ELTE | FACULTY OF INFORMATICS

PROGRAMMING

Lecture 8

Zsuzsa Pluhár

pluharzs@inf.elte.hu

Content

- Construction by copy
- Selection + summation
- Selection+ maximum selection
- Maximum selection + multiple item selection
- Decision+ counting
- Decision+ decision
- Sequence calculations for matrix
- Decision for matrix

Construction by copy

Specification

Input: $X[1..] \in \mathbb{S}^*$, $f: \mathbb{S}_1 \rightarrow \mathbb{S}_2$,

Output: $Y[1..] \in \mathbb{S}_2^{\ *}$

Precondition: –

Postcondition:
$\forall i(1 \leq i \leq length(X)): Y[i]=f(X[i])$

Construction by copy works for all patterns of algorithms.

Instead of the X_i elements of the $X \in \mathbb{S}$ sequence in the input, you only have to write $f(X[i])$,

… in the **output**:

$$\sum_{i=1}^{length(X)} X[i] \rightarrow \sum_{i=1}^{length(X)} f(X[i])$$

Construction by copy

The copy PoA had, however, a version that gives way to new opportunities:

Postcondition: $\forall$`i(1≤i≤length(X)): Y[p(i)]=X[i]` where `p(i)` could be eg. `length(X)-i+1`, which means **reversing** the order of elements of the sequence.

Many PoAs make use of the order of elements, eg. it found the first among the possible solutions or gave all the expected elements in the order of input.

With this construction, you could process the sequence backwards.

Copy + Search

Specification

Input: `X[1..]`$\in\mathbb{S}^*$, `A:`$\mathbb{S}\to\mathbb{L}$

Output: **exists**$\in\mathbb{L}$, **ind**$\in\mathbb{N}$

Precondition: –

Postcondition: exists=(∃i(1≤i≤length(X)):A(X[i])) and exists → 1≤ind≤length(X) and A(X[ind])

Task: Find the **last** element that has a certain attribute.

Specification:

Input: `X[1..]`$\in\mathbb{S}^*$, `A:`$\mathbb{S}\to\mathbb{L}$

Output: exists$\in\mathbb{L}$, ind$\in\mathbb{N}$

Precondition: –

Postcondition:

exists=∃i(1≤i≤length(X)): A(X[i]) and

exists → 1≤ind≤length(X) and A(X[ind])and

∀i(ind≤i≤length(X)): not A(X[i])

Copy + Search

Algorithm

```
i:=1
i≤length(X) and not A(X[i])
    i:=i+1
exists:=(i≤length(X))
exists
T                      | F
ind:=i                 | -
val:=X[i]              |
```

```
i=1..length(X)
    Y[length(X)-i+1]:=X[i]
i:=1
i≤length(X) and not A(Y[i])
    i:=i+1
exists:=(i≤length(X))
exists
T                      | F
ind:=length(X)-i+1     | -
```

length(X)=lenght(Y)

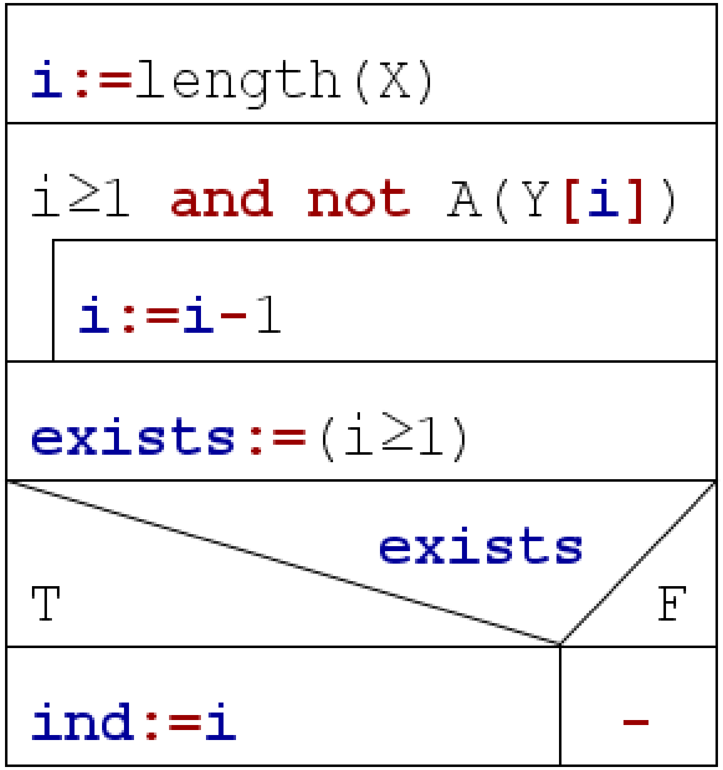
Copy + Search

Let's introduce the `j=length(X)-i+1` notation.

Then in the case of `i=1` → `j=length(X)`, and when we increase i, variable j will decrease;

instead of `i≤length(X)`, we use `length(X)-j+1≤length(X)`, i.e. `1≤j`.

Then the algorithm is the following:



Multiple item selection + summation

Task: Sum of elements with a certain attribute – conditional summation.

Specification:

Input: `X[1..]`$\in\mathbb{Z}^*$, `A`:$\mathbb{Z}\to\mathbb{L}$

Output: `sum`$\in\mathbb{Z}$

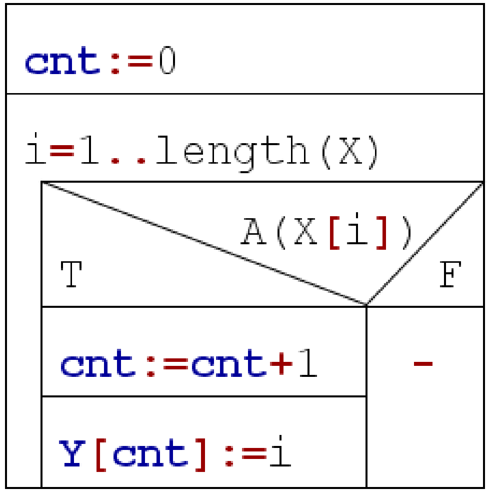
Precondition: –

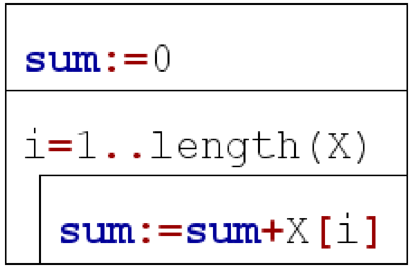
Postcondition:

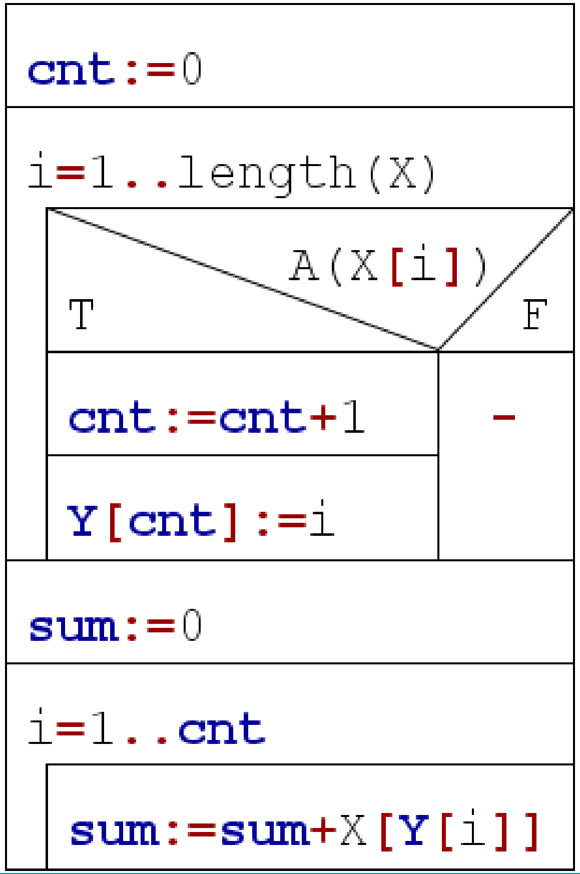
$$sum = \sum_{\substack{i=1 \\ A(X[i])}}^{length(X)} X[i]$$

Multiple item selection + summation

1. solution idea$_a$: Select all the elements with the given attribute, then add them.

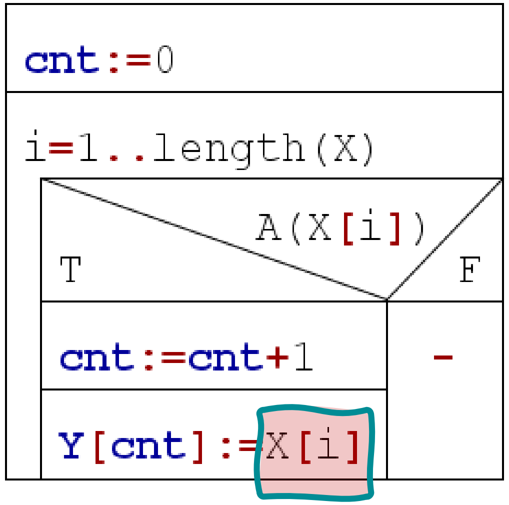


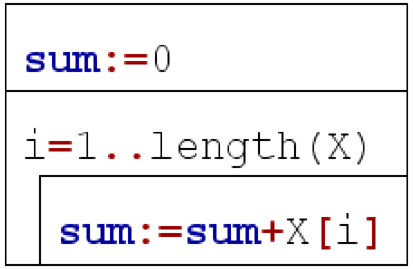


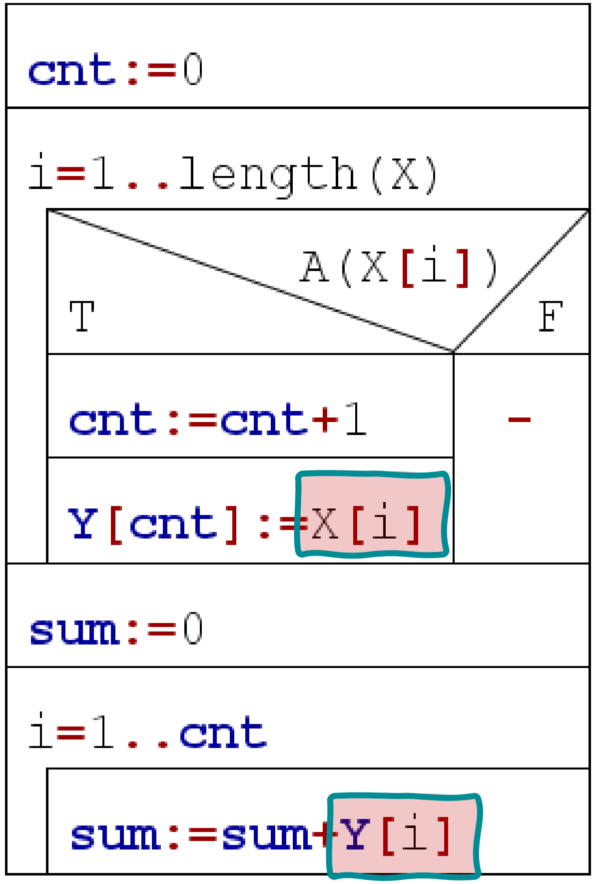


Multiple item selection + summation

1. solution idea$_b$: Select all the elements with the given attribute, then add them.

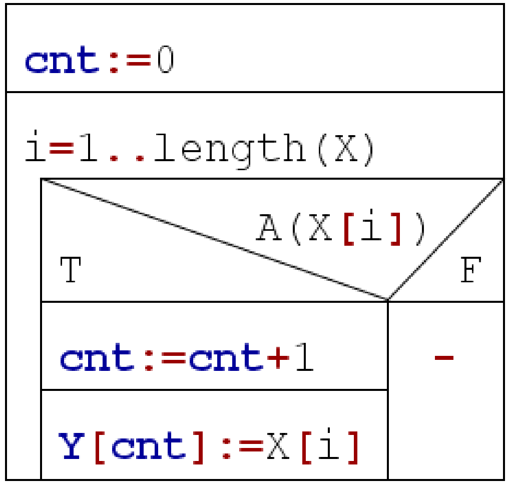


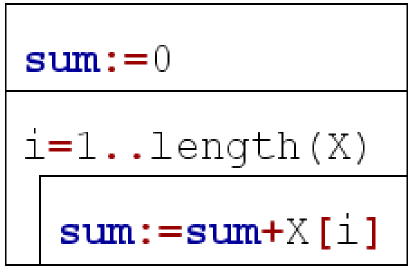


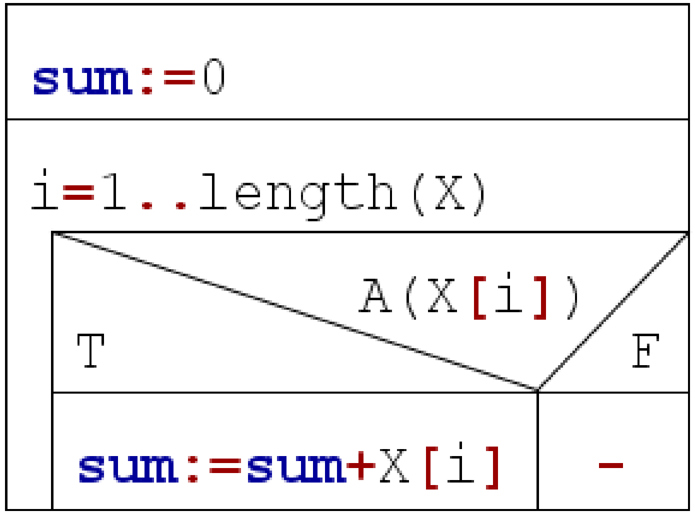


Multiple item selection + summation

2. solution idea: Instead of selecting the elements, add them immediately if they meet the condition → no need for storing values/indexes (array Y), no need for counting (variable cnt)







Multiple item selection + maximum selection

Task: Find the maximum of the elements that have a certain attribute – conditional maximum search.

Specification:

Input: X[1..]∈$\mathbb{S}^*$, A:$\mathbb{S}\rightarrow\mathbb{L}$

Output: exists∈$\mathbb{L}$, maxI∈$\mathbb{N}$

Precondition: –

Postcondition:

exists=∃i(1≤i≤length(X)): A(X[i]) and

exists → 1≤maxI≤length(X) and A(X[maxI]) and

∀ i(1≤i≤length(X)): A(X[i]) → X[maxI]≥ X[i]

Specification

Input: X[1..]∈$\mathbb{S}^*$, A:$\mathbb{S}\rightarrow\mathbb{L}$

Output: **exists**∈$\mathbb{L}$, **ind**∈$\mathbb{N}$

Precondition: –

Postcondition: exists=(∃i(1≤i≤length(X)):A(X[i])) and exists → 1≤ind≤length(X) and A(X[ind])

Specification

Input: X[1..]∈$\mathbb{S}^*$

Output: maxInd∈$\mathbb{N}$

Precondition: length(X)>0

Postcondition: 1≤maxInd≤length(X) and ∀i(1≤i≤length(X)):X[maxInd]≥X[i]

Multiple item selection + maximum selection

We could get the **idea** of the algorithm: Select the elements with the given attribute, then, select the maximum, if it makes sense.

Specification':

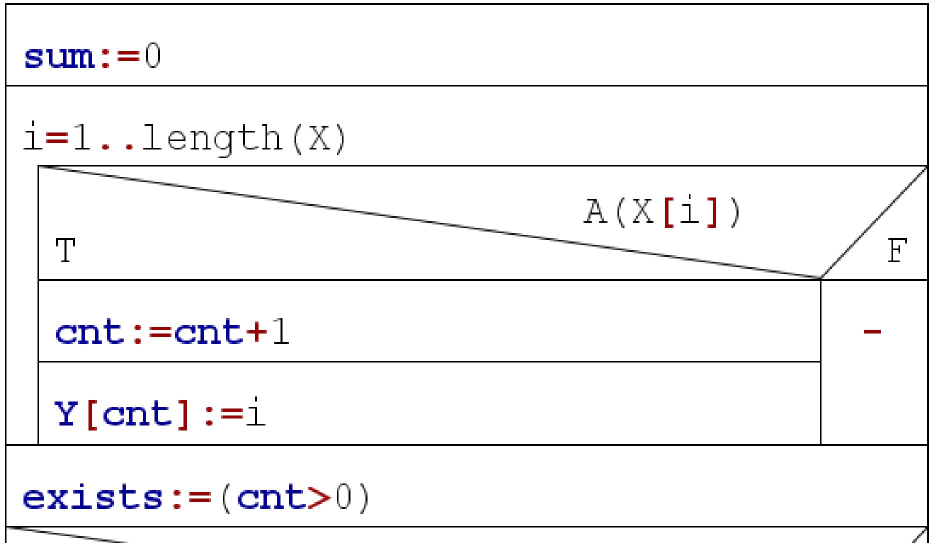
Postcondition: $cnt = \sum_{\substack{i=1 \\ A(X[i])}}^{length(X)} 1$ and ∀i(1≤i≤cnt):A(X[Y[i]])and Y⊆(1,2,…,length(X))

exists=(Cnt>0) and

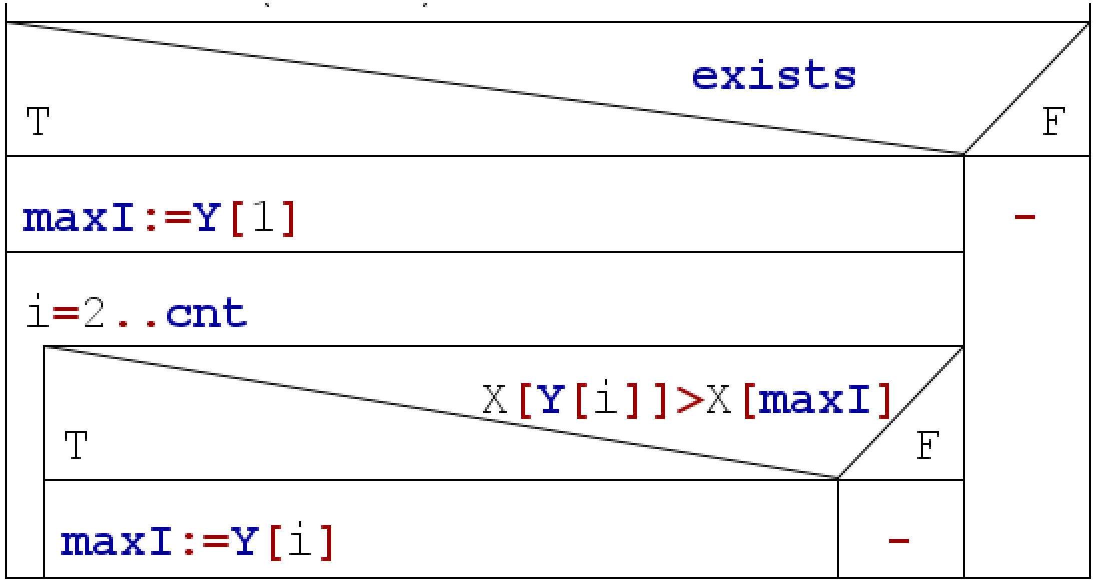
exists → 1≤maxI≤length(X) and A(X[maxI]) and ∀i(1≤i≤cnt):X[Y[i]]≤X[maxI]

Multiple item selection + maximum selection

Select the elements with the given attribute...



... then, select the maximum, if it makes sense.

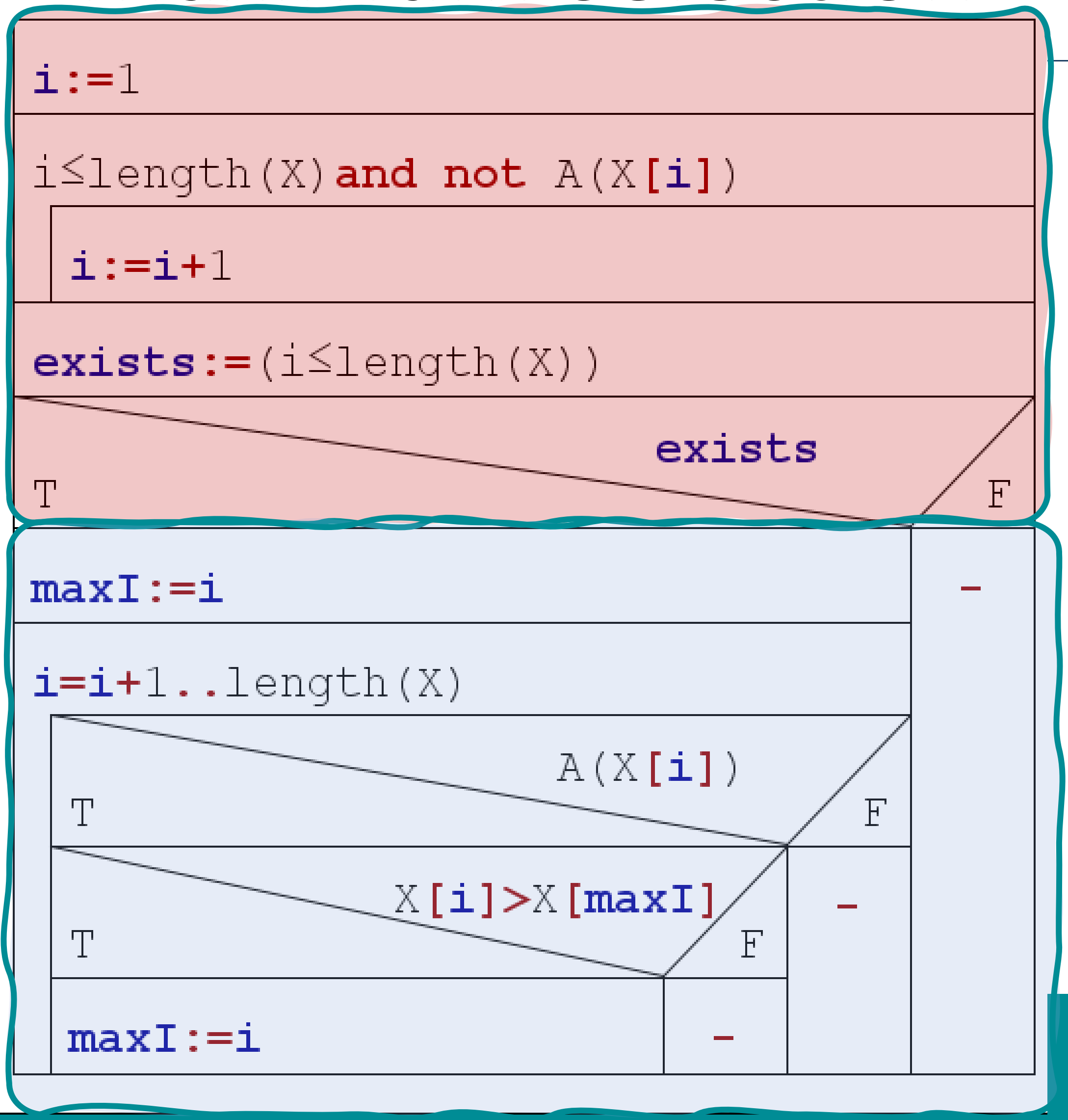


Multiple item selection + maximum selection

2. solution idea$_a$ (and algorithm):

Let's start from the PoA-s that we noticed in the specification. Instead of multiple item selection,

- let's find the first element with A attribute, then
- select the maximum of such elements.



Multiple item selection + maximum selection

2. solution idea$_b$ (and algorithm):

Let's start from the PoA-s that we noticed in the specification. Instead of multiple item selection,

- let's find the first element with A attribute, then
- select the maximum of such elements.

```
i:=1
i≤length(X)and not A(X[i])
    i:=i+1
exists:=(i≤length(X))
exists
T:
    maxI:=i
    i=i+1..length(X)
        A(X[i]) and X[i]>X[maxI]
        T:
            maxI:=i
        F:
            -
F:
    -
```

Multiple item selection + maximum selection

3. solution idea (and algorithm):

Instead of selecting the elements first, let’s **find the maximum immediately**. For this, we need a **fictive 0. element** that is **less than all the other elements**..

```
X[0]:=-∞
maxI:=0
i=1..length(X)
    A(X[i]) and X[i]>X[maxI]
    T                                   F
    maxI:=i                           | -
exists:=(maxI>0)
```

Maximum selection + multiple item selection

Task: Selecting **all** maximum elements.

Specification:

Input: $X[1..]\in\mathbb{S}^*$

Output: $cnt\in\mathbb{N},\ maxI[1..]\in\mathbb{N}^*$

using only the first cnt element
$maxI[1..cnt]\in\mathbb{N}^{cnt}$

Precondition: $length(X)>0$

Postcondition: $$cnt = \sum_{\substack{i=1 \\ X[i]=X[maxI[i]]}}^{length(X)} 1$$

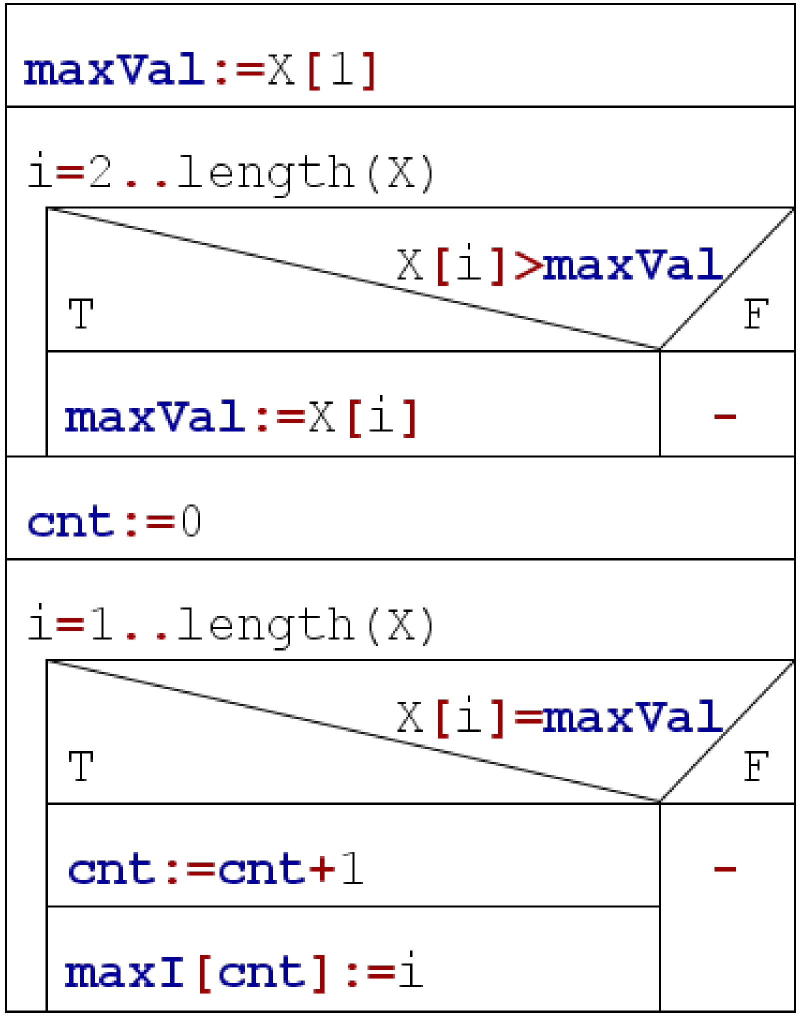
and $\forall i(1\le i\le cnt): \forall j(1\le j\le length(X)): X[maxI[i]]\ge X[j]$

and $maxI\subseteq(1,2,\dots,length(X))$

Maximum selection + multiple item selection

1. solution idea (and algorithm):

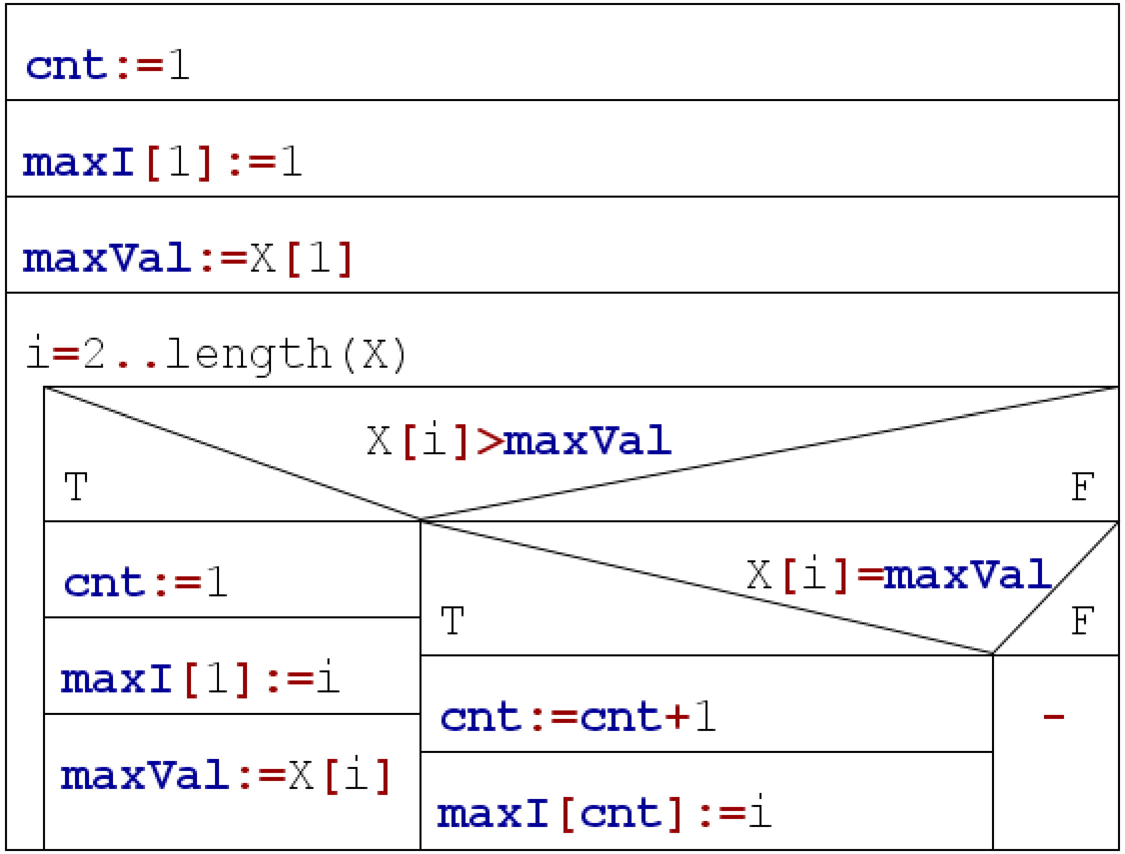
- First, let's find the maximum.
- Then select all the elements that are equal to it.



Maximum selection + multiple item selection

2. solution idea (and algorithm):

Let’s select the items equal to the current maximum. If we find a „bigger maximum”, we overwrite the previously selected elements.



Decision + counting

Task: Are there **at least K** elements with the given attribute in a sequence?

Specification:

Input: $K \in \mathbb{N}$, $X[1..] \in \mathbb{S}^*$, $A: \mathbb{S} \to \mathbb{L}$

Output: $exists \in \mathbb{L}$

Precondition: $K>0$

Postcondition:

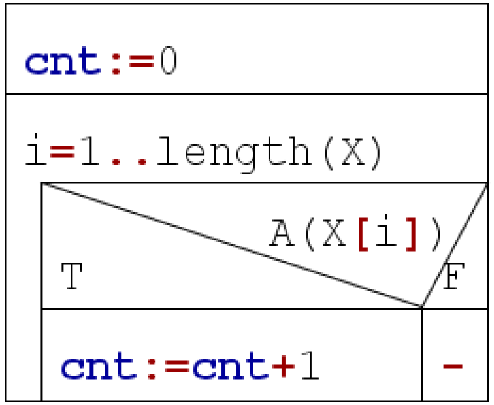
$$cnt = \sum_{\substack{i=1 \\ A(X[i])}}^{length(X)} 1 \ and \ exists = (cnt \geq K)$$

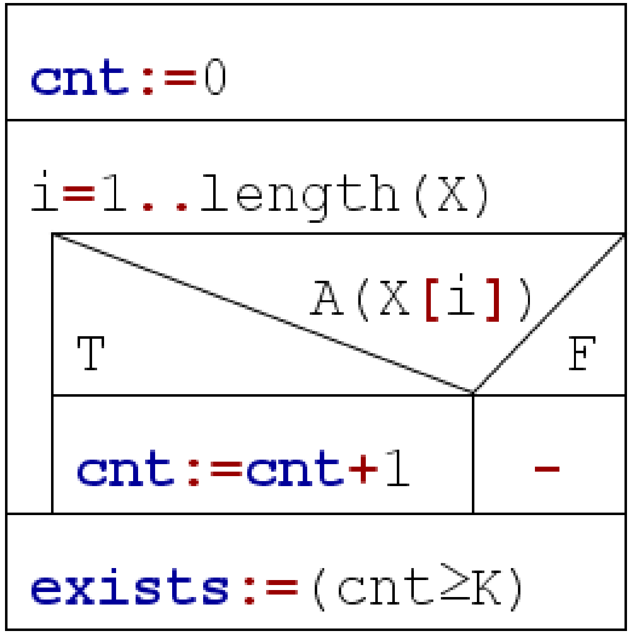
cnt is a local variable of the Postcondition

Decision + counting

1. solution idea (and algorithm):

Count how many elements have the given attribute, then see **whether it's more than K**. (So, in fact, there is no decision PoA.)

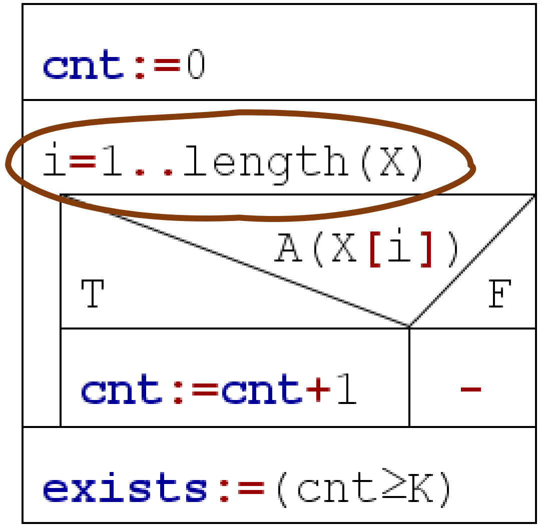


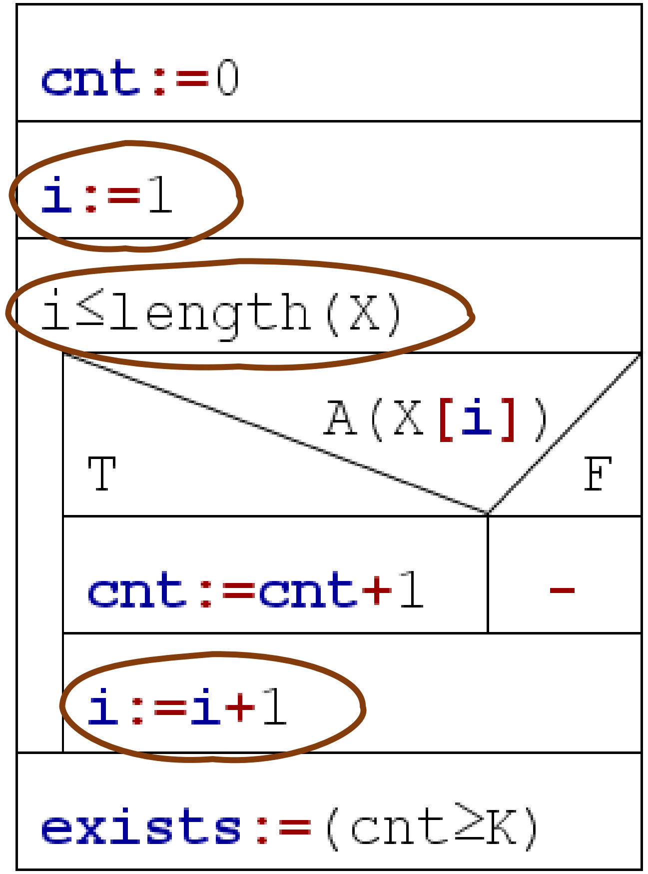


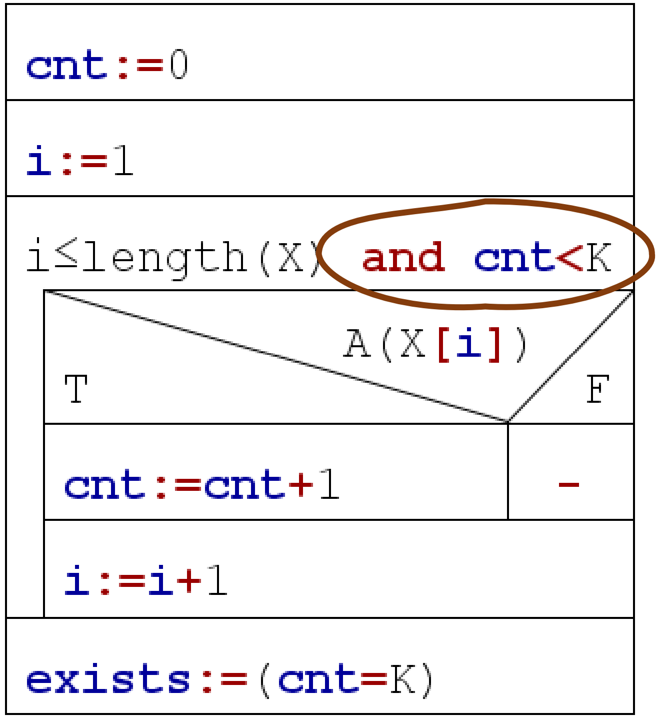
Decision + counting

2. solution idea (and algorithm):

If we have found K elements with the given attribute, then do not check any longer.







postcondition:

$$exists = \exists i(1 \le i \le length(X)): \left(\sum_{\substack{j=1 \\ A(X[j])}}^{i} 1 \right) = K$$

Search + counting

Task: In a sequence, **which** is the **K.** element with a given attribute (if there are at least K elements with that attribute)?

Specification:

Input: $K \in \mathbb{N}$, $X[1..] \in \mathbb{S}^*$, $A: \mathbb{S} \to \mathbb{L}$

Output: $exists \in \mathbb{L}$, $kI \in \mathbb{N}$

Precondition: $K > 0$

Postcondition:

$$exists = \exists i\left(1 \le i \le length(X): \left(\sum_{\substack{j=1 \\ A(X[j])}}^{i} 1\right) = K \text{ and } A(X[i])\right)$$

$$exists \to \left(1 \le kI \le length(X): \left(\sum_{\substack{j=1 \\ A(X[j])}}^{kI} 1\right) = K \text{ and } A(X[kI])\right)$$

Search + counting

1. solution idea:

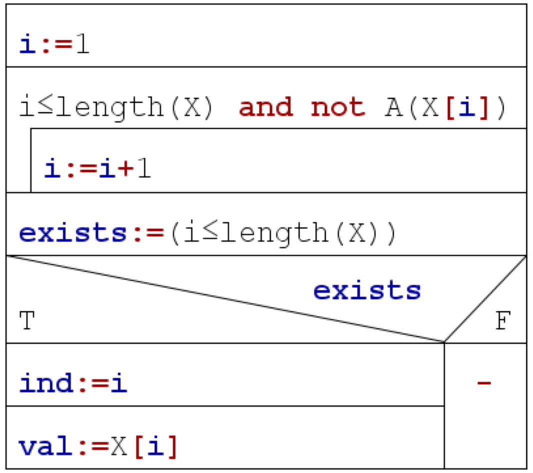
- First, let's **count** how many elements have the given attribute, then observe **whether it's at least K**. But this is not enough, we have to go back, and find the K. element.

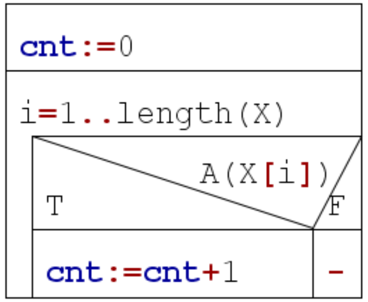
- A solution that seems to be working: instead of counting, **multiple item selection** is needed. Then we **do not need search**. But this is memory consuming , and a long process.

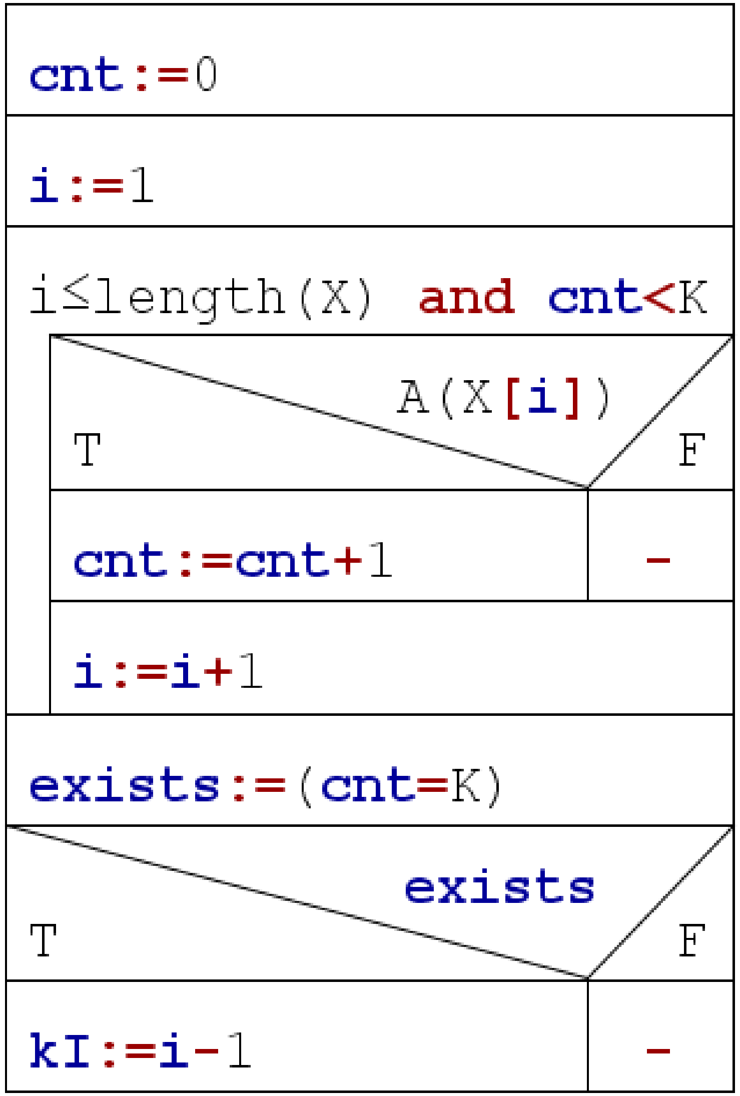
Search + counting

2. solution idea:

If we have found K elements with the given attribute, do not check any longer: search until the K. element. Then note the index of the K. element.







Search + copy

Task: Selection of all the elements of a sequence that are before an element having attribute A. (If no element with the attribute, then all the elements.)

Specification:

Input: $X[1..]\in\mathbb{S}^*,\ A:\mathbb{S}\to\mathbb{L}$

Output: $cnt\mathbb{N},\ Y[1..]\in\mathbb{S}^*$

Precondition: –

Postcondition: $exists=\exists i(1\le i\le length(X)): A(X[i])$ and

$(exists$ and $1\le cnt<length(X)$ and $A(X[cnt+1])$ or $cnt=1)$

and $\forall i(1\le i\le cnt):$ not $A(X[i])$ and $Y[i]=X[i]$

Search + copy

1. solution idea:

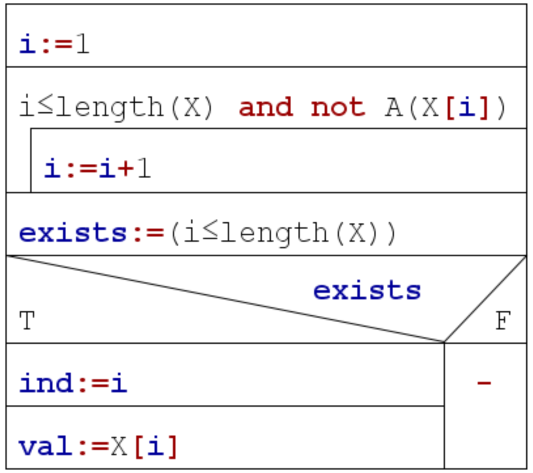
Let's **find** the first element with the given attribute, then **copy the elements** before it.

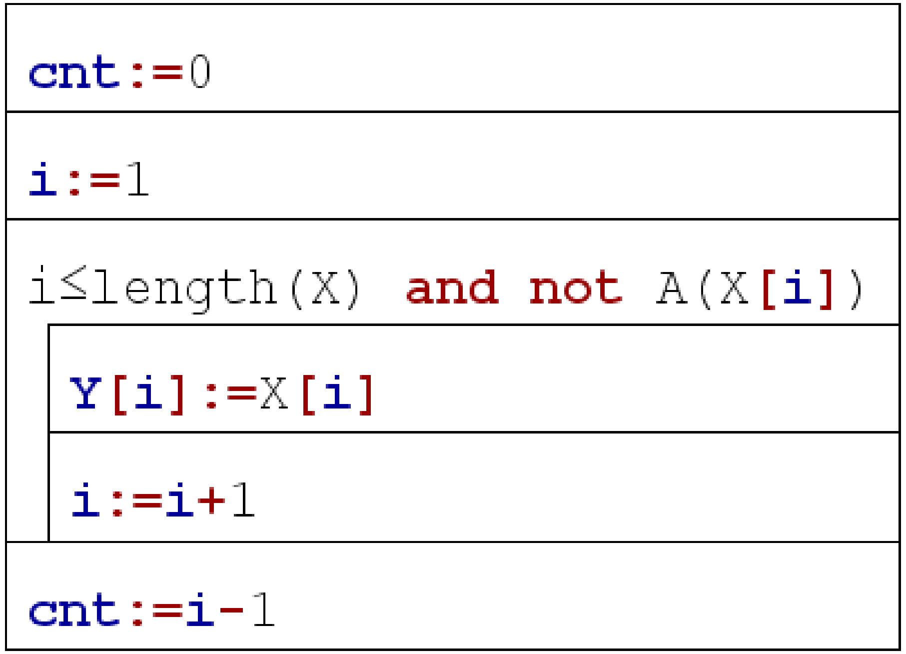
… long process!

Search + copy

2. solution idea:

Copy the elements **while searching** for the first element with the given attribute:





Decision + decision

Task: Do two sequences have a common element?

Specification:

Input: $X[1..] \in \mathbb{S}^*,\ Y[1..] \in \mathbb{S}^*$

Output: $exists \in \mathbb{L}$

Precondition: –

Postcondition:

$exists = \exists i(1 \le i \le length(X)): \exists j(1 \le j \le length(Y)): Y[j] = X[i]$

Think about the similarities of this task and the decision in a matrix)

Decision + decision

1. solution idea:

- Let’s collect all the common elements (**intersection**), and if the count of elements is **at least 1**, then there is a common element

Specification: The „rewriting” of postcondition:

- The sub result of intersection: $\texttt{cnt}\in\mathbb{N}$
- The modified postcondition: the postcondition of intersection and `exists=cnt>0`

Comment:

intersection = multiple item selection + decision

Decision + decision

2. solution idea:

If there is at least one common element, do not check any longer.

```
i:=1
i≤length(X) and not A(X[i])
    i:=i+1
exists:=(i≤length(X))
```

```
i:=0
exists:=false
i≤length(X) and not exists
    i:=i+1
    exists:=A(X[i])
```

```
i:=0
exists:=false
i≤length(X) and not exists
    i:=i+1
    j:=1
    j≤length(Y) and X[i]≠Y[j]
        j:=j+1
    exists:=(j≤length(Y))
```

Program transformation

Program transformation: all operation, that takes an algorithm (or computer program) and generates another algorithm (program) semantically equivalent to the original.

Main aims:

- Increase effectiveness;
- Simplify;
- Make it more feasible.

Program transformation – simplying, increase effectiveness

Task: the fartherst point from the origin…

```
Max:=1; MaxVal:=sqrt(p[1].x²+p[1].y²)
i=2..N
    sqrt(p[i].x²+p[i].y²)>MaxVal
    T:
        Max:=i
        MaxVal:=sqrt(p[i].x²+p[i].y²)
    F:
        —
```

The square root is a monotonuos function – it's not needed to define the maximum.

Program transformation – simplying, increase effectiveness

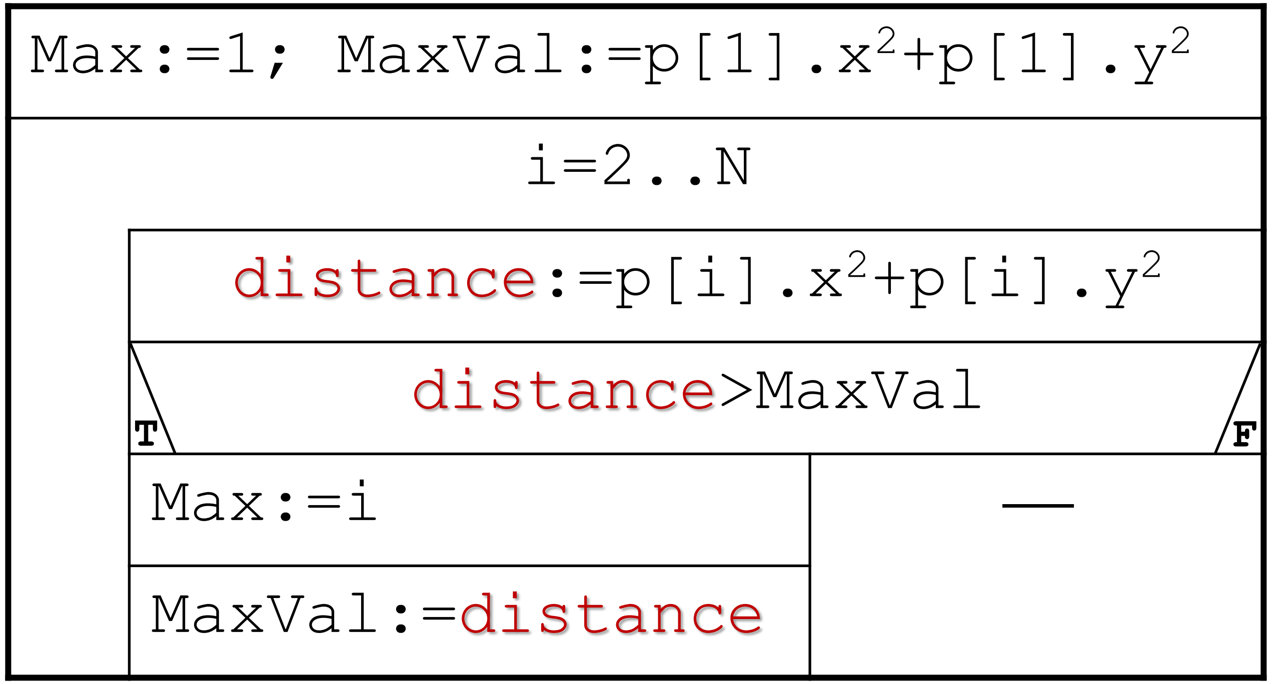
Task: the farthest point from the origin…

```
Max:=1; MaxVal:=p[1].x²+p[1].y²
i=2..N
    p[i].x²+p[i].y²>MaxVal
    T:
        Max:=i
        MaxVal:=p[i].x²+p[i].y²
    F:
        —
```

Here, we calculate the same expression several times.

Program transformation – to omit recalculation

Task: the farthest point from the origin…



Program transformation – expanding paralell value assignment

```
a,b,c:=f(x),g(x),h(x);
```

- Can be divided into consecutive calculation, if the relation is cycle free

```
a:=f(x); b:=g(x); c:=h(x);
```

Program transformation – expanding paralell value assignment

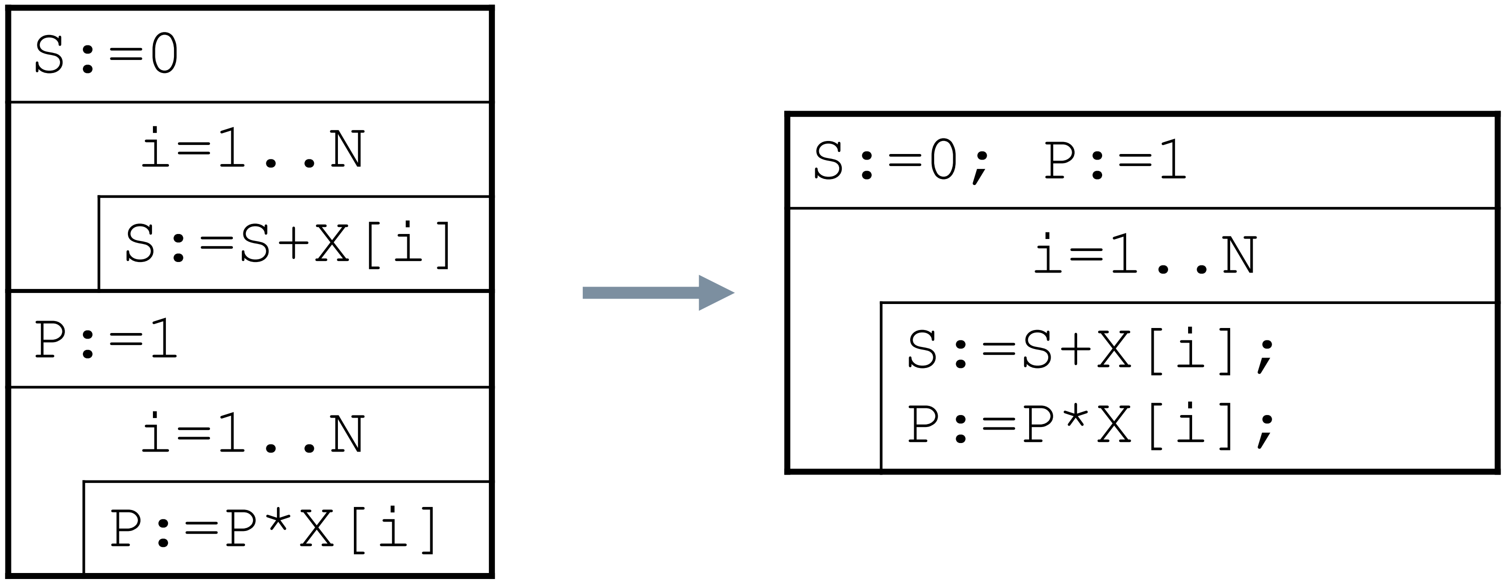
```
a,b,c:=b,c,a;
```

- Can be divided into consecutive calculations with the help of an ***auxiliary variable,*** if the relation is **not cycle free** – ontains a cycle

```
av:=a; a:=b; b:=c; c:=av;
```

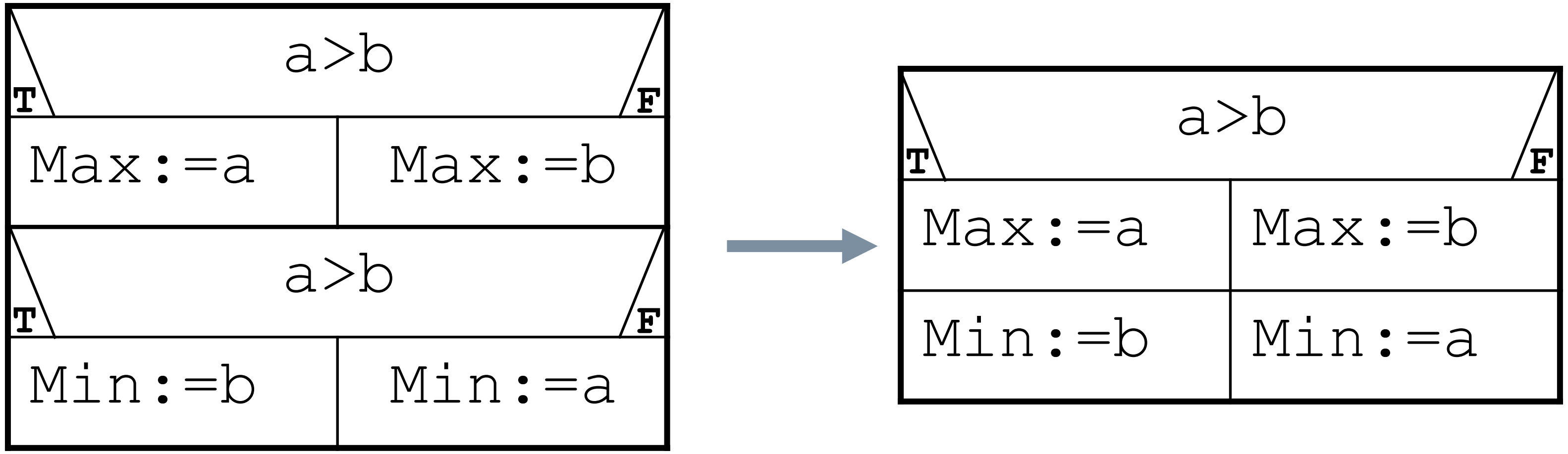
Program transformation – combining loops, loop fusion

- Loops having the **same number of loop cycles**, can be combined, if they are **independent** of each other



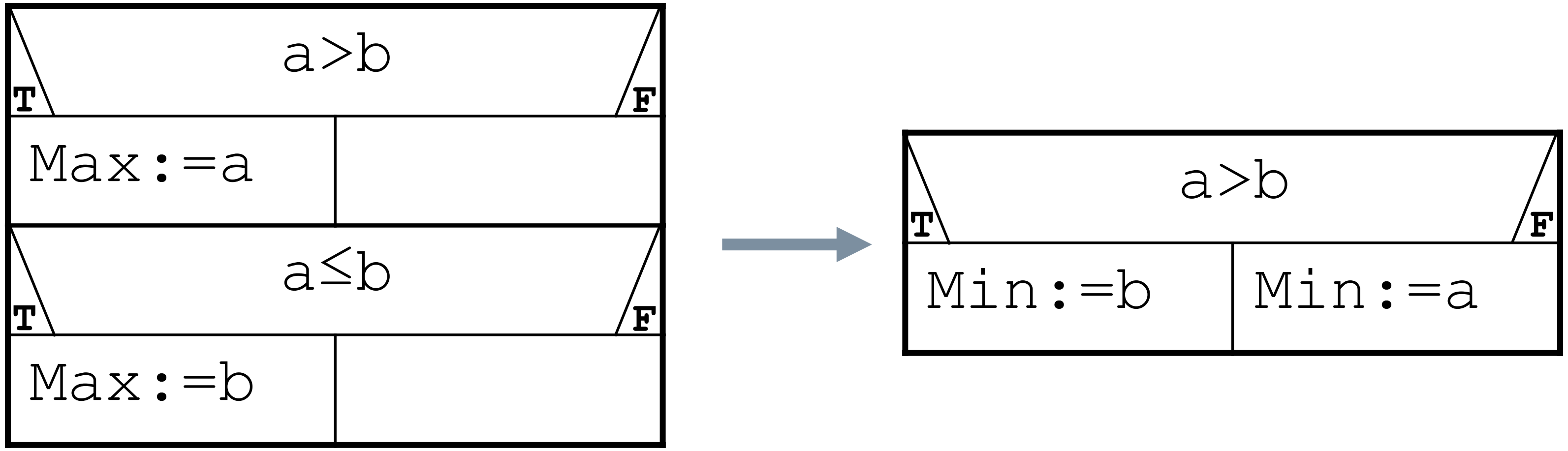
Program transformation – combining conditional statements

- Conditional statements having the **same conditions** can be combined, if they are **independent** of each other

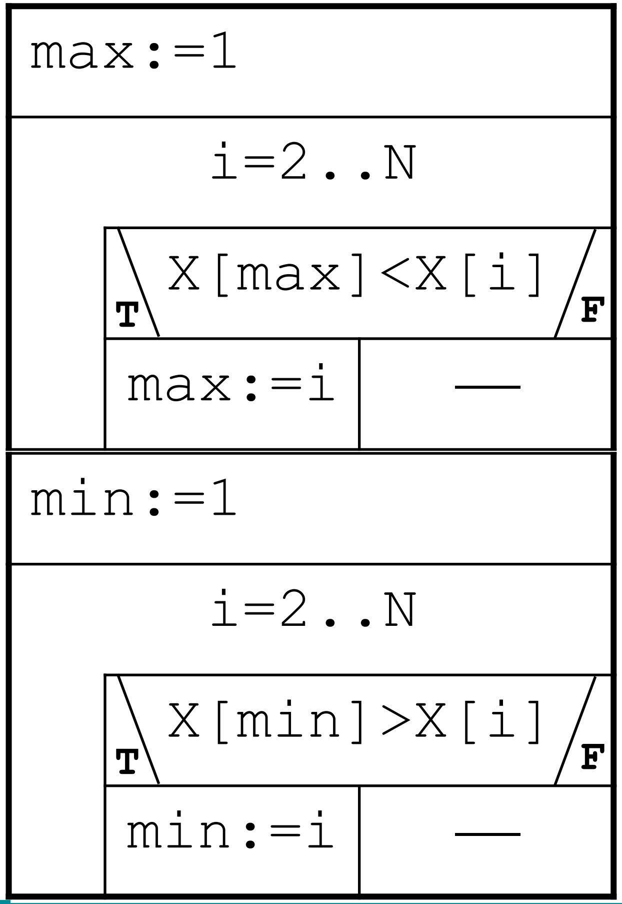


Program transformation – combining conditional statements

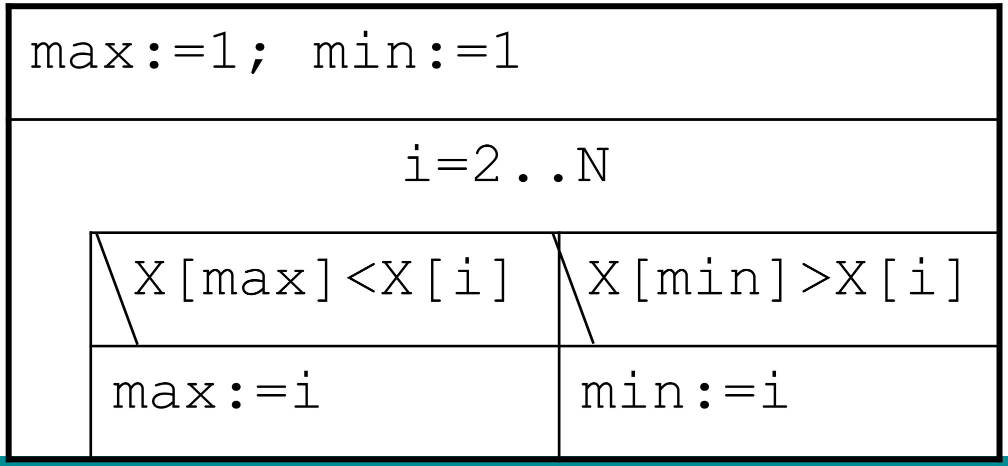
- Combination of conditionals having exclusive, full conditionals, if they are **independent** of each other



Program transformation – fusion of loops and conditionals

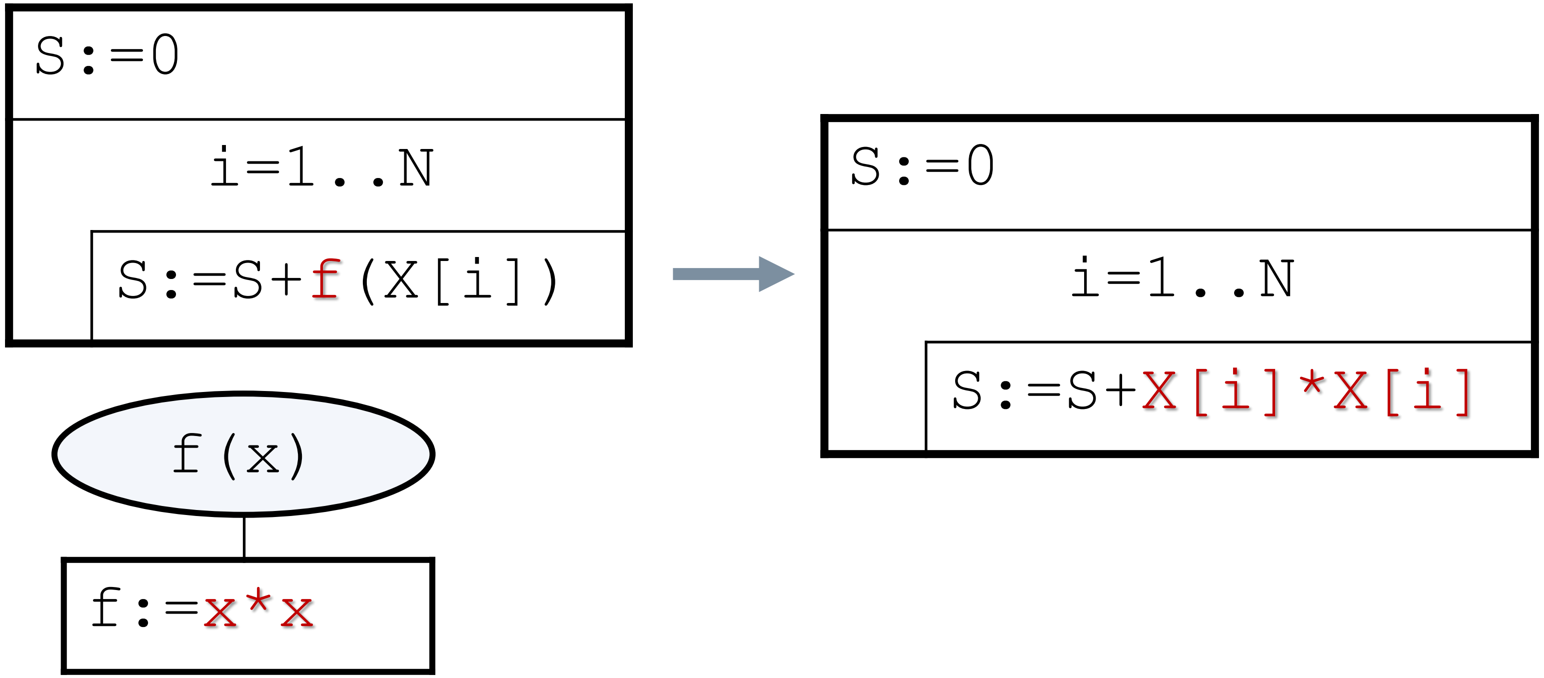


- Loops having the **same number of steps** and conditional statements having **exclusive conditions** could also be combined, if they are **independent** of each other



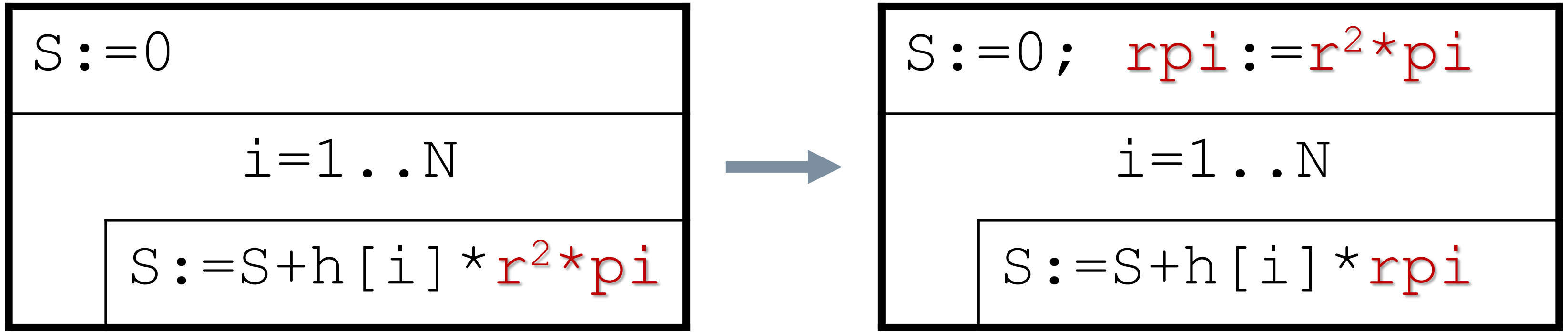
Program transformation – function inlining

- Instead of a function call, the formula of a simple function (the body of the function) can be written. (C++ compilers can do such optimalization.)



Program transformation – hoisting: moving expression outside loop

- **Loop-invariant** expressions can be **hoisted out of loops**, thus improving run-time performance by **executing the expression only once** rather than at each iteration.



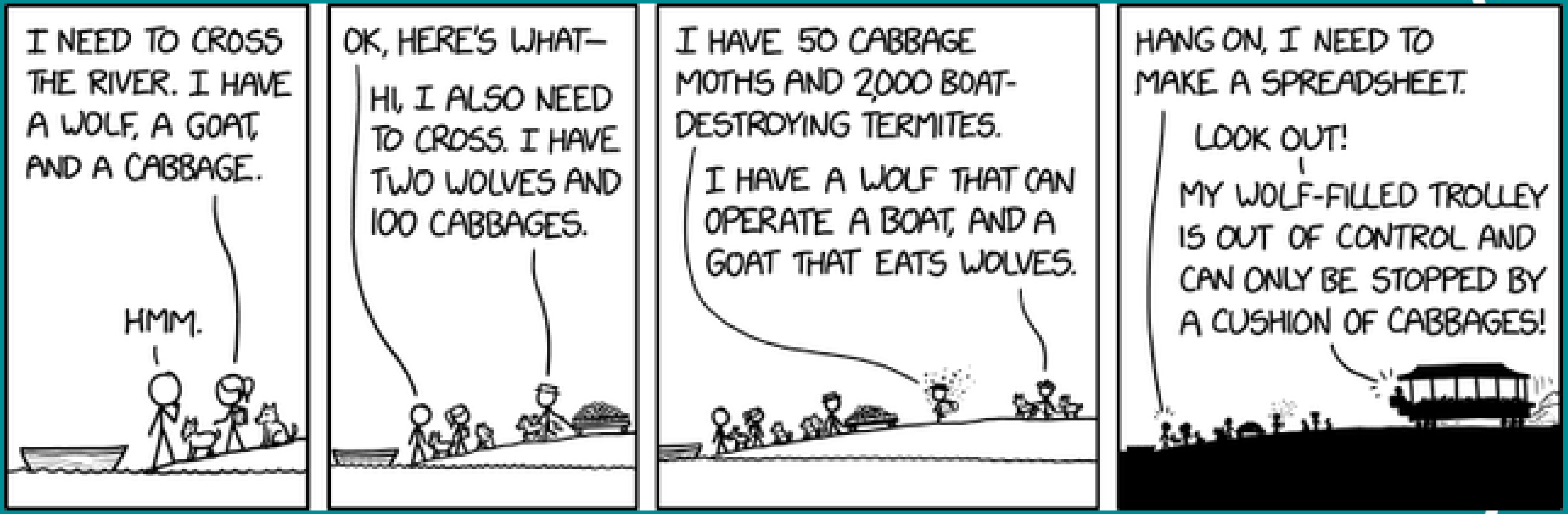
Program transformation

- searching, deciding → selecting
- We put a special element with A attribute at the end of the sequence – certainly will be find one

```
i:=1
i≤N and A(X[i])
    i:=i+1
Exists:=i≤N
```

```
i:=1;
X[N+1]:=Titem
A(X[i])
    i:=i+1
Exists:=i≤N
```

ELTE | FACULTY OF INFORMATICS



Thank you for your attention!