Universidad de La Laguna. Escuela Técnica Superior de Ingeniería Informática
Tercero del Grado de Informática
PROCESADORES DE LENGUAJES: 2ª PARTE
05/04/2017 5 páginas

Nombre: _____
Alu: _____ GitHub Id: _____ GitHub Team: _____

Preguntas de Repaso de Expresiones Regulares

1. Escriba expresiones regulares que casen con las siguientes especificaciones:

    1. car and cat
    2. pop and prop
    3. ferret, ferry, and ferrari
    4. Any word ending in ious
    5. A whitespace character followed by a dot, comma, colon, or semicolon
    6. A word longer than six letters
    7. A word without the letter e

2. Escriba una expresión regular que reconozca las cadenas de doble comillas. Debe permitir la presencia de comillas y caracteres escapados.

3. Escriba una expresión regular que reconozca los números en punto flotante (por ejemplo `-2.3e-1`, `-3e2`, `23`, `3.2`). numbers = /^ ... $/, matching exacto

4. Escriba una expresión regular que case con los números no primos expresados en unario. Pruebe con `1111`, `111`, `111111`, `1111111`, ...

5. Escriba una expresión regular que case con los comentarios JavaScript.

6. Escriba una expresión JavaScript que permita reemplazar todas las apariciones de palabras consecutivas repetidas (como `hello hello`) por una sóla aparición de la misma

7. ¿Cual es la salida?

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

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

    Justifique su respuesta.

8. 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 ____________;
  });
```

2. Se quiere poner un espacio en blanco después de la aparición de cada coma:

```
> 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'
```

Se pide que si hay ya un espacio después de la coma, no se duplique.

3. Se pide una expresión regular que case con expresiones del tipo `identifier = number` y retorne con cada paréntesis el identificador y el número. Pruebe con `h = 4`, `temp = 5.6`, `x23= -2.3e1` y `z += 3`

4. Imagine you have written a story and used single quotation marks throughout to mark pieces of dialogue. Now you want to replace all the dialogue quotes with double quotes, while keeping the single quotes used in contractions like *aren't*. Think of a pattern that distinguishes these two kinds of quote usage and craft a call to the replace method that does the proper replacement.

```
var text = "'I'm the cook,' he said, 'it's my job.'";
// Change this call.
var result = text.replace(/.../g, '...');
console.log(result);
var expected = `"I'm the cook," he said, "it's my job."`;
if (expected === result) console.log("OK")
else console.log("ERROR!");
```

Preguntas de Repaso de Análisis Léxico

1. Escriba un método `bexec` para los objetos `RegExp`que funcione como `exec` pero sólo case en la posición `lastIndex`:

```
RegExp.prototype.bexec = function(str) {
  -------------------------
  -------------------------
  -------------------------
  -------------------------
  -------------------------
  -------------------------
  -------------------------
};
```

2. Basándose en el método `bexec` del ejercicio anterior escriba las partes que faltan del método `tokens` que implementa un analizador léxico para un lenguaje de tipo JavaScript en el que `if` y `then` son palabras reservadas:

```
String.prototype.tokens = function() {
  var RESERVED_WORD, from, getTok, i, key, m, make, n, result, rw, tokens, value;
  from = 0;
  i = 0;
  n = 0;
  m = 0;
  result = [];
  tokens = {
    WHITES: ___________________________________________,
    ID: ________________________________________,
    NUM: _______________________________________,
    STRING: ____________________________________________,
    ONELINECOMMENT: ____________________________________________,
    MULTIPLELINECOMMENT: ____________________________________________,
    COMPARISONOPERATOR: ____________________________________________,
    ONECHAROPERATORS: /([=()&|;:,{}[\]])/g,
    ADDOP: /[+-]/g,
    MULTOP: /[*\/]/g
  };
  RESERVED_WORD = {
    "if": "IF",
    then: "THEN"
  };
```

Explique como se usa en el análisis el objeto `RESERVED_WORD` y para que sirve. Observe su uso en el bucle

```
make = function(type, value) {
  return {
    type: type,
    value: value,
    from: from,
    to: i
  };
};
getTok = function() {
  var str;
  str = m[0];
  i += _____________________;
  return str;
};
if (!this) {
  return;
}
while (i < this.length) {
  for (key in tokens) {
    value = tokens[key];
    ________________________
  }
  from = __________;
  if (m = tokens.WHITES.bexec(this) ||
     (m = tokens.ONELINECOMMENT.bexec(this)) ||
     (m = tokens.MULTIPLELINECOMMENT.bexec(this))) {
    _________________________________;
  } else if (m = tokens.ID.bexec(this)) {
    rw = RESERVED_WORD[m[0]];
    if (rw) {
      result.push(make(rw, getTok()));
    } else {
      result.push(_____________________________________);
    }
  } else if (m = tokens.NUM.bexec(this)) {
    n = +getTok();
    if (isFinite(n)) {
      result.push(_____________________________________);
    } else {
      make("NUM", m[0]).error("Bad number");
    }
  } else if (m = tokens.STRING.bexec(this)) {
    result.push(make("STRING", getTok().replace(________________________/g, "")));
  } else if (m = tokens.COMPARISONOPERATOR.bexec(this)) {
    result.push(make("COMPARISON", getTok()));
  } else if (m = tokens.ADDOP.bexec(this)) {
```

```
      result.push(make("ADDOP", getTok()));
    } else if (m = tokens.MULTOP.bexec(this)) {
      result.push(make("MULTOP", getTok()));
    } else if (m = tokens.ONECHAROPERATORS.bexec(this)) {
      result.push(make(m[0], getTok()));
    } else {
      throw "Syntax error near '" + (this.substr(i)) + "'";
    }
  }
  return result;
};
```