Universidad de La Laguna. ESIT Informática
Tercero del Grado de Informática. Computación
PROCESADORES DE LENGUAJES. JULY UNIQUE CALL
Use English for your answers. 2 pages

Name, Alu and GitHub ID:: _____

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

1. An *object* is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by `:` (colon) and the name/value pairs are separated by `,` (comma).

2. An *array* is an ordered collection of values. An array begins with `[` (left bracket) and ends with `]` (right bracket). Values are separated by `,` (comma).

3. A *value* can be a

    1. *string* in double quotes,
    2. or a *number*,
    3. or `true`
    4. or `false`
    5. or `null`,
    6. or an *object*
    7. or an *array*.

4. Whitespace can be inserted between any pair of tokens.

These structures can be nested. That completely describes the language.

1. **To do if you have not passed the 3d exam (Pegs, LR)**.

   Write a PEGJS that receives as input a JSON input and returns the corresponding JavaScript object

   (a) Be cautious when parsing `null` in PEGjs. In PEGjs you can't return `null` inside a semantic action because that would mean parse failure

   (b) A *string* is a sequence of zero or more Unicode characters, wrapped in double quotes. To help you with your work, here is the rule for a single `char`:

```
char
  = [^"\\\0-\x1F\x7f]
  / '\\"'  { return '"';  }
  / "\\\\" { return "\\"; }
  / "\\/"  { return "/";  }
  / "\\b"  { return "\b"; }
  / "\\f"  { return "\f"; }
  / "\\n"  { return "\n"; }
  / "\\r"  { return "\r"; }
  / "\\t"  { return "\t"; }
  / "\\u" digits:$(hexDigit hexDigit hexDigit hexDigit) {
      return String.fromCharCode(parseInt("0x" + digits));
    }

hexDigit
  = [0-9a-fA-F]
```

The `String` method `fromCharCode()` converts a Unicode number into a character.

(c) A *number* is very much like a C or JavaScript number, except that the octal and hexadecimal formats are not used. To simplify your work, here are some PEGjs definitions you can assume in your solution:

```
int
  = digit19 digits
  / digit
  / "-" digit19 digits
  / "-" digit

frac = "." digits

exp = e digits

digits = digit+

e = [eE] [+-]?

digit = [0-9]

digit19 = [1-9]
```

2. (a) Write a Jison program that solves the problem posed in the first question. It has to include the lexical analyzer, the grammar and the necessary semantic actions for building the JavaScript Object. (**To do if you have not passed the 3d exam (PEGs, LR)**).

   (b) Write a trace of the LR algorithm for the input `{"a": 4 }` (**skip if you go to cont. eval.**)

3. **To do if you have not passed the 2nd exam (PDR)**.

   Write a JavaScript Recursive Descent Parser for the JSON language as posed in the first question (**skip the lexer if you go to cont. eval.**)