stack egg

Provides LIFO queue (stack) operations.
Extension for Chicken Scheme
Version 1.3

Kon Lovett

Table of Contents

1 About this egg

1.1 Version history

`1.3` Exports

`1.2` Added clear-stack

`1.1` Doc fix [thanks to Toby Butzon], rename of stack-pop-1!, bug fix

`1.0` Initial release

1.2 Requirements

This egg requires the following extensions:

`srfi-1`, `syntax-case`

1.3 Usage

Load this egg like so:

```
(require-extension stack)
```

2 Documentation

stack is a set of procedures and macros supporting LIFO queue operations. The stack is treated as an independent object, which is modified in place. In opposition to the common pattern of a variable being re-assinged a new stack representation after each mutating operation.

`make-stack` [macro]

```
(make-stack [LIST])
```

Returns a new stack, with optional initial elements from `LIST`.

`clear-stack` [macro]

```
(clear-stack STACK)
```

Make `STACK` empty.

`stack?` [macro]

```
(stack? STACK)
```

Is `STACK` a stack?

`stack-empty?` [macro]

```
(stack-empty? STACK)
```

Returns #t for an empty `STACK`, #f otherwise.

`stack-length` [macro]

```
(stack-length STACK)
```

Returns the number of elements on the `STACK`.

`stack-depth` [macro]

```
(stack-depth STACK)
```

Returns the index of the last element on the `STACK`, or -1 when stack empty.

`stack-peek` [macro]

```
(stack-peek STACK [INDEX])
```

Returns the element in `STACK` at `INDEX`. Index must be 0 <= & <= (stack-depth), defaults to 0 (top of stack).

`stack-poke!` [macro]

```
(stack-poke! STACK VALUE [INDEX])
```

Changes the `STACK` element at `INDEX` to `VALUE`. Returns the modified stack. The stack is modified in place. Index must be 0 <= & <= (stack-depth), defaults to 0 (top of stack)

`stack-push!` [macro]

```
(stack-push! STACK VALUE ...)
```

Pushes `VALUE ...` onto the `STACK`. Returns the modified stack. The stack is modified in place.

`stack-pop*!` [macro]

```
(stack-pop*! STACK)
```

Removes the top element from the `STACK` and returns it. The stack is modified in place. Does not check for the stack-empty condition!

`stack-pop!` [procedure]

```
(stack-pop! STACK)
```

Removes the top element from the `STACK` and returns it. The stack is modified in place.

`stack-cut!` [procedure]

```
(stack-cut! STACK START-DEPTH [END-DEPTH])
```

Removes the `STACK` elements from `START-DEPTH` thru `END-DEPTH` and returns a list of the stack elements. The stack is modified in place. The start-depth must be 0 <= & <= (stack-depth). The end-depth must be start-depth <= & <= (stack-depth), defaults to start-depth.

`list->stack` [macro]

```
(list->stack LIST)
```

Returns the `LIST` as a stack. The resulting stack may share memory with the list and the list should not be modified after this operation.

`stack->list` [macro]

```
(stack->list STACK)
```

Returns the `STACK` as a list, where the first element of the list is the top element of the stack. The resulting list may share memory with the stack object and should not be modified.

`stack-for-each` [macro]

```
(stack-for-each STACK PROCEDURE)
```

Applies the `PROCEDURE` to each element of the `STACK`, in order of top to bottom.

`print-stack` [macro]

```
(print-stack STACK)
```

Prints the elements of the `STACK` to the (current-output-port).

3 License

Index