# Package: rstackdeque (via r-universe)

August 31, 2024

**Type** Package

**Title** Persistent Fast Amortized Stack and Queue Data Structures

**Version** 1.1.1

**Date** 2014-12-01

**URL** https://github.com/oneilsh/rstackdeque

**BugReports** https://github.com/oneilsh/rstackdeque/issues

**Author** Shawn T. O'Neil

**Maintainer** Shawn T. O'Neil <shawn.oneil@cgrb.oregonstate.edu>

**Description** Provides fast, persistent (side-effect-free) stack, queue and deque (double-ended-queue) data structures. While deques include a superset of functionality provided by queues, in these implementations queues are more efficient in some specialized situations. See the documentation for rstack, rdeque, and rpqueue for details.

**License** MIT + file LICENSE

**Suggests** testthat

**Depends** utils

**Repository** https://oneilsh.r-universe.dev

**RemoteUrl** https://github.com/oneilsh/rstackdeque

**RemoteRef** HEAD

**RemoteSha** c3215330a0c1d24db4c5b17ea598b5da0b46a291

## Contents

---

`as.data.frame.rdeque` *Convert an rdeque to a data.frame*

---

## Description

Converts the elements of an rdeque into rows of a data.frame, if this is reasonable.

## Usage

```
## S3 method for class 'rdeque'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `x` | rdeque to convert. |
| `row.names` | passed on to `as.data.frame` before final conversion. |
| `optional` | passed onto `as.data.frame` before final conversion. |
| `...` | passed onto `as.data.frame` before final conversion. |

## Details

This function runs in $O(N)$ time in the size of the rdeque, and will only work if all elements of the deque have the same `length()` (e.g., same number of columns), and if any of the elements have names, then those names do not conflict (e.g., same column names where used). This is accomplished by a call to `do.call("rbind", as.list.rdeque(x))`, where `as.list.rdeque` converts the rdeque to a list where the front element becomes the first element of the list.

## Value

a data.frame with the first row the previous front of the deque and last row the previous back.

## See Also

`as.list.rdeque` for conversion to a list and the generic `as.data.frame`.

## Examples

```
d <- rdeque()
d <- insert_front(d, data.frame(names = c("Bob", "Joe"), ages = c(25, 18)))
d <- insert_front(d, data.frame(names = c("Mary", "Kate", "Ashley"), ages = c(27, 26, 21)))
print(d)

dd <- as.data.frame(d)
print(dd)

## Elements may be similarly-named lists as well, representing individual rows:
d <- rdeque()
d <- insert_front(d, list(name = "Bob", age = 25))
d <- insert_front(d, list(name = "Mary", age = 24))
print(d)

dd <- as.data.frame(d)
print(dd)

## Building a deque in a loop, converting to a dataframe after the fact:
d <- rdeque()
for(i in 1:1000) {
 if(runif(1,0,1) < 0.5) {
   d <- insert_front(d, data.frame(i = i, type = "sqrt", val = sqrt(i)))
 } else {
   d <- insert_back(d, data.frame(i = i, type = "log", val = log(i)))
 }
 if(i %% 100 == 0) {
   print(i/1000)
 }
}
print(head(as.data.frame(d)))
```

---

# as.data.frame.rstack *Convert an rstack to a data.frame*

---

## Description

Converts the elements of an rstack into rows of a data.frame, if this is reasonable.

## Usage

```
## S3 method for class 'rstack'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

## Arguments

| | |
|---|---|
| x | rstack to convert. |
| row.names | passed on to as.data.frame before final conversion. |
| optional | passed onto as.data.frame before final conversion. |
| ... | passed onto as.data.frame before final conversion. |

## Details

This function runs in $O(N)$ time in the size of the rstack, and will only work if all elements of the stack have the same length() (e.g., same number of columns), and if any of the elements have names, then those names do not conflict (e.g., same column names where used). This is accomplished by a call to `do.call("rbind", as.list.rstack(x))`, where `as.list.rstack` converts the rstack to a list where the top element becomes the first element of the list.

## Value

a data.frame with the first row the previous top of the stack.

## See Also

`as.list.rstack` for conversion to a list and the generic `as.data.frame`.

## Examples

```
s <- rstack()
s <- insert_top(s, data.frame(names = c("Bob", "Joe"), ages = c(25, 18)))
s <- insert_top(s, data.frame(names = c("Mary", "Kate", "Ashley"), ages = c(27, 26, 21)))
print(s)

sd <- as.data.frame(s)
print(sd)

## Elements may be similarly-named lists as well, representing individual rows:
s <- rstack()
s <- insert_top(s, list(name = "Bob", age = 25))
s <- insert_top(s, list(name = "Mary", age = 24))
print(s)

sd <- as.data.frame(s)
print(sd)

## Building a stack in a loop, converting to a dataframe after the fact:
s <- rstack()
for(i in 1:1000) {
 if(runif(1,0,1) < 0.5) {
```

```
    s <- insert_top(s, data.frame(i = i, type = "sqrt", val = sqrt(i)))
  } else {
    s <- insert_top(s, data.frame(i = i, type = "log", val = log(i)))
  }
  if(i %% 100 == 0) {
    print(i/1000)
  }
}
print(head(as.data.frame(s)))
```

---

`as.list.rdeque` *Convert an rdeque to a list*

---

## Description

Converts an rdeque to a list, where the front of the deque becomes the first element of the list, the second-from-front the second, and so on.

## Usage

```
## S3 method for class 'rdeque'
as.list(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | rdeque to convert. |
| `...` | additional arguments passed to as.list after initial conversion to list. |

## Details

Runs in $O(N)$ time in the size of the rdeque, but the generated list is pre-allocated for efficiency.

## Value

a list containing the elements of the rdeqeue in front-to-back order.

## See Also

`as.data.frame.rstack` and the generic `as.list`.

## Examples

```
d <- rdeque()
d <- insert_front(d, "a")
d <- insert_front(d, "b")

dlist <- as.list(d)
print(dlist)
```

`as.list.rstack` *Convert an rstack to a list*

## Description

Converts an rstack to a list, where the top of the stack becomes the first element of the list, the second-from-top the second, and so on.

## Usage

```
## S3 method for class 'rstack'
as.list(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | rstack to convert. |
| `...` | additional arguments passed to as.list after initial conversion to list. |

## Details

Runs in $O(N)$ time in the size of the stack, but the generated list is pre-allocated for efficiency.

## Value

a list containing the elements of the stack in top-to-bottom order.

## See Also

`as.data.frame.rstack`

## Examples

```
s <- rstack()
s <- insert_top(s, "a")
s <- insert_top(s, "b")

slist <- as.list(s)
print(slist)
```

`as.rdeque` *Create a pre-filled rdeque from a given input*

## Description

Creates a new rdeque from a given input. Coerces input to a list first using `as.list`, the element at `x[[1]]` becomes the front of the new rdeque.

## Usage

```
as.rdeque(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | input to convert to an rdeque. |
| `...` | additional arguments to be passed to or from methods (ignored). |

## Details

Runs in $O(N)$ in the size of the input. Because data.frames return a list of columns when run through `as.list`, running `as.rdeque` results in a deque of columns, rather than a deque of rows.

## Value

a new rdeque.

## See Also

`rdeque`.

## Examples

```
d <- as.rdeque(1:20)
print(d)

d <- as.rdeque(1:200000)
print(d)

## A deck with only 5 elements, one for each column
oops <- as.rdeque(iris)
print(oops)
```

`as.rstack` *Create an rstack pre-filled from a given input*

## Description

Creates a new rstack from a given input. Coerces input to a list first using `as.list`, the element at `x[[1]]` becomes the top of the new rstack.

## Usage

```
as.rstack(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | input to convert to a stack. |
| `...` | additional arguments to be passed to or from methods. |

## Details

Runs in $O(N)$ in the size of the input. Because data frames return a list of columns when run through `as.list`, running `as.rstack` results in a stack of columns, rather than a stack of rows.

## Value

a new rstack.

## See Also

`rstack`.

## Examples

```
s <- as.rstack(1:20)
print(s)

s <- as.rstack(1:200000)
print(s)

## A stack with only 5 elements, one for each column
oops <- as.rstack(iris)
print(oops)
```

`head.rstack` *Return the head (top) of an rstack*

## Description

Returns the top $n$ elements of an rstack as an stack, or all of the elements if `length(x) < n`.

## Usage

```
## S3 method for class 'rstack'
head(x, n = 6L, ...)
```

## Arguments

| | |
|---|---|
| `x` | rstack to get the head/top of. |
| `n` | number of elements to get. |
| `...` | arguments to be passed to or from other methods (ignored). |

## Details

Runs in $O(n)$ time (in the size of the number of elements requested).

## Value

an `rstack`.

## See Also

`rstack`.

## Examples

```
s <- rstack()
s <- insert_top(s, "a")
s <- insert_top(s, "b")
s <- insert_top(s, "c")

st <- head(s, n = 2)
print(st)
print(s)
```

`insert_back` *Insert an element into the back of an rdeque or rpqueue*

## Description

Returns a version of the deque/queue with the new element in the back position.

## Usage

```
insert_back(x, e, ...)
```

## Arguments

| | |
|---|---|
| `x` | rdeque or rpqueue to insert onto. |
| `e` | element to insert. |
| `...` | additional arguments to be passed to or from methods (ignored). |

## Details

Runs in $O(1)$ time worst-case. Does not modify the original.

## Value

modified version of the rdeque or rpqueue.

## References

Okasaki, Chris. Purely Functional Data Structures. Cambridge University Press, 1999.

## Examples

```
d <- rdeque()
d <- insert_back(d, "a")
d <- insert_back(d, "b")
print(d)

d2 <- insert_back(d, "c")
print(d2)
print(d)
```

`insert_front` *Insert an element into the front of an rdeque*

## Description

Returns a version of the deque with the new element in the front position.

## Usage

```
insert_front(d, e, ...)
```

## Arguments

| | |
|---|---|
| `d` | rdeque to insert onto. |
| `e` | element to insert. |
| `...` | additional arguments to be passed to or from methods (ignored). |

## Details

Runs in $O(1)$ time worst-case. Does not modify the original rdeque.

## Value

modified version of the rdeque.

## References

Okasaki, Chris. Purely Functional Data Structures. Cambridge University Press, 1999.

## See Also

`without_front` for removing the front element.

## Examples

```
d <- rdeque()
d <- insert_front(d, "a")
d <- insert_front(d, "b")
print(d)

d2 <- insert_front(d, "c")
print(d2)
print(d)
```

`insert_top` *Insert an element onto the top of an rstack*

## Description

Insert an element onto the top of an rstack.

## Usage

```
insert_top(s, e, ...)
```

## Arguments

| | |
|---|---|
| `s` | rstack to insert onto. |
| `e` | element to insert. |
| `...` | additional arguments to be passed to or from methods (ignored). |

## Details

Runs in $O(1)$ time worst-case. Does not semantically modify the original structure (i.e, this function is "pure").

## Value

modified version of the stack with new element at top.

## References

Okasaki, Chris. Purely Functional Data Structures. Cambridge University Press, 1999.

## See Also

`rstack` for information on rstacks, `without_top` for removal of top elements.

## Examples

```
s <- rstack()
s <- insert_top(s, "a")
s <- insert_top(s, "b")
print(s)

s2 <- insert_top(s, "c")
print(s2)
print(s)
```

## length.rdeque *Return the number of elements in an rdeque*

### Description

Returns the number of elements in an rdeque.

### Usage

```
## S3 method for class 'rdeque'
length(x)
```

### Arguments

| | |
|---|---|
| x | rdeque to get the length of. |

### Details

Runs in $O(1)$ time, as this information is stored seperately and updated on every insert/remove.

### Value

a vector of length 1 with the number of elements.

### See Also

empty for checking whether an rdeque is empty.

### Examples

```
d <- rdeque()
d <- insert_front(d, "a")
print(length(d))         # 1
d <- insert_back(d, "b")
print(length(d))         # 2
```

## length.rstack *Return the number of elements in an rstack*

### Description

Returns the number of elements in an rstack.

### Usage

```
## S3 method for class 'rstack'
length(x)
```

### Arguments

| | |
|---|---|
| x | rstack to get the length of. |

### Details

Runs in $O(1)$ time, as this information is stored seperately and updated on every insert/remove.

### Value

a vector of length 1, which the number of elements of the stack.

### See Also

empty for checking whether an rstack is empty.

### Examples

```
s <- rstack()
s <- insert_top(s, "a")
print(length(s))         # 1
s <- insert_top(s, "b")
print(length(s))         # 2
```

---

## peek_back *Return the data element at the back of an rdeque*

---

### Description

Simply returns the data element sitting at the back of the rdeque, leaving the rdeque alone.

### Usage

```
peek_back(d, ...)
```

### Arguments

| | |
|---|---|
| d | rdeque to peek at. |
| ... | additional arguments to be passed to or from methods (ignored). |

### Details

Runs in O(1) worst-case time.

### Value

data element existing at the back of the rdeque.

## See Also

without_back for removing the front element.

## Examples

```
d <- rdeque()
d <- insert_front(d, "a")
d <- insert_front(d, "b")
e <- peek_back(d)
print(e)
print(d)

## Assigning to the front data element with peek_front:
d <- rdeque()
d <- insert_front(d, data.frame(a = 1, b = 1))
d <- insert_front(d, data.frame(a = 1, b = 1))

peek_back(d)$a <- 100
print(d)

peek_back(d) <- data.frame(a = 100, b = 100)
print(d)
```

---

# peek_front *Return the data element at the front of an rdeque*

---

## Description

Simply returns the data element sitting at the front of the deque, leaving the deque alone.

## Usage

```
peek_front(x, ...)
```

## Arguments

| | |
|---|---|
| x | rdeque to look at. |
| ... | additional arguments to be passed to or from methods (ignored). |

## Details

Runs in $O(1)$ worst-case time.

## Value

data element at the front of the rdeque.

## Examples

```
d <- rdeque()
d <- insert_front(d, "a")
d <- insert_back(d, "b")
e <- peek_front(d)
print(e)
print(d)
```

---

peek_top *Return the data element at the top of an rstack*

---

## Description

Simply returns the data element sitting at the top of the rstack, leaving the rstack alone.

## Usage

```
peek_top(s, ...)
```

## Arguments

| | |
|---|---|
| s | rstack to peek at. |
| ... | additional arguments to be passed to or from methods (ignored). |

## Details

Runs in O(1) worst-case time.

## Value

data element existing at the top of the rstack.

## See Also

without_top for removing the top element.

## Examples

```
s <- rstack()
s <- insert_top(s, "a")
s <- insert_top(s, "b")
e <- peek_top(s)
print(e)
print(s)

## Assigning to the top data element with peek_top:
s <- rstack()
s <- insert_top(s, data.frame(a = 1, b = 1))
s <- insert_top(s, data.frame(a = 1, b = 1))
```

```
peek_top(s)$a <- 100
print(s)

peek_top(s) <- data.frame(a = 100, b = 100)
```

---

`rdeque` *Create a new empty rdeque*

---

## Description

Creates a new, empty, rdeque ready for use.

## Usage

```
rdeque()
```

## Details

An rdeque provided both "Last In, First Out" (LIFO) and "First In, First Out" (FIFO) access; envisaged as queue, elements may be added or removed from the front or the back with `insert_front`, `insert_back`, `without_front`, and `without_back`. The front and back elements may be retrieved with `peek_front` and `peek_back`.

Internally, rdeques are stored as a pair of rstacks; they provide $O(1)$-amortized insertion and removal, so long as they are not used persistently (that is, the variable storing the deque is always replaced by the modified version, e.g. `s <- without_front(s)`). When used persistently (e.g. `s2 <- without_front(s)`, which results in two deques being accessible), this cannot be gauranteed. When an $O(1)$ worst-cast, fully persistent FIFO queue is needed, the rpqueue from this package provides these semantics. However, there is a constant-time overhead for rpqueues, such that rdeques are often more efficient (and flexible) in practice, even in when used persistently.

Other useful functions include `as.list` and `as.data.frame` (the latter of which requires that all elements can be appended to become rows of a data frame in a reasonable manner).

## Value

a new empty rdeque.

## References

Okasaki, Chris. Purely Functional Data Structures. Cambridge University Press, 1999.

## See Also

`rstack` for a fast LIFO-only structure.

## Examples

```
d <- rdeque()
d <- insert_front(d, "a")
d <- insert_front(d, "b")
d <- insert_back(d, "c")
d <- insert_back(d, "d")
print(d)

d2 <- without_back(d)
print(d2)
print(d)

b <- peek_front(d)
print(b)
```

---

`rev.rstack` *Reverse an rstack*

---

## Description

Returns a reversed version of an rstack, where the old last element (generally inaccessible) is now the top (and thus now accessible).

## Usage

```
## S3 method for class 'rstack'
rev(x)
```

## Arguments

| | |
|---|---|
| `x` | rstack to reverse. |

## Details

This method runs in $O(N)$ in the size of the rstack, though it works behind-the-scenes for efficiency by converting the input stack to a list, reversing the list, and building the result as a new rstack. The original is thus left alone, preserving $O(1)$ amortized time for the original (assuming the "cost" of reversing is charged to the newly created stack) at the cost of additional memory usage. But, if the stack is not being used in a preserved manner, e.g. `s <- rev(s)`, the garbage collector will be free to clean up the original data if it is no longer usable.

## Value

a reversed version of the rstack.

## See Also

`as.list.rstack` for converting an rstack to a list.

## Examples

```
s <- rstack()
s <- insert_top(s, "a")
s <- insert_top(s, "b")
s <- insert_top(s, "c")

r <- rev(s)
print(r)
print(s)
```

---

`rstack` *Create a new, empty rstack*

---

## Description

An rstack is a "Last In, First Out" (LIFO) structure imagined as being organized from top (last in) to bottom (first in), supporting efficient insertion into the top, removal from the top, and peeking/accessing the top element. All functions supported by rstacks are side-effect free.

## Usage

```
rstack()
```

## Details

Other handy functions supported by rstacks include `as.list` and `as.data.frame` (the latter of which requires that all elements can be appended to become rows of a data frame in a reasonable manner). Operations are amortized $O(1)$.

The rstack class also supports `rev` - this operation is $O(N)$, and results in a copy. This means previous versions will retain their $O(1)$ amortized nature (if we assume the cost of the reverse is charged to the newly created stack), at the cost of memory usage. However, this also means that if stacks are used in a non-persistent way, e.g. `s <- rev(s)`, then the garbage collector is free to clean up old versions of the data.

## Value

an empty rstack.

## References

Okasaki, Chris. Purely Functional Data Structures. Cambridge University Press, 1999.

## See Also

`insert_top` for insertion, `without_top` for removal, and `peek_top` for peeking.

## Examples

```
s <- rstack()
s <- insert_top(s, "a")
s <- insert_top(s, "b")
print(s)

sl <- without_top(s)
print(sl)
print(s)

b <- peek_top(s)
print(b)
```

---

`rstacknode` *Internal structure used by rstacks, rdeques, and rpqueues*

---

## Description

For use by rstacks and rdeques. Simply an environment with no parent, reference for the data and the next node.

## Usage

```
rstacknode(data)
```

## Arguments

| | |
|---|---|
| `data` | data to reference with this node. |

## Value

an environment.

---

`without_back` *Return a version of an rdeque without the back element*

---

## Description

Simply returns a version of the given rdeque without the back element The original rdeque is left alone.

## Usage

```
without_back(d, ...)
```

### Arguments

| | |
|---|---|
| d | rdeque to remove elements from. |
| ... | additional arguments to be passed to or from methods (ignored). |

### Details

Runs in $O(1)$-amortized time if the rdeque is used non-persistently (see documentation of rdeque for details). If the given rdeque is empty, an error will be generated.

### Value

version of the rdeque with the back element removed.

### References

Okasaki, Chris. Purely Functional Data Structures. Cambridge University Press, 1999.

### See Also

insert_back for inserting elements.

### Examples

```
d <- rdeque()
d <- insert_front(d, "a")
d <- insert_front(d, "b")
d <- insert_front(d, "c")

d2 <- without_back(d)
print(d2)

d3 <- without_back(d)
print(d3)

print(d)
```

---

## without_front *Return a version of an rdeque or rpqueue without the front element*

---

### Description

Return a version of an rdeque or rpqueue without the front element

### Usage

```
without_front(x, ...)
```

### Arguments

| | |
|---|---|
| x | rdeque or rpqueue to remove elements from. |
| ... | additional arguments to be passed to or from methods (ignored). |

### Details

Simply returns a version of the given structure without the front element. The original is left alone.

### Value

a version of the rdeque or rpqueue with the front element removed.

### References

Okasaki, Chris. Purely Functional Data Structures. Cambridge University Press, 1999.

### Examples

```
d <- rdeque()
d <- insert_front(d, "a")
d <- insert_front(d, "b")
d <- insert_back(d, "c")

d2 <- without_front(d)
print(d2)

d3 <- without_front(d2)
print(d3)

print(d)
```

---

## without_top *Return a version of an rstack without the top element*

---

### Description

Simply returns a version of the given stack without the top element. Results in an error if the structure is empty. The original rstack is left alone.

### Usage

```
without_top(s, ...)
```

### Arguments

| | |
|---|---|
| s | rstack to remove elements from. |
| ... | additional arguments to be passed to or from methods (ignored). |

## Details

Runs in $O(1)$ time worst case.

## Value

version of the stack with the top $n$ elements removed.

## References

Okasaki, Chris. Purely Functional Data Structures. Cambridge University Press, 1999.

## See Also

insert_top for inserting elements.

## Examples

```
s <- rstack()
s <- insert_top(s, "a")
s <- insert_top(s, "b")
s <- insert_top(s, "c")

s2 <- without_top(s)
print(s2)

print(s)
```

# Index