



Einführung in die Programmierung Introduction to Programming

Prof. Dr. Bertrand Meyer

Exercise Session 6



Mock exam in 2 weeks (November 10)

- You have to be present
- No assignment that week
- The week after we will discuss the results

Today



- Conditional
- Loop
- Linked list

Inside the routine body



- The body of each routine consists of instructions (command calls, creations, assignments, etc.)
- In the programs you've seen so far they were always executed in the order they were written

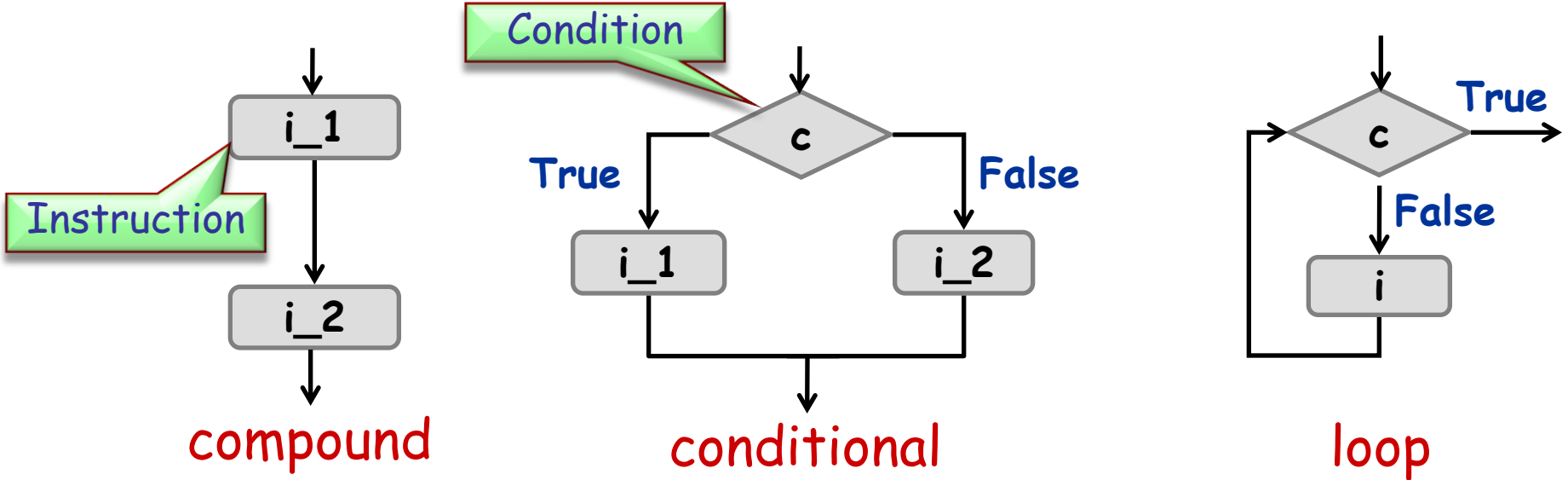
- ➡ *create passenger.make_with_route (Route3, 1.5)*
- ➡ *passenger.go*
- ➡ *passenger.set_reiterate (True)*
- ➡ *Paris.put_passenger (passenger)*
- ➡ *create tram.make_with_line (Line1)*
- ➡ *tram.start*
- ➡ *Paris.put_tram (tram)*

- Programming languages have structures that allow you to change the order of execution

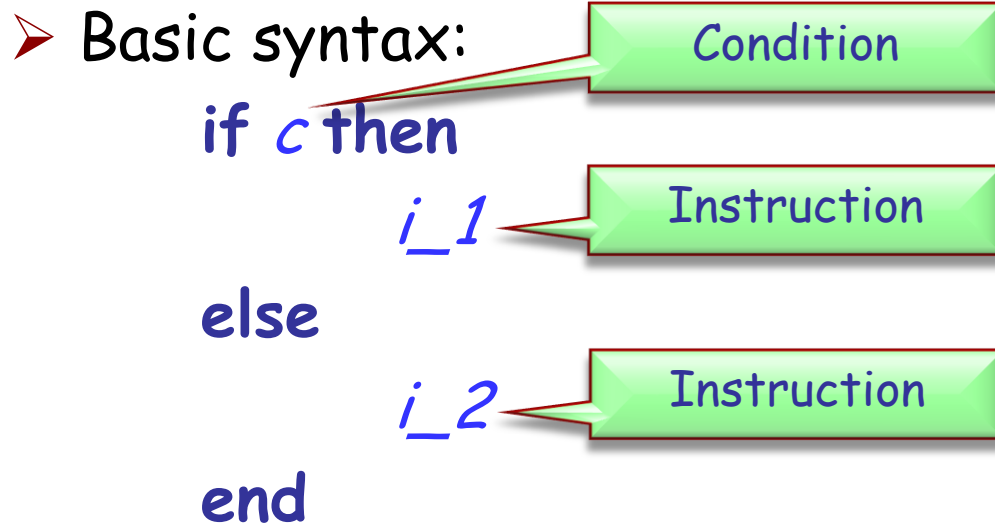
Structured programming



- If the order of execution could be changed arbitrarily, it would be hard to understand programs
- In **structured programming** instructions can be combined only in three ways:



- Each of these blocks has a single entrance and exit and is itself an instruction



➤ *c* is a boolean expression (e.g., entity, query call of type *BOOLEAN*)

➤ *else*-part is optional:

```
if c then
    i_1
end
```

Compilation error? Runtime error? (1)



Hands-On

```
f(x, y: INTEGER): INTEGER
```

```
do
```

```
if (x // y) then
```

```
1
```

```
else
```

```
0
```

```
end
```

```
end
```

Compilation error:
integer expression
instead of boolean

Compilation error:
expression instead of
instruction

Compilation error? Runtime error? (2)



Hands-On

```
f(x, y: INTEGER): INTEGER
do
  if (False) then
    Result := x // y
  end
  if (x /= 0) then
    Result := y // x
  end
end
```

Everything is OK
(during both compilation
and runtime)

Calculating function's value



Hands-On

```
f(max: INTEGER; s: STRING): STRING
do
    if s.is_equal("Java") then
        Result := "J**a"
    else
        if s.count > max then
            Result := "<an unreadable German word>"
        end
    end
end
end
```

Calculate the value of:

- $f(3, \text{"Java"}) \rightarrow \text{"J**a"}$
- $f(20, \text{"Immatrikulationsbestätigung"}) \rightarrow \text{"<an unreadable German word>"}$
- $f(6, \text{"Eiffel"}) \rightarrow \text{Void}$

What does this routine do?



```
abs (x: REAL): REAL
  do
    if (x >= 0) then
      Result := x
    else
      Result := -x
    end
  end
end
```

Hands-On

Write a routine...



Hands-On

- ... that computes the maximum of two integers:

```
max(a, b: INTEGER): INTEGER
```

- ... that increases time by one second inside class *TIME*:

```
class TIME
```

```
    hour, minute, second: INTEGER
```

```
    second_forth
```

```
        do ... end
```

```
    ...
```

```
end
```

Comb-like conditional



If there are more than two alternatives, you can use the syntax:

```
    Condition
if c1 then
    i1
elseif c2 then
    i2
...
elseif c_n then
    i_n
else
    i_e
end
```

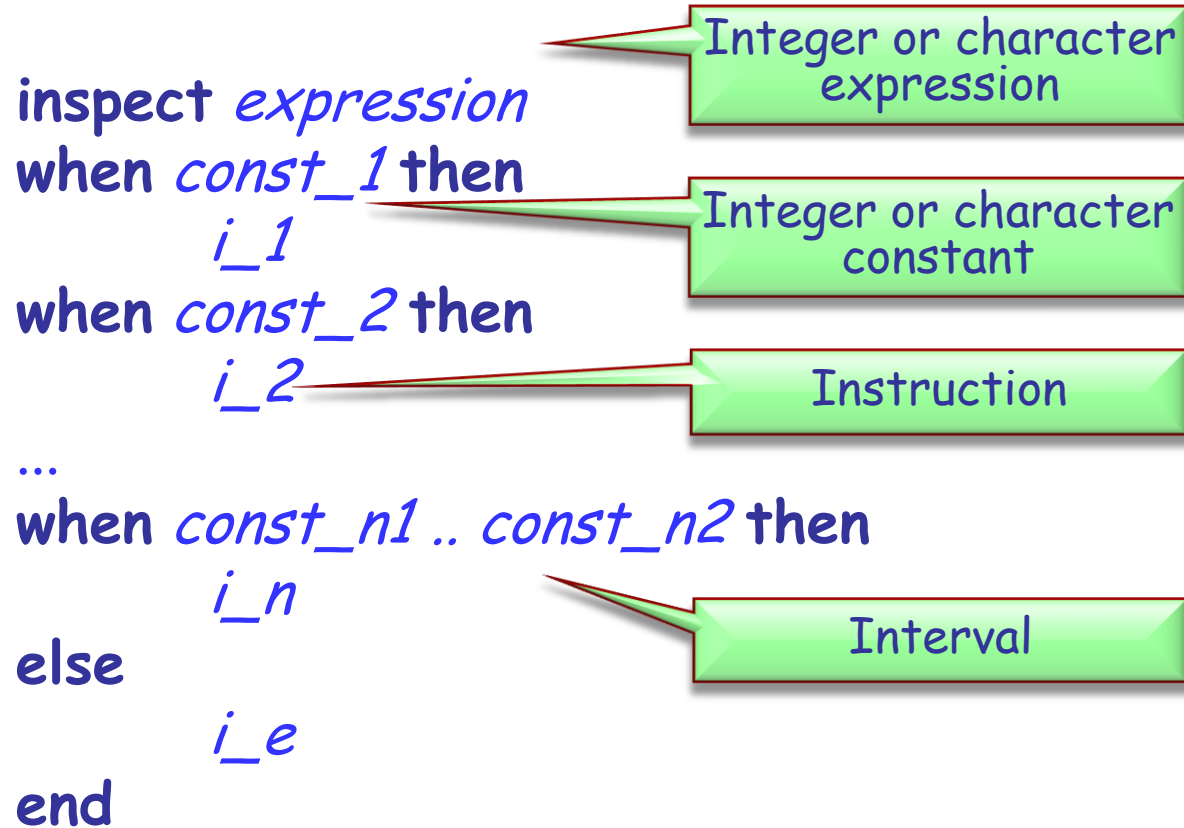
instead of:

```
if c_1 then
    i_1
else
    if c_2 then
        i_2
    else
        ...
        if c_n then
            i_n
        else
            i_e
        end
    end
end
end
```

Multiple choice



If all the conditions have a specific structure, you can use the syntax:





Rewrite the following multiple choice:

- using a comb-like conditional
- using nested conditionals

```
inspect user_choice
when 0 then
    print ("Here is your hamburger")
when 1 then
    print ("Here is your Coke")
else
    print ("Sorry, not on the menu today!")
end
```

Lost in conditions: solution



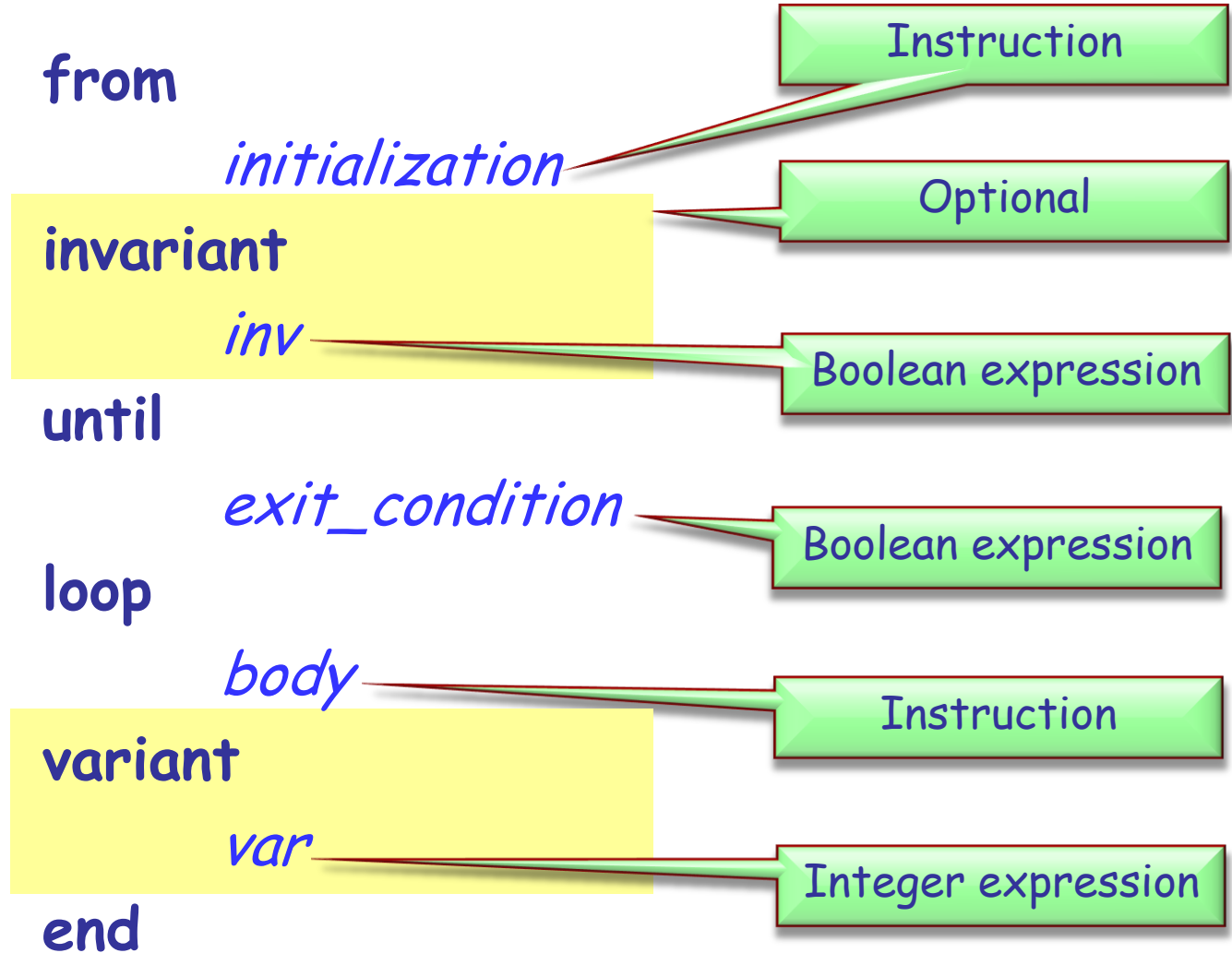
```
if user_choice = 0 then
    print ("Here is your hamburger")
elseif user_choice = 1 then
    print ("Here is your Coke")
else
    print ("Sorry, not on the menu today!")
end
```

```
if user_choice = 0 then
    print ("Here is your hamburger")
else
    if user_choice = 1 then
        print ("Here is your Coke")
    else
        print ("Sorry, not on the menu today!")
    end
end
```

Loop



Syntax:



Simple loop (1)



Hands-On

How many times will the body of the following loop be executed?

i: INTEGER

...

from

i := 1

In Eiffel we usually start counting from 1

until

i > 10

10

loop

print ("I will not say bad things about assistants")

i := i + 1

end

Simple loop (2)



Hands-On

And what about this one?

i: INTEGER

...

from

i := 10



until

i < 1



loop

print ("I will not say bad things about assistants")

end

What does this function do?



Hands-On

factorial (*n*: *INTEGER*): *INTEGER* is

local

i: *INTEGER*

do

from

i := 2

Result := 1

until

i > *n*

loop

Result := **Result** * *i*

i := *i* + 1

end

end



Loop invariant (do not mix with class invariant)

- holds after execution of **from** clause and after each execution of **loop** clause
- captures how the loop iteratively solves the problem: e.g. "to calculate the sum of all n elements in a list, on each iteration i ($i = 1..n$) the sum of first i elements is obtained"

Loop variant

- integer expression that is nonnegative after execution of **from** clause and after each execution of **loop** clause and strictly decreases with each iteration
- a loop with a variant can not be infinite (why?)

Invariant and variant



Hands-On

What are the invariant and variant of the “factorial” loop?

from

$i := 2$

Result := 1

invariant

Result = $factorial(i - 1)$

Result = 6 = 3!

until

$i > n$

loop

Result := Result * i

$i := i + 1$

variant

$n - i + 2$

end

Writing loops



Hands-On

Implement a function that calculates Fibonacci numbers, using a loop

fibonacci (*n*: INTEGER): INTEGER

-- *n*-th Fibonacci number

require

n_non_negative: $n \geq 0$

ensure

first_is_zero: $n = 0$ implies Result = 0

second_is_one: $n = 1$ implies Result = 1

other_correct: $n > 1$ implies Result =
 $fibonacci(n - 1) + fibonacci(n - 2)$

end

Writhing loops (solution)



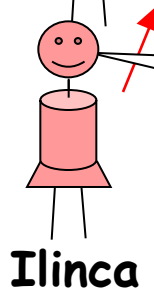
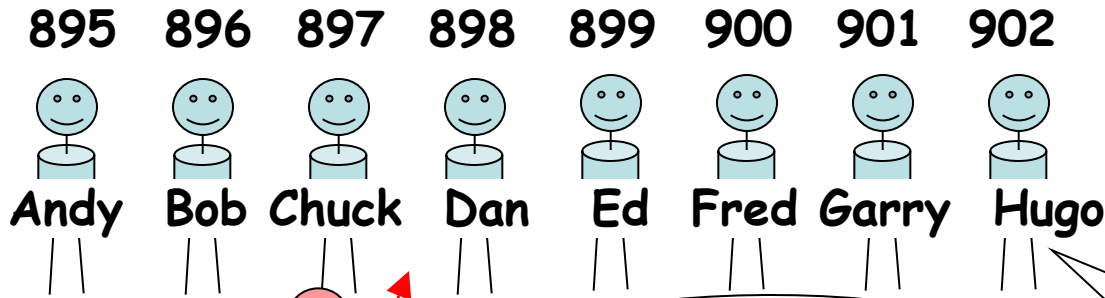
Hands-On

```
fibonacci(n: INTEGER): INTEGER
  local
    a, b, i: INTEGER
  do
    if n <= 1 then
      Result := n
    else
      from
        a := fibonacci(0)
        b := fibonacci(1)
        i := 1
      invariant
        a = fibonacci(i - 1)
        b = fibonacci(i)
      until
        i = n
      loop
        Result := a + b
        a := b
        b := Result
        i := i + 1
      variant
        n - i
    end
  end
end
```

Two kinds of queues



Electronic queue (like in the post office)



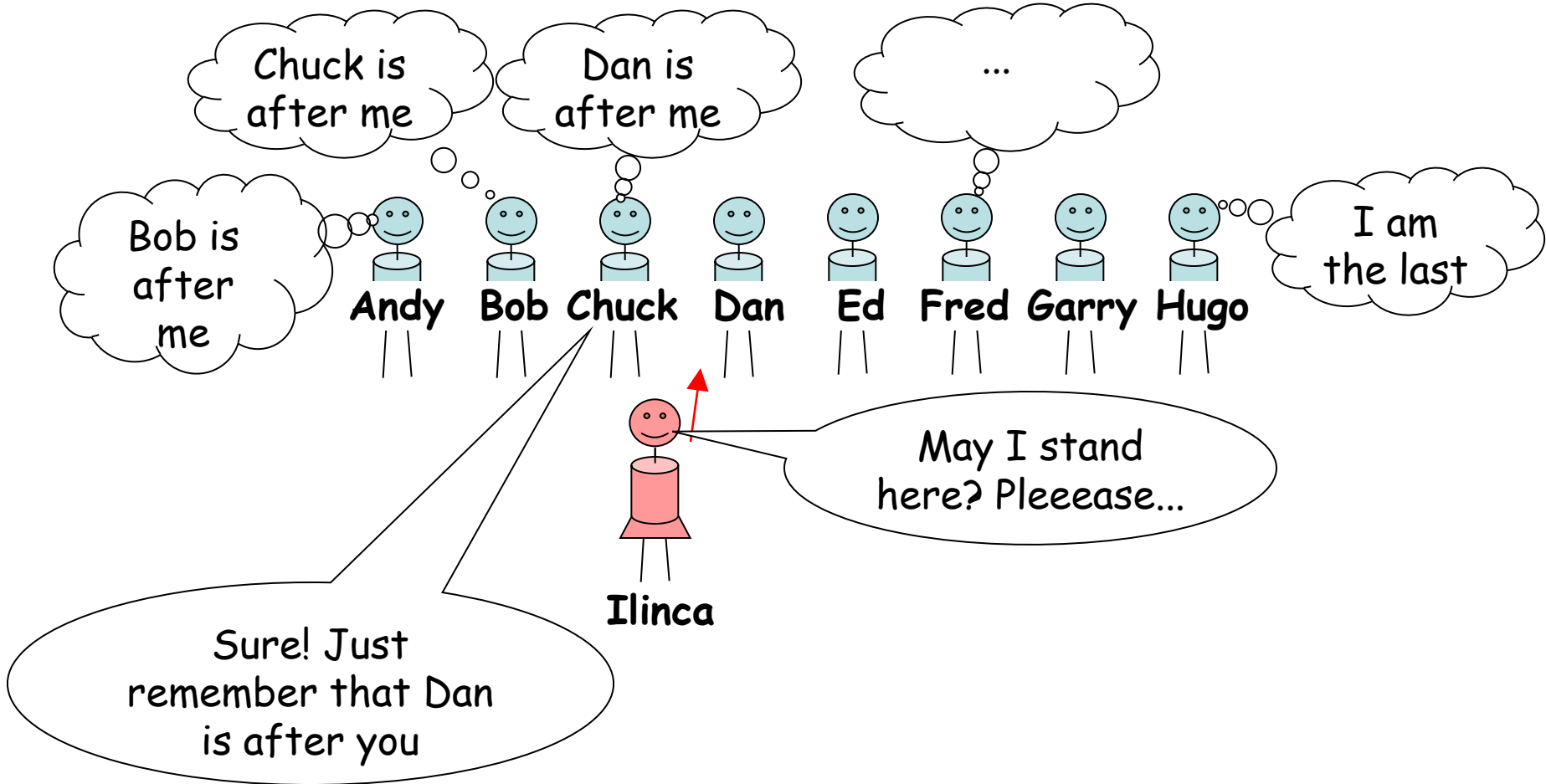
May I stand here? Pleeese...

No! We'll have to take tickets again!

Two kinds of queues



Live queue



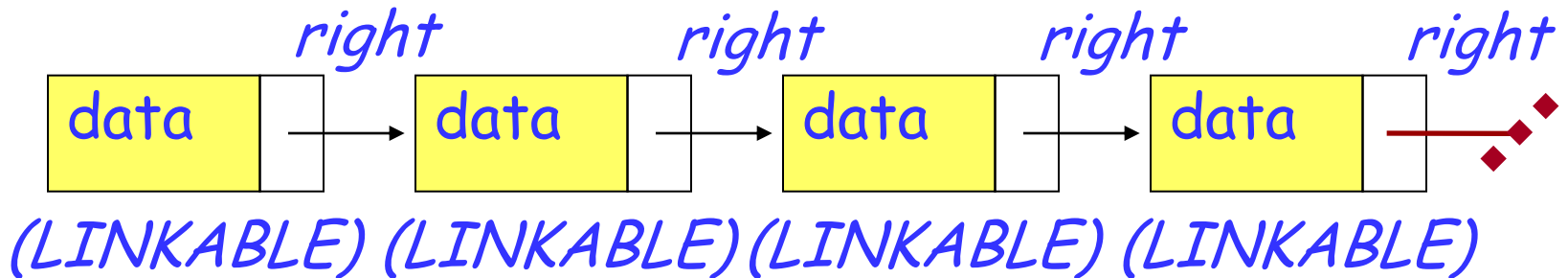
To make it possible to link infinitely many similar elements together, each element should contain a reference to the next element

```
class LINKABLE  
  feature
```

```
    ...
```

```
    right: LINKABLE
```

```
end
```



INT_LINKABLE



class INT_LINKABLE

create put

feature

item: INTEGER

put (i: INTEGER)

do item := i end

right: INT_LINKABLE

put_right (other: INT_LINKABLE)

do right := other end

end

INT_LINKED_LIST



class INT_LINKED_LIST

feature

first_element: INT_LINKABLE

-- First cell of the list

last_element: INT_LINKABLE

-- Last cell of the list

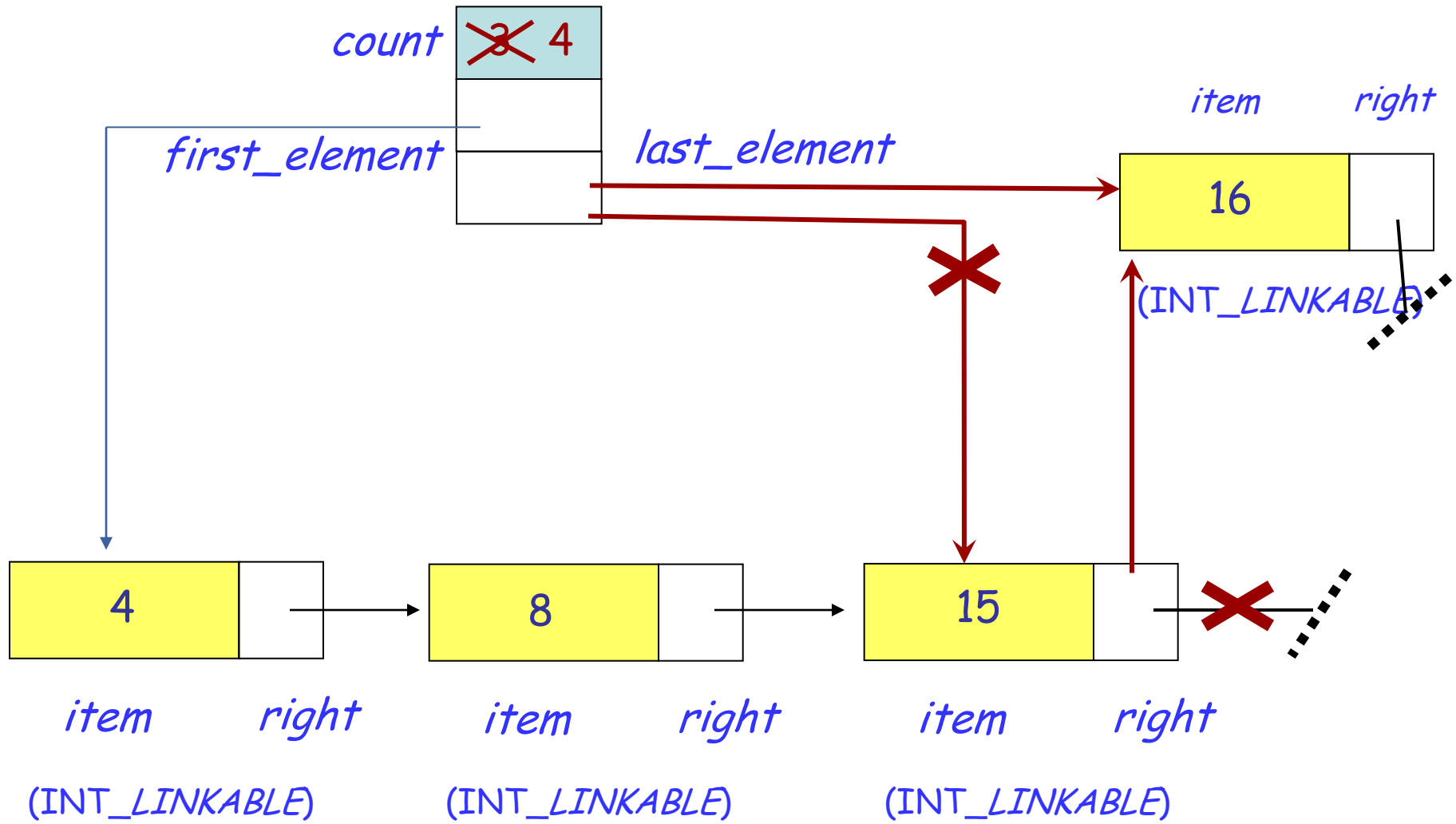
count: INTEGER

-- Number of elements in the list

...

end

INT_LINKED_LIST: inserting at the end



INT_LINKED_LIST: inserting at the end



```
extend(v: INTEGER)  
    -- Add v to end.  
    local  
        new: INT_LINKABLE  
    do  
        create new.put(v)  
        if first_element = Void then  
            first_element := new  
        else  
            last_element.put_right(new)  
        end  
        last_element := new  
        count := count + 1  
    end  
end
```

INT_LINKED_LIST: search



has (v. INTEGER): BOOLEAN

-- Does list contain v?

local

temp: INT_LINKABLE

do

from

temp := first_element

until

(temp = Void) or Result

loop

if temp.item = v then

Result := True

end

temp := temp.right

end

end

Write a routine that

- calculates the sum of all positive values in a list

```
sum_of_positive: INTEGER  
do ... end
```

- inserts an element after the first occurrence of a given value and does nothing if the value is not found

```
insert_after(i, j: INTEGER)  
do ... end
```

INT_LINKED_LIST: sum_of_positive



```
sum_of_positive: INTEGER
  -- Some of positive elements
local
  temp: INT_LINKABLE
do
  from
    temp := first_element
  until
    temp = Void
  loop
    if temp.item > 0 then
      Result := Result + temp.item
    end
    temp := temp.right
  end
end
```

INT_LINKED_LIST: insert_after



```
insert_after (i, j: INTEGER)
    -- Insert `j` after `i` if present
    local
        temp, new: INT_LINKABLE
    do
        from
            temp := first_element
        until
            temp = Void or else temp.item = i
        loop
            temp := temp.right
        end
        if temp /= Void then
            create new.put (j)
            new.put_right (temp.right)
            temp.put_right (new)
            count := count + 1
            if temp = last_element then
                last_element := new
            end
        end
    end
end
```