CS-101 & IT-101 : Data Structures and Algorithms

Unit I
Lecture 3: Introduction to DS

Dr. Ajay S. Patil

School of Computer Sciences, KBC NMU, Jalgaon (Maharashtra)

Syllabus of Unit I:

Introduction to Data Structures and Algorithms:

Data types and Abstract data types, Types of Data structures; Primitive, Non primitive, Linear and Nonlinear Data structures

3

Algorithmic Notation: Format Conventions, Statement and Control Structures.

Time and Space Analysis

What we have studied so far …

- Discussion of some prerequisites
  - What is an IDE
  - Include, Bin, Lib directories in C/C++ program development environment
  - How a Program is Compiled to create an executable file
  - Role of editor, pre-processor, compiler, linker, libraries etc
  - Run Time, Compile Time
  - Local and Global variables
  - Static and Auto variables
  - Pointers
- What is Data Structure? ✓
- Types of Data Structures ✓

- **Algorithmic Notations**
  - **Format Conventions**
  - **Statement and Control Structures**
  - **Name, Introductory Comment, Steps, Comments, Assignment,**
  - **If-then, if-then-else, select case**
  - **Repeat for, repeat while, repeat step … through**
  - **Go To, Exitloop and Exit Statements, Variable names**
  - **Data structures: (Non-primitive) Array, Dynamic Storage**
  - **Arithmetic Operations and Expressions**
  - **Relations and Relational Operators**
  - **Logical Operations and Expressions**
  - **Input and Output**
  - **Sub Algorithms: Functions, Procedures**

What we will study today …

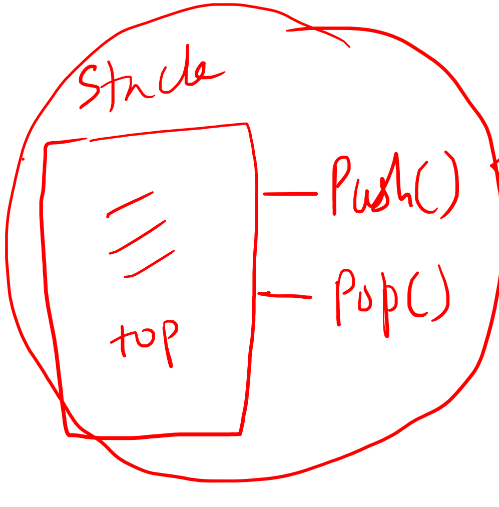
- Abstract Data Types
- Algorithmic Notations:
- Format Conventions,
  - Statement and Control Structures

Web Reference:

http://what-when-how.com/compiler-writing/algorithmic-notation-compiler-writing-part-1/

Abstract Data Types

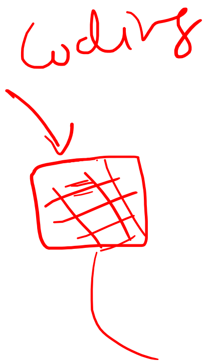
Q. Explain 'stack' as a abstract data type.



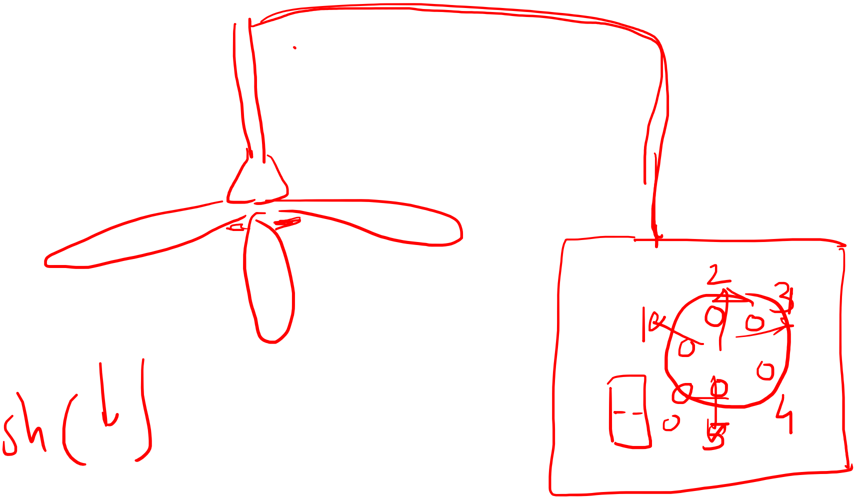
```
<stack.h>

stack a;

a.push(10);
z=a.pop();
```



push()
pop()

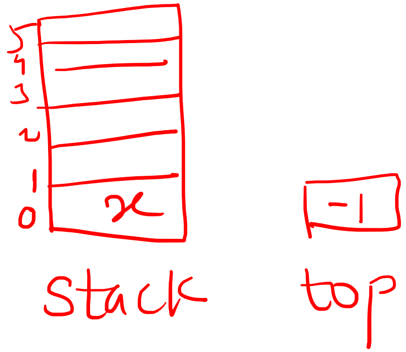


ADT mathematical model of a Data structure:

```
struct stack
{
  int array[100];
  int top;
};
```

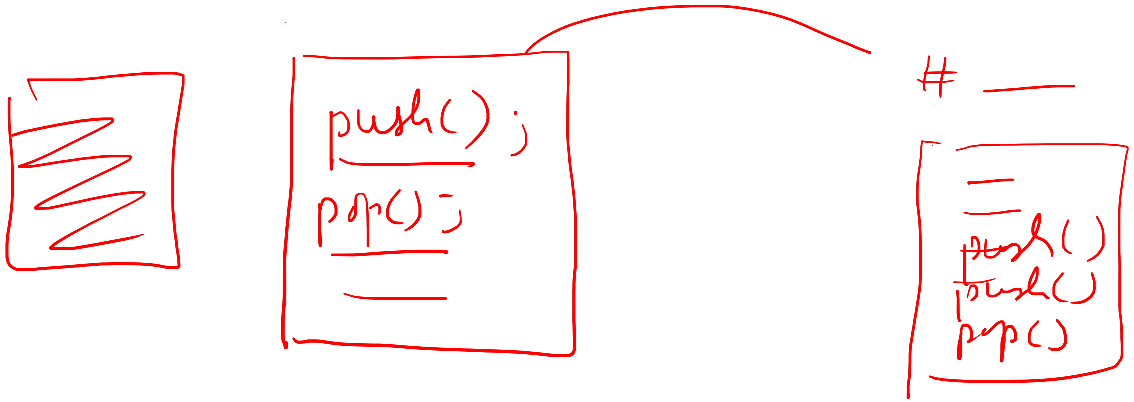
```
push( )
{
  ==
}
```

```
pop()
{
  ==
}
```



```
push(int)

if (top==-1)
  top=0;
  stack[top]=x
```



Algorithmic Notations

Name of Algorithm

✓ (Jay Pandit)

Every algorithm is given an identifying name written in capital letters.

Algorithm PRIME

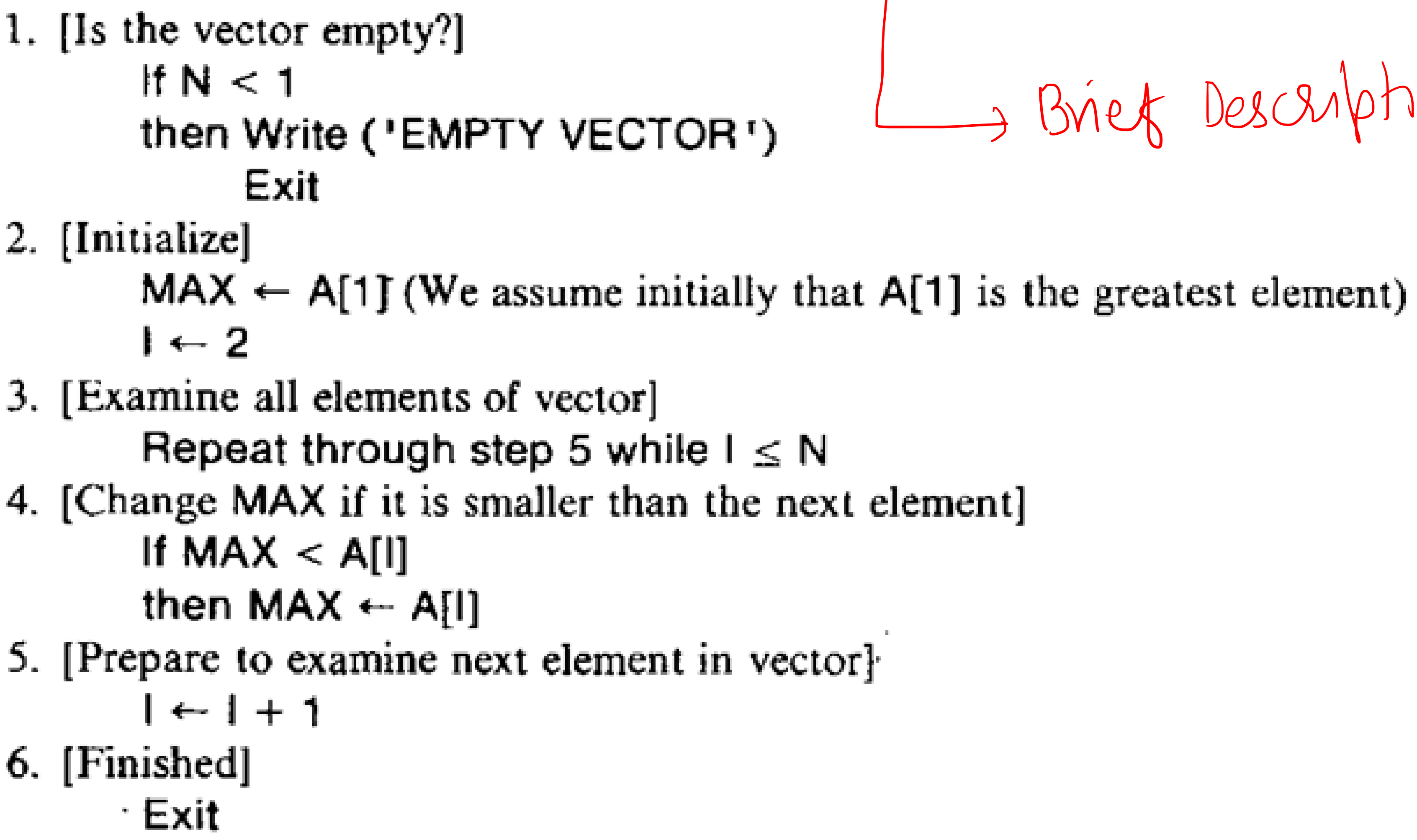
Algorithm FIBO

Algorithm evenodd ✗

11

Algorithm GREATEST. This algorithm finds the largest algebraic element of vector A which contains N elements and places the result in MAX. I is used to subscript A.

Brief Description



```
1. [Is the vector empty?]
        If N < 1
        then Write ('EMPTY VECTOR')
                Exit
2. [Initialize]
        MAX ← A[1] (We assume initially that A[1] is the greatest element)
        I ← 2
3. [Examine all elements of vector]
        Repeat through step 5 while I ≤ N
4. [Change MAX if it is smaller than the next element]
        If MAX < A[I]
        then MAX ← A[I]
5. [Prepare to examine next element in vector]
        I ← I + 1
6. [Finished]
        Exit
```

Introductory Comment

The algorithm name is followed by a **brief description** of the **tasks the algorithm performs** and **any assumptions that have been made**.

The description gives the **names and types of the variables** used in the algorithm.

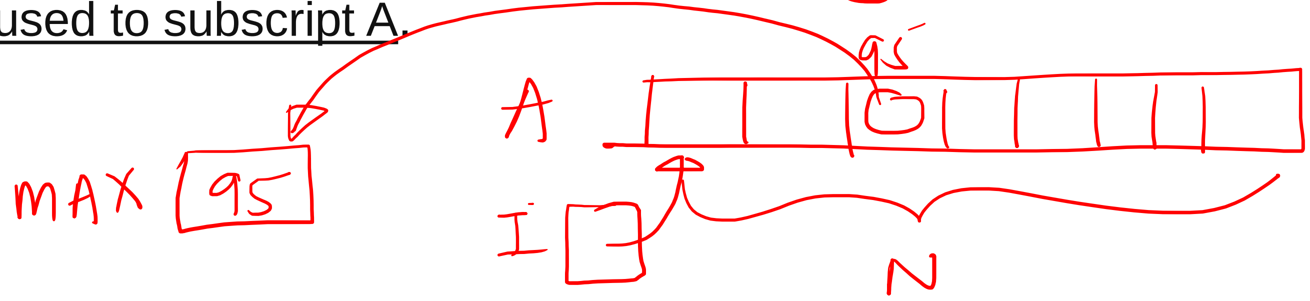
Vector A

Anuja

task

A[I]

Algorithm GREATEST. This algorithm finds the largest algebraic element of vector A which contains N elements and places the result in MAX. I is used to subscript A.

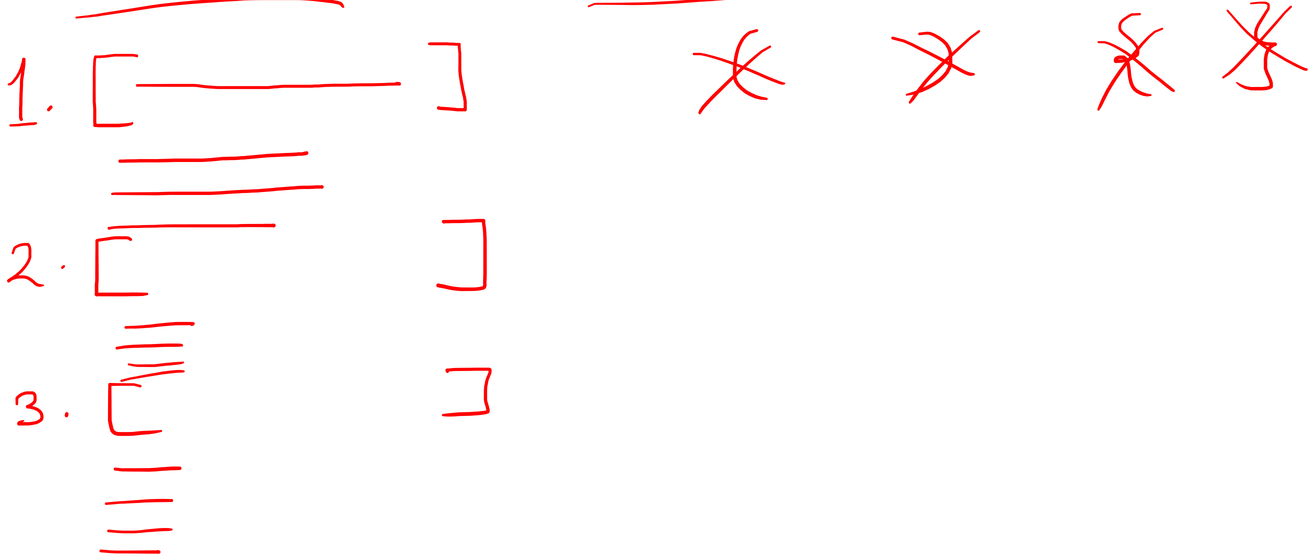


Steps

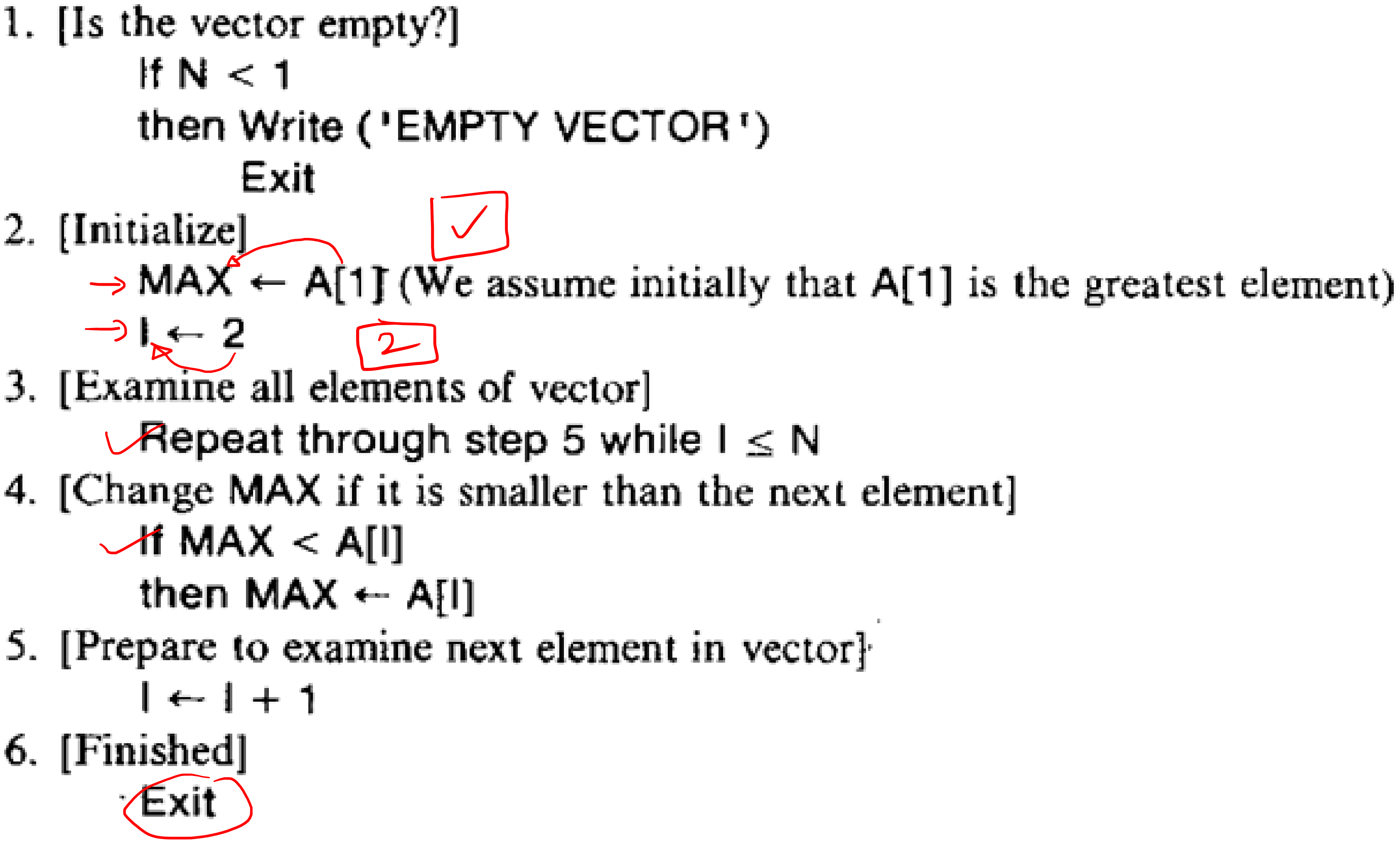
The actual algorithm is made up of a sequence of **numbered steps**, each beginning with a **phrase enclosed in square brackets** which gives an abbreviated description of that step.

shortcut

Following this phrase is an **ordered sequence of statements** which describe actions to be executed or tasks to be performed.



Algorithm GREATEST. This algorithm finds the largest algebraic element of vector A which contains N elements and places the result in MAX. I is used to subscript A.



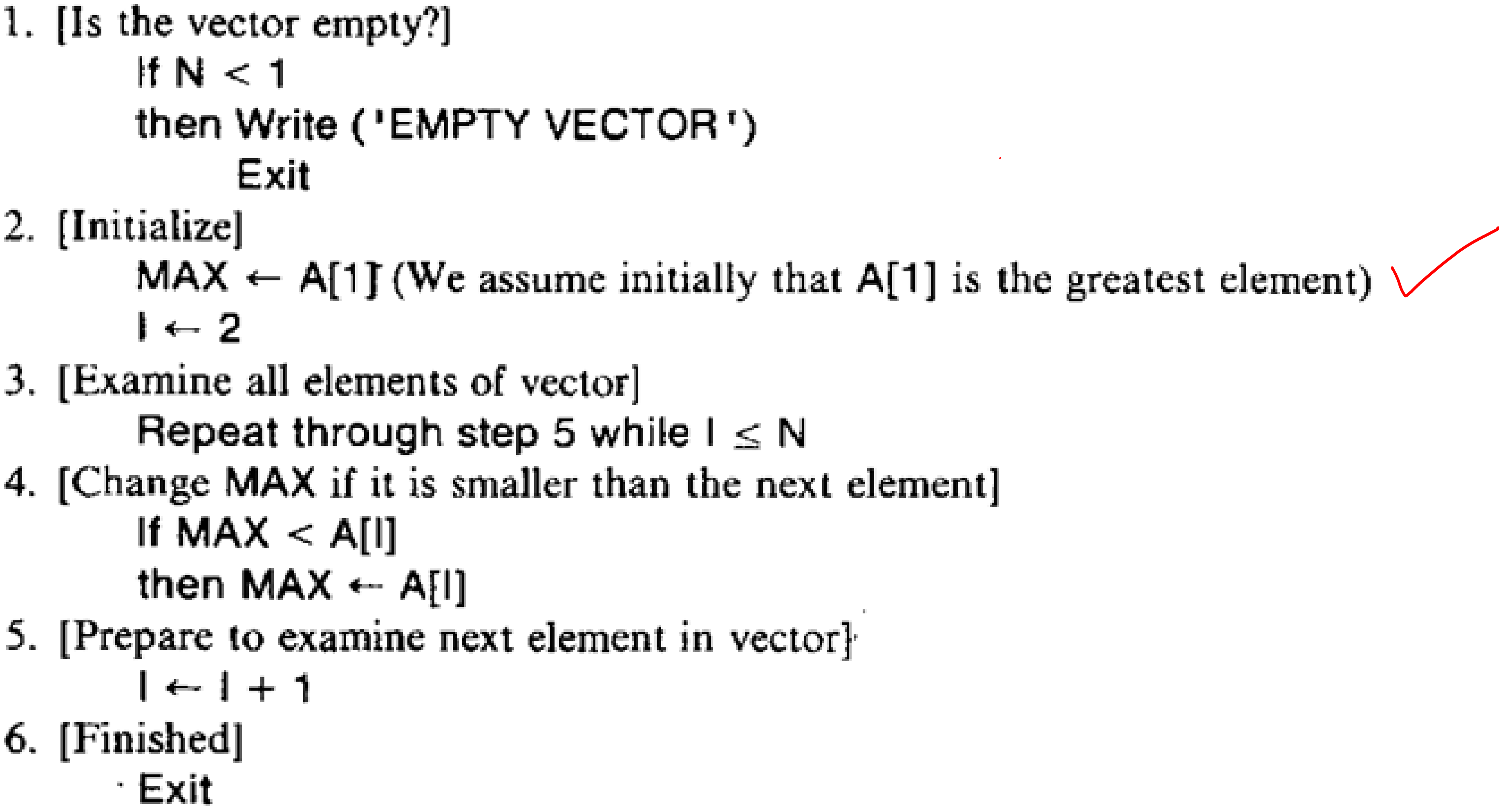
```
1. [Is the vector empty?]
        If N < 1
        then Write ('EMPTY VECTOR')
             Exit
2. [Initialize]                    ✓
        MAX ← A[1] (We assume initially that A[1] is the greatest element)
        I ← 2             2
3. [Examine all elements of vector]
        Repeat through step 5 while I ≤ N
4. [Change MAX if it is smaller than the next element]
        If MAX < A[I]
        then MAX ← A[I]
5. [Prepare to examine next element in vector]
        I ← I + 1
6. [Finished]
        Exit
```

Comments

Comments specify no action and are included only for clarity.

Comments help the reader better understand a step.

Algorithm GREATEST. This algorithm finds the largest algebraic element of vector A which contains N elements and places the result in MAX. I is used to subscript A.



```
1. [Is the vector empty?]
        If N < 1
        then Write ('EMPTY VECTOR')
            Exit
2. [Initialize]
        MAX ← A[1] (We assume initially that A[1] is the greatest element) ✓
        I ← 2
3. [Examine all elements of vector]
        Repeat through step 5 while I ≤ N
4. [Change MAX if it is smaller than the next element]
        If MAX < A[I]
        then MAX ← A[I]
5. [Prepare to examine next element in vector]
        I ← I + 1
6. [Finished]
        Exit
```

STATEMENTS AND CONTROL STRUCTURES

a = 2;

a 2

Assignment Statement

The assignment statement is indicated by placing an arrow between the right-hand side of the statement and the variable receiving the value.

~~**A → 0**~~ A ← 0

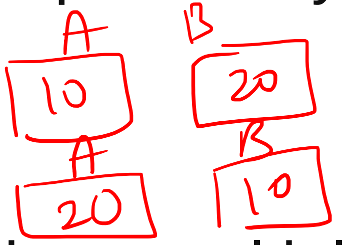
The symbol = is used as a relational operator and never as an assignment operator.

A = 0

if (a == 0)

An exchange of the values of two variables is accomplished by:

A ↔ B



C = A
A = B
B = C

HW

Many variables can be set to the same value by using a multiple assignment:

A ← B ← C ← 0

a = b = c = 0;

The if statement has one of the following two forms:

1. If *condition*
   then ________
   ____
   ⋮
   ____

2. If *condition*
   then ________
   ____
   ⋮
   ____
   else ________
   ____
   ⋮
   ____

① ✓ a=10; ② a=100

```
if ( a==10)
printf("Hello");
printf("world");
```

```
if (a= =10)
printf("Hello");
printf("world");
```

```
if(a==0)
{
printf("Hello");
printf("world");
}
```

① Hello/World ② ~~no output~~

3 World ✓

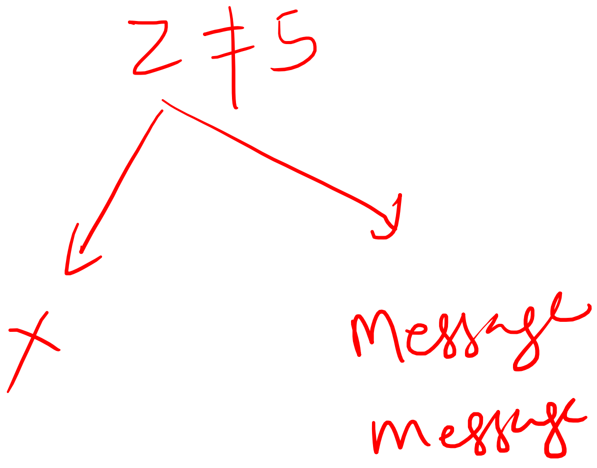
1. [ ]

```
If A=10
then write('Hello')
else write('World');
```

simple

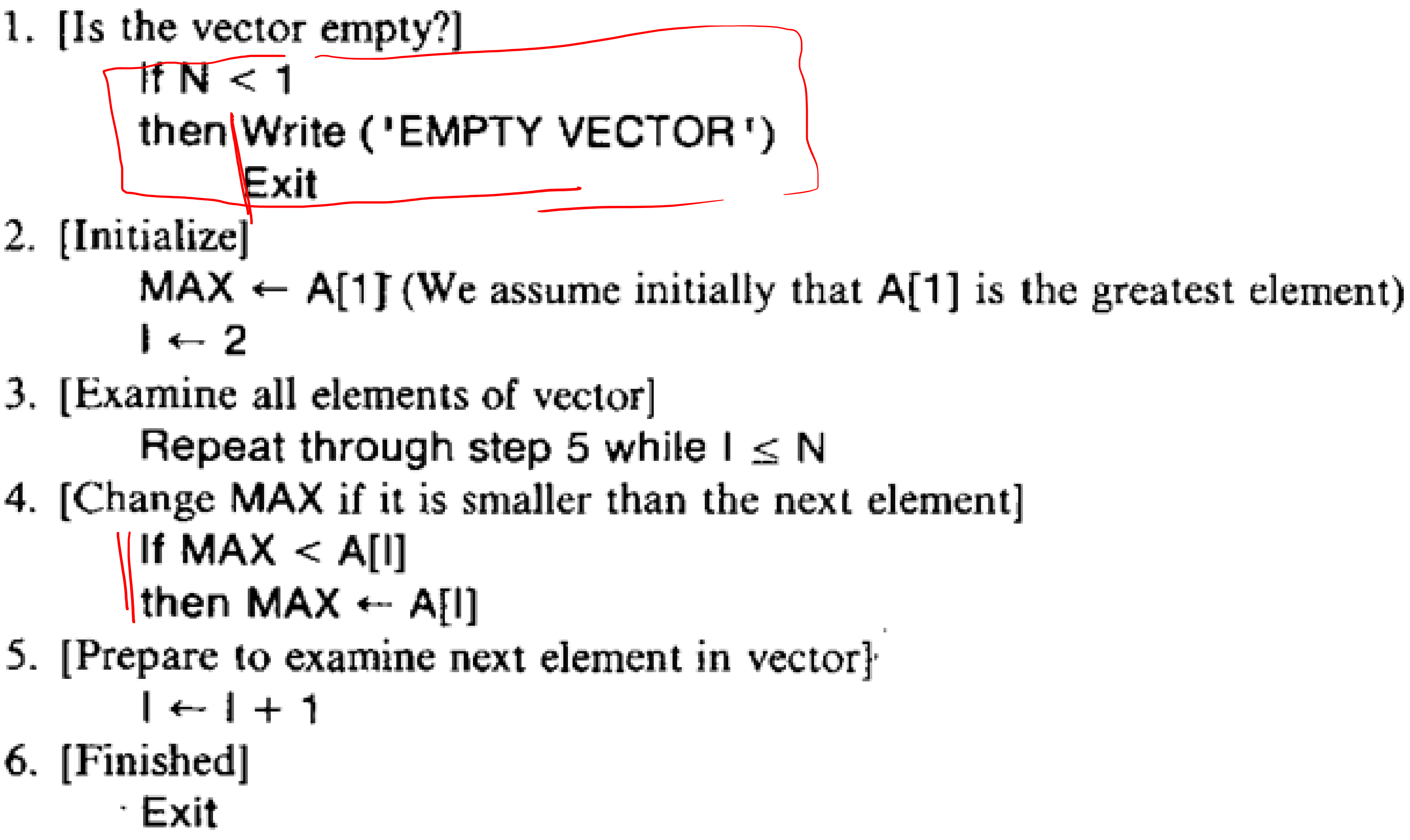
Compound statement

```
If Z=5
then A<-3
     write('Message')
     write('message')
```



```
If Z=5
then A<-3
write('message')
write('Message')
```

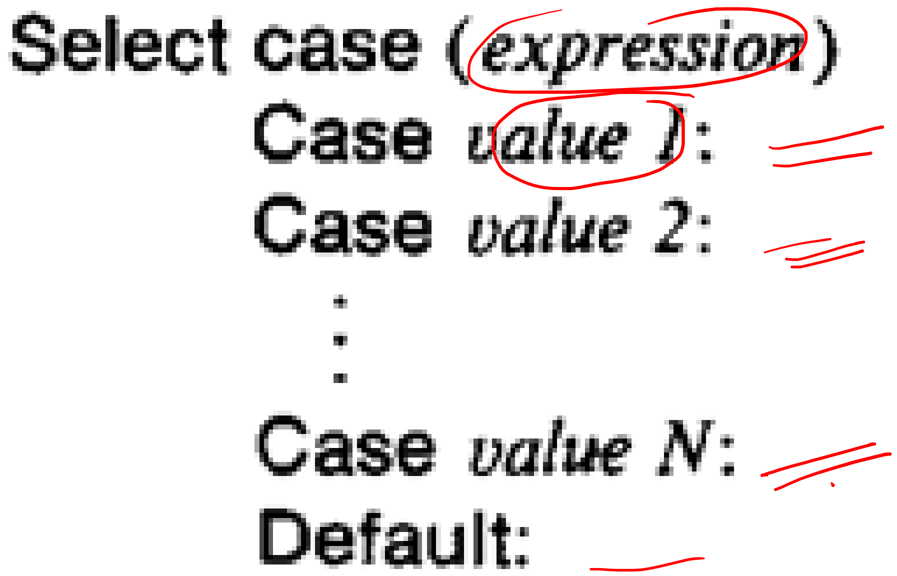
Algorithm GREATEST. This algorithm finds the largest algebraic element of vector A which contains N elements and places the result in MAX. I is used to subscript A.



```
1. [Is the vector empty?]
        If $N < 1$
        then Write ('EMPTY VECTOR')
             Exit
2. [Initialize]
        MAX $\leftarrow$ A[1] (We assume initially that A[1] is the greatest element)
        I $\leftarrow$ 2
3. [Examine all elements of vector]
        Repeat through step 5 while $I \leq N$
4. [Change MAX if it is smaller than the next element]
        If MAX < A[I]
        then MAX $\leftarrow$ A[I]
5. [Prepare to examine next element in vector]
        I $\leftarrow$ I + 1
6. [Finished]
        Exit
```

Case Statement

The case statement is used when a choice from among several mutually exclusive alternatives must be made on the basis of the value of an expression.



```
Select case (expression)
        Case value 1:
        Case value 2:
          :
        Case value N:
        Default:
```

```
Select Case (z)
    Case 1: —
    Case 2: —
    Case 3: —
    Default: —
```

```
scanf("%d",&n);

switch(n )
{
case 1:  printf("one");break;

case 2:  printf("Two");break;

case 3:  printf("Three");break;

default: printf("other");

}
```

5	other
3	Three ✓ other ✓
2	Two ✓ Three ✓
1	one ✓

Repeat Statement

For easy control of iteration (looping), a repeat statement has been provided. This statement has one of the following forms:

1. Repeat for INDEX = sequence ✓

2. Repeat while logical expression ✓

3. Repeat for INDEX = sequence while logical expression

Examples:

Repeat for I = 1,2,…, 25,

Repeat for TOP = N + K, N + K- 1…..0,

Repeat for K = 9,11,…,2*MAX + 1

```
Repeat for I = 1, 2, ..., 5
    Write (I)
```

Step 1

1
2
3
4
5

```
Repeat for I = 5, 4, ..., 1
    Write(I)
```

Step -1

5
4
3
2
1

```
Repeat for I = 6, 8, ..., 12
    Write(I)
```

6
8
10
12

Repeat for I = -1, – 2,…., 10 and ✗ -1 -1 -2 -1

Repeat for K = 5, 6,…., -17 ✗

would not cause any statements to be executed;
instead, the repeat statements would be treated as having completed their execution.

Repeat while logical expression

```
A ← 1
Repeat while A < 5
    write ('Hello')
    A ← A+1
```

5

A $\boxed{\cancel{1}\cancel{2}\cancel{3}4}$

Hello
Hello
Hello
Hello

Repeat while is used to repeat a step until a given logical expression becomes false. The evaluation and testing of the logical expression is performed at the beginning of the loop.

As a special case we may write "Repeat while true." Since true is a valid logical expression, this is, in effect, an infinite loop.

```
    ⋮
5. [Loop to read data while there remains input]
        Repeat while true
            Read(ARRAY[I])
            If there is no more data
            then Exitloop
            else I ← I + 1
            ⋮
```

```
main()
{
  —
  —
  —
  break;
  —
  —
  —
  break;
}
```

Exitloop ⇒ break;

Repeat for INDEX = sequence while logical expression

Type 3 is a combination of types 1 and 2 and is used to repeat a step for a sequence whose values are taken successively by INDEX until a logical expression is false.

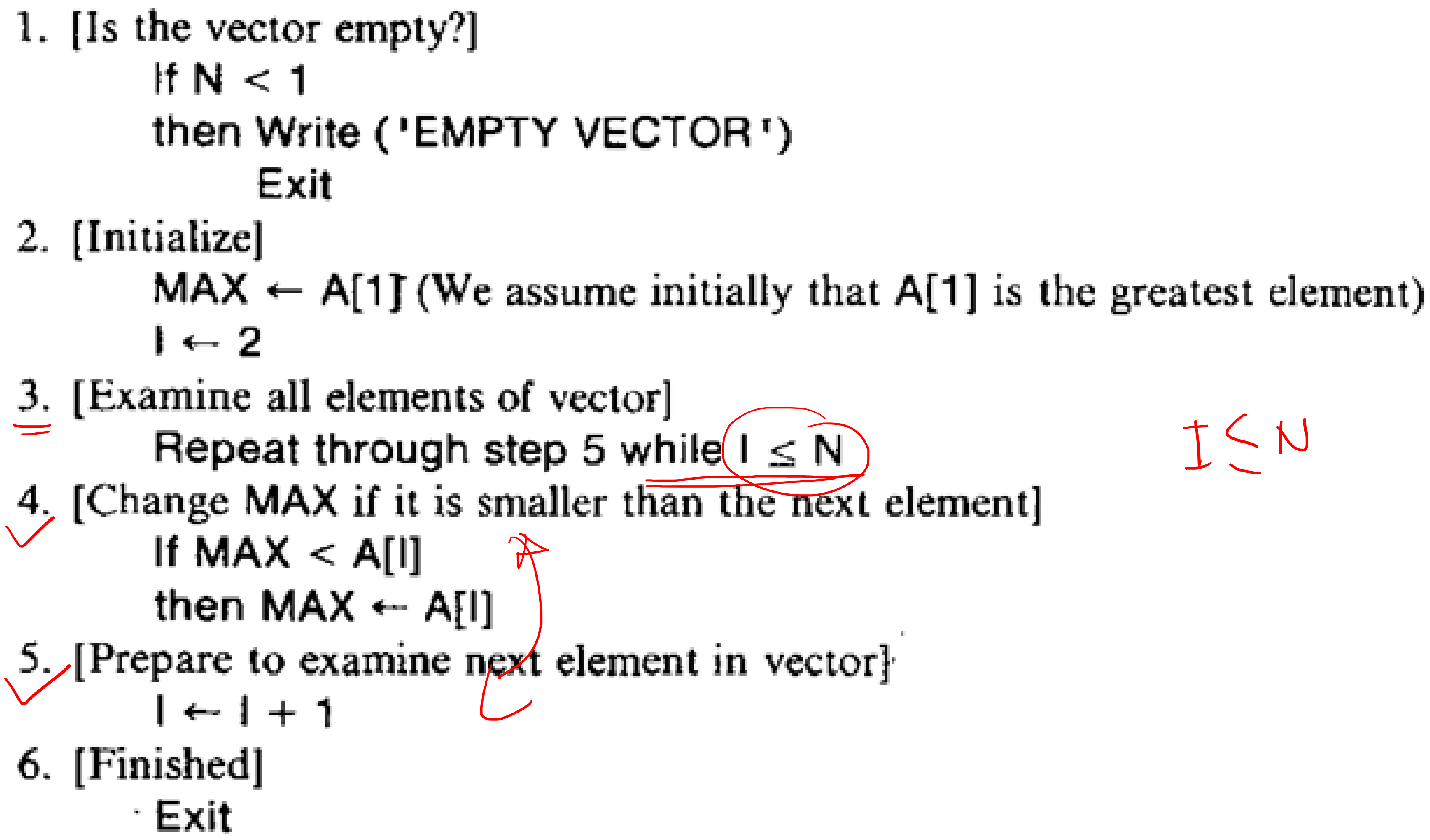
For each of the three types we have discussed, the loop may extend over more than one step, in which case the repeat statement has the form "Repeat through step N…."

1. Repeat **through step N** for INDEX = sequence

2. Repeat **through step N** while logical expression

3. Repeat **through step N** for INDEX = sequence while logical expression

Algorithm GREATEST. This algorithm finds the largest algebraic element of vector A which contains N elements and places the result in MAX. I is used to subscript A.



```
1. [Is the vector empty?]
        If N < 1
        then Write ('EMPTY VECTOR')
             Exit
2. [Initialize]
        MAX ← A[1] (We assume initially that A[1] is the greatest element)
        I ← 2
3. [Examine all elements of vector]
        Repeat through step 5 while I ≤ N
4. [Change MAX if it is smaller than the next element]
        If MAX < A[I]
        then MAX ← A[I]
5. [Prepare to examine next element in vector]
        I ← I + 1
6. [Finished]
        Exit
```

I ≤ N

Thank You All !