Lecture 4
Wednesday January 15

Shorter Office Hours today: 3pm to 4pm

Lab0: Tutorial Videos (basic syntax, debugger)

Lab1: See due date in course wiki

plagiarism check

This Friday: in-lab demo at 10:30

Precondition & Postcondition Exercise

∀ ∃

change_at (a: **ARRAY**[**STRING**]; i: **INTEGER**; ns; **STRING**)

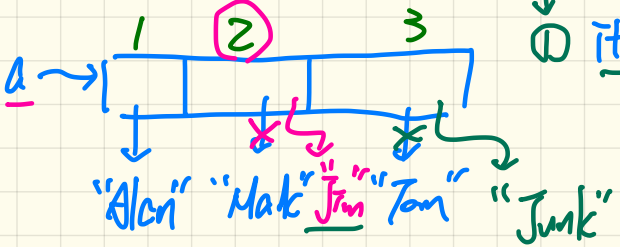
- - Change index `i` in array `a` to string `ns`

require

?? 1 <= i and i <= a.count

ensure

??



change_at(a, 2, "Jim")

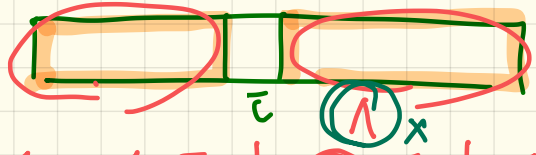
① item_at_i_changed : a[i] ~ ns

② others_unchanged :

valid index

not the position to change.

$\forall j \mid 1 \le j \le a.count \wedge j \neq i \bullet a[j] \sim old\ a[j]$

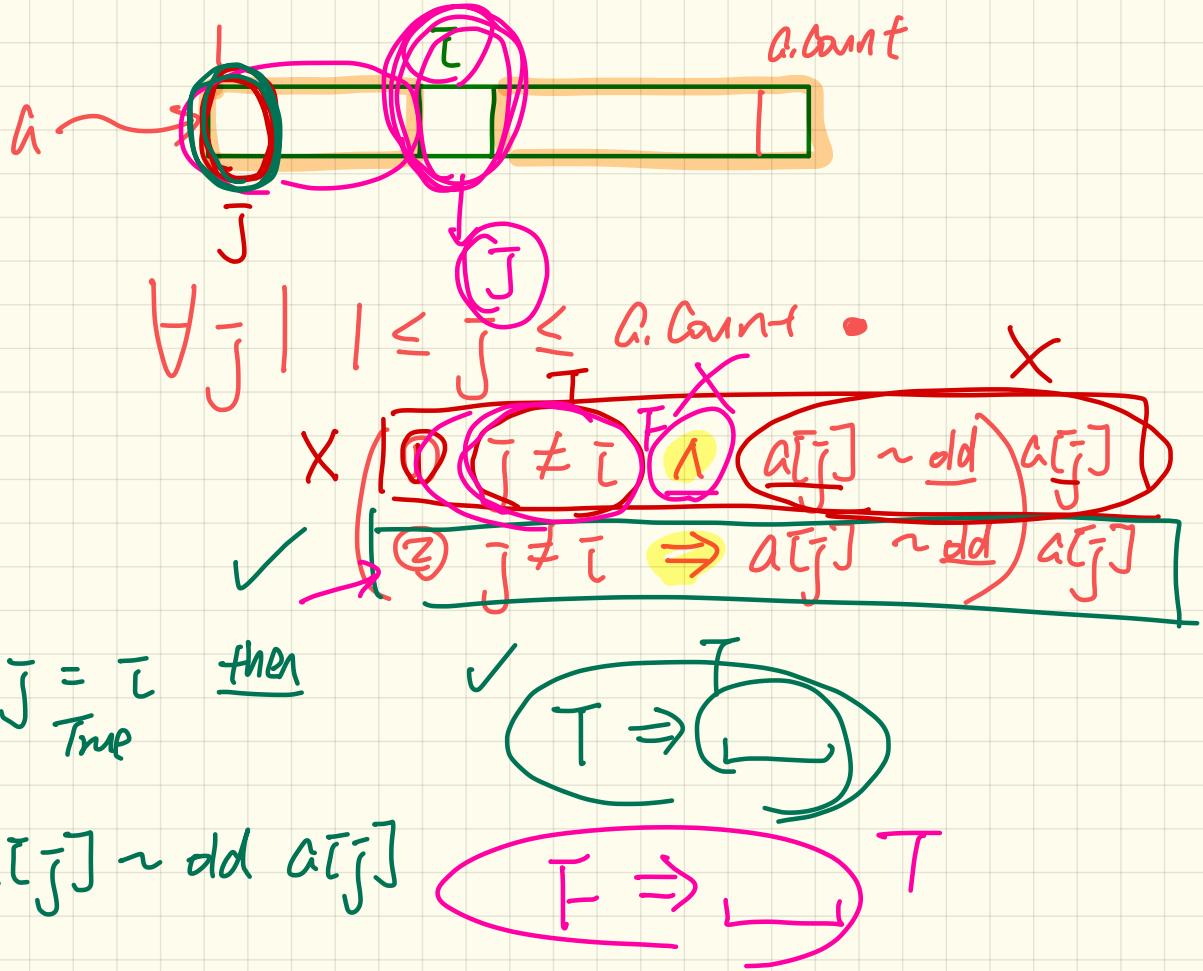


✓

$\forall j \mid 1 \le j \le i-1 \vee i+1 \le j \le a.\text{Count} \cdot$

$a[j] \sim dd\ a[j]$

$$\left(\forall x \mid R(x) \cdot P(x)\right) \equiv \left(\forall x \cdot R(x) \Rightarrow P(x)\right)$$


a
i
j
a.count
∀ j | 1 ≤ j ≤ a.count •
X
X
① j ≠ i ∧ a[j] ~ dd a[j]
② j ≠ i ⇒ a[j] ~ dd a[j]
if j = i then
True
else
a[j] ~ dd a[j]
T ⇒
F ⇒
T

change_at (a: **ARRAY**[**STRING**]; i: **INTEGER**; ns: **STRING**)

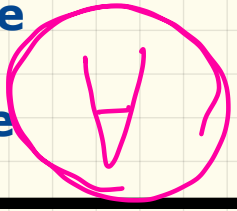
- - Change index `i` in array `a` to string `ns`

require

??

ensure

??

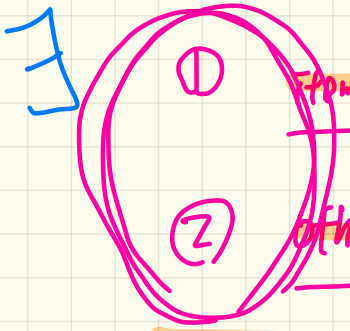


= ⇔

and ∧ ==

or ∨

implies ⇒



① from_at_i_changed : a[i] ~ ns

② others_unchanged : ∀ j | 1 ≤ j ≤ a.count •

j ≠ i ⇒ a[j] ~ old a[j]

across 1 |..| a.count is j

all some

j = i implies a[j] ~ old a[j]

end

equals

Exercise: translate to Eiffel.

$$\forall j \mid 1 \le j \le a.\text{count} \bullet$$

$$i = j \Rightarrow a[j] \sim v$$

$$\wedge$$

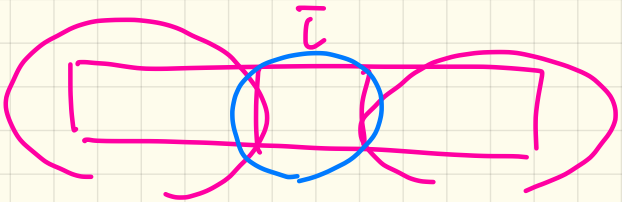
$$i \neq j \Rightarrow a[j] \sim \underline{\text{old}}\ a[j]$$

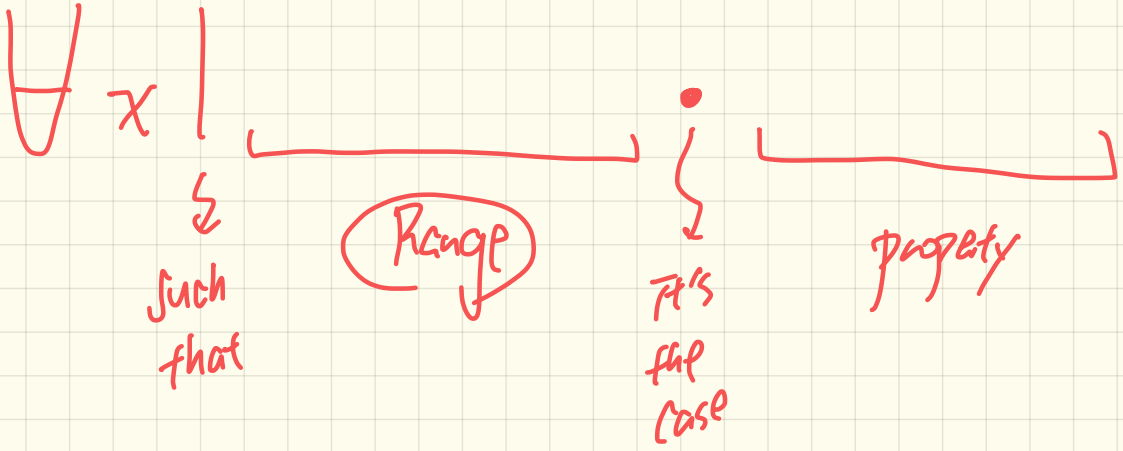
ensure

① ② ③ ⋮

① ∧ ② ∧ ③ $a[i] \sim v$ and.

across ... end





$$\forall x \mid 1 \leq x \leq 5 \cdot x^2 \geqslant 25 \;\; F \;\; x \geqslant 25 \;\; F$$

change_at (a: **ARRAY**[**STRING**]; i: **INTEGER**; ns; **STRING**)

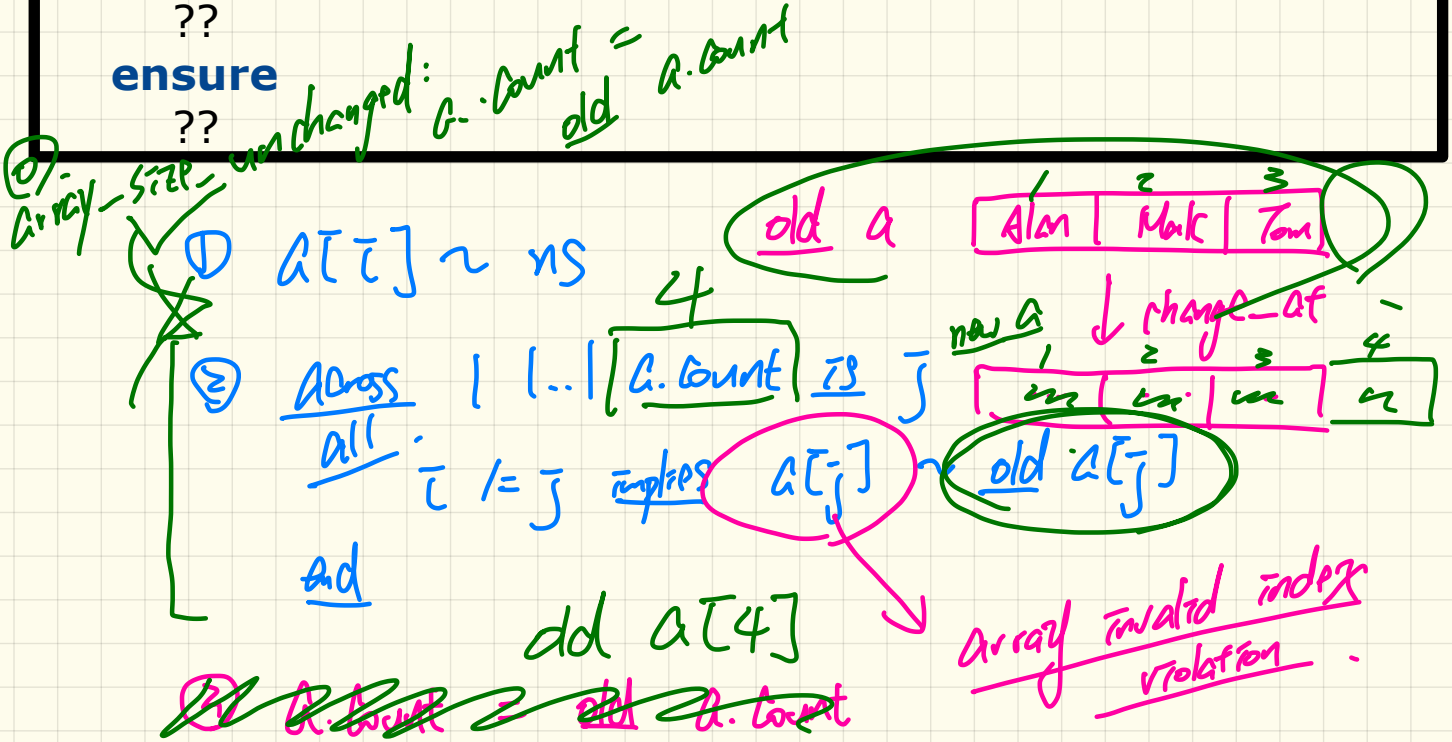
- - Change index `i` in array `a` to string `ns`

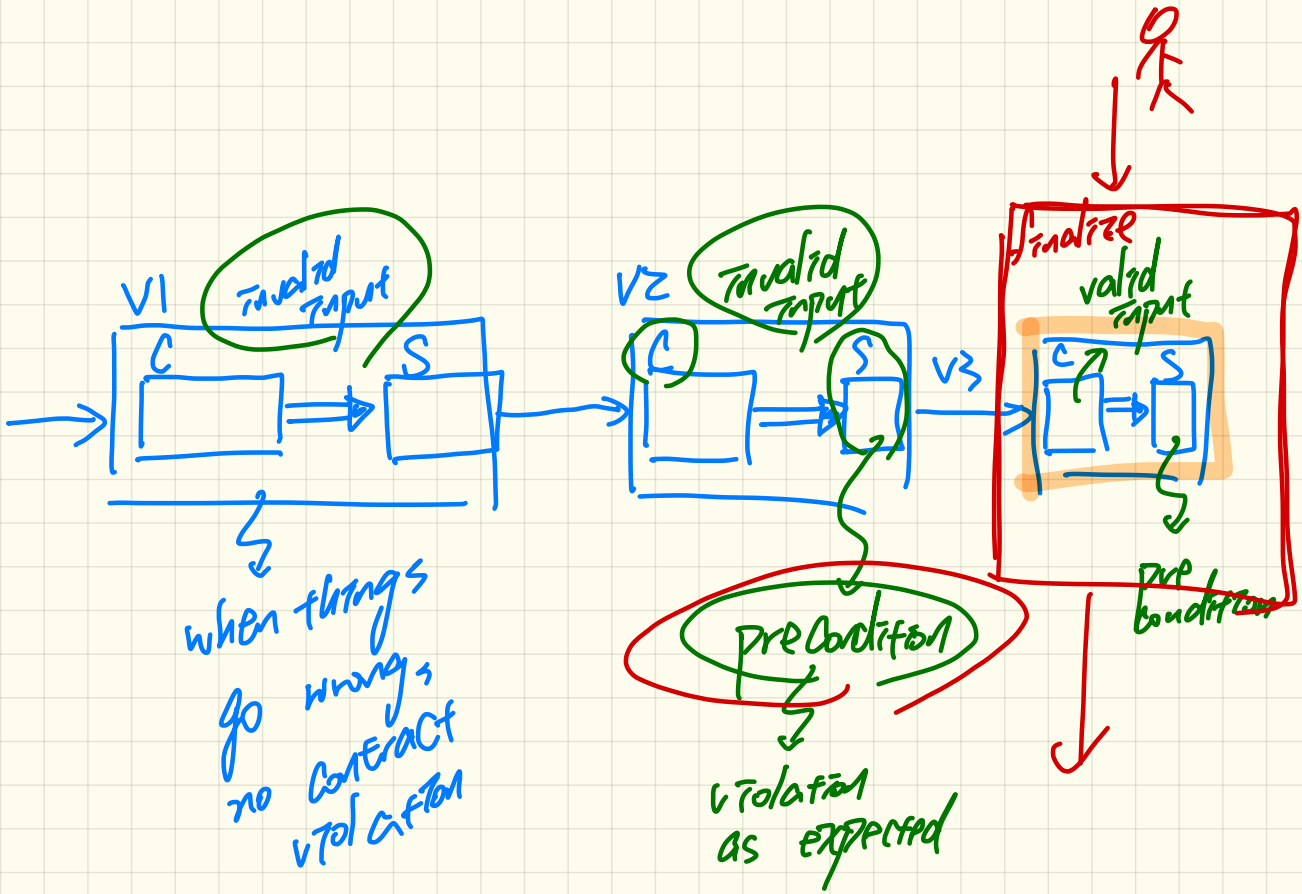
require

??

ensure

??




V1
invalid input
C
S
when things go wrong, no contract violation
V2
invalid input
C
S
V3
finalize
valid input
C
S
pre condition
precondition
violation as expected

How is DbC Useful in Guiding System Development?

Client's View:

- A console application.
- Keep entering names randomly until done.
- Keep inquiring if a name exists until quit.

Expected Run

```
Enter a name, or `done` to start inquiring: e
Enter a name, or `done` to start inquiring: c
Enter a name, or `done` to start inquiring: d
Enter a name, or `done` to start inquiring: a
Enter a name, or `done` to start inquiring: b
Enter a name, or `done` to start inquiring: done
a  b  c  d  e
Enter a name, or `quit` to stop inquiring: a
a exists!
Enter a name, or `quit` to stop inquiring: b
b exists!
Enter a name, or `quit` to stop inquiring: c
c exists!
Enter a name, or `quit` to stop inquiring: d
d exists!
Enter a name, or `quit` to stop inquiring: e
e exists!
Enter a name, or `quit` to stop inquiring: f
f does not exist!
Enter a name, or `quit` to stop inquiring: g
g does not exist!
Enter a name, or `quit` to stop inquiring: quit
```

Supplier's Implementation Strategy

- Store names in an array.
- Upon an inquiry: Binary Search,

Version 1: Wrong Implementation, No Contracts

```
class interface
   DATABASE_V1

create
   make

feature -- Constructor

   add_name (n: STRING_8)
         -- Add `n` to database.

   data_exists (n: STRING_8): BOOLEAN
         -- Does `n` exist in the database?

   make
         -- Create an empty database.

end -- class DATABASE_V1
```


```
class interface
   UTILITIES_V1

create
   default_create

feature -- Binary Search

   search (a: ARRAY [STRING_8]; a_name: STRING_8): BOOLEAN

end -- class UTILITIES_V1
```

- Data array in DATABASE is not kept sorted.
- Binary search in UTILITIES does not require a sorted input array.
- When user enters names in an unsorted order, output is wrong.
- But no contract violation!
- A bad design is when something goes wrong, there is no party to blame.

Version 1: User Interaction Session

```
Enter a name, or `done` to start inquiring: e
Enter a name, or `done` to start inquiring: c
Enter a name, or `done` to start inquiring: d
Enter a name, or `done` to start inquiring: a
Enter a name, or `done` to start inquiring: b
Enter a name, or `done` to start inquiring: done
e  c  d  a  b
Enter a name, or `quit` to stop inquiring: a
a does not exist!
Enter a name, or `quit` to stop inquiring: b
b does not exist!
Enter a name, or `quit` to stop inquiring: c
c does not exist!
Enter a name, or `quit` to stop inquiring: d
d exists!
Enter a name, or `quit` to stop inquiring: e
e does not exist!
Enter a name, or `quit` to stop inquiring: f
f does not exist!
Enter a name, or `quit` to stop inquiring: g
g does not exist!
Enter a name, or `quit` to stop inquiring: quit
```

Version 2: Wrong Implementation, Proper Precondition

```
class interface
    DATABASE_V2

create
    make

feature -- Constructor

    add_name (n: STRING_8)
            -- Add `n` to database.

    data_exists (n: STRING_8): BOOLEAN
            -- Does `n` exist in the database?

    make
            -- Create an empty database.

end -- class DATABASE_V2
```

u

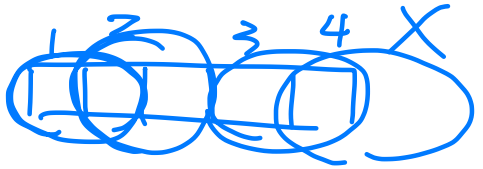
```
class interface
    UTILITIES_V2

create
    default_create

feature -- Binary Search

    search (a: ARRAY [STRING_8]; a_name: STRING_8): BOOLEAN
        require
            array_sorted: across
                a.lower |..| (a.upper - 1) is i
            all
                a [i] < a [i + 1]
            end

end -- class UTILITIES_V2
```



$\forall i \mid a.lower \le i \le a.upper - 1 \bullet a[i] < a[i+1]$

- Data array in DATABASE is not kept sorted.
- Binary search in UTILITIES now requires a sorted input array.
- When an unsorted array is passed for search, a contract violation occurs!
- A good design is when something goes wrong, there is one party to blame.

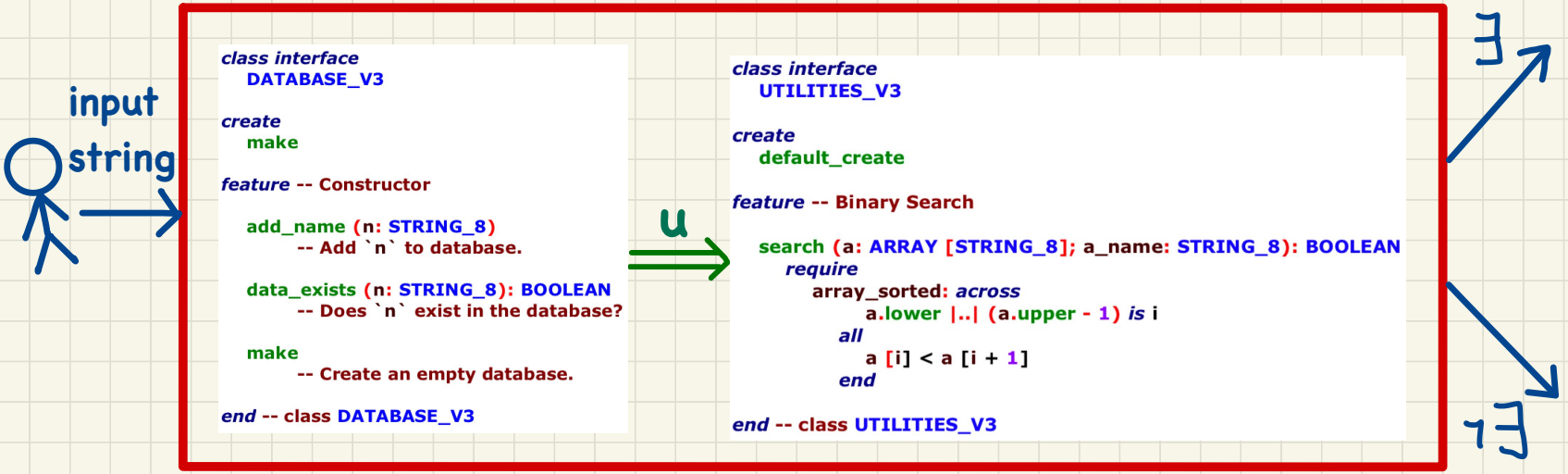
Version 2: User Interaction Session

```
Enter a name, or `done` to start inquiring: e
Enter a name, or `done` to start inquiring: c
Enter a name, or `done` to start inquiring: d
Enter a name, or `done` to start inquiring: a
Enter a name, or `done` to start inquiring: b
Enter a name, or `done` to start inquiring: done
e  c  d  a  b
Enter a name, or `quit` to stop inquiring: a

why-dbc-useful: system execution failed.
Following is the set of recorded exceptions:

******************************** Thread exception ********************************
In thread           Root thread           0x0 (thread id)
**********************************************************************************
----------------------------------------------------------------------------------
Class / Object      Routine               Nature of exception          Effect
----------------------------------------------------------------------------------
UTILITIES_V2        search @1             array_sorted:
<000000010EFF0FB8>                        Precondition violated.       Fail
----------------------------------------------------------------------------------
DATABASE_V2         data_exists @2
<000000010EFEFDF8>                        Routine failure.             Fail
----------------------------------------------------------------------------------
ROOT                make @30
<000000010EFEF548>                        Routine failure.             Fail
----------------------------------------------------------------------------------
ROOT                root's creation
<000000010EFEF548>                        Routine failure.             Exit
----------------------------------------------------------------------------------
```

Version 3: Fixed Implementation, Proper Precondition



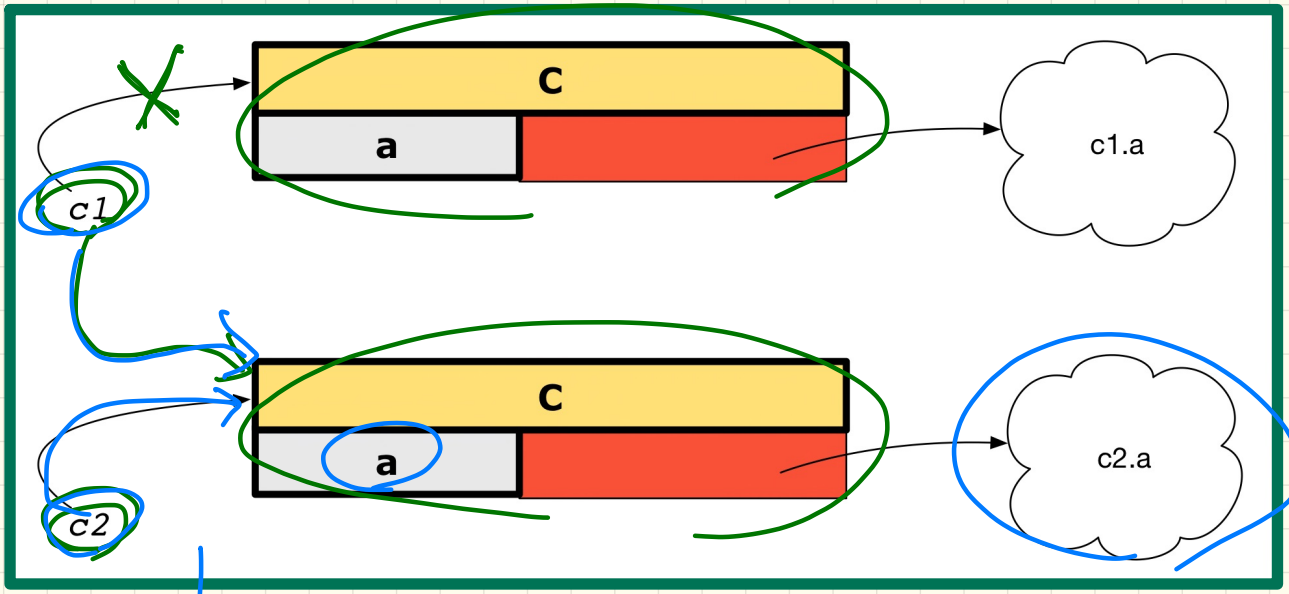
- Data array in DATABASE is now kept sorted (so as to avoid contract violation).
- Binary search in UTILITIES still requires a sorted input array.
- A sorted array is always passed for search, a contract violation never occurs!
- Now finalize/deliver the working system with contracts checking turned off.

Version 3: User Interaction Session

```
Enter a name, or `done` to start inquiring: e
Enter a name, or `done` to start inquiring: c
Enter a name, or `done` to start inquiring: d
Enter a name, or `done` to start inquiring: a
Enter a name, or `done` to start inquiring: b
Enter a name, or `done` to start inquiring: done
a  b  c  d  e
Enter a name, or `quit` to stop inquiring: a
a exists!
Enter a name, or `quit` to stop inquiring: b
b exists!
Enter a name, or `quit` to stop inquiring: c
c exists!
Enter a name, or `quit` to stop inquiring: d
d exists!
Enter a name, or `quit` to stop inquiring: e
e exists!
Enter a name, or `quit` to stop inquiring: f
f does not exist!
Enter a name, or `quit` to stop inquiring: g
g does not exist!
Enter a name, or `quit` to stop inquiring: quit
```

Reference Copy: c1 := c2

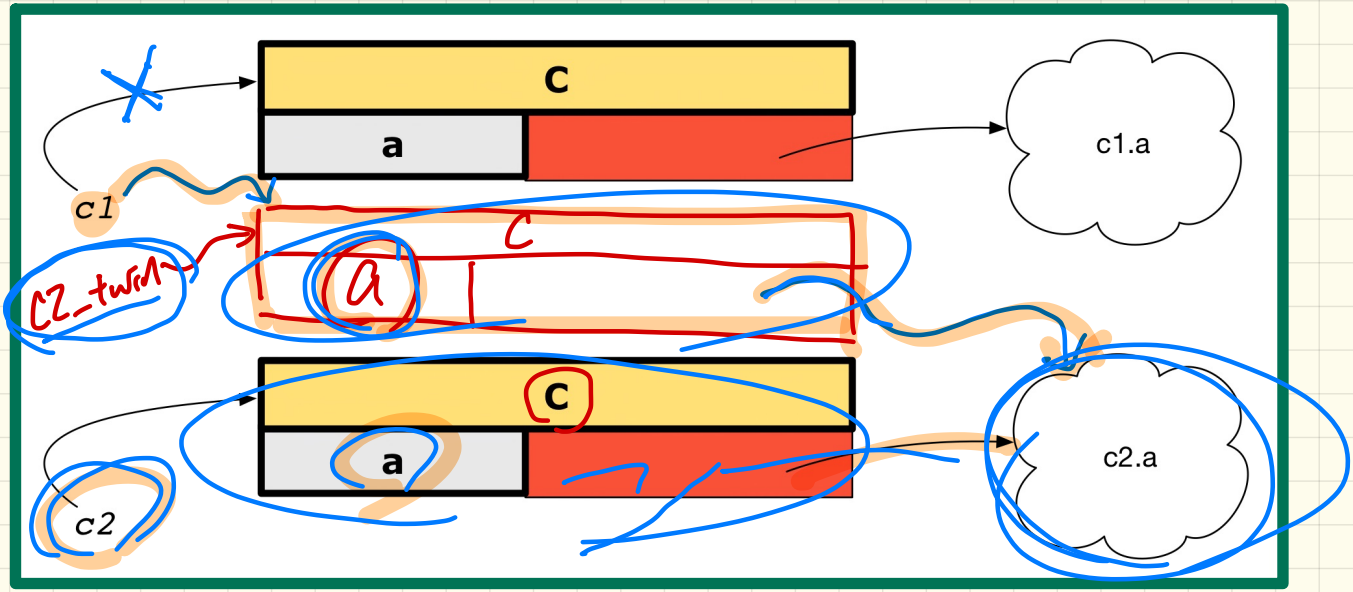
cheapest



c1 = c2 T

c1.a = c2.a T.

Shallow Copy: c1 := c2.twin

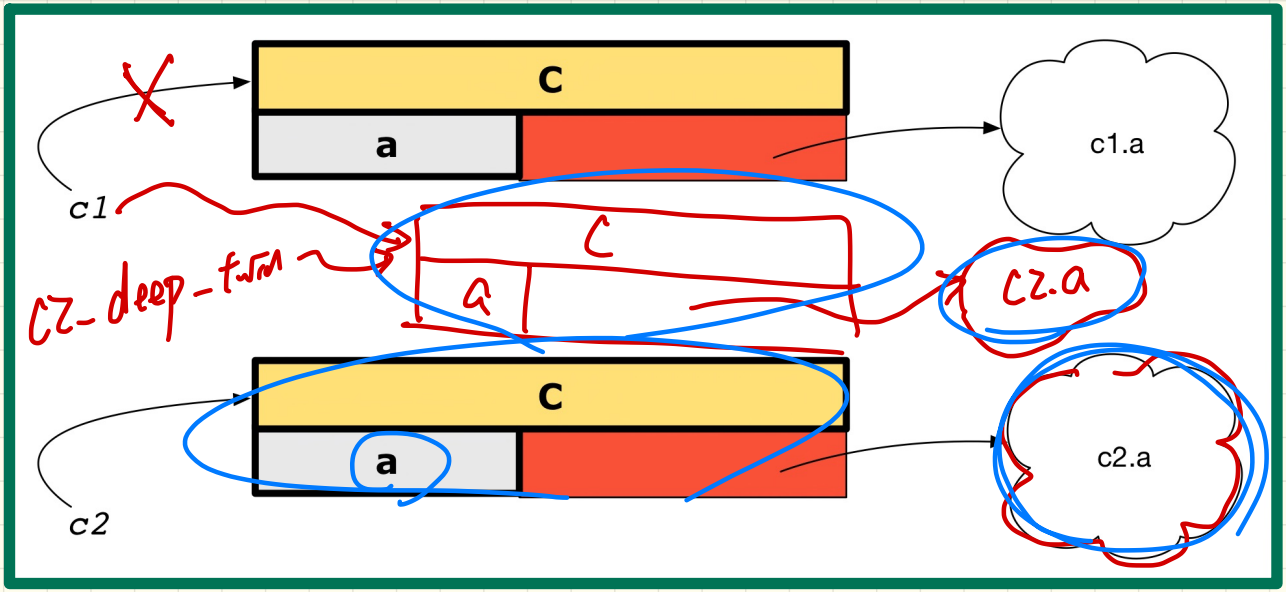


c2_twin.a := c2.a

c2 = c2.twin (F)

c2.a = c2.twin.a (T)

Deep Copy: c1 := c2.deep_twin



c2 = c2_deep_twin (T)

c2.a = c2_deep_twin.a (T).