## **Aims**

## The aims of this project are as follows:

## To encourage you to use regular expressions to implement a trivial scanner.

## To make you implement a recursive-descent parser for a small language.

## To use [JSON](https://www.json.org/json-en.html) to represent the results of parsing.

## **Requirements**

* A string in the language consists of a sequence of zero-or-more **declarations**.
* A **declaration** consists of the reserved word var, an **identifier**, a colon :, a **type** and a semi-colon ;.
* A **type** consists of one of the following:
  1. The reserved word number.
  2. The reserved word string.
  3. The reserved word record followed by one-or-more **field declarations** followed by the reserved word end.
* A **field declaration** consists of an **identifier**, a colon :, a **type** and a semi-colon ;.
* An **identifier** consists of a sequence of alphanumerics or underscores \_ not starting with a digit.

Whitespace and #-to-end-of-line comments should be ignored.

1. Run the make.sh and decls.sh to compile and run the program
2. Running decls.sh from any directory will parse text read from standard input as per the above grammar and output on standard output a JSON representation of the parse:
   * The declarations should be output as a JSON array of 2-element JSON arrays [ *id*, *type* ].
   * A *type* should be output as follows:
     1. If the *type* is number, then it should be output as the JSON string "number".
     2. If the *type* is string, then it should be output as the JSON string "string".
     3. If the *type* is record, then it should be output as a JSON array of *field declarations*.
   * A *field declaration* declaring *id* to have type *type* should be output as a JSON 2-element array [ *id*, *type* ], where *type* is output as above.

The JSON output should consist of a single line without any whitespace other than the newline terminator.

If there are errors in the content, the program should exit with a non-zero status after detecting the first syntax error. It should output a suitable error message on standard error.

An annotated log of the running project and the provided tests should help clarify the above requirements.

## **Rationale for the Requirements**

The requirements are based on the following rationale:

* The specified language is a simple language containing nested record-end constructs. Implementing a parser for this language allows you to understand the basic principles of recursive-descent parsing.

At first glance, the specified JSON output appears unnatural. For example, the natural JSON for  
 var a: number;

var pt: record

x: number;

y: number;

end;

appears to be something like:  
 { "a": "number",

"pt": {

"x": "number",

"y": "number"

}

}

The above JSON may be produced naturally from some kind of map data structure, but in many programming languages, the order of the keys in a map is undefined. Hence it is possible that the order of the keys within the output may vary depending on the implementation language.  
To avoid such issues, the required JSON output uses only JSON arrays:  
 [ ["a", "number"],

["pt", [

["x", "number"],

["y", "number"],

]]

]