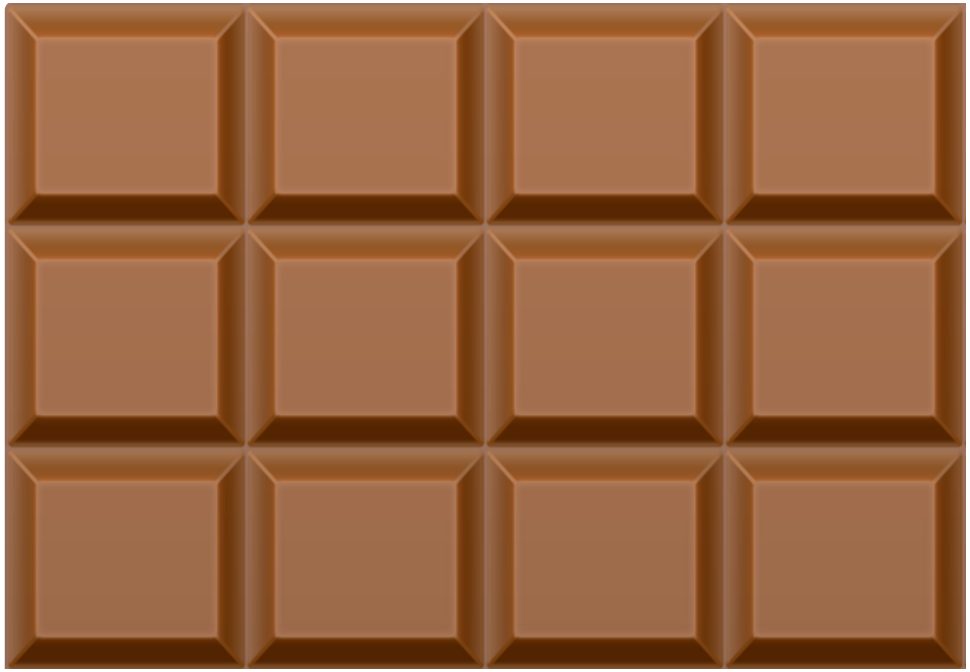
Iterative Programs

R S Milton
Department of CSE, SSN College of Engineering

25 July 2019

Chocolate Bar: Invariant



$3 \times 4 = 12$ squares

iteration	p	c
1	1	0
2	2	1
2	3	2
4	4	3
5	5	4
6	6	5

```
# p - c = k
p, c := p+1, c+1
# p - c = k
```

Loop Invariant

```
while
   # L
   C:
   B
   # L
```

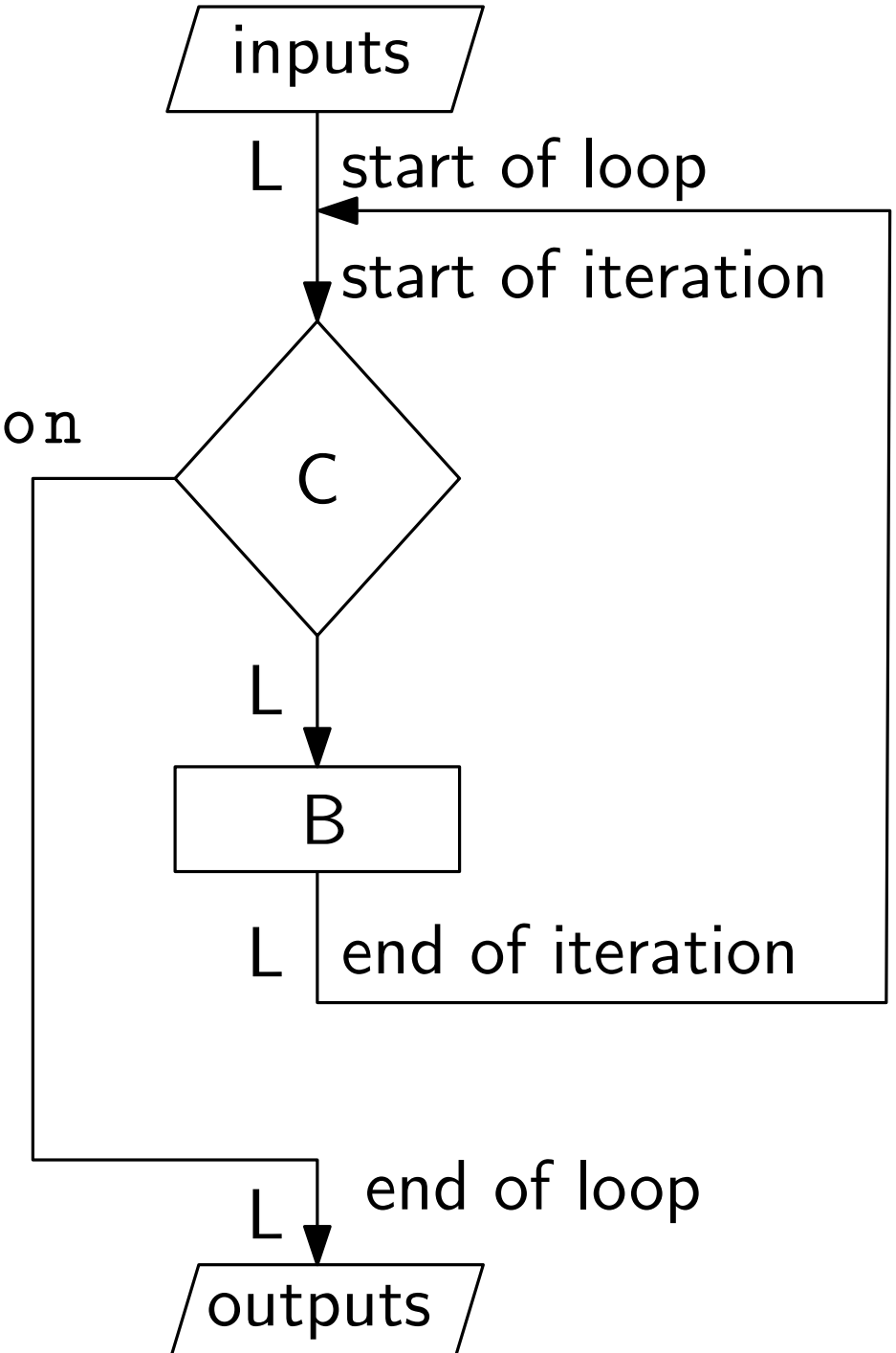
- In each iteration, the loop body changes the state (value of at least one of the variables).
- However, there is a property of the state (variables), which is left unchanged by the loop body.
- The value of this property of the variables is the same before and after the loop body.
- This invariant of the loop body is known as the loop invariant.

Breeaking Chocolate Bar

```
Algorithm BreakChocolateBar
# Precondition
# Initialize
# p - c = k
until Terminal condition
   # p - c = k
   p, c := p+1, c+1
   # p - c = k
# p - c = k
# Postcondition
```

Control Points Where the Loop Invariant is True

```
# L,start of loop
while
   # L,start of iteration
   C:
   B
   # L,end of iteration
# L,end of loop
```



Chocolate Bar: LI at the Start and the End of the Loop

- At the start of the loop:

1	$\mathtt{p} - \mathtt{c} = \mathtt{k}$	loop invariant
2	$\mathtt{p} - \mathtt{c} = 1$	start of loop
3	$\mathtt{k} = 1$	from 1, 2
4	$\mathtt{p} - \mathtt{c} = 1$	loop invariant

```
p, c := 1, 0
# p - c = k
while p ≠ n
   # p - c = k
   p, c := p+1, c+1
   # p - c = k
# p - c = k, p = n
```

- At the end of the loop:

1	$\mathtt{p} - \mathtt{c} = 1$	loop invariant
2	$\mathtt{p} = \mathtt{n}$	end of the loop
3	$\mathtt{n} - \mathtt{c} = 1$	from 1, 2
4	$\mathtt{c} = \mathtt{n} - 1$	from 3

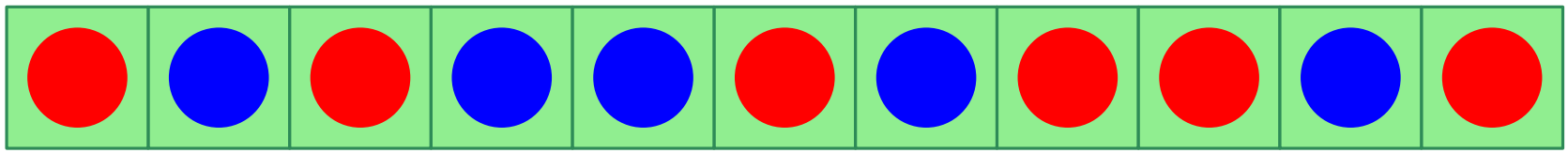
Construct Iterative Algorithm

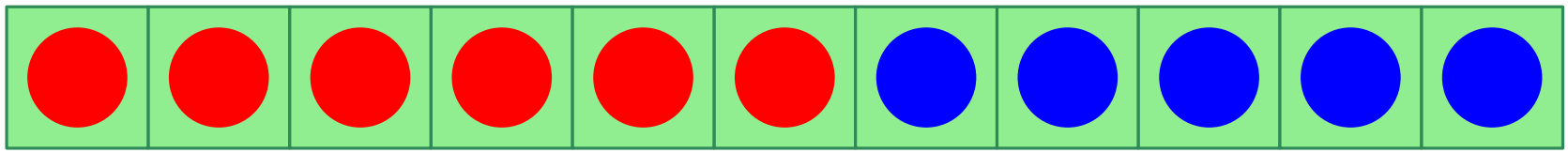
1. Initialize: Establish the loop invariant at the start of the loop.
2. Loop body: Update the variables so as to progress toward the end, but re-establish the loop invariant, at the same time.
3. Terminate: The termination condition and the loop invariant should establish the input–output relation.

Dutch National Flag (DNF)

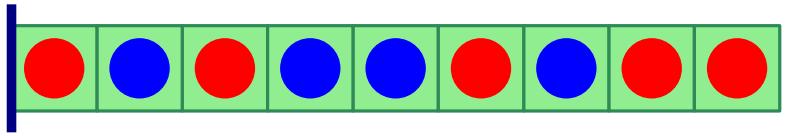
- Dutch National Flag
- E W Dijkstra

Polish National Flag (PNF)

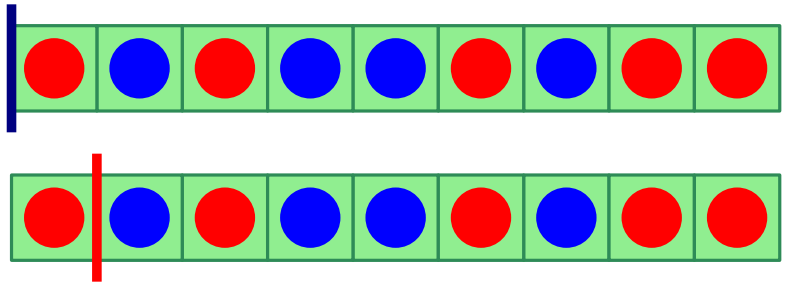




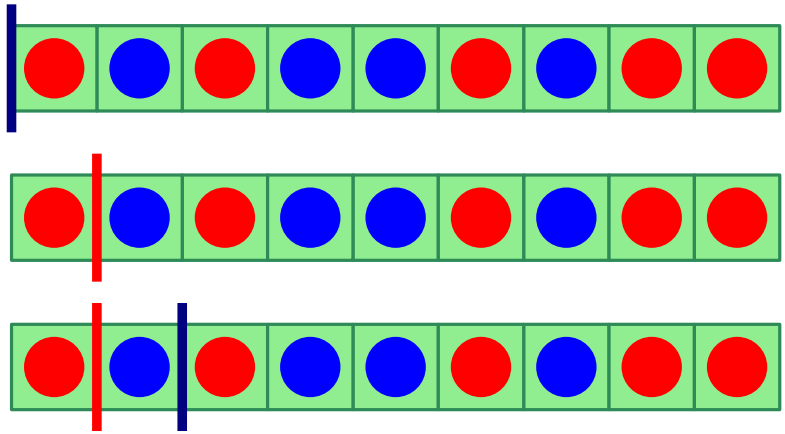
PNF Trace



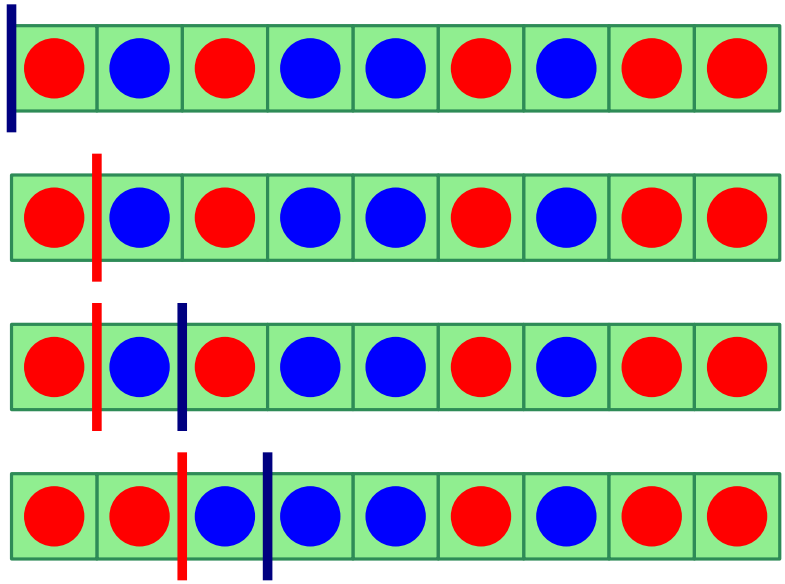
PNF Trace



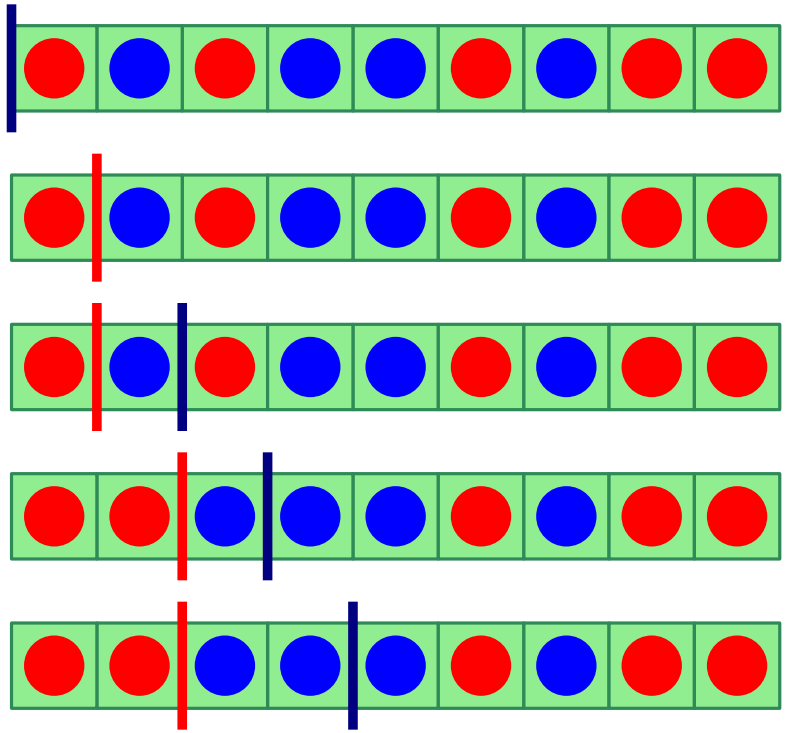
PNF Trace



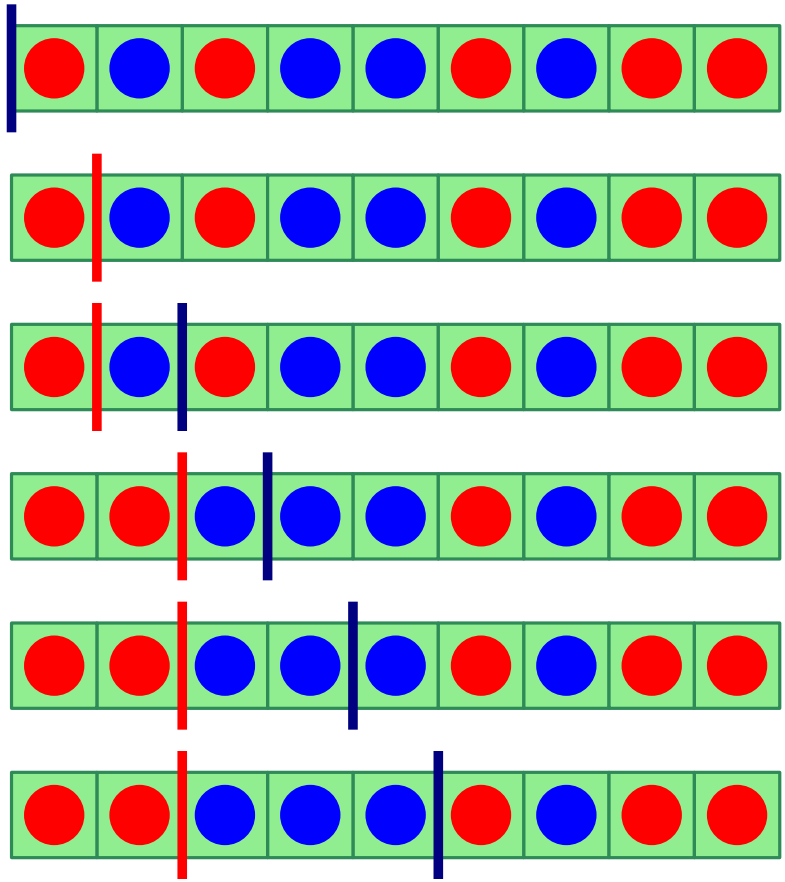
PNF Trace



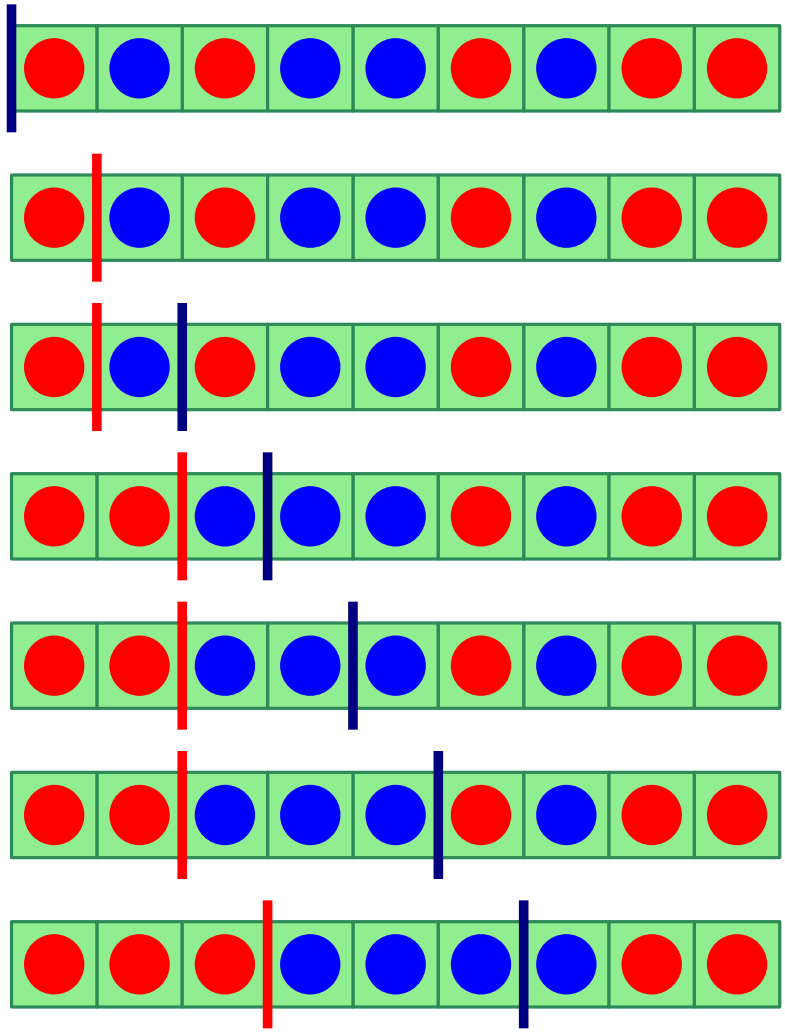
PNF Trace



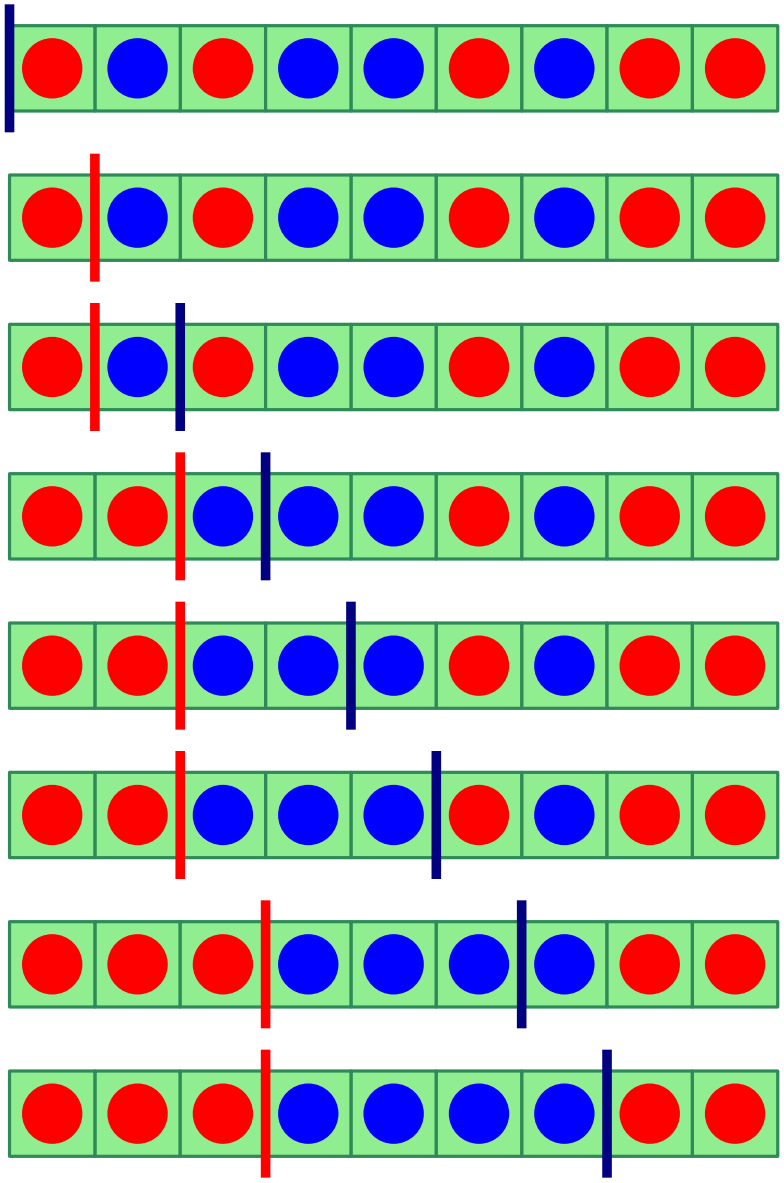
PNF Trace



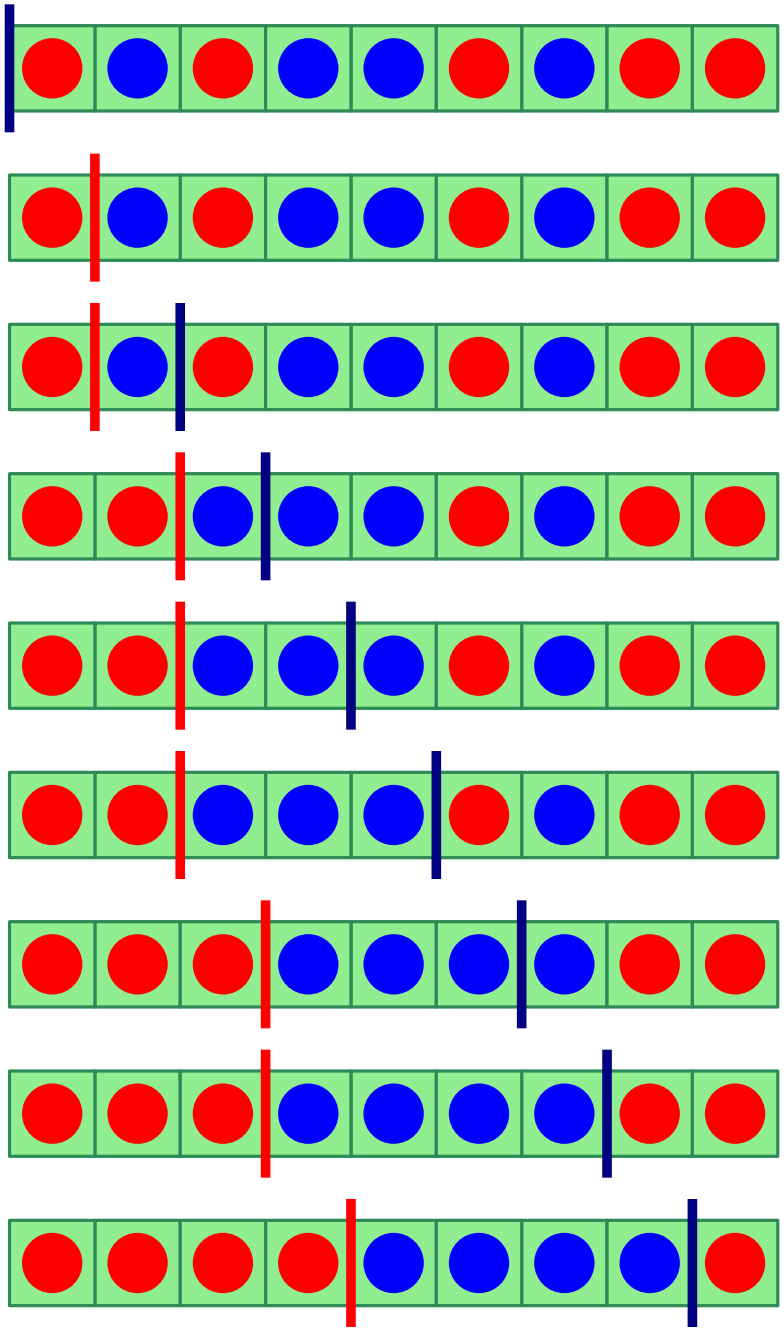
PNF Trace



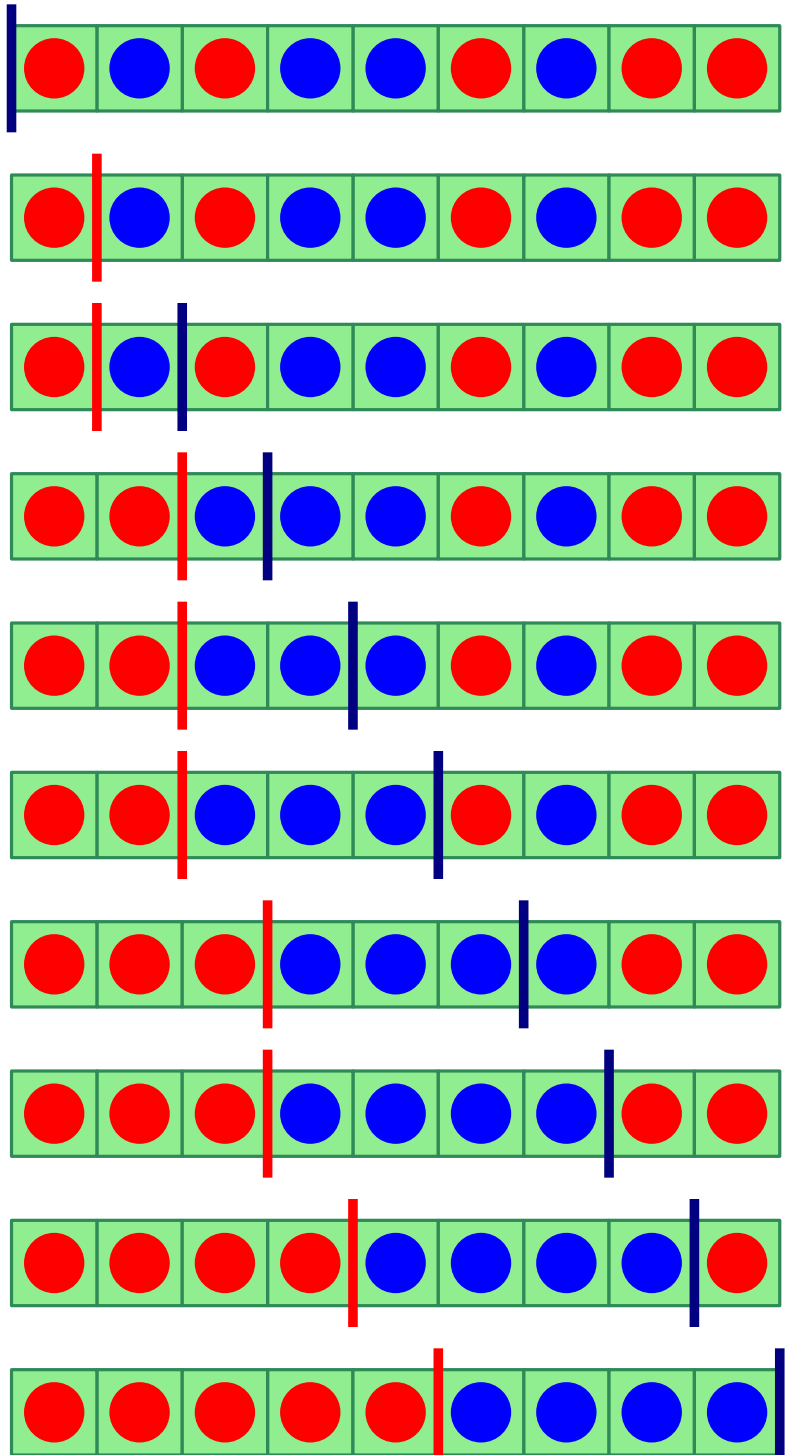
PNF Trace



PNF Trace

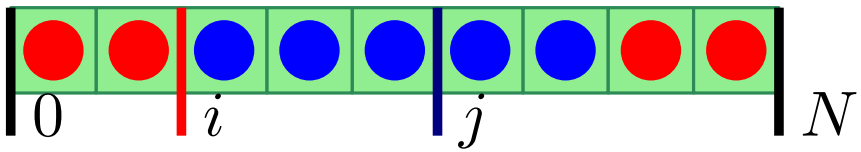


PNF Trace

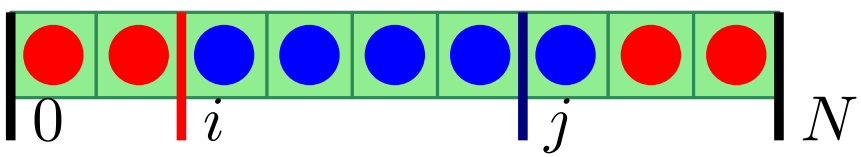


Loop Body

case [j] = 

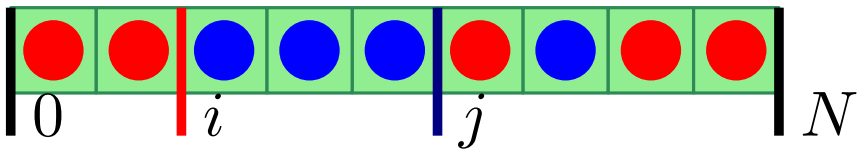


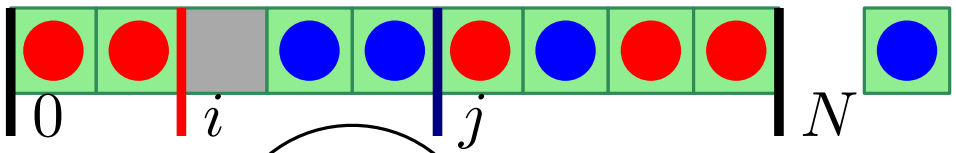

```
j := j+1
```

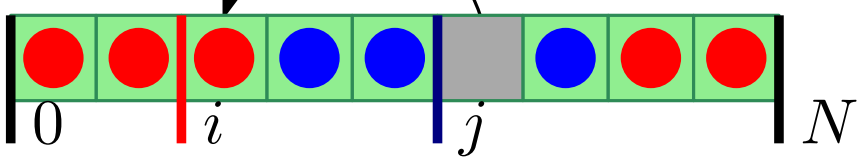


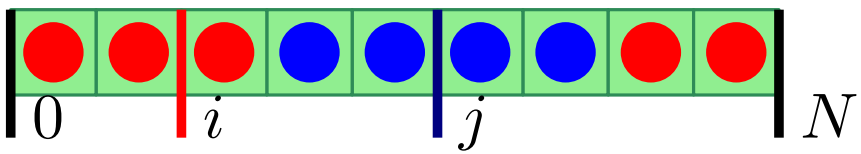
Loop Body

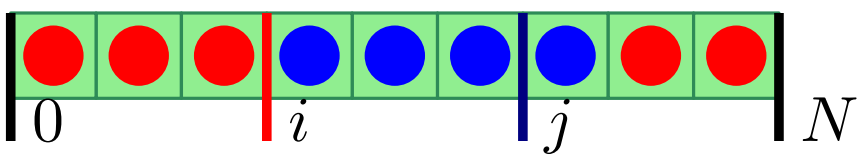
case [j] = 





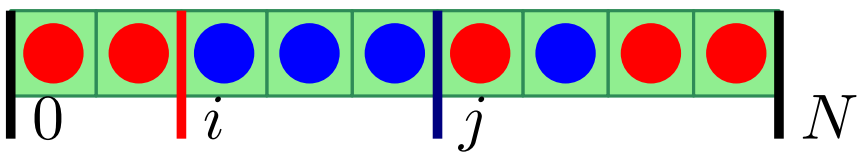


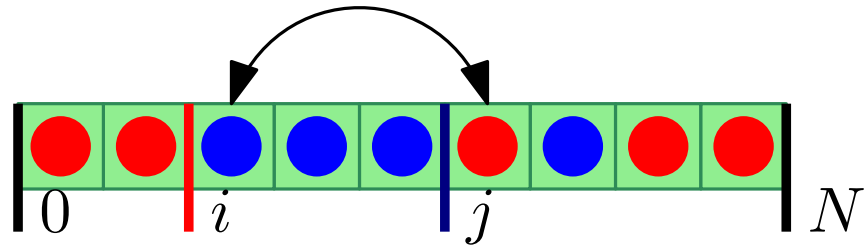


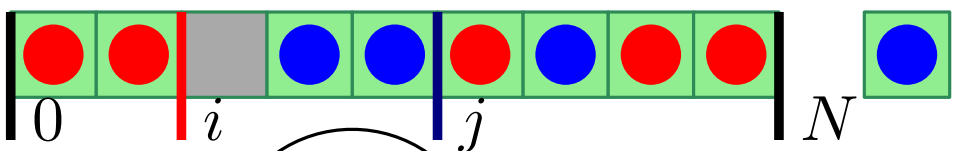


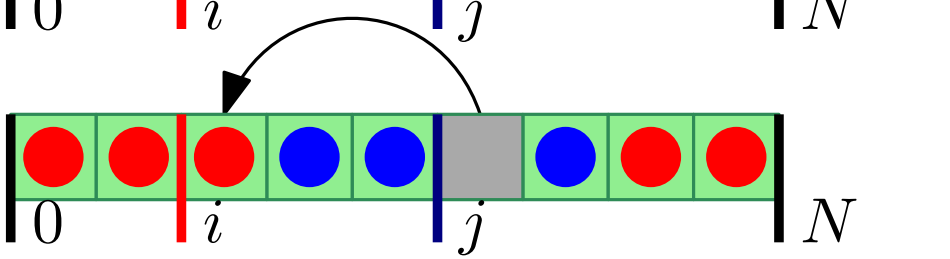
Loop Body

case [j] =

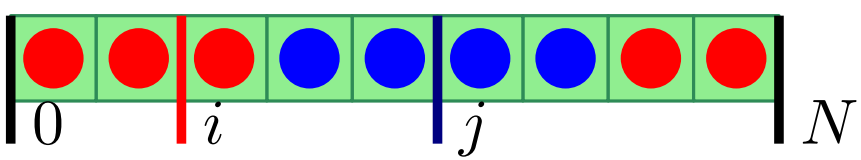


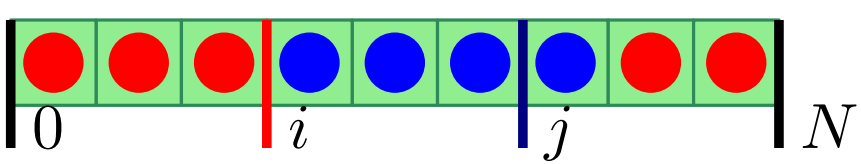


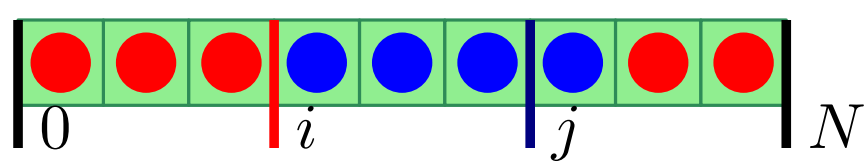





```
[i],[j] := [j],[i]
i, j := i+1, j+1
```

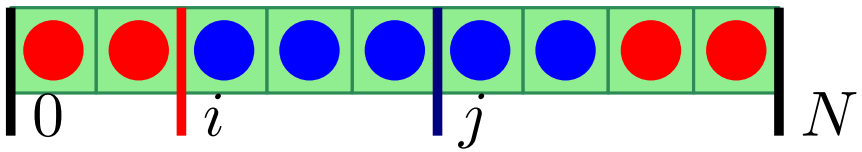




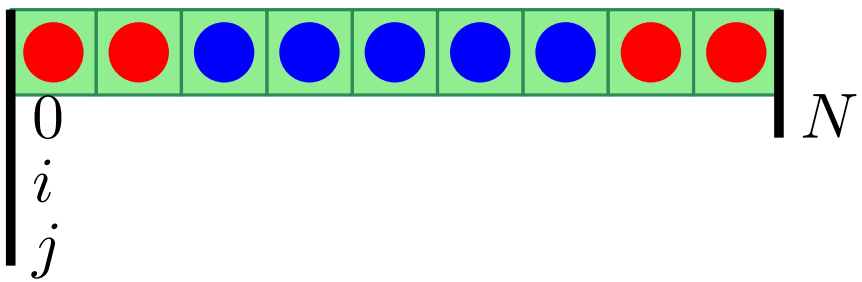


Initialize, Terminate

Loop Invariant

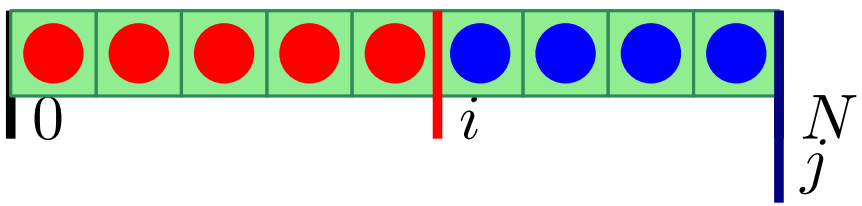


Intialize




```
i, j = 0, 0
```

Terminate




```
j = N
```

Algorithm PNF

```
i, j = 0, 0
until j = N:
   if [j] = ●
      j = j+1
   else:
      [i],[j] = [j],[i]
      i = i+1
      j = j+1
```

```
i, j = 0, 0
until j = N:
   if [j] = ●
      :
   else:
      [i],[j] = [j],[i]
      i = i+1
   j = j+1
```

```
i, j = 0, 0
until j = N:
   if [j] = ●
      [i],[j] = [j],[i]
      i = i+1
   j = j+1
```