ELTE | FACULTY OF INFORMATICS

PROGRAMMING

Lecture 2

Zsuzsa Pluhár

pluharzs@inf.elte.hu

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: $\mathbf{1}<d\leq n$ and $\mathbf{d|n}$ and

$\forall i\ (2\leq i<d):\ i \nmid n$

„for all i between 2 and d-1 is true that i does not divide n”

The „ ∤” means not divisible:
a ∤ b means a does not divide b

Loops

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

Representation of the solution:

1. Declaring program variables

Specification:
Input: $n \in \mathbb{N}$
Output: $d \in \mathbb{N}$

Variables
n, d: **Integer**

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

$\mathbb{N} \rightarrow$ Integer (or variation)

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:

Specification:
Input: $n \in \mathbb{N}$
Output: $d \in \mathbb{N}$
Precondition: $n>1$
Postcondition: $1<d \leq n$ and $d|n$ and
$\boxed{\forall i\ (2 \leq i<d):\ i \nmid n}$

```
i:=2
i ∤ n
    i:=i+1
d:=i
```

Variables
n, d: **Integer**
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: $1<sd\leq n$ and $1\leq nd<n$ and $sd|n$ and $\forall i(1<i<sd): i \nmid n$ and $gd|n$ and $\forall i(gd<i<n): i \nmid n$

Loops

Note:

By knowing sd and gd, the postcondition can be defined in another way:

`sd*gd=n`

The algorithm based on this:

```
i:=2
i ∤ n
    i:=i+1
sd:=i
gd:=n div sd
```

Variables
  n, sd,gd: **Integer**
  i: Integer

Loops

Task: Let's calculate both the non-1 and the non-n smallest divisor of an integer (n>1), if it exists!

Specification:

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

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

Precondition: $n>1$

Postcondition: $exists = \exists i(2 \le i < n): i|n$ and

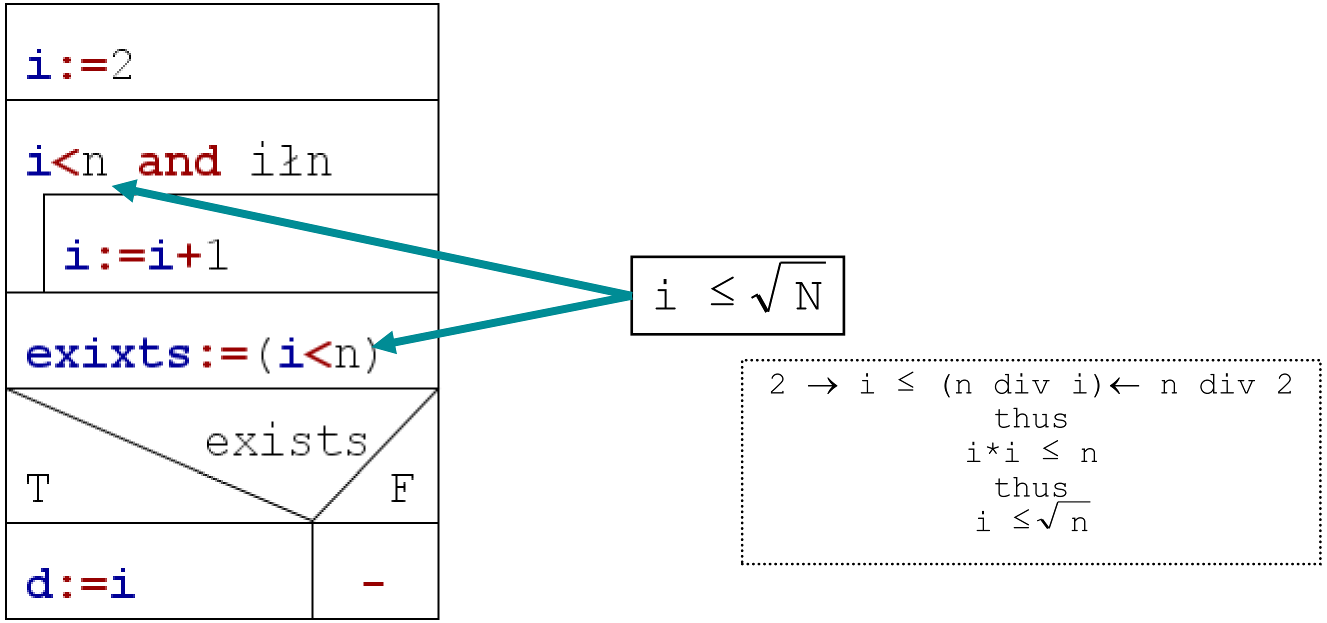
$exists \rightarrow 2 \le d < n$ and $d|n$ and

$\forall i(2 \le i < d): i \nmid n$

„there exists an i between 2 and d-1 that is true that i divides n"

Loops

Algorithm:



Note:

Whenever i is a divisor of d, 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∤15) → S=1+0

i=3 : (3|15) → S=1+3

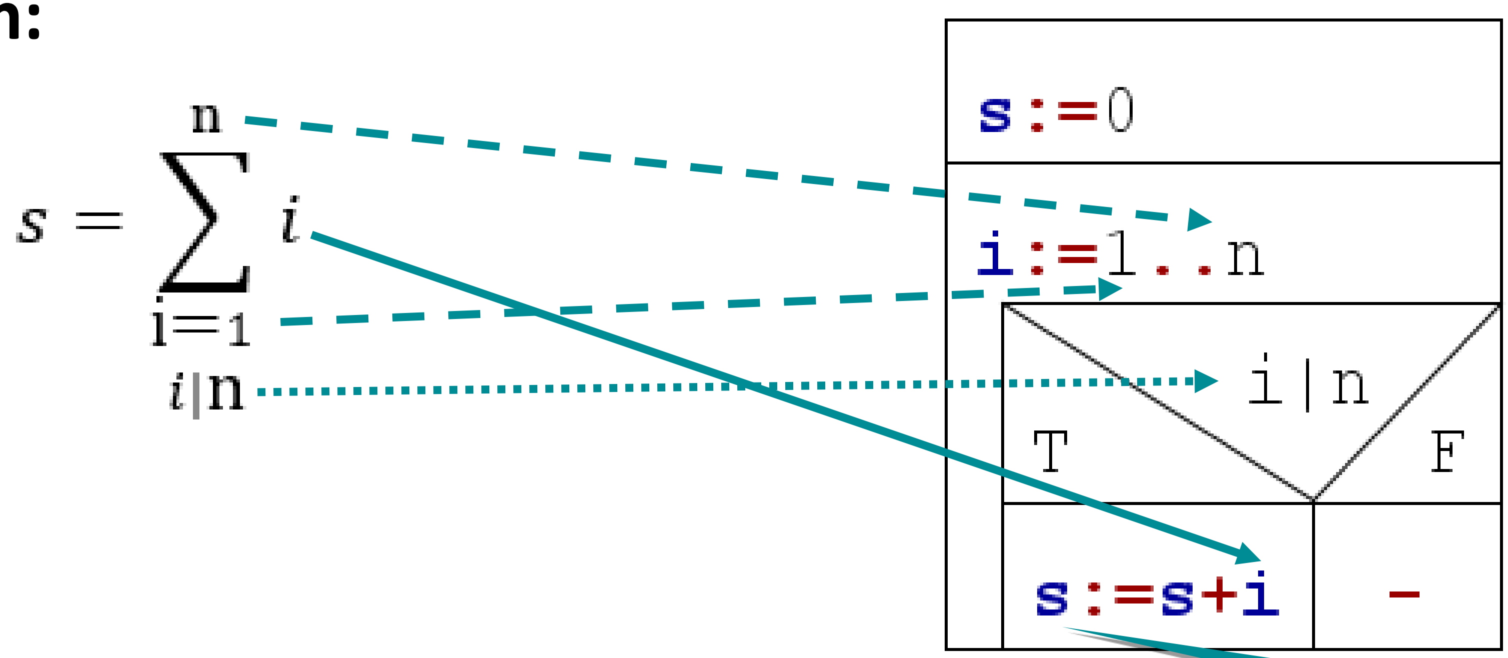
i=4 : (4∤15) → S=1+3+0

…

i=15: (15|15) → S=1+3+…+15

Loops

Algorithm:



we store the result in s variable

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:

$$s = \sum_{\substack{i=1 \\ i|n \text{ and } odd(i)}}^{n} i$$

odd(i)=????

Loops

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

Specification:

Input: n∈ℕ

Output: s∈ℕ

Precondition: n>1

Postcondition:

$$s = \sum_{\substack{i=1 \\ i|n \text{ and } odd(i)}}^{n} i$$

Definition:

odd: ℕ→𝕃

odd(x):=(x mod 2)=1

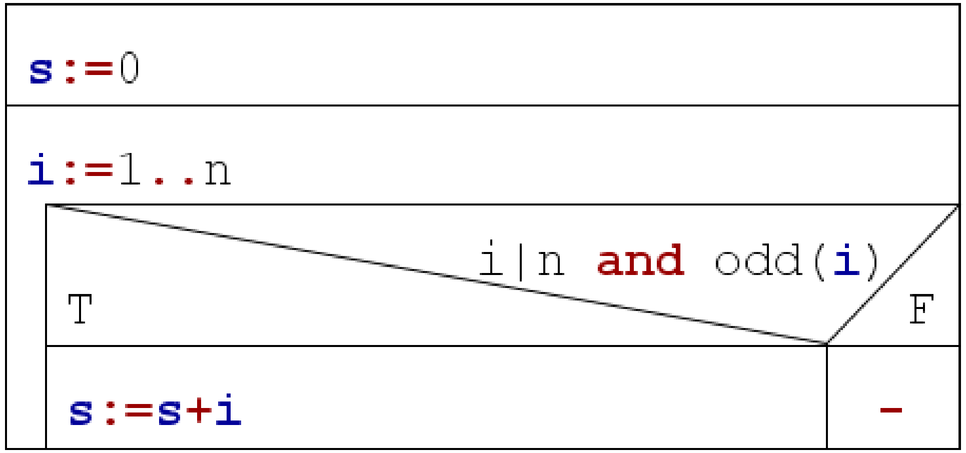
Loops

Specification:

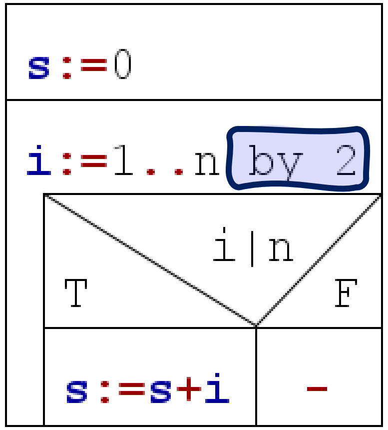
Input: n∈ℕ
Output: s∈ℕ
Precondition: n>1
Postcondition:

$$s = \sum_{\substack{i=1 \\ i|n \text{ and } odd(i)}}^{n} i$$

Algorithm$_1$:



Algorithm$_2$:



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:

$$s = \sum_{\substack{i=1 \\ i|n \text{ and } isprime(i)}}^{n} i$$

isprime=(i)=???

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

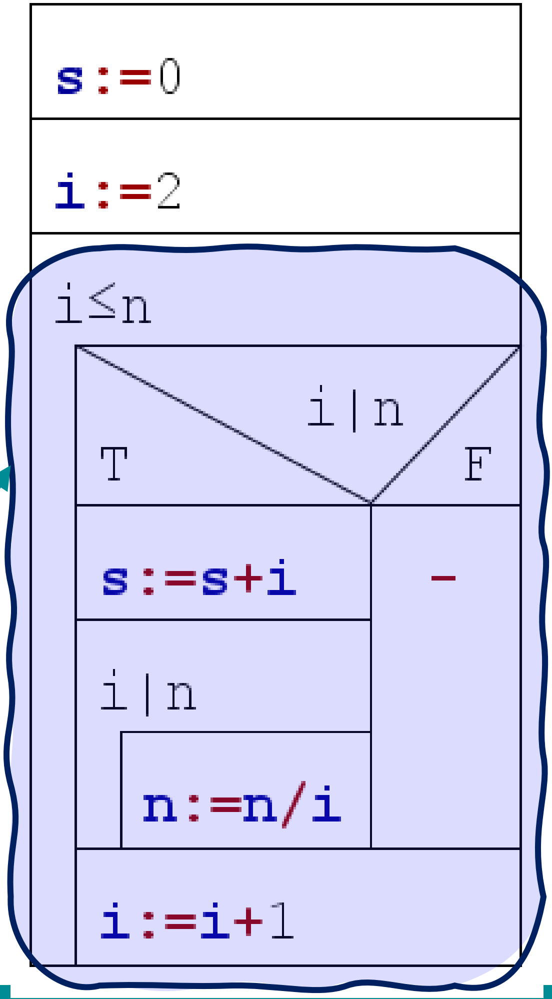
$$\downarrow$$

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

Loops

Algorithm:

- The *first* 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 **∃, ∀** or **Σ** sign in postcondition, then the solution always contains a **loop**.
- If there is **∃,** or **∀** sign in postcondition, then the solution is mostly a **conditional loop**.
- If there is **Σ** sign in postcondition, 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**.

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 that determines the color assigned to a given m year (1984≤m≤2043)*!

Task for conditional – or do we need something else?

Specification$_1$:

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

Output: $c \in Color$

Precondition: $1984 \leq year$ 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** …

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

Definition:

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

Task for conditional – or do we need something else?

Specification$_2$:

Input: `year∈ℕ`

Output: `c∈Color`

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

Color set can be defined here

Precondition: `1984≤year and year≤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?

Specification$_2$:

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

Output: $c \in Color$

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

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

Postcondition:

$((year-1984) \bmod 10)$ div $2=0 \rightarrow c="green"$ **and**

$((year-1984) \bmod 10)$ div $2=1 \rightarrow c="red"$ **and** …

Task for conditional – or do we need something else?

Algorithm:

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?

array

Sequences

Algorithm:

- **Sequence**: an ordered list of same typed data elements
- **Element**: reference to the i^th^ element of the sequence: **`S[i]`**
- **Index**: `1..SequenceLength`

Example:

`MonthLengths[1..12]`$\in\mathbb{N}^{12}$ – `MonthLengths` consists of 12 elements, indexed from 1 to 12, the elements are natural numbers $\cong$(MonthLengths$_1$, … , MonthLengths$_{12}$),

`Seasons[1..4]`$\in\mathbb{T}^{4}$ – `Seasons` consists of 4 elements, indexed from 1 to 4, the elements are character strings $\cong$(Seasons$_1$, Seasons$_2$, Seasons$_3$, Seasons$_4$)

Array

- **Array**: finite length sequence, (the smallest and the greatest index, or the count of elements are known).
- **Index**: `1..n`, sometimes `0..n−1`, where `n` stands for the count of elements. Sometimes it is represented as `a..b` ($a \le b$).
- **Operations** on elements of an array:
  - referencing an element: `X[i]`
  - modification of an element (the element is selected by a given index)

Sequence ⟶ Array

Example:

In specification:

`X,Y,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

Array

Color-task example:

In specification:

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

In algorithm:

```
Constant
Colors:Array[1..5:String]=("green","red","yellow","white","black")
```

The equal count of elements is essential!

Array instead of conditionals

Specification:

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

Output: $c \in Color$

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

$\boxed{\textbf{Colors[1..5]} \in \textbf{Color}^5 \textbf{= ("green","red","yellow","white","black")}}$

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

Postcondition: $c = Colors[((year-1984) \bmod 10) \text{ div } 2)\boxed{\mathbf{+1}}]$

Array instead of conditionals

Specification:

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

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

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**]

Array instead of conditionals

Representation of data:

Variables
  year: **Integer**
  c: **String**
Constant
  Colors: **Array[**1..5**:String]=**("green","red","yellow","white","black")

Algorithm

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

Array – algorithm → code

Some languages start indexing arrays at 0! → SOLUTION$_1$

Declaration examples:

```
X:Array[1..n:Real]
```

in C#:

```
float X=new float[n+1];
```

The X[0] will be not used

Algorithm:

```
i=1..n
  x[i]:=i
```

in C#:

```
for (int i = 1; i <= n; i++){
  X[i]=i;
}
```

Array – algorithm → code

Some languages start indexing arrays at 0! → SOLUTION$_2$

Declaration examples:

```
X:Array[1..n:Real]
```

in C#:

```
float X=new float[n];
```

Algorithm:

```
i=1..n
   x[i]:=i
```

Change the code

in C#:

```
for (int i = 0; i < n; i++){
   X[i]=i+1;
}
```

Using constant arrays

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

Specification:

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

$Ones[1..20] \in \mathbb{T}^{20} = („”, ”one”, …, ”nineteen”)$

$Tens[1..9] \in \mathbb{T}^{9} = („”, ”twenty”, …, ”ninety”)$

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

Precondition: $1 < n \leq 99$

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

Using constant arrays

Specification:

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

$Ones[1..20] \in \mathbb{T}^{20}$ = („”,”one”,…,”nineteen”)

$Tens[1..9] \in \mathbb{T}^{9}$ = („”,”twenty”,…,”ninety”)

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

Precondition: $1 < n \leq 99$

Postcondition:

$n<20 \rightarrow s=Ones[n+1]$ and

$n \geq 20 \rightarrow s=Tens[(n \text{ div } 10)+ones(n \text{ mod } 10)+1]$

Using constant arrays

Specification:

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

Ones[1..20]$\in \mathbb{T}^{20}$=(„”,”one”,…,”nineteen”)

Tens[1..9]$\in \mathbb{T}^{9}$=(„”,”twenty”,…,”ninety”)

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

Precondition: $1<n\leq 99$

Postcondition:

n<20 → s=Ones[n+1] and

n≥20 → s=Tens[(n div 10)+ones(n mod 10)+1]

Declaring program parameters

Variable
  n:**Integer**
  constant Ones:**Array**[1..20:**String**]=("","one",...,"nineteen")
  constant Tens:**Array**[1..9:**String**]=("","twenty",...,"ninety")
  s:**String**

Algorithm:

n<20	
T	F
s:=Ones[n+1]	s:=Tens[n div 10]+Ones[(n mod 10) +1]

Code it!

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] $\in \mathbb{T}^{12}$=(„January”,…,”December”)

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

Precondition: h∈MonName

Postcondition: $1 \le s \le 12$ and MonName[s]=h

Using constant arrays

Declaring program parameters

Variable
  h:**String**
  s: **Integer**
  constant MonNames:**Array**[1..12:**String**]=(„January",...,„December")

Algorithm:

```
s:=1
MonName[s]≠h
    s:=s+1
```

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

Using constant arrays – what do we store?

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

Specification:

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: $s = d + \sum_{i=1}^{m-1} Mon[i]$

Using constant arrays – what do we store?

Declaring program parameters

Variable
  m,d,s: **Integer**
  constant Mon:**Array**[1..12:**Integer**]=(31,28,31,…,31)

Algorithm:

```
s:=d
i=1..(m-1)
    s:=s+Mon[i]
```

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

Using constant arrays – what do we store?

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

Specification$_2$:

Input: $m, d \in \mathbb{N}$

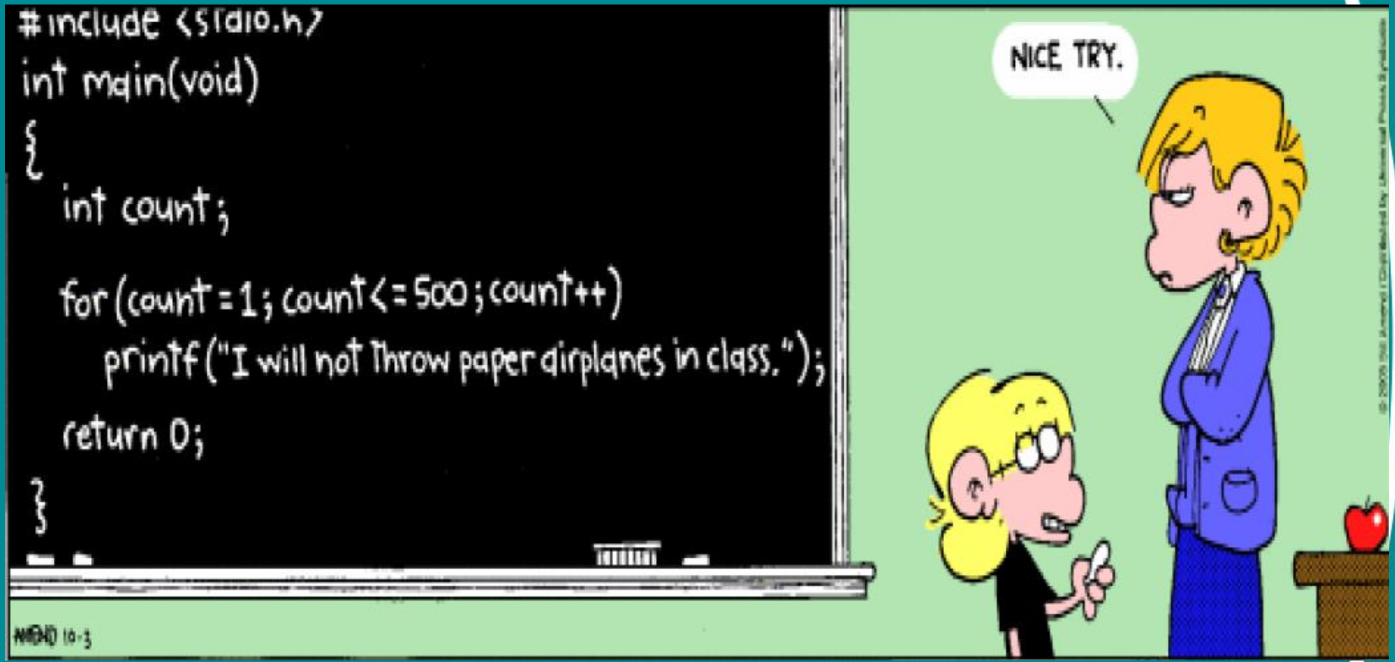
$\boxed{\texttt{Mon[1..12]} \in \mathbb{N}^{12} = (0,31,59,\dots,334)}$

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

Precondition: $1 \le m \le 12$ and $1 \le d \le \boxed{??}$

Postcondition: $\boxed{s = \texttt{Mon[m]} + d}$

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



Thank you for your attention!