Bonuses

This Document might be nicer to read on GitHub

REPL

All of the following bonuses may be run in our fully featured interactive prompt.

You may build it by running...

```
make build
```

...and then run it by doing

```
./sun-tzu_lnk
```

Syntax Documentation for compiled code

Arrays

We implemented syntactic sugar for LISP style lists so that they may be written using a more familiar style

`(1 2 3 4 5)` -> `[1, 2, 3, 4, 5]`

We chose syntactic sugar over native handling as it will result in less written code in project overall.

Arrays can contain any amount of Atomic values.

Array Nesting

Arrays can have an infinite amount of nesting, to represent 2d arrays, 3d arrays and so on.

`[[1,2,3,4,5],[1,2,3,4,5]]`

Array Operation

There are a total of 7 array operations:

Head

`head` returns the first element of an array.

```
> (define a [1, 2, 3])
> (head a)
1
```

Last

`last` returns the first element of an array.

```
> (define a [1, 2, 3])
> (last a)
3
```

Tail

`tail` returns the array with its first element removed.

```
> (define a [1, 2, 3])
> (tail a)
[2,3]
```

Init

`init` returns the array with its last element removed.

```
> (define a [1, 2, 3])
> (init a)
[1,2]
```

Join

`join` concatenates 2 arrays and returns it.

```
> (define a [1, 2, 3])
> (define b [4, 5, 6])
> (join a b)
[1,2,3,4,5,6]
```

PushFront

`pushFront` adds an element to the beginning of an array and returns the array.

```
> (define a [2, 3, 4])
> (define b 1)
> (pushFront a b)
[1,2,3,4]
```

Pushback

`pushBack` adds an element to the end of an array and returns the array.

```
> (define a [1, 2, 3])
> (define b 4)
> (pushBack a b)
[1,2,3,4]
```

Strings and character escaping

It is possible to use strings in our code, they are written using the following syntax : `"Hello World!"`. It is also possible to escape the character `"` to use it inside of strings.

```
> (print "\"Hello World!\"")
"Hello World!"
```

Input/Output

It is possible to interact with the local filesystem and ask for user input to built interactive programs.

Reading from Stdin

```

```

```
> (+ (readInt) 2)
in: 3
out: 5
```

```
> (+ (readInt) (readInt))
in: 5
in: 6
out: 11
```

Writing to Stdout

There are 2 functions for writing to stdout, `print` and `println`.

```
> (print "Hello World!")
Hello World! -- no newline
```

```
> (println "Hello World!")
Hello World! -- newline
```

Floating point numbers

Our language supports **floating point numbers** as well as any **corresponding basic arithmetic operation** and full array support.

```
> (3.2 + 4.5)
7.7
```

```
> (3.2 - 4.5)
-1.3
```

```
> (3.2 * 4.5)
14.400001
```

```
> (3.2 / 4.5)
0.7111111
```

Rigor

We would also like to take the time to point out a couple best practices that we implemented rigorously while working on this project.

Here are a couple of these points :

- Issue tracking - **Over 60 issues**
- Pull request workflow - **Over 90 pull requests**
- Concise and meaningful commits - **Over 500 commits**
- Commit linting using commitizen
- Lots of code commenting
- Unit Testing - **Over 100** tests
- Integration Testing - **Over 20** end to end integration tests

We also added **an other ABNF style** document along with some documentation for our custom assembly like language that we compile into.

Return to the main page