rearrancar el servidor:

```
[~/srcPLgrado/pegjscalc(master)]$ heroku ps:restart
Restarting dynos... done
```

pg:pull `pg:pull` can be used to pull remote data from a Heroku Postgres database to a database on your local machine. The command looks like this:

```
[~/srcPLgrado/pegjscalc(master)]$ pg_ctl -D /usr/local/var/postgres -l /usr/local/var/postgres
server starting

$ heroku pg:pull HEROKU_POSTGRESQL_MAGENTA mylocaldb --app sushi
```

This command will create a new local database named `mylocaldb` and then pull data from database at `DATABASE_URL` from the app `sushi`.

In order to prevent accidental data overwrites and loss, the local database must not exist. You will be prompted to drop an already existing local database before proceeding.

pg:push Like pull but in reverse, `pg:push` will push data from a local database into a remote Heroku Postgres database. The command looks like this:

```
$ heroku pg:push mylocaldb HEROKU_POSTGRESQL_MAGENTA --app sushi
```

This command will take the local database `mylocaldb` and push it to the database at `DATABASE_URL` on the app `sushi`. In order to prevent accidental data overwrites and loss, the remote database must be empty. You will be prompted to `pg:reset` an already a remote database that is not empty.

31.4. Troubleshooting

31.4.1. Crashing

If you push your app and it crashes, `heroku ps` shows state crashed:

```
=== web (1X): 'bundle exec thin start -R config.ru -e $RACK_ENV -p $PORT'
web.1: crashed 2013/10/24 20:21:34 (~ 1h ago)
```

check your logs to find out what went wrong.

Here are some common problems.

Failed to require a sourcefile

If your app failed to require a sourcefile, chances are good you're running Ruby 1.9.1 or 1.8 in your local environment.

The load paths have changed in Ruby 1.9 which applies to Ruby 2.0.

Port your app forward to Ruby 2.0.0 making certain it works locally before trying to push to Cedar again.

Encoding error Ruby 1.9 added more sophisticated encoding support to the language which applies to Ruby 2.0.

Not all gems work with Ruby 2.0. If you hit an encoding error, you probably haven't fully tested your app with Ruby 2.0.0 in your local environment.

Port your app forward to Ruby 2.0.0 making certain it works locally before trying to push to Cedar again.

Missing a gem

If your app crashes due to missing a gem, you may have it installed locally but not specified in your Gemfile.

You must isolate all local testing using `bundle exec`.

For example, don't run `ruby web.rb`, run

```
bundle exec ruby web.rb
```

Don't run `rake db:migrate`, run

```
bundle exec rake db:migrate.
```

Another approach is to create a blank RVM gemset to be absolutely sure you're not touching any system-installed gems:

```
$ rvm gemset create myapp
$ rvm gemset use myapp
```

Runtime dependencies on development/test gems

If you're still missing a gem when you deploy, check your Bundler groups.

Heroku builds your app without the `development` or `test` groups, and if you app depends on a gem from one of these groups to run, you should move it out of the group.

One common example using the `RSpec tasks` in your Rakefile. If you see this in your Heroku deploy:

```
$ heroku run rake -T
Running 'rake -T' attached to terminal... up, ps.3
rake aborted!
no such file to load -- rspec/core/rake_task
```

Then you've hit this problem.

First, duplicate the problem locally like so:

```
$ bundle install --without development:test
...
$ bundle exec rake -T
rake aborted!
no such file to load -- rspec/core/rake_task
```

Now you can fix it by making these Rake tasks conditional on the gem load. For example:

```
begin
  require "rspec/core/rake_task"

  desc "Run all examples"
  RSpec::Core::RakeTask.new(:spec) do |t|
    t.rspec_opts = %w[--color]
    t.pattern = 'spec/*_spec.rb'
  end
rescue LoadError
end
```

Confirm it works locally, then push to Heroku.

Versiones soportadas por Heroku

Véase Heroku Ruby Support

Rack::Sendfile

Heroku does not support the use of Rack::Sendfile.

Rack:Sendfile usually requires that there is a frontend webserver like nginx or apache is running on the same machine as the application server.

This is not how Heroku is architected. Using the Rack::Sendfile middleware will cause your file downloads to fail since it will send a body with Content-Length of 0.

31.4.2. heroku run: Timeout awaiting process

The heroku run command opens a connection to Heroku on port 5000. If your local network or ISP is blocking port 5000 (el caso de la ULL), or you are experiencing a connectivity issue, you will see an error similar to:

```
[~/srcPLgrado/pegjscalc(master)]$ heroku run console
Running `console` attached to terminal... up, run.4357
 !
 !    Timeout awaiting process
```

You can test your connection to Heroku by trying to connect directly to port 5000 by using `telnet` to `rendezvous.runtime.heroku.com`.

Desde la universidad fracasa:

```
[~/srcPLgrado/pegjscalc(master)]$ telnet rendezvous.runtime.heroku.com 5000Trying 50.19.103.36
telnet: connect to address 50.19.103.36: Operation timed out
telnet: Unable to connect to remote host
```

A successful session will look like this:

```
$ telnet rendezvous.runtime.heroku.com 5000
Trying 50.19.103.36...
Connected to ec2-50-19-103-36.compute-1.amazonaws.com.
Escape character is '^]'.
```

If you do not get this output, your computer is being blocked from accessing our services. We recommend contacting your IT department, ISP, or firewall manufacturer to move forward with this issue.

31.5. Configuration

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku help config
Usage: heroku config

 display the config vars for an app

 -s, --shell  # output config vars in shell format

Examples:

 $ heroku config
 A: one
 B: two

 $ heroku config --shell
 A=one
 B=two

Additional commands, type "heroku help COMMAND" for more details:

  config:get KEY                            #  display a config value for an app
  config:set KEY1=VALUE1 [KEY2=VALUE2 ...]  #  set one or more config vars
  config:unset KEY1 [KEY2 ...]              #  unset one or more config vars

[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku config -s
DATABASE_URL=postgres://bhhatrhjjhwcvt:hjgjfhgjfhjfuWH7ls_PJKK5QD@ec2-54-204-35-132.compute-1.
HEROKU_POSTGRESQL_BLACK_URL=postgres://bhjshfdhakwcvt:hQssnhq1y1jhgfhgls_PGNu5QD@ec2-54-204-35

[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku config:set C=4
Setting config vars and restarting crguezl-songs... done, v6
C: 4
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku config:get C
4
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku config:unset C
Unsetting C and restarting crguezl-songs... done, v7
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku config:get C

[~/sinatra/sinatra-datamapper-jump-start(master)]$]]
```

31.6. Make Heroku run non-master Git branch

Make Heroku run non-master Git branch You can push an alternative branch to Heroku using Git.

```
git push heroku-dev test:master
```

This pushes your local test branch to the remote's master branch (on Heroku).

El manual de `git push` dice:

To push a local branch to an established remote, you need to issue the command:

```
git push  <REMOTENAME> <BRANCHNAME>
```

This is most typically invoked as `git push origin master`.

If you would like to give the branch a different name on the upstream side of the push, you can issue the command:

```
git push  <REMOTENAME> <LOCALBRANCHNAME>:<REMOTEBRANCHNAME>
```

31.7. Account Verification and add-ons

You must verify your account by adding a credit card before you can add any add-on to your app other than `heroku-postgresql:dev` and `pgbackups:plus`.

Adding a credit card to your account lets you

1. use the free add-ons,

2. allows your account to have more than 5 apps at a time (verified accounts may have up to 100 apps),

3. and gives you access to turn on paid services any time with a few easy clicks.

4. The easiest way to do this is to go to your account page and click `Add Credit Card`.

5. Alternatively, when you attempt to perform an action that requires a credit card, either from the Heroku CLI or through the web interface, you will be prompted to visit the credit card page.

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku addons:add rediscloud:20
Adding rediscloud:20 on dgjgxcl-songs... failed
 !    Please verify your account to install this add-on
 !    For more information, see http://devcenter.heroku.com/categories/billing
 !    Verify now at https://heroku.com/verify
```

31.8. Véase

- Heroku: Getting Started with Ruby on Heroku
- SitePoint: Get Started with Sinatra on Heroku by Jagadish Thaker. Published August 12, 2013
- Deploying Rack-based Apps
- Heroku: List of Published Articles for Ruby
- Foreman
  1. Introducing Foreman by David Dollar
  2. Foreman man pages
  3. Applying the Unix Process Model to Web Apps by Adam Wiggins
- Ruby Kickstart - Session 6 de Joshua Cheek (Vimeo)
- sinatra-rock-paper-scissors
- The Procfile is your friend 13 January, 2012. Neil Middleton

Capítulo 32

DataMapper

32.1. Introducción a Los Object Relational Mappers (ORM)

What is a Object Relational Mapper?

A simple answer is that you wrap your tables or stored procedures in classes in your programming language, so that instead of writing SQL statements to interact with your database, you use methods and properties of objects.

In other words, instead of something like this:

```
String sql = "SELECT ... FROM persons WHERE id = 10"
DbCommand cmd = new DbCommand(connection, sql);
Result res = cmd.Execute();
String name = res[0]["FIRST_NAME"];
```

you do something like this:

```
Person p = Person.Get(10);
```

or similar code (lots of variations here). The framework is what makes this code possible.

Now, benefits:

1. First of all, you hide the SQL away from your logic code
2. This has the benefit of allowing you to more easily support more database engines
3. For instance, MS SQL Server and Oracle have different names on typical functions, and different ways to do calculations with dates. This difference can be put away from your logic code.
4. Additionally, you can focus on writing the logic, instead of getting all the SQL right.
5. The code will typically be more readable as well, since it doesn't contain all the plumbing necessary to talk to the database.

32.2. Patterns Active Record y DataMapper

Active Record In software engineering, the active record pattern is an architectural pattern found in software that stores its data in relational databases. It was named by Martin Fowler in his 2003 book *Patterns of Enterprise Application Architecture*.

The interface of an object conforming to this pattern would include functions such as

- Insert,
- Update, and

- Delete,

plus properties that correspond more or less directly to the columns in the underlying database table.

Active record is an approach to accessing data in a database.

- A database table or view is wrapped into a class.
- Thus, an object instance is tied to a single row in the table.
- After creation of an object, a new row is added to the table upon save.
- Any object loaded gets its information from the database.
- When an object is updated the corresponding row in the table is also updated.
- The wrapper class implements accessor methods or properties for each column in the table or view.
- This pattern is commonly used by object persistence tools, and in object-relational mapping (ORM).
- Typically, foreign key relationships will be exposed as an object instance of the appropriate type via a property.

Las gemas activerecord y `DataMapper` siguen el patrón *Active Record*.

- Proyecto sinatra-datamapper-sample en GitHub
- Documentación de DataMapper
- Sinatra Recipes: DataMapper
- Sinatra Book: DataMapper

DataMapper Martin Fowler (Catalog of Patterns of Enterprise Application Architecture):

1. Objects and relational databases have different mechanisms for structuring data.
2. Many parts of an object, such as collections and inheritance, aren't present in relational databases.
3. When you build an object model with a lot of business logic it's valuable to use these mechanisms to better organize the data and the behavior that goes with it.
4. Doing so leads to variant schemas; that is, the object schema and the relational schema don't match up.
5. You still need to transfer data between the two schemas, and this data transfer becomes a complexity in its own right.
6. If the in-memory objects know about the relational database structure, changes in one tend to ripple to the other.
7. The *Data Mapper* is a layer of software that separates the in-memory objects from the database.
8. *Its responsibility is to transfer data between the two and also to isolate them from each other*
9. With Data Mapper
   a) the in-memory objects needn't know even that there's a database present;
   b) they need no SQL interface code,
   c) and certainly no knowledge of the database schema.

10. (The database schema is always ignorant of the objects that use it.)

The gem perpetuity implements the DataMapper pattern.

- DataMapper en la Wikipedia
- Martin Fowler: DataMapper
- https://github.com/crguezl/perpetuity-example

32.3. Ejemplo de Uso de DataMapper

Donde

- ```
  [~/sinatra/sinatra-datamapper-jump-start(master)]$ pwd -P
  /Users/casiano/local/src/ruby/sinatra/sinatra-datamapper-jump-start
  ```
- ```
  [~/sinatra/sinatra-datamapper-jump-start(master)]$ git remote -v
  origin  git@github.com:crguezl/sinatra-datamapper-jump-start.git (fetch)
  origin  git@github.com:crguezl/sinatra-datamapper-jump-start.git (push)
  ```
- Este ejemplo en GitHub
- http://sinadm.herokuapp.com/ (Puede que este caída)

Enlaces

1. Documentación del módulo DataMapper en RubyDoc
2. https://github.com/crguezl/datamapper_example
3. https://github.com/crguezl/datamapper-intro

La Clase Song

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ cat song.rb
require 'dm-core'
require 'dm-migrations'

class Song
  include DataMapper::Resource
  property :id, Serial
  property :title, String
  property :lyrics, Text
  property :length, Integer

end
```

The `Song` model is going to need to be persistent, so we'll include `DataMapper::Resource`.

The convention with model names is to use the singular, not plural version... but that's just the convention, we can do whatever we want.

```
configure do
  enable :sessions
  set :username, 'frank'
  set :password, 'sinatra'
end
```

DataMapper.finalize

`DataMapper.finalize`

This method performs the necessary steps to finalize `DataMapper` for the current repository. It should be called after loading all models and plugins. It ensures foreign key properties and anonymous join models are created. These are otherwise lazily declared, which can lead to unexpected errors. It also performs basic validity checking of the `DataMapper` models.

Mas código de Song.rb

```
get '/songs' do
  @songs = Song.all
  slim :songs
end

get '/songs/new' do
  halt(401,'Not Authorized') unless session[:admin]
  @song = Song.new
  slim :new_song
end

get '/songs/:id' do
  @song = Song.get(params[:id])
  slim :show_song
end

get '/songs/:id/edit' do
  @song = Song.get(params[:id])
  slim :edit_song
end
```

Song.create

If you want to create a new resource with some given attributes and then save it all in one go, you can use the `#create` method:

```
post '/songs' do
  song = Song.create(params[:song])
  redirect to("/songs/#{song.id}")
end

put '/songs/:id' do
  song = Song.get(params[:id])
  song.update(params[:song])
  redirect to("/songs/#{song.id}")
end

delete '/songs/:id' do
  Song.get(params[:id]).destroy
  redirect to('/songs')
end
```

Una sesión con pry probando DataMapper

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ pry
[1] pry(main)> require 'sinatra'
=> true
[2] pry(main)> require './song'
=> true
```

DataMapper.setup

We must specify our database connection.

We need to make sure to do this before you use our models, i.e. before we actually start accessing the database.

```
  # If you want the logs displayed you have to do this before the call to setup
  DataMapper::Logger.new($stdout, :debug)

  # An in-memory Sqlite3 connection:
  DataMapper.setup(:default, 'sqlite::memory:')

  # A Sqlite3 connection to a persistent database
  DataMapper.setup(:default, 'sqlite:///path/to/project.db')

  # A MySQL connection:
  DataMapper.setup(:default, 'mysql://user:password@hostname/database')

  # A Postgres connection:
  DataMapper.setup(:default, 'postgres://user:password@hostname/database')
Note: that currently you must setup a :default repository to work with DataMapper (and to be a
```

In our case:

```
[4] pry(main)> pry(main)> DataMapper.setup(:default,'sqlite:development.db')
```

Multiple Data-Store Connections

DataMapper sports a concept called a *context* which encapsulates the *data-store context* in which you want operations to occur. For example, when you setup a connection you are defining a context known as `:default`

```
   DataMapper.setup(:default, 'mysql://localhost/dm_core_test')
```

If you supply another context name, you will now have 2 database contexts with their own unique loggers, connection pool, identity map....one *default context* and one *named context.*

```
 DataMapper.setup(:external, 'mysql://someother_host/dm_core_test')
```

To use one context rather than another, simply wrap your code block inside a `repository` call. It will return whatever your block of code returns.

```
 DataMapper.repository(:external) { Person.first }
 # hits up your :external database and retrieves the first Person
```

This will use your connection to the `:external` data-store and the first `Person` it finds. Later, when you call `.save` on that person, it'll get saved back to the `:external` data-store; An object is aware of what context it came from and should be saved back to.

El Objeto DataMapper::Adapters

```
=> #<DataMapper::Adapters::SqliteAdapter:0x007fad2c0f6a50
 @field_naming_convention=DataMapper::NamingConventions::Field::Underscored,
 @name=:default,
 @normalized_uri=
  #<DataObjects::URI:0x007fad2c0f62a8
   @fragment="{Dir.pwd}/development.db",
   @host="",
   @password=nil,
   @path=nil,
   @port=nil,
   @query=
    {"scheme"=>"sqlite3",
     "user"=>nil,
     "password"=>nil,
     "host"=>"",
     "port"=>nil,
     "query"=>nil,
     "fragment"=>"{Dir.pwd}/development.db",
     "adapter"=>"sqlite3",
     "path"=>nil},
   @relative=nil,
   @scheme="sqlite3",
   @subscheme=nil,
   @user=nil>,
 @options=
  {"scheme"=>"sqlite3",
   "user"=>nil,
   "password"=>nil,
   "host"=>"",
   "port"=>nil,
   "query"=>nil,
   "fragment"=>"{Dir.pwd}/development.db",
   "adapter"=>"sqlite3",
   "path"=>nil},
 @resource_naming_convention=
  DataMapper::NamingConventions::Resource::UnderscoredAndPluralized>
```

Creando las tablas con DataMapper.auto_migrate! We can create the table by issuing the following command:

```
[4] pry(main)> DataMapper.auto_migrate!
```

1. This will issue the necessary `CREATE` statements (`DROP`ing the table first, if it exists) to define each storage according to their properties.
2. After `auto_migrate!` has been run, the database should be in a pristine state.
3. All the tables will be empty and match the model definitions.

DataMapper.auto_upgrade! This wipes out existing data, so you could also do:

```
DataMapper.auto_upgrade!
```

1. This tries to make the schema match the model.

2. It will CREATE new tables, and add columns to existing tables.
3. It won't change any existing columns though (say, to add a NOT NULL constraint) and it doesn't drop any columns.
4. Both these commands also can be used on an individual model (e.g. `Song.auto_migrate!`)

Métodos de la Clase Mapeada

```
[5] pry(main)> song = Song.new
=> #<Song @id=nil @title=nil @lyrics=nil @length=nil @released_on=nil>
[6] pry(main)> song.save
=> true
[7] pry(main)> song
=> #<Song @id=1 @title=<not loaded> @lyrics=<not loaded> @length=<not loaded> @released_on=<no
[8] pry(main)> song.title = "My Way"
=> "My Way"
[9] pry(main)> song.lyrics
=> nil
[10] pry(main)> song.lyrics = "And now, the end is near ..."
=> "And now, the end is near ..."
[11] pry(main)> song.length = 435
=> 435
[42] pry(main)> song.save
=> true
[43] pry(main)> song
=> #<Song @id=1 @title="My Way" @lyrics="And now, the end is near ..." @length=435 @released_o
```

El método create If you want to create a new resource with some given attributes and then save it all in one go, you can use the `#create` method.

```
[28] pry(main)> Song.create(title: "Come fly with me", lyrics: "Come fly with me, let's fly, l
=> #<Song @id=2 @title="Come fly with me" @lyrics="Come fly with me, let's fly, let's fly away
```

1. If the creation was successful, `#create` will return the newly created `DataMapper::Resource`
2. If it failed, it will return a new resource that is initialized with the given attributes and possible default values declared for that resource, but that's not yet saved
3. To find out wether the creation was successful or not, you can call `#saved`? on the returned resource
4. It will return `true` if the resource was successfully persisted, or `false` otherwise

first_or_create If you want to either find the first resource matching some given criteria or just create that resource if it can't be found, you can use `#first_or_create`.

```
s = Song.first_or_create(:title => 'New York, New York')
```

This will first try to find a `Song` instance with the given `title`, and if it fails to do so, it will return a newly created `Song` with that `title`.

If the criteria you want to use to query for the resource differ from the attributes you need for creating a new resource, you can pass the attributes for creating a new resource as the second parameter to `#first_or_create`, also in the form of a `#Hash`.

```
s = Song.first_or_create({ :title => 'My Way' }, { :lyrics => '... the end is not near' })
```

This will search for a Song named 'My Way' and if it can't find one, it will return a new Song instance with its name set to 'My Way' and the lyrics set to `.. the end is not near`

1. You can see that for creating a new resource, both hash arguments will be merged so you don't need to specify the query criteria again in the second argument Hash that lists the attributes for creating a new resource

2. However, if you really need to create the new resource with different values from those used to query for it, the second Hash argument will overwrite the first one.

```
s = Song.first_or_create({ :title => 'My Way' }, {
  :title  => 'My Way Home',
  :lyrics => '... the end is not near'
})
```

This will search for a Song named `'My Way'` but if it fails to find one, it will return a Song instance with its title set to `'My Way Home'` and its lyrics set to `'... the end is not near'`.

Comprobando con sqlite3
Podemos abrir la base de datos con el gestor de base de datos y comprobar que las tablas y los datos están allí:

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ sqlite3 development.db
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema
CREATE TABLE "songs" ("id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
      "title" VARCHAR(50), "lyrics" TEXT, "length"
      INTEGER, "released_on" TIMESTAMP);
sqlite> select * from songs;
1|My Way|And now, the end is near ...|435|
2|Come fly with me|Come fly with me, let's fly, let's fly away ...|199|
sqlite>
```

Búsquedas y Consultas DataMapper has methods which allow you to grab a single record by key, the first match to a set of conditions, or a collection of records matching conditions.

```
song  = Song.get(1)                     # get the song with primary key of 1.
song  = Song.get!(1)                    # Or get! if you want an ObjectNotFoundError on failur
song  = Song.first(:title => 'Girl')    # first matching record with the title 'Girl'
song  = Song.last(:title => 'Girl')     # last matching record with the title 'Girl'
songs = Song.all                        # all songs
```

```
[29] pry(main)> Song.count
=> 2
[30] pry(main)> Song.all
=> [#<Song @id=1 @title=nil @lyrics=<not loaded> @length=nil @released_on=nil>, #<Song @id=2 @
[31] pry(main)> Song.get(1)
=> #<Song @id=1 @title=nil @lyrics=<not loaded> @length=nil @released_on=nil>
[32] pry(main)> Song.first
=> #<Song @id=1 @title=nil @lyrics=<not loaded> @length=nil @released_on=nil>
[33] pry(main)> Song.last
=> #<Song @id=2 @title="Come fly with me" @lyrics=<not loaded> @length=199 @released_on=nil>
[35] pry(main)> x = Song.first(title: 'Come fly with me')
=> #<Song @id=2 @title="Come fly with me" @lyrics=<not loaded> @length=199 @released_on=nil>
```

```
[44] pry(main)> y = Song.first(title: 'My Way')
=> #<Song @id=1 @title="My Way" @lyrics=<not loaded> @length=435 @released_on=nil>
[45] pry(main)> y.length
=> 435
[46] pry(main)> y.update(length: 275)
=> true
```

En Sqlite3:

```
sqlite> select * from songs;
1|My Way|And now, the end is near ...|275|
2|Come fly with me|Come fly with me, let's fly, let's fly away ...|199|
```

Borrando

```
[47] pry(main)> Song.create(title: "One less lonely girl")
=> #<Song @id=3 @title="One less lonely girl" @lyrics=<not loaded> @length=<not loaded> @relea
[48] pry(main)> Song.last.destroy
=> true
[49] pry(main)> Song.all
=> [#<Song @id=1 @title="My Way" @lyrics=<not loaded> @length=275 @released_on=nil>, #<Song @i
```

Búsqueda con Condiciones Rather than defining conditions using SQL fragments, we can actually specify conditions using a hash.

The examples above are pretty simple, but you might be wondering how we can specify conditions beyond equality without resorting to SQL. Well, thanks to some clever additions to the `Symbol` class, it's easy!

```
exhibitions = Exhibition.all(:run_time.gt => 2, :run_time.lt => 5)
# => SQL conditions: 'run_time > 1 AND run_time < 5'
```

Valid symbol operators for the conditions are:

```
gt    # greater than
lt    # less than
gte   # greater than or equal
lte   # less than or equal
not   # not equal
eql   # equal
like  # like
```

Veamos un ejemplo de uso con nuestra clase `Song`:

```
[31] pry(main)> Song.all.each do |s|
[31] pry(main)*   s.update(length: rand(400))
[31] pry(main)* end
=> [#<Song @id=1 @title="My Way" @lyrics=<not loaded> @length=122 @released_on=nil>,
   #<Song @id=2 @title="Come fly with me" @lyrics=<not loaded> @length=105 @released_on=nil>,
   #<Song @id=4 @title="Girl from Ipanema" @lyrics=<not loaded> @length=389 @released_on=nil>]
[32] pry(main)> long = Song.all(:length.gt => 120)
=> [#<Song @id=1 @title="My Way" @lyrics=<not loaded> @length=122 @released_on=nil>,
   #<Song @id=4 @title="Girl from Ipanema" @lyrics=<not loaded> @length=389 @released_on=nil>]
```

Insertando SQL Sometimes you may find that you need to tweak a query manually:

```
[40] pry(main)> songs = repository(:default).adapter.select('SELECT title FROM songs WHERE len
=> ["My Way", "Girl from Ipanema"]
```

Note that this will not return `Song` objects, rather the raw data straight from the database

main.rb

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ cat main.rb
require 'sinatra'
require 'slim'
require 'sass'
require './song'

configure do
  enable :sessions
  set :username, 'frank'
  set :password, 'sinatra'
end

configure :development do
  DataMapper.setup(:default, "sqlite3://#{Dir.pwd}/development.db")
end

configure :production do
  DataMapper.setup(:default, ENV['DATABASE_URL'])
end

get('/styles.css'){ scss :styles }

get '/' do
  slim :home
end

get '/about' do
  @title = "All About This Website"
  slim :about
end

get '/contact' do
  slim :contact
end

not_found do
  slim :not_found
end

get '/login' do
  slim :login
end

post '/login' do
  if params[:username] == settings.username && params[:password] == settings.password
```

```
    session[:admin] = true
    redirect to('/songs')
  else
    slim :login
  end
end

get '/logout' do
  session.clear
  redirect to('/login')
end
```

32.4. Configurando la Base de Datos en Heroku con DataMapper. Despliegue

Heroku utiliza la base de datos PostgreSQL y una URL en una variable de entorno `ENV['DATABASE_URL']`.

```
configure :development do
  DataMapper.setup(:default, "sqlite3://#{Dir.pwd}/development.db")
end

configure :production do
  DataMapper.setup(:default, ENV['DATABASE_URL'])
end
```

Estas líneas especifican que se usa SQLite en desarrollo y PostgreSQL en producción. Obsérvese que el `Gemfile` debe estar coherente:

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ cat Gemfile
source 'https://rubygems.org'
gem "sinatra"
gem "slim"
gem "sass"
gem "dm-core"
gem "dm-migrations"
gem "thin"
gem "pg", :group => :production
gem "dm-postgres-adapter", :group => :production
gem "dm-sqlite-adapter", :group => :development
```

o mejor:

```
group :production do
    gem "pg"
    gem "dm-postgres-adapter"
end

heroku create ...

git push heroku master

heroku open

heroku logs --source app
```

Ahora ejecutamos la consola de heroku:

```
heroku run console
```

lo que nos abre una sesión `irb`.

Ahora creamos la base de datos en Heroku:

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku run console
Running ‘console‘ attached to terminal... up, run.8011
irb(main):001:0> require ’./main’
=> true
irb(main):002:0> DataMapper.auto_migrate!
=> #<DataMapper::DescendantSet:0x007fb89c878230 @descendants=#<DataMapper::SubjectSet:0x007fb8
irb(main):003:0>
```

Véase también la practica TicTacToe 18.28 y el capítulo *Despliegue en Heroku* **??**. .

Capítulo 33

Slim

Capítulo 34

Oauth: Google, Twitter, GitHub, Facebook

34.1. Introduction to OAuth

OAuth 2 is rapidly becoming a preferred authorization protocol, and is used by major service providers such as Google, Facebook and Github.

Valet Key for the Web

Many luxury cars come with a valet key. It is a special key you give the parking attendant and unlike your regular key, will only allow the car to be driven a short distance while blocking access to the trunk and the onboard cell phone.

Regardless of the restrictions the valet key imposes, the idea is very clever. You give someone limited access to your car with a special key, while using another key to unlock everything else.

As the web grows, more and more sites rely on distributed services and cloud computing:

- a photo lab printing your Flickr photos,
- a social network using your Google address book to look for friends, or
- a third-party application utilizing APIs from multiple services.

The problem is, in order for these applications to access user data on other sites, they ask for usernames and passwords. Not only does this require exposing user passwords to someone else – often the same passwords used for online banking and other sites – it also provides these application unlimited access to do as they wish. They can do anything, including changing the passwords and lock users out.

OAuth provides a method for users (you) to grant third-party (parking attendant) access to their resources (your luxury car) without sharing their passwords (the key of your car). It also provides a way to grant limited access (in scope, duration, etc. the equivalent of not having access to the trunk or the onboard cell phone).

For example,

- a web user (resource owner) can grant a
- printing service (client)
- access to her private photos (partial resource)
- stored at a photo sharing service (server),
- without sharing her username and password with the printing service.

Instead, she authenticates directly with the photo sharing service which issues the printing service delegation-specific credentials.

In OAuth, the client requests access to resources controlled by the resource owner and hosted by the resource server, and *is issued a different set of credentials than those of the resource owner.*

Instead of using the resource owner's credentials to access protected resources, the client obtains an access token – *a string denoting a specific scope, lifetime, and other access attributes.*

Access tokens are issued to third-party clients by an authorization server with the approval of the resource owner.

The client uses the access token to access the protected resources hosted by the resource server.

1. The OAuth 2.0 Authorization Framework proposed standard document

Roles in OAuth OAuth defines four roles:

1. *resource owner*

   An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an *end-user.*

2. *resource server*

   The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

3. *client*

   An application making protected resource requests on behalf of the resource owner and with its authorization.

   The term "client"does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).

4. *authorization server*

   The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

The authorization server may be the same server as the resource server or a separate entity. A single authorization server may issue access tokens accepted by multiple resource servers.

Véase

- Nacho Coloma: Our love-hate relationship with OAuth

34.2. Google Developers Console

34.2.1. Managing projects and applications

A project consists of

1. a set of applications,
2. along with activated APIs,
3. Google Cloud resources, and
4. the team and billing information associated with those resources.

Credentials such as API keys are specific to an application rather than to a project.

However, all applications within a given project use the same branding information (logo, email address, etc.) on their user consent screen, as described in Setting up OAuth 2.0.

Applications within a project also share

1. activated APIs,
2. permissions, and
3. billing information.

Managing project members

If you create a project, you have owner-level permissions and can grant owner-level permissions to other project members. Those with owner-level permissions are project owners.

Only project owners can add and remove other project members and edit their permission levels. Project owners can share a project with an email address that represents a group, but every project must have at least one project member that is an individual, not a group.

To manage project members, do the following:

1. Visit the Google Developers Console
2. Select a project, or create a new one.
3. In the sidebar on the left, select `Permissions`.
4. To add a team member or group, select `Add Member`.
   - You must provide an email address that is associated with a Google account.
   - If the email address belongs to an individual, an invitation flow is triggered, and the new project member must accept the invitation before they can access the project.
   - If the email address belongs to a group, the group is added right away, with no invitation step.
5. To change the permission setting for a project member, click the dropdown box in the Permission column and select a new permission level. The new permission level is saved automatically.
6. To delete a project member, click the trash icon to the right of the project member's permission setting.

34.2.2. Keys, access, security, and identity

Each request to an API that is represented in the Google Developers Console must include a unique identifier.

Unique identifiers enable the Developers Console to tie requests to specific projects in order to

- monitor traffic,
- enforce quotas, and
- handle billing.

Google supports two mechanisms for creating unique identifiers:

1. OAuth 2.0 client IDs

   For applications that use the OAuth 2.0 protocol to call Google APIs, you can use an OAuth 2.0 client ID to generate an access token. The token contains a unique identifier.

2. API keys

   An API key (either a server key or a browser key) is a unique identifier that you generate using the Developers Console. Using an API key does not require user action or consent. API keys do not grant access to any account information, and are not used for authorization.

Use an API key when your application is running on a server and accessing one of the following kinds of data:

- Data that the data owner has identified as public, such as a public calendar or blog.
- Data that is owned by a Google service such as Google Maps or Google Translate. (Access limitations may apply.)
- If you are only calling APIs that do not require user data, such as the Google Custom Search API, then API keys might be simpler to use than OAuth 2.0 access tokens. However, if your application already uses an OAuth 2.0 access token, then there is no need to generate an API key as well. Google ignores passed API keys if a passed OAuth 2.0 access token is already associated with the corresponding project.

34.3. OmniAuth gem: Standardized Multi-Provider Authentication for Ruby

- OmniAuth

OmniAuth is a library that standardizes multi-provider authentication for web applications. Any developer can create *strategies* for OmniAuth that can authenticate users via disparate systems.

OmniAuth strategies have been created for everything from Facebook to LDAP.

To use OmniAuth, you need only

1. to redirect users to `/auth/:provider`, where `:provider` is the name of the strategy (for example, `developer` or `twitter`).
2. From there, OmniAuth will take over and take the user through the necessary steps to authenticate them with the chosen strategy.
3. Once the user has authenticated, OmniAuth sets a special hash called the *Authentication Hash* on the Rack environment of a request to `/auth/:provider/callback`.
4. This hash contains as much information about the user as OmniAuth was able to glean from the utilized strategy.
5. You should set up an endpoint in your application that matches to the callback URL and then performs whatever steps are necessary for your application.

Getting Started

To use OmniAuth in a project with a Gemfile, just add each of the strategies you want to use individually:

```
gem 'omniauth-github'
gem 'omniauth-openid'
```

Now you can use the `OmniAuth::Builder` Rack middleware to build up your list of OmniAuth strategies for use in your application:

Para saber mas sobre Rack y sobre Middlewares Rack, véanse las secciones

- Rack en 16,
- *Middleware y la Clase Rack::Builder* en 16.18 y
- *La Estructura de una Aplicación Rack* en 16.14

```
use OmniAuth::Builder do
  provider:github, ENV['GITHUB_KEY'], ENV['GITHUB_SECRET']
  provider:openid, :store => OpenID::Store::Filesystem.new('/tmp')
end
```

By default, OmniAuth will return auth information to the path `/auth/:provider/callback` inside the Rack environment.

In Sinatra, for example, a callback might look something like this:

```
# Support both GET and POST for callbacks
%w(get post).each do |method|
  send(method, "/auth/:provider/callback") do
    env['omniauth.auth'] # => OmniAuth::AuthHash
  end
end
```

Also of note, by default, if user authentication fails on the provider side, OmniAuth will catch the response and then redirect the request to the path `/auth/failure`, passing a corresponding error message in a parameter named `message`.

You may want to add an action to catch these cases. Continuing with the previous Sinatra example, you could add an action like this:

```
get '/auth/failure' do
  flash[:notice] = params[:message] # if using sinatra-flash or rack-flash
  redirect '/'
end
```

Strategies

In this link we can find a list of the strategies that are available for OmniAuth: List of Strategies for Omniauth.

34.3.1. Auth Hash Schema

OmniAuth is a flexible authentication system utilizing Rack middleware.

OmniAuth will always return a hash of information after authenticating with an external provider in the Rack environment under the key `omniauth.auth`.

This information is meant to be as normalized as possible, so the schema below will be filled to the greatest degree available given the provider upon authentication. Fields marked required will always be present.

-
- `provider (required)` The provider with which the user authenticated (e.g. `twitter` or `facebook`)
- `uid (required)` An identifier unique to the given provider, such as a Twitter user ID. Should be stored as a string.
- `info (required)` A hash containing information about the user
  - `name (required)` The best display name known to the strategy. Usually a concatenation of first and last name, but may also be an arbitrary designator or nickname for some strategies
  - `email` The e-mail of the authenticating user. Should be provided if at all possible (but some sites such as Twitter do not provide this information)
  - `nickname` The username of an authenticating user (such as your `@-name` from Twitter or GitHub account name)
  - `first_name`

- `last_name`
- `location` The general location of the user, usually a city and state.
- `description` A short description of the authenticating user.
- `image` A URL representing a profile image of the authenticating user. Where possible, should be specified to a square, roughly 50x50 pixel image.
- `phone` The telephone number of the authenticating user (no formatting is enforced).
- `urls` A hash containing key value pairs of an identifier for the website and its URL.

  For instance, an entry could be

  `"Blog" => "http://intridea.com/blog"`

- `credentials` If the authenticating service provides some kind of access token or other credentials upon authentication, these are passed through here.
- `token` Supplied by OAuth and OAuth 2.0 providers, the access token.
- `secret` Supplied by OAuth providers, the access token secret.
- `extra` Contains extra information returned from the authentication provider. May be in provider-specific formats.
- `raw_info` A hash of all information gathered about a user in the format it was gathered.

  For example, for Twitter users this is a hash representing the JSON hash returned from the Twitter API.

Ejemplos

- Omniauth Sinatra Example (Twitter y OpenID)
- Omniauth Sinatra Gist Example: Facebook, Twitter, GitHub

Documentación de la Gema

- API doc: OmniAuth: Standardized Multi-Provider Authentication
- Module: OmniAuth
- Esta clase contiene información sobre el usuario. Class: OmniAuth::AuthHash::InfoHash

Véase

- Blog Post: ColdFusion and OAuth part 3- Google authentication. Raymond Camden
-

34.4. OmniAuth OAuth2 gem

- omniauth-oauth2

This gem contains a generic OAuth2 strategy for OmniAuth.

It is meant to serve as a building block strategy for other strategies and not to be used independently, since it has no inherent way to gather uid and user info.

34.5. The gem omniauth-google-oauth2

Introducción The

gem omniauth-google-oauth2 provides a strategy to authenticate with Google via OAuth2 in OmniAuth.

Get your API key at:

https://code.google.com/apis/console/

Note the Client ID and the Client Secret.

For more details, read the Google docs:

https://developers.google.com/accounts/docs/OAuth2.

Configuration

You can configure several options, which you pass in to the `provider` method via a hash:

-

- `scope` A comma-separated list of permissions you want to request from the user. See the Google OAuth 2.0 Playground for a full list of available permissions.

  Caveats:

  - The `userinfo.email` and `userinfo.profile` scopes are used by default. By defining your own scope, you override these defaults. If you need these scopes, don't forget to add them yourself!
  - Scopes starting with `https://www.googleapis.com/auth/` do not need that prefix specified.

    So while you can use the smaller scope `books` since that permission starts with the mentioned prefix, you should use the full scope URL `https://docs.google.com/feeds/` to access a user's docs, for example.

- `prompt` A space-delimited list of string values that determines whether the user is re-prompted for authentication and/or consent. Possible values are:

  - `none` No authentication or consent pages will be displayed; it will return an error if the user is not already authenticated and has not pre-configured consent for the requested scopes. This can be used as a method to check for existing authentication and/or consent.
  - `consent` The user will always be prompted for consent, even if he has previously allowed access a given set of scopes.
  - `select_account` The user will always be prompted to select a user account. This allows a user who has multiple current account sessions to select one amongst them.

  - If no value is specified, the user only sees the authentication page if he is not logged in
  - and only sees the consent page the first time he authorizes a given set of scopes.

- `image_aspect_ratio` The shape of the user's profile picture. Possible values are:

  - `original` Picture maintains its original aspect ratio.
  - `square` Picture presents equal width and height. Defaults to `original`.

- `image_size` The size of the user's profile picture.

  The image returned will have width equal to the given value and variable height, according to the `image_aspect_ratio chosen`.

  Additionally, a picture with specific width and height can be requested by setting this option to a hash with `width` and `height` as keys.

If only width or height is specified, a picture whose width or height is closest to the requested size and requested aspect ratio will be returned.

Defaults to the original width and height of the picture.

- `name` The name of the strategy. The default name is `google_oauth2` but it can be changed to any value, for example `google`. The OmniAuth URL will thus change to `/auth/google` and the `provider` key in the auth hash will then return `google`.
- `access_type` Defaults to `offline`, so a refresh token is sent to be used when the user is not present at the browser. Can be set to `online`.

  Note that if you need a refresh token, google requires you to also to specify the option prompt: 'consent', which is not a default.
- `login_hint` When your app knows which user it is trying to authenticate, it can provide this parameter as a hint to the authentication server. Passing this hint suppresses the account chooser and either pre-fill the email box on the sign-in form, or select the proper session (if the user is using multiple sign-in), which can help you avoid problems that occur if your app logs in the wrong user account. The value can be either an email address or the sub string, which is equivalent to the user's Google+ ID.
- `include_granted_scopes` If this is provided with the value `true`, and the authorization request is granted, the authorization will include any previous authorizations granted to this user/application combination for other scopes. See Google's Incremental Autorization for additional details.

Here's an example of a possible configuration where

- the strategy name is changed,
- the user is asked for extra permissions,
- the user is always prompted to select his account when logging in and
- the user's profile picture is returned as a thumbnail:

```
Rails.application.config.middleware.use OmniAuth::Builder do
  provider :google_oauth2, ENV["GOOGLE_CLIENT_ID"], ENV["GOOGLE_CLIENT_SECRET"],
    {
      :name => "google",
      :scope => "userinfo.email, userinfo.profile, plus.me, http://gdata.youtube.com",
      :prompt => "select_account",
      :image_aspect_ratio => "square",
      :image_size => 50
    }
end
```

Auth Hash Here's an example of an authentication hash available in the callback by accessing `request.env["omniauth.auth"]`:

```
{
    :provider => "google_oauth2",
    :uid => "123456789",
    :info => {
        :name => "John Doe",
        :email => "john@company_name.com",
        :first_name => "John",
```

```
        :last_name => "Doe",
        :image => "https://lh3.googleusercontent.com/url/photo.jpg"
    },
    :credentials => {
        :token => "token",
        :refresh_token => "another_token",
        :expires_at => 1354920555,
        :expires => true
    },
    :extra => {
        :raw_info => {
            :id => "123456789",
            :email => "user@domain.example.com",
            :verified_email => true,
            :name => "John Doe",
            :given_name => "John",
            :family_name => "Doe",
            :link => "https://plus.google.com/123456789",
            :picture => "https://lh3.googleusercontent.com/url/photo.jpg",
            :gender => "male",
            :birthday => "0000-06-25",
            :locale => "en",
            :hd => "company_name.com"
        }
    }
}
```

34.6. Using OAuth 2.0 to Access Google APIs

Véase Using OAuth 2.0 to Access Google APIs.

Google APIs use the OAuth 2.0 protocol for authentication and authorization. Google supports common OAuth 2.0 scenarios such as those for web server, installed, and client-side applications.

1. OAuth 2.0 is a relatively simple protocol. To begin, you obtain OAuth 2.0 credentials from the Google Developers Console.

2. See the Video in YouTube Obtaining a developer key for the YouTube Data API v3 and the Analytics API

3. Then your client application requests an access token from the Google Authorization Server, extracts a token from the response, and sends the token to the Google API that you want to access.

To get access keys, go to the Google APIs Console and specify your Google Developers Console. application's name and the Google APIs it will access. For simple access, Google generates an API key that uniquely identifies your application in its transactions with the Google Auth server.

For authorized access, you must also tell Google your website's protocol and domain. In return, Google generates a client ID. Your application submits this to the Google Auth server to get an OAuth 2.0 access token.

34.7. Google OAuth 2.0 Playground

Google OAuth 2.0 Playground

34.8. Sign-in with Google +

Google+ Sign-in provides the OAuth 2.0 authentication mechanism along with additional access to Google desktop and mobile features.

- https://developers.google.com/+/ provides the OAuth 2.0 authentication mechanism along with additional access to Google desktop and mobile features.
- Direct access to an authentication service based on the standardized OpenID Connect mechanism is also available. https://developers.google.com/accounts/docs/OAuth2Login

34.9. Revoking Access to an App

Go to https://security.google.com/settings/security/permissions

Or to your configuration https://www.google.com/settings/personalinfo and there to **security**. From there go to the section **Account permissions** (*Control which apps and websites have access to your account information*). Choose the link **View All**.

- Remember to kill the session if you want to check that the permit has been effectively removed.
- Go to chrome and open a window in incognito mode
- Attempt to access the URL that requires Google Authentication in your service
- It has to ask you for permits again

34.10. Google + API for Ruby

Donde

- https://github.com/crguezl/gplus-quickstart-ruby
- ```
  [~/sinatra/gplus-quickstart-ruby(master)]$ pwd -P
  /Users/casiano/local/src/ruby/sinatra/gplus-quickstart-ruby
  [~/sinatra/gplus-quickstart-ruby(master)]$ git remote -v
  googleplus      git@github.com:googleplus/gplus-quickstart-ruby.git (fetch)
  googleplus      git@github.com:googleplus/gplus-quickstart-ruby.git (push)
  origin  git@github.com:crguezl/gplus-quickstart-ruby.git (fetch)
  origin  git@github.com:crguezl/gplus-quickstart-ruby.git (push)
  ```
- https://github.com/crguezl/gplus-quickstart-ruby
- https://developers.google.com/+/quickstart/ruby

The app demonstrates:

- Using the Google+ Sign-In button to get an OAuth 2.0 refresh token.
- Exchanging the refresh token for an access token.
- Making Google+ API requests with the access token, including getting a list of people that the user has circled.
- Disconnecting the app from the user's Google account and revoking tokens.
  ```

Step 1: Enable the Google+ API

Create a Google APIs Console project, OAuth 2.0 client ID, and register your JavaScript origins: To register a new application, do the following:

- Go to the Google Cloud Console.
- Select a project, or create a new one.
- In the sidebar on the left, select `APIs & auth`. In the displayed list of APIs, make sure the `Google+ API` status is set to `ON`.
- In the sidebar on the left, select `Registered apps`.
- At the top of the page, select Register App.
- Fill out the form and select Register.

Register the origins where your app is allowed to access the Google APIs. The origin is the unique combination of protocol, hostname, and port. You can enter multiple origins to allow for your app to run on different protocols, domains or subdomains. Wildcards are not allowed.

- Expand the OAuth 2.0 Client ID section.
- In the Web origin field, enter your origin:

  ```
  http://localhost:4567
  ```

- Press ENTER to save your origin. You can then click the + symbol to add additional origins.
- Note or copy the client ID and client secret that your app will need to use to access the APIs.

Client Secrets

34.11. Google+ Sign-In for server-side apps

To take advantage of all of the benefits of Google+ Sign-In you must use a hybrid server-side flow where a user authorizes your app on the client side using the JavaScript API client and you send a special one-time authorization code to your server.

Your server exchanges this one-time-use code to acquire its own access and refresh tokens from Google for the server to be able to make its own API calls, which can be done while the user is offline.

This one-time code flow has security advantages over both a pure server-side flow and over sending access tokens to your server.

The Google+ Sign-In server-side flow differs from the OAuth 2.0 for Web server applications flow.

34.12. Authentication using the Google APIs Client Library for JavaScript

See.

Parte V

PARTE: BITÁCORA DEL CURSO

Capítulo 35

2014

35.1. 01

35.1.1. Semana del 27/01/14 al 01/02/2014

- Presentación de la Asignatura
- Ejercicio: Darse de alta en la comunidad de google plus PL Grado ULL 13/14
- JavaScript Review
- *Expresiones Regulares y Análisis Léxico en JavaScript* 1
- *Conversor de Temperaturas* 1.2
- *GitHub Project Pages* 13.4.4

35.2. 02

35.2.1. Semana del 4/02/14 al 7/02/2014

- Martes 4/02. *Comma Separated Values. CSV* Sección 1.3. Secciones: Donde, Introducción al formato CSV, Ejemplo de ejecución, Aproximación al análisis mediante expresiones regulares de CSV.

35.2.2. Semana del 24/02/14 al 02/03/14. Repaso para el micro-examen del 05/03/14

1. ¿Que retorna?

   ```
   "hello small world and blue sky".match(/(\S+)\s+(\S+)/);
   ```

2. Indique que casa con el primer paréntesis y que con el segundo en las siguientes expresiones regulares:

   ```
   > x = "I have 2 numbers: 53147"
   > pats = [ /(.*)(\d*)/,
              /(.*)(\d+)/,
              /(.*?)(\d*)/,
              /(.*?)(\d+)/,
              /(.*)(\d+)$/,
              /(.*?)(\d+)$/,
              /(.*)\b(\d+)$/,
              /(.*\D)(\d+)$/ ]
   ```

Es decir, compute la salida de:

```
pats.map( function(r) { return r.exec(x).slice(1); })
```

3. ¿Que retorna el matching?:

```
>  a = "hola juan"
 => "hola juan"
> a.match(/(?:hola )*(juan)/)
```

4. ¿Que salidas se obtienen?

```
> "a\na".match(/a$/)
--------------------------------
> "a\na".match(/a$/m)
--------------------------------
> "a\na".match(/^a/gm)
------------
> "a\na".match(/^a/g)
-------
```

5. Escriba la expresión regular que da lugar a este resultado (enumerar las líneas):

```
> x = "one\ntwo\nthree\nfour"
'one\ntwo\nthree\nfour'
> a = (c = 1, x.replace(_____, function(t) { return  c++ + ' ' + t; }))
'1 one\n2 two\n3 three\n4 four'
> console.log(a)
1 one
2 two
3 three
4 four
undefined
```

6. Supongamos dado el método

```
String.prototype.repeat = function( num ) {
    return new Array( num + 1 ).join( this );
}
```

de manera que podamos escribir expresiones como:

```
> x = 'a'.repeat(40)
'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
```

Encontremos una solución de la ecuación diofántica `3x + 2y + 5z = 40`

```
> m = x.match(/^______________________________$/).slice(1)
[ 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa',
  'aa',
  'aaaaa' ]
```

Calculemos las longitudes de las tres cadenas:

```
> r = m.map(function(s) { return s.length; })
[ 33, 2, 5 ]
```

Dividamos por los coeficientes para obtener la solución:

```
> coef = [3, 2, 5]
> i = 0; w = r.map(function(x) { return x/coef[i++]; }
[ 11, 1, 1 ]
```

Encuentre la expresión regular usada.

7. Escriba una expresión regular que reconozca cadenas de dobles comillas como `"hello world"` y en las que las comillas puedan aparecer escapadas como en `"Hello \"Jane\" and Jakes"`

8. Escriba una expresión regular que reconozca los números en punto flotante como `2.34`, `-5.2e-1` y `0.9e3`

9. ¿Que queda en `m[0]`?

   ```
   m = 'main() /* 1c */ { /* 2c */ return; /* 3c */ }'.match(new RegExp('/\\*.*\\*/'))
   ```

   ¿Por qué?

10. ¿Por qué debemos duplicar el carácter de escape \ en la expresión regular `new RegExp('/\\*.*\\*/')` de la pregunta anterior 9?

11. Se quiere poner un espacio en blanco después de la aparición de cada coma:

    ```
    > 'ab,cd,4,3,   de,   fg'.replace(/,/, ', ')
    => "ab, cd, 4, 3,    de,    fg"
    ```

    pero se quiere que la sustitución no tenga lugar si la coma esta incrustada entre dos dígitos. Además se pide que si hay ya un espacio después de la coma, no se duplique

    Como función de reemplazo use:

    ```
    f = function(match, p1, p2, offset, string) { return (p1 || p2 + " "); }
    ```

12. Escribe un patrón regular que reconozca las cadenas que representan números no primos en unario de manera que el primer paréntesis case con el divisor mas grande del número.

13. Escribe un patrón regular que reconozca las cadenas que representan números no primos en unario de manera que el primer paréntesis case con el divisor mas pequeño del número.

14. Escriba una expresión regular que reconozca los comentarios del lenguaje JavaScript de la forma `// ...`

15. Escriba una expresión regular que reconozca los comentarios del lenguaje JavaScript de la forma `/* ... */`

16. Rellene lo que falta para que la salida sea la que aparece en la sesión de node:

    ```
    > re = __________
    > str = "John Smith"
    'John Smith'
    > newstr = str.replace(re, "______")
    'Smith, John'
    ```

17. Rellene las partes que faltan:

```
> re = /d(b+)(d)/ig
/d(b+)(d)/gi
> z = "dBdxdbbdzdbd"
'dBdxdbbdzdbd'
> result = re.exec(z)
[ ______, _____, ______, index: __, input: 'dBdxdbbdzdbd' ]
> re.lastIndex
______
> result = re.exec(z)
[ ______, _____, ______, index: __, input: 'dBdxdbbdzdbd' ]
> re.lastIndex
______
> result = re.exec(z)
[ ______, _____, ______, index: __, input: 'dBdxdbbdzdbd' ]
> re.lastIndex
______
> result = re.exec(z)
_____
```

18. Escriba la expresión regular `r` para que produzca el resultado final:

```
> x = "hello"
> r = /l(___)/
> z = r.exec(x)
[ 'l', index: 3, input: 'hello' ]
```

19.
```
> z = "dBdDBBD"
> re = /d(b+)(d)/ig
> re.lastIndex = ________
> result = re.exec(z)
[ 'DBBD',
  'BB',
  'D',
  index: 3,
  input: 'dBdDBBD' ]
```

20. Conteste:

    a) Explique que hace el siguiente fragmento de código:

```
> RegExp.prototype.bexec = function(str) {
...   var i = this.lastIndex;
...   var m = this.exec(str);
...   if (m && m.index == i) return m;
...   return null;
... }
[Function]
```

    b) Rellene las salidas que faltan:

```
> re = /d(b+)(d)/ig
/d(b+)(d)/gi
> z = "dBdXXXXDBBD"
'dBdXXXXDBBD'
```

```
> re.lastIndex = 3
> re.bexec(z)
-----------------------------------------------------
> re.lastIndex = 7
> re.bexec(z)
-----------------------------------------------------
```

21. Escriba una expresión JavaScript que permita reemplazar todas las apariciones de palabras repetidas en una String por una sóla aparición de la misma

22. Supongamos que se usa una función como segundo argumento de `replace`. ¿Que argumentos recibe?

23. ¿Cual es la salida?

```
> "bb".match(/b|bb/)

> "bb".match(/bb|b/)
```

24. El siguiente fragmento de código tiene por objetivo escapar las entidades HTML para que no sean intérpretadas como código HTML. Rellene las partes que faltan.

```
var entityMap = {
    "&": "&___;",
    "<": "&__;",
    ">": "&__;",
    '"': '&quot;',
    "'": '&#39;',
    "/": '&#x2F;'
  };

function escapeHtml(string) {
  return String(string).replace(/_________/g, function (s) {
    return ____________;
  });
```

25. ¿Cual es la salida?

```
> a = [1,2,3]
[ 1, 2, 3 ]
> b = [1,2,3]
[ 1, 2, 3 ]
> a == b
--------
```

26. ¿Como se llama el método que permite obtener una representación como cadena de un objeto? ¿Que parámetros espera? ¿Como afectan dichos parámetros?

27. ¿Cual debe ser el valor del atributo `rel` para usar la imagen como favicon?

```
<link rel="_____________" href="etsiiull.png" type="image/x-icon">
```

28. Escriba un código JavaScript que defina una clase `Persona` con atributos `nombre` y `apellidos` y que disponga de un método `saluda`.

29. Reescriba la solución al problema anterior haciendo uso del método `template` de `underscore` y ubicando el template dentro de un tag `script`.

30. Rellene lo que falta:

```
[~/srcPLgrado/temperature/tests(master)]$ cat tests.js
var assert = chai.______;

suite('temperature', function() {
    test('[1,{a:2}] == [1,2]', function() {
      assert._________([1, {a:2}], [1, {a:2}]);
    });
    test('5X = error', function() {
        original.value = "5X";
        calculate();
        assert._____(converted.innerHTML, /ERROR/);
    });
});
```

31. ¿Cómo se llama el directorio por defecto desde el que una aplicación sinatra sirve los ficheros estáticos?

32. Explique la línea:

```
set :public_folder, File.dirname(__FILE__) + '/starterkit'
```

¿Que es `__FILE__`? ¿Que es `File.dirname(__FILE__)`? ¿Que hace el método `set`? (Véase http://www.sinatrarb.com/configuration.html)

33. Escriba un programa sinatra que cuando se visite la URI `/chuchu` muestre una página que diga `"hello world!"`

34. ¿Cual es el signifcado de `__END__` en un programa Ruby?

35. Esta y las preguntas 36 y 37 se refieren al mismo programa ruby sinatra. Explique este fragmento de dicho programa ruby sinatra.

```
@@layout
  <!DOCTYPE html>
  <html>
    <head>
        <meta charset="utf-8" />
        <title>Demo</title>
    </head>
    <body>
        <a href="http://jquery.com/">jQuery</a>
        <div class="result"></div>
        <script src="jquery.js"></script>
        <%= yield %>
    </body>
  </html>
```

   a) ¿En que lugar del fichero que contiene el programa está ubicada esta sección?
   b) ¿Cómo se llama el lenguaje en el que esta escrita esta sección?
   c) ¿Para que sirve la sección `layout`?

d) ¿Cual es la función del `<div class="result"></div>`?

e) ¿Para que sirve el `<%= yield %>`?

36. Explique este fragmento de un programa ruby sinatra.

```
@@index
  <script>
  $( document ).ready(function() {
      $( "a" ).click(function( event ) {
          event.preventDefault();
          $.get( "/chuchu", function( data ) {
            $( ".result" ).html( data );
            alert( "Load was performed." );
          });
      });
  });
  </script>
```

a) ¿Cuando ocurre el evento `ready`?

b) ¿Que hace `event.preventDefault()`?

c) ¿Que hace `$.get( "/chuchu", function( data ) { ... }`? ¿Cuando se dispara la callback?

d) ¿que hace la línea `$( ".result" ).html( data )`?

37. Explique este fragmento de código ruby-sinatra:

```
get '/chuchu' do
  if request.xhr?
    "hello world!"
  else
    erb :tutu
  end
end
```

38. En el siguiente programa - que calcula la conversión de temperaturas entre grados Farenheit y Celsius - rellene las partes que faltan:

a) index.html:

```
<html>
  <head>
      <meta http-equiv="Content-Type" content="text/html; charset=_____">
      <title>JavaScript Temperature Converter</title>
      <link ____="global.css" ___="stylesheet" ____="text/css">

     <script type="_______________" src="temperature.js"></script>
  </head>
  <____>
    <h1>Temperature Converter</h1>
    <table>
      <tr>
        <th>Enter  Temperature (examples: 32F, 45C, -2.5f):</th>
        <td><input id="________" ________="calculate();"></td>
      </tr>
```

```
        <tr>
          <th>Converted Temperature:</th>
          <td><span class="output" id="_________"></span></td>
        </tr>
      </table>
    </____>
  </html>
```

b) Rellene las partes del código JavaScript que faltan en `temperature.js`:

```
"use strict"; // Use ECMAScript 5 strict mode in browsers that support it
function calculate() {
  var result;
  var original       = document.getElementById("________");
  var temp = original.value;
  var regexp = /________________________________/;

  var m = temp.match(______);

  if (m) {
    var num = ____;  // paréntesis correspondiente
    var type = ____;
    num = parseFloat(num);
    if (type == 'c' || type == 'C') {
      result = (num * 9/5)+32;
      result = ______________________________ // 1 sólo decimal y el tipo
    }
    else {
      result = (num - 32)*5/9;
      result = ____________________________ // 1 sólo decimal y el tipo
    }
    converted._________ = result; // Insertar "result" en la página
  }
  else {
    converted._________ = "ERROR! Try something like '-4.2C' instead";
  }
}
```

39. ¿Que hace `autofocus`?

```
<td><textarea autofocus cols = "80" rows = "5" id="original"></textarea></td>
```

40. ¿Que hacen las siguientes pseudo-clases estructurales CSS3?

```
tr:nth-child(odd)    { background-color:#eee; }
tr:nth-child(even)    { background-color:#00FF66; }
```

41. ¿Que contiene el objeto `window` en un programa JavaScript que se ejecuta en un navegador?

42. *a*) ¿Que es Local Storage? ¿Que hace la siguiente línea?

```
if (window.localStorage) localStorage.original  = temp;
```

b) ¿Cuando se ejecutará esta callback? ¿Que hace?

```
window.onload = function() {
  // If the browser supports localStorage and we have some stored data
```

```
  if (window.localStorage && localStorage.original) {
    document.getElementById("original").value = localStorage.original;
  }
};
```

43. ¿Cómo se hace para que elementos de la página web permanezcan ocultos para posteriormente mostrarlos? ¿Que hay que hacer en el HTML, en la hoja de estilo y en el JavaScript?

44. Rellene los estilos para los elementos de las clases para que su visibilidad case con la que su nombre indica:

```
.hidden      { display: ____; }
.unhidden    { display: _____; }
```

45. Los siguientes textos corresponden a los ficheros de la práctica de construcción de un analizador léxico de los ficheros de configuración INI. Rellena las partes que faltan.

    a) Rellena las partes que faltan en el contenido del fichero `index.html`. Comenta que hace el tag `<input>`. Comenta que hace el tag `<pre>`.

```
<html>
  <head>
     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
     <title>INI files</title>
     <link href="global.css" rel="__________" type="text/css">

     <script type="_______________" src="underscore.js"></script>
     <script type="_______________" src="jquery.js"></script>
     <script type="_______________" src="______"></script>
  </head>
  <body>
    <h1>INI files</h1>
    <input type="file" id="_________" />
    <div id="out" class="hidden">
    <table>
      <tr><th>Original</th><th>Tokens</th></tr>
      <tr>
        <td>
          <pre class="input" id="____________"></pre>
        </td>
        <td>
          <pre class="output" id="___________"></pre>
        </td>
      </tr>
    </table>
    </div>
  </body>
</html>
```

    b) A continuación siguen los contenidos del fichero `ini.js` conteniendo el JavaScript.

    1) Rellena las partes que faltan. El siguiente ejemplo de fichero `.ini` le puede ayudar a recordar la parte de las expresiones regulares

```
; last modified 1 April 2001 by John Doe
[owner]
name=John Doe
organization=Acme Widgets Inc.
```

2) Explica el uso del template.
3) Explica el uso de JSON.stringify

```
"use ______"; // Use ECMAScript 5 strict mode in browsers that support it

$(document)._____(function() {
   $("#fileinput").______(calculate);
});

function calculate(evt) {
  var f = evt.target.files[0];

  if (f) {
    var r = new __________();
    r.onload = function(e) {
      var contents = e.target.______;

      var tokens = lexer(contents);
      var pretty = tokensToString(tokens);

      out.className = 'unhidden';
      initialinput.________ = contents;
      finaloutput.________ = pretty;
    }
    r.__________(f); // Leer como texto
  } else {
    alert("Failed to load file");
  }
}

var temp = '<li> <span class = "<%= ______ %>"> <%= _ %> </span>\n';

function tokensToString(tokens) {
   var r = '';
   for(var i in tokens) {
     var t = tokens[i];
     var s = JSON.stringify(t, undefined, 2); //______________________________
     s = _.template(temp, {t: t, s: s});
     r += s;
   }
   return '<ol>\n'+r+'</ol>';
}

function lexer(input) {
  var blanks         = /^___/;
  var iniheader      = /^________________/;
  var comments       = /^________/;
  var nameEqualValue = /^________________________/;
  var any            = /^_______/;

  var out = [];
  var m = null;

  while (input != '') {
```

```
      if (m = blanks.____(input)) {
        input = input.substr(m.index+___________);
        out.push({ type : ________, match: _ });
      }
      else if (m = iniheader.exec(input)) {
        input = input.substr(__________________);
        _______________________________________ // avanzemos en input
      }
      else if (m = comments.exec(input)) {
        input = input.substr(__________________);
        _________________________________________
      }
      else if (m = nameEqualValue.exec(input)) {
        input = input.substr(__________________);
        _______________________________________________
      }
      else if (m = any.exec(input)) {
        _______________________________________
        input = '';
      }
      else {
        alert("Fatal Error!"+substr(input,0,20));
        input = '';
      }
    }
    return out;
  }
```

35.3. Proyecto: Diseña e Implementa un Lenguaje de Dominio Específico

Se trata de realizar un proyecto relacionado con el procesamiento de lenguajes. El objetivo puede ser:

1. Diseñar un lenguaje de dominio específico para simplificar cualquier tarea en la que estés interesado:

    - Para escribir exámenes,
        - Por ejemplo se puede escribir un traductor para el formato Moodle gift que traduzca a javascript + HTML + css y que evalúe al usuario
        - Por ejemplo se puede escribir un traductor para el formato Moodle XML que traduzca a javascript + HTML + css y que evalúe al usuario
        -
    - Para dibujar árboles,
    - Para calcular fechas,
    - Para generar emails
    - Para escribir música
    - Para escribir autómatas finitos
    - Para procesar CSS
    - etc.

2. Estudiar un traductor existente en profundidad como:
   - ECMAscript 5.1: Creating a JavaScript Parser Una implementación de ECMAScript 5.1 usando Jison disponible en GitHub en https://github.com/cjihrig/jsparser. Puede probarse en: http://www.cjihrig.com/development/jsparser/
   - Roy
   - CoffeScript
   - Jison
   - Javascript 1.4
   - etc.
3. También puedes proponer tu propio tema relacionado al profesor

Se recomienda para ello organizar equipos de no menos de dos y no mas de cuatro.
Las presentaciones de los proyectos tendrán lugar el último día de clase Martes 21 de Mayo.

35.4. 03

35.5. 04

35.5.1. Semana del 07/04/14 al 11/04/14. Repaso para el micro-examen del 09/04/14

35.6. 05

35.6.1. Repaso para la prueba del 14/05/2014

1. *a*) Complete las partes que faltan de esta calculadora escrita en Jison:

```
%___
%%

\s+                   /* skip whitespace */
[0-9]+("."[0-9]+)?\b  return 'NUMBER'
"*"                   return '*'
"/"                   return '/'
"-"                   return '-'
"+"                   return '+'
"^"                   return '^'
"!"                   return '!'
"%"                   return '%'
"("                   return '('
")"                   return ')'
"PI"                  return 'PI'
"E"                   return 'E'
<<EOF>>               return 'EOF'
.                     return 'INVALID'

/___

%left _______
%left _______
%____'^'
%_____    '!'
%______   '%'
```

```
%left UMINUS

%start ___________

%% /* language grammar */

expressions
    : e EOF
        { typeof console !== 'undefined' ? console.log($1) : print($1);
          return $1; }
    ;

e
    : e '+' e
        {___________}
    | e '-' e
        {___________}
    | e '*' e
        {___________}
    | e '/' e
        {___________}
    | e '^' e
        {__ = Math.pow(______);}
    | e '!'
        {{
          $$ = (function fact (n) { return n==0 ? _ : _____________ })($1);
        }}
    | e '%'
        {$$ = $1/100;}
    | '-' e %____ ______
        {_________}
    | '(' e ')'
        {________}
    | NUMBER
        {____________________}
    | E
        {$$ = Math.E;}
    | PI
        {$$ = Math.PI;}
    ;
```

b) ¿Cómo habría que modificar la calculadora para que las frases de la forma `4-5-8` se interpretaran como `4-(5-8)`?

c) ¿Cómo habría que modificar la calculadora para que las frases de la forma `4-5*8` se interpretaran como `(4-5)*8)`?

d) ¿Cómo habría que modificar la calculadora para que las frases de la forma `-5*8` se interpretaran como `-(5*8)`?

e) ¿Cómo habría que modificar el analizador léxico para que se admitieran comentarios tipo javascript `/* ... */`?

f) ¿Con que comando compilamos la gramática Jison anterior para producir el parser?

2. Complete el algoritmo de análisis LR.

- $|x|$ denota la longitud de la cadena x.

- La función top(k) devuelve el elemento que ocupa la posición k contando desde el *top* de la pila
- La función pop(k) extrae k elementos de la pila.
- La notación state.attr hace referencia al atributo asociado con cada estado.
- Denotamos por Sem el hash de acciones semánticas

```
push(__);
b = yylex();
for( ; ; ;) {
  s = top(0); a = b;
  switch (action[_][_]) {
    case "shift t" :
      t.attr = _.____;
      push(_);
      b = _______;
      break;
    case "reduce A ->alpha" :
      eval(Sem{_ __ _____}(top(|alpha|-1).attr, ... , top(0).attr));
      pop(|_____|);
      push(goto[______][_]);
      break;
    case "accept" : return (1);
    default : yyerror("syntax error");
  }
}
```

3. a) Escriba un autómata finito no determinista que reconozca el lenguaje de los prefijos viables de la gramática:

   ```
   e: e '-' e | NUM ;
   ```

   b) Usando la construcción del subconjunto encuentre un autómata finito determinista que sea equivalente al construído en el apartado anterior

   c) Encuentre el conjunto de terminales que puede aparecer a continuación de e en una derivación de la gramática. No se olvide de considerar el terminale END-OF-INPUT. Use el carácter '$' para representarlo.

4. Complete las partes que faltan del fichero auth.rb que permite autentificación usando OAuth mediante la gema omniauth:

```
require 'omniauth-oauth2'
require 'omniauth-google-oauth2'

___ OmniAuth::Builder do
  config = YAML.load_file 'config/config.yml'
  ________ :google_oauth2, config['identifier'], config['secret']
end

get '/auth/:name/callback' do
  session[:auth]  = @auth = request.env['________.____']
  session[:name]  = @auth['info'].____
  session[:image] = @auth['info']._____
  flash[:notice] =
      %Q{<div class="success">Authenticated as #{session[:name]}.</div>}
```

```
  redirect '/'
end

get '/auth/failure' do
  flash[:notice] = params[:message]
  redirect '/'
end
```

5. Complete el código para el análisis de ámbito en esta extensión de la calculadora que admite funciones anidadas.

   En primer lugar tenemos las estructuras de datos necesarias:

```
%{
var symbolTables = [{ name: '', father: null, vars: {} }];
var scope = 0;
var symbolTable = symbolTables[scope];
```

   a) Complete este accessor para `scope`:

```
function getScope() {
  return _____;
}
```

   b) La función `getFormerScope` se llama cuando se sale de un ámbito:

```
function getFormerScope() {
   _______;
   symbolTable = symbolTables[_____];
}
```

   c) La función `makeNewScope` se llama cada vez que se entra en un nuevo ámbito:

```
function makeNewScope(id) {
   _______;
   symbolTable.vars[id].symbolTable = symbolTables[_____] =
                 { name: id, father: symbolTable, vars: {} };
   symbolTable = symbolTables[_____];
   return symbolTable;
}
```

   d) Necesitamos una función `findSymbol` para encontrar donde está la definición del objeto cuyo nombre es `x`:

```
function findSymbol(x) {
  var f;
  var s = scope;
  do {
    f = symbolTables[s].vars[x];
    ___;
  } while (______ && !f);
  s++;
  return [f, s];
}
```

   e) Cuando se detecta una llamada se hacen comprobaciones en la tabla de símbolos:

```
function functionCall(name, arglist) {
  var info = findSymbol(name);
```

```
    var s = info[1];
    info = info[0];

    if (!info || info.type != 'FUNC') {
      ________________________________________;
    }
    else if(arglist.length != info.arity) {
      _________________________________________;
    }
    ....
  }
```

f) Complete el código que falta en la acción semántica asociada con la definición de una función:

```
dec
    : DEF functionname  optparameters "{" decs statements "}"
                  {
                     ______________();

                     $$ = translateFunction(...);
                  }
    | VAR varlist ';'   { ... }
    ;
```

g) En la produción anterior aparecía `functionname` el cual produce simplemente un identificador:

```
functionname
    : ID
                  {
                     if (symbolTable.vars[$ID])
                       throw ____________________________________________;
                     symbolTable.vars[$ID] = { type: ______, name: ___ };

                     ____________($ID);

                     $$ = $ID;
                  }
    ;
```

¿Por qué se crea esta regla de producción? Rellene el código que falta.

h) Cuandos se analiza la definición de parámetros es preciso crear la entrada y guardar la información relevante. Complete el código:

```
parameters
    : ID                {
                          symbolTable.vars[___] = { type : '_____' };
                          $$ = [ $ID ];
                        }
    | parameters "," ID {
                          symbolTable.vars[___] = { type : '_____' };
                          $$ = $1;
                          $$.push($ID);
                        }
    ;
```

i) Cuando se usa un identificador hay que comprobar que su uso es conforme a su definición:

```
e
    : ID "=" e
        {
           var info = findSymbol($ID);
           var s = info[1];
           info = info[0];

           if (info && info.type === 'VAR') {
             $$ = binary($e,unary("&"+$ID+", "+(getScope()-s)), "=");
           }
           else if (info && info.type === 'PAR') {
             $$ = binary($e,unary("&$"+$ID+", "+(getScope()-s)), "=");
           }
           else if (info && info.type === 'FUNC') {
              ______________________________________________________;
           }
           else {
              ______________________________________________;
           }
        }
    | ....
```

6. Responda a las siguientes preguntas sobre transformaciones árbol:
   a) Defina alfabeto con aridad
   b) Defina el conjunto de todos los árboles sobre un alfabeto
   c) Defina gramática árbol
   d) Defina lenguaje generado por una gramática árbol
   e) ¿Que es un patrón árbol? ¿Que significa que un árbol casa con un patrón árbol?
   f) Defina que es un esquema de transformaciones árbol
   g) Escriba en pseudocódigo un esquema de transformaciones árbol para el plegado de constantes
7. Calcule los conjuntos `FIRST` y `FOLLOW` para las variables sintácticas de esta gramática:

```
S : A B 'a' A | 'b' S B
  ;
A : /* vacío */ | A 'a'
  ;
B : /* vacío */ | B 'b'
  ;
```

8. Escriba un traductor de expresiones en infijo a postfijo usando Jison. Una expresión como `a+2*b` deberá traducirse como `a 2 b * +`.
9. Explique que código se debería añadir al ejercicio anterior para traducir una expresion `if expression then e`
10. Haga un diagrama de un stackframe de una llamada explicando las diferentes partes para un lenguaje que admite procedimientos anidados.

Índice general

Índice de figuras

Índice de cuadros

Índice alfabético

Bibliografía

[1] Mark Pilgrim. *Dive into HTML5*. http://diveinto.html5doctor.com/index.html, 2013.

[2] Jeffrey E.F. Friedl. *Mastering Regular Expressions*. O'Reilly, USA, 1997. ISBN 1-56592-257-3.

[3] G. Wilson and A. Oram. *Beautiful Code: Leading Programmers Explain How They Think*. O'Reilly Media, 2008.

[4] Nathan Whitehead. *Create Your Own Programming Language. http://nathansuniversity.com/*. 2012.

[5] Nathan Whitehead. *What's a Closure?. http://nathansjslessons.appspot.com/*. 2012.

[6] Steven S. Muchnick. *Advanced compiler design and implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[7] T. Mogensen and T.A. Mogensen. *Introduction to compiler design*. Undergraduate topics in computer science. Springer London, Limited, 2011.

[8] Todd A. Proebsting. Burg, iburg, wburg, gburg: so many trees to rewrite, so little time *(invited talk)*. In *ACM SIGPLAN Workshop on Rule-Based Programming*, pages 53–54, 2002.