ELTE | FACULTY OF INFORMATICS

PROGRAMMING

Lecture 3

Zsuzsa Pluhár

pluharzs@inf.elte.hu

Patterns of Algorithms (PoA) – the essence

Its **aim:**

A proven template as a basis, on which we can build our solutions later.
(Development will be quicker and safer)

Its **structure**:

1. abstract task specification [+program specification]
2. abstract algorithm

Previous comment: the input is at least a sequence (later:intervall)

Patterns of Algorithms (PoA) – the essence

How we use them:

1. create a specification for the given task
2. guess the PoA from specification
3. map the parameters of the given task to the parameters of the corresponding abstract task
4. "generate" the task-specific algorithm based on the algorithm of the PoA, by changing the parameters as per point 3.
5. create a more efficient algorithm using program transformations

Patterns of Algorithms

What is PoA? It is the general solution of a typical programming task.

- sequence → single value
- sequence → sequence
- sequence → sequences
- sequences → sequence

1. Sequence calculations

Tasks (examples):

1. We know the monthly income and expenses of a person. Let's calculate by **how much** money his asset will change by the end of the year!
2. We know the lap times of a car racer. Let's determine his **average** lap time!
3. Let's calculate the **factorial** of N!
4. We know N words. Give the sentence we get by **joining** them.

1. Sequence calculations

Grouping:

- Sum of numbers: “asset”, “lap times”
- Product of numbers: “factorial”
- Words joined: “words”

What is common?

We have a sequence of "somethings", and we have to calculate a single "something" from their elements.

eg. Σ – income/lap time; Π – factorial; **&** – words

1. Sequence calculations

Specification

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

Output: $sc \in \mathbb{S}$

Precondition: –

Postcondition: $sc=F(X[1]..X[length(X)])$

$F: \mathbb{S}^* \rightarrow \mathbb{S}$

Σ – sum of N elements;

Π – product of N elements;

$\cup$ - union of N sets

& – concatenation of N texts …

$\mathbb{S}$: an arbitrary set;

$\mathbb{S}^* = \{(s_1,\ldots,s_?) \mid s_i \in \mathbb{S}\}$

The items are indexed from 1 to the Length(X)

1. Sequence calculations

$\mathbb{S}$: $\mathbb{N}$,$\mathbb{Z}$ or $\mathbb{R}$

Specification (Summation)

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

Output: $sum \in \mathbb{S}$

Precondition: –

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

Recursive definition (math): $$\sum_{i=1}^{N} X_i := \begin{cases} 0 & , N = 0 \\ \left(\sum_{i=1}^{N-1} X_i \right) + X_N & , N > 0 \end{cases}$$

1. Sequence calculations

General problem:

F: operation with N parameters, where N is variable

Solution:

Decomposition to *2-parameter operations* (e.g. Σ to +) and to a *neutral-value* (in the case of + it is 0).

$$F(X_{1..N})=f(F(X_{1..N-1}),X_N) \quad , \text{ if } N>0$$
$$F(-)=F_0 \quad \text{otherwise}$$

1. Sequence calculations

Specification (general)

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

Output: $sc \in \mathbb{S}$

Precondition: –

Postcondition: $sc = F(X[1]..X[length(X)])$

Definition : $F: \mathbb{S}^* \to \mathbb{S}$

$$F(X_{1..N}) := \begin{cases} F_0 & , N = 0 \\ f(F(X_{1..N-1}), X_N) & , N > 0 \end{cases}$$

$f: \mathbb{S} x \mathbb{S} \to \mathbb{S}, \quad F_0 \in \mathbb{S}$

1. Sequence calculations

Specification (generalising further)

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

Output: $sc \in \mathbb{S}_2$

Precondition: –

Postcondition: $sc = F(X[1]..X[length(X)])$

Definition : $F: \mathbb{S}_1^* \to \mathbb{S}_2$

$$F(X_{1..N}) := \begin{cases} F_0 & , N = 0 \\ f(F(X_{1..N-1}), X_N) & , N > 0 \end{cases}$$

$f: \mathbb{S}_2 x \mathbb{S}_1 \to \mathbb{S}_2, \quad F_0 \in \mathbb{S}_2$

1. Sequence calculations

Algorithm – declaring program variables

Input: $X[1..]\in\mathbb{S}_1^*$
Output: $sc\in\mathbb{S}_2$

Constant
  maxN: **Integer**=???
Variables
  n: **Integer**
  X: **Array**[1..maxN:Ts]
  sc: Ts

maxN: the maximum size of the array

Ts: Type of the elements (typeset)

Declaring array for the sequence in a **static** way

1. Sequence calculations

Algorithm (general)

```
sc:=F0
i=1..length(X)
    sc:=f(sc,X[i])
```

Algorithm (summation, Σ)

```
sum:=0
i=1..length(X)
    sum:=sum+X[i]
```

1. Sequence calculations - example

We know the lap times of a car racer. Let's determine his **average** lap time!

n: the number of lap times
n=length(X)

Specification

Input: $n \in \mathbb{N}$, $X[1..n] \in \mathbb{N}^n$
Output: $av \in \mathbb{R}$
Precondition: –
Postcondition: $av = \left(\sum_{i=1}^{length(X)} X[i] \right) : length(n)$

sum – helper variable

```
sum:=0
i=1..length(X)
    sum:=sum+X[i]
av:=sum/length(X)
```

Code it!

1. Sequence calculations - example

We know the monthly income and expenses of a person. Let's calculate by how much money his asset will change by the end of the year!

Specification

Input: $n \in \mathbb{N}$, $Inc[1..n] \in \mathbb{N}^n$, $Outc[1..n] \in \mathbb{N}^n$

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

Precondition: –

Postcondition:

$$sum = \left(\sum_{i=1}^{length(X)} (Inc[i] - Outc[i]) \right)$$

1. Sequence calculations - example

We know the monthly income and expenses of a person. Let's calculate by how much money his asset will change by the end of the year!

Algorithm

```
sc:=F0
i=1..length(X)
    sc:=f(sc,X[i])
```

length(X)	→	n
X[ ]	→	Inc[ ]-Outc[ ]
sc	→	sum
F0	→	0
f()	→	+

```
sum:=0
i=1..n
    sum:=sum+(Inc[i]-Outc[i])
```

Code it!

Lessons to learn

1. The **precondition** of a given task might be **stricter** than that of the used pattern of the algorithm.
2. The **postcondition** of a given task might be **weaker** than that of the used pattern of algorithm (we will meet this).
3. Indexing: from 1 to N $\rightarrow$ from B to E, using length()
4. Instead of working with elements of an array, the function could ask for the value of the ith element (from more arrays, from more array elements, or with a function independent of the array).

2. Counting

Tasks (examples):

1. We know the monthly income and expenses of a person. Let's count the number of months where his assets grow!
2. Let's calculate the number of divisors of an integer!
3. Let's determine how many letter "a" can be found in the name of a person!
4. Based on the annual daily statistics, let's count the number of days when frozen!
5. We have birth date (month) data from N people. Let's count how many of them have birthday during the winter!

2. Counting

What is common?

We have a sequence of „somethings”, and we have to count how many of their items have a given attribute.

2. Counting

$\mathbb{S}$: an arbitrary set;

Specification

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

A is an arbitary attribute function

Output: `cnt`$\in\mathbb{N}$

Precondition: –

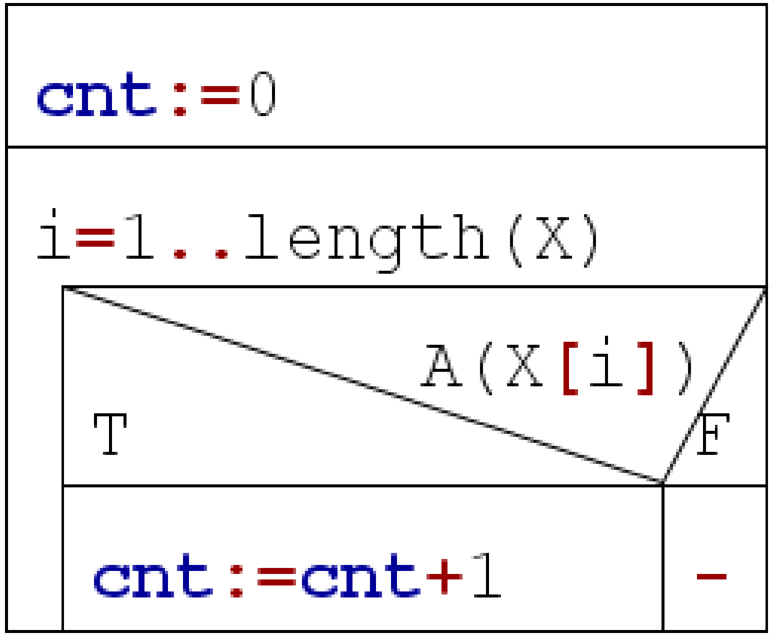
Postcondition:

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

Note: A is an attribute that can be given as a Boolean function. One can tell about each element of X (and $\mathbb{S}$) whether they have this attribute or not.

2. Counting

Algorithm



2. Counting - example

We have birth date (month) data from N people. Let's count how many of them have birthday during the winter!

Specification

Input: $n \in \mathbb{N}$, $Mon[1..n] \in \mathbb{N}^n$,

$Winter?: \mathbb{N} \to \mathbb{L}$; $Winter?(x) = (x<3 \text{ or } x=12)$

Output: $cnt \in \mathbb{N}$

Precondition: $\forall i(1 \leq i \leq N): 1 \leq Mon[i] \leq 12$

Postcondition:

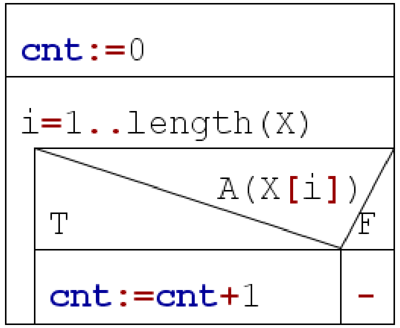
$$cnt = \sum_{\substack{i=1 \\ Winter?(X[i])}}^{N} 1$$

Note: Precondition can be stricter than that of the used pattern.

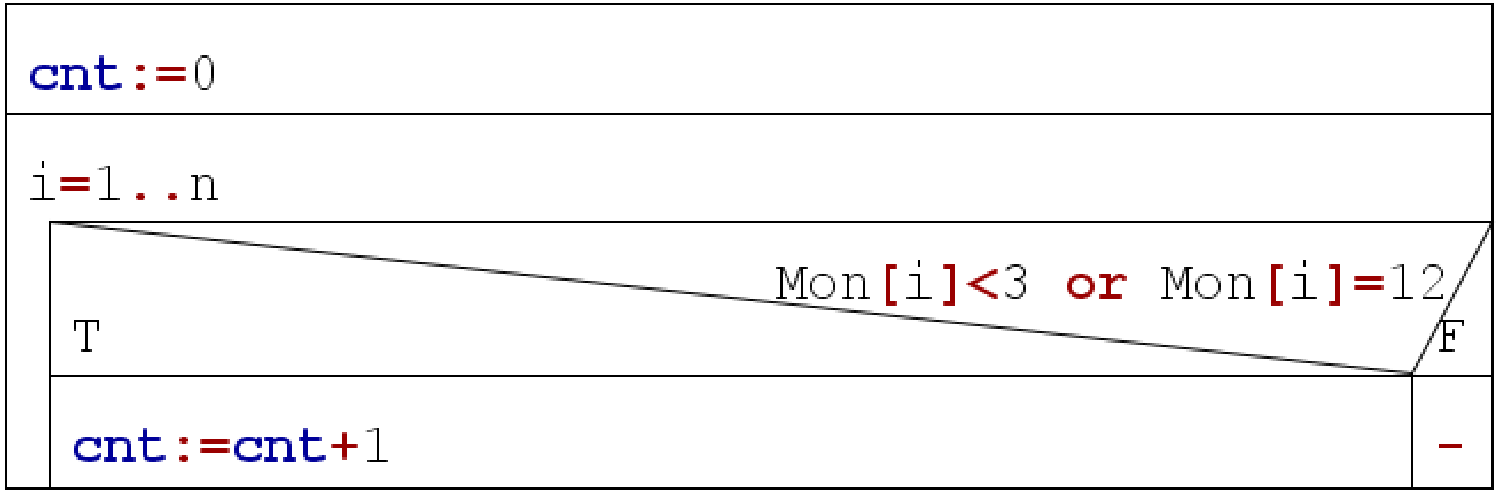
2. Counting - example

We have birth date (month) data from N people. Let's count how many of them have birthday during the winter!

Algorithm



length(X)	→	n
X[ ]	→	Mon[ ]
cnt	→	cnt
A(X[i])	→	Mon[i]<3 or Mon[i]=12



What would happen if the precondition was not met?

3. Maximum selection

Tasks (examples):

1. We know the monthly income and expenses of a person. Let's determine the month where his assets grow the **most**!
2. Let's pick the name of the person who is the **last** in the alphabetical order!
3. Let's find the person, who likes the **most** types of food!
4. Based on the annual daily statistics, let's define the **warmest** day during the year!
5. We have birth dates from N people, let's find who has birthday in the year **first**!

3. Maximum selection

What is common?

We have to pick/find the greatest (or least) something from a sequence of „somethings”.

Important:

- The somethings have an attribute based on which we can sort them (sorting relation).
- If we have at least 1 element, then we know that there exists a maximum (or minimum).

3. Maximum selection

Specification

Input: `X[1..]∈`$\mathbb{S}^*$

Output: `maxInd∈`$\mathbb{N}$`, maxVal∈`$\mathbb{S}$

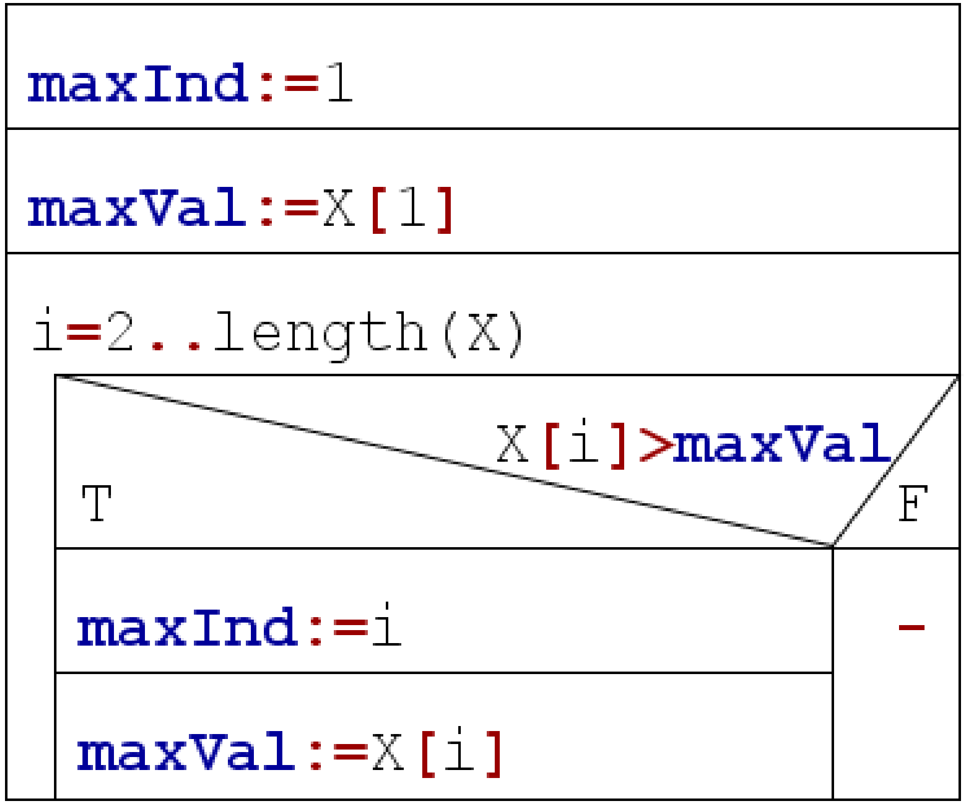
Precondition: `length(X)>0`

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

Short: $(maxInd, MaxVal) = \underset{i=1}{\overset{length(X)}{\boldsymbol{MAX}}} (X[i])$

3. Maximum selection

Algorithm:



Note: If there are more elements equal to the greatest then this algorithm will find the first one.

Questions:

- How can we find the last greatest one?
- How can we find the (first) least element?

Modify the algorithm

3. Maximum selection

Comments:

- We assume the following sort operator exists:

Someting$\times$Something $\rightarrow$ Boolean;

- The sequence number of an element is more general information, thus we provide that instead of the element itself.

3. Maximum selection (index)

Specification

Input: X[1..]∈$\mathbb{S}^*$

Output: maxInd∈ℕ

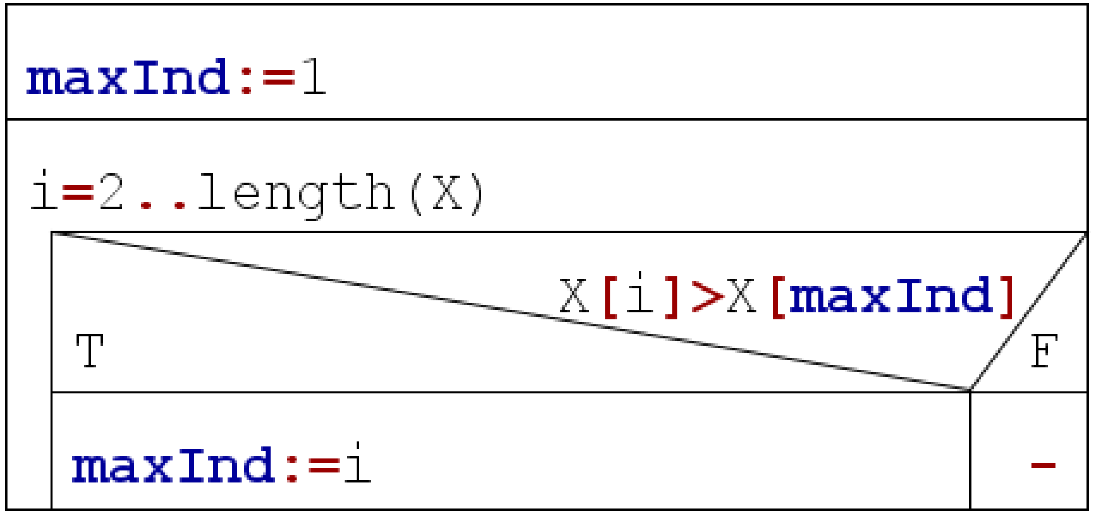
Precondition: length(X)>0

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

Short: $$(maxInd) = \underset{i=1}{\overset{length(X)}{\boldsymbol{MAXIND}}}\,(i)$$

3. Maximum selection (index)

Algorithm



3. Maximum selection (value)

Specification

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

Output: `maxVal`$\in \mathbb{S}$

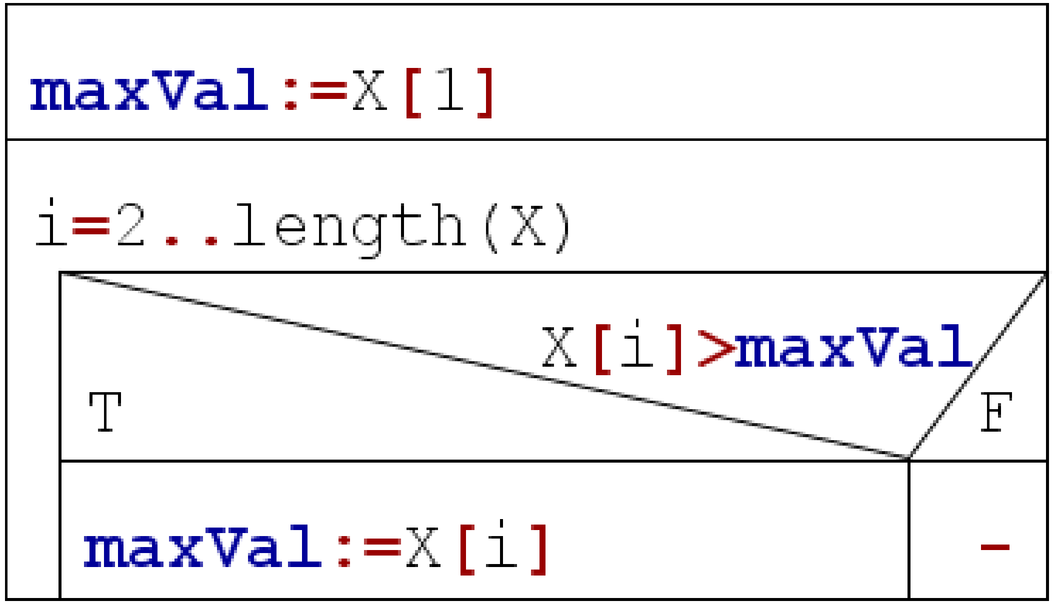
Precondition: `length(X)>0`

Postcondition: `maxVal∈X and`
`∃i(1≤i≤length(X)):maxVal=X[i] and`
`∀i(1≤i≤N):maxVal≥X[i]`

Short: $$(maxVal) = \underset{i=1}{\overset{length(X)}{\boldsymbol{MAXVAL}}}\,(X[i])$$

3. Maximum selection (value)

Algorithm



3. Maximum selection - example

Based on the annual daily statistics, let's define the warmest day during the year!

Specification

Input: Temp[1..365]∈$\mathbb{N}^{365}$,

Output: wdi∈$\mathbb{N}$

Precondition: 365>0 ☺

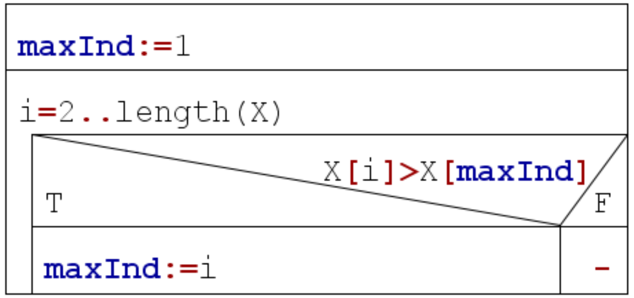
Postcondition: 1≤wdi≤365 and ∀i(1≤i≤365):Temp[wdi]≥Temp[i]

$$(wdi) = \underset{i=1}{\overset{365}{\boldsymbol{MAXIND}}}\,(Temp[i])$$

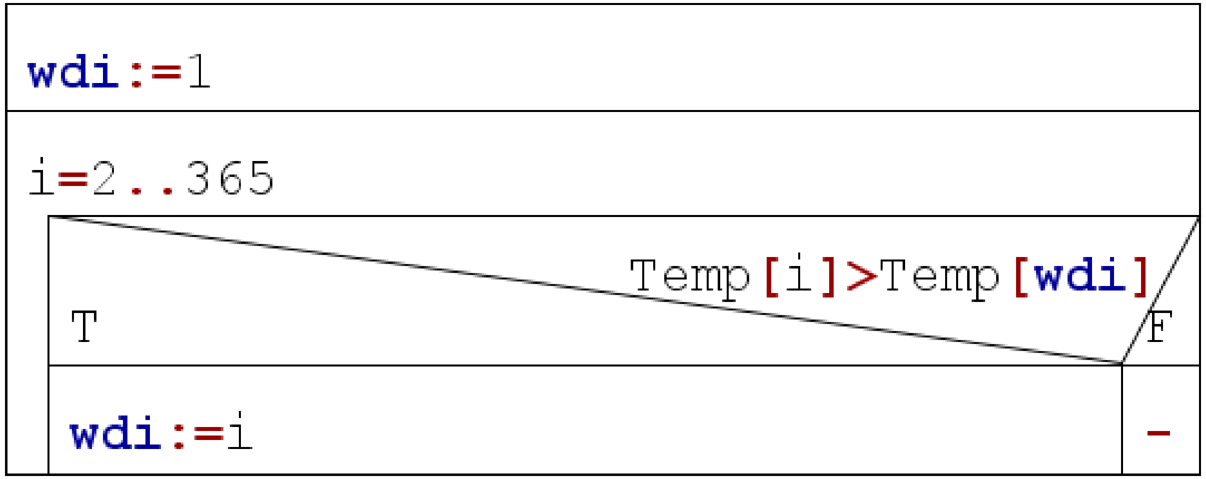
3. Maximum selection - example

Based on the annual daily statistics, let's define the warmest day during the year!

Algorithm



length(X)	→	n
X[ ]	→	Temp[ ]
maxInd	→	wdi
A(X[i])	→	Temp[i]>temp[wdi]



4. Search

Tasks (examples):

1. We know the monthly incomes and expenses of a person. His assets grow by the end of the year. Let's give a month, when his assets didn't grow!
2. Let's define a non-1 and non-N divisor of the integer N!
3. Let's search for letter "a" in the name of a person!
4. Let's search for a course in which the student failed!
5. Let's find an element in a sequence that is greater than the previous element!

4. Search

What is common?

We have a sequence of „somethings”, and we have to search for an element that has a given attribute, and we do not know whether such an element exists in the sequence

4. Search

Specification

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

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

„Is there any” => decision

Precondition: –

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

„return the value/index” => selection

Short: $$(exists, ind, val) = \underset{\substack{i=1 \\ A(X[i])}}{\overset{length(X)}{\boldsymbol{SEARCH}}}\,(X[i])$$

4. Search

Algorithm$_1$

Specification

Input: X[1..]∈𝕊*, A:𝕊→𝕃

Output: **exists**∈𝕃, **ind**∈ℕ, **val**∈𝕊

Precondition: –

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

Modify the algorithm

Find the last item

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

4. Search

Algorithm$_2$

Specification

Input: X[1..]∈𝕊*, A:𝕊→𝕃

Output: **exists**∈𝕃, **ind**∈ℕ, **val**∈𝕊

Precondition: –

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

```
i:=0
exists:=false
i<length(X) and not exists
    i:=i+1
    exists:=A(X[i])
exists
T                         F
ind:=i                    –
```

Why: difference between the start indexes (0/1)

4. Search - example

Let's search for a course in which the student failed!

Specification

Input: $n \in \mathbb{N}$, $Mark[1..n] \in \mathbb{N}^n$, $A:\mathbb{N} \to \mathbb{L}$

Output: $fails \in \mathbb{L}$, $ci \in \mathbb{N}$

Precondition: $\forall i(1 \le i \le n): Mark[i] \in [1..5]$

Postcondition: $fails = (\exists i(1 \le i \le n) \boxed{: Mark[i]=1})$ and $fails \to 1 \le ci \le n$ and $Mark[ci]=1$

The attribute function

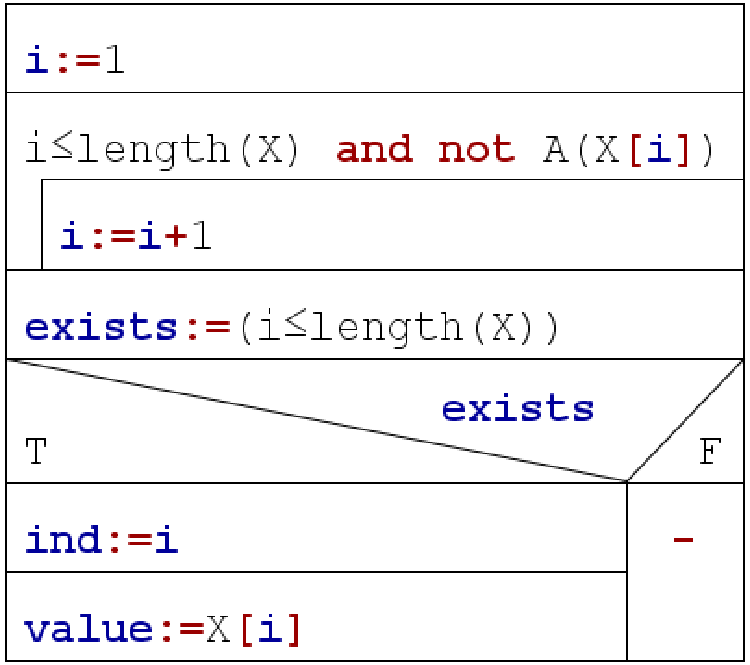
Short:

$$(fails, ci) = \underset{\substack{i=1 \\ Mark[i]=1}}{\overset{n}{\boldsymbol{SEARCH}}}(X[i])$$

4. Search - example

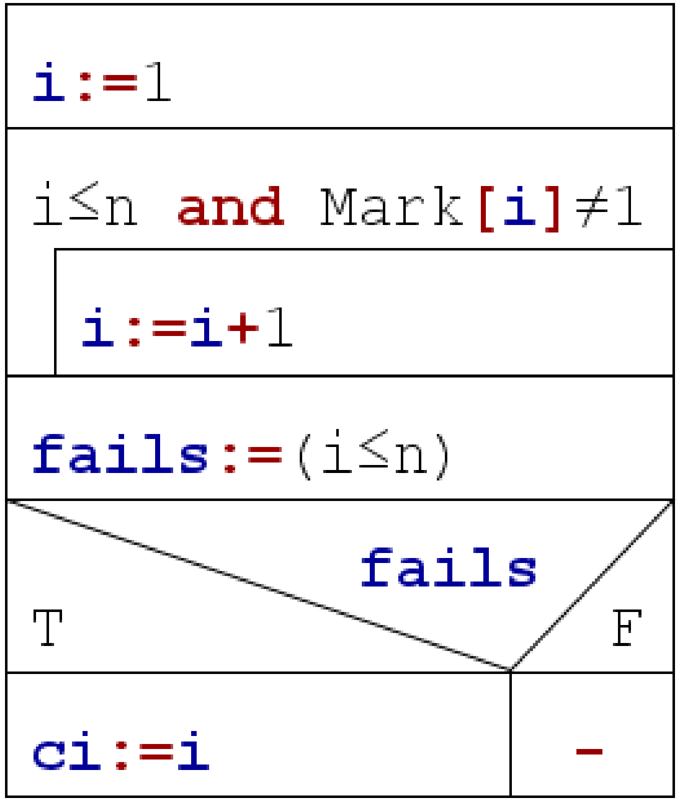
Let's search for a course in which the student failed!

Algorithm



length(X) → n
exists → fails
ind → ci

A(X[i]) → Mark[i]=1



5. Decision

Tasks (examples):

1. Let's **decide** if an integer is prime or not!
2. Let's **tell if** a given word is the name of a month!
3. Based on the final marks of a student, **let's determine** if he/she fails the semester!
4. **Let's find if** a given word contains a vowel!
5. **Let's decide** if a sequence is monotonically increasing!
6. Based on the final marks, **let's determine** if the student is excellent (all marks are the best).

5. Decision

What is common?

Let's determine if there is an item with given attribute in a sequence of „somethings”

A „narrowed” (output) version of search

5. Decision

Specification

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

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

Precondition: –

Postcondition: `exists=(∃i(1≤i≤length(X)):A(X[i]))`

Short: $$(exists) = \overset{length(X)}{\underset{i=1}{\exists}} (A(X[i]))$$

5. Decision

Algorithm$_1$

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

Algorithm$_2$

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

Why: difference between the start indexes (0/1)

5. Decision - all

Task Variant:

… all the items are of attribute A …

Specification (only the difference):

Output: `All∈𝕃`

Post condition: `All=(∀i(1≤i≤length(X)):A(X[i])`

`All=NOT((∃i(1≤i≤length(X)):NOT A(X[i]))`

short: $(all) = \underset{i=1}{\overset{length(X)}{\forall}} (A(X[i]))$

5. Decision - all

Algorithm

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

```
i:=1
i≤length(X) and A(X[i])
    i:=i+1
    all:=(i>length(X))
```

All=NOT((∃i(1≤i≤length(X)):NOT A(X[i]))

5. Decision - example

Based on the final marks of a student, let's determine if he/she fails the semester!

Specification

Input: $n \in \mathbb{N}$, $Mark[1..n] \in \mathbb{N}^n$, $A:\mathbb{N} \to \mathbb{L}$

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

Precondition: $\forall i (1 \le i \le n) : Mark[i] \in [1..5]$

Postcondition: $fails = (\exists i (1 \le i \le n) : \boxed{Mark[i]=1})$

The attribute function

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

length(X)	→	n
exists	→	fails
A(X[i])	→	Mark[i]=1

```
i:=1
i≤n and Mark[i]≠1
  i:=i+1
fails:=(i≤n)
```

6. Selection

Tasks (examples):

1. We know the monthly income and expenses of a person. His assets grow by the end of the year. Let's **give** a month, **when** his asset grows!
2. Let's **define** the least divisor of a non-1 integer!
3. Let's **find** a vowel in an English word!
4. Let's **define** the serial number of a month given by its name!

6. Selection

What is common?

We have a sequence of „somethings”, and we have to select **an** element that has a given attribute, and we know that at least one such element exists in the sequence.

This is the *version of search* when we do not have to prepare for the case when the element cannot be found.

6. Selection

Specification

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

Output: **ind**$\in \mathbb{N}$, **val**$\in \mathbb{S}$

Precondition: length(X)>0 and $\exists$i(1≤i≤N):A(X[i])

Postcondition: 1≤ind≤length(X) and A(X[ind])and val=X[ind]

Short:

$$(ind, val) = \underset{\substack{i=1 \\ A(X[i])}}{\overset{length(X)}{\boldsymbol{SELECT}}} (X[i])$$

6. Selection

Algorithm

```
i:=1
not A(X[i])
    i:=i+1
ind:=i
val:=X[i]
```

Comment:

We know more: this solution gives the very first element that is of attribute A – so the program does more than expected.

How can we find the last one?

Modify the algorithm

6. Selection - example

Let's find a vowel in an English word!

Specification

Input: $\texttt{Word}\in\mathbb{T}$

Output: $\texttt{vw}\in\mathbb{N}$

Precondition: $\texttt{length(Word)>0 and}$
$\exists(1\le i\le \texttt{length(Word)}):\texttt{isVowel(Word[i])}$

Postcondition: $1\le \texttt{vw} \le \texttt{length(Word) and isVowel(Word[vw])}$

$\mathbb{T}$ = set of texts (string)

The attribute function

Definition: $\texttt{isVowel}: \mathbb{C}\texttt{h} \to \mathbb{L}$ (Character → Boolean)

$\texttt{isVowel(ch)= capital(ch)}\in\{\texttt{'A'},\ldots,\texttt{'U'}\}$

6. Selection - example

Let's find a vowel in an English word!

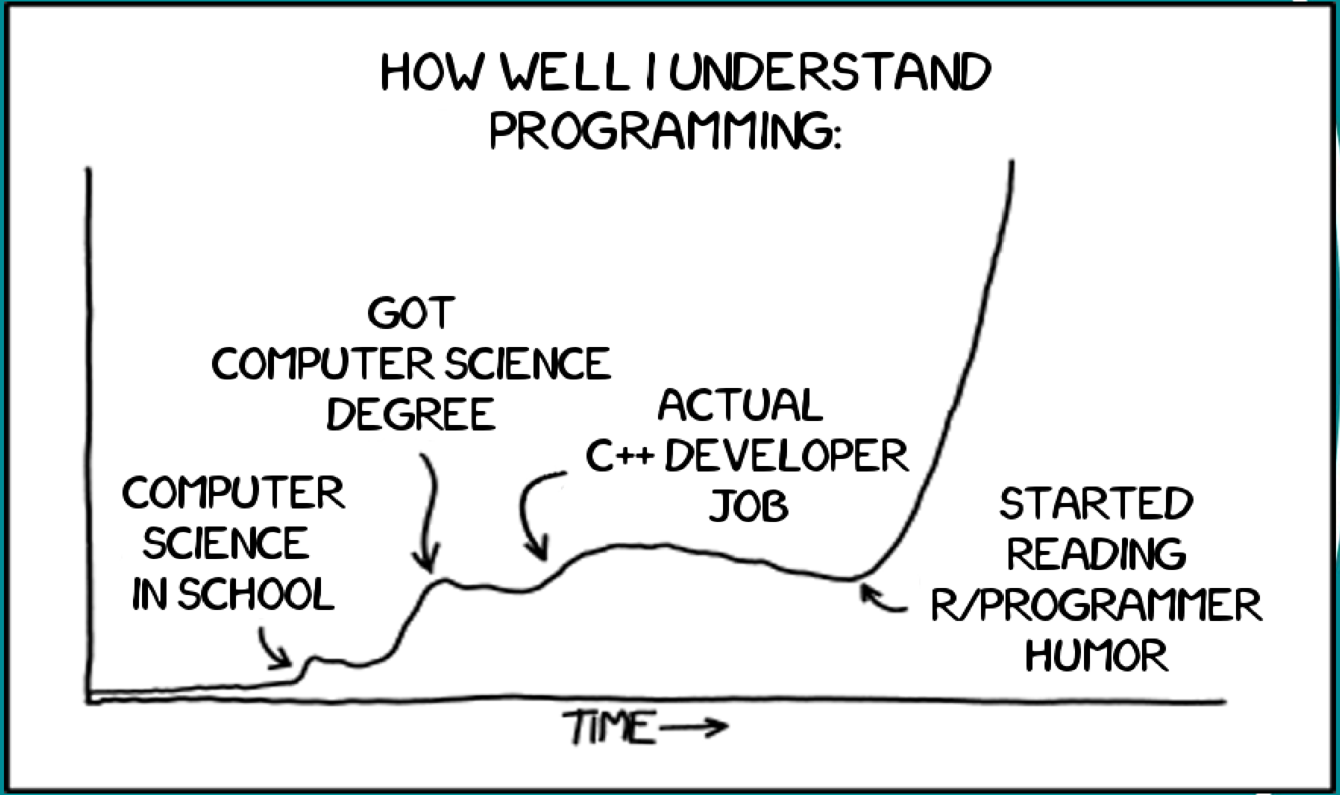
Algorithm

```
i:=1
not A(X[i])
    i:=i+1
ind:=i
val:=X[i]
```

ind → vw
A(X[i]) → isVowel(Word[i])

```
i:=1
not isVowel(Word[i])
    i:=i+1
vw:=i
```

Which pattern is used in isVowel function?



Thank you for your attention!