Stream

Thread Number

1

Thread Number

2

Thread Number

3

Thread Number

4

Thread Number

5

Thread Number

6

Thread State

NEW

Thread State

RUNNABLE

Thread State

BLOCKED

Thread State

WAITING

Thread State

TIMED_WAITING

Thread State

TERMINATED

Task: Increment

Increment the shared value by 1.

```
val++
   1. mv var1 val          [read]
   2. add var1 #1
   3. mv val var1          [write]
```

Task: Add

Roll a die, and increment the shared value by that amount.

```
r = rand(6)
   1. load arg1 #6
   2. call rand
   3. mv var1 ret
val = val + r
   4. mv var2 val          [read]
   5. add var2 var1
   6. mv val var2          [write]
```

Task: Sleep

Roll a die, and wait that number of instructions.

```
r = rand(6)
   1. load arg1 #6
   2. call rand
   3. mv var1 ret
sleep(r)
   4. mv arg1 var1
   5. call sleep
      (→TIMED_WAITING)
```

Task: Conditional

Roll a die. If it is less than the shared value, execute another task from your hand.

```
r = rand(6)
   1. load arg1 #6
   2. call rand
   3. mv var1 ret
if ( r < val ) then another_task
   4. mv var2 val                [read]
   5. sub var2 var1
   6. jgz var2 +2
   7. jmp another_task
   8. nop
```

Task: Lock A

Acquire lock or semaphore A.
This is atomic.

`sem_wait(lock_A)` *[lock]*

(→BLOCKED)

Task: Lock B

Acquire lock or semaphore B.
This is atomic.

```
sem_wait(lock_B)                    [lock]
```
(→BLOCKED)

Task: Unlock A

Release lock or semaphore A. This is atomic.

```
sem_release(lock_A)          [lock]
```

Task: Unlock B

Release lock or semaphore B. This is atomic.

```
sem_release(lock_B)          [lock]
```

Task: Start

Start a new thread.

```
t = new Thread()
```

1. call create_thread
2. mv var1 ret
3. mv arg1 var1
4. call setup_thread_table
5. mv thr ret

```
t.start()
```

6. mv arg1 thr
7. call launch *[launch]*

Task: Wait

Wait on a thread.

`t.wait()`

1. `mv arg1 thr`
2. `call join_thread` *[join]*
   (→WAITING)

Task: Infinite Loop

Begin an infinite loop block.

```
while(true) {
```

Task: Finite Loop

Begin a finite loop block.

```
while(count > 0) {
   1. jzero count loop_end
count--
   2. sub count #1
```

Task: Loop End

End a loop block.

}

```
1. jump loop_begin
```

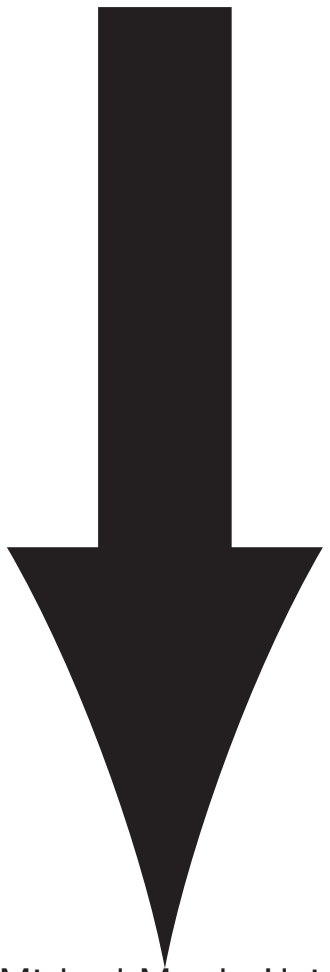
Lock/Semaphore

A

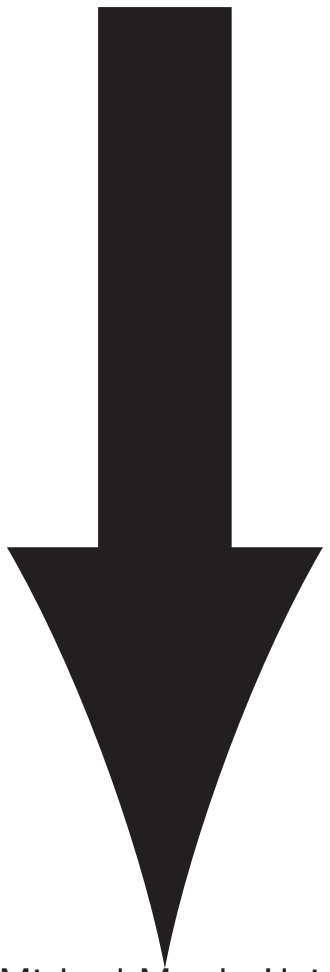
Lock/Semaphore

B

Head Pointer



Tail Pointer



Item

α

Item

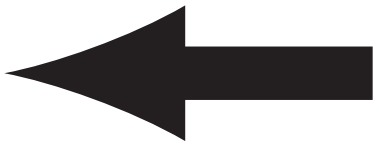
β

Item

γ

Item

δ



Integer

0

Integer

1

Integer

2

Integer

3

Integer

4

Integer

5

Integer

6

Integer

7

Integer

8

Integer

9

Integer

10

Integer

11

Integer

12

Integer

13

Integer

14

Integer

15

Integer

16

Integer

17

Integer

18

Integer

19

Integer

20

Boolean

True

Boolean

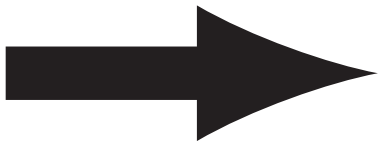
False

Option

Generator Operation

Integer Range

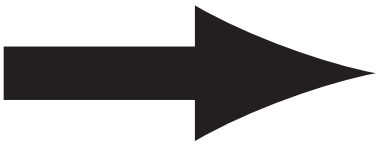
$$a..b$$

next 

Generator Operation

Iterate

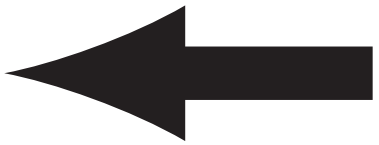
$$a, n \rightarrow n + b$$

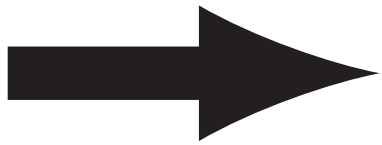
next 

Intermediate Operation

Filter

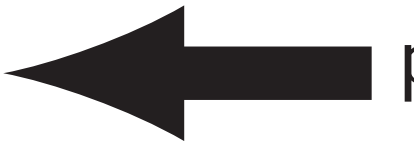
$$x \rightarrow x\%m == 0$$





Intermediate Operation

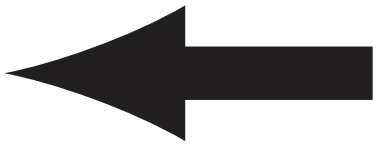
Distinct



Intermediate Operation

Limit

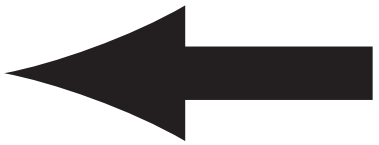
l



Intermediate Operation

Skip

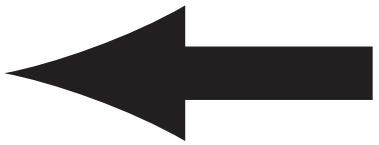
s

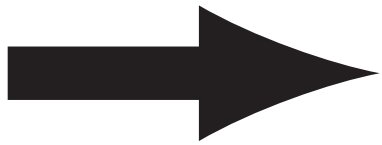


Intermediate Operation

Peek

$$x \rightarrow \text{print}(x)$$

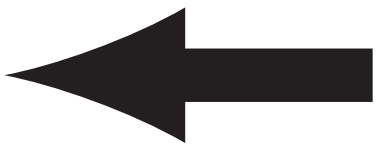


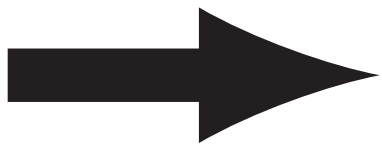


Intermediate Operation

Map

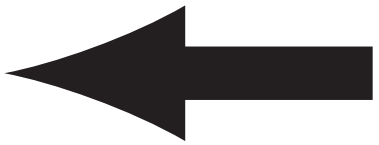
$$x \rightarrow x \times c$$





Terminal Operation

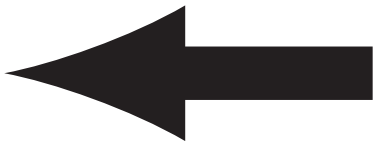
Print



Terminal Operation

AllMatch

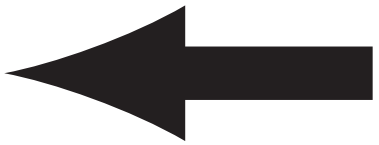
$$x \rightarrow x == p$$



Terminal Operation

AllMatch

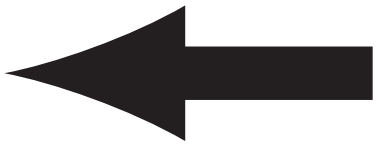
$$x \rightarrow x > p$$



Terminal Operation

AnyMatch

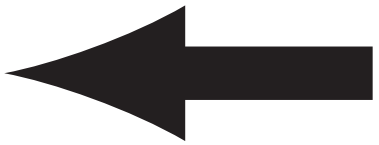
$$x \rightarrow x == p$$



Terminal Operation

AnyMatch

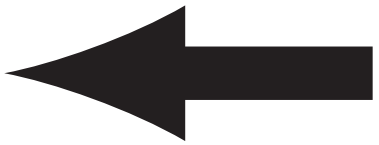
$$x \rightarrow x > p$$



Terminal Operation

NoneMatch

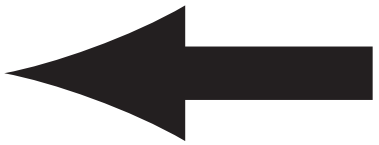
$$x \rightarrow x == p$$



Terminal Operation

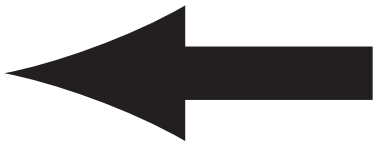
NoneMatch

$$x \rightarrow x > p$$



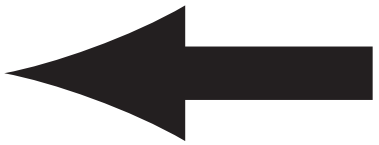
Terminal Operation

FindFirst



Terminal Operation

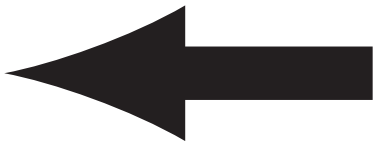
FindAny



Terminal Operation

Reduce

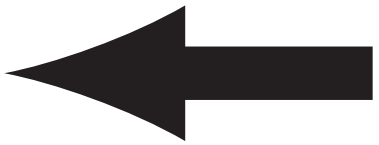
$$0, (a, b) \rightarrow a + b$$



Terminal Operation

Reduce

$$1, (a, b) \rightarrow a \times b$$



Terminal Operation

Collect