Anna Christiane Kolandjian

Part III-Algorithms: Flow charts and Pseudocode

Part A: Flowcharts

I-Algorithm definition

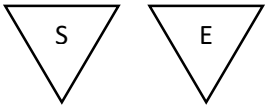
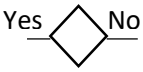
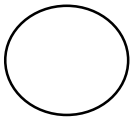
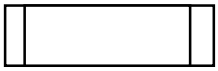
An algorithm is a procedure consisting of a finite set of instructions which specify a finite sequence of operations that provides the solution to a problem.
An algorithm is a step by step procedure to solve a given problem. In the problem solving phase of computer programming, we will be able to design algorithms.
There are two common tools used to document an logic program (an algorithm):
1. Flow charts (for small problems)
2. Pseudo code (fo large problems)

II-Flowcharts

Flowcharting is a tool developed in the computer industry for showing the steps involved in a process.
It's a type of diagram made up of boxes of various kinds, diamonds, rectangles, and other shapes connected by arrows.
Each shape represents a step in the process and arrows show the order in which they occur.

III-Flowcharting symbol

There are 7 basic symbols commonly used in flowcharting of assembly language programs.

Symbol	Name	Function
	Terminal	Used to represent the beginning and the end of a table
	Input	Used for Input operations, indicates that the computer is ready to obtain data
	Process	Used for arithmetic and data manipulation operations
	Output	Used for output operations
	Decision	Used for a logic and comparaison operations
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse arrow It is used to join different flow lines
	Predifined process	Used for subroutines. It represents a group of statements that performs one processing task
	Flow lines	Shows the direction of flow used to connect symbol, and indicates the flow of central, flow of logic

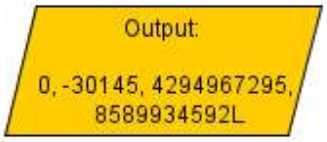
Note:

Decision is a switching logic that consists of two components:

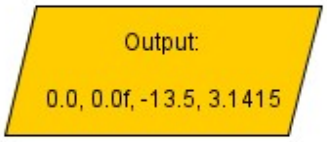
1. A condition
2. A “go to” command depending on the results of condition test

Symbol	Meaning
==	Equal
!=	Not equal
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

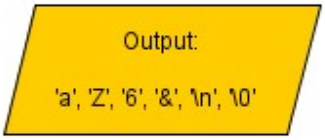
Some of the most used data types are:



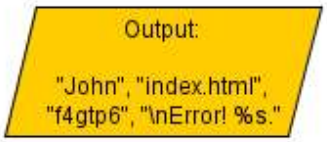
- **Integer** – contains integer numbers. There are several integer types with different size and range. By default they are signed. We use them to store information like quantities, sizes, period of time etc.



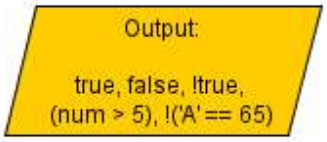
- **Real** – Also called "floating point numbers", contain real numbers with some precision. It depends on the type we use. By default is signed. You can use these to save price, salaries and others.



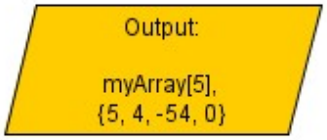
- **Character** – Saves a single symbol. It is represented with a code. Languages like C, C++ use one byte code, and others like Java, C# use two bytes code. Character values are enclosed by apostrophes – 'character'. A character type can be used when reading the input from the keyboard symbol by symbol. There are special characters called escape sequences. They begin with a backslash '\'. Such symbols are new line('\n'), null('\0') and others.



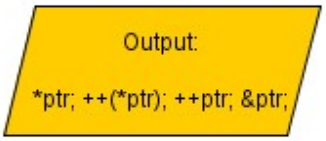
- **String** – represents a sequence of characters. It does not have a fixed range. Usually it takes as many bytes as needed to save the information. String values are enclosed by quotation marks – "string". Use this to store names, words or any other sequence of characters. It can also contain escape sequences (see character data type above).



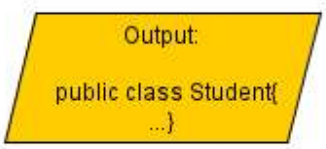
- **Boolean** – Stores a truth value. It can contain only “true” or “false”. It is in use when we want to save the result of a logical(Boolean) calculation.



- **Array Data Type.** An array is not really a new type. It is a sequence of many values of the same type. They are called elements. Each element has an unique index number. The first element has index=0, the second 1...
To access an element from the array use the name of the array and the element index(the third element of array “myArray” is : myArray[2]). Arrays are useful when dealing with many records of similar information. For instance it is convenient if you want to save the names for a given number of people.

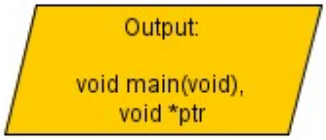


- **Pointers** – This also is not really a type, but instead a data pointing where a certain information is stored. Pointers are extremely powerful, but it can be very hard to master them.



If you plan learning C, you will work with pointers very often.

- **User defined** – When solving a complex task it is often useful to define our own data type. For instance : you want to save information about students. It will be much more convenient if there is a "student" type and all information for one student is kept in one place and accessed with one variable. For this reason most of the computer languages offer this functionality. In C these are structures and unions. In object oriented languages this is done with classes.



- **Void** – This means "no data type". Usually this is used with sub-programs(methods) that don't return a result. Other usage are void pointers.

Math Operators

Sign operators

-,+ We put them before a number. The plus is not mandatory, because by default all numbers are positive. So we will use only the minus sign. Examples:

```
count = 5; (count = +5;)
temp = -10;
```

Arithmetic math operators

+ Addition (Sum). It takes exactly two operands – left and right. The result is their sum.

```
3 + 5 = 8;    6.4 + 3.14 = 9.24
```

++ Increment. It has only one argument (a variable) – left or right. The result is the variable's value, incremented by one.

number = 5;
number ++;
Now number is equal to 6.

iNumber = 5;
++iNumber;
Again iNumber is equal to 6.

When we use ++ with a left parameter(iNumber++), we call the operator “suffix” and when it has a right operand (++iNumber) – “prefix”. When used as a suffix, the increment is done, **after** the value of the variable is used in the statement. Used as a prefix, the increment is done **before** the calculations.

number = 5;
count = number++;
"number" has value 6, but "count" is equal to 5.

number = 5;
count = ++number;
Both variables are equal to 6.

– Subtraction. Also has a left and a right parameter. It subtracts the right operand from the left.
5 – 2 = 3; 2 – 6 = –4; 7.5 – 3 = 4.5;

-- Decrement. It is the same as ++, except that the variable is the **decremented** by 1. Can be used as prefix or suffix and the same rules apply.

*** Multiplication**. Again – one left and one right. The result is the product of the two numbers.
3 * 2 = 6; 2 * (–6) = –12; 3 * 2.5 = 7.5;

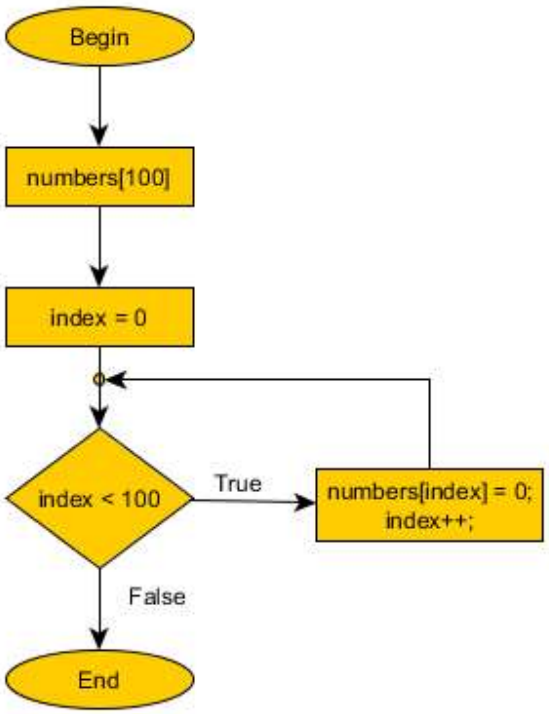
/ Division. The result is the left part, divided by the right.
Attention: if the operands are integers, the result will always be an integer!
4 / 2 = 2; **5/2 = 2!** 5/2.0 = 2.5;

% Division with a remainder. Accepts left and right argument. They must be integers.
Returns **only the remainder** from their division.
4 % 2 = 0; 3 % 2 = 1; 23 % 4 = 3;

De Morgan’s laws

When you work with logical operators you could come across the De Morgan's laws. This is a little bit more advanced stuff. You may not find this very useful right now, or it could be difficult to understand. So don’t worry if you don’t get it on 100% ;–).

The laws can be proved mathematically, but it is far out of our scope. Instead we want to focus on the logical operators and how they work. For exercise we will test the laws with example values.



For this example we will use letters for the names of our variables.

First law:

!((A && B) || (C && D)) = (!A || !B) && (!C || !D)

Let's give values to our variables and check if this is indeed correct. We accept: A= true, B = false, C = true, D = true.

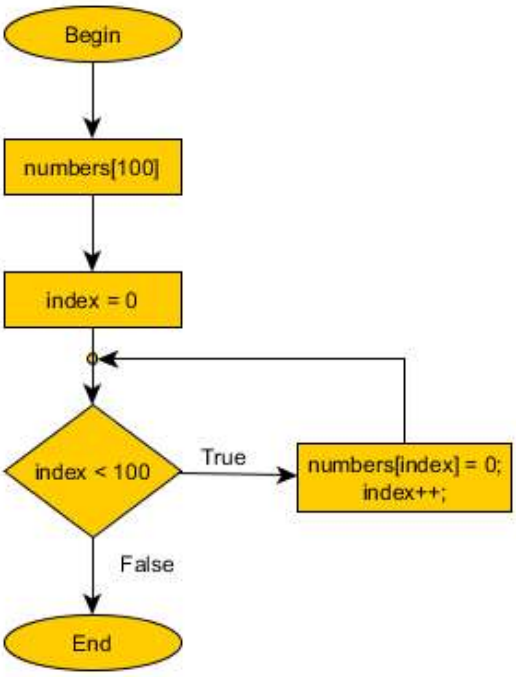
!((true && false) || (true && true))
=?= (!true || !false) && (!true || !true)
=>
!(false || true) =?= (false || true) && (false || false)
=>
!true =?= true && false
=>
false == false

OK, in this case it works. I suggest you substitute the variables with another set of values and do the calculations on paper. This is a very good training.

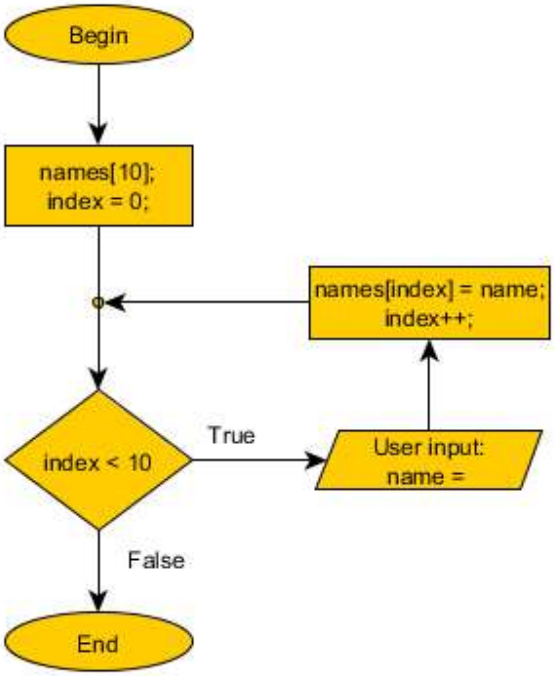
Second law:
!((A || B) && (C || D)) = (!A && !B) || (!C && !D)

There are many other laws in the world of logical operators, but they are out of our way.

Example: create an integer array with 100 elements and initialize all its elements with 0.

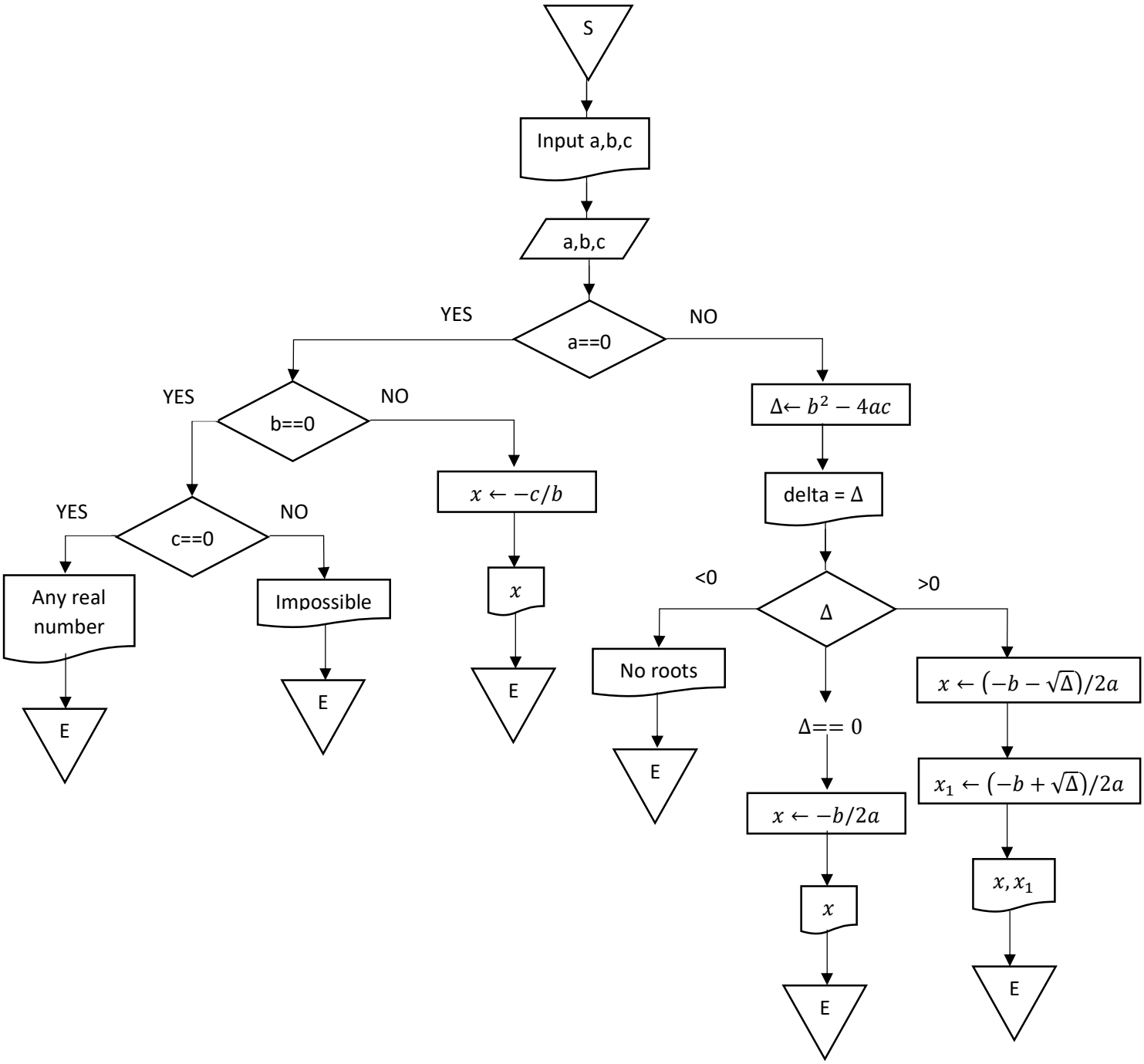


Probably you noticed – the last value of the index that will enter the body of the loop is 99. If we try to access element with bigger index the program will crash, because of "index out of boundaries exception".Example: Create an



algorithm which allows you to save 10 names.

Exercise 1: Design the flowchart relative to a real solution of $ax^2 + bx + c = 0\ where\ x \in \mathbb{R}$



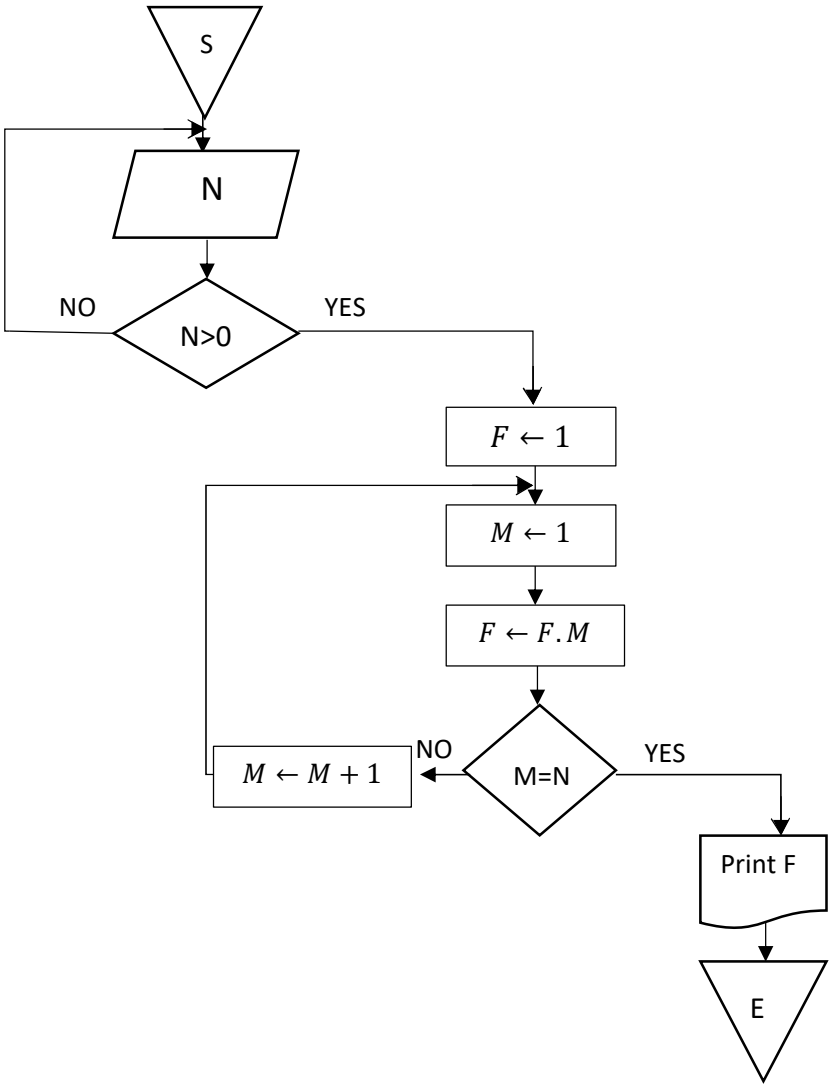
Exercise 2: Design the corresponding flowchart for computing factorial N where N!=1 x 2 x 3 x 4…(N-1) x N

1! = 1

2! = 1 x 2

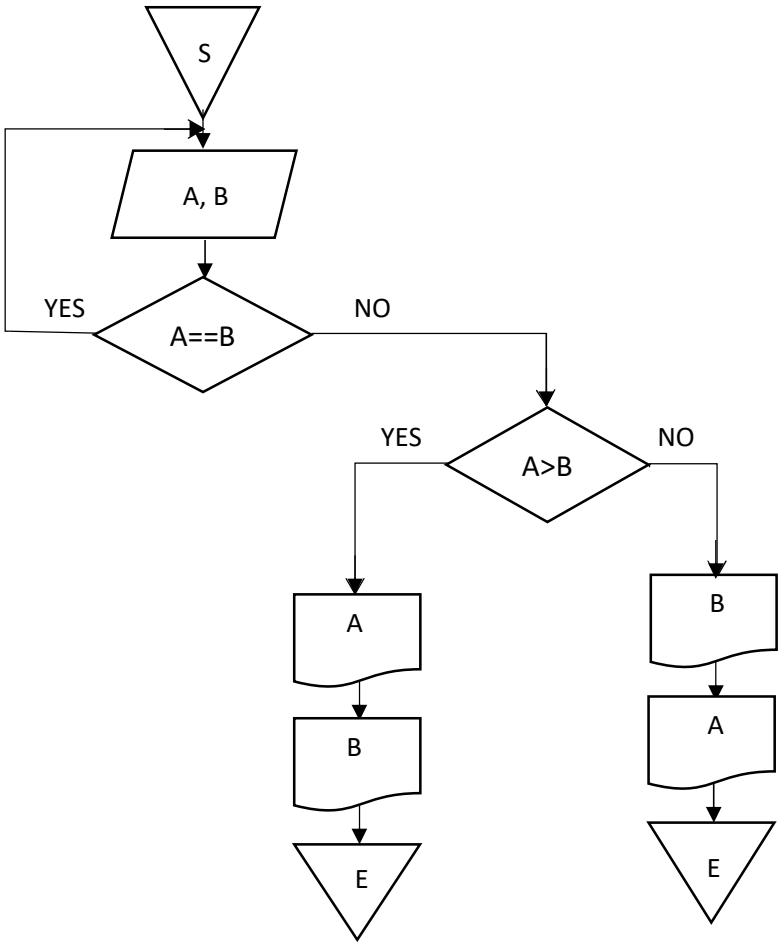
3! = 1 x 2 x 3

4! = 1 x 2 x 3 x 4

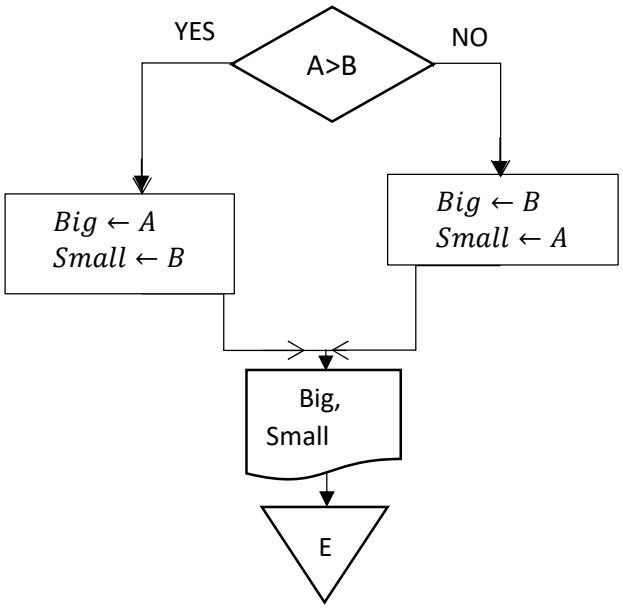


Exercise 3: Design the flowchart relative to a program that reads 2 numbers and display the numbers read in decreasing order.

1st method:

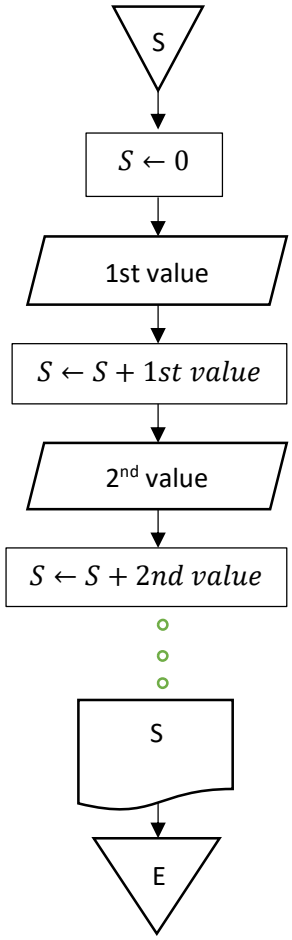


2nd method:

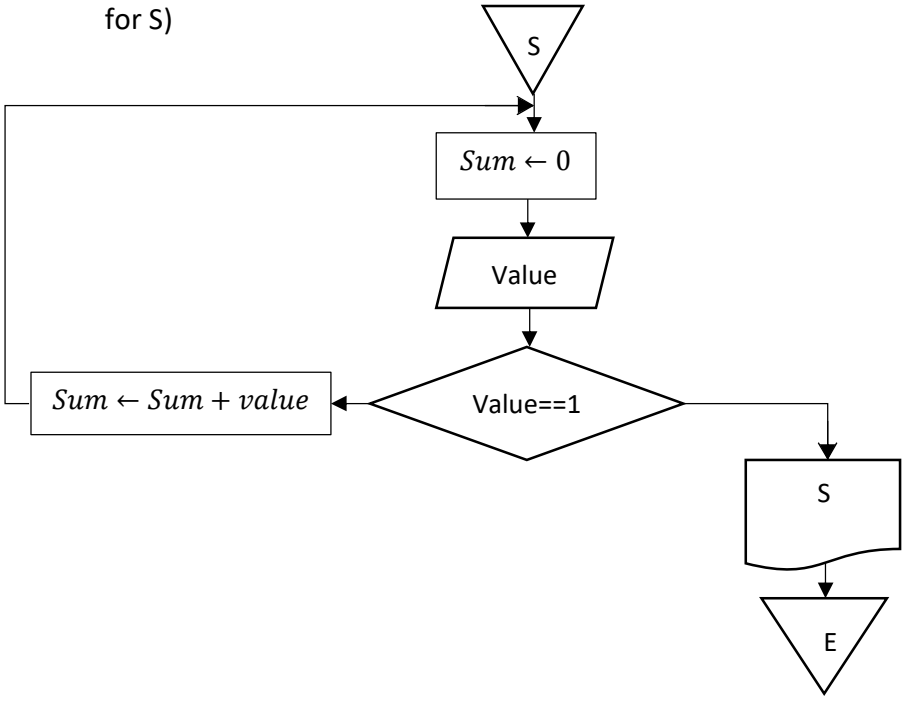


Exercise 4: Design the corresponding flowchart for adding 4 or N numbers input by the user.

4a.



4b. Infinite loop because no conditions imposed on the number of values entered (no display for S)

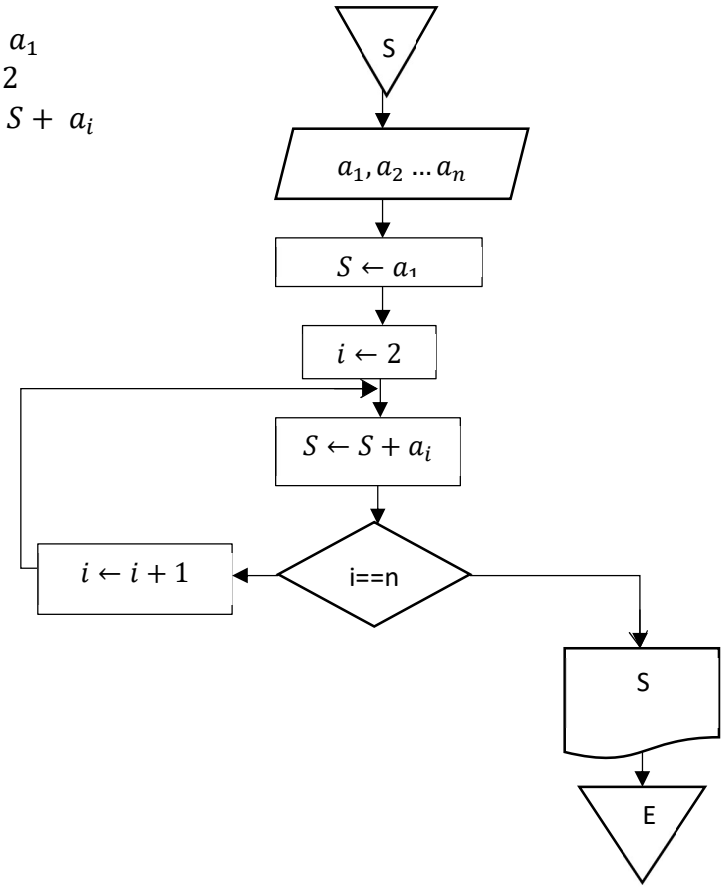


4c. One-dimensional array of n elements

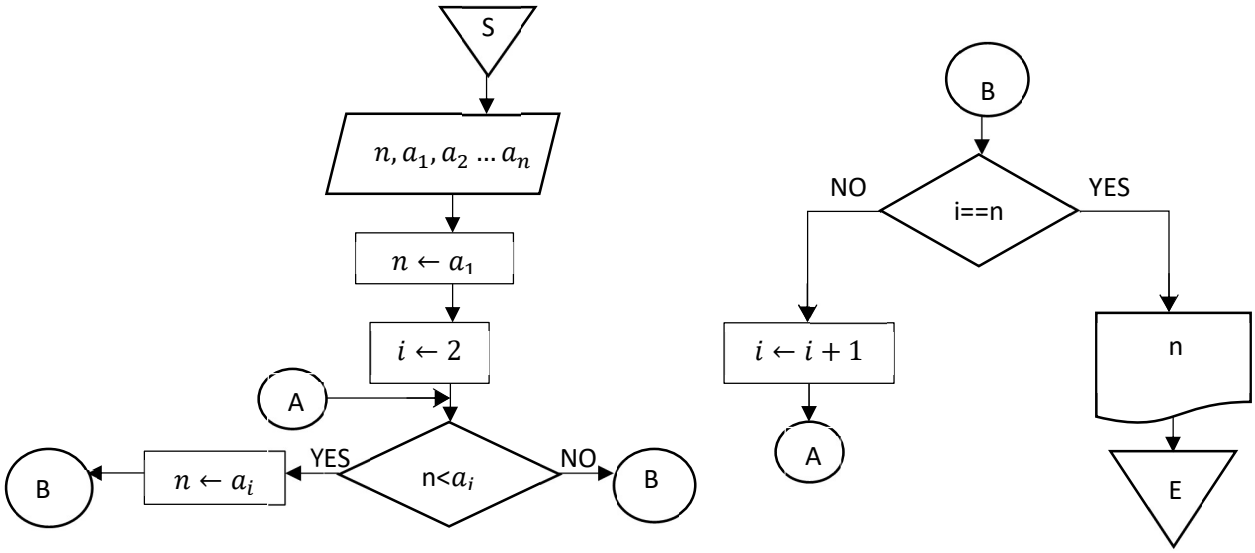
a_1	a_2	a_3				a_n

$S \leftarrow a_1$
$S \leftarrow S + a_2$
$S \leftarrow S + a_3$

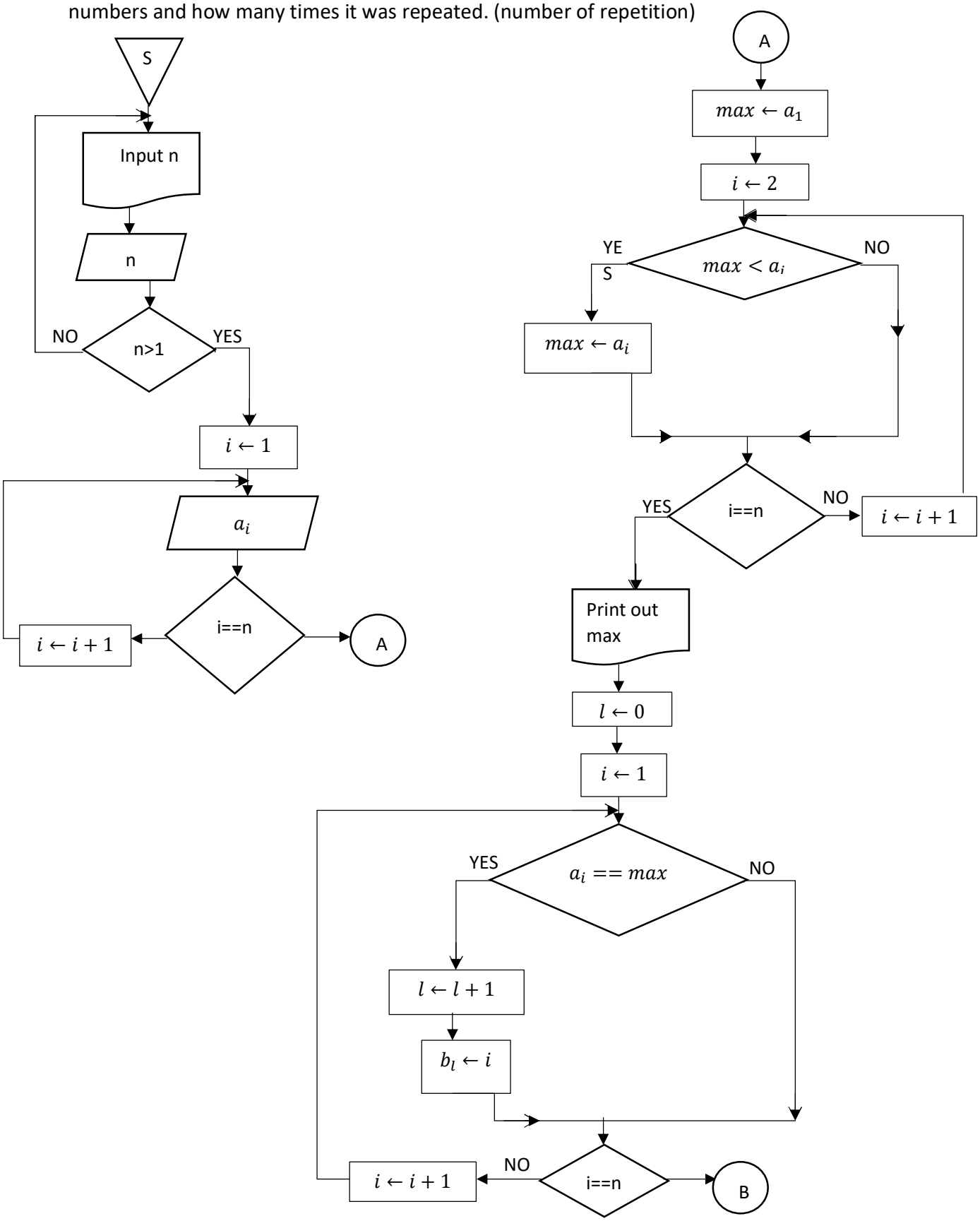
$S \leftarrow a_1$
$i \leftarrow 2$
$S \leftarrow S + a_i$
.
.
.
n

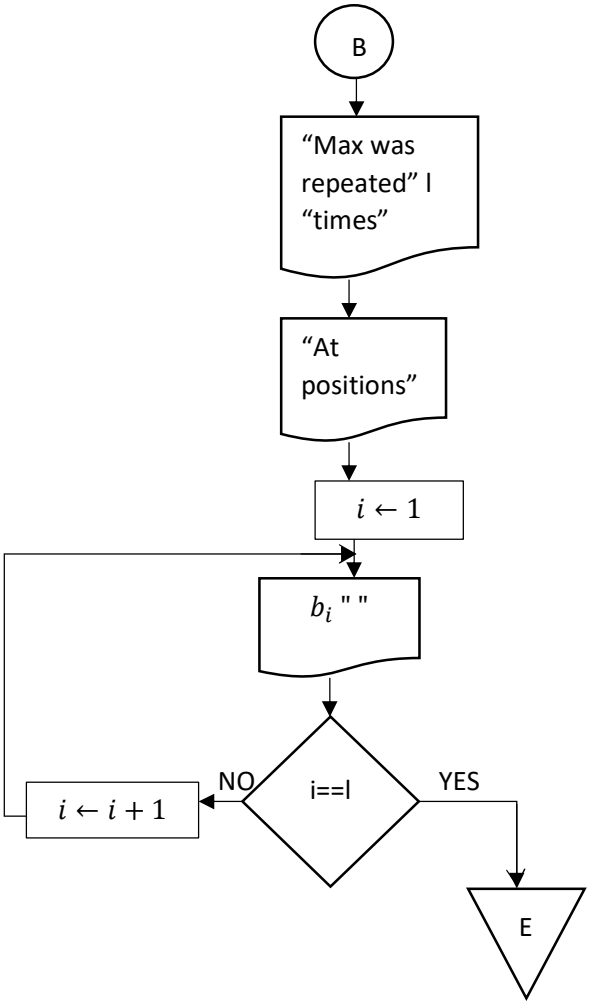


Exercise 5: Design the flowchart for computing the maximum of a sequence of "n" real numbers



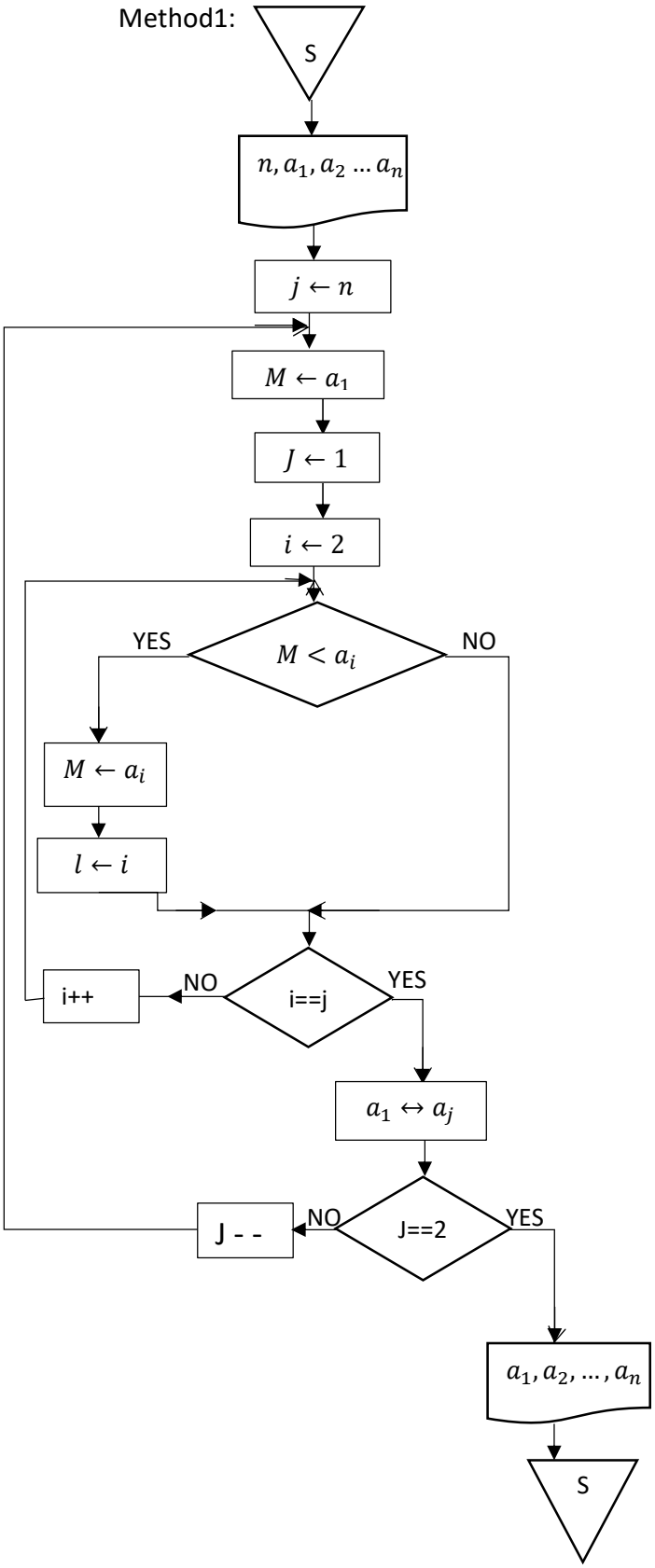
Exercise 6: Design the flowchart for computing the positions of the max of a sequence of n numbers and how many times it was repeated. (number of repetition)



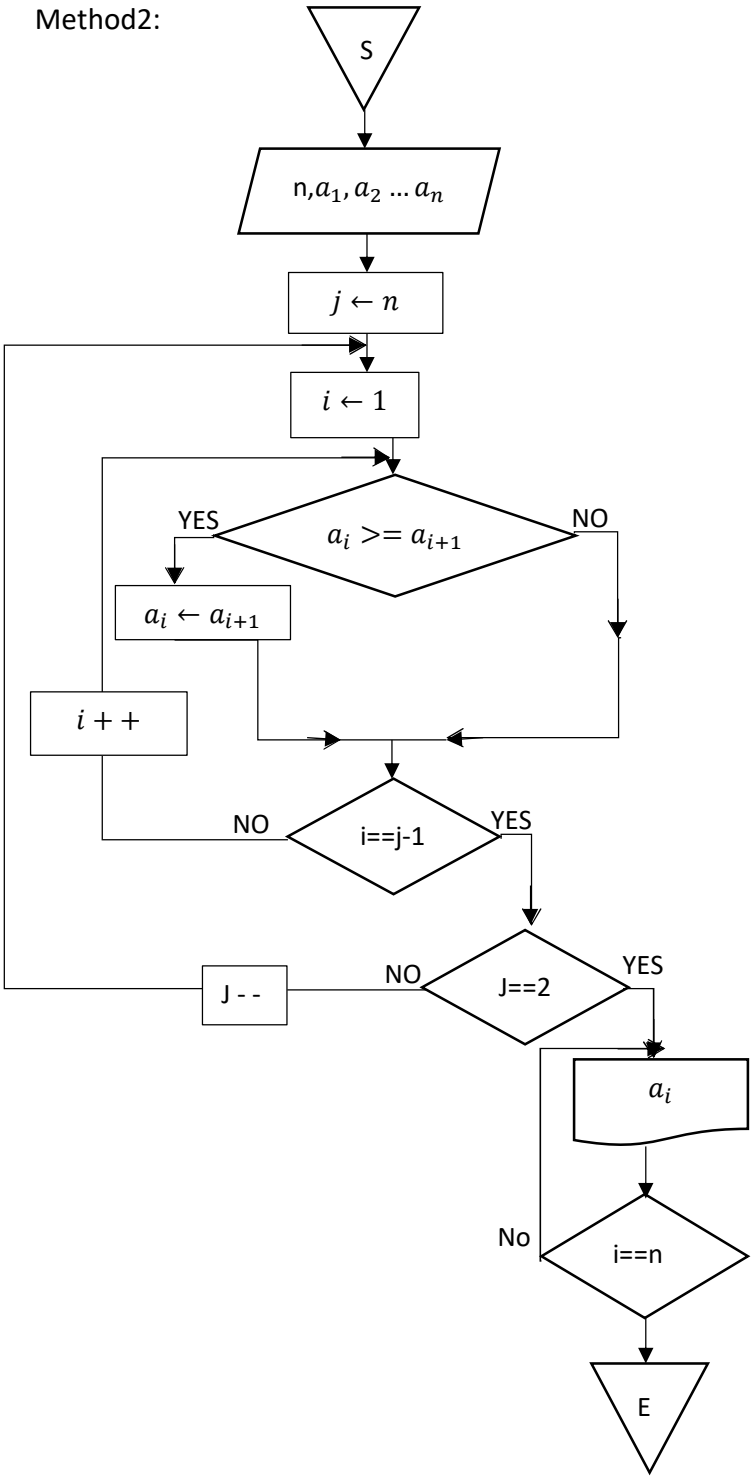

B
“Max was repeated” I “times”
“At positions”
$i \leftarrow 1$
b_i " "
i==I
NO
YES
$i \leftarrow i + 1$
E

Exercise 7: Let a sequence of numbers: $a_1, a_2 \dots a_n$.
Design a flowchart for displaying a sequence of n numbers in increasing order.

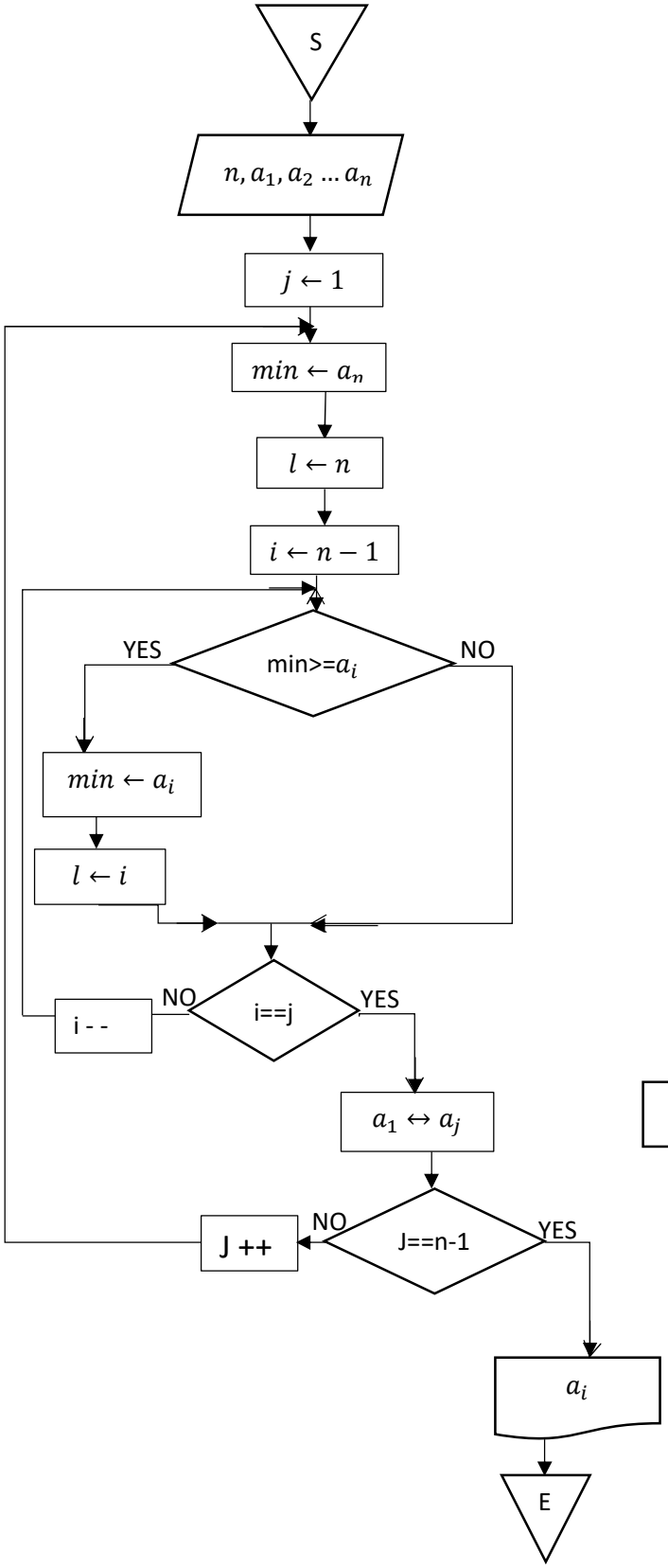
Method1:



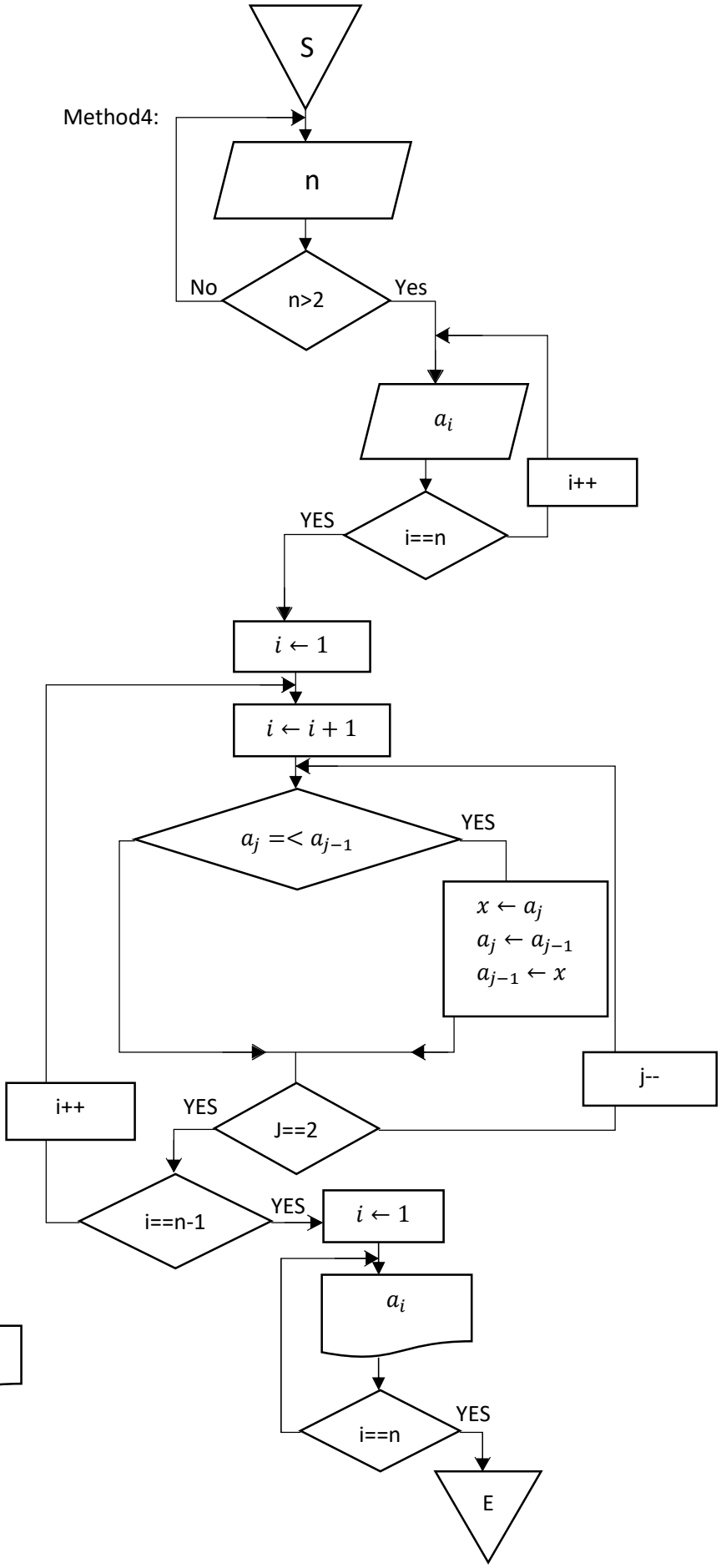
Method2:



Method3:



Method4:



Exercise 8: Design a flowchart relative to the program that reads one positive integer N(1234) and output (4321), where N>10

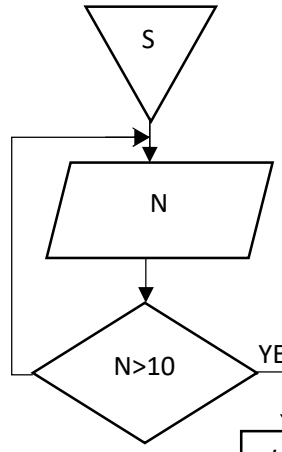
CONDITION: Do not use an array

Ex: <1234> and displays <4321>

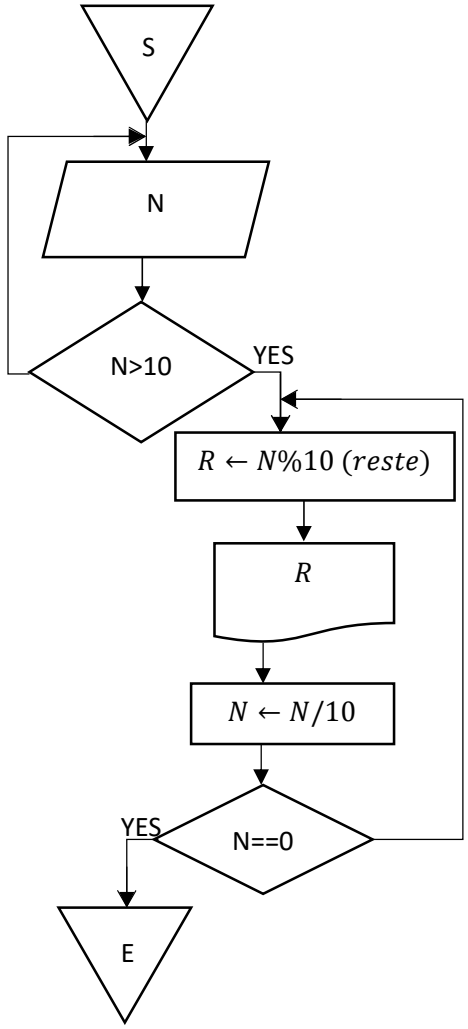
1234	10
Rest: 4	123
3	12
2	1
1	0

$R \leftarrow N\%10$

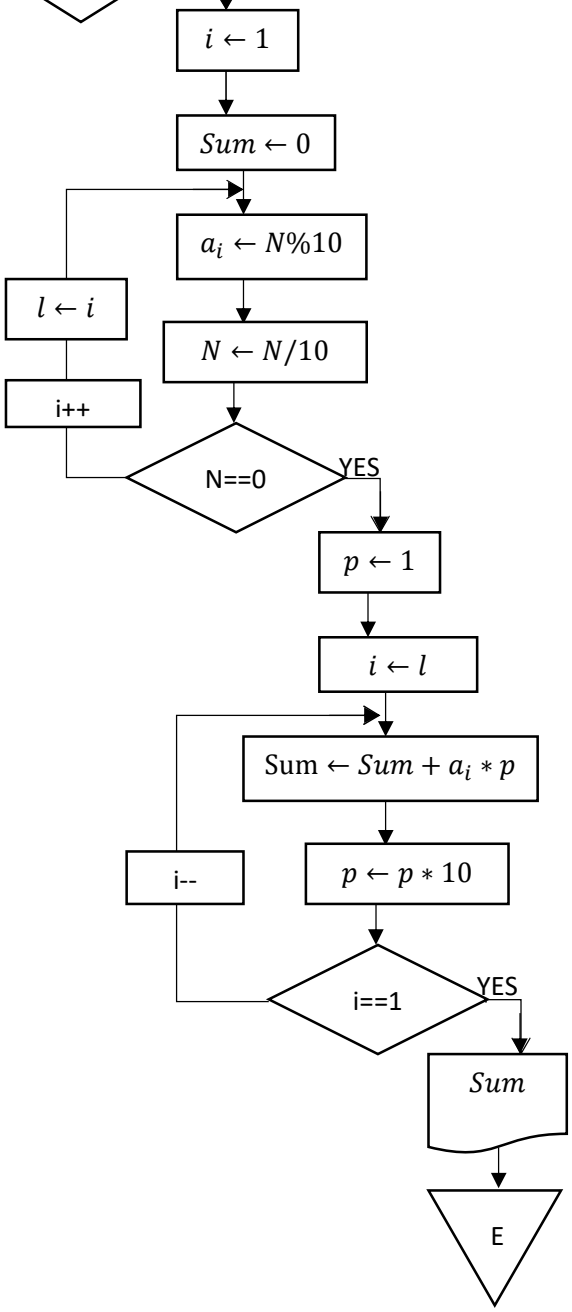
$N \leftarrow N/10$



method 1:

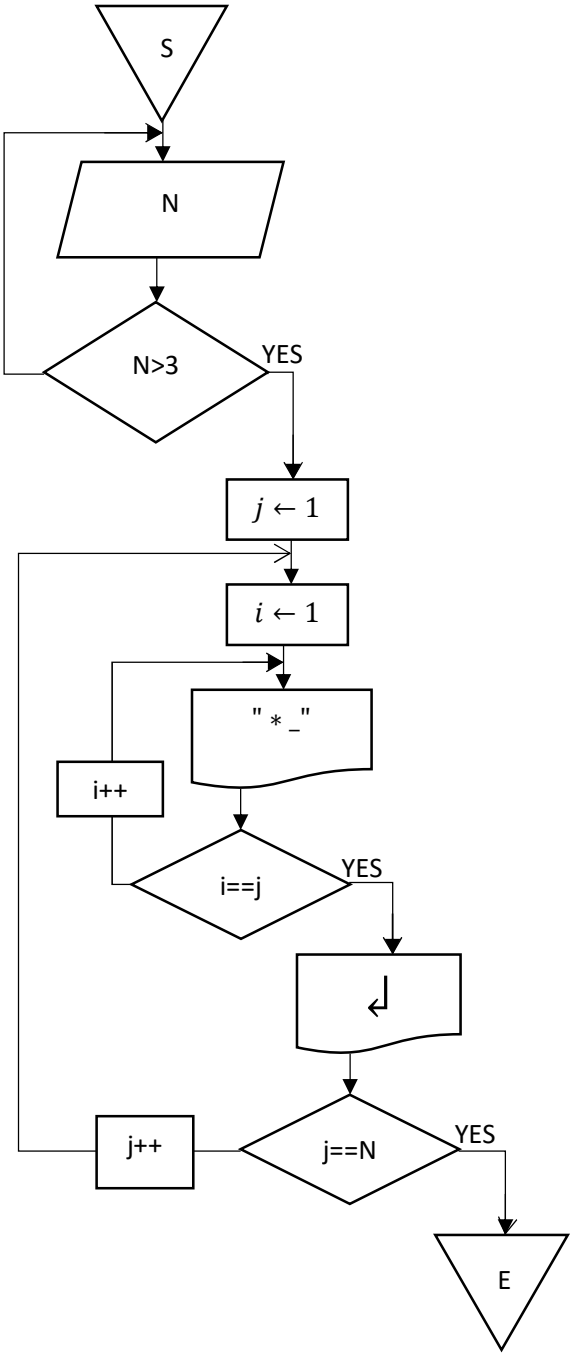


method 2:



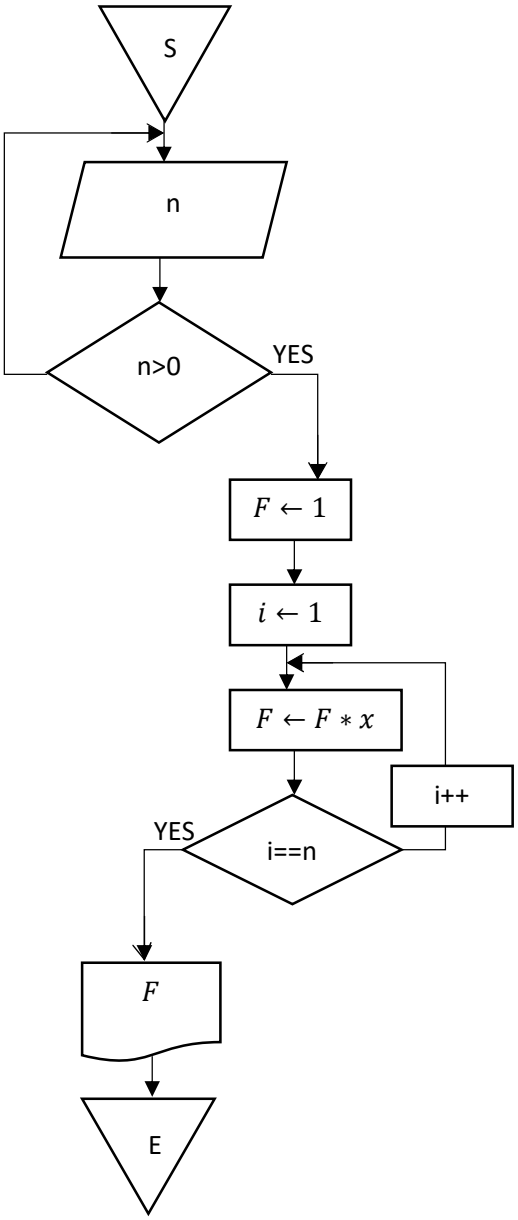
Exercise 9: Draw the flowchart that asks as input a positive integer N and displays as output the following shape (a triangle of stars over N rows where N>3)

line 1: *
line 2: * *
line 3: * * *
line 4: * * * *
line 5: * * * * *



Exercise 10: Design a flowchart that calculates x^n

$x * x * x * x * x \ldots = x^n$: Calculation of power



Exercise 11:

P(x)= $a_0x^n + \ a_1x^{n-1} + \cdots + \ a_{n-1}x + a_n$

a and n are input by the user

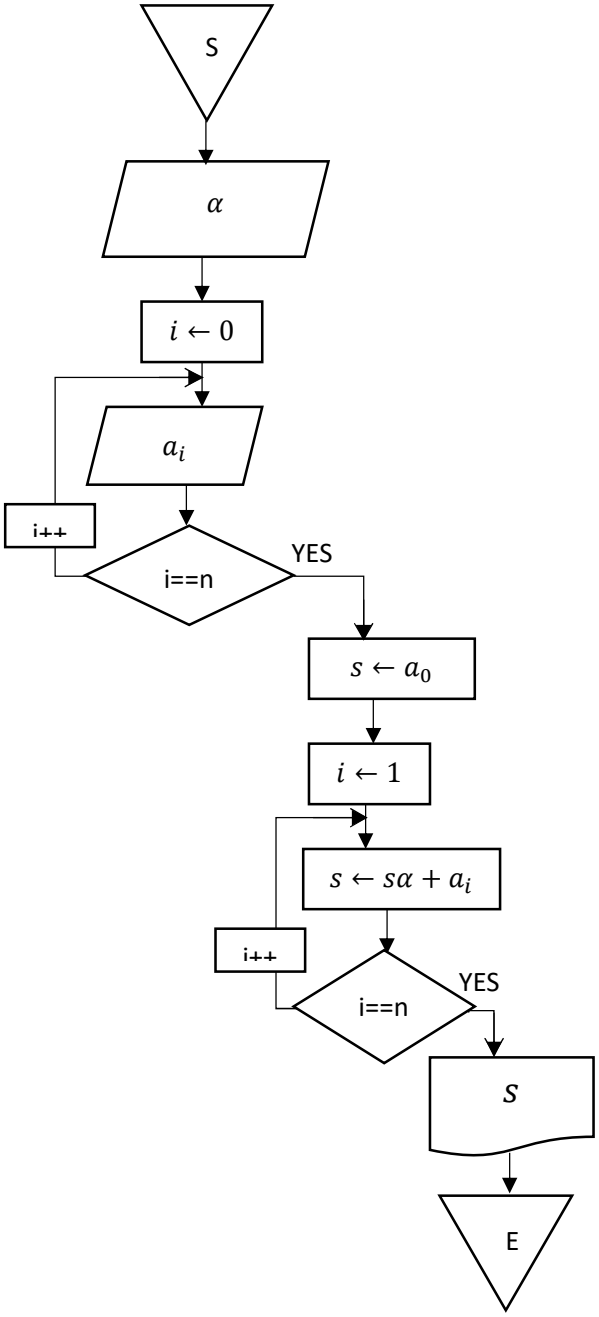
Design the flowchart of P(x)

Ex: P(x)= $4x^3 - 2x^2 - 3x + 5$

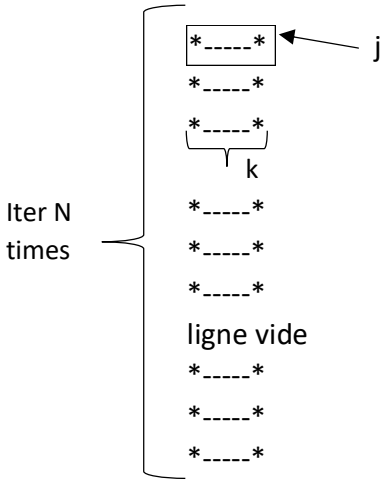
P(x)= $4x^3 - 2x^2 - 3x + 5$ = ((4x-2) x -3)x + 5

S1: 4x-2 ; S2: ((4x-2) x -3) ; S3: ((4x-2) x -3)x + 5

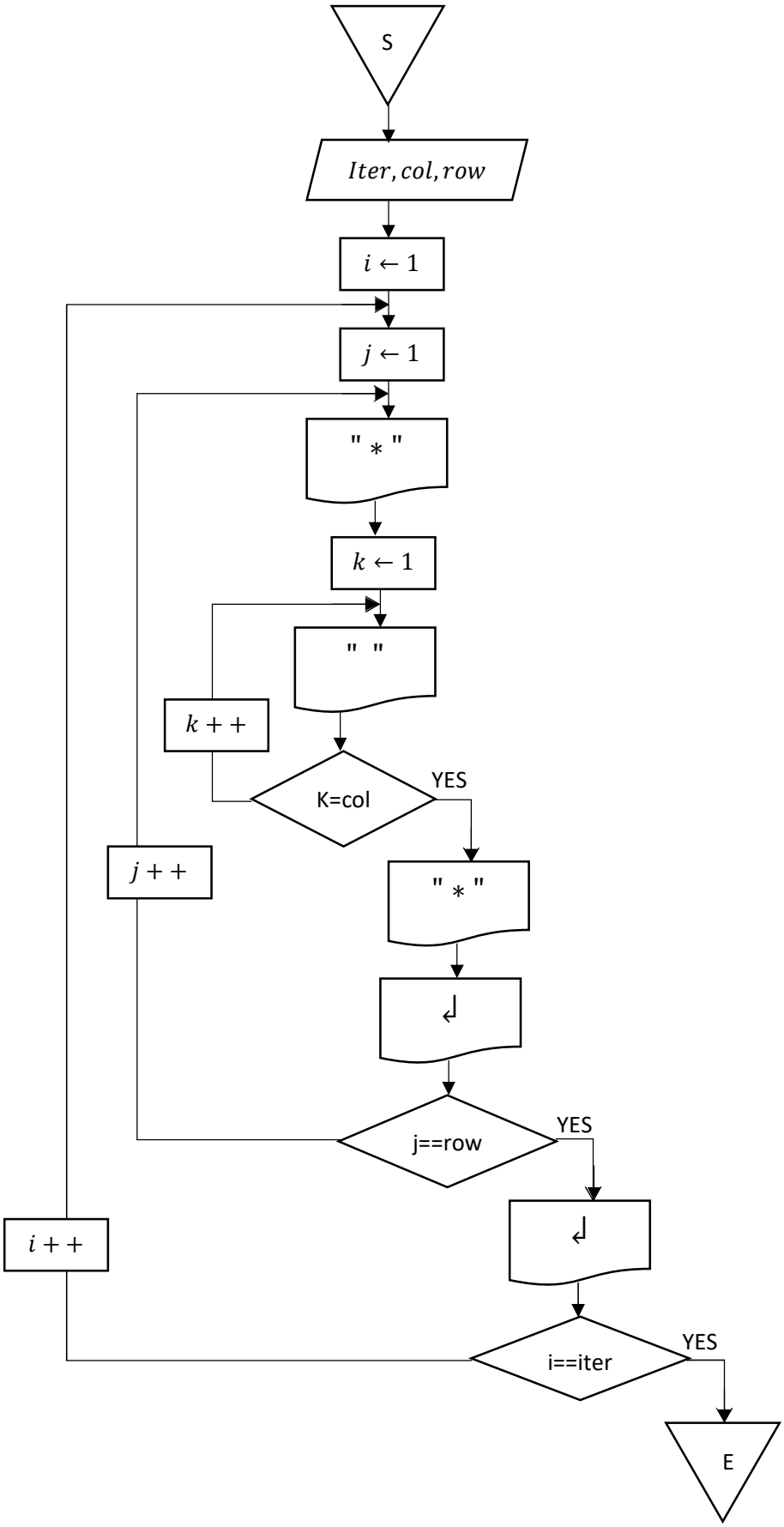
$$S \leftarrow S\alpha + a_i$$



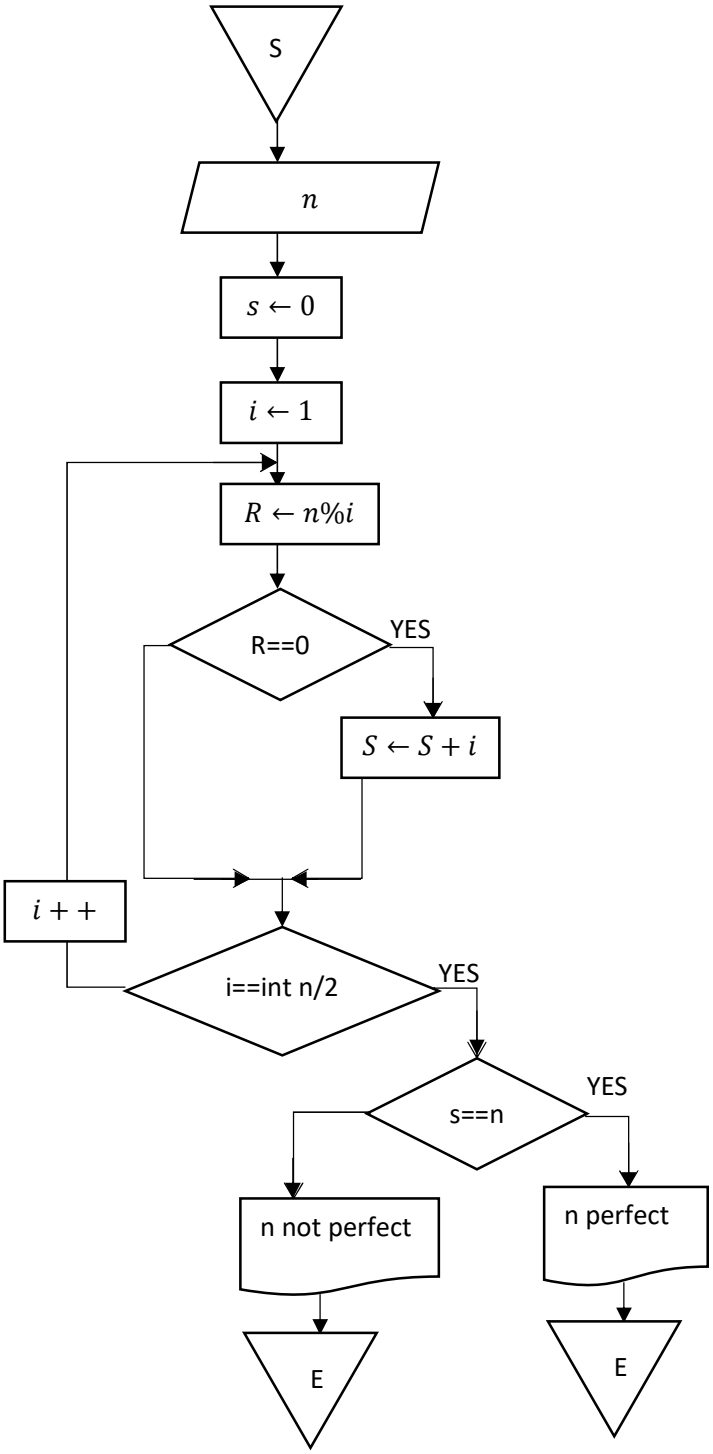
Exercise 12: Design a flowchart that reads 3 integers: iter, lig, col and displas the following shape:



Loop i has j and j has k (execution begin from k then j then i)



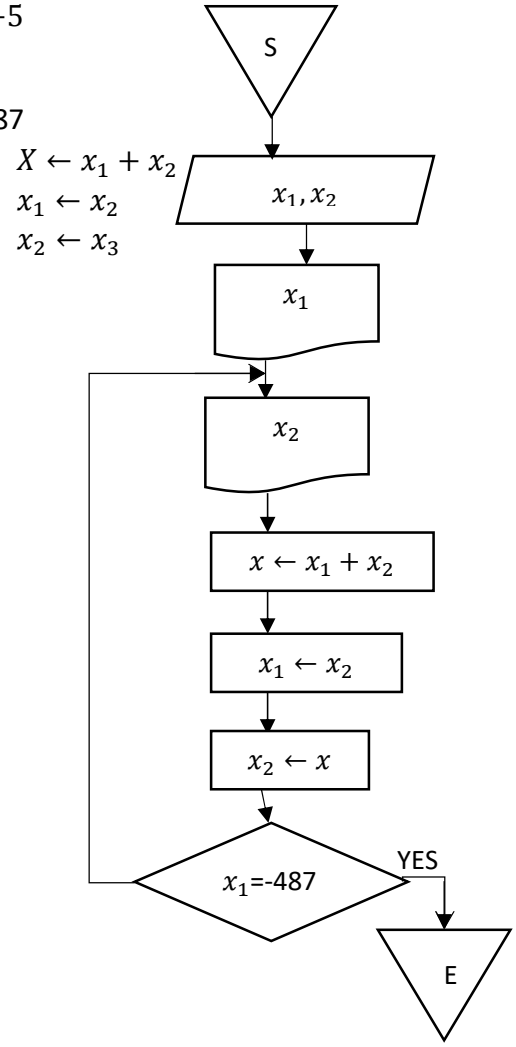
Exercise 13: A perfect number is a positif integer or a number equal to the sum of all its divisions except itself.
Ex: 6 = 1+2+3
Design a flowchart that determines if a number input by a user is perfect or not.



Exercise 14: Design the flowchart that displays the following sequence until reaching the number -487; where $x_1 = 6\ and\ x_2 = -5$

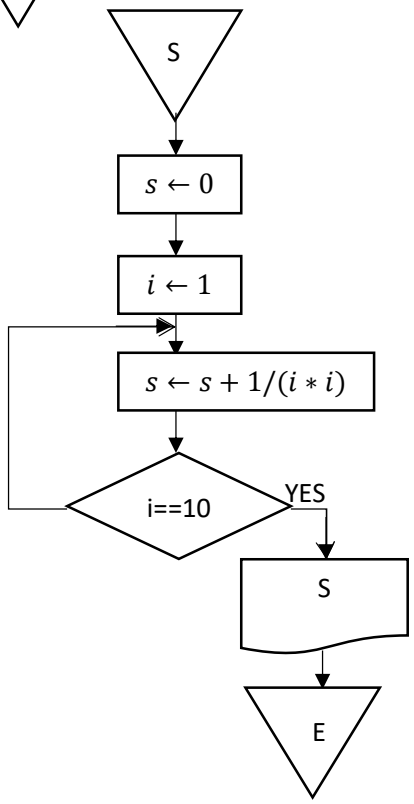
N.B.: $x_1\ and\ x_2\ are\ entered\ as\ input.$

6 -5 1 -4 -3 -7 -10 … -487



Exercise 15: Design the flowchart for computing the following sum:

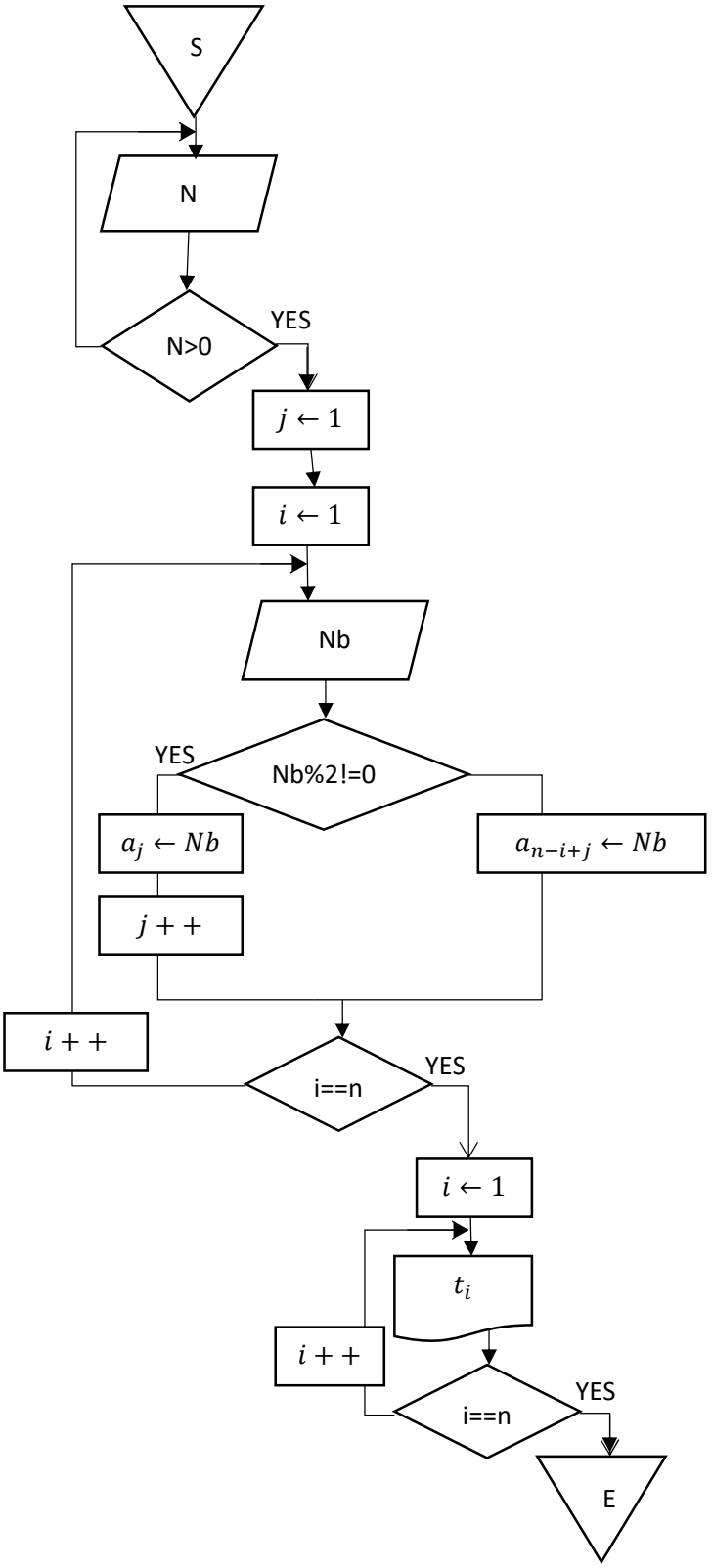
$$S = 1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{100}$$



Exercise 16: Design a flowchart relating to a program that reads n integers and places them in a table in a way that odd numbers are displayed first, then the even numbers. You may use only one array of integers a_j, and 4 other variables of type integers, including one named “Nb” for representing the “n” numbers.

Odd numbers	Even numbers

j odd numbers

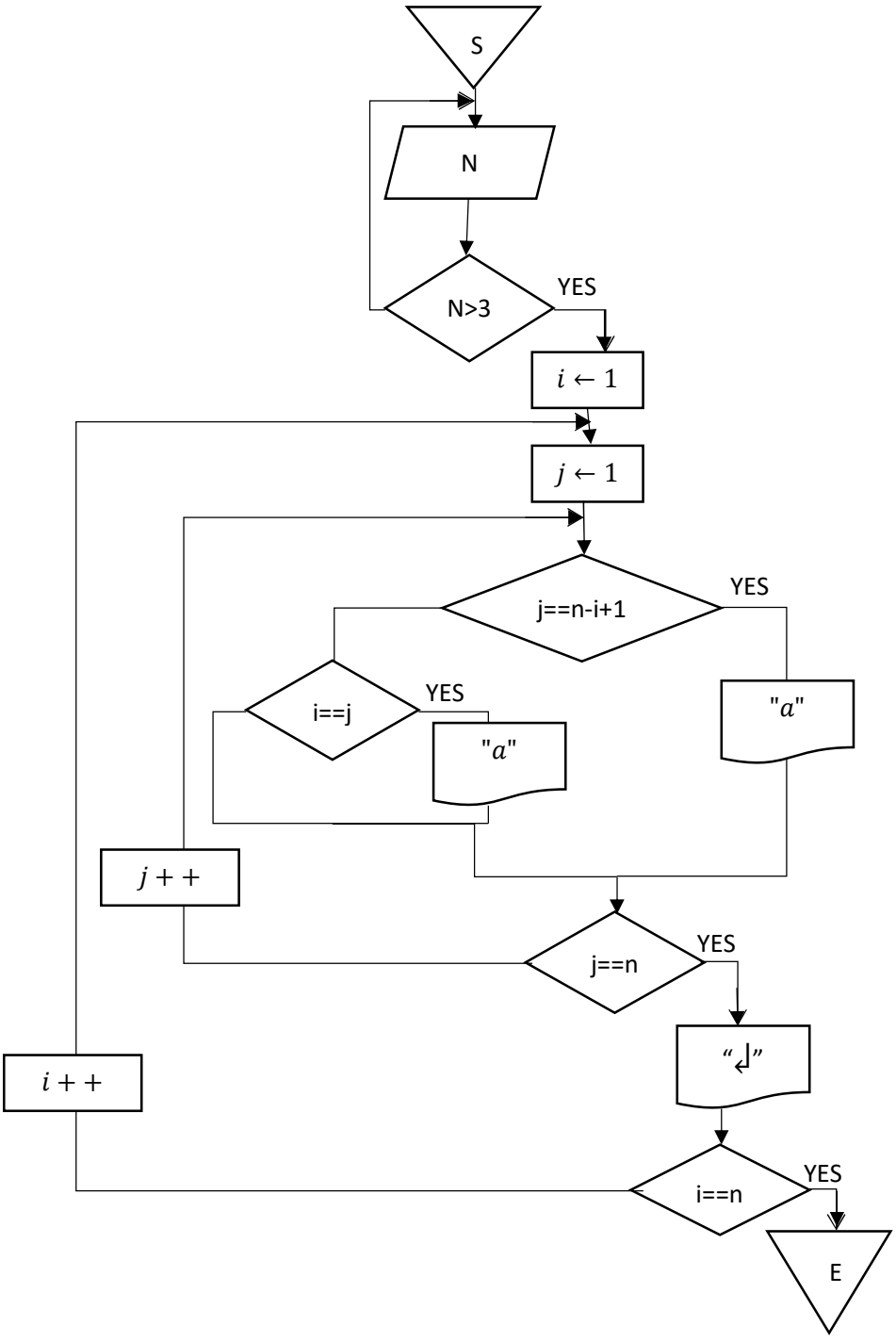


Exercise 17: Design the flowchart that read an odd integer n greater than 3 (responsible for the height) and displays the following shape:

Height=5

N odd

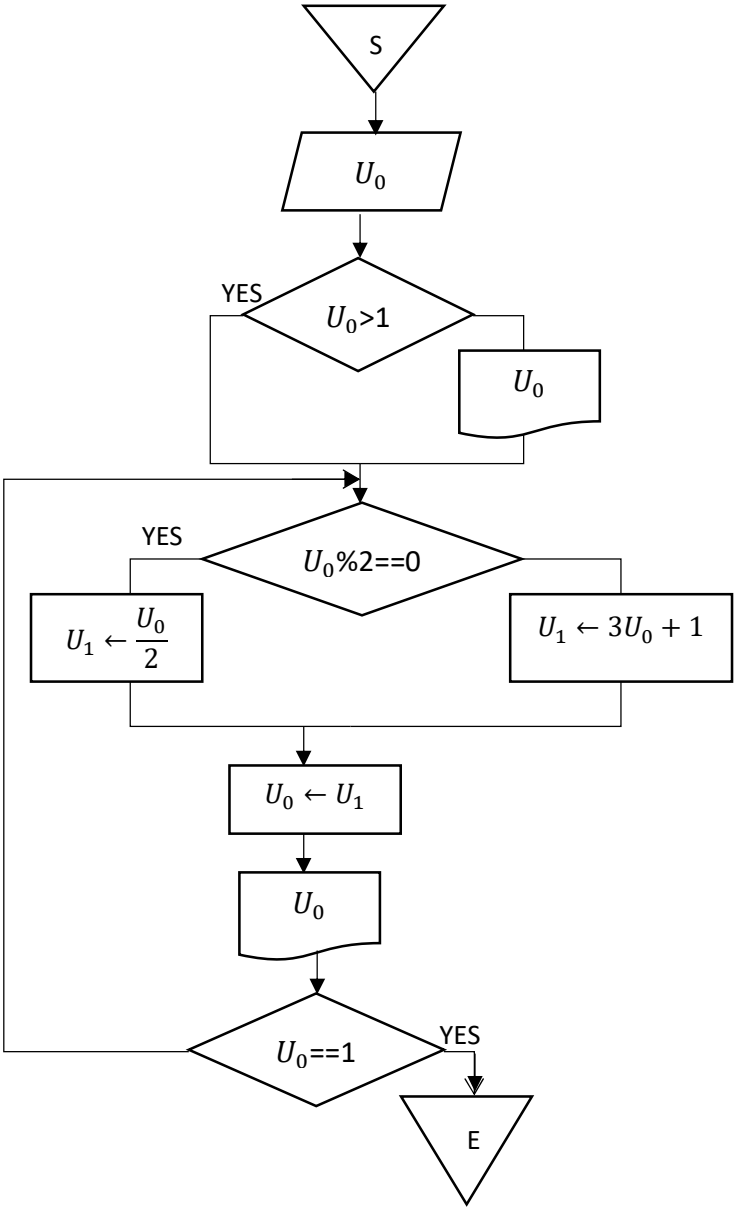
a___a
 a_a
  a
 a_a
a___a



Exercise 18: Let the numerical sequence be as following

$$U_{n+1} = \begin{cases} \frac{U_n}{2} \; if \; U_n \; is \; even \\ 3U_n + 1 \; if \; U_n \; is \; odd \end{cases}$$

Design a flowchart related to a program that reads the value of $U_0 \; and \; outputs$ on the screen the values of U_n respectively until obtaining the value "1"

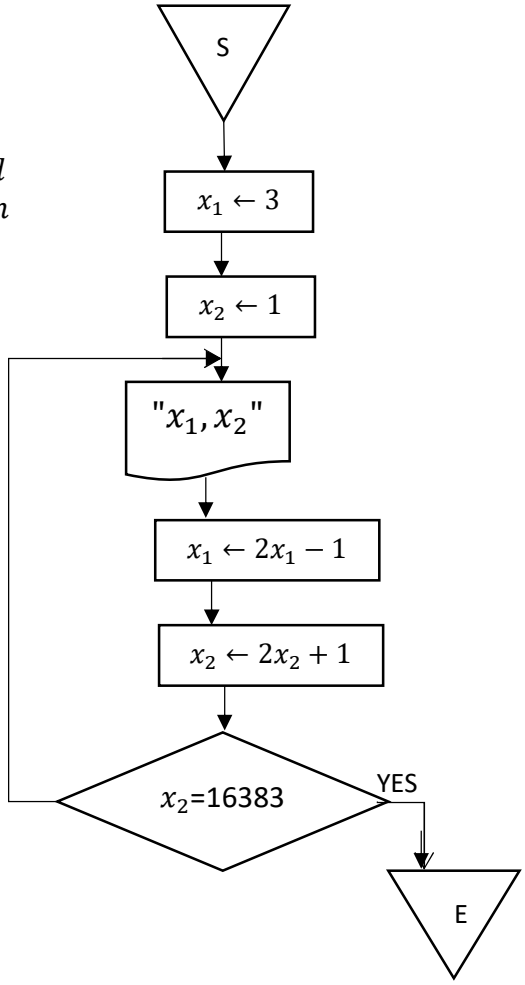


Exercise 19: Design the relationship between the following elements of the sequence until reaching $x_2 = 16383$ where $x_1 = 3\ and\ x_2 = 1\ are\ entered\ as\ input$

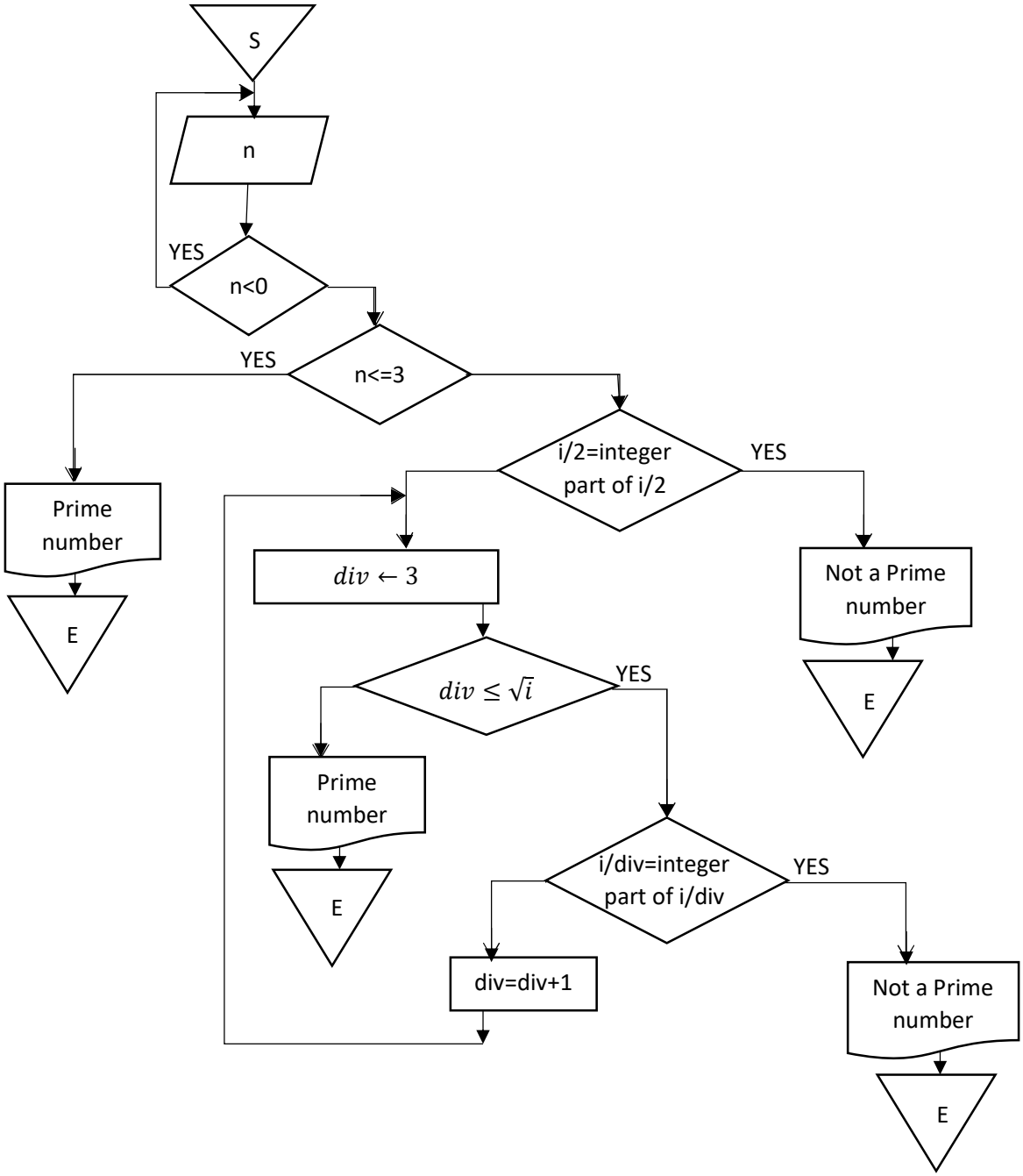
3 1 5 3 … 16383

Solution:

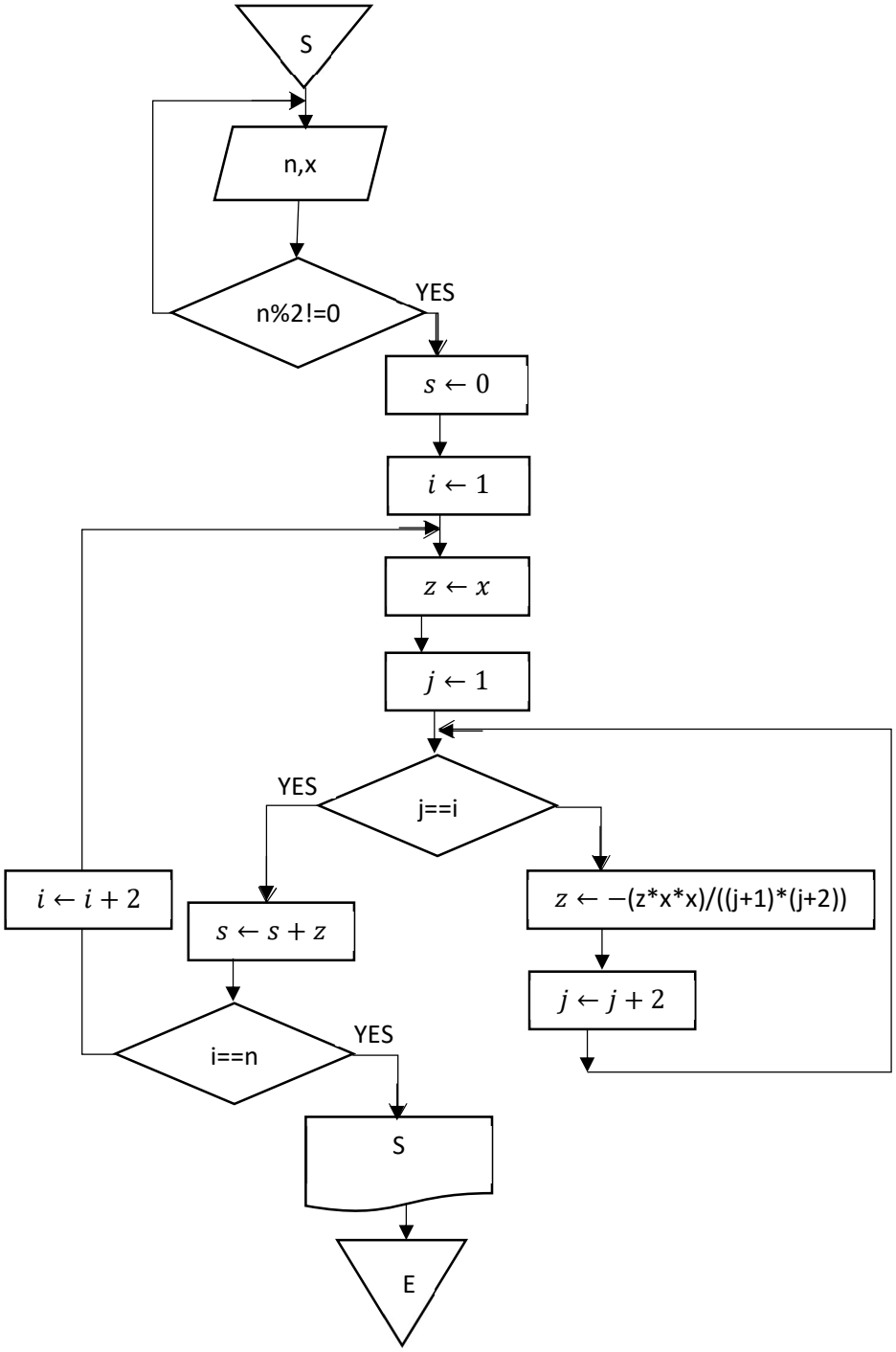
$$5 = 2x_1 - 1$$
$$3 = 2x_2 + 1$$
$$\begin{cases} 2x_{n-2} - 1 & n: odd \\ 2x_{n-2} + 1 & n: even \end{cases}$$



Exercise 20: Draw the flowchart that indicates if a number is prime or not.



Exercise 21: $Sinx = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$

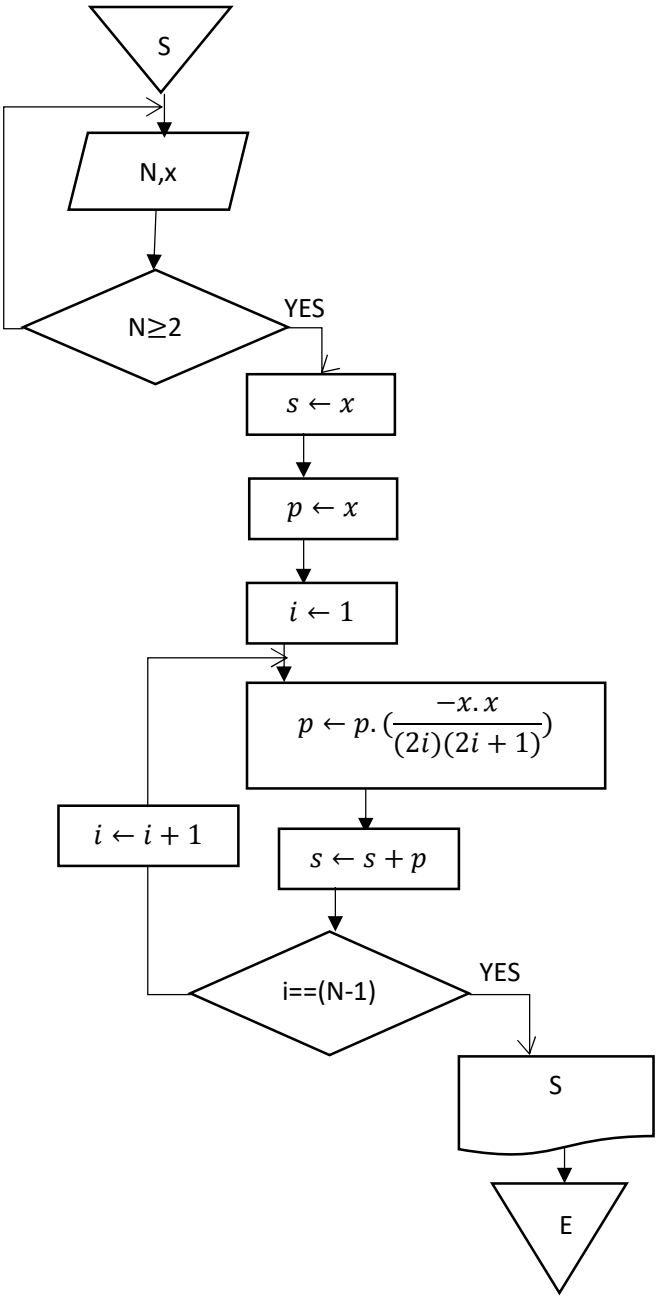


Session 21/11/2013
Prof: W. Hobeika

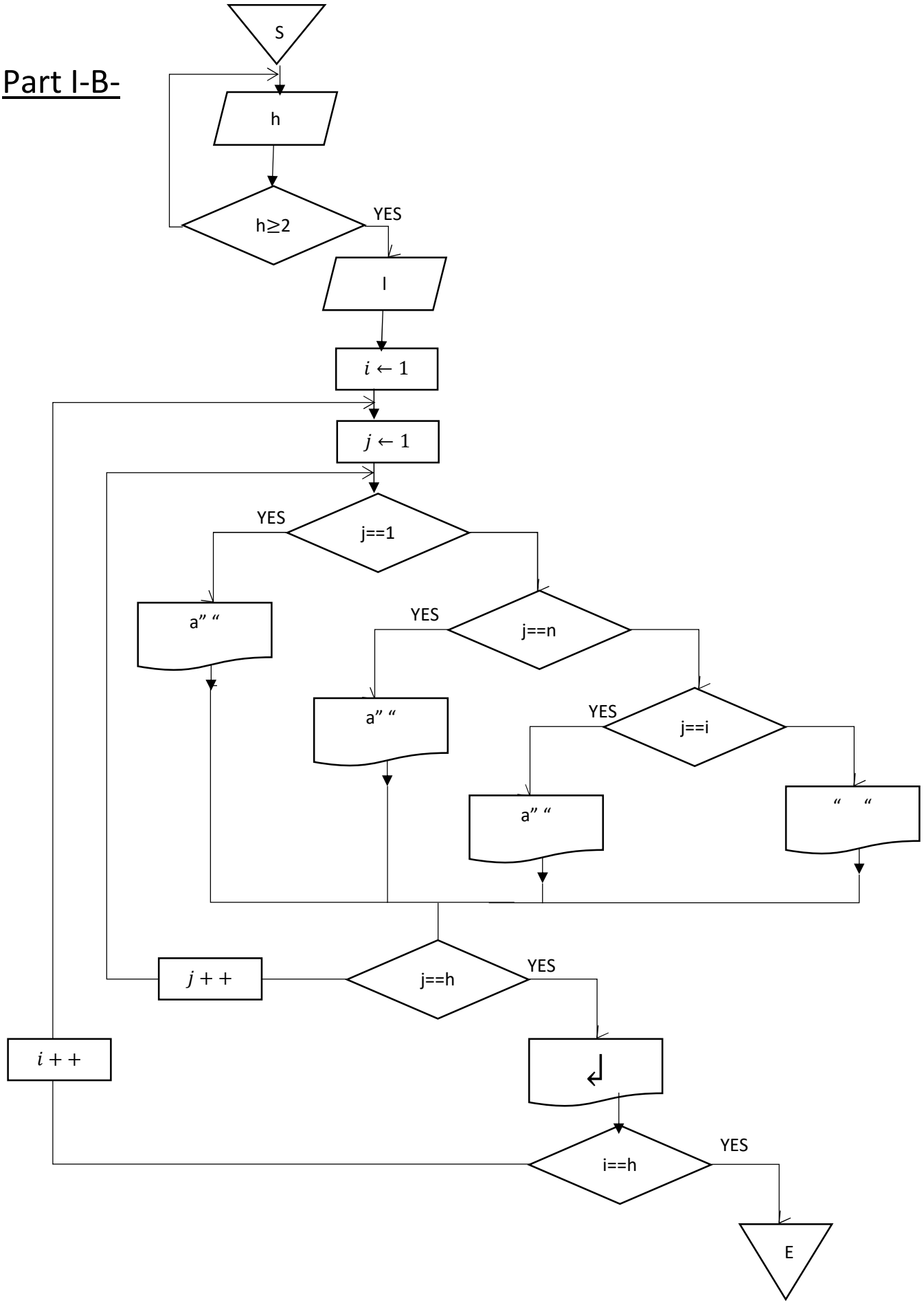
Part I-A- $Sinx = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$

2nd method:

Use 3 variables
Arrays are forbidden



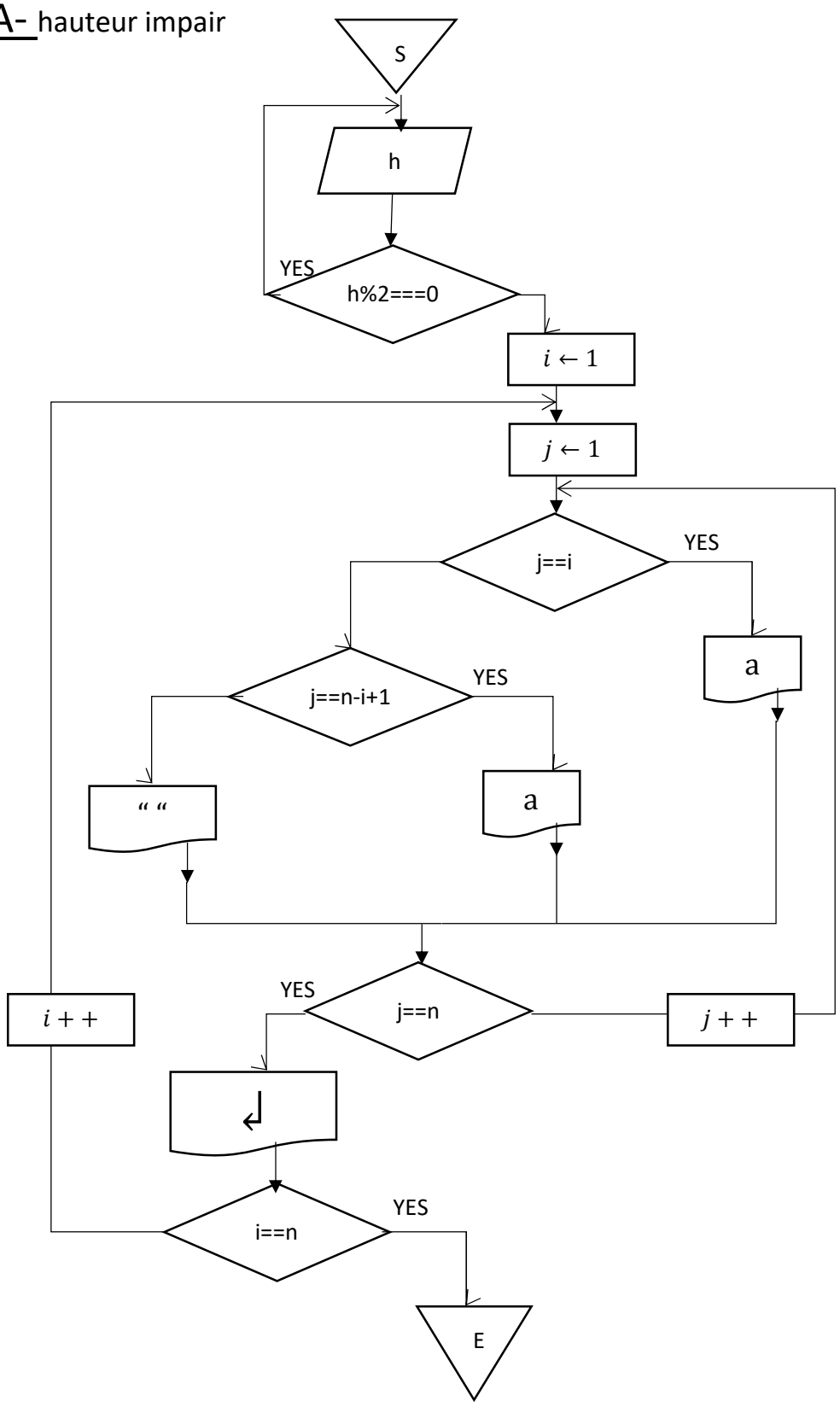
Part I-B-


S
h
h≥2
YES
l
$i \leftarrow 1$
$j \leftarrow 1$
j==1
YES
a" "
j==n
YES
a" "
j==i
YES
a" "
" "
j==h
YES
$j++$
$i++$
↲
i==h
YES
E

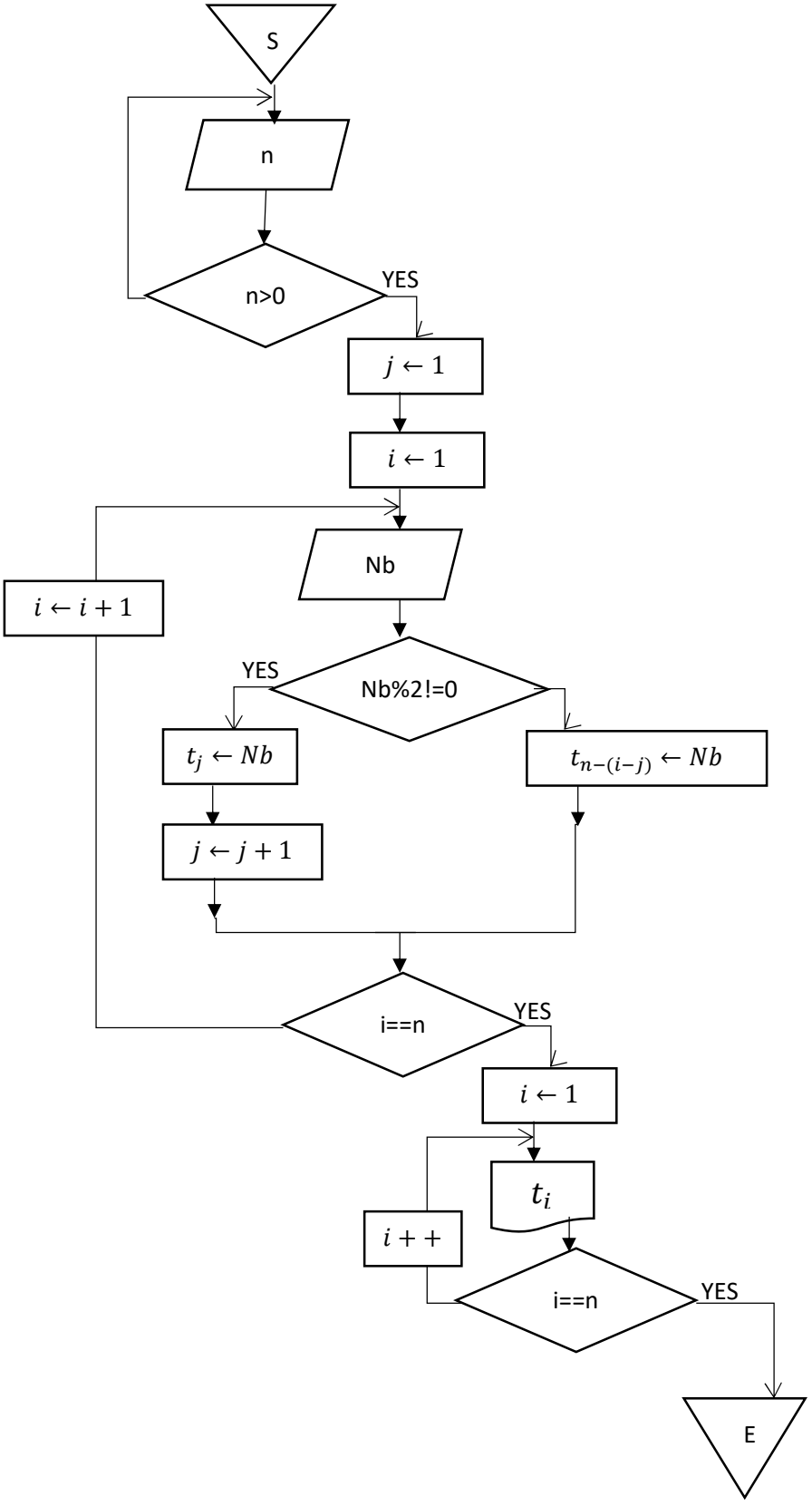
Session 30/12/2012

Prof : W. Hobeika

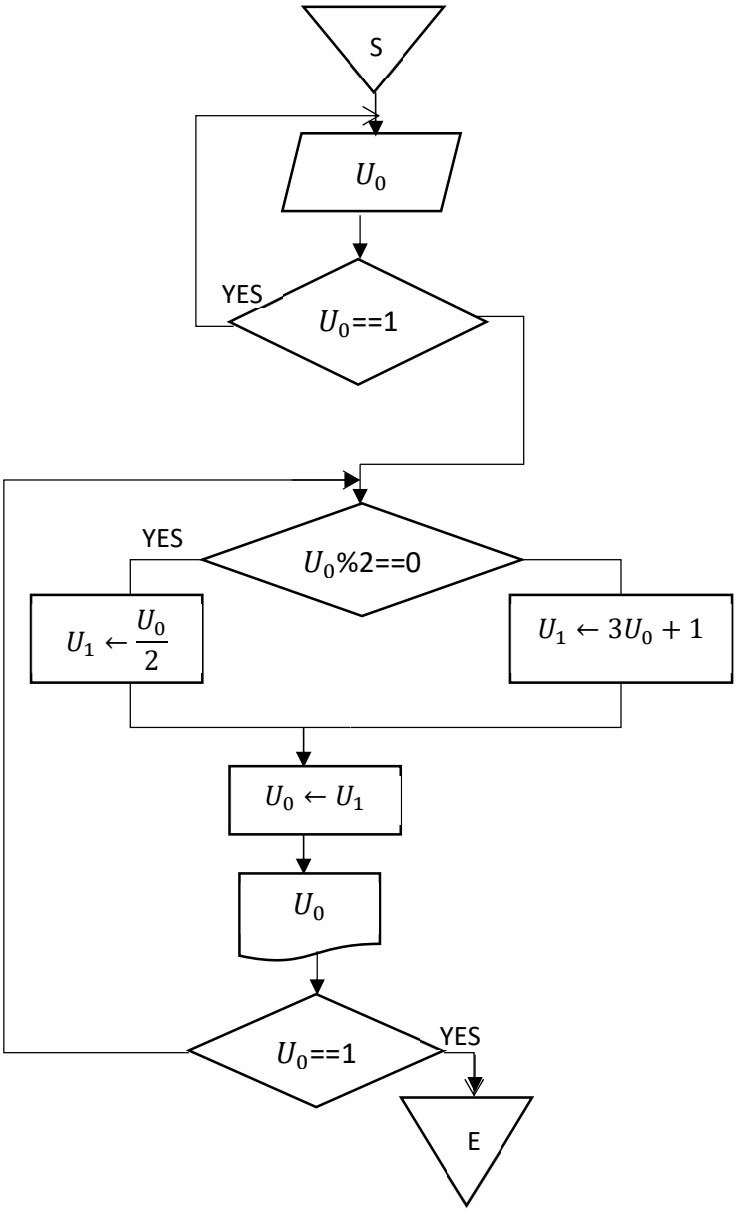
Part III-A- hauteur impair



Part III-B-


S
n
n>0
YES
$j \leftarrow 1$
$i \leftarrow 1$
Nb
$i \leftarrow i + 1$
YES
Nb%2!=0
$t_j \leftarrow Nb$
$t_{n-(i-j)} \leftarrow Nb$
$j \leftarrow j + 1$
i==n
YES
$i \leftarrow 1$
t_i
$i++$
i==n
YES
E

Part III-C-

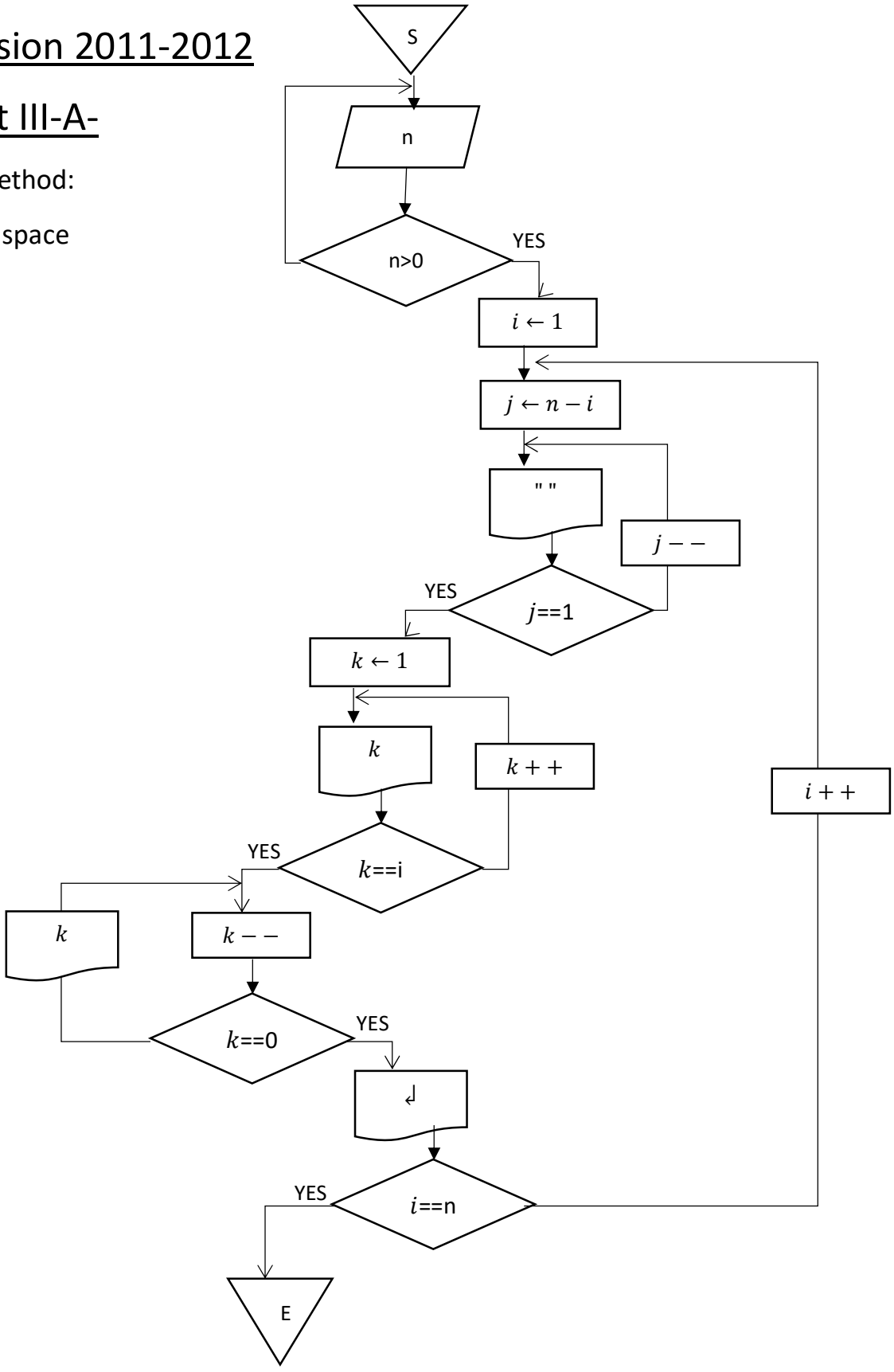

S
U_0
YES
U_0==1
YES
U_0%2==0
$U_1 \leftarrow \frac{U_0}{2}$
$U_1 \leftarrow 3U_0 + 1$
$U_0 \leftarrow U_1$
U_0
U_0==1
YES
E

Session 2011-2012

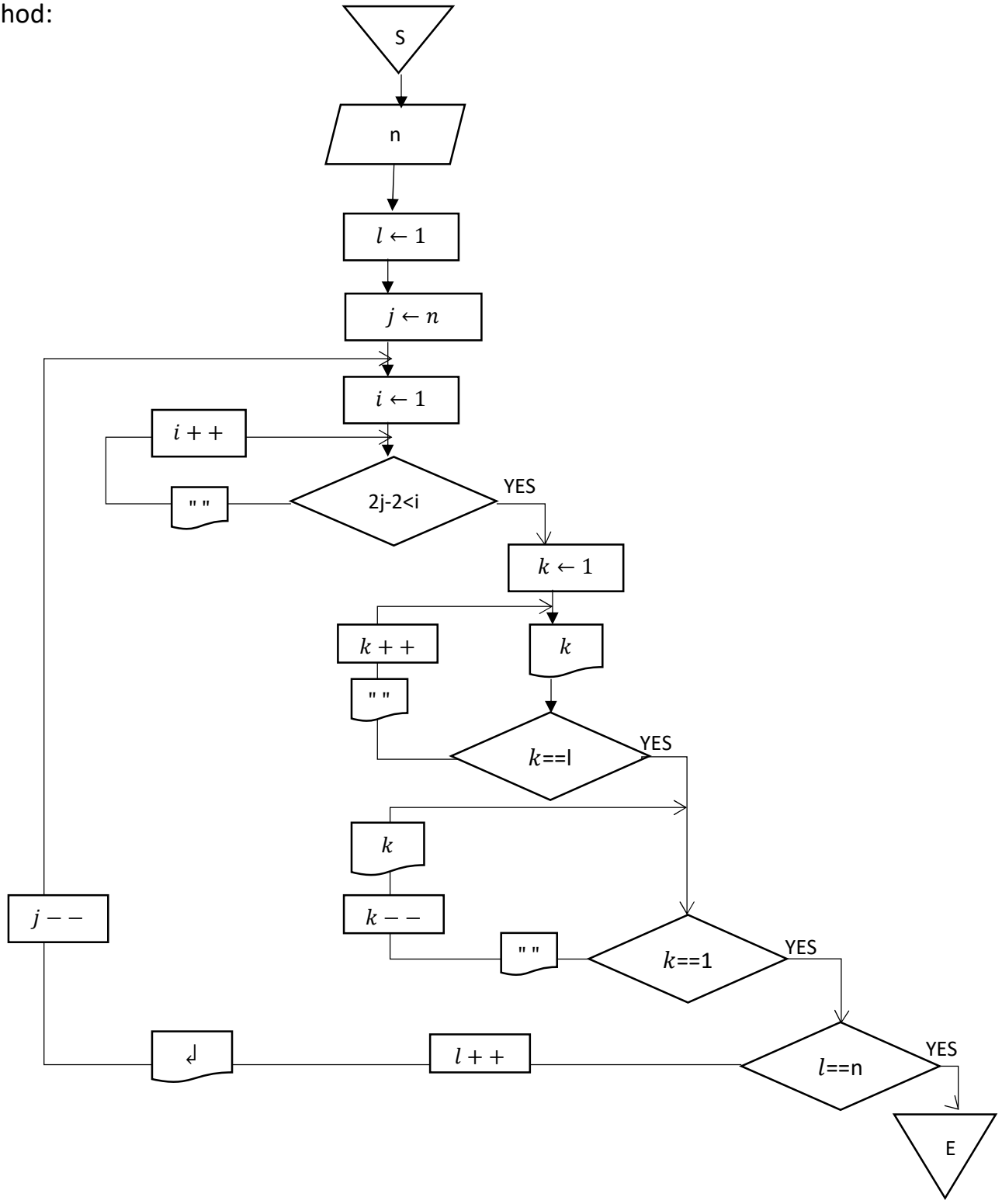
Part III-A-

1st method:

’’ ‘‘ : space



2nd method:


S
n
$l \leftarrow 1$
$j \leftarrow n$
$i \leftarrow 1$
$i++$
" "
2j-2<i
YES
$k \leftarrow 1$
$k++$
" "
k
k==l
YES
k
$k--$
" "
k==1
YES
$j--$
↲
$l++$
l==n
YES
E