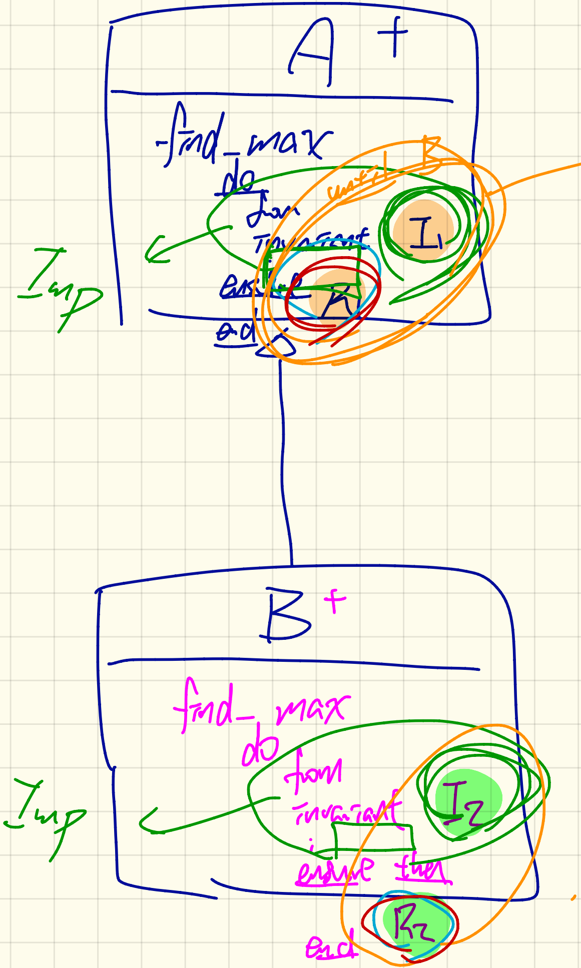
Thursday April 11

Review Lecture



I_1 I_2

$I_1 \wedge B \Rightarrow R_1$

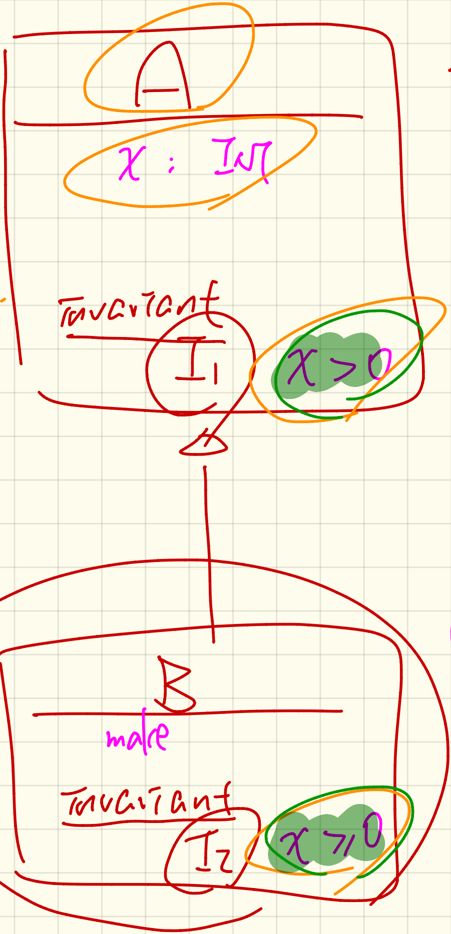
runtime check

$\boxed{R_1 \wedge R_2}$

judge design correctness

$\boxed{R_2 \Rightarrow R_1}$

Justify whether or not the class inv. in A and B are appropriate.



b : B

create b.make

runtime $I_1 \wedge I_2$.

prove $I_2 \Rightarrow I_1$.

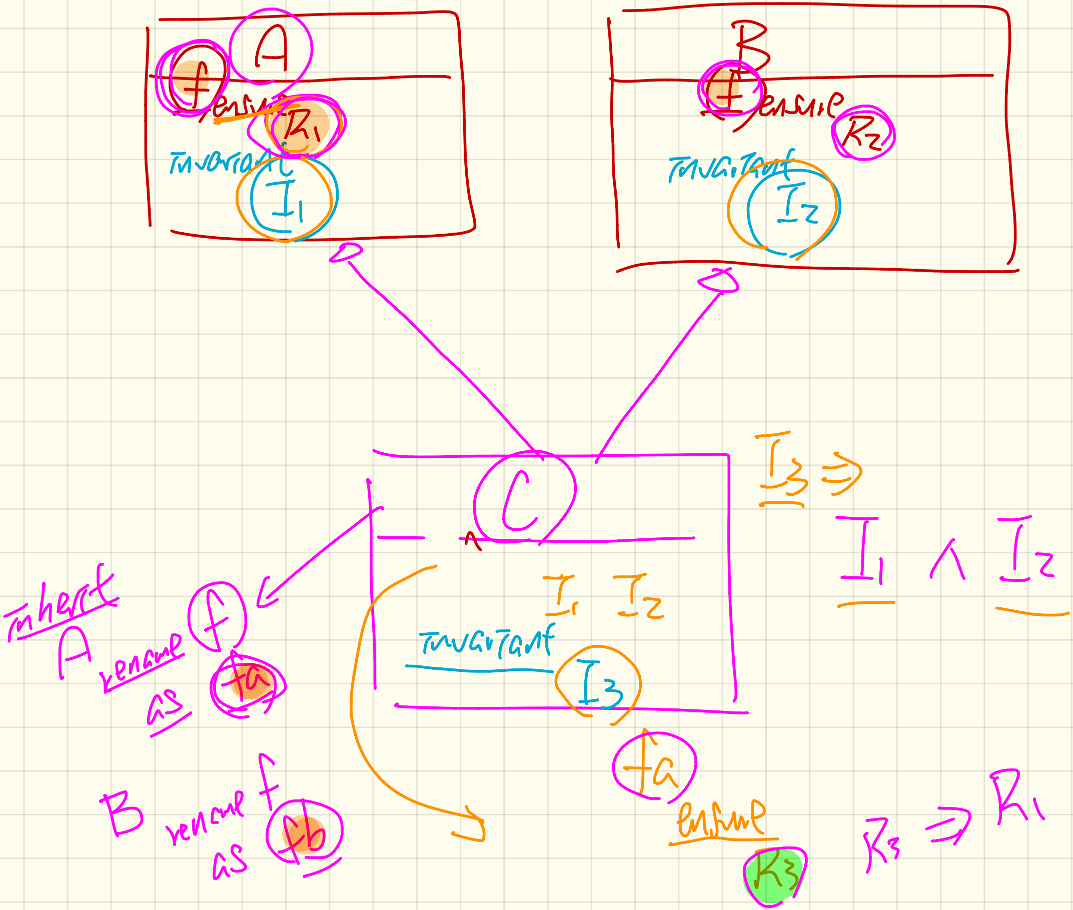
At runtime

↳ $x > 0 \wedge x \geq 0 = x > 0$

Prove: T F

$x \geq 0 \Rightarrow x > 0$

Counter example: $x = 0$.

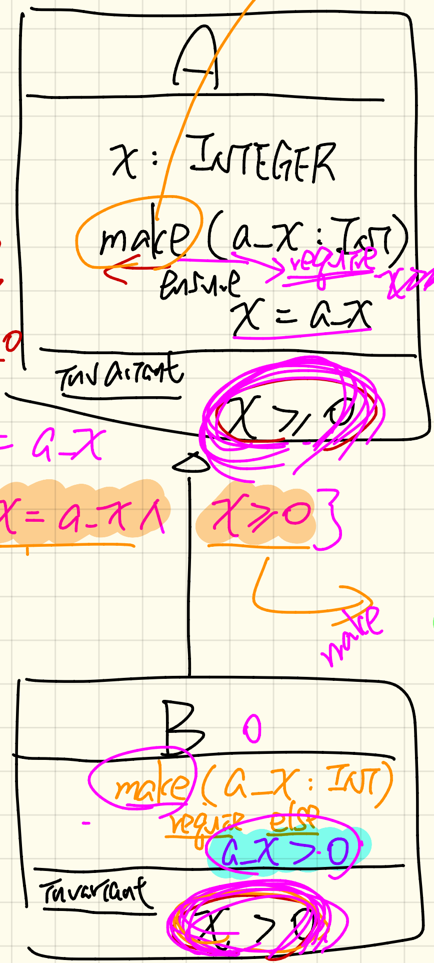


$I_3 \Rightarrow I_1 \wedge I_2$

$R_3 \Rightarrow R_1$

require ??

$a\_x \geq 0$



$\{True\}$ $x := a\_x$

$\{x = a\_x \wedge x \geq 0\}$

require

$a\_x \geq 0$

b : B

Create b. make (0) → CI violation

-- $x \geq 0 \wedge x > 0 \equiv x > 0$

a : A

Create {B} a.make (0)

$x \geq 0$

~~x>0~~

$x \geq 0$

Compiler/runtime assertion monitor

check $[x \geq 0 \wedge x > 0]$

True $\Rightarrow wp(x := a\_x,\ x = a\_x \wedge x \geq 0)$

$$\underbrace{a\_x = a\_x}_{T} \wedge a\_x \geq 0 \equiv a\_x \geq 0$$

True $\Rightarrow a\_x \geq 0$

✗

Counter example: $a\_x = -1$

Runtime

$a\_x \geq 0$

$\vee$

$a\_x > 0$

$\equiv$

$a\_x \geq 0$

$$wp(x := \underline{23},\ x > 22)$$

$=$ { wp rule for assignment

programming assignment

$$wp(\underline{x} := e,\ R) = R[x := e]$$

substitution of free occ. of x.

```
max_of ( x, y: INT) : INT
  require
    x ≠ y
  do
    if x > y then
      Result := x        S1
    else
      Result := y        S2
    end
  ensure
    Result ≥ x ∧ Result ≥ y
```

max_of (4, 2)

S

Result = x ∨ Result = y

Q: Prove or disprove max_of is correct.

1. Formulate program:

$\{x \neq y\}$ if $x > y$ then S_1 else S_2

$\{R \geq x \wedge R \geq y\}$

2. Calculate wp:

$wp($ if $x>y$ then $R := \overset{x+1}{x}$ else $R := y,$

$R \geq x \wedge R \geq y)$

$= \{$ wp rule for alternation $\}$

$x > y \Rightarrow wp(R := \overset{x+1}{\cancel{x}},\ \underset{x+1}{\cancel{R}} \geq x \wedge \underset{x+1}{\cancel{R}} \geq y)$

$\neg(x > y) \Rightarrow wp(R := y,\ \overset{y}{\cancel{R}} \geq x \wedge \overset{y}{\cancel{R}} \geq y)$

$= \{$ wp for assignment twice $\}$

$x > y \Rightarrow \overset{x+1}{\cancel{x}} \geq x \wedge \overset{x+1}{\cancel{x}} \geq y$

$$x > y \Rightarrow \underline{x \geq x} \wedge x \geq y$$

$$\wedge$$

$$x \leq y \Rightarrow y \geq x \wedge \underline{y \geq y}$$

$$= \{ x \geq x \equiv T,\ y \geq y \equiv T,\ T \wedge P \equiv P \}$$

$$\underline{\overset{4}{x} > \overset{3}{y} \Rightarrow \overset{4}{x} \geq \overset{3}{y}\quad T}$$

$$\wedge$$

$$x \leq y \Rightarrow y \geq x \,]\, T$$

$$= \{ \text{arithmetic},\ T \wedge T \equiv T \}$$

$$\boxed{T} \longrightarrow \boxed{wp(prog, R)}$$

$$\underline{x > y} \Rightarrow \boxed{\overset{T}{\underline{x+1 \geq x}}} \wedge \underline{\underset{T}{x+1 \geq y}}$$

3. Prove:

$$\boxed{x \neq y} \Rightarrow \overset{F}{\boxed{T}} \equiv \overset{F}{T}$$

∴ Program is correct.

a and then b

$p \wedge q \wedge r$

$\equiv p \wedge r \wedge q$

&&

a and b

✓

$x \neq 0$ and then $y/x > 2$

$\wedge$ $\wedge$

v1 a.lower $\leq$ i and ~~then~~ i $\leq$ a.upper and then a[i] > 3

$\wedge$ $\wedge$

v2 i $\leq$ a.upper and then a[i] > 3 and then a.lower $\leq$ i

0

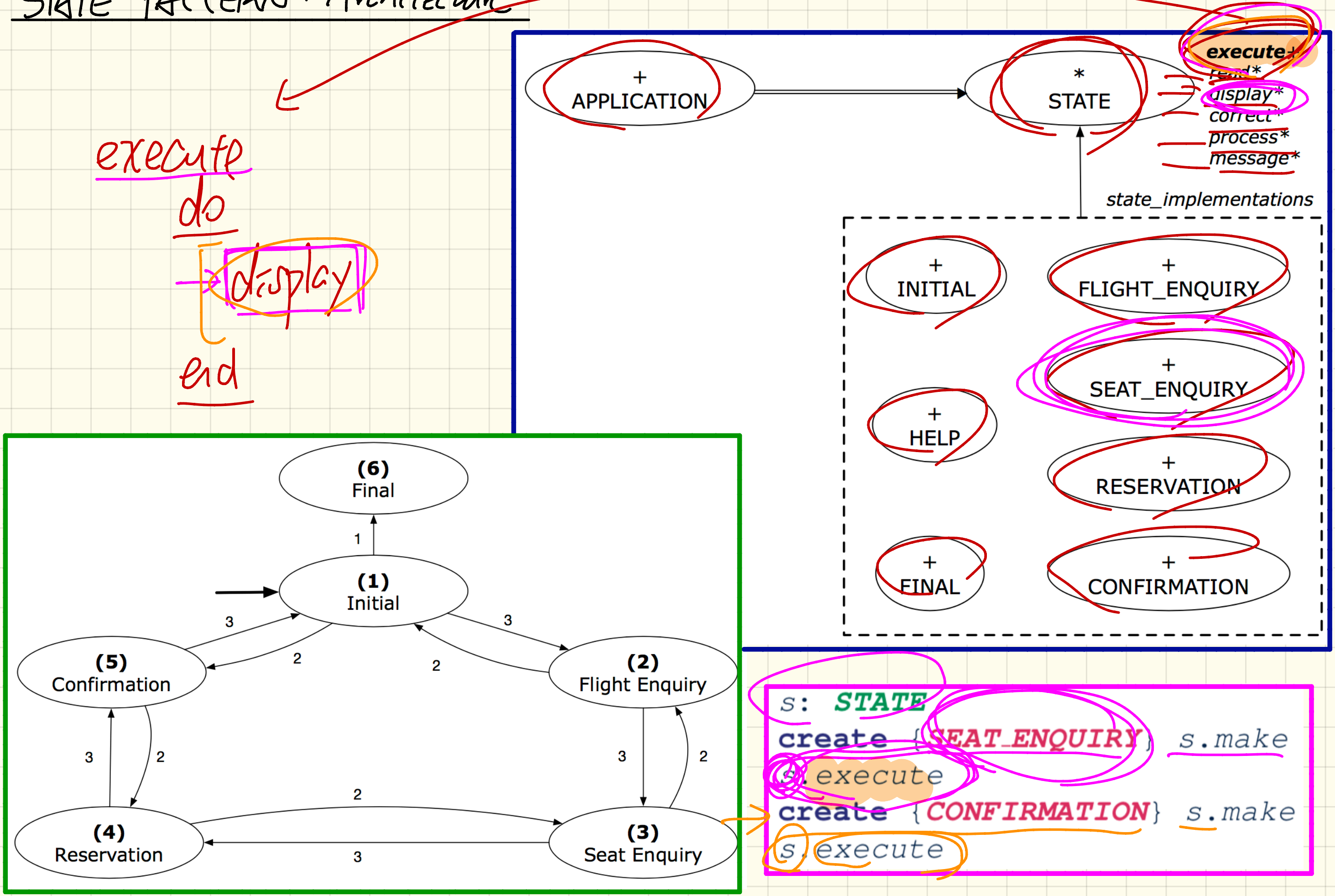
$\overset{T}{b1}$ and then $b2$ → evaluated only if b1 is T

$b1$ or else $b2$ → evaluated only if b1 is F

$F \vee P \equiv P$

STATE PATTERN : Architecture

```
execute
    do
        display
    end
```



```
s: STATE
create {SEAT_ENQUIRY} s.make
s.execute
create {CONFIRMATION} s.make
s.execute
```

Weather Station: Testing the Observer Pattern

```
class WEATHER_STATION create make
feature -- Attributes
  cc: CURRENT_CONDITIONS ; fd: FORECAST ; sd: STATISTICS
  wd: WEATHER_DATA
feature -- Commands
  make
    do create wd.make (9, 75, 25)
      create cc.make (wd) ; create fd.make (wd) ; create sd.make(wd)

      wd.set_measurements (15, 60, 30.4)
      wd.notify
      cc.display ; fd.display ; sd.display
      cc.display ; fd.display ; sd.display

      wd.set_measurements (11, 90, 20)
      wd.notify
      cc.display ; fd.display ; sd.display
  end
end
```

class WEATHER_DATA

fd

up to date

```
class FORECAST
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
    do weather_data := a_weather_data
       weather_data.attach (Current)
    ensure weather_data = a_weather_data
           weather_data.observers.has (Current)
    end
```

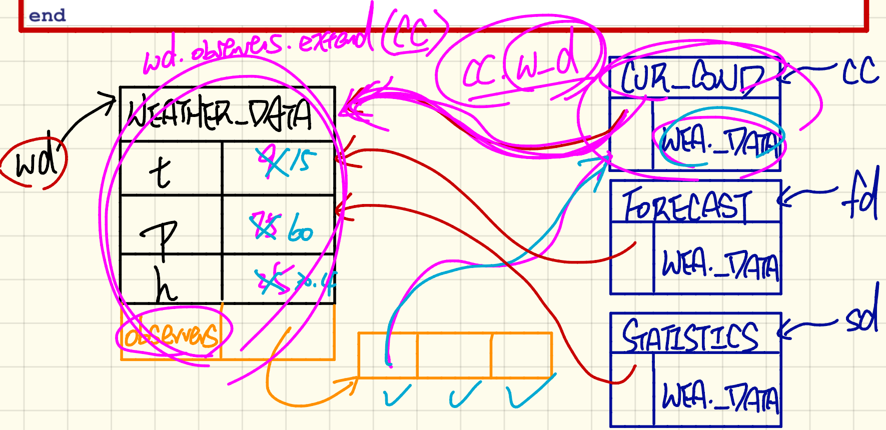
```
class CURRENT_CONDITIONS
inherit OBSERVER
feature -- Commands
  make(~~a_weather_data: WEATHER_DATA~~)
    do weather_data := ~~a_weather_data~~
       ~~weather_data~~.attach (Current)
    ensure weather_data = a_weather_data
           weather_data.observers.has (Current)
    end
```

wd

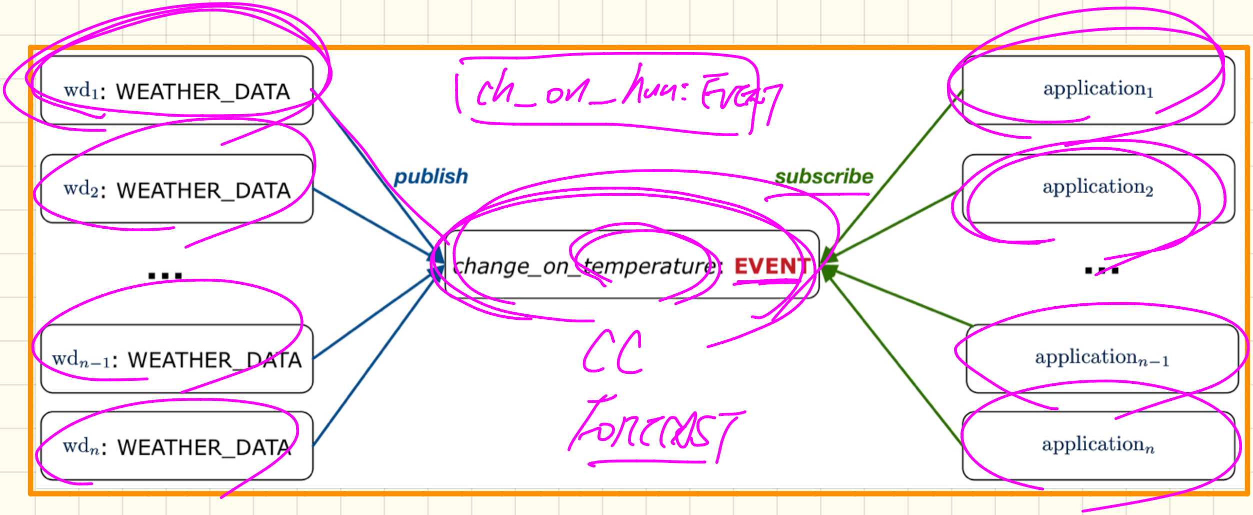
wd

wd

```
class STATISTICS
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
    do weather_data := a_weather_data
       weather_data.attach (Current)
    ensure weather_data = a_weather_data
           weather_data.observers.has (Current)
    end
```



Event-Driven Design: Multiple Subjects and Observers



wd_1: WEATHER_DATA
wd_2: WEATHER_DATA
...
wd_{n-1}: WEATHER_DATA
wd_n: WEATHER_DATA
publish
ch_on_hum: EVENT
change_on_temperature: EVENT
CC
FORECAST
subscribe
$\mathrm{application}_1$
$\mathrm{application}_2$
...
$\mathrm{application}_{n-1}$
$\mathrm{application}_n$

Complexity ?

Adding a new subject? Adding a new observer?

Adding a new event type?

Event-Driven Design in Eiffel

```
class WEATHER_STATION create make
feature
  cc: CURRENT_CONDITIONS
  make
    do create wd.make (9, 75, 25)
       create cc.make (wd)
       wd.set_measurements (15, 60, 30.4)
       cc.display
       wd.set_measurements (11, 90, 20)
       cc.display
    end
end
```

```
class CURRENT_CONDITIONS
create make
feature -- Initialization
  make(wd: WEATHER_DATA)
    do
      wd.change_on_temperature.subscribe (agent update_temperature)
      wd.change_on_temperature.subscribe (agent update_humidity)
    end
feature
  temperature: REAL
  humidity: REAL
  update_temperature (t: REAL) do temperature := t end
  update_humidity (h: REAL) do humidity := h end
  display do ... end
end
```

humidity

agent update_hum(?)

agent u_f(?, ?, ?, _)

```
class WEATHER_DATA
create make
feature -- Measurements
  temperature: REAL ; humidity: REAL ; pressure: REAL
  correct_limits(t,p,h: REAL): BOOLEAN do ... end
  make (t, p, h: REAL) do ... end
feature -- Event for data changes
  change_on_temperature : EVENT[TUPLE[REAL]]once create Result end
  change_on_humidity : EVENT[TUPLE[REAL]]once create Result end
  change_on_pressure : EVENT[TUPLE[REAL]]once create Result end
feature -- Command
  set_measurements(t, p, h: REAL)
    require correct_limits(t,p,h)
    do temperature := t ; pressure := p ; humidity := h
       change_on_temperature .publish ([t])
       change_on_humidity .publish ([p])
       change_on_pressure .publish ([h])
    end
invariant correct_limits(temperature, pressure, humidity) end
```

```
class EVENT [ARGUMENTS -> TUPLE ]
create make
feature -- Initialization
  actions: LINKED_LIST[PROCEDURE[ARGUMENTS]]
  make do create actions.make end
feature
  subscribe (an_action: PROCEDURE[ARGUMENTS])
    require action_not_already_subscribed: not actions.h
    do actions.extend (an_action)
    ensure action_subscribed: action.has(an_action) end
  publish (args: G)
    do from actions.start until actions.after
       loop actions.item.call (args) ; actions.forth end
    end
end
```

[ ]

[t]

[t1, t2]

PROCEDURE

actions.item(args) ✓