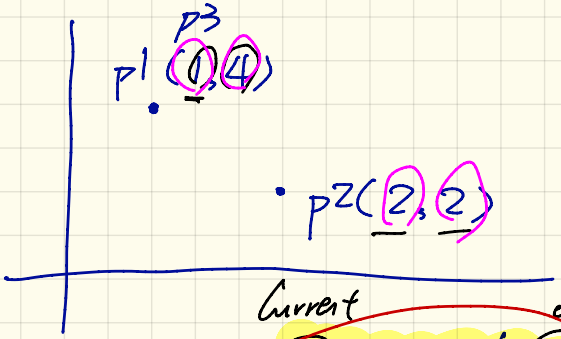
Friday Oct. 19

Midterm Review



class POINT

inherit ANY
redefine is_equal end

feature

feature x, y : INTEGER

feature
is_equal (other : like Current) : BOOLEAN

ensure

p1: x = other.x and y = other.y

end

p2 : Current ~ old Current.deep_twin

Current: p1. is_equal (p3) → POINT

Current: p1. is_equal (p2)

Current		other
x 1	F	x 2
y 4	F	y 2

Is the postcondition appropriate?

obligation of supplier

Fix?

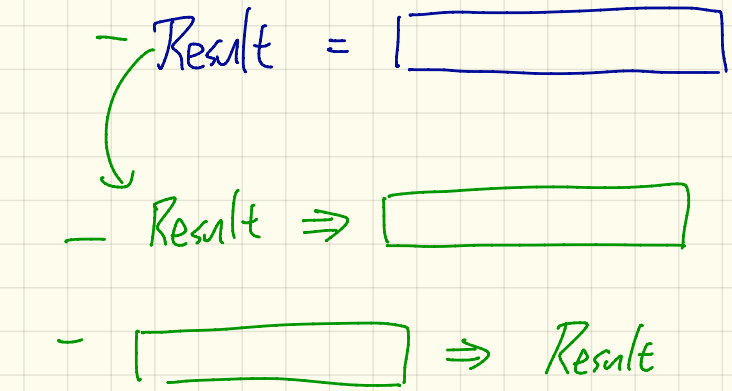
P.C. violation

Result = (x = other.x and y = other.y)

F F T

1 2 4 2

ensure



$$(p \equiv q) = (p \Rightarrow q \wedge q \Rightarrow p)$$

do

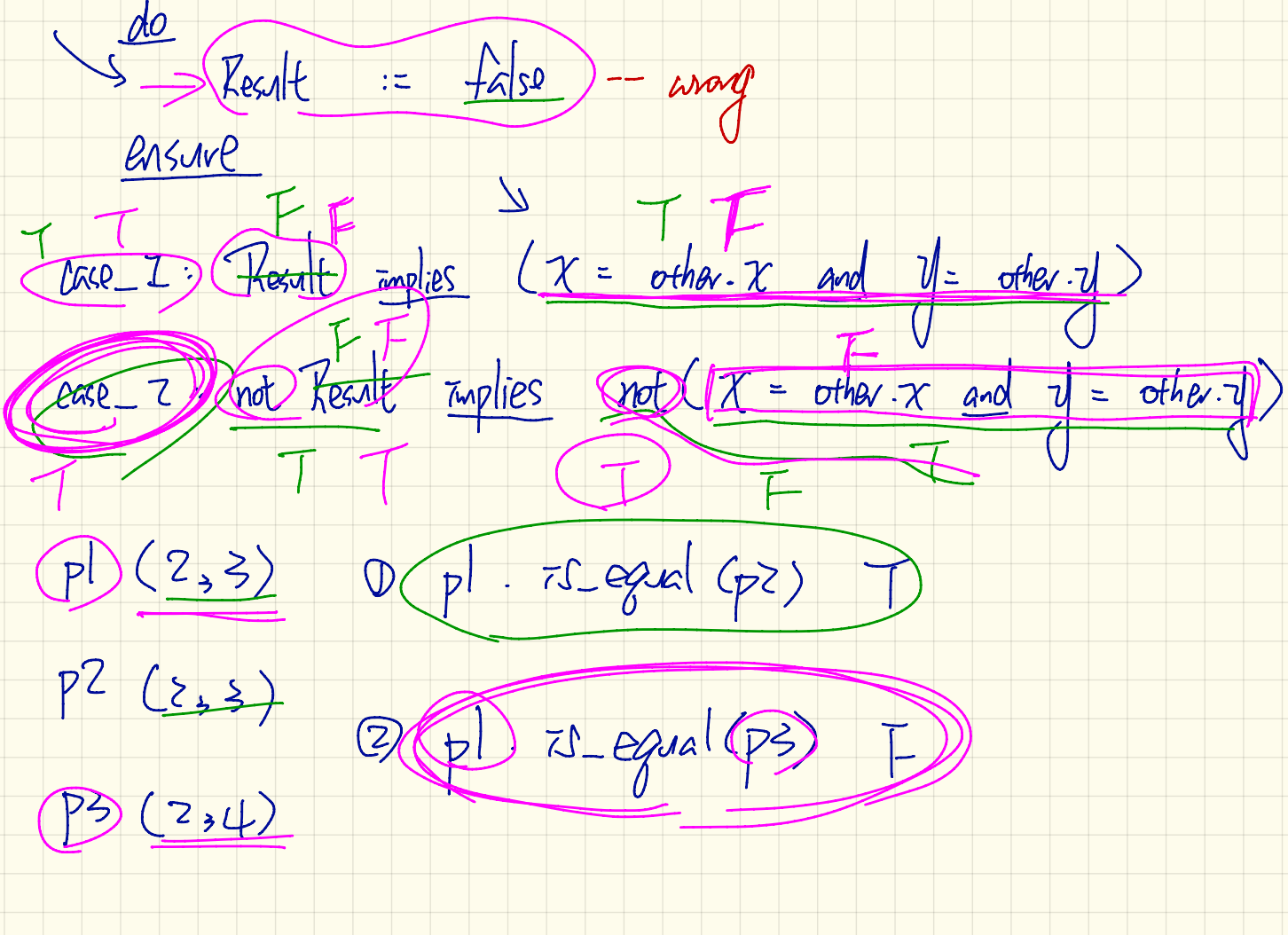
Result := false

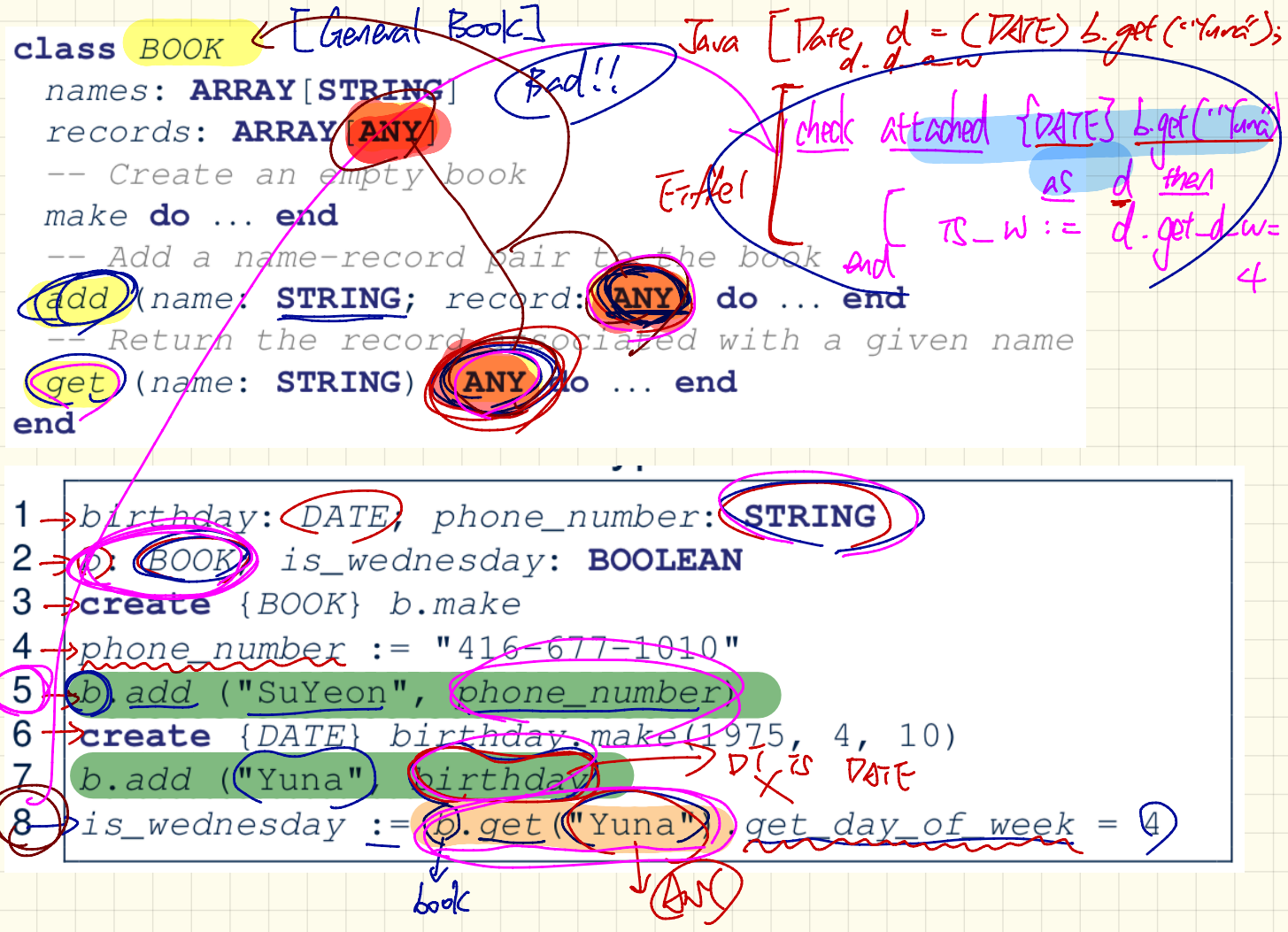
ensure

p: Result implies (x = other.x and y = other.y)

Hint.. F ⇒ T = T

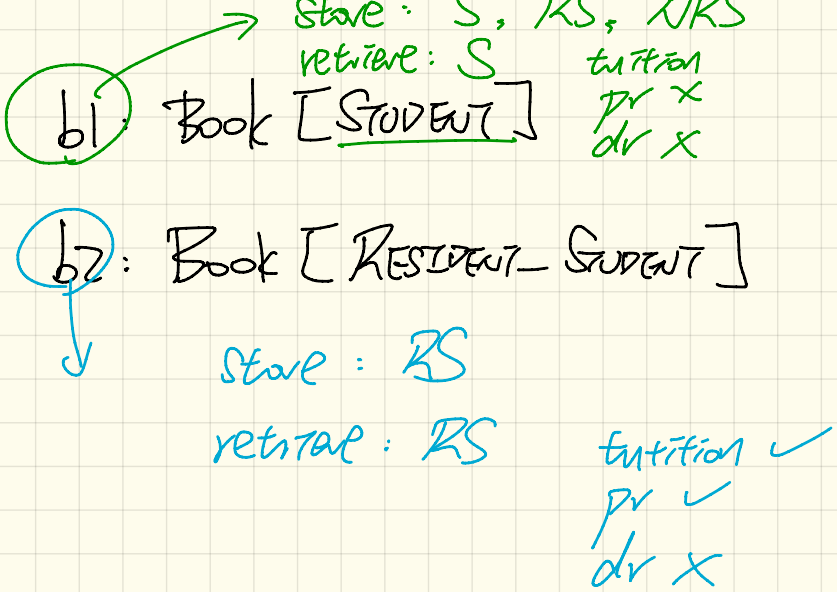
produced by supplier


do
Result := false -- wrong
ensure
case_1: Result implies (x = other.x and y = other.y)
case_2: not Result implies not (x = other.x and y = other.y)
p1 (2, 3)
p2 (2, 3)
p3 (2, 4)
① p1.is_equal (p2) T
② p1.is_equal (p3) F


class BOOK
names: ARRAY[STRING]
records: ARRAY[ANY]
-- Create an empty book
make do ... end
-- Add a name-record pair to the book
add (name: STRING; record: ANY) do ... end
-- Return the record associated with a given name
get (name: STRING): ANY do ... end
end
[General Book]
Bad!!
Java [Date d = (DATE) b.get("Yuna");
d.d_o_w
Eiffel
check attached {DATE} b.get("Yuna") as d then
ts_w := d.get_d_o_w = 4
end
1 birthday: DATE; phone_number: STRING
2 b: BOOK; is_wednesday: BOOLEAN
3 create {BOOK} b.make
4 phone_number := "416-677-1010"
5 b.add ("SuYeon", phone_number)
6 create {DATE} birthday.make(1975, 4, 10)
7 b.add ("Yuna", birthday)
8 is_wednesday := b.get("Yuna").get_day_of_week = 4
D1 is DATE
book
ANY

```
class BOOK[K, V]
  names: ARRAY[STRING]
  records: ARRAY[V]
  -- Create an empty book
  make do ... end
  /* Add a name-record pair to the book */
  add (name: STRING; record: V) do ... end
  /* Return the record associated with a given name */
  get (name: STRING): V do ... end
end
```

```
birthday: DATE; phone_number: STRING
b: BOOK[DATE]; is_wednesday: BOOLEAN
create BOOK[DATE] b.make
phone_number := "416-67-1010"
b.add ("SuYeon", phone_number)
create {DATE} birthday.make (1975, 4, 10)
b.add ("Yuna", birthday)
is_wednesday := b.get("Yuna").get_day_of_week == 4
```

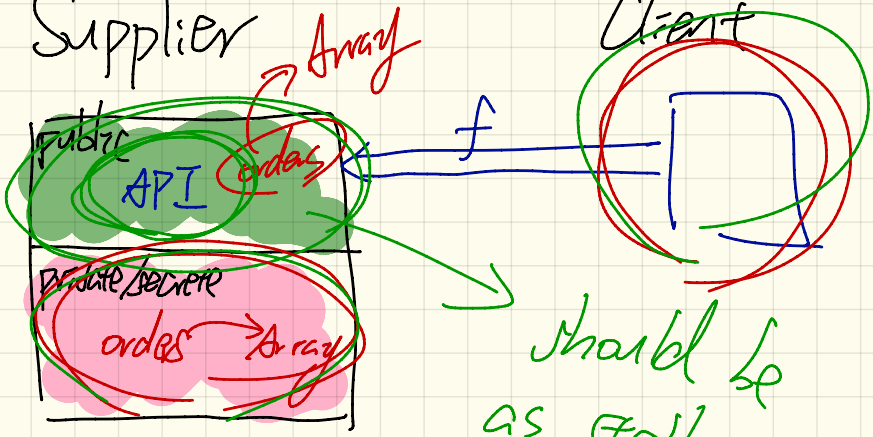

store : S, RS, NRS
retrieve : S
tuition
pr ×
dr ×
b1 : Book [STUDENT]
b2 : Book [RESIDENT_STUDENT]
store : RS
retrieve : RS
tuition ✓
pr ✓
dr ×

Information Hiding

e.g. choice of data. str.

Design decisions that are subject to constant changes should be hidden from the clients.

Supplier



Eiffel

deferred *

effective +

class Point

end

Java

abstract

non-abstract / Concrete

```
class BOOK[ G ]
  names: ARRAY[STRING]
  records: ARRAY[ G ]
  -- Create an empty book
  make do ... end
  /* Add a name-record pair to the book */
  add (name: STRING; record: G ) do ... end
  /* Return the record associated with a given name */
  get (name: STRING): G do ... end
end
```

STUDENT

STUDENT

b1: BOOK[STUDENT]

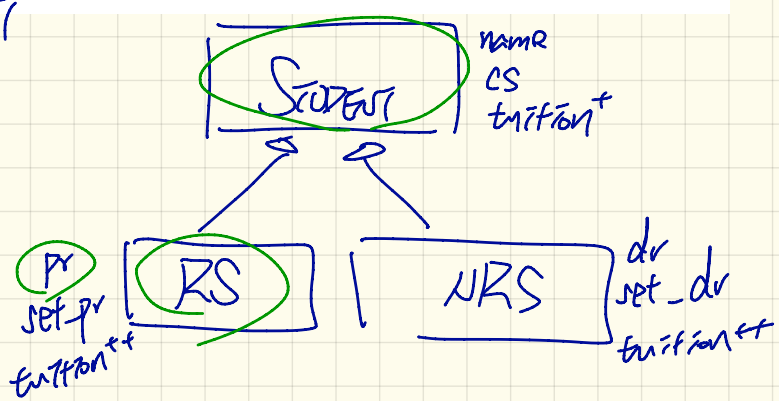
⋮

- b1.get("Stella").tuition

STUDENT

b1.get("Rachael").set_pr ✗

STUDENT



Say a client wants to store objects of types $C_1, C_2, \ldots, C_n$,

KS URS

IP_bS IP_bSP

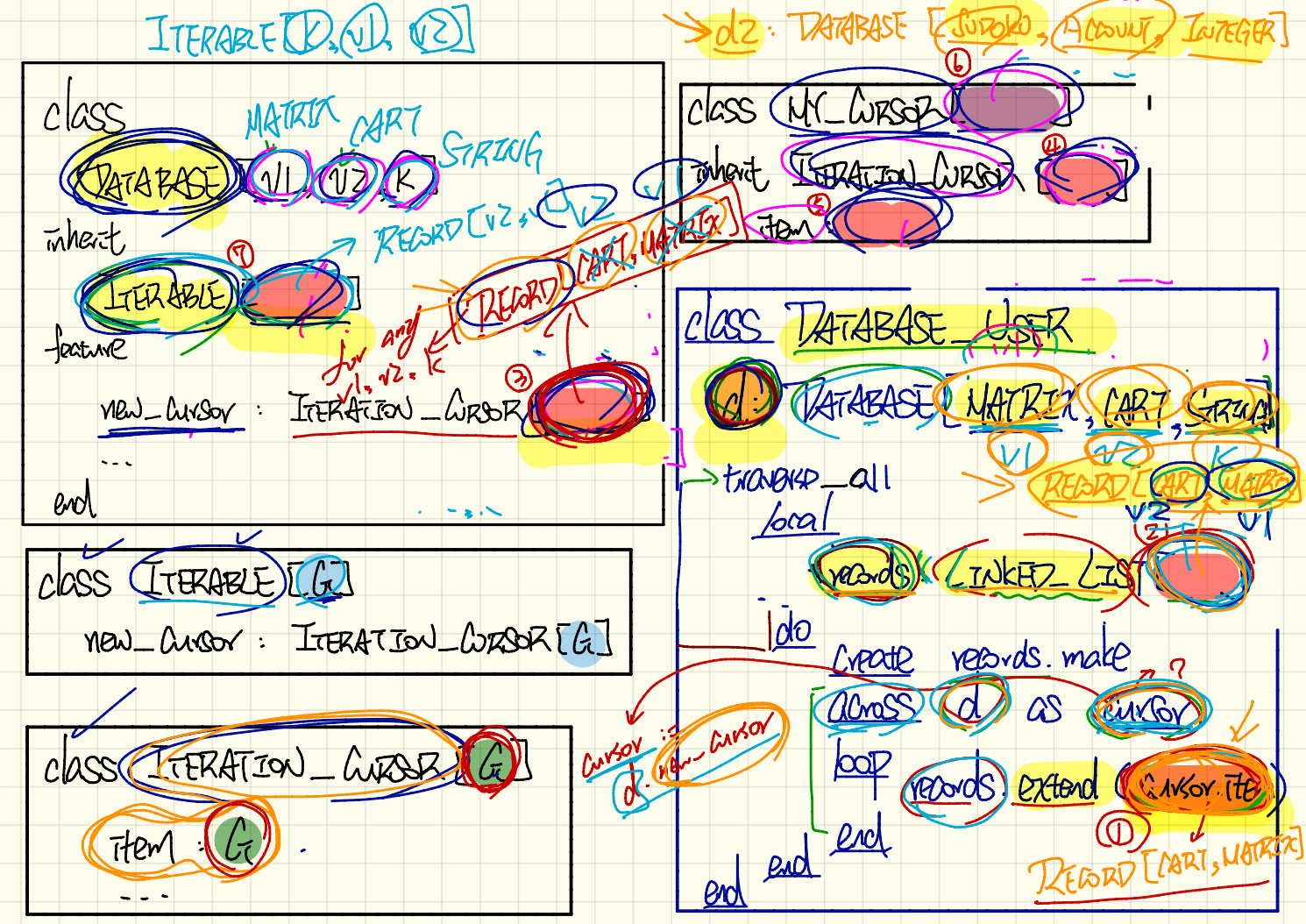
how should they declare using the generic book?

```
b : Book[ ? ]
```

STUDENT

IoS

choose the "lowest" common ancestor of $C_1, C_2 \ldots, C_n$


ITERABLE [K, V1, V2]
class
DATABASE
MATRIX
CART
STRING
V1
V2
K
inherit
ITERABLE
RECORD [V2, K]
RECORD [CART, MATRIX]
for any V1, V2, K
feature
new_cursor : ITERATION_CURSOR
...
end
d2 : DATABASE [SUDOKU, ACCOUNT, INTEGER]
class MY_CURSOR
inherit ITERATION_CURSOR
item :
class DATABASE_USER
d : DATABASE [MATRIX, CART, STRING]
RECORD [CART, MATRIX]
traverse_all
local
records
LINKED_LIST
do
create records.make
across d as cursor
loop
records.extend (cursor.item)
RECORD [CART, MATRIX]
end
end
end
class ITERABLE [G]
new_cursor : ITERATION_CURSOR [G]
class ITERATION_CURSOR [G]
item : G
...
cursor := d.new_cursor

Invariant

keys_unique:

such that

the case

$\forall i : 1 \le i \le keys.count \bullet$

$\forall j : 1 \le j \le keys.count \bullet$

$i \neq j \Rightarrow keys[i] \neq keys[j]$

$keys[i] \sim keys[j] \Rightarrow i = j$

$\forall i, j : 1 \le i \le keys.count \wedge 1 \le j \le keys.count \bullet$

$i \neq j \Rightarrow keys[i] \neq keys[j]$

keys

across |1..|keys.count as i

all

across |1..|keys.count as j

all

i.item /= j.item implies keys[i.item] /~ keys[j.item]

end

end

across keys as k1

all

across keys as k2

all

get_index_of_key(k1)

k1.cursor_index /= k2.cursor_index

implies

k1 /~ k2

end

end