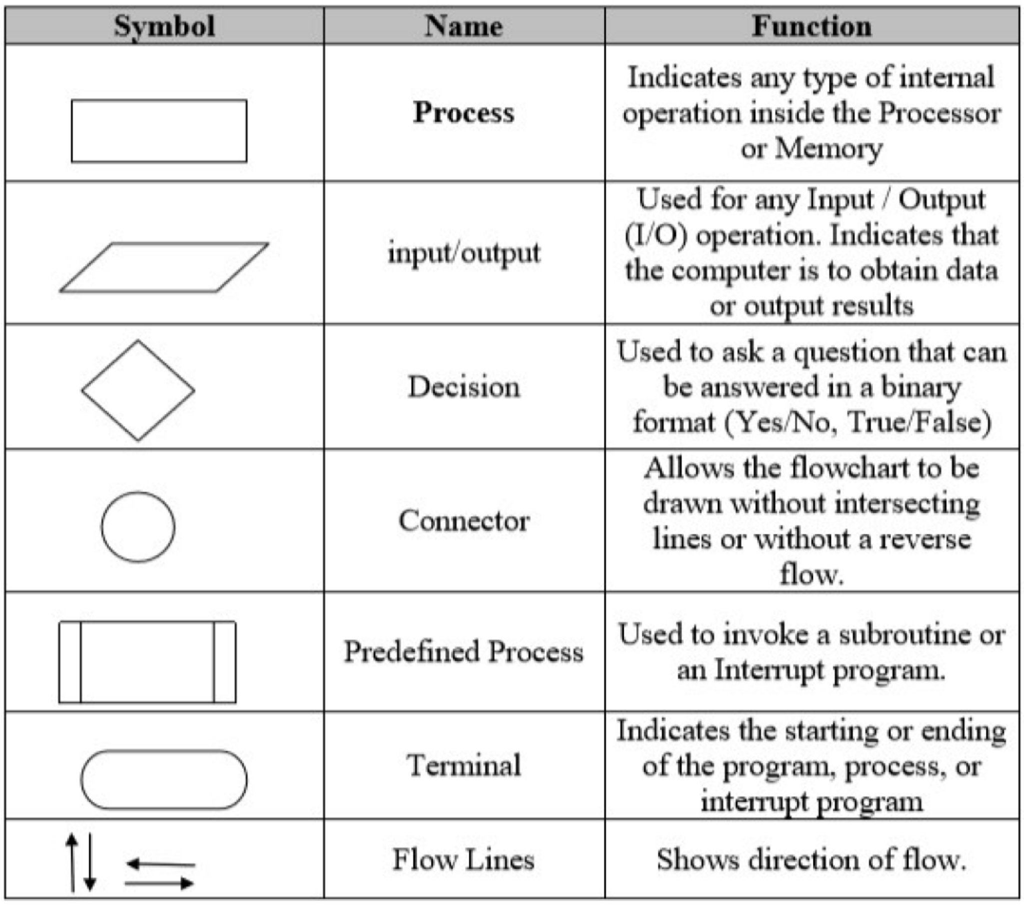
1. Introductory examples of the design and analysis of algorithms

1.1 Flow chart – Notations:

There are 6 basic symbols commonly used in flowcharts:

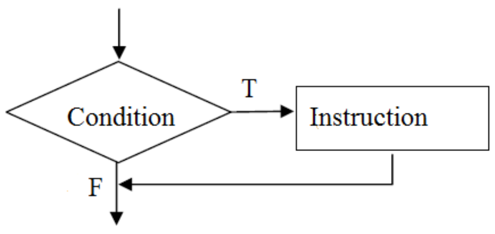
- Process
- input/output
- Decision
- Connector
- Predefined Process
- Terminal



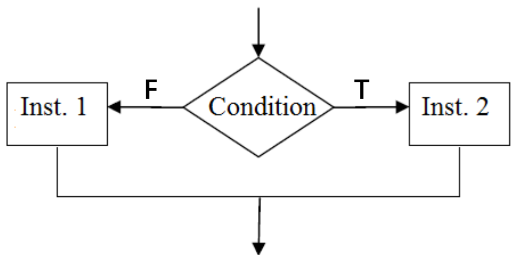
Symbol	Name	Function
▭	**Process**	Indicates any type of internal operation inside the Processor or Memory
▱	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
◇	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
○	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
◫	Predefined Process	Used to invoke a subroutine or an Interrupt program.
⬭	Terminal	Indicates the starting or ending of the program, process, or interrupt program
↑↓ ⇄	Flow Lines	Shows direction of flow.

Notice:
1. " = " typically denotes the equality operation, while " **:=** " represents the assignment operation.
2. We never mention any keyword as "if/while" in the decision/condition box, it only shows a logical expression.

- Program structures - **Conditional branching:**

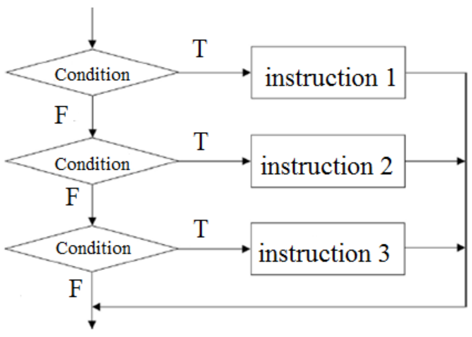


Single branch



Double branch

- Program structures - **Embedded condition:**

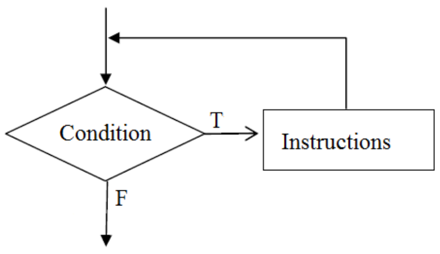


multiple branching

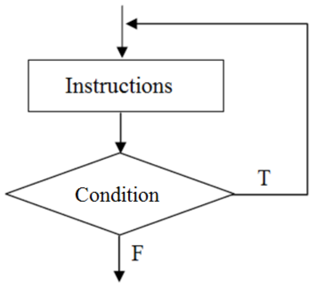
If {…} else if {…} else if {…} else {…}

- Program structures - **Pre-test loop:**

Post-test loop:

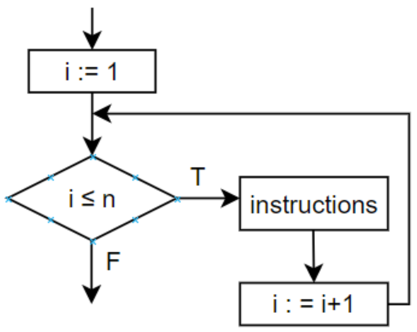


while() { … }



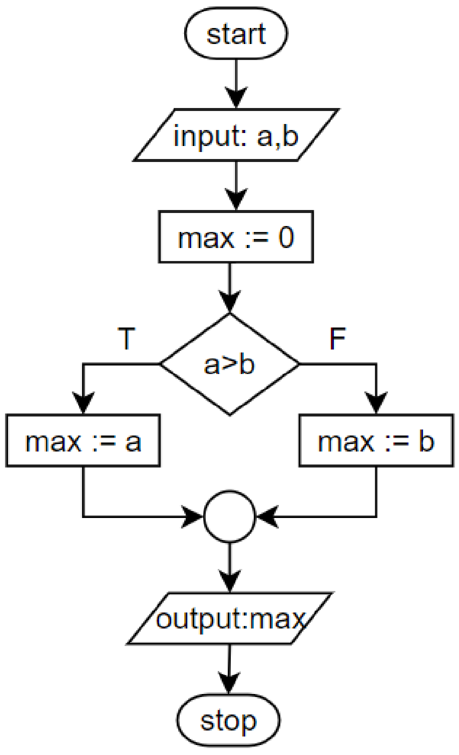
do{ … }while

- Program structures - **For loop:**



1.2 Flow chart – Example:

Draw a flow chart to get the maximum of two numbers.



1.3 Flow chart – Exercises:

Write the algorithm of the following exercises using the following language of description: **Flow chart**

i. Absolute value of the given number
ii. The sum of the first N numbers
iii. The factorial of the first N numbers
iv. The maximum of three numbers

2.1 Pseudocode – Notations:

Keywords:

- START, BEGIN, END, STOP
- DO, WHILE, DO WHILE, FOR, UNTIL, DO UNTIL, REPEAT, END WHILE, END UNTIL, END REPEAT
- IF THEN, IF, ELSE, IF ELSE, END IF, THEN, ELSE THEN, ELSE IF, SO, CASE
- EQUAL, LT, LE, GT, GE, NOT, TRUE, FALSE, AND, OR, XOR
- GET, WRITE, PUT, UPDATE, CLOSE, OPEN, CREATE, DELETE, EXIT, FILE, READ, EOF, EOT, WITH, RETURN

- **Selection:**

```
if condition then
    instructions
[ else if condition then
    Instructions ] . . .
[ else
    Instructions ]
end if
```

- **Iterations – loops:**
  - For loop:

```
for loop variable    //initial value to end value do
    instructions
end for
```

  - Pre-test loop:

```
while condition do
    instructions
end while
```

  - Post-test loop:

```
repeat                          do
    instructions                    instructions
until condition                 while condition
```

- **Function and procedure:**

```
function Function_name[(parameters )]          procedure Procedure_name [(parameters )]
    function body/instructions                     procedure body/instructions
end function                                   end procedure
```

- **Arrays:** Array is a container which can hold a fix number of items and these items should be of the same type. Following are the important terms to understand the concept of array:
  - Element - Each item stored in an array is called an element.
  - Index - Each location of an element in an array has a numerical index, which is used to identify the element. **In pseudocode - array index starts from 1 to length**.

- **Pseudocode conventions**

  Example:

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1..j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

1) Indentation indicates block structure.

2) Use the keyword ***to*** when a for loop increments its loop counter in each iteration, and we use the keyword ***downto*** when a for loop decrements its loop counter.

3) Symbol “**//**” indicates comments.

4) A multiple assignment of the form ***i=j=e*** assigns to both variables i and j the value of expression

5) Variables (such as i, j, and key) are **local** to the given procedure. We shall not use global variables without explicit indication.

6) **A[i]** indicates the **i** *th* element of the array A.

7) We typically organize compound data into objects, which are composed of attributes. Example of accessing a particular attribute of a object: ***A.length*** (specify the number of elements in an array A).

8) We pass parameters to a procedure **by value**.

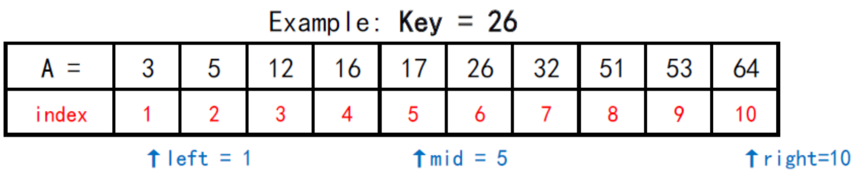
9) We allow multiple values to be returned in a single **return** statement.

2.2 Pseudocode – Example:

Write pseudocodes to get the maximum of two numbers.

```
INPUT: a, b
IF a>b THEN
    max=a
ELSE
    max=b
END IF
OUTPUT: max
```

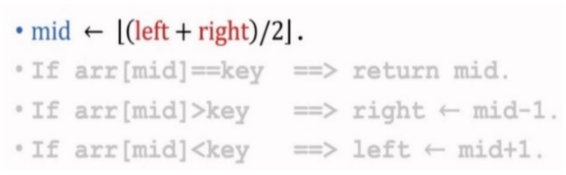
2.3 Pseudocode for Binary Search



Example: Key = 26

A =	3	5	12	16	17	26	32	51	53	64
index	1	2	3	4	5	6	7	8	9	10

↑left = 1 ↑mid = 5 ↑right=10



- mid ← ⌊(left + right)/2⌋.
- If arr[mid]==key ==> return mid.
- If arr[mid]>key ==> right ← mid-1.
- If arr[mid]<key ==> left ← mid+1.

- **Recursive binary search:**

```
FUNCTION binary_search(A, left, right, key)
  IF left > right
    RETURN NIL
  ELSE
    mid = floor((left + right) / 2)
    IF A[mid] == key
      RETURN mid
    ELSE IF A[mid] > key
      RETURN binary_search(A, left, mid-1, key)
    ELSE
      RETURN binary_search(A, mid+1, right, key)
    END IF
  END IF
END FUNCTION
```

- **Iterative binary search:**

```
FUNCTION binary_search(A, key)
  left = 1
  right = length of A
  WHILE left <= right
    mid = floor((left + right) / 2)
    IF A[mid] == key
      RETURN mid
    ELSE IF A[mid] > key
      right = mid - 1
    ELSE
      left = mid + 1
    END IF
  END WHILE
  RETURN NIL
END FUNCTION
```

Recursive and Iterative are two different approaches in programming to solve problems, which differ in their implementation.

Recursive is a method to solve problems by calling itself within a function. Recursive functions usually have two cases: base case and recursive case. The base case is where the input reaches a specific value, and the function directly returns a result. The recursive case is where the input has not reached the base case yet, and the function calls itself to handle smaller inputs until it reaches the base case. Recursion often makes the code look more concise but can lead to lower efficiency or stack overflow issues.

Iterative, on the other hand, **is a method to solve problems using loops instead of calling itself.** Iteration usually requires some loop variables to track the progress of the processing and typically uses a while or for loop. Iteration often has higher efficiency but can result in more complex code.

In binary search, **recursive** implementation involves dividing the array to be searched into two sub-arrays, and recursively searching the left and right sub-arrays until the target element is found or determined to not exist. The recursive implementation may be more concise and easier to understand, but may also require additional stack space and may result in stack overflow issues.

On the other hand, **iterative** implementation uses loops to perform binary search. Two pointers are initially set to the first and last elements of the array. In each iteration, the middle index is calculated, and if the target element is less than the element at the middle index, the right pointer is moved to the left of the middle index, and if the target element is greater, the left pointer is moved to the right of the middle index. If the target element is found at the middle index, the search is completed. The loop continues until the left pointer is greater than the right pointer, indicating that the target element does not exist. The iterative implementation may be more complex, but requires less memory and is generally faster than the recursive implementation.

2.4 Pseudocode – Exercises:

Write the algorithm of the following exercises using the following language of description: **Pseudocode**

i. Absolute value of the given number
ii. The sum of the first N numbers
iii. The factorial of the first N numbers
iv. Maximum of three numbers
v. Reverse an integer
vi. Find the greatest common divisor of integers a and b (a>b>0)
vii. Check if a number is prime
viii. Calculate the sum of even numbers between two given numbers
ix. Find the maximum number in an array
x. Check if a given string is a palindrome