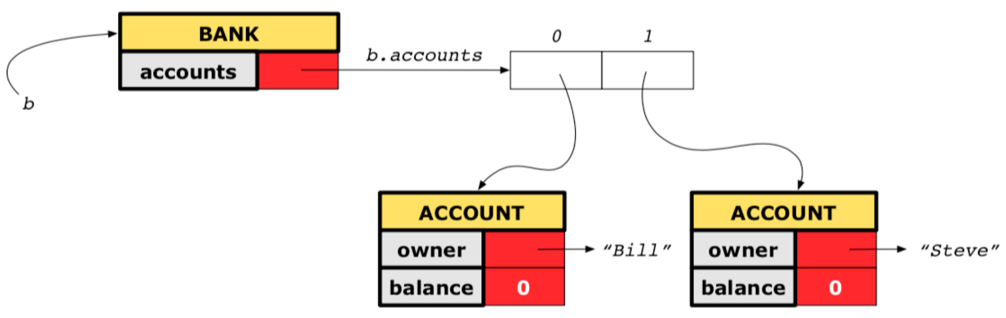
Wednesday January 23

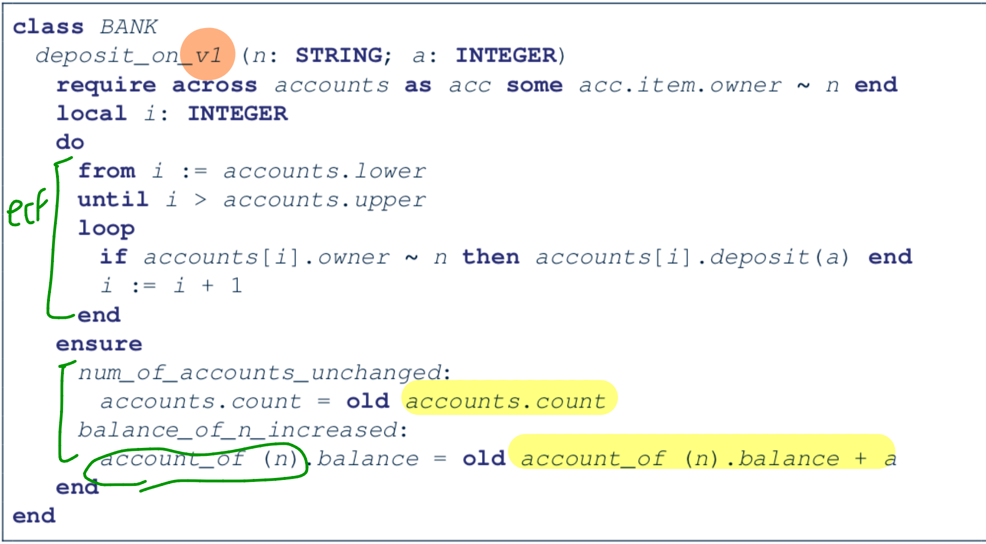
Lecture 6

Version I: Incomplete Contracts, Correct Implementation

b.deposit("Steve", 100)



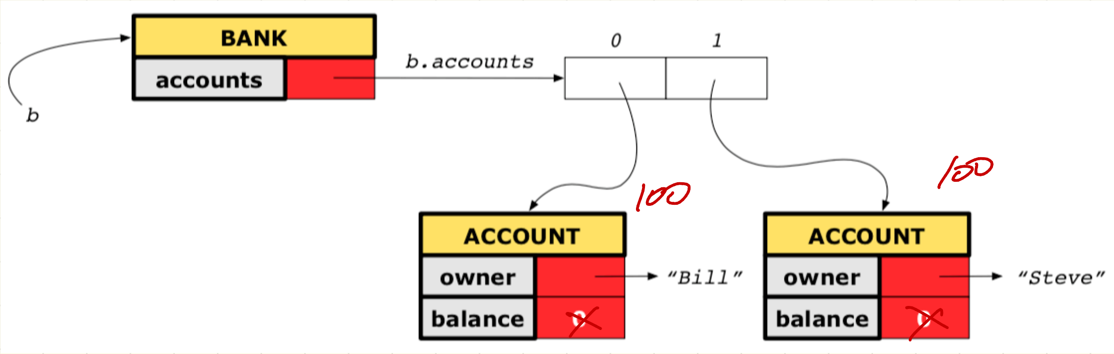
correct



```
class BANK
  deposit_on_v1 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      from i := accounts.lower
      until i > accounts.upper
      loop
        if accounts[i].owner ~ n then accounts[i].deposit(a) end
        i := i + 1
      end
    ensure
      num_of_accounts_unchanged:
        accounts.count = old accounts.count
      balance_of_n_increased:
        account_of (n).balance = old account_of (n).balance + a
    end
end
```

Version 2: Incomplete Contracts, Wrong Implementation

b.deposit("Steve", 100)



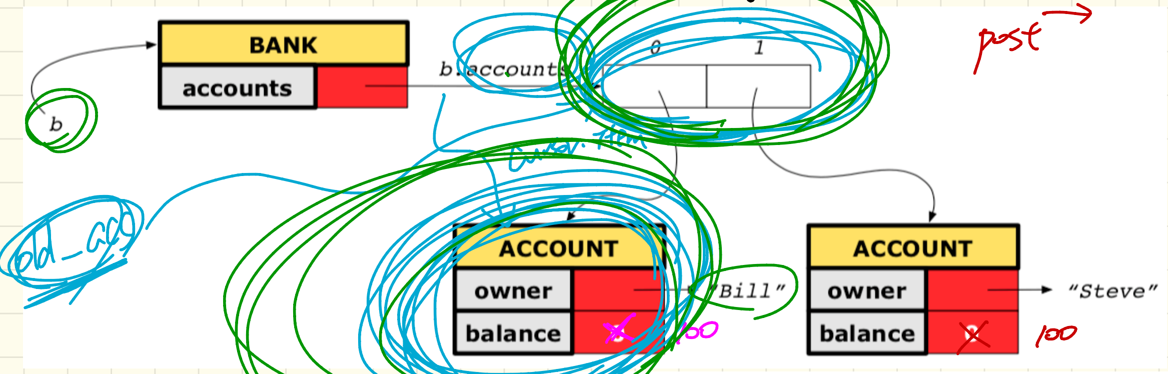
```
class BANK
  deposit_on_v2 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      -- same loop as in version 1

      -- wrong implementation: also deposit in the first account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged:
        accounts.count = old accounts.count
      balance_of_n_increased:
        account_of (n).balance = old account_of (n).balance + a
    end
end
```

Version 3: Complete Contracts, Wrong Implementation

<Reference Copy>

pre → b.deposit("Steve", 100)

post →



old_acc := accounts

```
class BANK
  deposit_on_v3 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      -- same loop as in version 1
      -- wrong implementation: also deposit in the first account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged: accounts.count = old accounts.count
      balance_of_n_increased:
        account_of(n).balance = old account_of(n).balance + a
      others_unchanged :
        across old accounts as cursor
        all cursor.item.owner /~ n implies
          cursor.item ~ account_of (cursor.item.owner)
        end
    end
end
```

old_acc

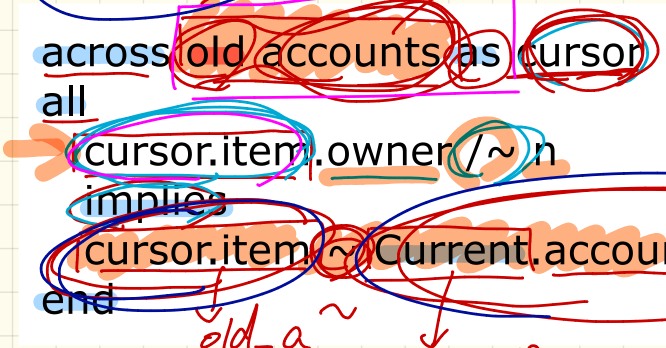
"Bill"

T

Current.

Use of across in Postcondition

Version 1



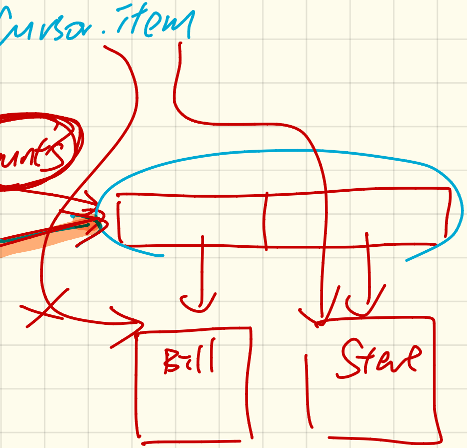
```
across old accounts as cursor
all
  cursor.item.owner /~ n
  implies
  cursor.item ~ Current.account_of(n)
end
```

old_a := accounts

accounts

old_a

cursor.item.owner

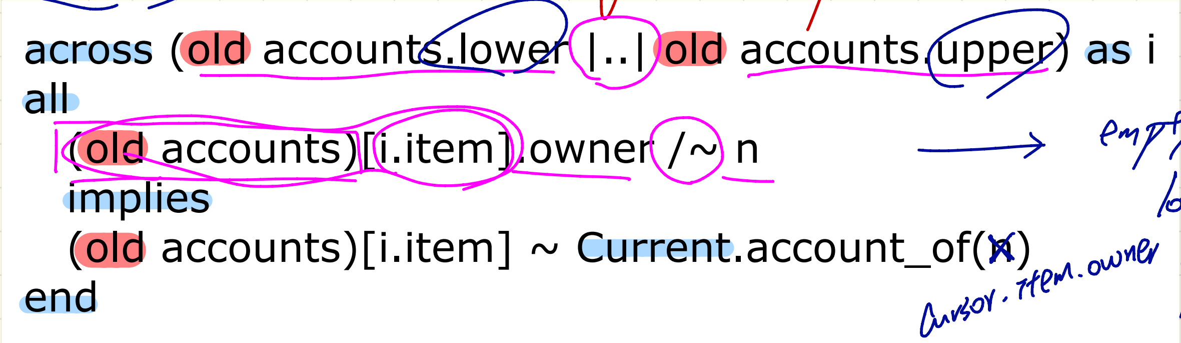


old_a ~

post-state after executing deposit_on tmp.

Cursor.item ~ Current.ac_of (n)

Version 2



```
across (old accounts.lower |..| old accounts.upper) as i
all
  (old accounts)[i.item].owner /~ n
  implies
  (old accounts)[i.item] ~ Current.account_of(n)
end
```

→ empty bank

cursor.item.owner

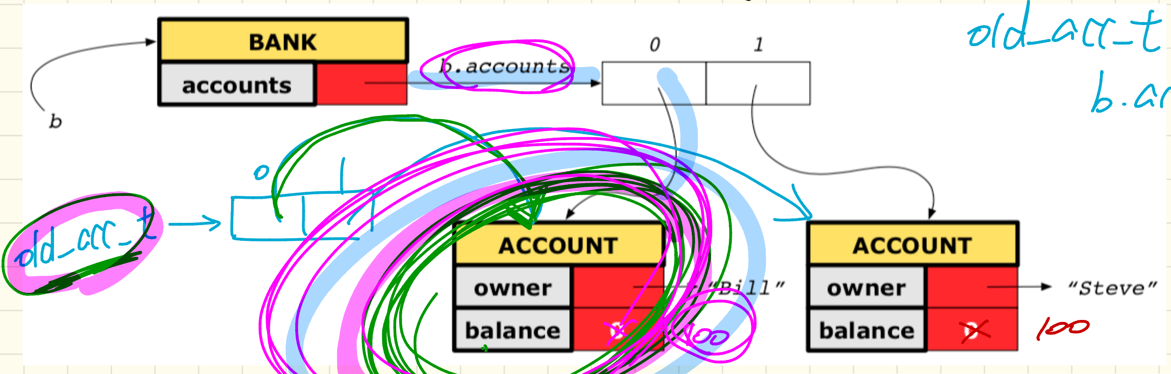
lower	upper
0	-1
1	0

Version 4 : Complete Contracts, Wrong Implementation

< Shallow Copy >

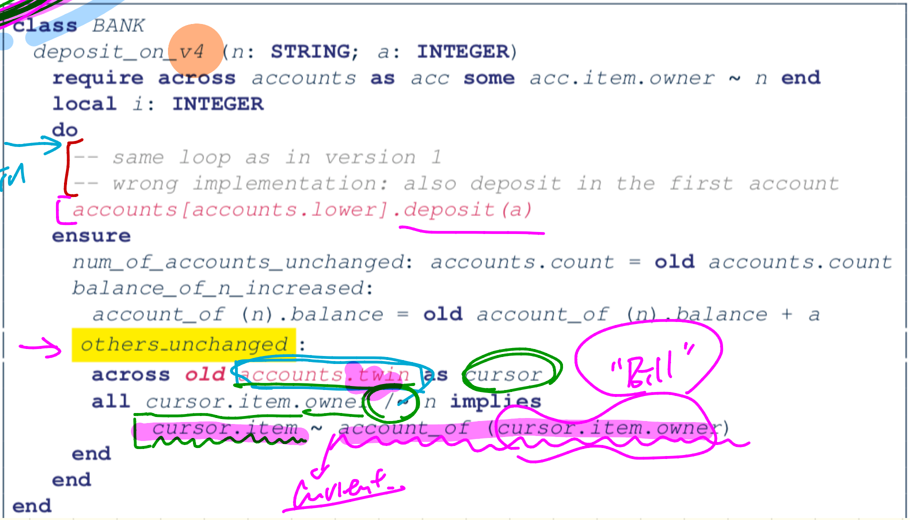
b.deposit("Steve", 100)

old_acc_t[0] := b.accounts[0]



cursor.item

old_acc_t := accounts.twin



```
class BANK
  deposit_on_v4 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      -- same loop as in version 1
      -- wrong implementation: also deposit in the first account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged: accounts.count = old accounts.count
      balance_of_n_increased:
        account_of (n).balance = old account_of (n).balance + a
      others_unchanged :
        across old accounts.twin as cursor
        all cursor.item.owner /~ n implies
          cursor.item ~ account_of (cursor.item.owner)
        end
    end
end
```

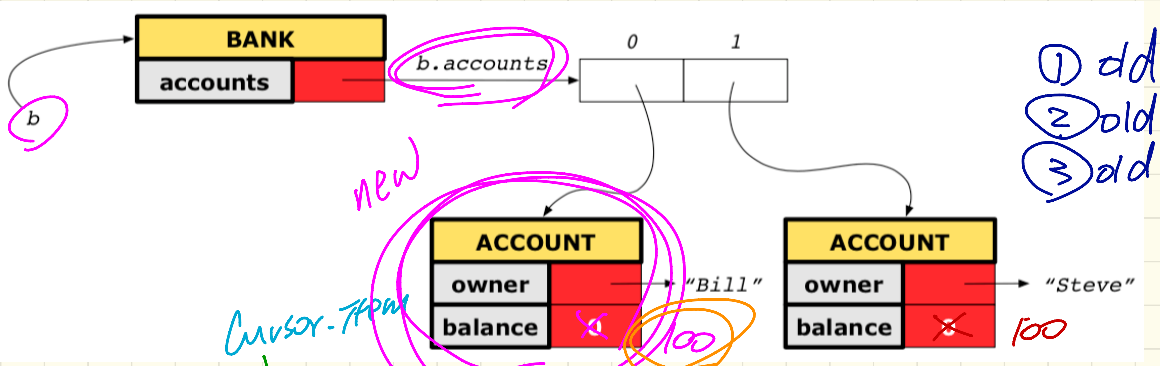
"Bill"

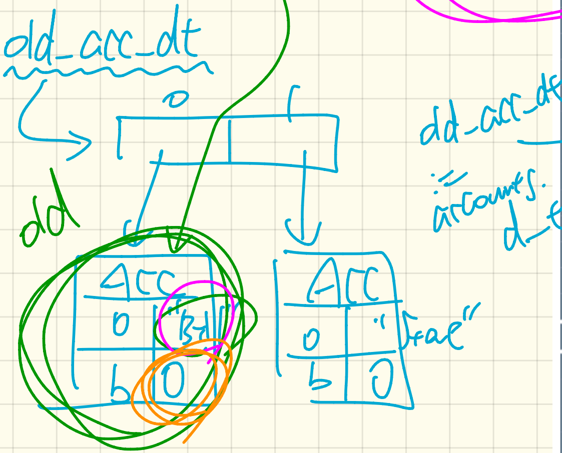
current

Version 5 : Complete Contracts, Wrong Implementation
<Deep Copy>

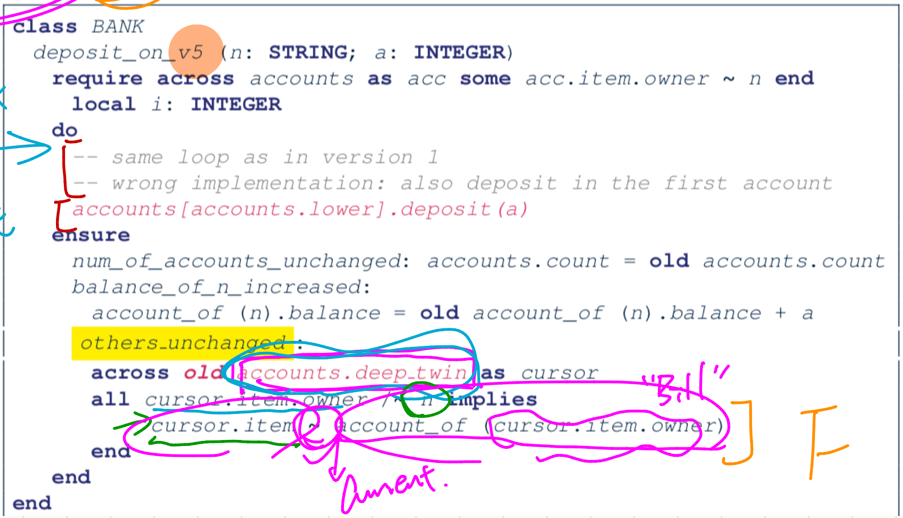
b.deposit("Steve", 100)

① old accounts
② old accounts.twin
③ old accounts.d_t



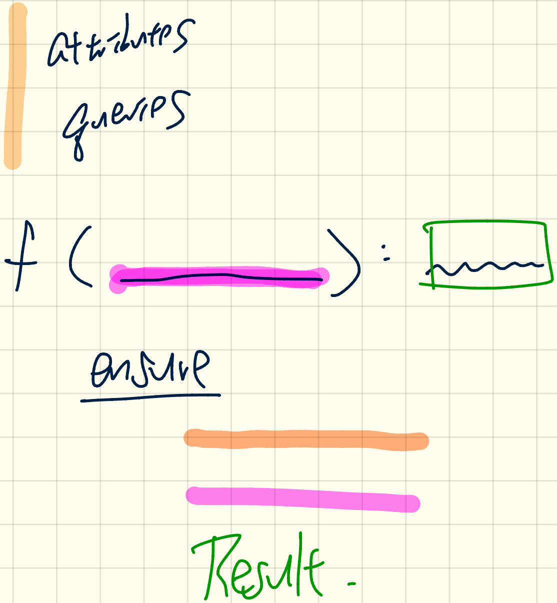


old_acc_dt := accounts.d_t



```
class BANK
  deposit_on_v5 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
     local i: INTEGER
    do
     -- same loop as in version 1
     -- wrong implementation: also deposit in the first account
     accounts[accounts.lower].deposit(a)
    ensure
     num_of_accounts_unchanged: accounts.count = old accounts.count
     balance_of_n_increased:
       account_of (n).balance = old account_of (n).balance + a
     others_unchanged :
       across old accounts.deep_twin as cursor
       all cursor.item.owner /~ n implies
        cursor.item ~ account_of (cursor.item.owner)
       end
    end
end
```

class Foo



attributes

queries

f ( ) :

ensure

Result.

Complete Postcondition: Exercise

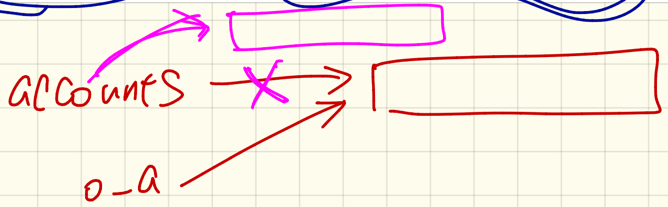
(assuming accounts is not re-assigned) Account

Consider the query *account_of* (*n: STRING*) of *BANK*.

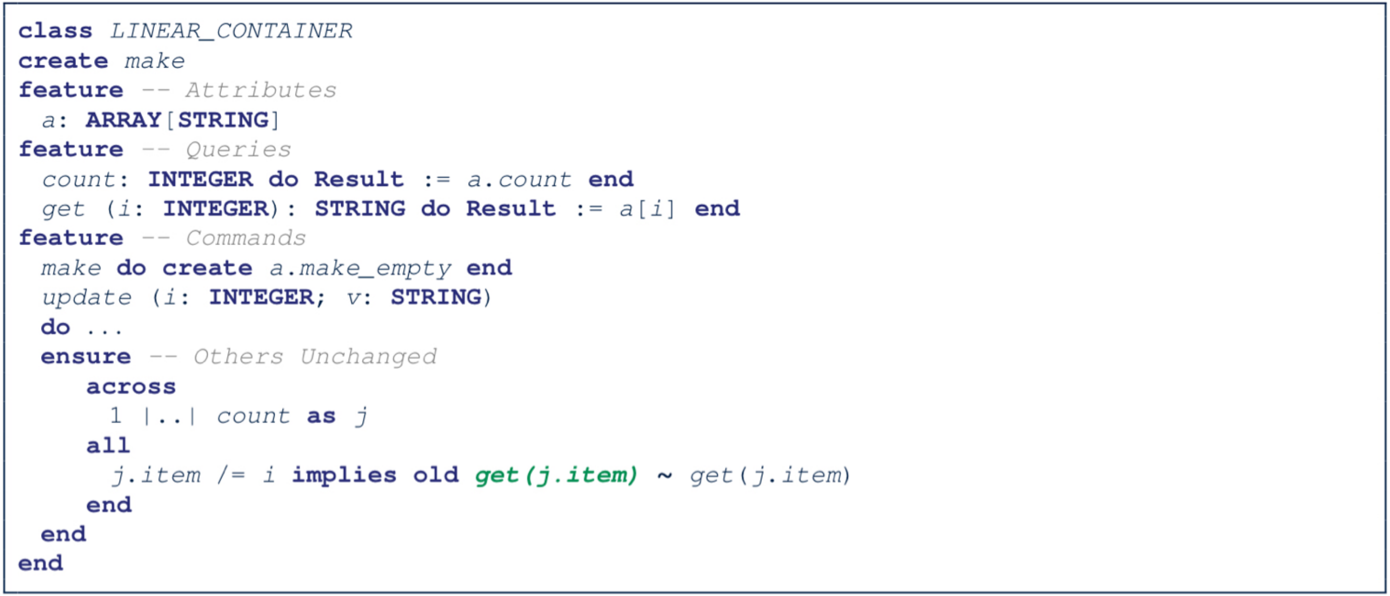
How do we specify (part of) its postcondition to assert that the state of the bank remains unchanged:

trivially true

- ○ `accounts = old accounts` [×] trivially F.
- ○ `accounts = old accounts.twin` [×] t. f.
- ○ `accounts = old accounts.deep_twin` [×]
- ○ `accounts ~ old accounts` [×] t. t.
- ○ `accounts ~ old accounts.twin` [×] only appropriate if the change is at 1st level e.g. accounts[1] := new account.
- ○ `accounts ~ old accounts.deep_twin` [✓]



Use of old in across expression in Postcondition



```
class LINEAR_CONTAINER
create make
feature -- Attributes
  a: ARRAY[STRING]
feature -- Queries
  count: INTEGER do Result := a.count end
  get (i: INTEGER): STRING do Result := a[i] end
feature -- Commands
  make do create a.make_empty end
  update (i: INTEGER; v: STRING)
  do ...
  ensure -- Others Unchanged
      across
        1 |..| count as j
      all
        j.item /= i implies old get(j.item) ~ get(j.item)
      end
  end
end
```

Hint: What value will be cached at runtime
before executing the imp. of update?

Writing Postcondition : Exercise

-2

0

```
is_positive (x: INTEGER) : BOOLEAN
    ensure
        ~~Result := x > 0~~
        x > 0
```

T

0

→ Result := False

-2 > 0

F

Result = (x > 0)

F   T   F

Result := (x * -1) > 0

-2

T

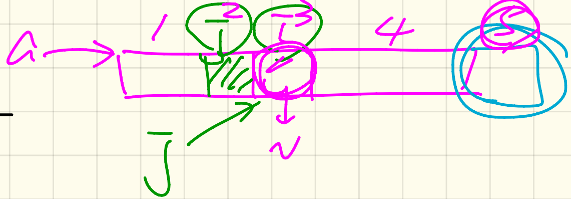
post. con. violate

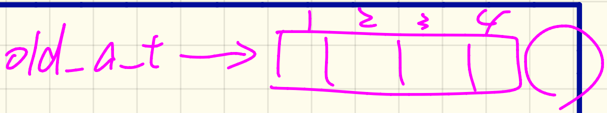
Result implies x > 0

F

ensure = Result ~~:=~~ x > 0

Writing Postcondition : Exercise

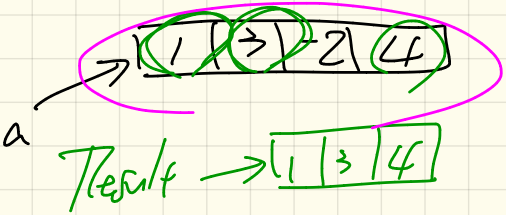




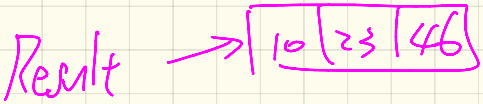
```
a: ARRAY [INTEGER]
change_at (i: INTEGER ; v: INTEGER)
                        { a.force (v, a.count + 1) }
  ensure
①        across a.lower |..| a.upper as j
          all
            j.item = i implies a[j.item] = v
          and
            j.item /= i implies a[j.item] = (old a.twin)[j.item]
②   end
```

a.count = old a.count

Writing Postcondition: Exercise



```
all_positive_values(a: ARRAY[INTEGER]): ARRAY[INTEGER]
    ensure
        across Result as x
        all
            x.item > 0 and a.has(x.item)
        end
```



S — Result vs T — all positive numbers

$$S = T \iff S \subseteq T \land T \subseteq S$$