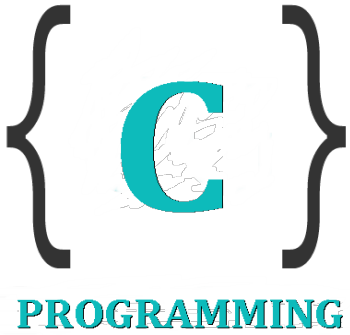
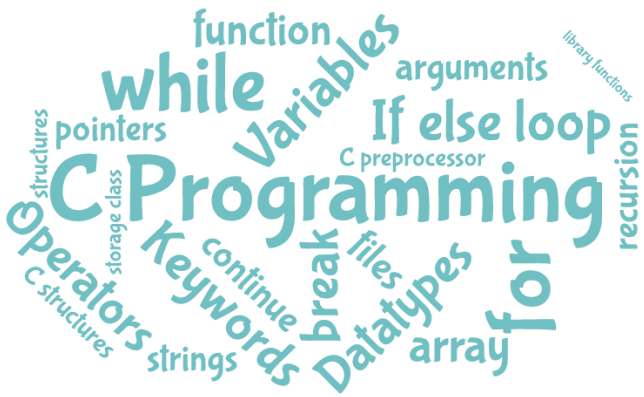



# IT 112: Introduction to Programming




Dr. Manish Khare
Dr. Bakul Gohel

- This course is to teach you how to solve problems using a computer.

- Programming nowadays is considered a basic skill similar to mathematics that is needed across all disciplines like engineering, in the sciences, and nowadays even in the arts.

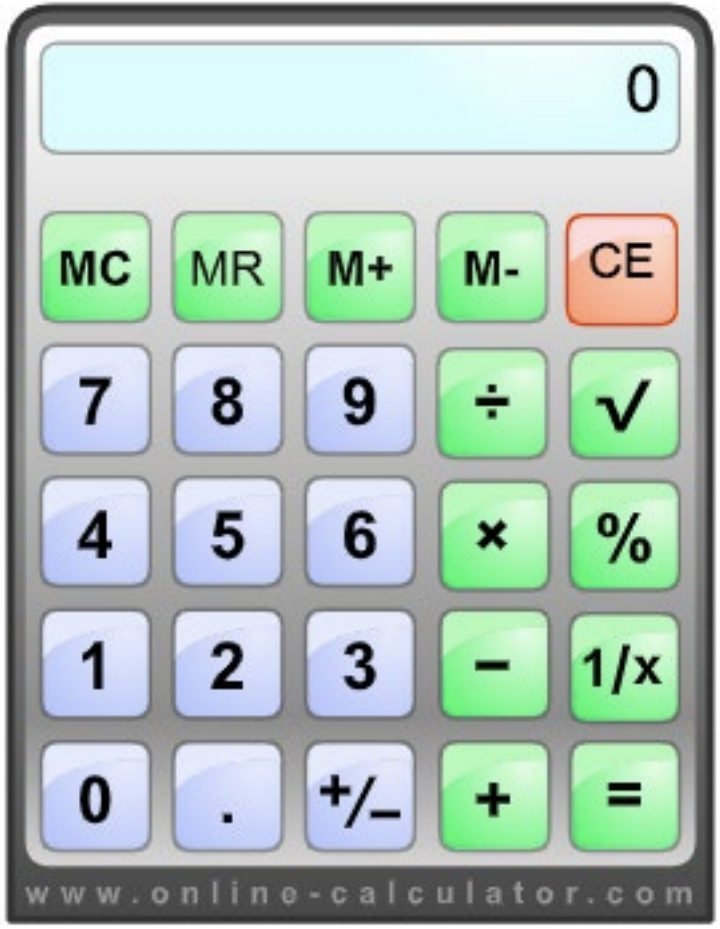
- we do not assume any prior experience in programming whether in C or in any other language. So, the focus will be to start from the basics.

# Terms to Understand

- **Calculate**
  - To determine by mathematical process.

- **Compute**
  - To determine, especially by mathematical means.

# Calculator

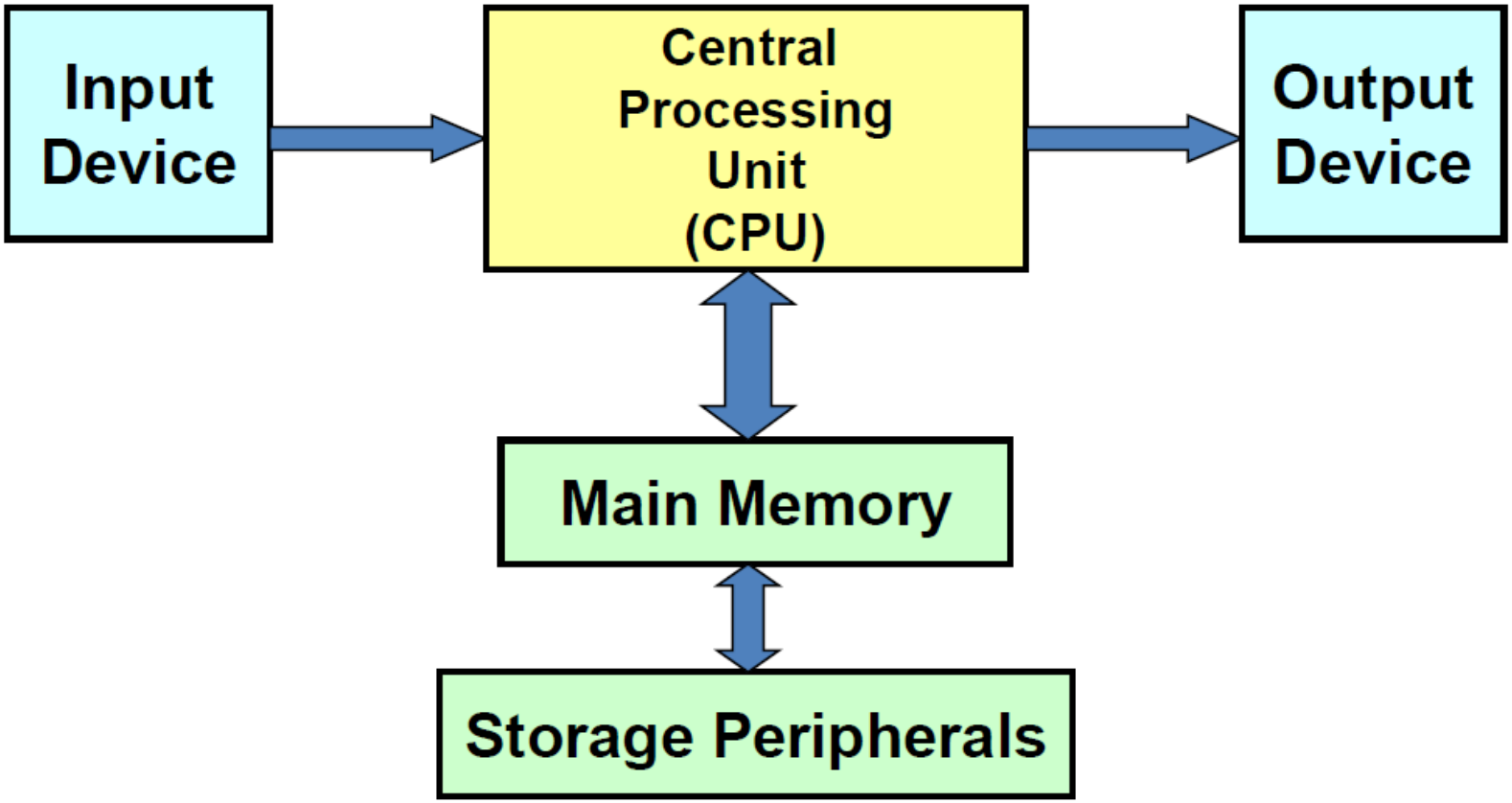


# What is Computer?

A computer, in simple terms, can be defined as an electronic device that is designed to accept data, perform the required mathematical and logical operations at high speed, and output the result. A computer accepts data, processes it, and produces information.

# What is Computer?

It is a machine which can accept data, process them, and output results.



- A digital computer is built out of tiny electronic switches.
  - From the viewpoint of ease of manufacturing and reliability, such switches can be in one of two states, ON and OFF.
  - A switch can represent a digit in the so-called *binary number system*, 0 and 1.

- A computer works based on the binary number system.

# Concept of Bits and Bytes

- Bit
  - A single binary digit (0 or 1).
- Nibble
  - A collection of four bits (say, 0110).
- Byte
  - A collection of eight bits (say, 01000111).
- Word
  - Depends on the computer.
  - Typically 4 or 8 bytes (that is, 32 or 64 bits).

# Computer Programming

- The computer hardware cannot think and make decisions on its own. So, it cannot be used to analyze a given set of data and find a solution on its own. The hardware needs a software (a set of programs) to instruct what has to be done. A program is a set of instructions that is arranged in a sequence to guide a computer to find a solution for the given problem. The process of writing a program is called *programming*.

# Computer Programming

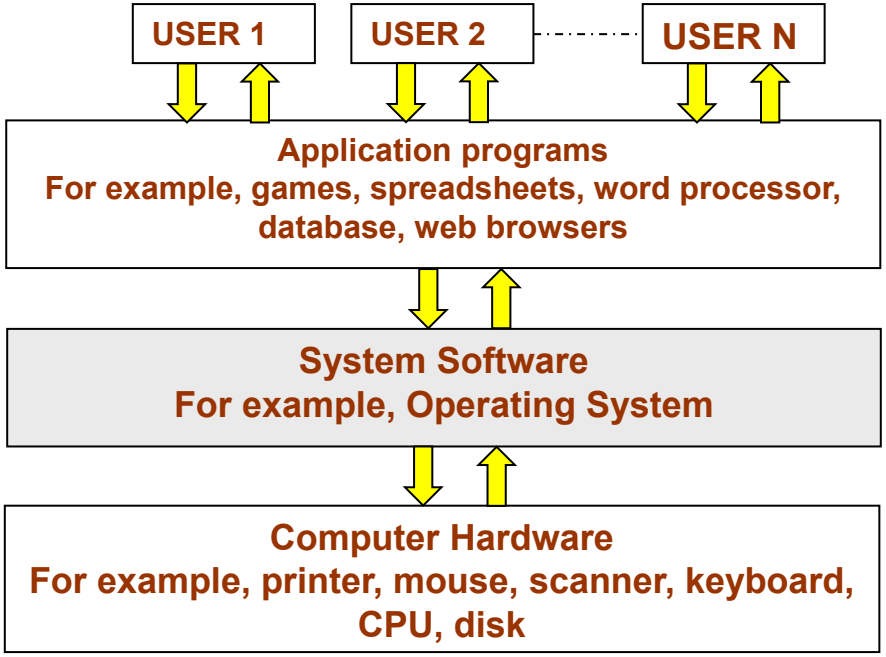
- Computer software is written by computer programmers using a programming language.
- The programmer writes a set of instructions (program) using a specific programming language. Such instructions are known as the s*ource code*.
- Another computer program called a *compiler* is then used on the source code, to transform the instructions into a language that the computer can understand. The result is an executable computer program, which is another name for software.
- Examples of computer software include:
  - *Computer Games*
  - *Driver Software*
  - *Educational Software*
  - *Media Players and Media Development Software*
  - *Productivity Software*
  - *Operating Systems* software

# Classification of Computer Software

- Computer software can be broadly classified into two groups: system software and application software.
- Application software is designed to solve a particular problem for users. It is generally what we think of when we say the word computer programs. Examples of application software include spreadsheets, database systems, desktop publishing systems, program development software, games, web browser, so on and so forth. Simply put, application software represents programs that allow users to do something besides simply run the hardware.

- On the contrary, system software provides a general programming environment in which programmers can create specific applications to suit their needs. This environment provides new functions that are not available at the hardware level and performs tasks related to executing the application program. System software represents programs that allow the hardware to run properly.

# Classification of Computer Software



# Compiler and Interpreter

- A **compiler** is a special type of program that transforms source code written in a programming language (the *source language*) into machine language comprising of just two digits- 1s and 0s (the *target language)*. The resultant code in 1s and 0s is known as the object *code*. The object code is the one which will be used to create an executable program.

- If the source code contains errors then the compiler will not be able to its intended task. Errors that limit the compiler in understanding a program are called *syntax errors*. Syntax errors are like spelling mistakes, typing mistakes, etc. Another type of error is logic error which occurs when the program does not function accurately. Logic errors are much harder to locate and correct.

# Compiler and Interpreter

- **Interpreter:** Like the compiler, the interpreter also executes instructions written in a high-level language.

- While the compiler translates instructions written in high level programming language directly into the machine language; the interpreter on the other hand, translates the instructions into an intermediate form, which it then executes.

- Usually, a compiled program executes faster than an interpreted program. However, the big advantage of an interpreter is that it does not need to go through the compilation stage during which machine instructions are generated. This process can be time-consuming if the program is long. Moreover, the interpreter can immediately execute high-level programs.

# Linker and Loader

- **Linker:** Also called *link editor* and *binder,* a linker is a program that combines object modules to form an executable program.

- Generally, in case of a large program, the programmers prefer to break a code into smaller modules as this simplifies the programming task. Eventually, when the source code of all the modules has been converted into object code, you need to put all the modules together. This is the job of the linker. Usually, the compiler automatically invokes the linker as the last step in compiling a program.

- **Loader:** A loader is a special type of program that copies programs from a storage device to main memory, where they can be executed. Most loaders are transparent to the users.

# Application Software

- Application software is a type of computer software that employs the capabilities of a computer directly to perform a user-defined task. This is in contrast with system software which is involved in integrating a computer's capabilities, but typically does not directly apply them in the performance of tasks that benefit the user.

- To better understand application software consider an analogy where hardware would depict the relationship of an electric light bulb (an application) to an electric power generation plant (a system).

- Typical examples of software applications are word processors, spreadsheets, media players, education software, CAD, CAM, data communication software, statistical and operational research software, etc. Multiple applications bundled together as a package are sometimes referred to as an application suite.

# Stored Program Concept

All digital computers are based on the principle of stored program concept, which was introduced by Sir John von Neumann in the late 1940s. The following are the key characteristic features of this concept:

- Before any data is processed, instructions are read into memory.
- Instructions are stored in the computer's memory for execution.
- Instructions are stored in binary form (using binary numbers—only 0s and 1s).
- Processing starts with the first instruction in the program, which is copied into a control unit circuit. The control unit executes the instructions.
- Instructions written by the users are performed sequentially until there is a break in the current flow.

# Stored Program Concept

- Input/Output and processing operations are performed simultaneously. While data is being read/written, the central processing unit (CPU) executes another program in the memory that is ready for execution.

**Note:** A stored program architecture is a fundamental computer architecture wherein the computer executes the instructions that are stored in its memory.

# Programming Languages

- A **programming language** is a language specifically designed to express computations that can be performed the computer. Programming languages are used to express algorithms or as a mode of human communication.

- While high-level programming languages are easy for the humans to read and understand, the computer actually understands the machine language that consists of numbers only.

- In between the machine languages and high-level languages, there is another type of language known as assembly language. Assembly languages are similar to machine languages, but they are much easier to program in because they allow a programmer to substitute names for numbers.

- However, irrespective of what language the programmer use, the program written using any programming languages has to be converted into machine language so that the computer can understand it. There are two ways to do this: compile the program or i*nterpret* the program.

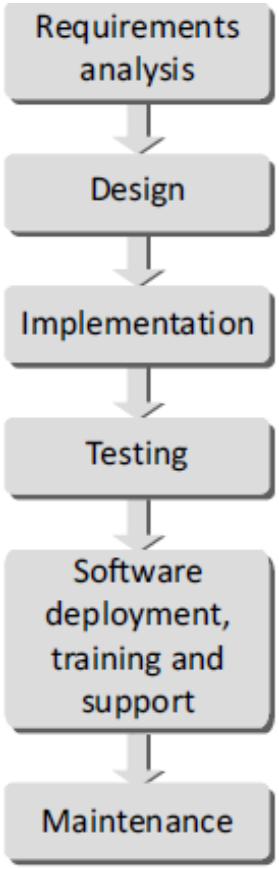
# Programming Languages

- The question of which language is best depends on the following factors:
- The type of computer on which the program has to be executed
- The type of program
- The expertise of the programmer
- For ex, FORTRAN is a good language for processing numerical data, but it does not lend itself very well to organizing large programs. Pascal can be used for writing well-structured and readable programs, but it is not as flexible as the C programming language. C++ goes one step ahead of C by incorporating powerful object-oriented features, but it is complex and difficult to learn.

# Design and Implementation of Efficient Programs

- The design and development of correct, efficient, and maintainable programs depends on the approach adopted by the programmer to perform various activities that need to be performed during the development process.
- The entire program or software (collection of programs) development process is divided into a number of phases where each phase performs a well-defined task. Moreover, the output of one phase provides the input for its subsequent phase.

# Phases in Software Development Life Cycle




Phases in software development life cycle

# Program Design Tools: Algorithms, Flowcharts, Pseudocodes

- In general terms, an algorithm provides a blueprint to writing a program to solve a particular problem. It is considered to be an effective procedure for solving a problem in a finite number of steps. That is, a well-defined algorithm always provides an answer, and is guaranteed to terminate.
- A flowchart is a graphical or symbolic representation of a process. It is basically used to design and document virtually complex processes to help the viewers to visualize the logic of the process, so that they can gain a better understanding of the process and find flaws, bottlenecks, and other less obvious features within it. When designing a flowchart, each step in the process is depicted by a different symbol and is associated with a short description.

# Program Design Tools: Algorithms, Flowcharts, Pseudocodes

- Pseudocode is a compact and informal high-level description of an algorithm that uses the structural conventions of a programming language.

# How does a computer work?

- Stored program concept.
  - Main difference from a calculator.
- What is a program?
  - Set of instructions for carrying out a specific task.
- Where are programs stored?
  - In secondary memory, when first created.
  - Brought into main memory, during execution.

# Some Terminologies

- Algorithm / Flowchart
  - A step-by-step procedure for solving a particular problem.
  - Should be independent of the programming language.

- Program
  - A translation of the algorithm/flowchart into a form that can be processed by a computer.
  - Typically written in a high-level language like C, C++, Java, etc.

# Problem Solving

- Step 1:
  - Clearly specify the problem to be solved.
- Step 2:
  - Draw flowchart or write algorithm.
- Step 3:
  - Convert flowchart (algorithm) into program code.
- Step 4:
  - Compile the program into object code.
- Step 5:
  - Execute the program.

# Some Terminology

- **Programming**
  - Instructing the computer about how to do a certain task
- **Program**
  - The set of instructions
- **Programmability**
  - The ability of the computer to accept a set of instructions and carry them out.

*Programmability* is what *primarily distinguishes a computer from a calculator.*

# Process of Programming

1. Define and model the problem.

2. Obtain logical solution to your problem
   - A logical solution is a finite and clear step by step procedure to solve your problem.
   - Also called as algorithm.

# Algorithms and Programs

- **An algorithm is also a sequence of steps to solve a problem**
  - But usually written in a natural language (such as English), precisely enough so as to be unambiguously understood by humans.

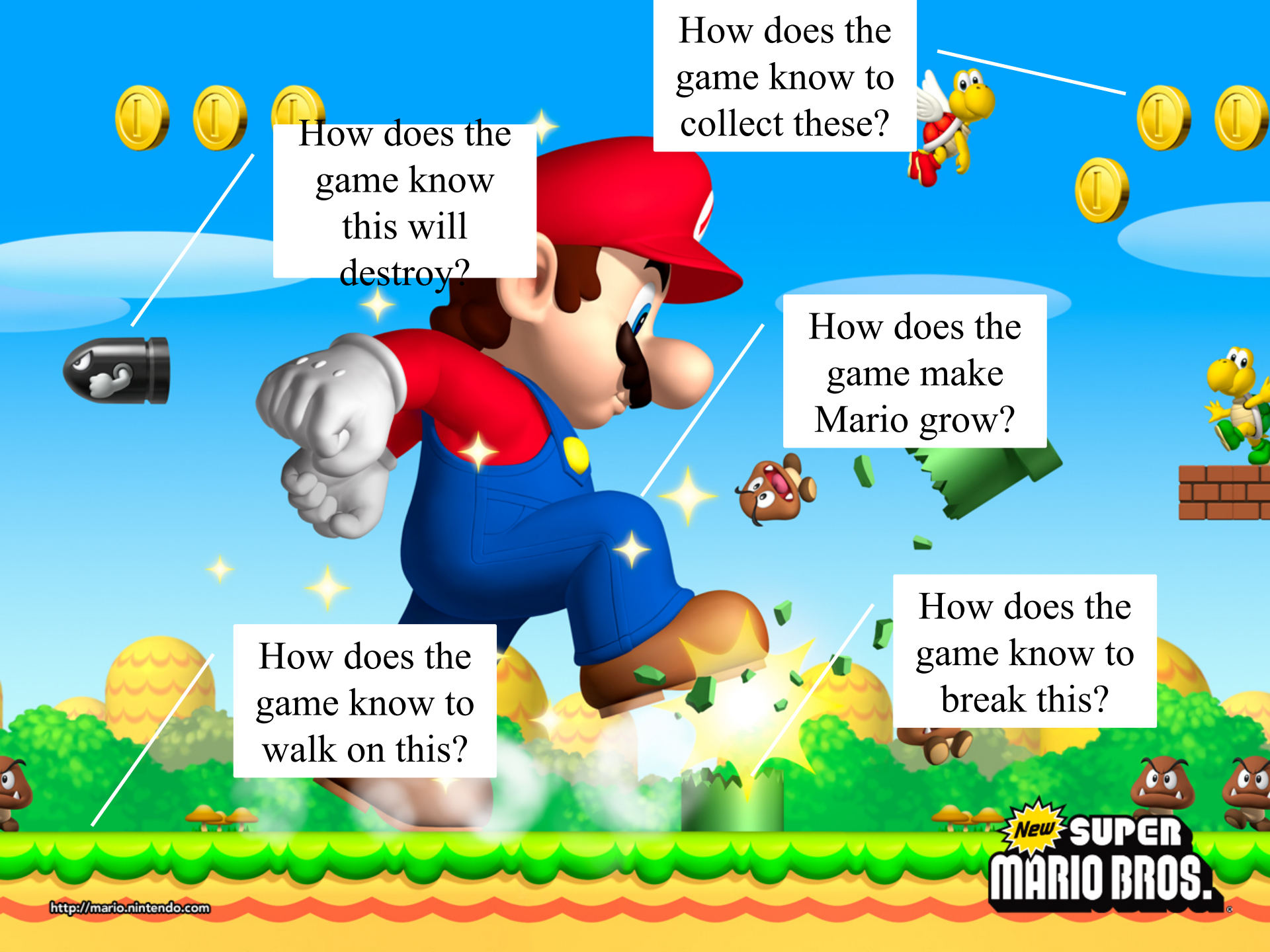
# Algorithm to compute n!

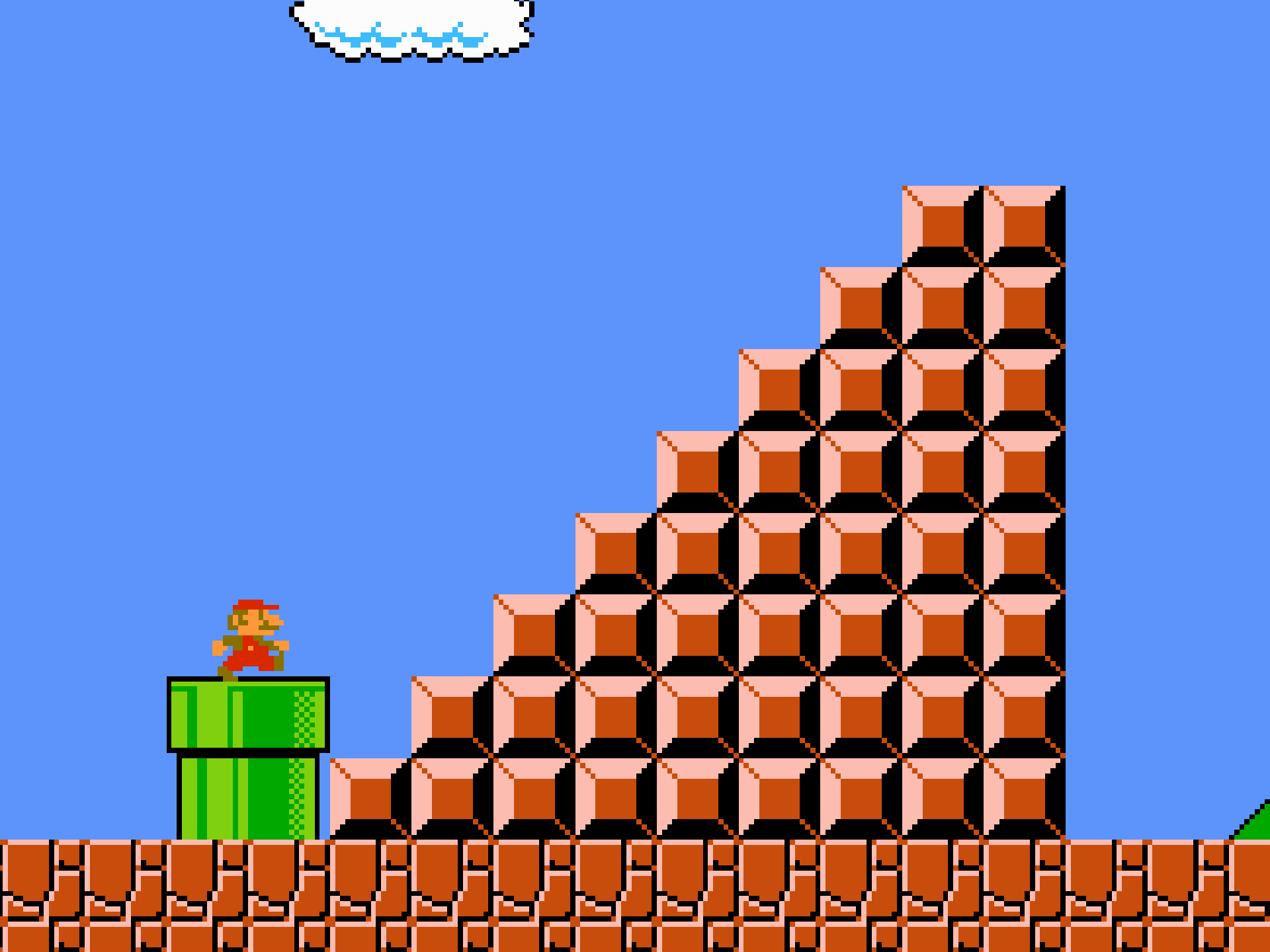
1. Read the value of *n*.
2. Set the value of *result* to 1
3. While n>1 do (steps 3.1 and 3.2)
    1. Set new value of result to (old value of result times the value of n).
    2. Set new value of n to (old value of n-1)
4. Print the value of result

# Algorithm in our Real-Life

- An algorithm is a very familiar concept - the most important example that you can think of are cooking recipes.
  - Ingredients
  - Instructions

A computer is a blank canvas
Waiting for you to instruct.
Plan out with pseudocode
Then program to construct.


How does the game know to collect these?
How does the game know this will destroy?
How does the game make Mario grow?
How does the game know to walk on this?
How does the game know to break this?
New SUPER MARIO BROS.
http://mario.nintendo.com



# Tower of Hanoi

# How do I get Minion Stuart to move to D1?

| | A | B | C | D |
|---|---|---|---|---|
| 1 |  | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |

Move right 3 squares

# How do I get Minion Stuart to move to D1 then to D4?

| | A | B | C | D |
|---|---|---|---|---|
| 1 |  | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |

Move right 3 squares
Move down 3 squares

# How do I get Minion Stuart to move to D1 then to D4, then move to A4 and finally to A1?

| | A | B | C | D |
|---|---|---|---|---|
| 1 |  | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |

Move right 3 squares
Move down 3 squares
Move left 3 squares
Move up 3 squares

# What have we just done

- You have created a series of instructions to solve a given problem
- This is called an **Algorithm**
- When we write it in a list of instructions it is called **Pseudocode**
- Computer Programmers use pseudocode to help plan out the code they will need.
  - For game making
  - Creating websites
  - Control software – robots / machinery
  - ANYTHING where planning is needed = pseudocode is used to layout the tasks/actions

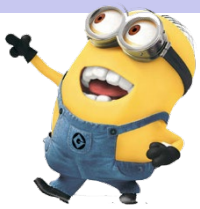
In your workbooks I want you to think about how to get Minion Stuart to move around the squares and collect the bananas. You will need to move him and also add actions needed to pick up the items.

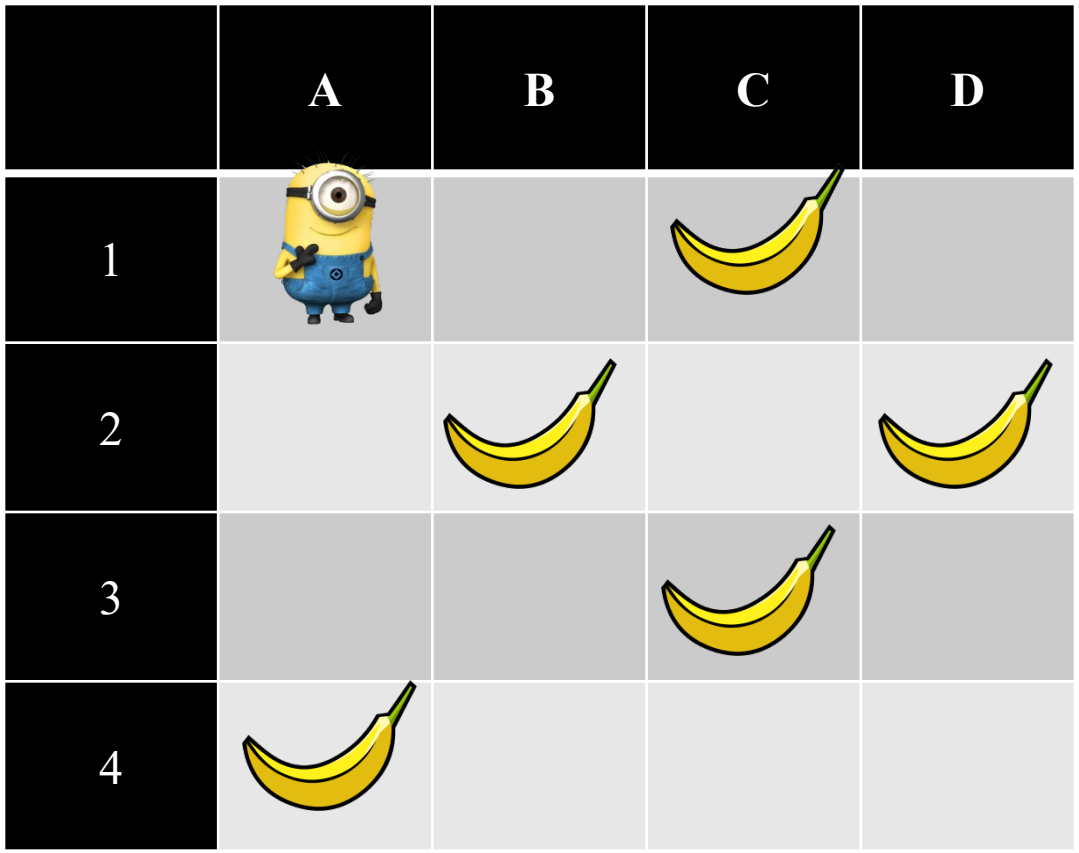
| | A | B | C | D |
|---|---|---|---|---|
| 1 |  | |  | |
| 2 | |  | |  |
| 3 | | |  | |
| 4 |  | | | |

## Your Turn

Extension = Can you add a 'question to say if the minion reaches a banana?

# Algorithm Solution written in pseudocode






1. Move right 2 squares
2. Pick up banana
3. Move right 1 square
4. Move down 1 square
5. Pick up banana
6. Move left 2 squares
7. Pick up banana
8. Move down 1 square
9. Move right 1 square
10. Pick up banana
11. Move down 1 square
12. Move left 2 squares
13. Pick up banana

# Can you ask a question?

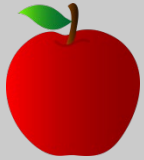
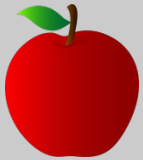
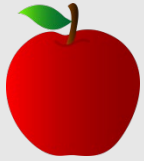
- When we think about the way Minion Stuart moves across could we ask a question as he moves from one side to the other?

1. Move across 3 squares
2. If you reach a banana
   a. Pick it up

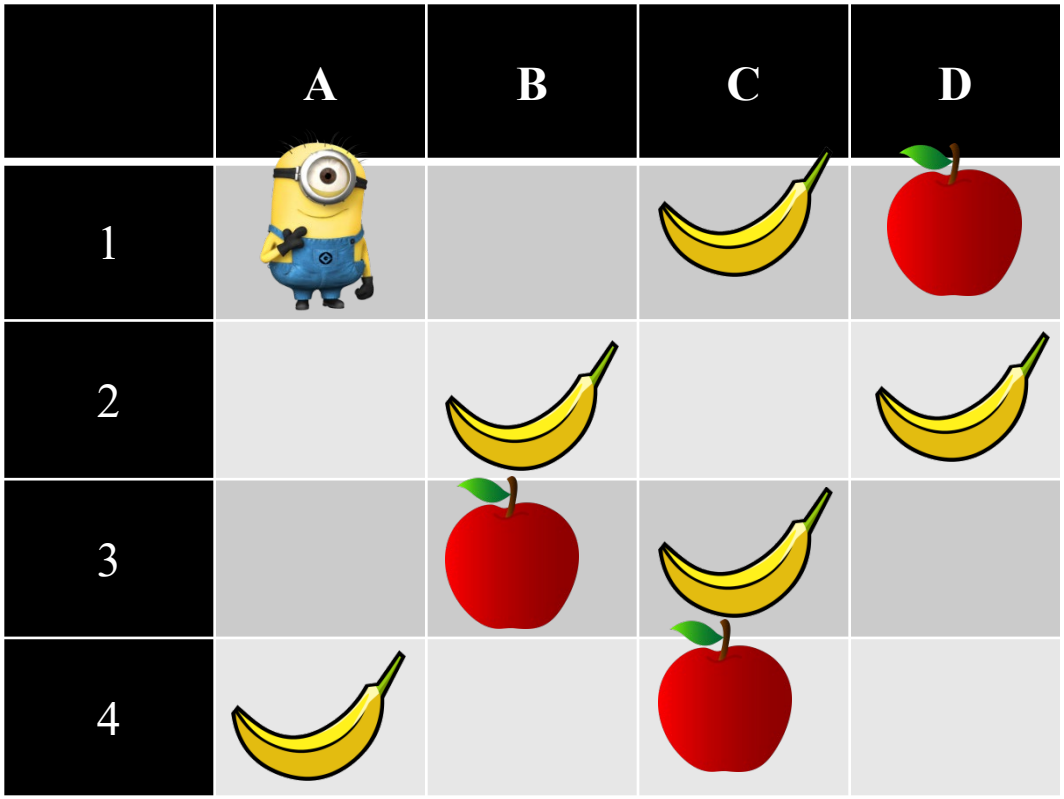
Ask the question – try and start it with an 'IF'

Ask the question – try and start it with an 'IF'

# Can we ask a question?

| | A | B | C | D |
|---|---|---|---|---|
| 1 |  | |  |  |
| 2 | |  | |  |
| 3 | |  |  | |
| 4 |  | |  | |

# Possible algorithm in pseudocode




1. Move right 3 squares
2. IF Minion reaches a banana
    a. THEN pick it up
3. IF Minion reaches an apple
    a. THEN leave it

This would continue to cover the whole board

This form of pseudocode would help a game designer plan out the code they would need to write to create it.

# Pseudocode: Keywords

- **begin ------ end** : begin is the first statement and end is the last statement of the pseudocode. All the instructions used in pseudocode is to be written between begin----end block.
- **Read:** Reading two values from the input given by the user
- **Compute :** The keyword compute is used for calculation of the result of the given expression.
- **Print :** It is used to display the output of the program
- **if--- else--- endif :** It is a decision construct used different action has to be taken depending on the condition.
- **while---- do :** This is also an iterative construct similar to do---while construct. The only difference is that in this construct, the testing statement is present at the top of set of the iterative statements.

# Pseudocode: Example

Write the pseudocode to accept two numbers and displays their sum and difference.

Begin *//starting the pseudocode*
Read first_number, second_number *//input two numbers*
Compute sum as first_number + second_number *// process of adding the given two numbers*
Compute difference as first_number - second_number *//process of calculating difference of the given two numbers*
Print 'The sum of the given two numbers is' sum *//Output to display the sum*
Print 'The difference of the given two numbers is' difference *//Output to display the difference*
End *//End of the pseudocode*

# Decision Structures

- The expression A>B is a logical expression
- *it describes a **condition** we want to test*
- ***if A>B is true (if A is greater than B)** we take the action on left*
- print the value of A
- ***if A>B is false (if A is not greater than B)** we take the action on right*
- print the value of B

# if-then-else Structure

- The structure is as follows

***If condition then***

***true alternative***

***else***

***false alternative***

***endif***

| Relational Operators | |
|---|---|
| **Operator** | **Description** |
| > | **Greater than** |
| < | **Less than** |
| = | **Equal to** |
| ≥ | **Greater than or equal to** |
| ≤ | **Less than or equal to** |
| ≠ | **Not equal to** |

# Nested if Structure

- One of the alternatives within an IF–THEN–ELSE statement
  - may involve further IF–THEN–ELSE statement

# Loop Structure

- For Loop
- While… Do Loop
- Do… While Loop

# Exercise

- pseudocode for **compute the area of a rectangle**

*Begin*

*Get the length, l, and width, w*

*Compute the area = l*w*

*Display the area*

*End*

- pseudocode for **compute the perimeter of a rectangle**:

*Begin*

*Enter length, l*

*Enter width, w*

*Compute Perimeter = 2*l + 2*w*

*Display Perimeter of a rectangle*

*end*

- pseudocode for Calculate Pay – sequence

```
Begin
input hours
input rate
pay = hours * rate
print pay
End
```

## pseudocode for Calculate Pay with Overtime – selection

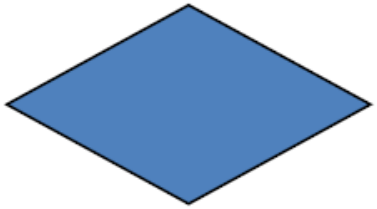
```
Begin
input hours, rate
if hours ≤ 40 then
      pay = hours * rate
else
      pay = 40 * rate + (hours – 40) * rate * 1.5
print pay
End
```

# Flow Chart

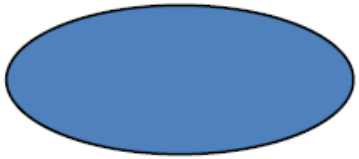
# Flow Chart: Basic Symbol

Computation

Input / Output



Decision Box

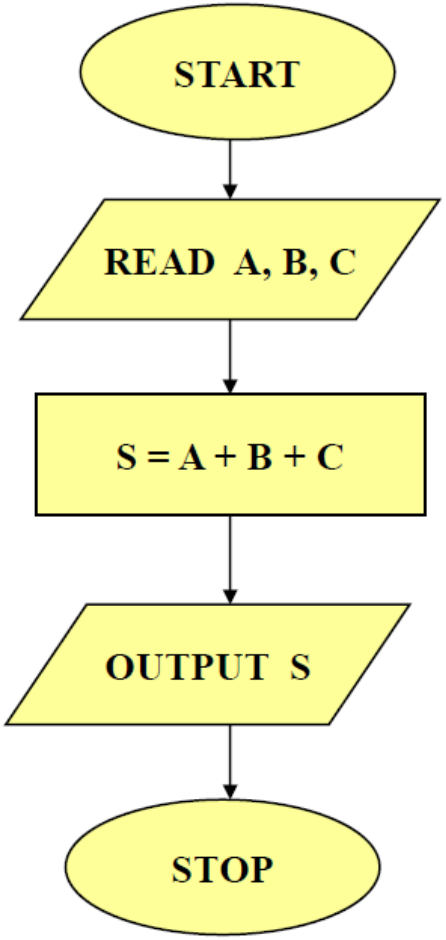


Start / Stop

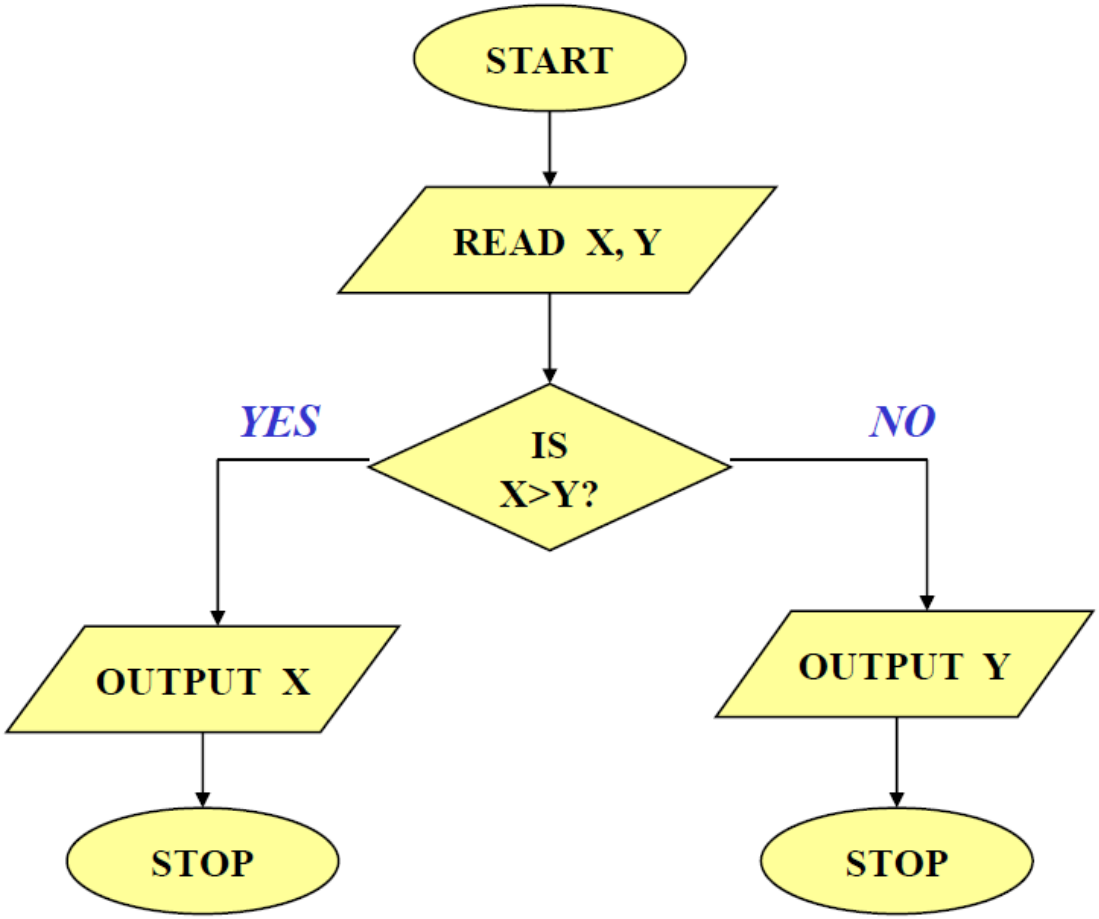
Flow of control

Connector

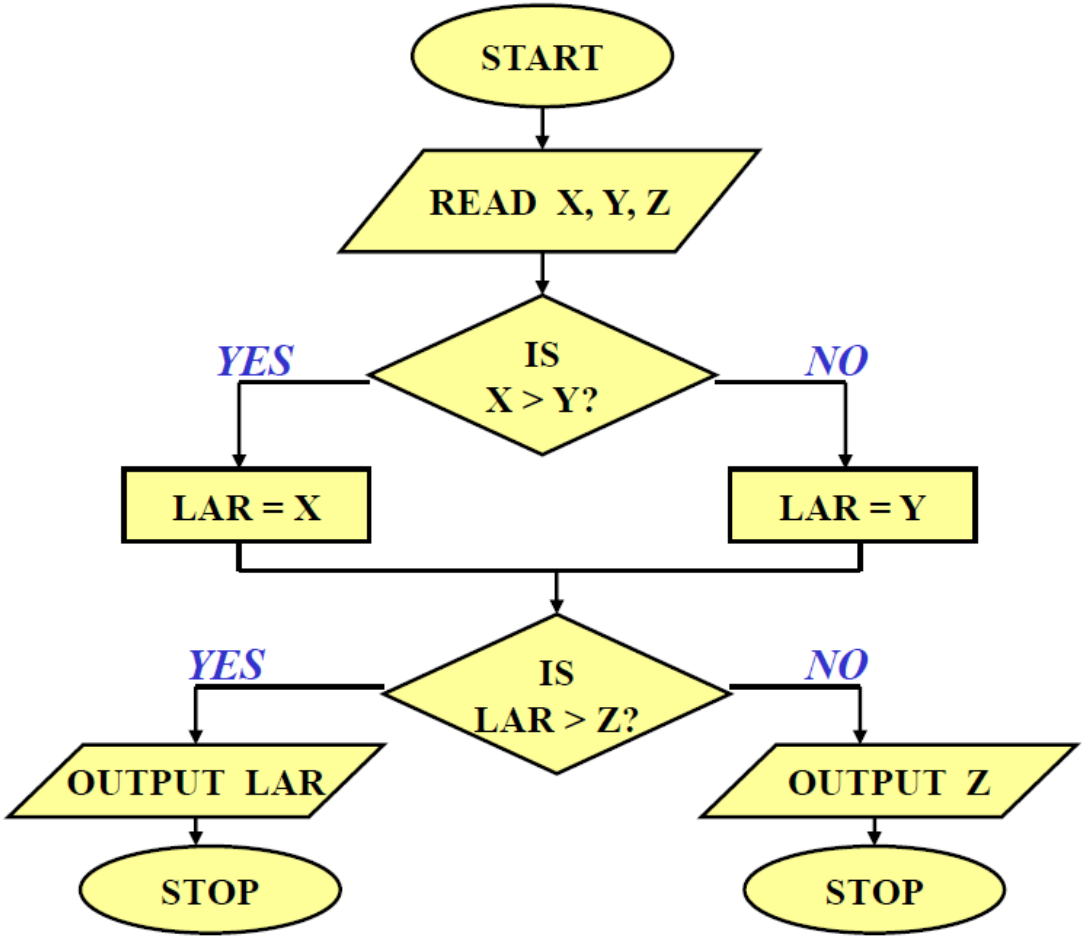
## **Example:** Adding Three Numbers



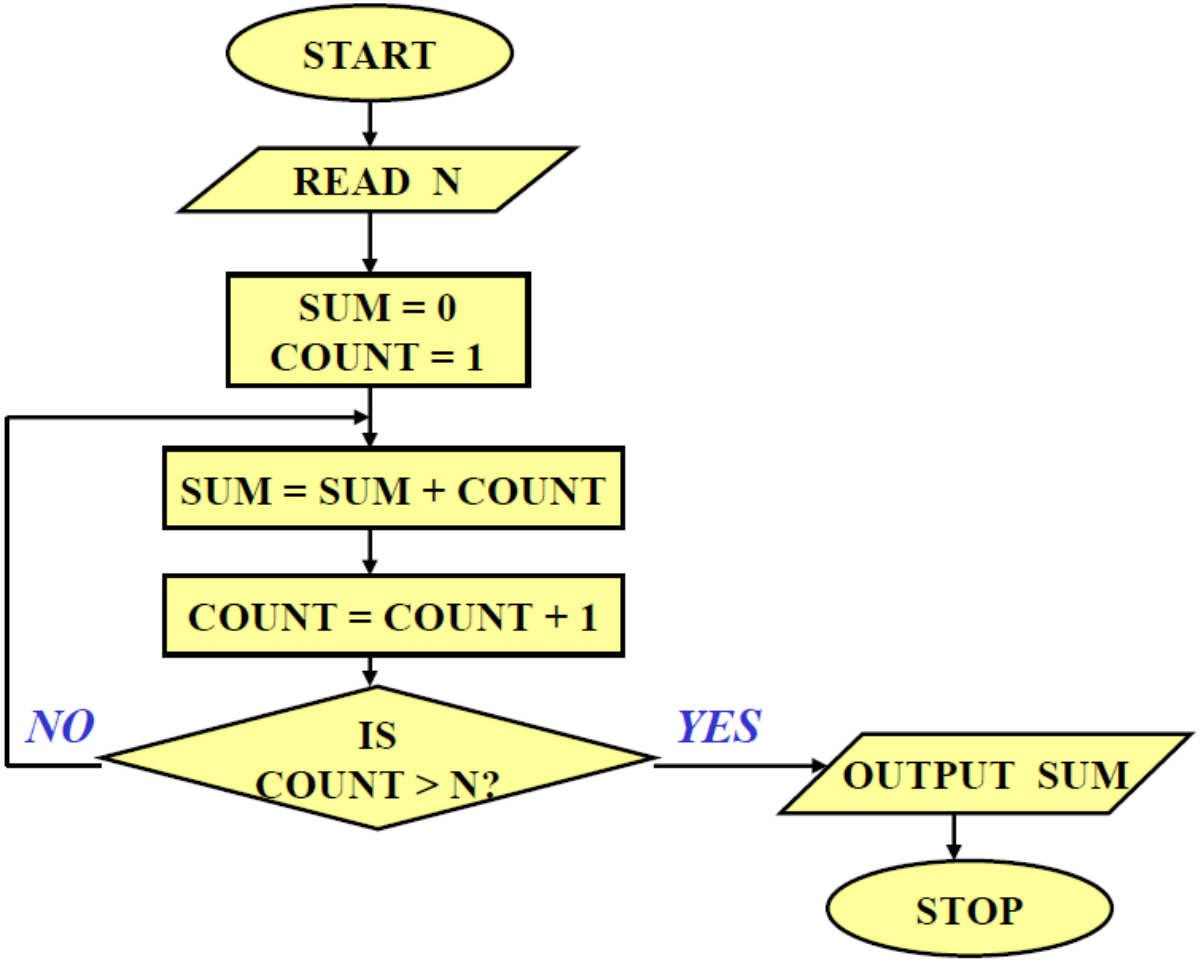
## Example: Larger of two numbers



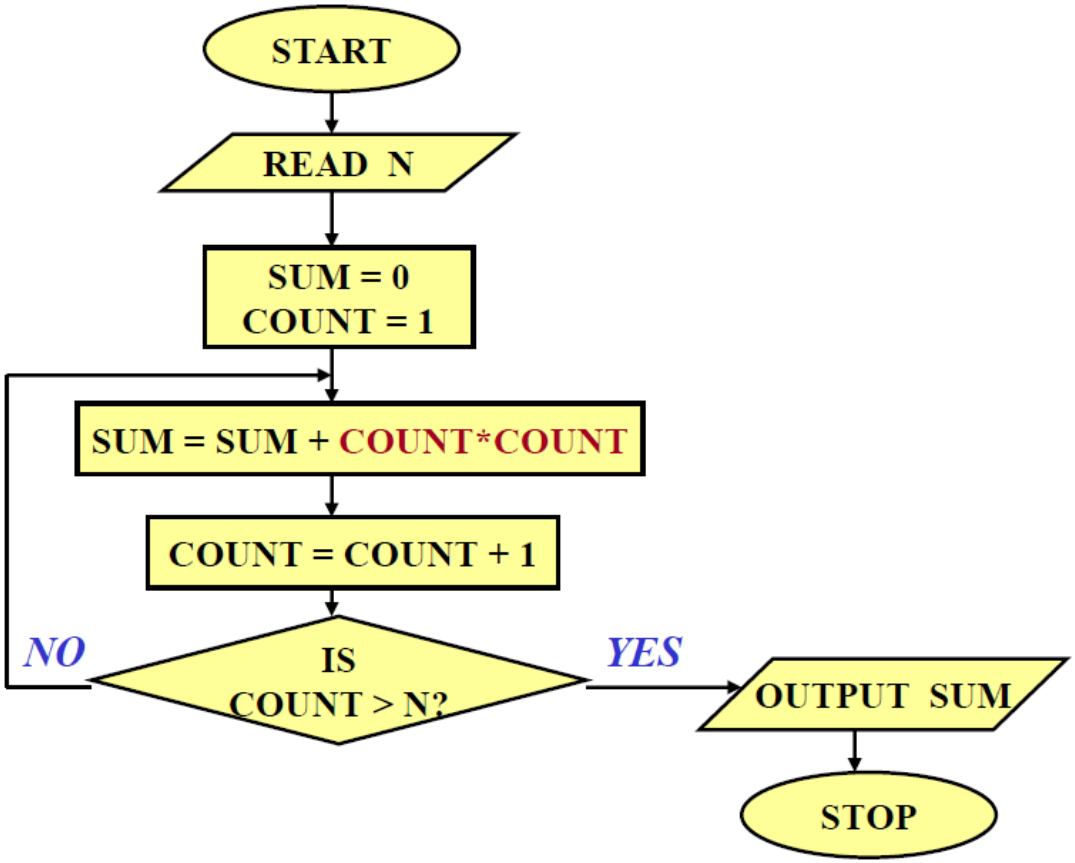
## Example: Largest of three numbers



**Example:** Sum of first N natural numbers

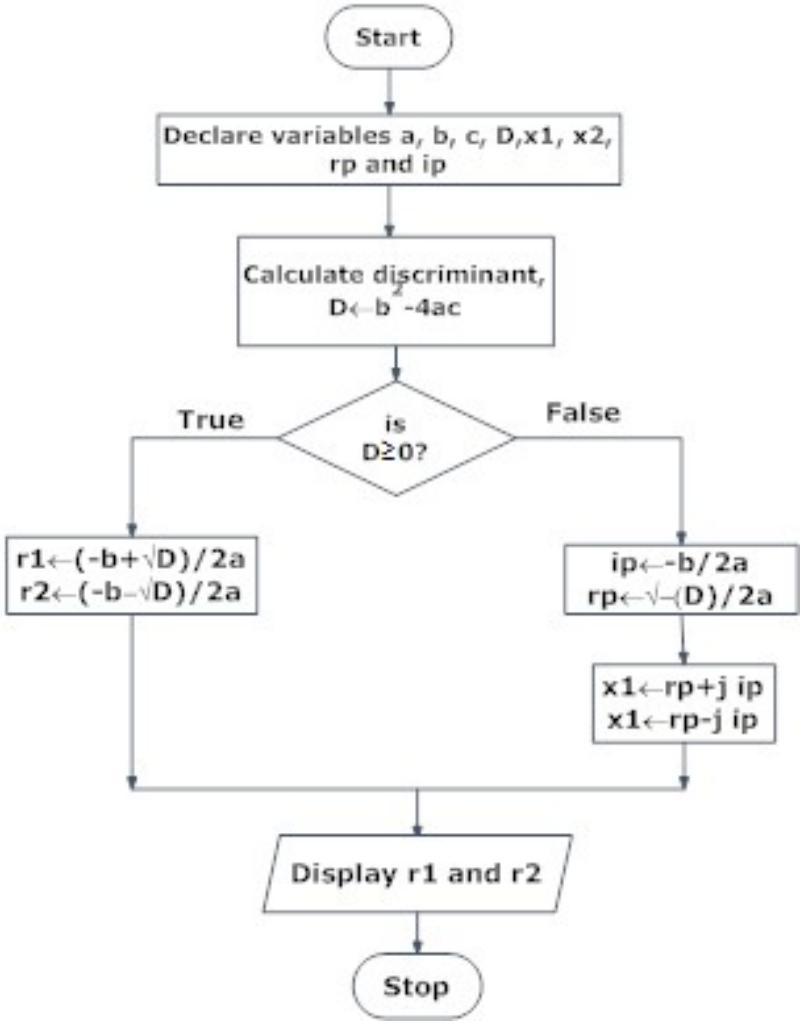


- Example: $SUM = 1^2 + 2^2 + 3^2 + N^2$



Example: *Roots of a quadratic equation*

$$ax^2 + bx + c = 0$$



**Example:** Grade Computation

$MARKS \geq 90$ → Ex

$89 \geq MARKS \geq 80$ → A

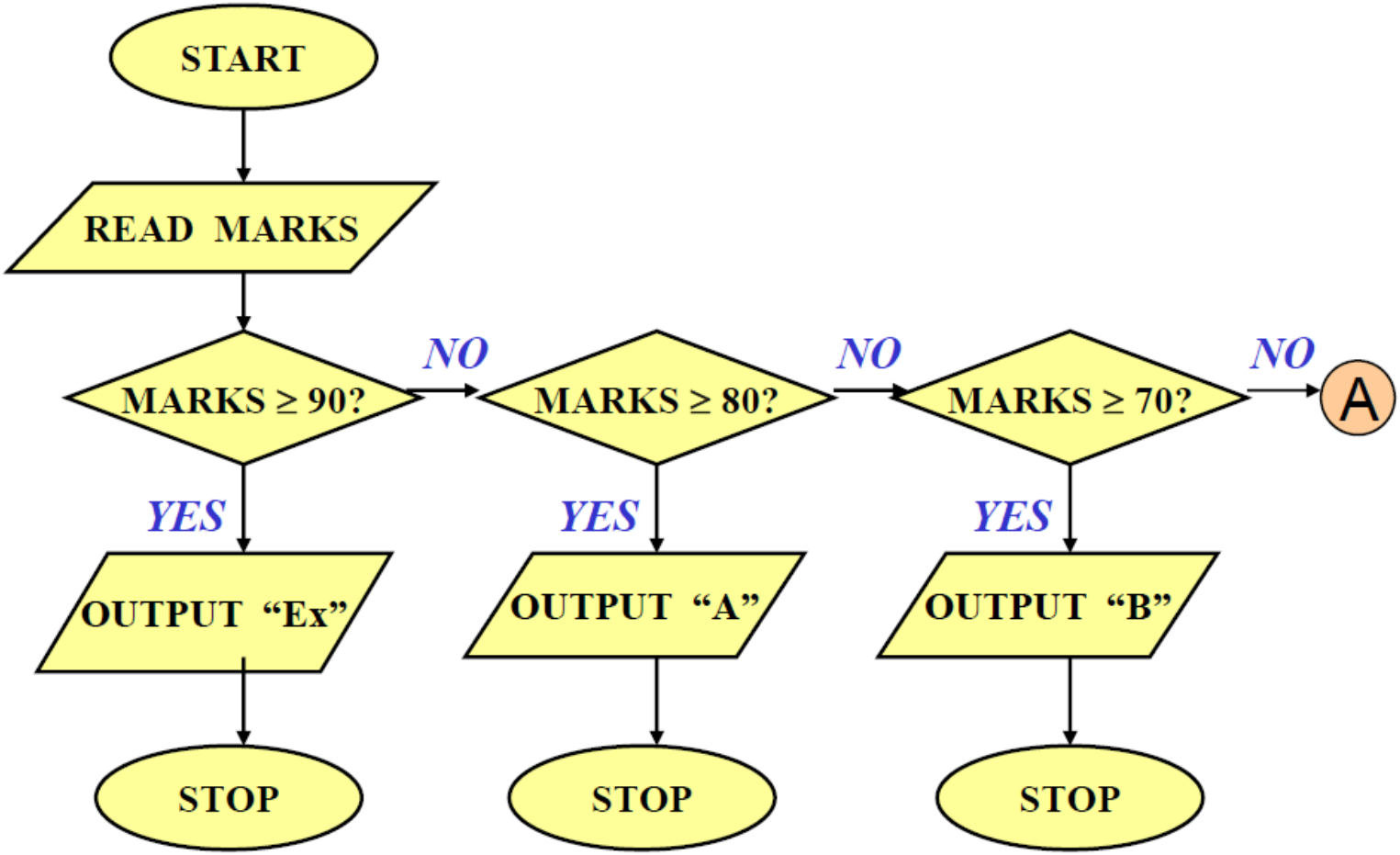
$79 \geq MARKS \geq 70$ → B

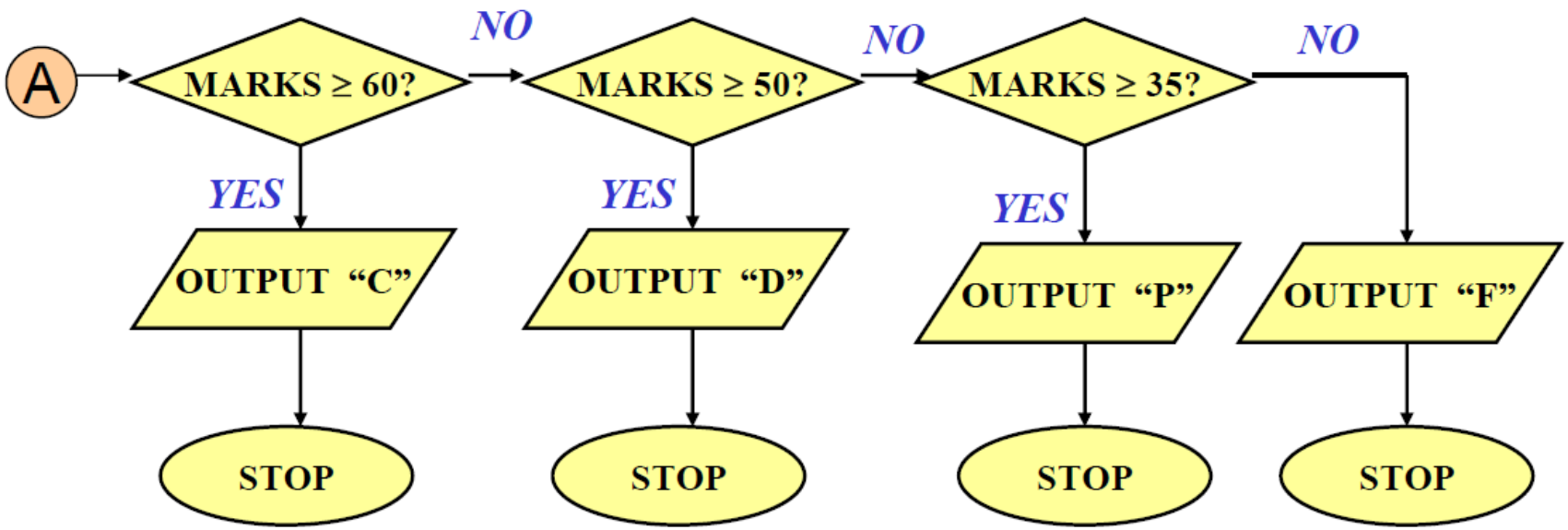
$69 \geq MARKS \geq 60$ → C

$59 \geq MARKS \geq 50$ → D

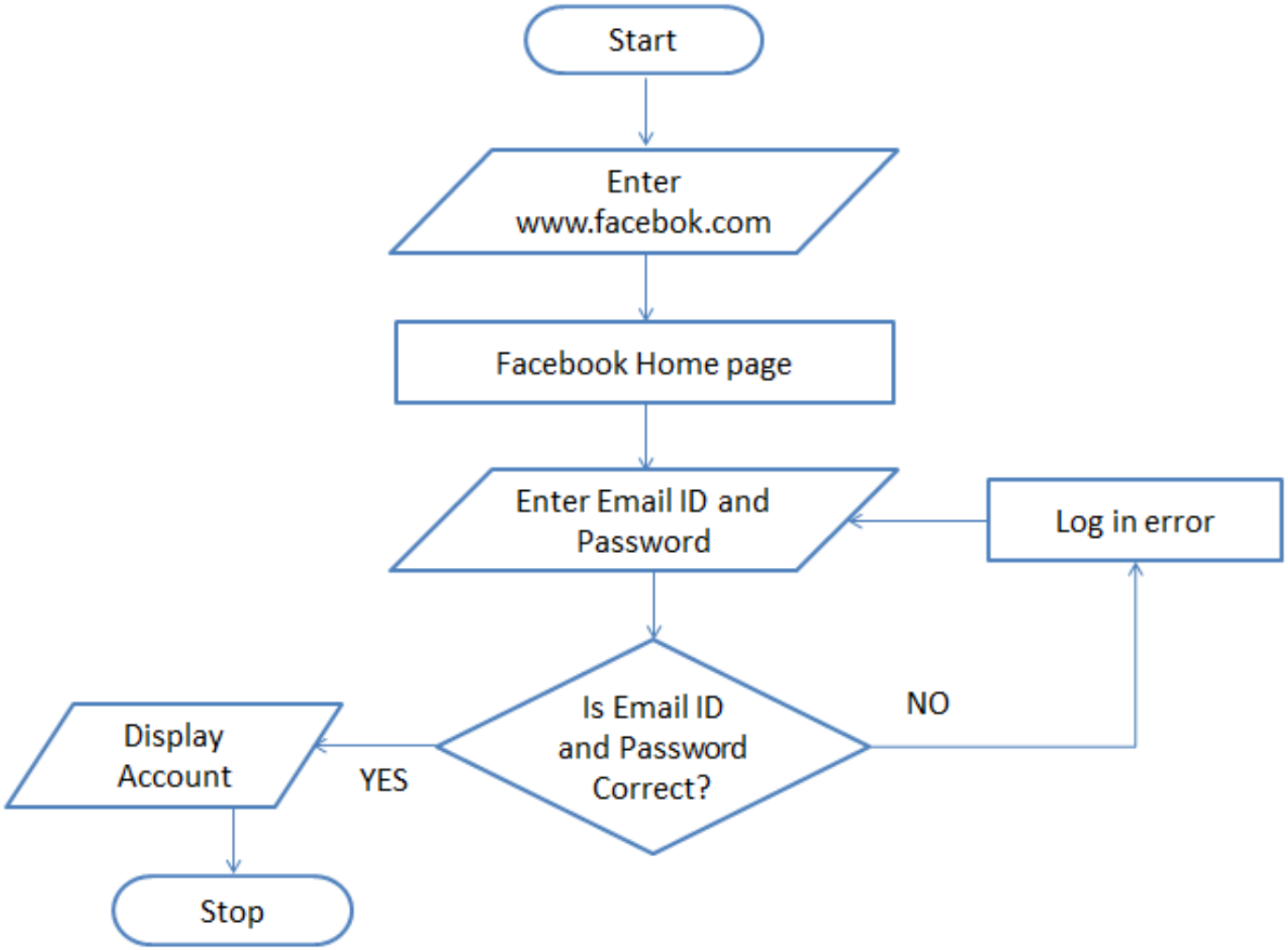
$49 \geq MARKS \geq 35$ → P

$34 \geq MARKS$ → F

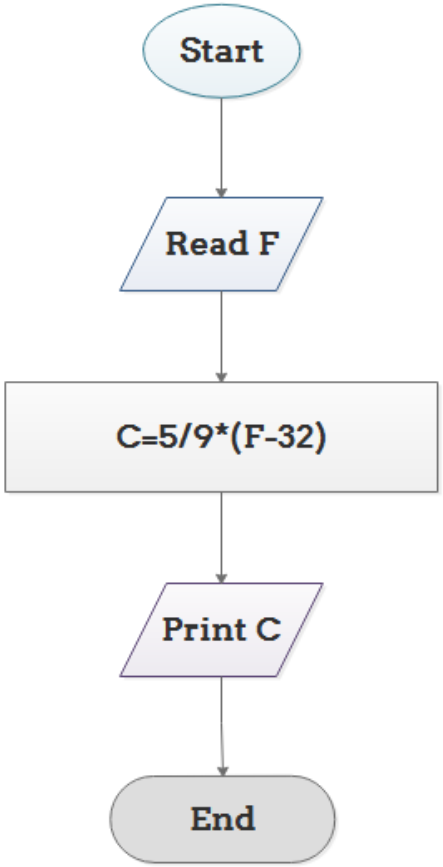

START
READ MARKS
MARKS ≥ 90?
NO
MARKS ≥ 80?
NO
MARKS ≥ 70?
NO
A
YES
OUTPUT "Ex"
STOP
YES
OUTPUT "A"
STOP
YES
OUTPUT "B"
STOP

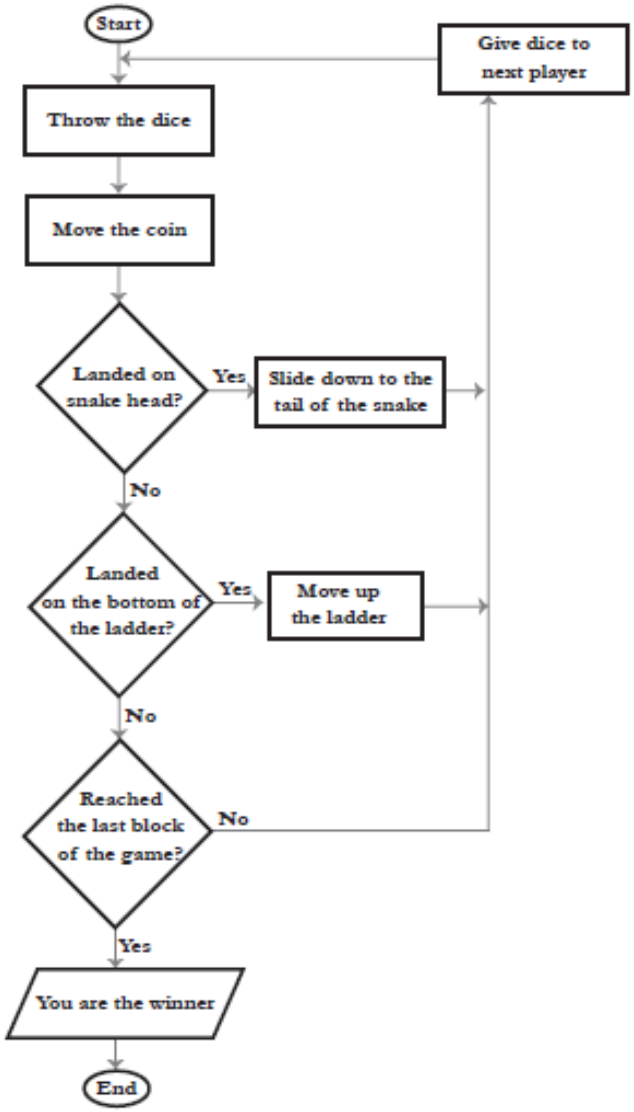

A
MARKS ≥ 60?
NO
MARKS ≥ 50?
NO
MARKS ≥ 35?
NO
YES
YES
YES
OUTPUT "C"
OUTPUT "D"
OUTPUT "P"
OUTPUT "F"
STOP
STOP
STOP
STOP

**Example:** Draw a flowchart to log in to facebook account



**Example:** Convert Temperature from Fahrenheit (°F) to Celsius (°C)




Start
Give dice to next player
Throw the dice
Move the coin
Landed on snake head?
Yes
Slide down to the tail of the snake
No
Landed on the bottom of the ladder?
Yes
Move up the ladder
No
Reached the last block of the game?
No
Yes
You are the winner
End