Lisp in Summer Projects Submission

Submission Date	2013-10-24 02:48:20
Full Name	Zachary Kanfer
Country	USA
Project Name	unit-test.arc
Type of software	library
General category	library
LISP dialect	Arc
GitHub URL	https://github.com/zck/unit-test.arc
Did you start this project?	No, I'm modifying or extending an existing project.
Which file or directory contains the majority of your work?	entire repository
Briefly describe your modifications	All the code in this repository is written by me, but I started this project before the summer kicked off. The only commit from before Lisp in Summer Projects is the first commit; that is, https://github.com/zck/unit-test.arc/commit/a227b2da032b10efe3ef476e6a19593cdac15ed8 In this commit, I had a very rough sketch of how everything should work. There was a lot of duplicate definitions, but the code worked to create suites and tests. It was not able to run the tests successfully. I added many features, and fixed a lot of bugs. The complete list can be found in the bug tracker (https://bitbucket.org/zck/unit-test.arc/issues?status=resolved), but a summary of features: the ability to unit test macros nested suites setup functions README documentation gensyms so variable scope isn't captured

	a lot of refactoring better usability: both in terms of the code the user writes and the output from the library
Project Description	I want to describe my project in this form.
Purpose	To be able to simply write and run unit tests in Arc. To easily re-run tests when the code is changed, and, at a glance, to see the status of all the tests run at a time.
Function	unit-test.arc provides functions and macros to define and run unit tests. A unit test is a small segment of code that tests a single function, and reports whether the function acts as expected.
Motivation	Existing unit test libraries in Arc are not elegant -- they all are missing one or more features that unit-test.arc has: it lets tests be simply defined, run as a group, and output is easy to parse, and is summarized.
Audience	The audience is any developer using Arc who wants to write unit tests. Hopefully that's every developer using Arc.
Methodology	The entry point to creating tests is the `suite` macro. It goes through several layers of macros until it gets to the `suite-partition` macro, which is the most complicated part of the library. It introspects the code passed into it, and determines the shape of the first thing in the suite -- is it a suite, a suite with variables to be setup, or a test? It then recursively creates suites, and creates tests. Suites, tests, and the results of running those two are stored as templates. In Arc, a template is somewhere between a struct and a hashtable. Suites are stored in the *unit-tests* global hash; Suite results are in the *unit-test-results* global hash. To facilitate running nested suites, not only root suites are stored in *unit-tests*, but also nested suites. This way, nested suites can be run without running any parent or sibling suites (if you define a suite _math_ with nested suites _adding_ and _subtracting_, you can run _math.adding_ without running tests from any other suite, including tests directly in _math_). The entry point to running suites is the `run-suites` macro. It's comparatively simple compared to the code-introspecting behavior required to create the suites. Eval is used once, in make-test-fn. It's used to delay macroexpansion until the unit test is run -- we want to be able to test macros without relying on the macroexpanded code from when the unit test was created. An interesting point is that there are 17 macros defined, and 18 functions. This is required to make tests simple to write -- test names are barewords, not strings or quoted symbols. Tests are simple blocks of code, not quoted code. One disappointing part of doing this project is finding how fragmented the Arc community is. Although many of them were incredibly helpful when macro problems arose, I was not able to convince any other developers to use this library. One was interested, but only after the library is ported to his

Arc-esque language.

Conclusion

This framework has accomplished the goals set out for it. There are no big features left to add.

One common feature of unit testing frameworks is teardown functionality; that is, at the end of a test run, run some code. A possible elegant solution is requiring it to be written as follows:

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

setup and _teardown_ would be literal symbols put in. This feature will wait until it's needed; no tests using this suite have yet required teardowns.

Other possibilities include: improving error messages -- strings are printed with single quotes, so code printed as errors can't be pasted back into the repl. I could also clean up the various asserts, refactor _suite-partition_, and run tests in the same order they were declared.

Some work can still be done to easily let developers find the failing tests. Striking a good balance between verbosity (printing out all the tests that pass) and conciseness (not burying failed tests in a sea of passed tests) is difficult, but necessary. The output is good now, but could be improved.

Build Instructions

Install arc:
(Note: these instructions supercede the official arc instructions)
Download http://ycombinator.com/arc/arc3.1.tar and racket

```
tar -xf arc3.1.tar
```

Run arc:

```
cd /location/of/arc
racket -f as.scm
```

Load unit-test.arc:

```
(load "/path/to/unit-test.arc")
```

Test Instructions

Load the tests:

```
(load "/path/to/tests.arc")
```

Run the tests:

```
(run-suite unit-test-tests)
```

Execution Instructions

If you've ran the unit tests for this project, congratulations! You've already executed the code! There's a quickstart guide in the repository, but try this code:

```
(suite lisp-in-summer-projects this-test-will-pass (assert-
same 21 (len "Thanks for reviewing!"))
this-test-will-fail (assert-t (pos 'zck *winners*)))
```

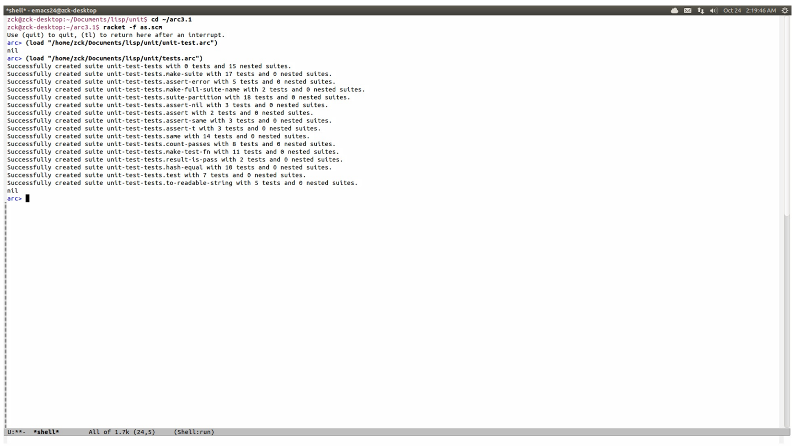
Then make the test pass.

Describe any bugs or caveats

There are no known bugs. I'm tracking the issues here (https://bitbucket.org/zck/unit-test.arc/issues?

status=new&status=open), and they consist only of a few improvements to the code that aren't bugs.

Screen shots



```
*shell* - emacs24@zck-desktop
zck@zck-desktop:~/Documents/lisp/unit$ cd ~/arc3.1
zck@zck-desktop:~/arc3.1$ racket -f as.scm
Use (quit) to quit, (tl) to return here after an interrupt.
arc> (load "/home/zck/Documents/lisp/unit/unit-test.arc")
nil
arc> (load "/home/zck/Documents/lisp/unit/tests.arc")
Successfully created suite unit-test-tests with 0 tests and 15 nested suites.
Successfully created suite unit-test-tests.make-suite with 17 tests and 0 nested suites.
Successfully created suite unit-test-tests.assert-error with 5 tests and 0 nested suites.
Successfully created suite unit-test-tests.make-full-suite-name with 2 tests and 0 nested suites.
Successfully created suite unit-test-tests.suite-partition with 18 tests and 0 nested suites.
Successfully created suite unit-test-tests.assert-nil with 3 tests and 0 nested suites.
Successfully created suite unit-test-tests.assert with 2 tests and 0 nested suites.
Successfully created suite unit-test-tests.assert-same with 3 tests and 0 nested suites.
Successfully created suite unit-test-tests.assert-t with 3 tests and 0 nested suites.
Successfully created suite unit-test-tests.same with 14 tests and 0 nested suites.
Successfully created suite unit-test-tests.count-passes with 8 tests and 0 nested suites.
Successfully created suite unit-test-tests.make-test-fn with 11 tests and 0 nested suites.
Successfully created suite unit-test-tests.result-is-pass with 2 tests and 0 nested suites.
Successfully created suite unit-test-tests.hash-equal with 10 tests and 0 nested suites.
Successfully created suite unit-test-tests.test with 7 tests and 0 nested suites.
Successfully created suite unit-test-tests.to-readable-string with 5 tests and 0 nested suites.
nil
arc> 

U:**-  *shell*       All of 1.7k (24,5)     (Shell:run)
```

load.png

test-run.png

Official

I have read rules and have abided by them.
I am 18 years of age or older.
I am not living in Brazil, Quebec, Saudi Arabia, Cuba, Iran, Myanmar (Burma), North Korea, Sudan, or Syria.