


Eötvös Loránd University
Faculty of Informatics

PROGRAMMING

2019/20 1st semester

Lecture 2

ZS. PLUHÁR, H. TORMA

Content

- Loops

  specification+algorithm+coding
  an introductory example for arrays
- The array
- Array instead of conditionals
- Constant arrays


Loops

Task: Let's calculate the non-1 smallest divisor of an integer (N>1)!

Specification:

Input: $N \in \mathbb{N}$

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

Precondition: $N>1$

Postcondition: $1<D \leq N$ and $D|N$ and
$\forall i\ (2 \leq i<D)\colon i \nmid N$

„for all i between 2 and D-1 is true that i does not divide N"

Loops

Representation of the solution:

Input: $N \in \mathbb{N}$

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

Precondition: $N>1$

Postcondition: $1<D \leq N$ and $D \mid N$ and $\forall i\ (2 \leq i<D): i \nmid N$

Variable

N: Integer

D: Integer

Declaring program variables

„Rule" of representation when switching from specification to representation:

$\mathbb{N} \rightarrow$ Integer

Loops

An idea for the solution:

Let's try 2; if not good then try 3; if it's still not good then try 4, ..., worst case N will be a divisor.

Algorithm expressing this:

Input: $N \in \mathbb{N}$

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

Precondition: $N>1$

Postcondition: $1<D \leq N$ and $D \mid N$ and $\forall i \ (2 \leq i<D): i \nmid N$

Smallest Divisor

```
i:=2
i ł N
    i:=i+1
D:=i
```

Variable
i:integer

Declaring the local variable

The role of variable i:
to go through numbers between 2 and N

Loops

Task: Let's calculate both the **non-1 smallest** and the **non-N greatest** divisor of an integer ($N>1$)!

Specification:

Input: $N \in \mathbb{N}$

Output: $SD, GD \in \mathbb{N}$

Precondition: $N>1$

Postcondition: $1<SD \leq N \quad 1 \leq GD<N$ and

$SD \mid N$ and $\forall i (2 \leq i < SD): i \nmid N$

and

$GD \mid N$ and $\forall i (GD < i < N): i \nmid N$

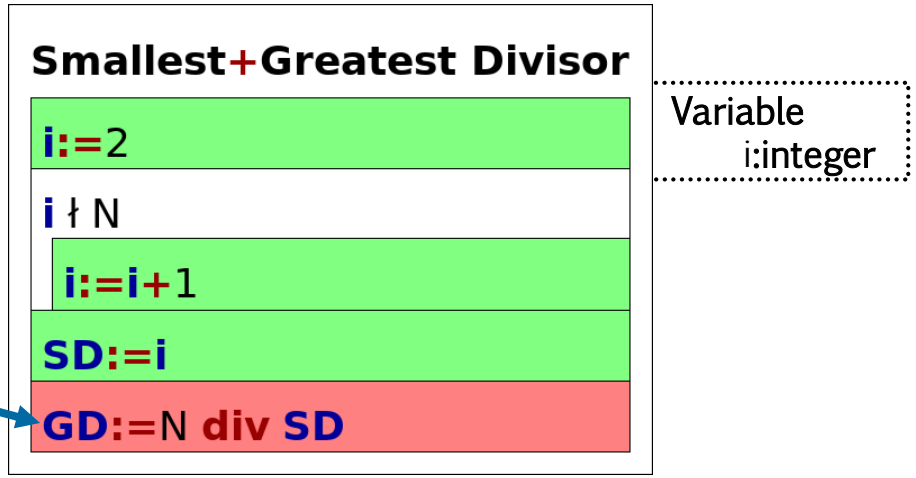
Loops

Note:

By knowing SD and GD, the post condition can be defined in another way: **SD*GD=N**

The algorithm based on this:

Input: $N \in \mathbb{N}$
Output: $SD, GD \in \mathbb{N}$
Precondition: $N > 1$
Postcondition: $1 < D \leq N \quad 1 \leq GD < N$ and
$SD \mid N$ and $\forall i\,(2 \leq i < SD): i \nmid N$ and
$SD * GD = N$



Loops

Task: Let's calculate the non-1 and non-N smallest divisor of and integer ($N>1$), if it exists.

Input: $N \in \mathbb{N}$

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

Precondition: $N>1$

Postcondition: $exists = \exists i (2 \leq i < D): i \mid N$ and

$exists \rightarrow 2 \leq D < N$ and $D \mid N$ and
$\forall i (2 \leq i < D): i \nmid N$

„there exists an i between 2 and D-1 that is true that i divides N"

Loops

Input: $N \in \mathbb{N}$

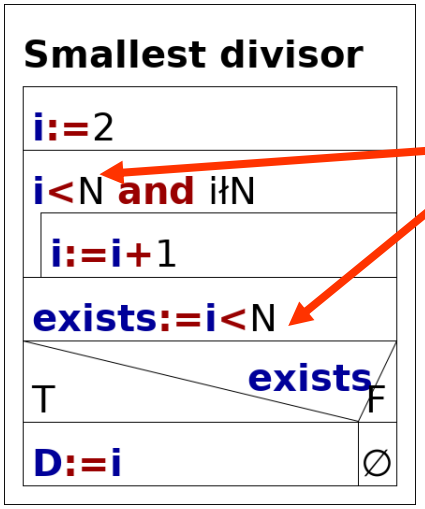
Output: $D \in \mathbb{N}$, $exists \in \mathbb{L}$

Precondition: $N>1$

Postcondition: $exists = \boxed{\exists i (2 \le i < D) : i \mid N}$ and

$exists \rightarrow \boxed{2 \le D < N \text{ and } D \mid N}$ and
$\forall i (2 \le i < N) : i \nmid N$

Algorithm:



$i \le \sqrt{N}$

$2 \rightarrow i \le (N \text{ div } i) \leftarrow N \text{ div } 2$
thus
$i*i \le N$
thus
$i \le \sqrt{N}$

Note:

Whenever i is a divisor of N, then (`N div i`) is also a divisor, so it's enough to check up to the square root of N

Loops

Task: Let's calculate the sum of all the divisors of an integer (N>1)

Specification:

Input: $N \in \mathbb{N}$

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

Precondition: $N>1$

Postcondition: $S = \sum_{\substack{i=1 \\ i|N}}^{N} i$

An example to understand conditional sum:
N=15 →
i=1 : (1 | 15) → S=1
i=2 : (2 ł 15) → S=1+0
i=3 : (3 | 15) → S=1+3
i=4 : (4 ł 15) → S=1+3+0
…
i=15: (15 | 15) → S=1+3+…+15

Loops

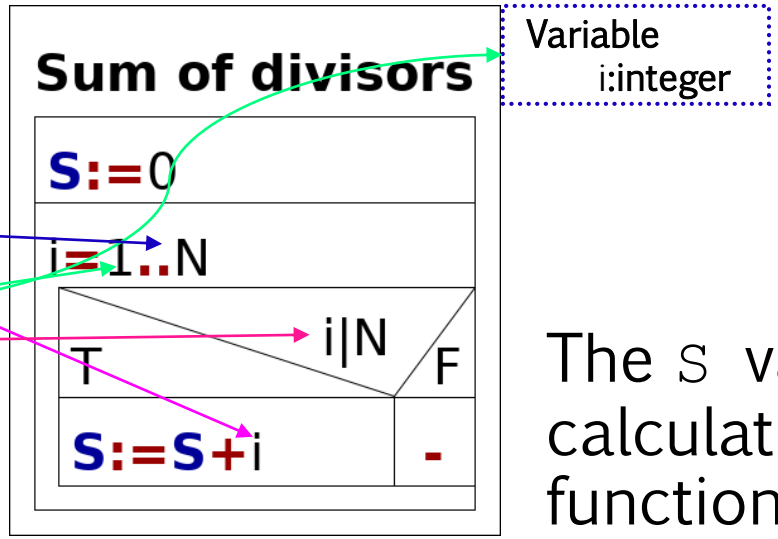
Algorithm:

Input: $N \in \mathbb{N}$

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

Precondition: $N>1$

Postcondition: $S = \sum_{i=1}^{N} 1$ $_{i|N}$



The `S` variable is not calculated with a function, but the result is stored in it.

Question:

Could we stop looping at `root(N)` again?
With `S:=S+i+(N div i)` value assignment?

Loops

Task: Let's calculate the sum of all odd divisors of an integer (N>1)

Specification:

Input: $N \in \mathbb{N}$

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

Precondition: $N>1$

Postcondition: $S = \sum_{i=1 \atop i|N \text{ and } \mathrm{odd}(i)}^{N} i$

odd(i)=???

Loops

Task: Let's calculate the sum of all odd divisors of an integer (N>1)

Specification:

Input: $N \in \mathbb{N}$

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

Precondition: $N>1$ **Postcondition:** $S=\sum_{\substack{i=1 \\ i|N \text{ and } odd(i)}}^{N} i$

odd(i)=???

Definition:

$odd: \mathbb{N} \rightarrow \mathbb{L}$

$odd(x) := (x \bmod 2) = 1$

Loops

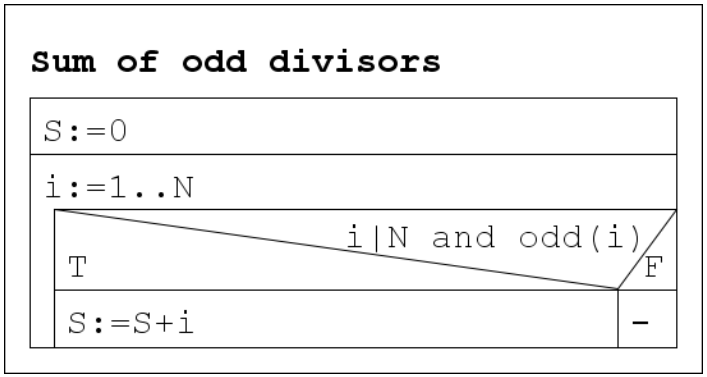
Input: $N \in \mathbb{N}$

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

Precondition: $N>1$

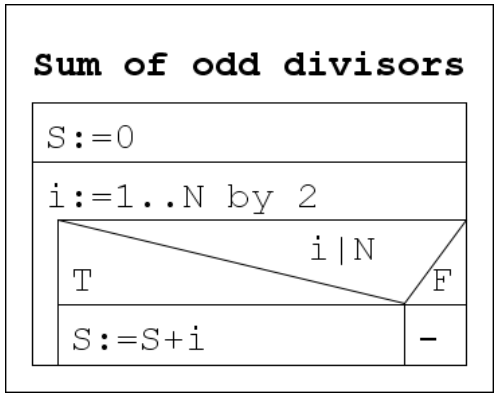
Postcondition: $S = \sum_{\substack{i=1 \\ i|N \text{ and } odd(i)}}^{N} i$

Algorithm1:



Variable
i:integer

Algorithm2:



Variable
i:integer

Loops

Task: Let's calculate the sum of all **prime** divisors of an integer (N>1)

Specification:

Input: $N \in \mathbb{N}$

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

Precondition: $N>1$ **Postcondition:** $S = \sum_{i=2,\ i|N \text{ and isprime}(i)}^{N} i$

isprime(i)=???

$$N = i_1^{m_1} * i_2^{m_2} *\ldots i_k^{m_k}$$

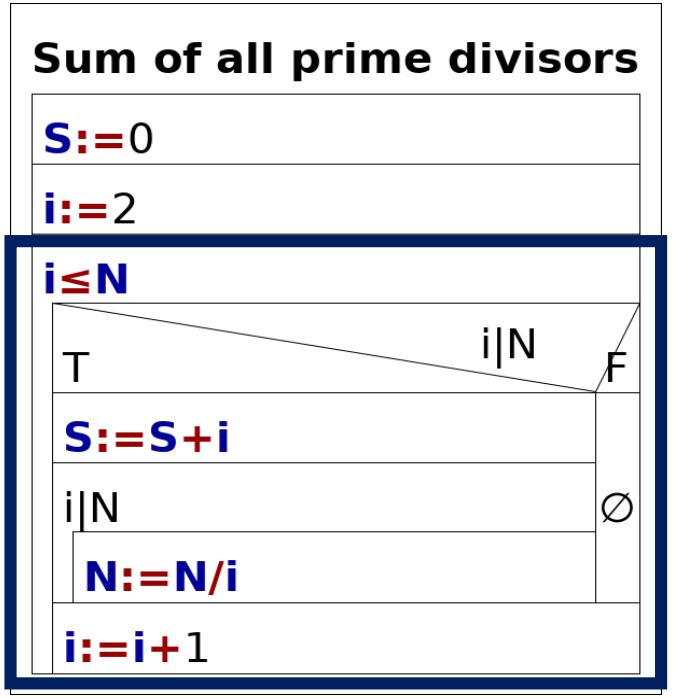
$$\downarrow$$

$$S = i_1 + i_2 + \ldots + i_k$$

Loops

Algorithm:

- The *least* divisor is a prime for sure; let's divide N by it as many times as we can; the *next* divisor of the reduced N will be *prime again*.
- Why don't we use a count loop as the outer loop?



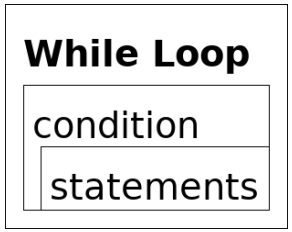
Loops

Let's observe that:

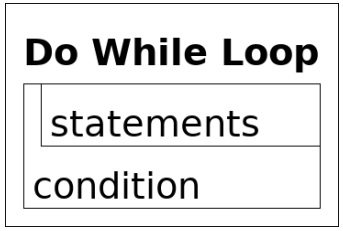
- If there is $\exists$, $\forall$ or Σ sign in post condition, then the solution always contains a **loop**.
- If there is $\exists$, or $\forall$ sign in post condition, then the solution is mostly a **conditional loop**.
- If there is Σ sign in post condition, then the solution is **mostly a for loop** (Π also).
- In the case of **two embedded** Σ signs, we will have **two for loops embedded** in each other.
- In the case of **conditional** Σ, there will be a **conditional statement within the loop**.

Loops – algorithm, code

Conditional loops:

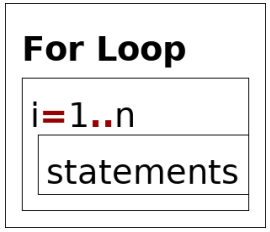



```
while (condition){
  statements
}
```

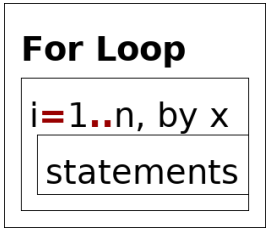



```
do{
  statement
}while (condition)
```

Count loops:




```
for (int i=1;i<=N;i++){
  statements
}
```




```
for (int i=1;i<=N;i+=x){
  statements
}
```

Task for conditional – or do we need something else?

Problem: The Japanese calendar contains 60 years cycles. The years are paired, and a color is assigned to each pair (green, red, yellow, white and black).

- 1, 2, 11, 12, …, 51, 52: green years
- 3, 4, 13, 14, …, 53, 54: red years
- 5, 6, 15, 16, …, 55, 56: yellow years
- 7, 8, 17, 18, …, 57, 58: white years
- 9, 10, 19, 20, …, 59, 60: black years

We know that the last cycle started in 1984, and will end in 2043. *Let's write a computer program which determines the color assigned to a given M year (1984≤M≤2043)!*

Task for conditional – or do we need something else?

- 1, 2, 11, 12, ..., 51, 52: green years
- 3, 4, 13, 14, ..., 53, 54: red years
- 5, 6, 15, 16, ..., 55, 56: yellow years
- 7, 8, 17, 18, ..., 57, 58: white years
- 9, 10, 19, 20, ..., 59, 60: black years

Specification1:

Input: $year \in \mathbb{N}$

A set that is not defined yet

Output: $c \in Color$

Precondition: $1984 \leq year \text{ and } year \leq 2043$

Postcondition:

((year-1984) mod 10) div 2=0 and c="green" or

((year-1984) mod 10) div 2=1 and c="red" or …

Definition:

$Color := (\text{"green"}, \text{"red"}, \text{"yellow"}, \text{"white"}, \text{"black"}) \subset \mathbb{T}$

We define the set Color, which is derived from $\mathbb{T}$ text set

Task for conditional – or do we need something else?

Specification2:

Input: $year \in \mathbb{N}$

Output: $c \in Color$

$Color = \{"green","red","yellow","white","black"\} \subset \mathbb{T}$

Color set can be defined here

Precondition: $1984 \leq year \text{ and } year \leq 2043$

Postcondition:

```
((year-1984) mod 10) div 2=0 and c="green" or
((year-1984) mod 10) div 2=1 and
c="red" or …
```

Task for conditional – or do we need something else?

Color set can be defined here

Specification2:

Input: `year∈ℕ`

Output: `c∈Color`

```
Color={"green","red","yellow","white","black"}
⊂ 𝕋
```

Precondition: `1984≤year and year≤2043`

Postcondition:

```
((year-1984) mod 10) div 2=0 → c="green" and

((year-1984) mod 10) div 2=1 →
c="red" and …
```

Task for conditional – or do we need something else?

Input: year∈ℕ

Output: c∈Color
Color={"green","red","yellow","white","black"} ⊂ 𝕊

Precondition: 1984≤year and year≤2043

Postcondition:

((year-1984) mod 10) div 2=0 → c="green" **and**

((year-1984) mod 10) div 2=1 → c="red" **and** …

Algorithm:

Declaring local variable

Variable y:integer

y:=((year-1984) Mod 10) Div 2				
y=0	y=1	y=2	y=3	y=4
c:="green'	c:="red"	c:="yellow"	c:="white"	c:="black"

Question:

- Would we do the same if we had 90 conditional branches?
- Before answering the question, we introduce a new data structure: **the array**.

Sequences

Concepts in the specification:

- Sequence: an ordered list of *same typed data elements*
- Element: we refer to the i^{th} element of the sequence with a subindex: $\mathbf{s_i}$
- Index: `1..SequenceLength`

Example:

- `MonthLengths`$\in\mathbb{N}^{12}$ – `MonthLengths` consists of 12 elements, the elements are natural numbers $\cong$(MonthLengths$_1$, … , MonthLengths$_{12}$)
- `Seasons`$\in\mathbb{S}^{4}$ – `Seasons` consists of 4 elements, the elements are character strings $\cong$(Seasons$_1$, Seasons$_2$, Seasons$_3$, Seasons$_4$)

Array

Concepts for the algorithm:

- Array: finite length sequence, we can define operations for the i^{th} element (we know the smallest and the greatest index, or the count of elements).
- Index: `1..N`, sometimes `0..N-1`, where `N` stands for the count of elements. Sometimes it is represented as `a..b` (`a≤b`).
- Operations on elements of an array: referencing an element, modification of an element (the element is selected by a given index)

Sequences → Arrays

Example$_1$:

In specification:

$X_{1..N}, Y_{1..N}, Z_{1..N} \in \mathbb{R}^N$ – example for declaration

$Z_1 = X_1 + Y_1$ – example for referencing

In algorithm:

`X,Y,Z:Array[1..N:Real]` – example for declaration

`Z[1]:=X[1]+Y[1]` – example for referencing

Sequences → Arrays

Example$_2$:

In specification:

$Str_{1..5} \in \mathbb{T}^5$ – example for declaration

Str_1="first word" – example for referencing

In algorithm:

```
Str:Array[0..4:String] – declaration example
Str[0]:="first word" – example for referencing
```

Sequences → Arrays

Example$_3$:

In the previous task, the set of colors is a sequence that consists of string constants:

$\texttt{Colors}_{1..5} \in \mathbb{T}^5 = (\texttt{"green"}, \texttt{"red"}, \texttt{"yellow"}, \texttt{"white"}, \texttt{"black"})$

In the algorithm, it can be represented as:

```
Constant Colors:Array[0..4:String]=
  ("green","red","yellow","white","black")
```

The equal count of elements is essential!

One should pay attention to the conversion of indexes.

Here: $\texttt{Colors}_{\texttt{i}} \rightarrow \texttt{Colors[i-1]}$


Array instead of conditionals

Specification:

Input: $year \in \mathbb{N}$

Output: $c \in Color$,

$Color = \{"green","red","yellow","white","black"\} \subset \mathbb{T}$

$Colors_{1..5} \in \mathbf{Color}^5 =$

("green","red","yellow","white","black")

Precondition: $1984 \leq year$ and $year \leq 2043$

Postcondition:

$c = Colors_{(((year-1984) \bmod 10) \operatorname{div} 2)+1}$

Array instead of conditionals

Specification (simplified):

Input: $year \in \mathbb{N}$

Output: $c \in \mathbb{T}$,

$Colors_{1..5} \in \mathbb{T}^5 =$

("green","red","yellow","white","black")

Precondition: $1984 \leq year \text{ and } year \leq 2043$

Postcondition:

$c = Colors_{(((year-1984) \text{ Mod } 10) \text{ Div } 2)+1}$

Array instead of conditionals

Input: year∈ℕ
Output: c∈𝕋,
$Colors_{1..5}$∈$\mathbb{T}^5$=
("green","red","yellow","white","black")

Declaration of program parameters

Algorithm:

Representation of data:

Variable
year: **Integer**
c: **Text**
Constant
Colors: **Array**[0..4:**String**]=

("green","red","yellow","white","black")

Array instead of conditionals

Input: year∈ℕ
Output: c∈𝕋,
$Colors_{1..5}$∈$\mathbb{T}^5$=
("green","red","yellow","white","black")
Precondition: 1984≤year and year≤2043
Postcondition:
c=$Colors_{(((year-1984)\ Mod\ 10)\ Div\ 2)+1}$

Algorithm:

```
y:=(((year-1984) Mod 10) Div 2)+1
c:=Colors[y-1]
```

Variable
y:Integer

Variable
year: Integer
c: Text
Constant
Colors: Array[0..4:String]=

("green","red","yellow","white","black")

Notice the option of simplification:

```
y:=((year-1984) Mod 10) Div 2
c:=Colors[y]
```

Variable
y:Integer

Array– algorithm, code

The C++ language starts indexing arrays at 0! But you could opt for not using the 0. element. ☺

Declaration examples:

```
X:Array[1..N:Real]
Y:Array[0..4:String]
```

in C++:

```
float X[N+1]
String X[5]
```

The previous colors example:

```
constant Colors:Array[0..4:String]=
("green","red","yellow","white","black")
```

```
const string Colors[5]=
{"green","red","yellow","white","black"};
```

Arrays- C++ code overview

Static arrays:

When size is known at compile time.

Declaration:

```
const int MaxN=???; // maxcount
type array[MaxN];  // array declaration
      //with indices 0..MaxN-1
…
```

This is a difference from usual specification

Referencing:

```
… array[ind]  // reference to an element
…
array[ind]=expr; // modification of elem.value
…
```

Arrays- C++ code overview

When size is known at compile time.

Static constant arrays:

Declaration:

```
const int N=???;//number of elements
const type array[N]={t1,t2,…,tN};
        //declaration of constant array,
        //with indices 0..N-1
```

or

```
const type array[]={t1,t2,…,tN};
        //declaration of const. array
const int N=sizeof array/sizeof(type);
        //number of elements
        //indices: 0..N-1
```

Arrays- C++ code overview

When size is only known at runtime.

Dynamic arrays$_1$:

Declaration:

```
int N;     //the real number of elements
 …
```

Creation:

```
N=???; // N to be defined, e.g. input
type array[N]; /* an array containing N
       elements with type "type" */
```

Referencing (the same):

```
… array[ind]  // reference to an element
…
array[ind]=expr; // modification of elem.value
…
```

Arrays- C++ code overview

When size is only known at runtime.

Dynamic arrays$_2$:

Declaration:

```
int N;     //the real number of elements
 …
```

Creation:

```
 N=???; // N to be defined, e.g. input
 type* array=new type[N]; /* allocating space
for N elements of type "type" */
```

Referencing (the same):

```
 … array[ind]  // reference to an element
 …
 array[ind]=expr; // modification of elem.value
 …
```

Using constant arrays

Task:

Let's create a computer program, which writes an integer between 1 and 99 with letters.

Specification:

Input: $N \in \mathbb{N}$

$ones_{1..20} \in \mathbb{T}^{20} = (\text{""}, \text{"one"}, \ldots, \text{"nineteen"})$

$tens_{1..9} \in \mathbb{T}^{9} = (\text{""}, \text{"twenty"}, \ldots, \text{"ninety"})$

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

Precondition: $1 \leq N \leq 99$

It's logical to put it here. From the point of the algorithm, the constant arrays are input.

Using constant arrays

Input: $N \in \mathbb{N}$

$ones_{1..20} \in \mathbb{T}^{20} = (\text{"","one",…,"nineteen"})$

$tens_{1..9} \in \mathbb{T}^{9} = (\text{"","twenty",…,"ninety"})$

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

Precondition: $1 \le N \le 99$

Postcondition:

$N<20 \rightarrow ones_{N+1}$

$N \ge 20 \rightarrow S = tens_{(N \text{ div } 10)} + ones_{(N \text{ mod } 10)+1}$

Using constant arrays

Input: $N \in \mathbb{N}$

$ones_{1..20} \in \mathbb{T}^{20} = (\text{"","one",…,"nineteen"})$

$tens_{1..9} \in \mathbb{T}^{9} = (\text{"","twenty",…,"ninety"})$

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

Precondition: $1 \le N \le 99$

Postcondition:

$N<20 \rightarrow ones_{N+1}$

$N \ge 20 \rightarrow S = tens_{(N \text{ div } 10)} + ones_{(N \text{ mod } 10)+1}$

Algorithm:

Declaring program parameters

```
Variable N:integer
constant ones:Array[0..19:String]=
                    ("","one",...,"nineteen")
constant tens:Array[0..8:String]=
                    ("","twenty",...,"ninety")
Variable S:Text
```

Taking index-shifting into account

Number to Text

N<20 T	N<20 F
S:=ones**[**N**]**	**S:=**tens**[(**N Div 10**)-1] +** ones**[**N **mod** 10**]**

Using constant arrays

Task:

Let's create a computer program that writes the serial number of a month based on the name of the month.

Specification:

Input: $H \in \mathbb{T}$

$MonName_{1..12} \in \mathbb{T}^{12} = (\text{"January"}, \ldots, \text{"December"})$

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

Precondition: $H \in MonName$

Postcondition: $1 \leq S \leq 12$ and $MonName_S = H$

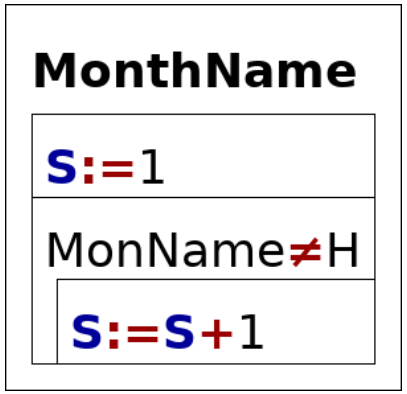
Using constant arrays

Specification:
Input: $H \in \mathbb{T}$
$MonName_{1..12} \in \mathbb{T}^{12} = (\text{"January"}, \ldots, \text{"December"})$
Output: $S \in \mathbb{N}$
Precondition: $H \in MonName$
Postcondition: $1 \leq S \leq 12$ and $MonName_S = H$

Declaring program parameters

Algorithm:

```
Variable          H:String  S:Integer
Constant  MonName:Array[1..12:String]=
                  ("January",...,"December")
```



Question:

What would happen if the precondition was not met?
Runtime error? Infinite loop?

Constant array – what do we store?

Task: We have a non leap year. What is the serial number of a given day (month, day)?

Specification$_1$:

Input: $M, D \in \mathbb{N}$

$$mon_{1..12} \in \mathbb{N}^{12} = (31, 28, 31, \ldots, 31)$$

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

Precondition: $1 \leq M \leq 12$ and $1 \leq D \leq mon_M$

Postcondition: $S = D + \sum_{i=1}^{M-1} mon_i$

Constant array – what do we store?

Input: $M, D \in \mathbb{N}$

$mon_{1..12} \in \mathbb{N}^{12} = (31,28,31,\ldots,31)$

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

Precondition: $1 \le M \le 12$ and $1 \le D \le mon_M$

Postcondition: $S = D + \sum_{i=1}^{M-1} mon_i$

Algorithm:

```
Variable M,D,S:Integer
Constant mon:Array[1..12:Integer]=(31,28,31,...,31)
```

Declaring program parameters

DayOfYear

```
S:=D
i=1..M-1
    S:=S+mon[i]
```

```
Variable
    i:integer
```

Declaring local variable

Note: In the case of a leap year, when H≥3, S should be increased by one. (The precondition should be modified, as well.)

Constant array – what do we store?

Another solution:

Let's store how many days there are before each month

Specification$_2$:

Input: … $mon_{1..12} \in \mathbb{N}^{12} = (0, 31, 59, 90, \ldots, 334)$

Postcondition: $S = mon_M + D$

Question: Is this a better solution? How do we express the precondition?