Package 'rstackdeque'

July 23, 2025

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

NeedsCompilation no

Repository CRAN

Date/Publication 2015-04-13 22:27:44

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.rpqueue` *Convert an rpqueue to a data.frame*

Description

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

Usage

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

Arguments

`x`	rpqueue 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 rpqueue, and will only work if all elements of the queue 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.rpqueue(x))`, where `as.list.rpqueue` converts the rpqueue 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 queue and last row the previous back.

See Also

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

Examples

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

qq <- as.data.frame(q)
print(qq)

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

qq <- as.data.frame(q)
print(qq)

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

`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.rpqueue` *Convert an rpqueue to a list*

Description

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

Usage

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

Arguments

`x`	rpqueue to convert.
`...`	additional arguments passed to as.list after initial conversion to list.

Details

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

Value

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

See Also

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

Examples

```
q <- rpqueue()
q <- insert_back(q, "a")
q <- insert_back(q, "b")

qlist <- as.list(q)
print(qlist)
```

`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.rdeque.default *Default method for converting to an rdeque*

Description

Default method for converting to an rdeque.

Usage

```
## Default S3 method:
as.rdeque(x, ...)
```

Arguments

x	the input to convert to an rdeque
...	arguments to be passed to or from other methods (ignored).

Details

Elements from the input (of any type) are first converted to a list with `as.list`, after this an rdeque of the appropriate size is created holding the elements. The element at `x[[1]]` becomes the front of the rdeque. Runs in time $O(n)$, in the size of the number of elements contained in the resulting rdeque.

Value

a filled `rdeque`.

See Also

`rdeque` for info about rdeques, `as.rdeque` for the generic function.

as.rpqueue *Create a pre-filled rpqueue from a given input*

Description

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

Usage

```
as.rpqueue(x, ...)
```

Arguments

x	input to convert to an rpqueue.
...	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.rpqueue` results in a queue of columns, rather than a queue of rows.

Value

a new rpqueue.

See Also

`rpqueue`.

Examples

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

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

---

as.rpqueue.default *Default method for converting to an rpqueue*

Description

Default method for converting to an rpqueue.

Usage

```
## Default S3 method:
as.rpqueue(x, ...)
```

Arguments

x	the input to convert to an rpqueue.
...	arguments to be passed to or from other methods (ignored).

Details

Elements from the input (of any type) are first converted to a list with `as.list`, after this an rpqueue of the appropriate size is created holding the elements. The element at `x[[1]]` becomes the front of the rpqueue. Runs in time $O(n)$.

Value

a filled rpqueue.

See Also

rpqueue for info about rpqueues, as.rpqueue for the generic function.

---

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)
```

as.rstack.default *Default method for converting to an rstack*

Description

Default method for converting to an rstack.

Usage

```
## Default S3 method:
as.rstack(x, ...)
```

Arguments

x	the input to convert to an rstack.
...	arguments to be passed to or from other methods (ignored).

Details

Elements from the input (of any type) are first converted to a list with `as.list`, after this an rstack of the appropriate size is created holding the elements. The element at `x[[1]]` becomes the top of the stack.

Value

a filled `rstack`.

See Also

`rstack` for info about rstacks, `as.rstack` for the generic.

empty *Check if an rstack, rdeque, or rpqueue is empty*

Description

Check if an rstack, rdeque, or rpqueue is empty.

Usage

```
empty(x, ...)
```

Arguments

x	rstack, rdeque, or rpqueue to check.
...	additional arguments to be passed to or from methods (ignored).

Details

Runs in $O(1)$ time for all types.

Value

logical vector of length 1.

Examples

```
s <- rstack()
print(empty(s))        ## TRUE
s <- insert_top(s, "a")
print(empty(s))        ## FALSE
```

---

empty.rdeque *Check if an rdeque is empty*

Description

Check if an rdeque is empty.

Usage

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

Arguments

x	the rdeque to check.
...	additional arguments to be passed to or from methods (ignored).

Details

Runs in $O(1)$ time.

Value

logical vector of length 1.

See Also

empty for the generic function that can be used on rstacks, rdeques, and rpqueues.

empty.rpqueue *Check if an rpqueue is empty*

Description

Check if an rpqueue is empty.

Usage

```
## S3 method for class 'rpqueue'
empty(x, ...)
```

Arguments

x	the rpqueue to check.
...	additional arguments to be passed to or from methods (ignored).

Details

Runs in $O(1)$ time.

Value

logical vector of length 1.

See Also

empty for the generic function that can be used on rstacks, rdeques, and rpqueues.

empty.rstack *Check if an rstack is empty*

Description

Check if an rstack is empty.

Usage

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

Arguments

x	the rstack to check.
...	additional arguments to be passed to or from methods (ignored).

Details

Runs in $O(1)$ time.

Value

logical vector of length 1.

See Also

`empty` for the generic function that can be used on rstacks, rdeques, and rpqueues.

`fixd` *Fix an rdeque*

Description

Maintains the invariant that there is always something in two stacks used by rdeques under the hood so long as there is 2 more elements in the rdeque.

Usage

```
fixd(d, ...)
```

Arguments

`d`	rdeque to fix.
`...`	additional arguments to be passed to or from methods (ignored).

Details

In fact, fix will be called whenever there are fewer than 6 elements in both the front and end of the deque. Generally this method is $O(N)$, and so a full copy is returned.

Value

fixed, "balanced" deque.

References

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

fixd.rdeque *Fix an rdeque*

Description

A method used behind the scenes to provide $O(1)$-amortized time for most operations. Runs in $O(n)$ time worst case; restructures the rdeque so that the two internal rstacks are roughly the same length.

Usage

```
## S3 method for class 'rdeque'
fixd(d, ...)
```

Arguments

d	The rdeque to fix.
...	additional arguments to be passed to or from methods.

Value

a fixed deque.

References

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

head.rdeque *Return the first n elements of an rdeque as an rdeque*

Description

Returns the first n elements of a deque as a deque, or all of the elements if its length is less than n.

Usage

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

Arguments

x	rdeque to get the head 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

a new rdeque.

Examples

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

dt <- head(d, n = 2)
print(dt)
```

---

head.rpqueue *Return the head (front) of an rpqueue*

Description

Returns the first n elements of an rpqueue as an rpqueue, or all of the elements if `length(x) < n`.

Usage

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

Arguments

`x`	rpqueue 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 `rpqueue`.

See Also

`rpqueue`.

Examples

```
q <- rpqueue()
q <- insert_back(q, "a")
q <- insert_back(q, "b")
q <- insert_back(q, "c")

qt <- head(q, n = 2)
print(qt)
```

`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_back.rdeque` *Insert an element into the back of an rdeque*

Description

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

Usage

```
## S3 method for class 'rdeque'
insert_back(x, e, ...)
```

Arguments

`x`	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.

Value

modified version of the rdeque.

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_back.rpqueue` *Insert an element into the back of an rpqueue*

Description

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

Usage

```
## S3 method for class 'rpqueue'
insert_back(x, e, ...)
```

Arguments

`x`	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 rpqueue.

References

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

Examples

```
q <- rpqueue()
q <- insert_back(q, "a")
q <- insert_back(q, "b")
print(q)

q2 <- insert_back(q, "c")
print(q2)
print(q)
```

`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_front.rdeque` *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

```
## S3 method for class 'rdeque'
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)
```

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

Description

Insert an element onto the top of an rstack.

Usage

```
## S3 method for class 'rstack'
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.rpqueue *Return the number of elements in an rpqueue*

Description

Returns the number of elements in an rpqueue.

Usage

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

Arguments

x	rpqueue 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 rpqueue is empty.

Examples

```
q <- rpqueue()
q <- insert_back(q, "a")
print(length(q))         # 1
q <- insert_back(q, "b")
print(length(q))         # 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
```

---

makeequal *Generic maintenance function for rpqueues*

Description

Generic maintenance function for rpqueues, called automatically when needed by other functions.

Usage

```
makeequal(rpqueue, ...)
```

Arguments

rpqueue	rpqueue to makeequal.
...	additional arguments to be passed to or from methods (ignored).

Details

See *Simple and Efficient Purely Functional Queues and Deques*, Okasaki 1995, J. Functional Programming, 5(4) 583 to 592 for information.

Value

a "fixed" rpqueue.

References

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

See Also

rotate helper function that calls this one.

makeequal.rpqueue *Maintenance function for rpqueues*

Description

Maintenance function for rpqueues, called automatically when needed by other functions.

Usage

```
## S3 method for class 'rpqueue'
makeequal(rpqueue, ...)
```

Arguments

rpqueue	rpqueue to makeequal.
...	additional arguments to be passed to or from methods (ignored).

Details

See *Simple and Efficient Purely Functional Queues and Deques*, Okasaki 1995, J. Functional Programming, 5(4) 583 to 592 for information.

Value

a "fixed" rpqueue.

References

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

See Also

rotate helper function that calls this one.

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_back.rdeque *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

```
## S3 method for class 'rdeque'
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_back<- *Assign to/modify the back of an rdeque*

Description

Allows modification access to the back of a deque.

Usage

```
peek_back(d, ...) <- value
```

Arguments

d	rdeque to modify the back element of.
...	additional arguments to be passed to or from methods.
value	value to assign to the back data element.

Details

Runs in $O(1)$ worst case time. Throws an error if the deque is empty.

Value

modified rdeque.

See Also

peek_back.rdeque for accessing the back element.

Examples

```
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)
```

---

peek_back<-.rdeque *Assign to/modify the back of an rdeque*

Description

Allows modification access to the back of a deque.

Usage

```
## S3 replacement method for class 'rdeque'
peek_back(d, ...) <- value
```

Arguments

d	rdeque to modify the back element of.
...	additional arguments to be passed to or from methods.
value	value to assign to the back data element.

Details

Runs in $O(1)$ worst case time. Throws an error if the deque is empty.

Value

modified rdeque.

See Also

peek_back.rdeque for accessing the back element.

Examples

```
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)
```

---

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_front.rdeque *Return the data element at the front of an rdeque*

Description

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

Usage

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

Arguments

x	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 front of the rdeque.

See Also

without_front for removing the front element.

Examples

```
d <- rdeque()
d <- insert_front(d, "a")
d <- insert_front(d, "b")
e <- peek_front(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_front(d)$a <- 100
print(d)

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

`peek_front.rpqueue` *Return the data element at the front of an rpqueue*

Description

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

Usage

```
## S3 method for class 'rpqueue'
peek_front(x, ...)
```

Arguments

`x`	rpqueue 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 front of the queue.

See Also

`without_front` for removing the front element.

Examples

```
q <- rpqueue()
q <- insert_back(q, "a")
q <- insert_back(q, "b")
e <- peek_front(q)
print(e)
print(q)

## Assigning to the front data element with peek_front:
q <- rpqueue()
```

```
q <- insert_back(q, data.frame(a = 1, b = 1))
q <- insert_back(q, data.frame(a = 1, b = 1))

peek_front(q)$a <- 100
print(q)

peek_front(q) <- data.frame(a = 100, b = 100)
print(q)
```

---

peek_front<- *Assign to/modify the front of an rdeque or rpqueue*

Description

Allows modification access to the front of a deque or queue.

Usage

```
peek_front(x, ...) <- value
```

Arguments

x	rdeque or rpqueue to modify the front element of.
...	additional arguments to be passed to or from methods.
value	value to assign to the front data element.

Details

Runs in $O(1)$ worst case time. Throws an error if the deque is empty.

Value

modified rdeque or rpqueue.

Examples

```
d <- rdeque()
d <- insert_front(d, data.frame(a = 1, b = 1))
d <- insert_front(d, data.frame(a = 1, b = 1))

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

peek_front(d) <- data.frame(a = 100, b = 100)



q <- rpqueue()
q <- insert_front(d, data.frame(a = 1, b = 1))
```

```
q <- insert_front(d, data.frame(a = 1, b = 1))

peek_front(q)$a <- 100
print(q)

peek_front(q) <- data.frame(a = 100, b = 100)
```

---

peek_front<-.rdeque *Assign to/modify the front of an rdeque*

Description

Allows modification access to the front of a deque.

Usage

```
## S3 replacement method for class 'rdeque'
peek_front(x, ...) <- value
```

Arguments

x	rdeque to modify the front element of.
...	additional arguments to be passed to or from methods (ignored).
value	value to assign to the front data element.

Details

Runs in $O(1)$ worst case time. Throws an error if the rdeque is empty. Modifies the element in place (i.e., is not side-effect-free).

Value

modified rdeque.

See Also

peek_front.rdeque for accessing the front data element.

Examples

```
d <- rdeque()
d <- insert_front(d, data.frame(a = 1, b = 1))
d <- insert_front(d, data.frame(a = 1, b = 1))

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

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

peek_front<-.rpqueue *Assign to/modify the front of an rpqueue*

Description

Allows modification access to the front of a queue.

Usage

```
## S3 replacement method for class 'rpqueue'
peek_front(x, ...) <- value
```

Arguments

x	rpqueue to modify the front element of.
...	additional arguments to be passed to or from methods (ignored).
value	value to assign to the front data element.

Details

Runs in $O(1)$ worst case time. Throws an error if the rpqueue is empty. Modifies the element in place (i.e., is not side-effect-free).

Value

modified rpqueue.

See Also

peek_front.rpqueue for accessing the front data element.

Examples

```
q <- rpqueue()
q <- insert_back(q, data.frame(a = 1, b = 1))
q <- insert_back(q, data.frame(a = 1, b = 1))

peek_front(q)$a <- 100
print(q)

peek_front(q) <- data.frame(a = 100, b = 100)
print(q)
```

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)
```

peek_top.rstack *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

```
## S3 method for class 'rstack'
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)
```

peek_top<- *Assign to/modify the top of an rstack*

Description

Allows modification access to the top of a stack.

Usage

```
peek_top(s, ...) <- value
```

Arguments

s	rstack to modify the first element of.
...	additional arguments to be passed to or from methods (ignored).
value	value to assign to the top data element.

Details

Runs in $O(1)$ worst case time. Throws an error if the rstack is `empty`. Modifies the element in place (i.e., is not side-effect-free).

Value

modified rstack.

See Also

`peek_top` for accessing the top data element.

Examples

```
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)
```

peek_top<-.rstack *Assign to/modify the top of an rstack*

Description

Allows modification access to the top of a stack.

Usage

```
## S3 replacement method for class 'rstack'
peek_top(s, ...) <- value
```

Arguments

s	rstack to modify the first element of.
...	additional arguments to be passed to or from methods (ignored).
value	value to assign to the top data element.

Details

Runs in $O(1)$ worst case time. Throws an error if the rstack is empty. Modifies the element in place (i.e., is not side-effect-free).

Value

modified rstack.

See Also

peek_top for accessing the top data element.

Examples

```
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)
```

print.rdeque *Print an rdeque*

Description

Prints a summary of the contents of an rdeque, including the number of elements and the first and last few.

Usage

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

Arguments

x	the rdeque to print.
...	additional arguments to be passed to or from methods (ignored).

Details

Depending on the internal state of the rdeque, this method is not gauranteed to run in $O(1)$ time.

See Also

as.list.rdeque for converting an rdeque into a list which can then be printed in full.

print.rpqueue *Print an rpqueue*

Description

Prints a summary of the contents of an rpqueue, including the number of elements and the first few.

Usage

```
## S3 method for class 'rpqueue'
print(x, ...)
```

Arguments

x	the rpqueue to print.
...	additional arguments to be passed to or from methods (ignored).

Details

Since only the first few elements are detailed, runs in $O(1)$ time.

See Also

as.list.rpqueue for converting an rpqueue into a list which can then be printed in full.

`print.rstack` *Print an rstack*

Description

Prints a summary of the contents of an rstack, including the number of elements and the top few.

Usage

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

Arguments

x	the rstack to print.
...	additional arguments to be passed to or from methods (ignored).

Details

Since only the top few elements are detailed, runs in $O(1)$ time.

See Also

as.list.rstack for converting an rstack into a list which can then be printed in full.

`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)
```

rotate *Generic maintenance function for rpqueues*

Description

Generic maintenance function for rpqueues, called automatically when needed by other functions.

Usage

```
rotate(rpqueue, acclazylist, ...)
```

Arguments

rpqueue	rpqueue to rotate.
acclazylist	lazy list accumulator.
...	additional arguments to be passed to or from methods (ignored).

Details

See *Simple and Efficient Purely Functional Queues and Deques*, Okasaki 1995, J. Functional Programming, 5(4) 583 to 592 for information.

Value

a fully rotated rpqueue, but with the l list delayed in evaluation.

References

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

See Also

makeequal helper function that calls this one.

rotate.rpqueue *Maintenance function for rpqueues*

Description

Maintenance function for rpqueues, called automatically when needed by other functions.

Usage

```
## S3 method for class 'rpqueue'
rotate(rpqueue, acclazylist, ...)
```

Arguments

rpqueue	rpqueue to rotate.
acclazylist	lazy list accumulator.
...	additional arguments to be passed to or from methods (ignored).

Details

See *Simple and Efficient Purely Functional Queues and Deques*, Okasaki 1995, J. Functional Programming, 5(4) 583 to 592 for information on this function.

Value

a fully rotated rpqueue, but with the l list delayed in evaluation.

References

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

See Also

makeequal helper function that calls this one.

---

rpqueue *Create a new empty rpqueue*

Description

Creates a new, empty, rpqueue ready for use.

Usage

```
rpqueue()
```

Details

An rpqueue provides "First In, First Out" (FIFO) access; envisaged as a queue, elements may be inserted at the back and removed from the front. Unlike rdeque, access is gauranteed $O(1)$ worst case even when used persistently, though in most situations rdeques will be faster in practice (see the documentation for rdeque for details).

Other handy 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 rpqueue.

References

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

See Also

`rstack` for a fast LIFO-only structure.

Examples

```
q <- rpqueue()
q <- insert_back(q, "a")
q <- insert_back(q, "b")
print(q)

q2 <- without_front(q)
print(q2)
print(q)

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

---

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_back.rdeque` *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

```
## S3 method for class 'rdeque'
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_front.rdeque` *Return a version of an rdeque without the front element*

Description

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

Usage

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

Arguments

`x`	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 front element removed.

References

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

See Also

`insert_front` for inserting elements.

Examples

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

d2 <- without_front(d)
print(d2)

d3 <- without_front(d)
print(d3)

print(d)
```

`without_front.rpqueue` *Return a version of an rpqueue without the front element*

Description

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

Usage

```
## S3 method for class 'rpqueue'
without_front(x, ...)
```

Arguments

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

Details

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

Value

version of the rpqueue with the front element removed.

References

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

See Also

`peek_front` for accessing the front element.

Examples

```
q <- rpqueue()
q <- insert_back(q, "a")
q <- insert_back(q, "b")
q <- insert_back(q, "c")

q2 <- without_front(q)
print(q2)

q3 <- without_front(q)
print(q3)

print(q)
```

`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)
```

`without_top.rstack` *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

```
## S3 method for class 'rstack'
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