The repository defines a compiler from LaTeX-mixed-with-J to LaTeX code. The script jlatex.ijs defines the verb `latex` to convert a J phrase into a well-formatted LaTeX math mode string. This is then used in a compiler, which converts a file (I use .jtex as an extension, although it doesn't matter) into a .tex document and compiles that with pdflatex.

This compiler uses the system defined in jdoc.ijs, which replaces statements enclosed by `\?()` for certain names `?` with the result of executing a specified J function on them. The mapping of names to functions is found in jdoc.ijs and is as follows:

- J: Execute as J, and print the result with `": .`
- A: Execute as J, but return an empty string (used for assignment).
- L: Execute using the verb `latex`.
- P: Execute using `pd`.
- S: The first line is a name, and the rest is a string. Assign the string to the name.
- I: Like L, but place in inline math mode.
- D: Place in display mode. Includes escapes for `.` and `,` at the end of the expression so they will be used as punctuation.

The argument strings are recursively expanded. Note that inside each, parentheses must be balanced (including, unfortunately, parentheses within quotes). These functions are used dynamically, which means you can change them merely by reassigning in the locale `_pjdoc_`. This document assigns `C` for an inline code snippet and `E` for examples (giving the code, then the result).

The function `latex` allows J to function as it usually does, but makes changes to the way arithmetic operators work. It manipulates both J variables and LaTeX expressions. Each LaTeX expression is contained in a box, so it can be treated as an atom and used in arrays without concern for details. Functions modified to produce LaTeX results will convert their inputs to LaTeX before operating on them if necessary.

The syntax is precisely J's, except for the special character \, which works as follows:

- `\const`, where `const` is a number or string, converts that value to a LaTeX expression.
- `\fun`, where `fun` begins with an alphabetic, gives a LaTeX function, which will apply as `\fun{y}` or `\fun{x}{y}`.
- `\_val` gives a LaTeX literal `\val`, which is a noun.
- `\\op` gives a verb which functions as an operator, applying as `\op y` or `x \op y`.

- `\op`, where `b` starts with a special character or ends with one of `.:`, gives the J function b.
- `\(expr)` (with parentheses) interprets `(expr)` as a J expression and does not change it.

Also, the new primitive `[.` forces parentheses around an expression.

The code redefines arithmetic functions (the script opdata.ijs gives the full list) to give LaTeX-formatted outputs. However, all other functions will work as they do in J, provided they are applied to arguments which have not yet been formatted in LaTeX (i.e. `i.4` will work fine, but `i.4+5` will give a domain error). Adverbs, conjunctions, hooks, and forks perform excellently, including rank.

Names which are not defined at execution are changed into LaTeX literals. This allows the use of J variables, while making use of literals easier. A name or string can always be enclosed in quotes if needed. One particular case is as the target of assignment—names must be quoted or J will return a domain error.

Parentheses are added when necessary and usually only when necessary—the converter is aware of order of operations, associativity, and which operations need parentheses (i.e. `+` does, `^` needs them only on the left, and `%` needs none).

The following are a few examples of J code converted to LaTeX.

```
(2*a)%~(-b)\\pm %:(b^2)-4*a*c
```

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
+/ *: a,b,c
```

$$a^2 + b^2 + c^2$$

```
\_pi * 2*rh+r^2
```

$$\pi \cdot 2(rh + r^2)$$

```
(+1&\cfrac)/ (}:,+&\_ddots@{:) a_"0 i.5
```

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \ddots}}}}$$