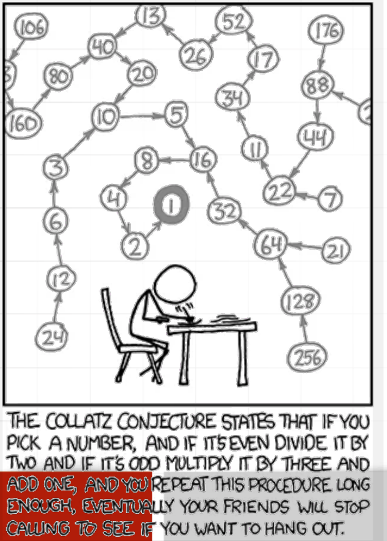

THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

Ways of Presenting Algorithms (Algorithms Part II)

ver. 14 z drobnymi modyfikacjami!

Wojciech Myszka

2023-11-21 07:29:41 +0100

Wrocław University of Science and Technology

Expressing/Presenting algorithms

1. **Natural language.** Use simple sentences in the imperative mood.

Expressing/Presenting algorithms

1. **Natural language.** Use simple sentences in the imperative mood.
2. **Flowcharts.** *In a moment...*

Expressing/Presenting algorithms

1. **Natural language.** Use simple sentences in the imperative mood.
2. **Flowcharts.** *In a moment...*
3. **Decision tables.** *A little bit later...*

Expressing/Presenting algorithms

1. **Natural language.** Use simple sentences in the imperative mood.
2. **Flowcharts.** *In a moment. . .*
3. **Decision tables.** *A little bit later. . .*
4. **Pseudocode** — formal type of writing, similar to. . .

Expressing/Presenting algorithms

1. **Natural language.** Use simple sentences in the imperative mood.
2. **Flowcharts.** *In a moment. . .*
3. **Decision tables.** *A little bit later. . .*
4. **Pseudocode** — formal type of writing, similar to. . .
5. **Programing languages.** *Never. . . ?!* (But remember about Blockly. . .)

Algorithm

Algorithm

a step-by-step procedure for solving a problem or accomplishing some end

Expressing algorithms

Flowchart

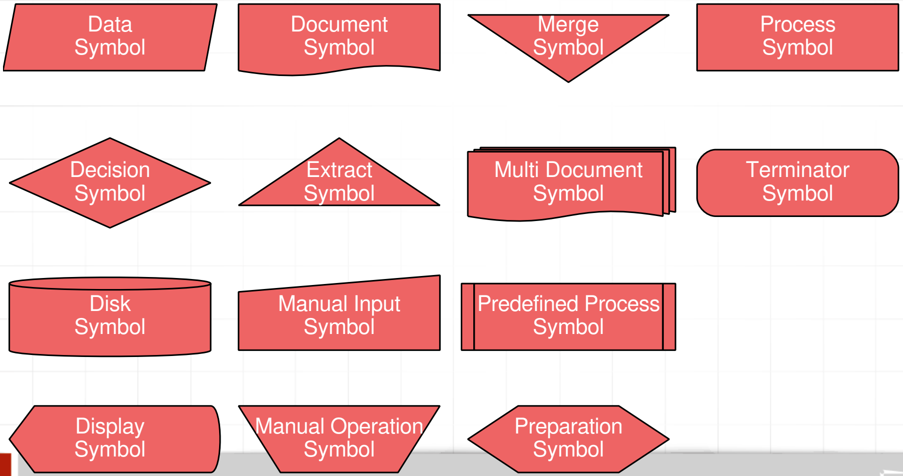
(also *block diagram*) A flowchart is a type of diagram that represents an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting these with arrows. This diagrammatic representation can give a step-by-step solution to a given problem. Process operations are represented in these boxes, and arrows connecting them represent the flow of control.

Block diagram reveals important steps in the algorithm and the logical relationships between them.

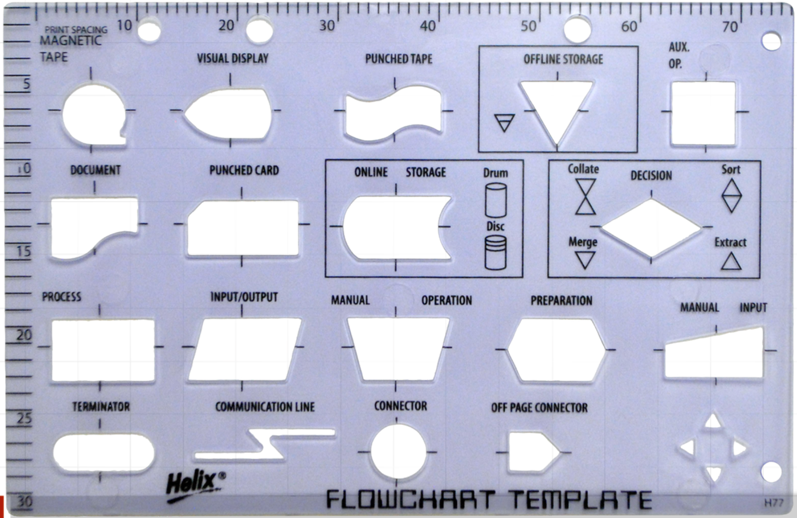
Introduced in the early twenties of the XX century. Used, among others, by von Neumann.

Symbols used in the flowcharts

Common shapes



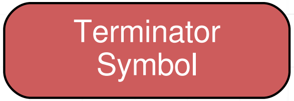
Flowchart template



Flowchart

Terminator Symbol

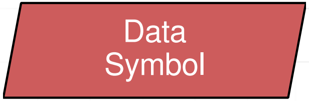
Terminator symbol signals the start or end of a process. It usually contains the word “Start”, “Begin” or “Stop”, “End”.



Flowchart

Data symbol

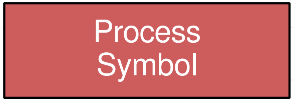
Data Symbol (also Input/Output) The Data flowchart shape indicates inputs to and outputs from a process. As such, the shape is more often referred to as an I/O shape than a Data shape. **READ z**, **WRITE z+10**.



Flowchart

Process

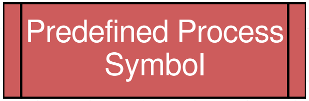
Process Symbol Generic processing step. Examples: “Add 1 to z” or “$\mathbf{z = z + 1}$”; “replace identified part”; “save changes” or similar.



Flowchart

Predefined process

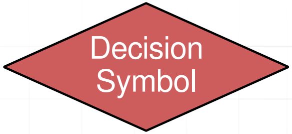
Predefined process (Subroutine) is used to show complex processing steps which may be detailed in a separate flowchart.



Flowchart

Decision Symbol

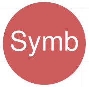
Decision Symbol shows where a decision is necessary, commonly a Yes/No question or True/False test. For example: Check if **a is equal to b**, if YES do something if NO do something else.



Flowchart

Connector

Connector If you need to connect to another page or another section of the chart, and can't draw a line, you can use a circle. You draw the line to the circle and label the circle with a letter or other symbol: **4.3**, **2**, **B2**.



Selecting the highest person in the room

Natural language

1. Select any person from the audience, think of it as the highest (and place it near the door).

Selecting the highest person in the room

Natural language

1. Select any person from the audience, think of it as the highest (and place it near the door).
2. Are there any people left in the room? if so — go to step 5.

Selecting the highest person in the room

Natural language

1. Select any person from the audience, think of it as the highest (and place it near the door).
2. Are there any people left in the room? if so — go to step 5.
3. If not — the highest person is one standing near the door.

Selecting the highest person in the room

Natural language

1. Select any person from the audience, think of it as the highest (and place it near the door).
2. Are there any people left in the room? if so — go to step 5.
3. If not — the highest person is one standing near the door.
4. End of the algorithm

Selecting the highest person in the room

Natural language

1. Select any person from the audience, think of it as the highest (and place it near the door).
2. Are there any people left in the room? if so — go to step 5.
3. If not — the highest person is one standing near the door.
4. End of the algorithm
5. Take another person from the room.

Selecting the highest person in the room

Natural language

1. Select any person from the audience, think of it as the highest (and place it near the door).
2. Are there any people left in the room? if so — go to step 5.
3. If not — the highest person is one standing near the door.
4. End of the algorithm
5. Take another person from the room.
6. Compare it with one standing near the door — is it higher?

Selecting the highest person in the room

Natural language

1. Select any person from the audience, think of it as the highest (and place it near the door).
2. Are there any people left in the room? if so — go to step 5.
3. If not — the highest person is one standing near the door.
4. End of the algorithm
5. Take another person from the room.
6. Compare it with one standing near the door — is it higher?
7. If not go to step 2

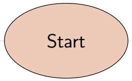
Selecting the highest person in the room

Natural language

1. Select any person from the audience, think of it as the highest (and place it near the door).
2. Are there any people left in the room? if so — go to step 5.
3. If not — the highest person is one standing near the door.
4. End of the algorithm
5. Take another person from the room.
6. Compare it with one standing near the door — is it higher?
7. If not go to step 2
8. If so — replace the person standing near the door, go to step 2

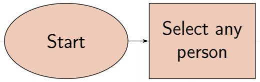
Selecting the highest person in the room

Flowchart



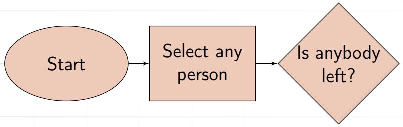
Selecting the highest person in the room

Flowchart



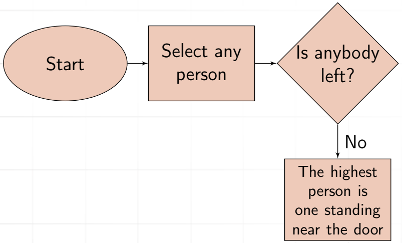
Selecting the highest person in the room

Flowchart



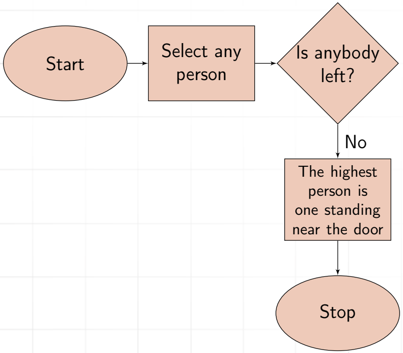
Selecting the highest person in the room

Flowchart



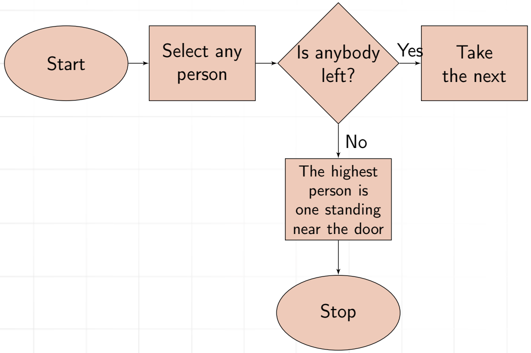
Selecting the highest person in the room

Flowchart



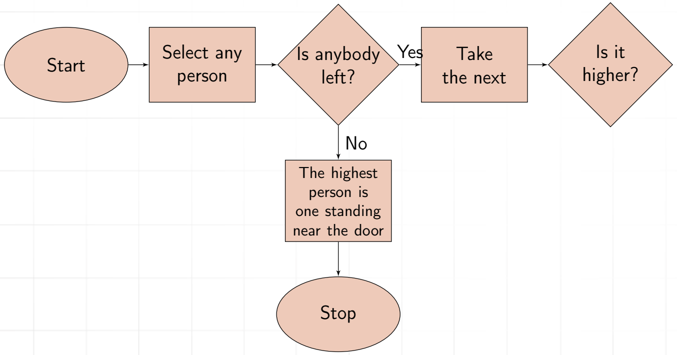
Selecting the highest person in the room

Flowchart



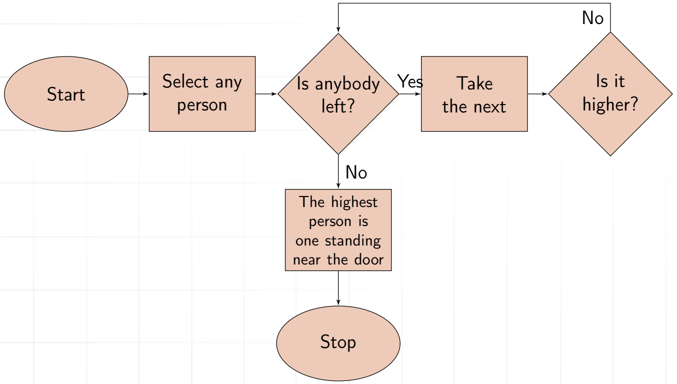
Selecting the highest person in the room

Flowchart



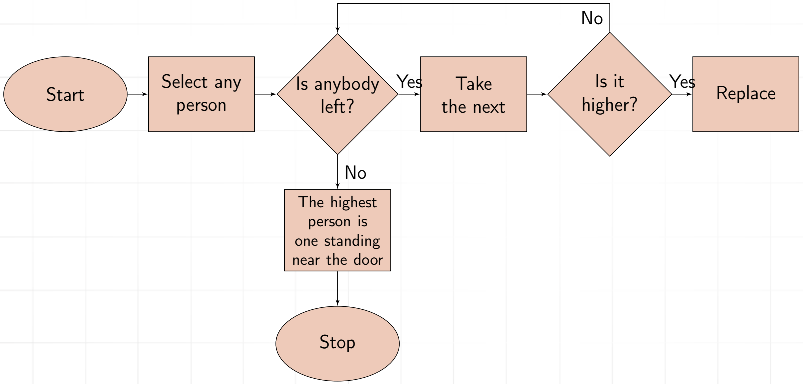
Selecting the highest person in the room

Flowchart



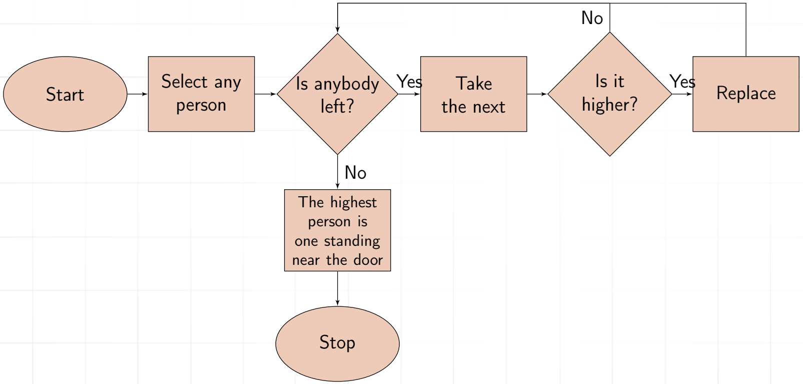
Selecting the highest person in the room

Flowchart

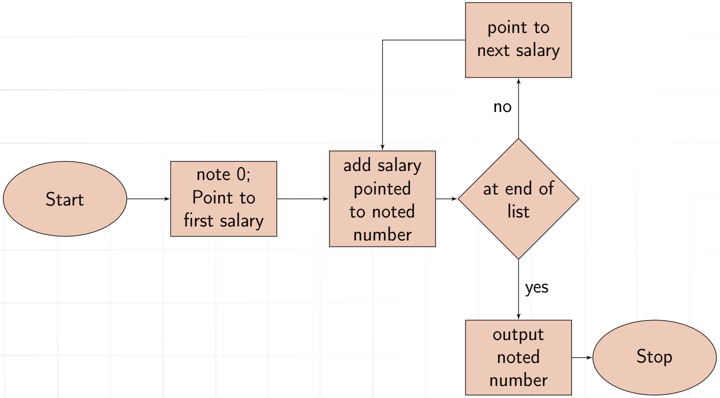


Selecting the highest person in the room

Flowchart



Salary



1. make a note of the number 0;
2. proceed through the list, adding each employee's salary to the noted number;
3. having reached the end of the list, produce the noted number as output.

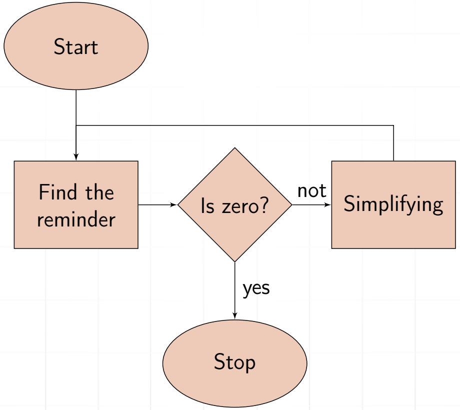
Euclidean Algorithm

Natural Language

1. [Find the reminder] Divide m by n let r be the reminder. (We have $0 \leq r < n$.)
2. [Is zero?] If $r = 0$ finish the procedure; the answer is n.
3. [Simplifying] Let $m \leftarrow n$, $n \leftarrow r$ and came back to the step 1.

Euclidean Algorithm

Flowchart



Decision table

- An alternative to the flowchart.
- Tables are a quick and easy way for humans to read, understand and execute a complex procedure.
- Work best with complex decision problems.
- It is not suitable to describe computational problems.
- Today it is a little forgotten.

Decision table cont.

The idea comes from the ’50s (20th century!)

Structure of DT

Conditions	Condition alternatives
Actions	Action entries

- Each decision corresponds to a variable whose possible values are listed among the condition alternatives.
- Each action is a procedure or operation to perform, and the entries specify whether (or in what order) the action is to be performed for the set of condition alternatives the entry corresponds to.

Example

		Rules							
Conditions	Printer does not print	Y	Y	Y	Y	N	N	N	N
	A red light is flashing	Y	Y	N	N	Y	Y	N	N
	Printer is unrecognized	Y	N	Y	N	Y	N	Y	N
Actions	Check the power cable			X					
	Check the printer-computer cable	X		X					
	Ensure printer software is installed	X		X		X		X	
	Check/replace ink	X	X			X	X		
	Check for paper jam		X		X				

Buing a car

	an excessive price	price OK	underestimated price
The car meets all requirements			
The car does not meet all the requirements but the condition and equipment are acceptable			
The car does not meet the requirements			

Buing a car

	an excessive price	price OK	underestimated price
The car meets all requirements	Bargain; if you do not succeed come back the next day and bargain again; buy even if you can not reduce the price		
The car does not meet all the requirements but the condition and equipment are acceptable			
The car does not meet the requirements			

Buing a car

	an excessive price	price OK	underestimated price
The car meets all requirements	Bargain; if you do not succeed come back the next day and bargain again; buy even if you can not reduce the price	Bargain; Buy It whatever the result of negotiations	
The car does not meet all the requirements but the condition and equipment are acceptable			
The car does not meet the requirements			

Buing a car

	an excessive price	price OK	underestimated price
The car meets all requirements	Bargain; if you do not succeed come back the next day and bargain again; buy even if you can not reduce the price	Bargain; Buy It whatever the result of negotiations	Buy
The car does not meet all the requirements but the condition and equipment are acceptable			
The car does not meet the requirements			

Buing a car

	an excessive price	price OK	underestimated price
The car meets all requirements	Bargain; if you do not succeed come back the next day and bargain again; buy even if you can not reduce the price	Bargain; Buy It whatever the result of negotiations	Buy
The car does not meet all the requirements but the condition and equipment are acceptable	Resign		
The car does not meet the requirements			

Buing a car

	an excessive price	price OK	underestimated price
The car meets all requirements	Bargain; if you do not succeed come back the next day and bargain again; buy even if you can not reduce the price	Bargain; Buy It whatever the result of negotiations	Buy
The car does not meet all the requirements but the condition and equipment are acceptable	Resign	Bargain; Buy if the price is reduced	
The car does not meet the requirements			

Buing a car

	an excessive price	price OK	underestimated price
The car meets all requirements	Bargain; if you do not succeed come back the next day and bargain again; buy even if you can not reduce the price	Bargain; Buy It whatever the result of negotiations	Buy
The car does not meet all the requirements but the condition and equipment are acceptable	Resign	Bargain; Buy if the price is reduced	Bargain; Buy regardless of the results the negotiations
The car does not meet the requirements			

Buing a car

	an excessive price	price OK	underestimated price
The car meets all requirements	Bargain; if you do not succeed come back the next day and bargain again; buy even if you can not reduce the price	Bargain; Buy It whatever the result of negotiations	Buy
The car does not meet all the requirements but the condition and equipment are acceptable	Resign	Bargain; Buy if the price is reduced	Bargain; Buy regardless of the results the negotiations
The car does not meet the requirements	Resign		

Buing a car

	an excessive price	price OK	underestimated price
The car meets all requirements	Bargain; if you do not succeed come back the next day and bargain again; buy even if you can not reduce the price	Bargain; Buy It whatever the result of negotiations	Buy
The car does not meet all the requirements but the condition and equipment are acceptable	Resign	Bargain; Buy if the price is reduced	Bargain; Buy regardless of the results the negotiations
The car does not meet the requirements	Resign	Resign	

Buing a car

	an excessive price	price OK	underestimated price
The car meets all requirements	Bargain; if you do not succeed come back the next day and bargain again; buy even if you can not reduce the price	Bargain; Buy It whatever the result of negotiations	Buy
The car does not meet all the requirements but the condition and equipment are acceptable	Resign	Bargain; Buy if the price is reduced	Bargain; Buy regardless of the results the negotiations
The car does not meet the requirements	Resign	Resign	Bargain the extras; Buy if the price of the extras is reasonable

Dressing a coat

		R1	R2	R3
C1	Raining	Y	Y	N
C2	Cold	Y	N	Y
A1	Dress warm raincoat	X		
A2	Dress raincoat		X	
A3	Dress warm coat			X

Dressing the coat cont.

In the table we have omitted the condition:
Raining N
Cold N
it does not require any special action, although it could be added by defining the action: 'Do not wear any coat".

Internet shop (B2B)

Another example is the decision table describing the activities related to the adoption and execution of the contract.

The table includes also the shop policy that can be described as follows:

1. The company serves registered customers **only**.
2. The company supplies goods from the list of goods **only**.

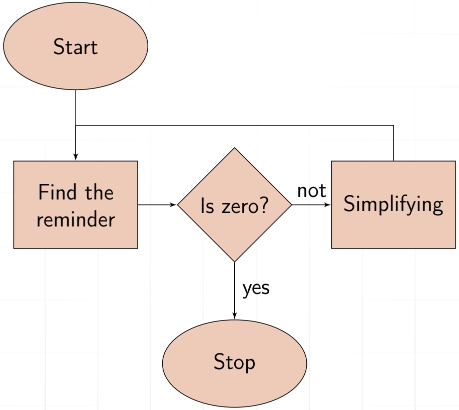
Internet schop cont.

C1	Item on the list?	Y	N	—	Y
C2	Customer registered?	Y	—	N	Y
C3	Sufficient supply of goods?	Y	—	—	N
A1	Book the item	X			
A2	Log the transaction	X			
A3	Save the item for the list to realize				X
A4	Deliver the item	X			
A5	Reject the transaction		X	X	

"—" means *Do not care*

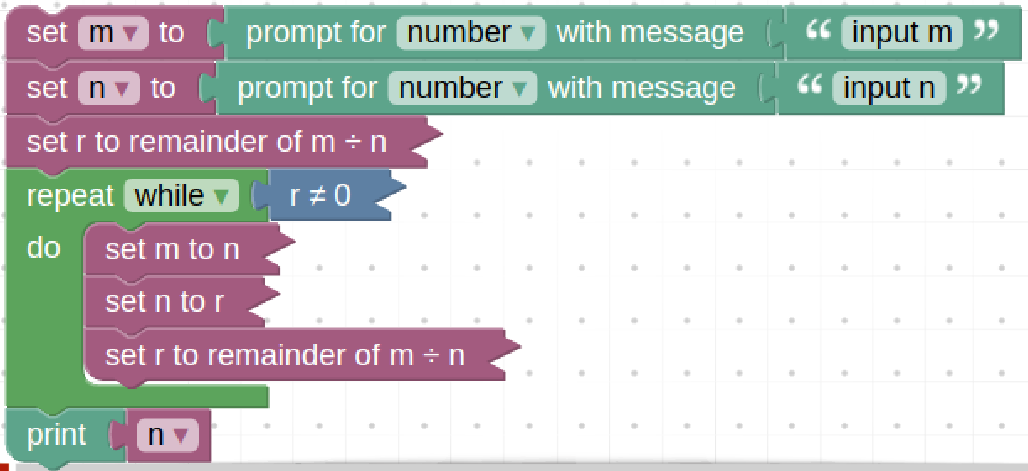
Programming Languages

Flowchart



Programming languages

Blockly



```
set m to prompt for number with message "input m"
set n to prompt for number with message "input n"
set r to remainder of m ÷ n
repeat while r ≠ 0
do  set m to n
    set n to r
    set r to remainder of m ÷ n
print n
```

Programming languages

Java Script

```
var m, n, r;
m = parseFloat(window.prompt('input m'));
n = parseFloat(window.prompt('input n'));
r = m % n;
while (r != 0) {
  m = n;
  n = r;
  r = m % n;
}
window.alert(n);
```

Programming languages

Python

```
m = None
n = None
r = None
m = float(text_prompt('input␣m'))
n = float(text_prompt('input␣n'))
r = m % n
while r != 0:
    m = n
    n = r
    r = m % n
print(n)
```

Programming languages

PHP

```
$m;
$n;
$r;
$m = floatval(readline('input␣m'));
$n = floatval(readline('input␣n'));
$r = $m % $n;
while ($r != 0) {
  $m = $n;
  $n = $r;
  $r = $m % $n;
}
print($n);
```

Programming languages

Lua

```
m = tonumber(text_prompt('input m'), 10)
n = tonumber(text_prompt('input n'), 10)
r = m % n
while r ~= 0 do
  m = n
  n = r
  r = m % n
end
print(n)
```

Programming languages

Dart

```
var m, n, r;
main() {
  m = Math.parseDouble(Html.window.prompt('input m', ''));
  n = Math.parseDouble(Html.window.prompt('input n', ''));
  r = m % n;
  while (r != 0) {
    m = n;
    n = r;
    r = m % n;
  }
  print(n);
}
```

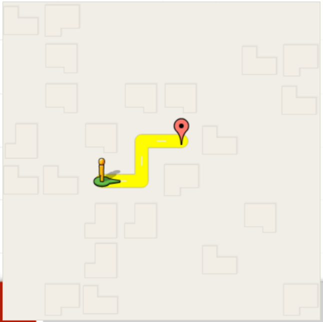
Control structure

- An algorithm can be thought of as being executed by a little robot, or a processor). The processor receives orders to run around doing this and that, where the "this and that" are the basic actions of the algorithm.
- It should be quite obvious that the order in which the basic actions are carried out is crucial.
- Creating an algorithm we will use the sequence of commands. We assume that after one command will be executed the next.
- Sometimes we will like to change the execution order of commands using *conditionals* **if** *something (happens)* **do** *this* **else do** *that*

Direct sequencing

Direct sequencing

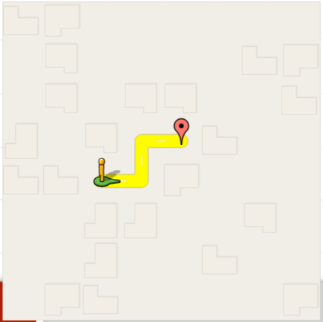
of the form “do A followed by B,” or “do A and then B.” (Every semicolon or period in the recipe hides an implicit “and then” phrase, for example, “gently fold in chocolate; [and then] reheat slightly . . .”)

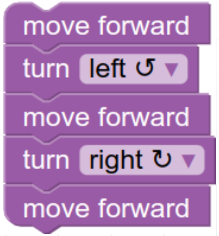


Direct sequencing

Direct sequencing

of the form “do A followed by B,” or “do A and then B.” (Every semicolon or period in the recipe hides an implicit “and then” phrase, for example, “gently fold in chocolate; [and then] reheat slightly . . .”)



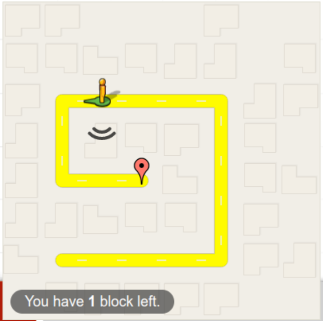


```
move forward
turn left ↺
move forward
turn right ↻
move forward
```

Conditional branching

Conditional branching

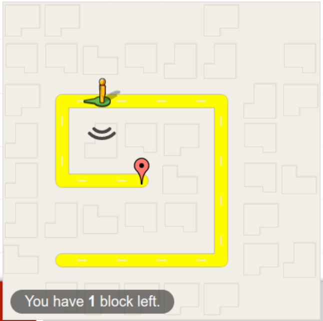
of the form “if Q then do A otherwise do B,” or just “if Q then do A,” where Q is some condition. (For example, in the recipe “reheat slightly to melt chocolate, if necessary,” or “serve with whipped cream, if desired.”)

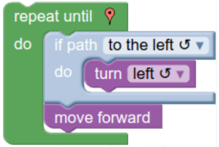


Conditional branching

Conditional branching

of the form "if Q then do A otherwise do B," or just "if Q then do A," where Q is some condition. (For example, in the recipe "reheat slightly to melt chocolate, if necessary," or "serve with whipped cream, if desired.")





```
repeat until 📍
do  if path to the left ↺
    do  turn left ↺
    move forward
```

Iterations I

Sequencing and branching, do not explain how an algorithm of fixed—maybe even short—length can describe processes that can grow increasingly long, depending on the particular input.

Bounded iteration

of the general form "do A exactly N times," where N is a number.

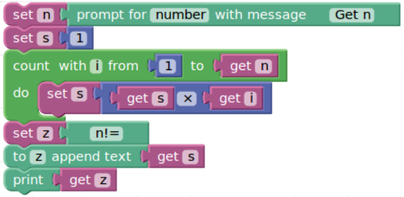
Conditional iteration

sometimes called unbounded iteration, of the form "repeat A until Q," or "while Q do A," where Q is a condition. (For example, in the recipe "beat egg whites until foamy.")

Iterations can be nested!

Bounded iteration

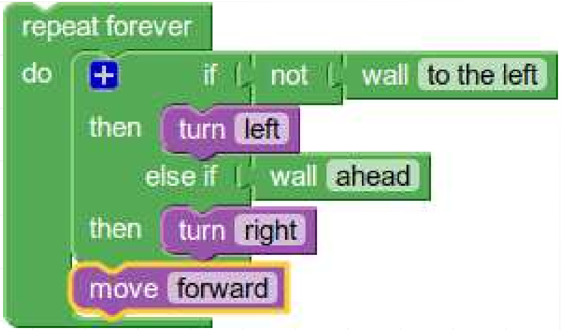
Factorial



```
set n ( prompt for number with message "Get n"
set s ( 1
count with i from ( 1 to ( get n
do  set s ( ( get s × ( get i
set z ( "n!="
to z append text ( get s
print ( get z
```

Iteration cont.

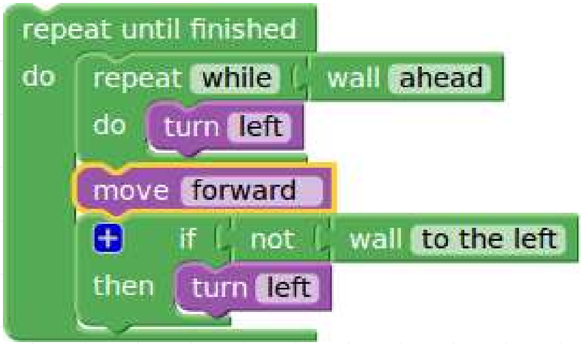
Nested iterations



```
repeat forever
do  if not wall to the left
    then turn left
    else if wall ahead
    then turn right
    move forward
```

Iteration cont.

Nested iterations



```
repeat until finished
do  repeat while wall ahead
    do  turn left
    move forward
    if not wall to the left
    then turn left
```

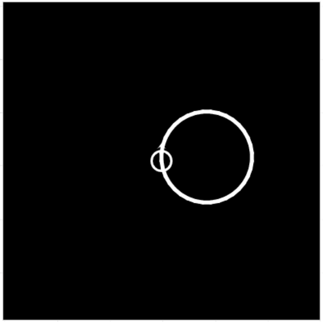
GoTo Instruction

GoTo

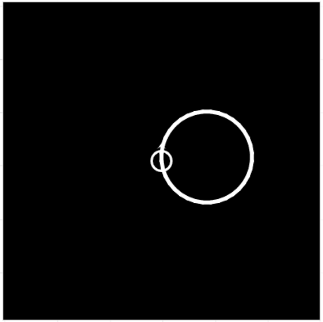
Instruction has the general form “goto G,” where G marks some point in the text of the algorithm.

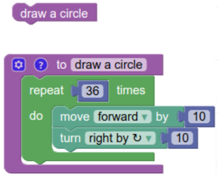
Many researchers are opposed to using “goto”s freely in algorithms.

Subroutines



Subroutines





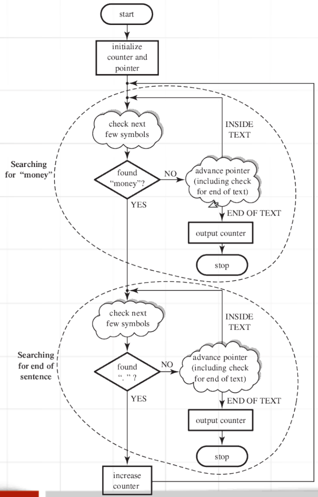
```
draw a circle

to draw a circle
    repeat 36 times
    do  move forward by 10
        turn right by ↻ 10
```

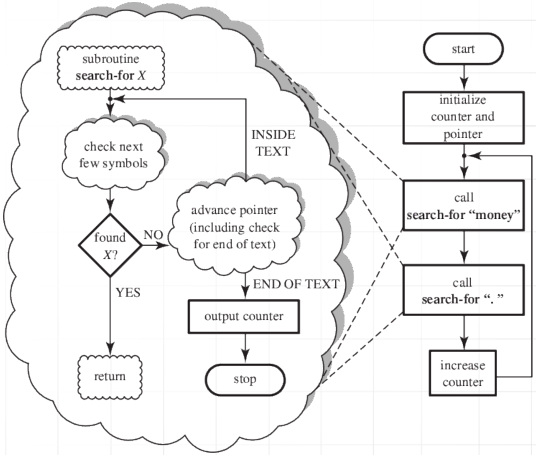
Subroutines I

1. We are given a lengthy text and we are interested in finding out how avaricious its author is by counting the number of sentences that contain the word “money.”
2. We are not interested in the number of times the word “money” occurs, but in the number of sentences in which it occurs.
3. An algorithm can be designed to run through the text looking for “money.”
4. Upon finding such an occurrence, it proceeds to run ahead looking for the end of a sentence, which for our purposes is assumed to be a period followed by a space; that is, the “. ”
5. When the end of a sentence is found, the algorithm adds 1 to a counter which was initialized to 0 at the start.
6. Algorithm then resumes its search for “money” from the beginning of the next sentence.

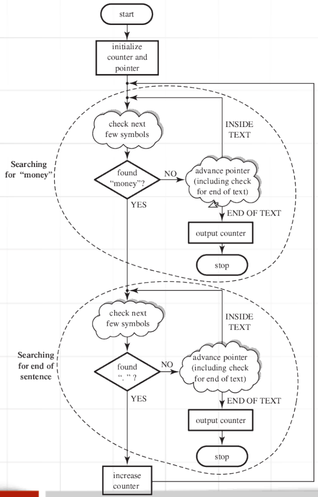
Subroutines

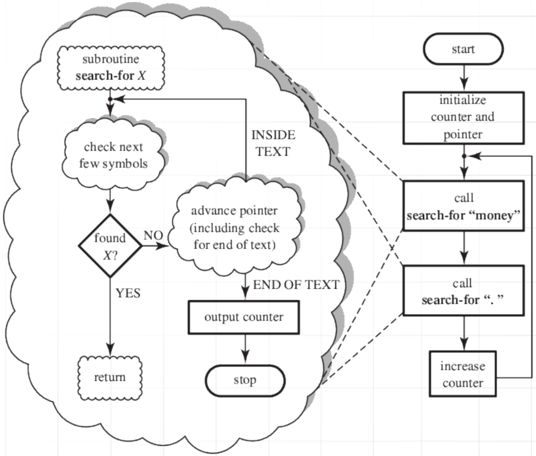


Subroutines



Subroutines





Subroutine

Subroutine

(also termed procedure, function, routine, method, or subprogram), is a part of source code (algorithm) within a larger computer program that performs a specific task and is relatively independent of the remaining code (algorithm).

Variables I

- A variable is not a number, a word, or some other item of data. Rather, it can be treated as a small box, or cell, or place, in which a single item can be kept.
- Algorithms utilize variables (with different names) for different purposes, bat generally for storing some values. They are kind of a **storage** or a **memory**.
- New value putted into the variable erases its previous content.
- Taking a value from the variable does not change its content.
- Each variable can contain only one value at a time.
- Notation similar to mathematical is used to describe operations on variables: $X = 5$, $X := X + 1$, or $X \leftarrow A + B$

Vector, array I

- Let us think about our employee list. It may be viewed simply as a multitude of data elements, which we might decide to keep, or store, in a multitude of variables, say $X, Y, Z, \ldots$
- Storing all salaries that way is not very suitable for the mentioned algorithm: each element in the list would have to be referred to in the algorithm by a unique name.
- We need **lists** of variables that can be "run through," or accessed in some other way, but without the need to name each of their elements explicitly.
- In mathematics, such elements are called "vectors", or **one-dimensional arrays**.
- The name is assigned to all structures (containing all data).

Vector, array II

- We can access each structure element, for example, by its position on the list, we note this: $V(I)$ means the content of the I-th element of the vector V. The alternative notation looks like this: $V[J]$. I, J, are so-called pointers or indexes

Here each cells represents one elementary variable.

$V(1)$	$V(2)$	$V(3)$	$V(4)$	$V(5)$

In mathematical notation:

V_1	V_2	V_3	V_4	V_5

Simple algorithm using vectors

```
1. do the following $N - 1$ times:
    1.1 $X \leftarrow 1$;
    1.2 while $X < N$ do the following:
        1.2.1 if $V[X + 1] < V[X]$ then exchange them;
        1.2.2 $X \leftarrow X + 1$.
```

Simple algorithm using vectors

```
1. do the following $N - 1$ times:
    1.1 $X \leftarrow 1$;
    1.2 while $X < N$ do the following:
        1.2.1 if $V[X+1] < V[X]$ then exchange them;
        1.2.2 $X \leftarrow X + 1$.
```

There are also special “indexed” versions of iterative control constructs, tailored towards vector traversal. For example, we can write:
for X going from 1 to 100 do the following
which is similar to:
do the following 100 times

Array (multidimensional) I

- In many cases, it is convenient to arrange the data, not in a simple, one-dimensional list, but in a **table**.
- The corresponding algorithmic data structure is called a **matrix**, or a **two-dimensional array**, or simply an **array** for short.
- The standard second-grade multiplication table is a 10 by 10 array in which the data item at each point is the product of the row and column indices;
- Referring to an array element is typically achieved using two indices, row and column. We write $A[5,3]$ for the element located in row 5 and column 3. Sometimes such referring will be noted: $A(5,3)$ or $A[5][3]$.

Array (multidimensional) II

- Running through the entire array can be achieved by an outer loop running through all rows and an inner one running through all of a particular row elements, or vice versa.

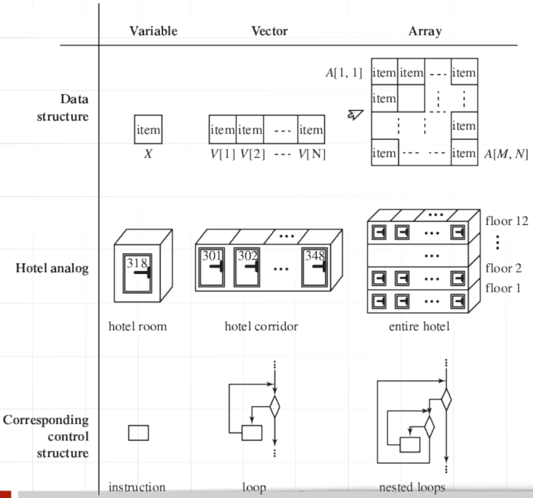
Mathematical notation:

$$\begin{matrix} v_{1,1} & v_{1,2} & v_{1,3} \\ v_{2,1} & v_{2,2} & v_{2,3} \\ v_{3,1} & v_{3,2} & v_{3,3} \\ v_{4,1} & v_{4,2} & v_{4,3} \end{matrix}$$

Algorithmic notation:

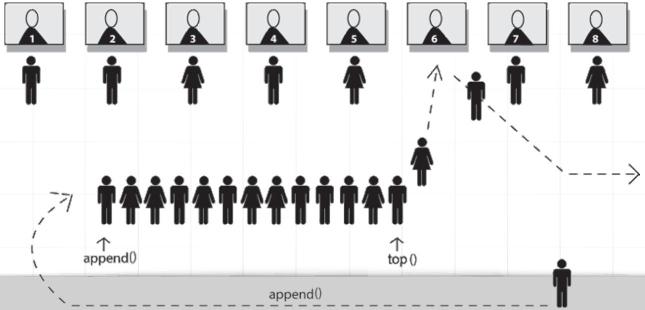
$$\begin{matrix} V(1,1) & V(1,2) & V(1,3) \\ V(2,1) & V(2,2) & V(2,3) \\ V(3,1) & V(3,2) & V(3,3) \\ V(4,1) & V(4,2) & V(4,3) \end{matrix}$$

Control structures for arrays



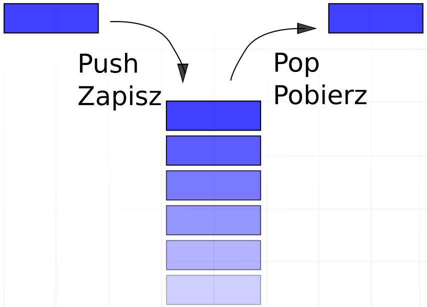
Queue

- An interesting variation on the vector/array.
- Sometimes a list is used just to model a queue, in which case all the algorithm needs in way of interaction with the list are the ability to add elements to its "back" and remove them from its "front."
- The structure is called FIFO *(First In First Out)*



Stack

- A structure similar to a vector (list)
- Adding elements to “front”
- Remove also from “front”.

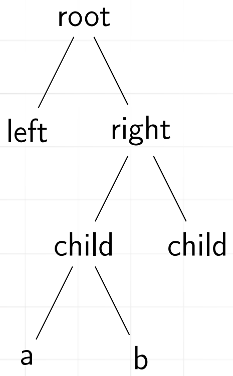


Tree or hierarchy

One of the most important and prominent data structures in existence is the **tree**.

- Structure with the order.
- There is a special object that is the "beginning" of the whole structure: **root**.
- Other elements are called **descendants** or **offspring**.
- Each object can have a number of **equivalent** descendants.
- Each offspring is called a **node**.
- Nodes at the "end" of the tree, having no offspring are **leaves**.
- **Branch** sequences of nodes corresponding to downward traversals in the direction from the root to a leaf.

Tree



Other data structures

- **lists** (some similarity to vectors or arrays).
- **databases** (some similarity to arrays).
- **graphs** (some resemblance to trees).

Part I

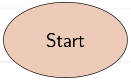
Excursus

Selecting the highest person

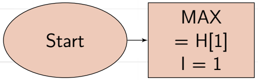
(in the room)

1. Height of all people are saved in an array called **H**. (This should be height, but it takes up too much space.)
2. Variable **N** contains the number of persons in the room (the length of array **H**).
3. Variable **MAX** contains the maximum value from array **H** (when the algorithm ends) i.e. height of the highest person in the room.
4. **I** — auxiliary variable.

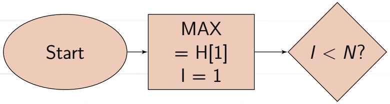
Maximum value in an array



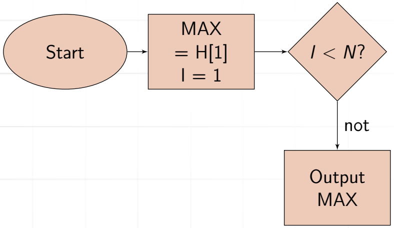
Maximum value in an array



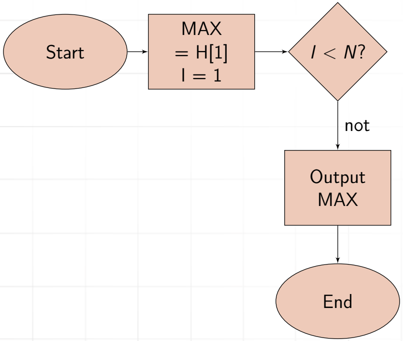
Maximum value in an array



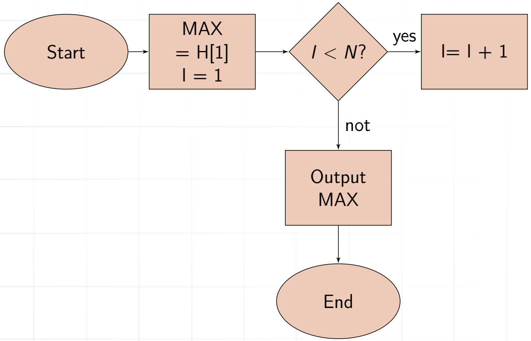
Maximum value in an array



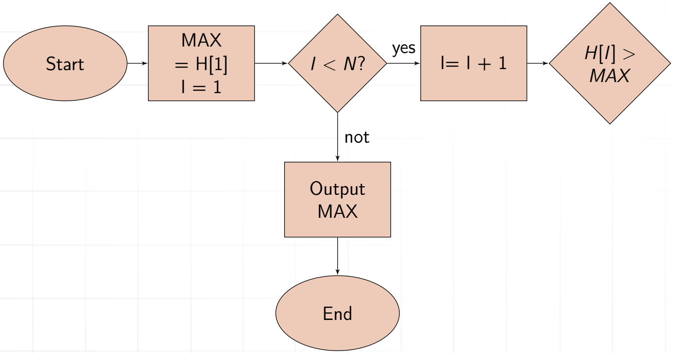
Maximum value in an array



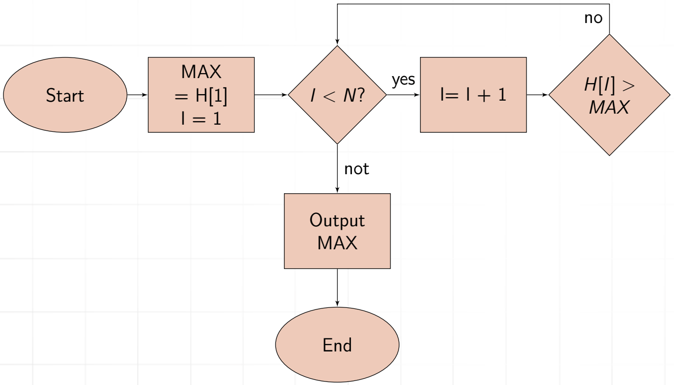
Maximum value in an array



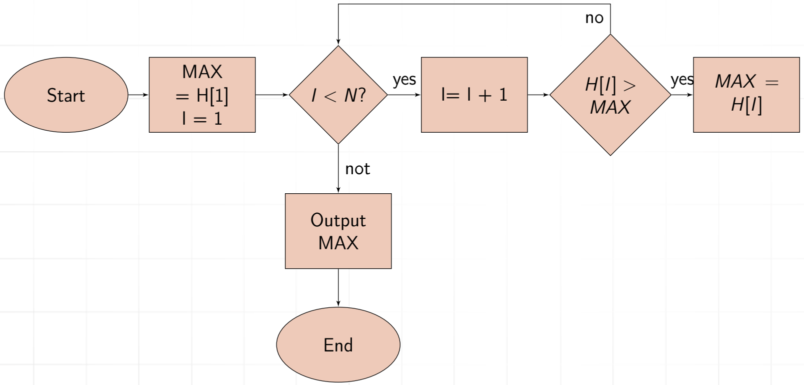
Maximum value in an array



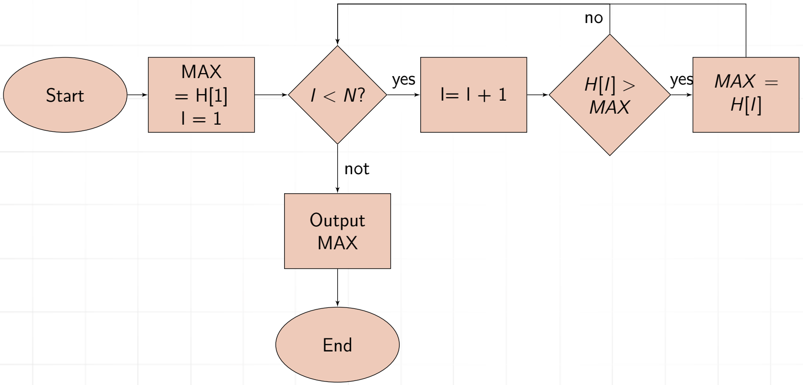
Maximum value in an array



Maximum value in an array



Maximum value in an array



Simple problem

1. While $X \neq 1$, do $X \leftarrow X - 2$.
2. Stop

Simple problem

1. While $X \neq 1$, do $X \leftarrow X - 2$.
2. Stop

When $X = 7$ we will got X equal to:

7, 5, 3, 1

…

Simple problem

1. While $X \neq 1$, do $X \leftarrow X - 2$.
2. Stop

When $X = 7$ we will got X equal to:
7, 5, 3, 1

. . .

When $X = 8$ algorithm generates:
8, 6, 4, 2, 0, -2, -4, -6, -8,. . .
and never stops

Simple problem

1. While $X \neq 1$, do $X \leftarrow X - 2$.
2. Stop

When $X = 7$ we will got X equal to:
7, 5, 3, 1

...

When $X = 8$ algorithm generates:
8, 6, 4, 2, 0, -2, -4, -6, -8,...
and never stops

This works for odd positive values and does not work for even values.

A little bit more complicated problem. . .

```
1. While $X \neq 1$, do:
    1.1 If $X$ is even put $X \leftarrow X/2$.
    1.2 otherwise put $X \leftarrow 3X + 1$
2. Stop
```

A little bit more complicated problem. . .

```
1. While $X \neq 1$, do:
    1.1 If $X$ is even put $X \leftarrow X/2$.
    1.2 otherwise put $X \leftarrow 3X + 1$
2. Stop
```

Let's start from 7: 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

A little bit more complicated problem. . .

```
1. While $X \neq 1$, do:
    1.1 If $X$ is even put $X \leftarrow X/2$.
    1.2 otherwise put $X \leftarrow 3X + 1$
2. Stop
```

Let's start from 7: 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

And what with other values?

A little bit more complicated problem. . . I

Collatz conjecture

```
1. While $X \neq 1$, do:
    1.1 If $X$ is even put $X \leftarrow X/2$.
    1.2 otherwise put $X \leftarrow 3X + 1$
2. Stop
```

A little bit more complicated problem. . . II

Collatz conjecture

It turns out that the above algorithm either completes relatively quickly or it generates a very long (an infinite?) chaotic sequence of numbers.

No one was able to prove that sequence generated by the algorithm begin to repeat (which means the algorithm will not stop never) or prove that for some particular initial value X the algorithm will stop.

A little bit more complicated problem. . . III

Collatz conjecture

The Collatz conjecture is: This process will eventually reach the number 1, regardless of which positive integer is chosen initially.