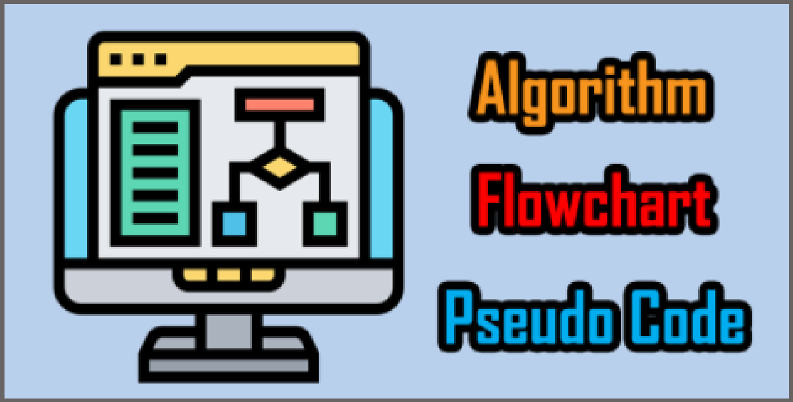
Algorithm development using Pseudo-codes and Flowcharts

Lecture 04 – ICT1132

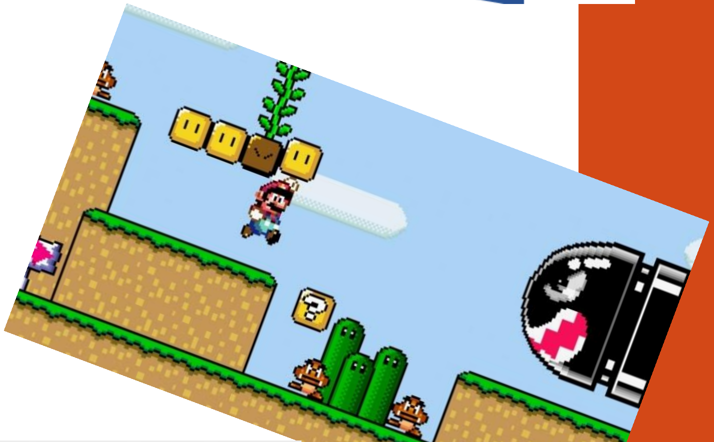


Piyumi Wijerathna
Department of ICT
Faculty of Technology

why do we use computers?

- Storage
- Retrieve
- Processing (Text, Sound, Image)
- Calculations
- Communication
- Entertaining

…… what else?



How do Computers do all these?

- How do you instruct a computer to do tasks?
  - Using Computer Programs
- How you create Programs?
  - Using Programming Languages
- Any other word for these Computer Programs?
  - Software

Computer Software

- Software has historically been considered an intermediary between electronic hardware and data.

- The physical components of a computer are the hardware; the digital programs stores and executes on the hardware are the software.

- Software can also be updated or replaced much easier than hardware.

- Software is often divided into application software and system software.

Program Design & Development

Two phases involved in the development of any program:

I. **Problem Solving Phase**

Produce an ordered sequence of steps that describe solution to the problem. This sequence is called an Algorithm.

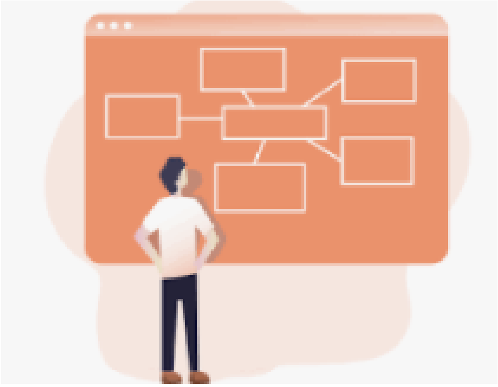
II. **Implementation Phase**

Implement the program/algorithm in some programming language.

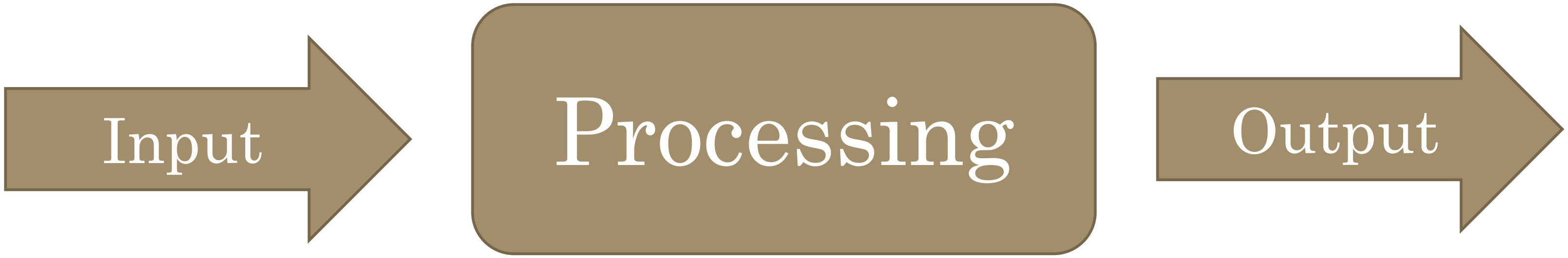
1. Problem Solving Phase

In the problem-solving phase the following steps are carried out:

- Define the problem.
- Outline the solution.
- Develop the outline into an algorithm.
- Test the algorithm for correctness.



1. Define the Problem



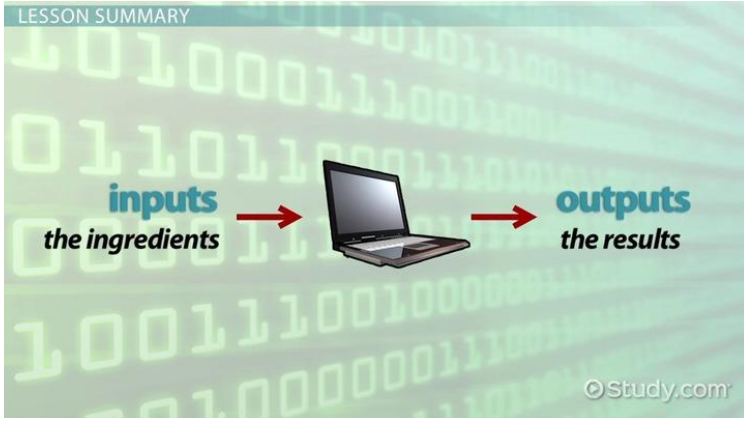
Input

Processing

Output

Keyboard
Files

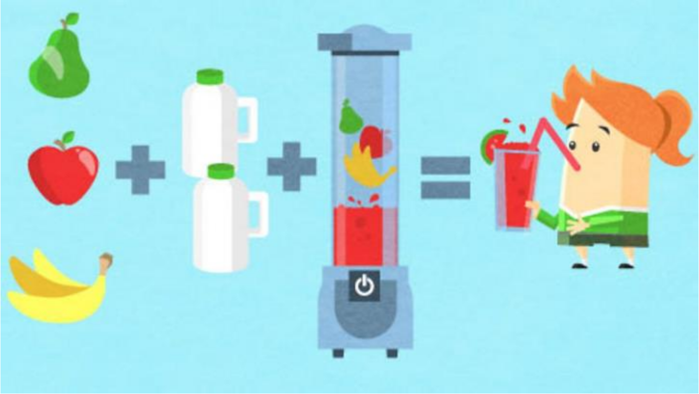
Monitor
Files



2. Outline the Solution

This is a rough draft of the solution. The outline may include,

- Major processing steps involved
- Major subtasks
- The main logic



3. Develop the outline into an algorithm

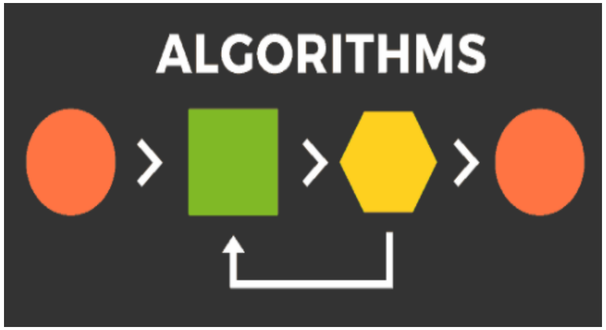
- Use algorithms to prepare the solution.

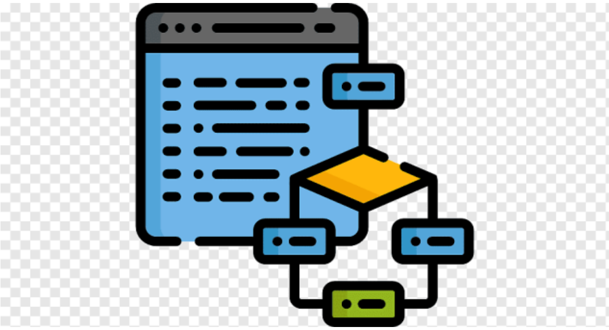
What is an Algorithm?

- An algorithm is a sequence of precise instructions for solving a problem in a finite amount of time.
- It is important to spend considerable time in designing your solution (algorithm) in order to ensure it is properly structured.
- If properly designed, the time and effort in ‘coding’ the solution, will be minimal.

4. Test the algorithm for correctness

Check for logical errors, create test data and test your algorithm.





2. Implementation Phase

The implementation phase comprises the following steps:

- Code the algorithm using a specific programming language.
- Run & test the program on the computer.
- Document and maintain the program.

An Algorithm must be

- **PRECISE**

Producing the correct solution.

- **LOGICAL**

Steps should be in a logical order.

- **CLEAR**

Every instruction is clearly and unambiguously specified.

- **EFFECTIVE**

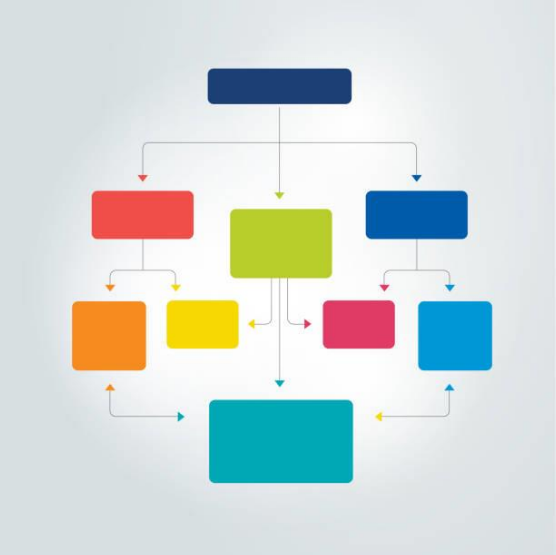
Steps are executable (be in a format which can easily implement by a programing language).

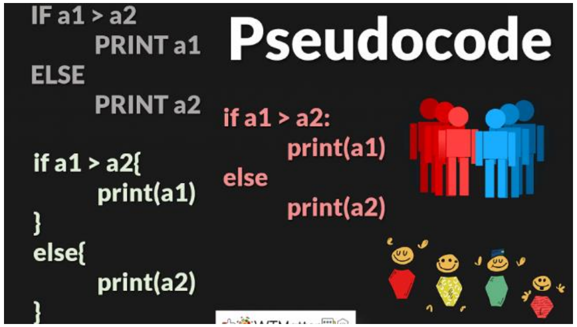
- **FINITENESS**

Obtain a solution within a finite time (terminate after specified number of steps).

Representing an Algorithm

- Flowcharts
- Pseudo Codes
- Structure Charts



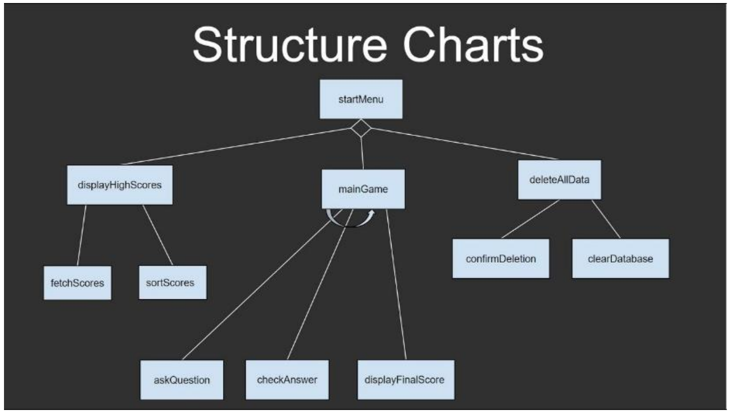


Pseudocode

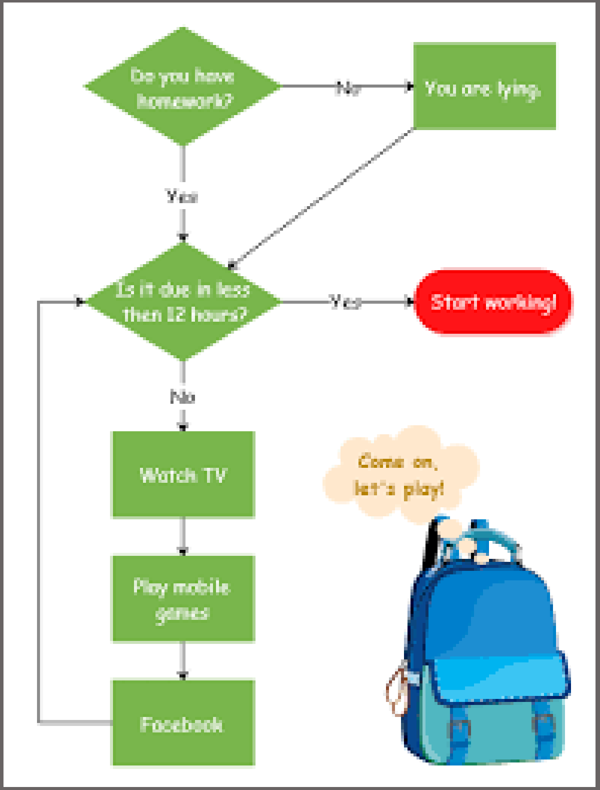
```
IF a1 > a2
        PRINT a1
ELSE
        PRINT a2

if a1 > a2{
        print(a1)
}
else{
        print(a2)
}
```

```
if a1 > a2:
        print(a1)
else
        print(a2)
```

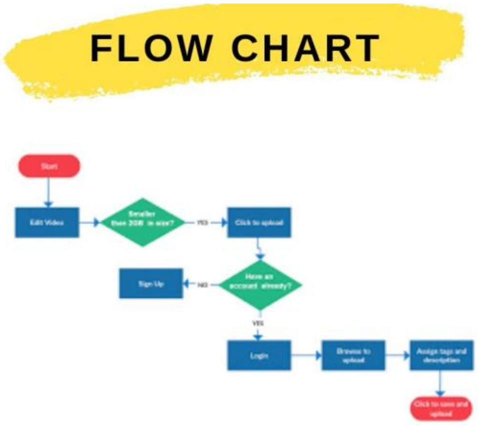


Flowcharts

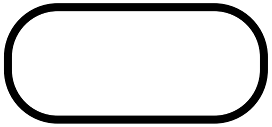
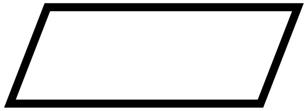
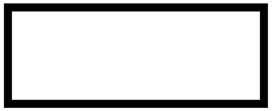
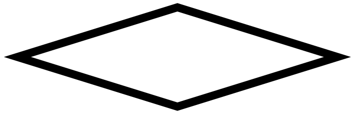
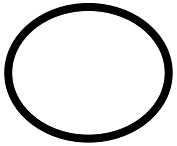
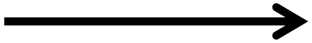

Do you have homework?
No
You are lying.
Yes
Is it due in less then 12 hours?
Yes
Start working!
No
Watch TV
Play mobile games
Facebook
Come on, let's play!

What is a flow chart?

- It is a step by step Diagrammatic representation of the program.
- Each type of task is represented by a symbol.
- The flowchart should flow from top to bottom.
- Avoid intersecting flow lines.
- Use meaningful description in the symbol.

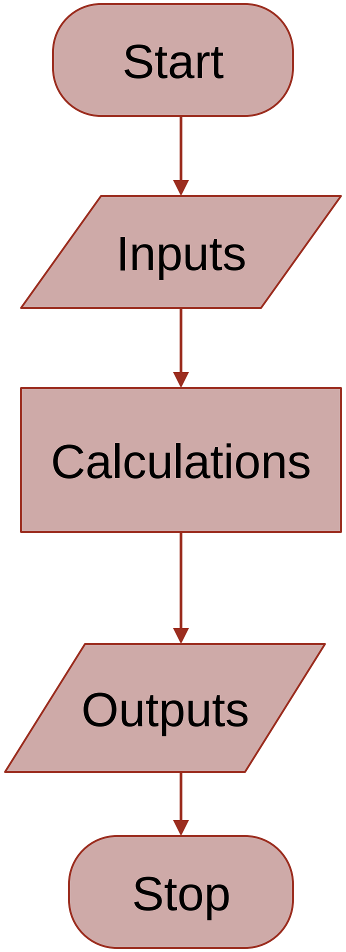


Flowchart Notations

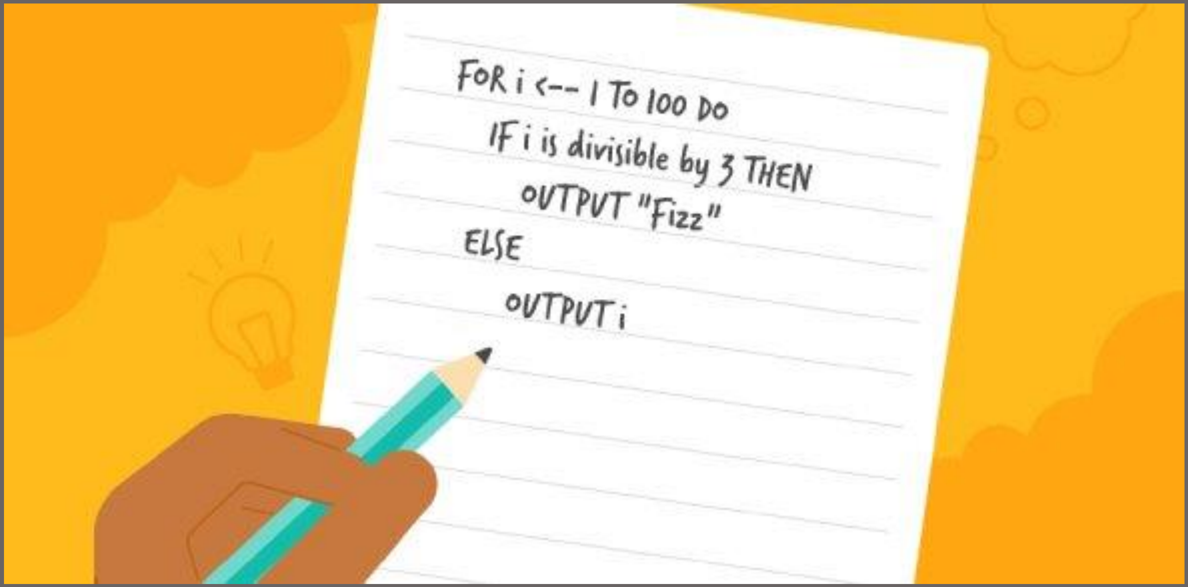
Symbol	Name	Representation
	Oval	**Start / End of a Program**
	Parallelogram	**Input / Output of Data**
	Rectangle	**Processing Operation**
	Rhombus	**Decision Box**
	Circle	**Connection**
	Arrow	**Direction of Flow**

A Typical Algorithm in a Flow Chart

Most simple algorithms will involve inputting some data, performing some calculations and finally displaying the output.



Pseudo Codes



```
FoR i <-- 1 To 100 Do
  IF i is divisible by 3 THEN
    OUTPUT "Fizz"
ELSE
   OUTPUT i
```

Pseudo Codes

- Set of specific instructions which is very similar to a computer code, but not specific to any computer language.
- Very similar to day to day English.
- It is an abbreviated version of actual computer code.
- Usually, instructions are written in uppercase, variables in lowercase and messages in sentence case.
- Once pseudo code is created, it is simple to translate into real programming code.

Benefits of Pseudo Codes

- Can plan the program.
- Can use pseudo code to describe the program to non-technical users.
- Can provide guidelines to a programmer to write the program.
- Opportunity to detect any logic error prior to actual coding.

Flow Control

- It is the order in which individual statements are executed within the program.
- Any proper algorithm can be written using following three control structures (Flow Control Structures).
  - Sequence
  - Selection (IF-ELSE)
  - Iteration (DO WHILE)

Simple Example

- Consider the process of getting up in the morning and going to work.

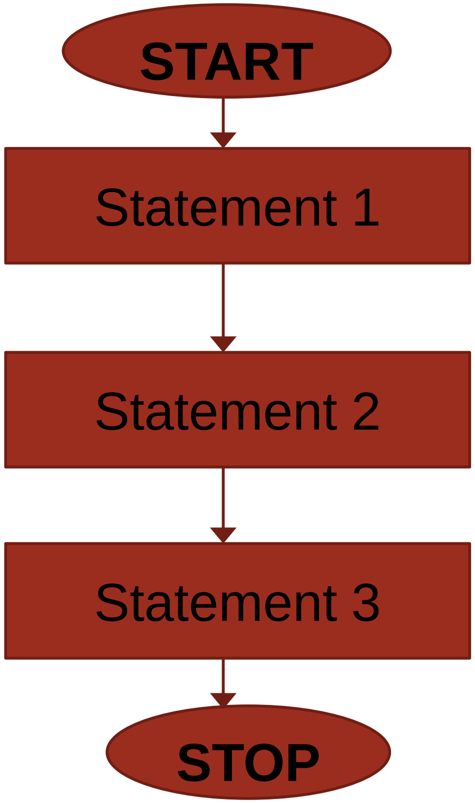
- Wake Up
- Brush Teeth
- Have a Shower
- Dry Yourself
- Get Dressed
- Have Breakfast
- Pack Bag/Lunch
- Leave

SEQUENCE

- SEQUENCE is performing one task after another sequentially.
- Solution steps must follow each other in a logical sequence.
- Computer executes the program from start to end in the same order as they are written.
- This is the basic assumption of all algorithm design.

Sequence Control

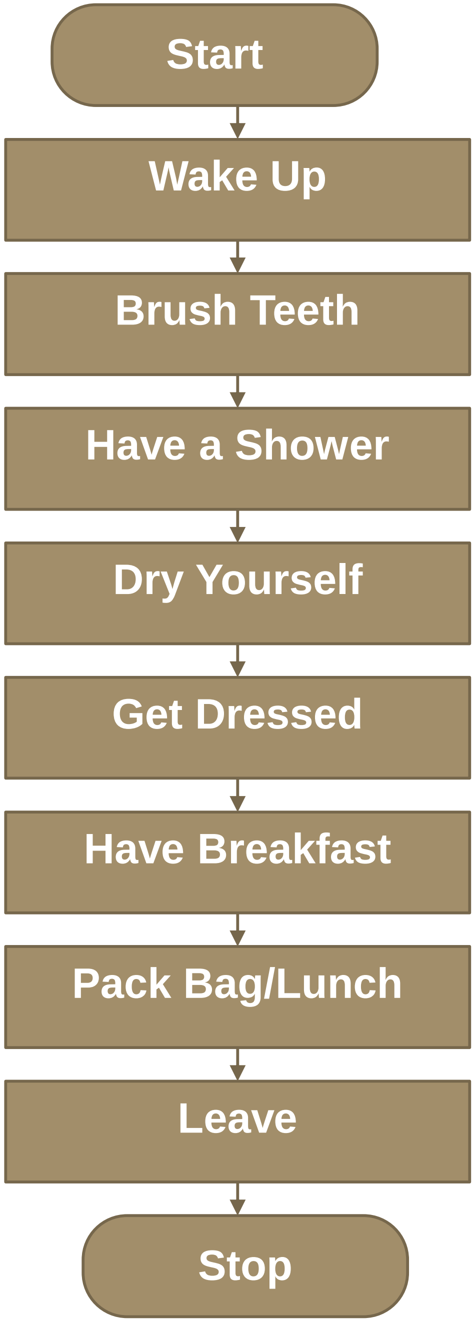
Flow chart



Pseudo code

```
statement1;
statement2;
statement3;
```

Flow Chart



Start

Wake Up

Brush Teeth

Have a Shower

Dry Yourself

Get Dressed

Have Breakfast

Pack Bag/Lunch

Leave

Stop

Pseudo Code

```
BEGIN

Wake UP;
Brush Teeth;
Have a shower;
Dry yourself;
Get dressed;
Have breakfast;
Pack bag/lunch;
Leave;

END
```

Note

- Certain events must occur in a particular order.

  for example, we should dry our self before getting dressed.

- Some events must occur prior to another event(s).

  for example, there is no drying yourself prior to a shower.

- Some other events may occur in any order and do not affect the overall solution.

  for example, we can pack bag/lunch before or after having breakfast.

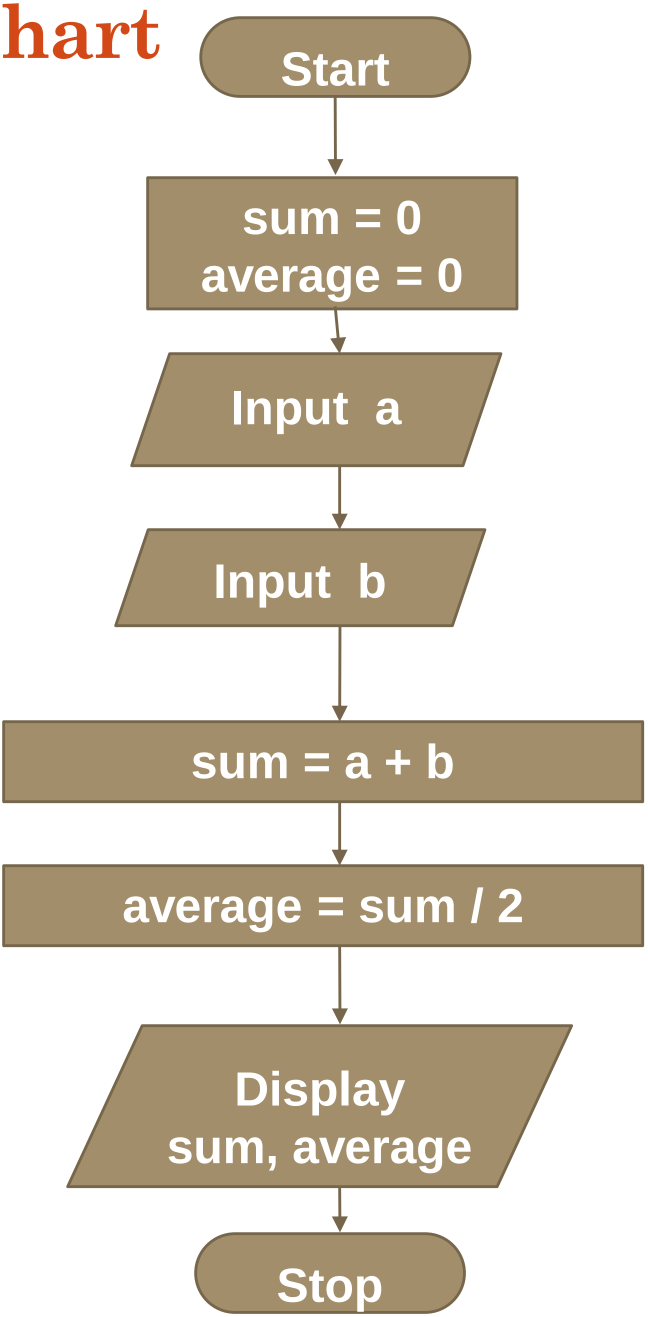
This sequence control structure can be used to represent four basic computer operations:

- Input information
- Output Information
- Perform arithmetic
- Assign values

Example

- An algorithm to find the sum and average of two numbers.

Flow chart



Pseudo Code

Comment – This Pseudo code finds the sum and average of two given numbers.

```
BEGIN
    INPUT a
    INPUT b
    sum = a + b
    average = sum/2
    OUTPUT sum &
average
END
```

- What if we want to make a choice, For Example:
    - Do we want to have breakfast or not ?
    - Do we want to have a shower or not ?
- We call this as a SELECTION.

SELECTION

- Selection statements allow programmers to ask questions(Conditions) and then, based on the decision(Selection), perform different actions/steps.

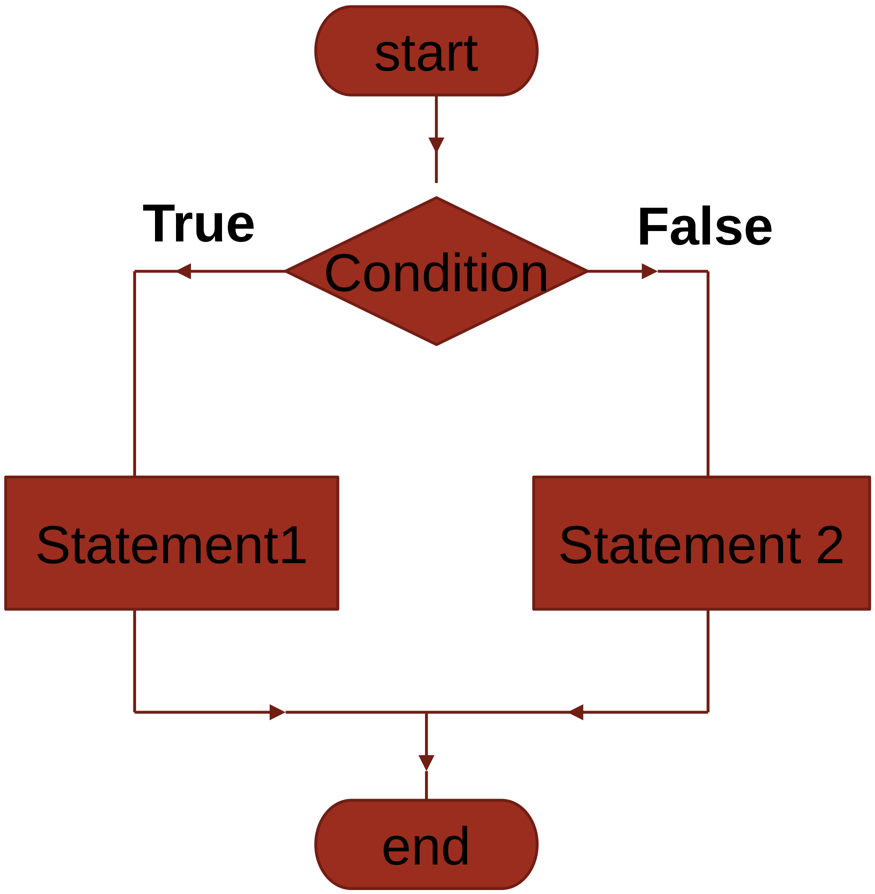
- There are two types of selection control structures.
  - Binary selection

    Two possible choices to select.
  - Multiple selection (Case selection)

    Decisions have more than two answers to select.

Binary Selection

Flow chart:

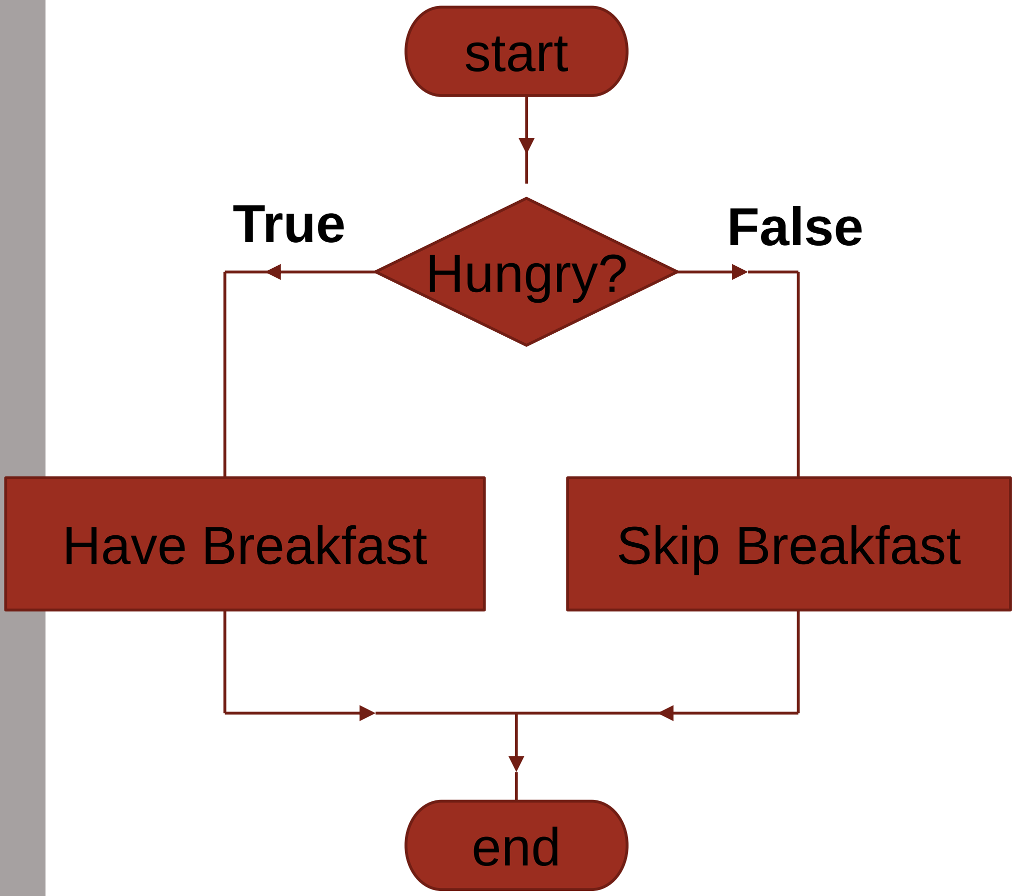


Pseudo code:

```
IF condition
THEN
    sequence-1(statements)
ELSE
    sequence-2(statements)
ENDIF
```

Example

Flow chart:

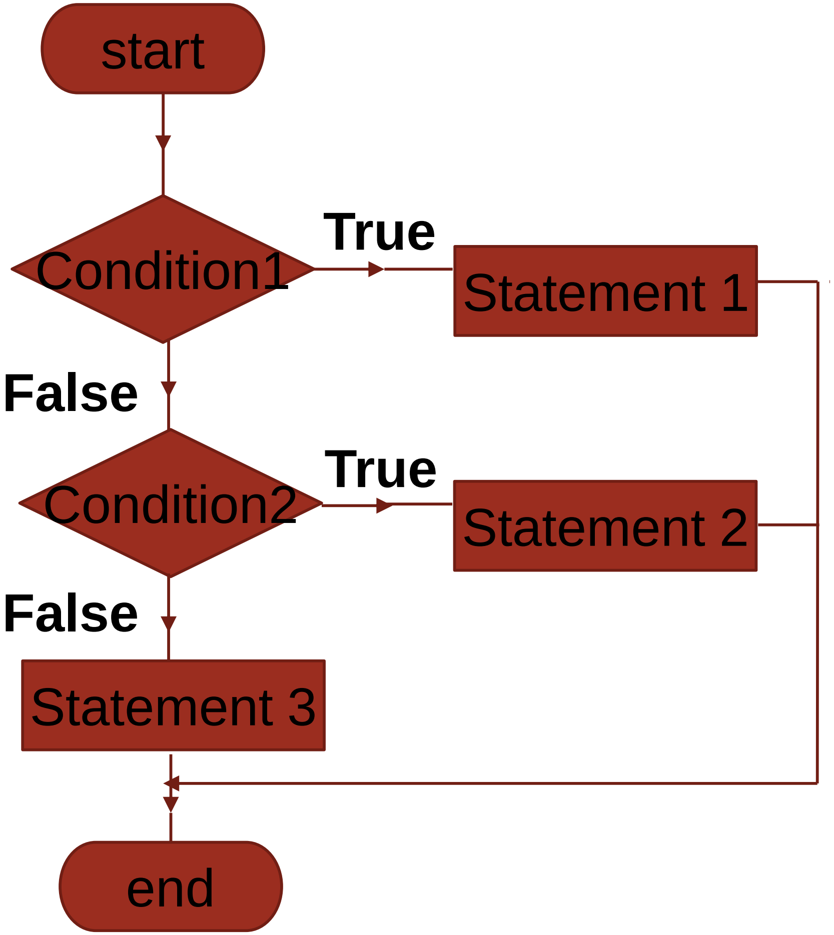


Pseudo code:

```
IF you are hungry
THEN
      Have your breakfast
ELSE
      You can skip breakfast
ENDIF
```

Multiple Selection

Flow chart:



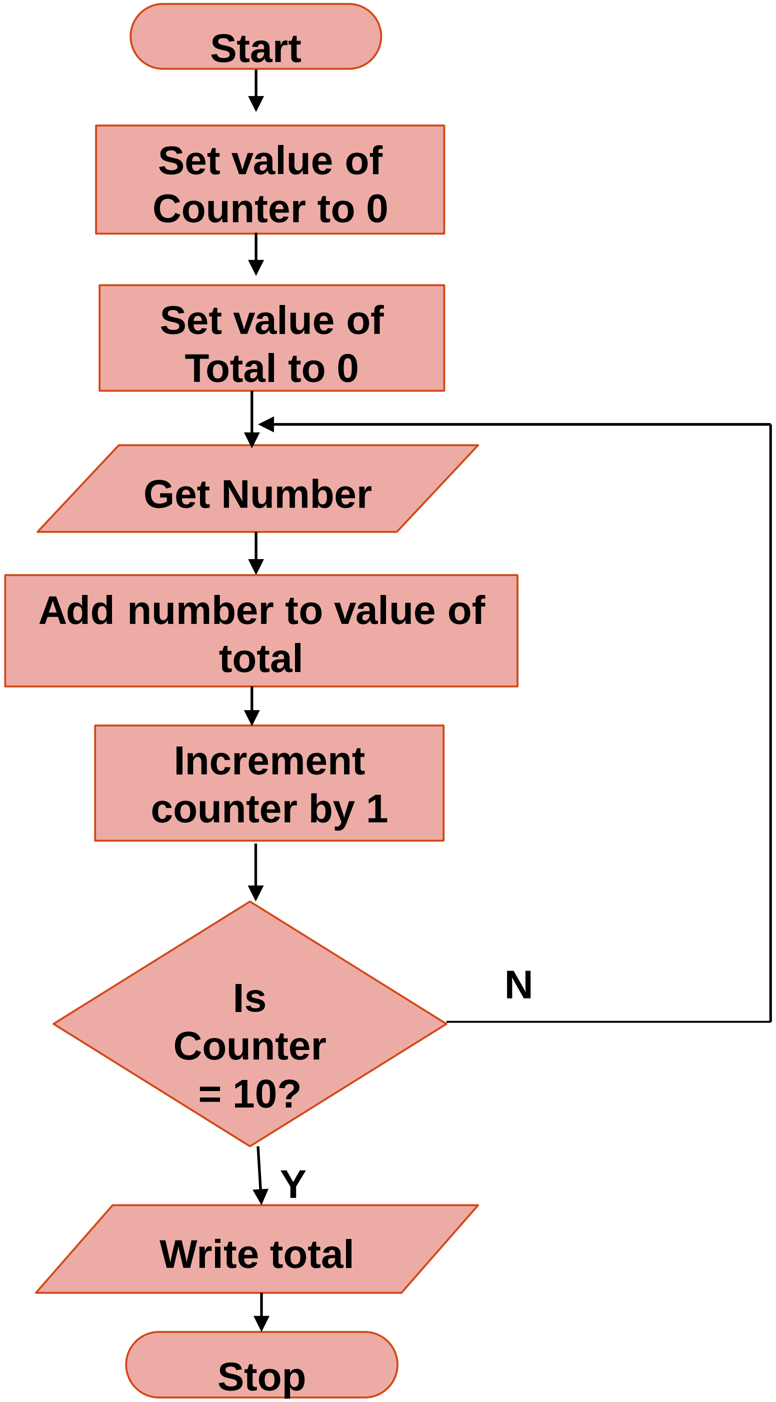
Pseudo code:

```
IF condition
THEN
     statement1
ELSE IF condition
THEN
     statement2
ELSE
     statement3
ENDIF
```

Note

- When a flowchart is too long to fit on a page and is to be continued on another page a "connector symbol" is used.
- A small circle is used to represent a connector in a flowchart.
- An alphabet or a number is written inside the circle to identify the pairs of connectors to be joined.

Example


Start

Set value of Counter to 0

Set value of Total to 0

Get Number

Add number to value of total

Increment counter by 1

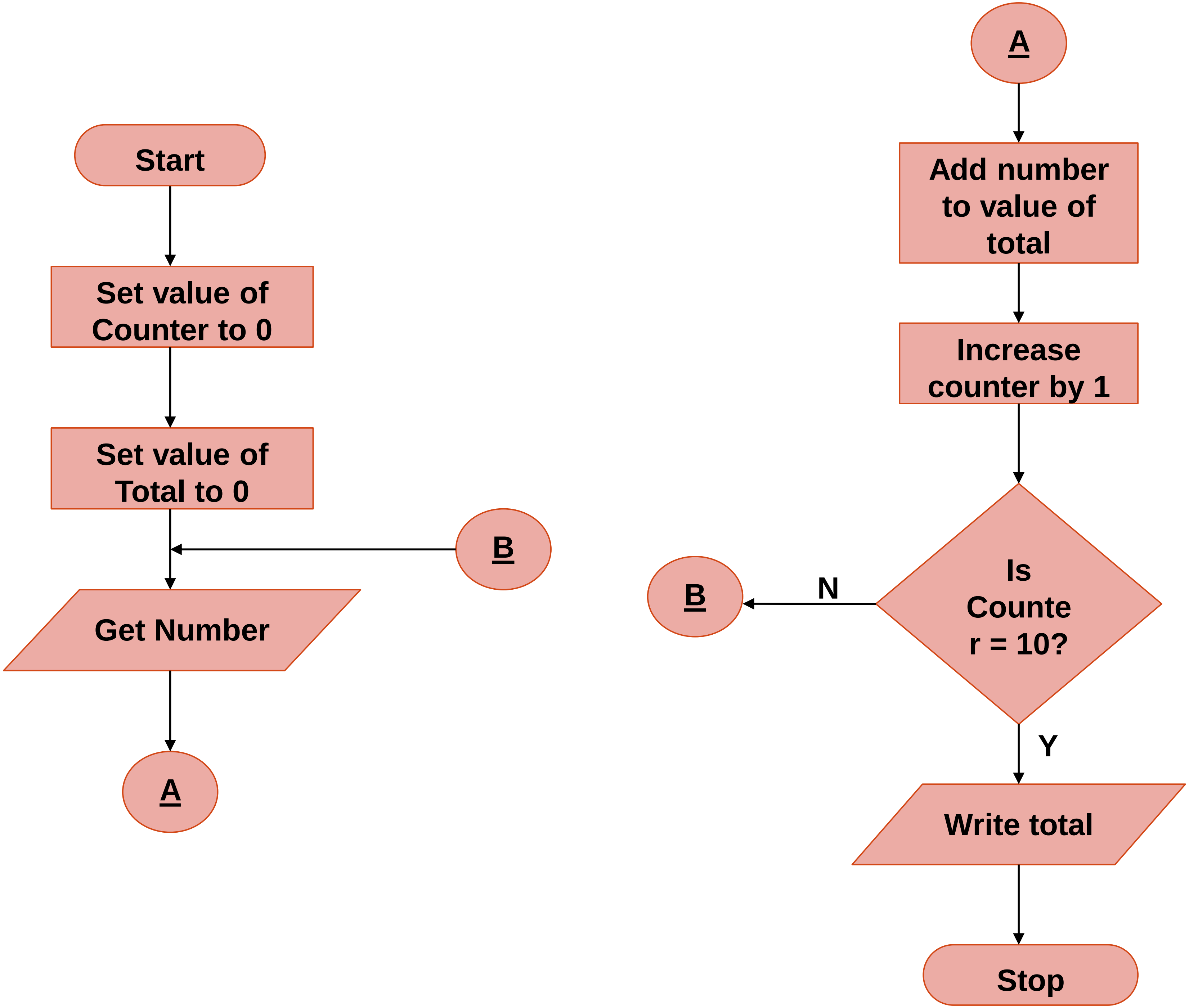
Is Counter = 10?

N

Y

Write total

Stop



Start
Set value of Counter to 0
Set value of Total to 0
B
Get Number
A
A
Add number to value of total
Increase counter by 1
Is Counte r = 10?
N
B
Y
Write total
Stop

ITERATION

- In Iteration, certain steps may need to be repeated while, or until, a certain condition is true.

For Example

- Wash yourself until get clean.
- Dry yourself until remove wet.
- Have breakfast while you are hungry.

- We call it as a Loop.

- Loops should eventually terminate and it is achieved by a test of whether a condition is *true* or *false*.
  - Repeat something while a condition is true.
  - Terminate the loop when it is false.
- There are three different types of loops in structured programming, all of which are available in the C language,
  - WHILE
  - DO WHILE
  - FOR

- *Consider the example of Morning Activities.*

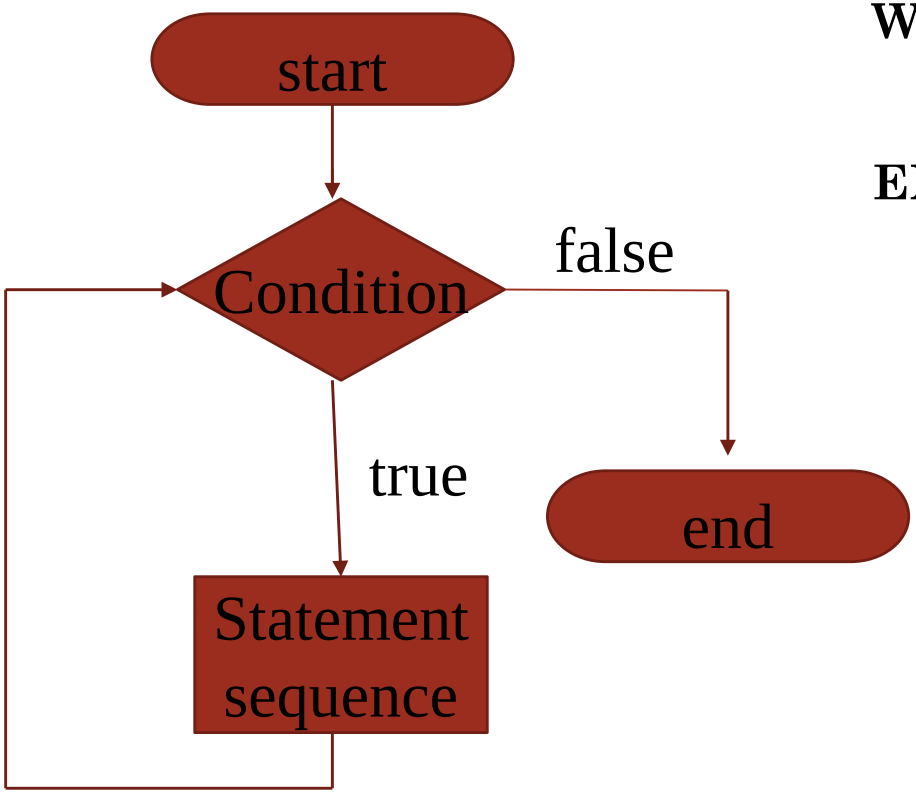
- For the step –Drying Yourself- the following loop could be substituted;

```
TEST dryness
WHILE I am wet
    DRY myself
    TEST dryness
END WHILE
```

- The loop continues until the condition of wetness removes.

ITERATION – While Loop

- Tests for terminating condition at the beginning of the loop.
- Statements will be executed if the condition evaluates to true otherwise stop the loop without any action.
- Check the condition again an again until it evaluates to false.



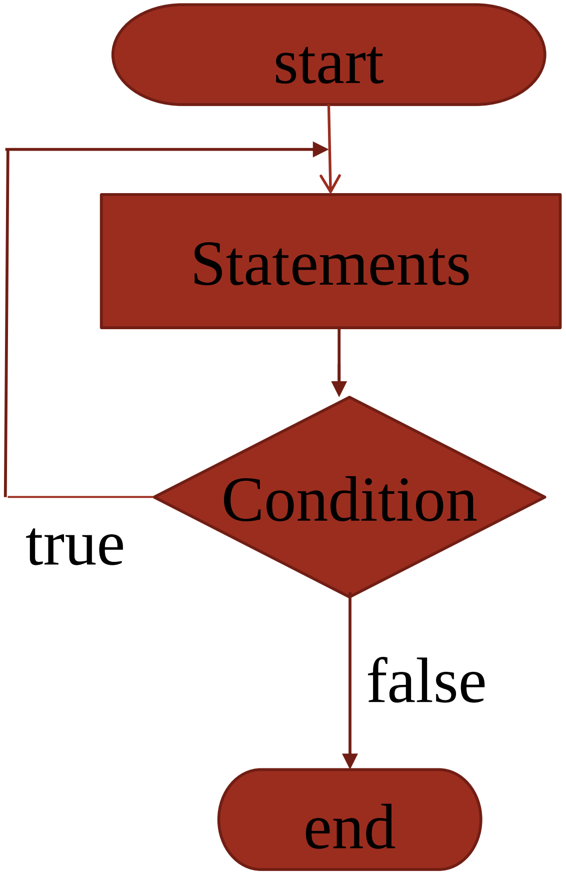
```
WHILE <Condition>
        Statement-Sequence
END WHILE
```

Example:

```
num = 3
WHILE (num<5)
        DISPLAY num
        num++
END WHILE
```

ITERATION – DO While Loop

- Not testing the condition at the beginning of the loop.
- Always execute the statements at least once.



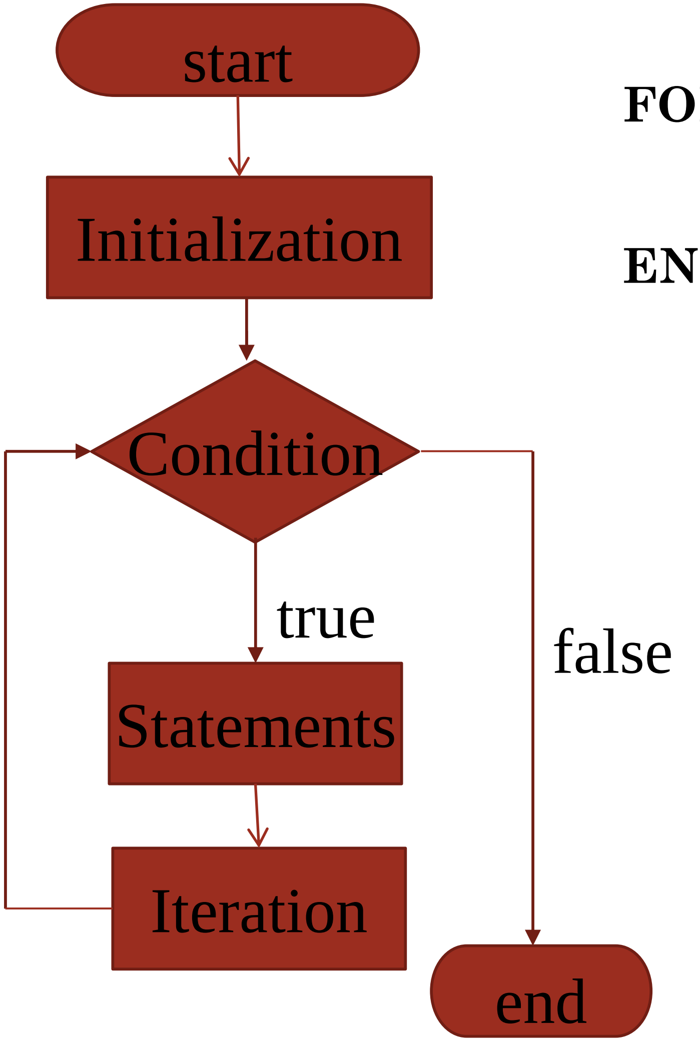
```
DO
    Statement-Sequence
WHILE<Condition>
```

Example:

```
num = 3
DO
    DISPLAY num
    num++
WHILE (num<5)
```

ITERATION – FOR Loop

- FOR loop is used when a loop is executed a specific number of times.



```
FOR (initialization, condition, iteration)
        statements
END FOR
```

Example:

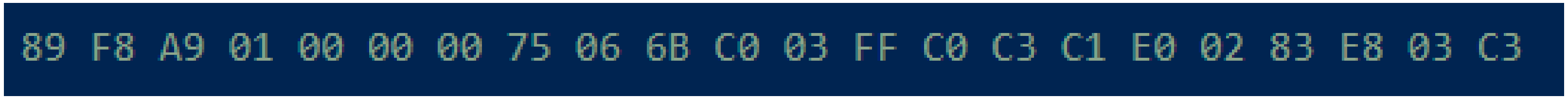
```
FOR (num=0,  num<5,  num++)
        DISPLAY num
END FOR
```

Classification of Programming Languages

- **Low-Level Programming Languages**
  - Machine language
  - Assembly language
- **High-Level Programming Languages**
  - Instructions look more like English and Math.
  - Generally result in multiple low level commands, for a single high level statement.

Machine Language

- That are interpreted directly in hardware.
- Used by early computers.
- Executable by machines, almost incomprehensible to humans.
- Programming in machine language is very tedious and prone to errors.
- Machine code is usually written in hex. Here is an example for the Intel 64 architecture:



```
89 F8 A9 01 00 00 00 75 06 6B C0 03 FF C0 C3 C1 E0 02 83 E8 03 C3
```

Assembly Language

- Thin wrappers over a corresponding machine language.
- Not directly understandable by machine. They must be translated.
- Easier for humans to use and still in use today.

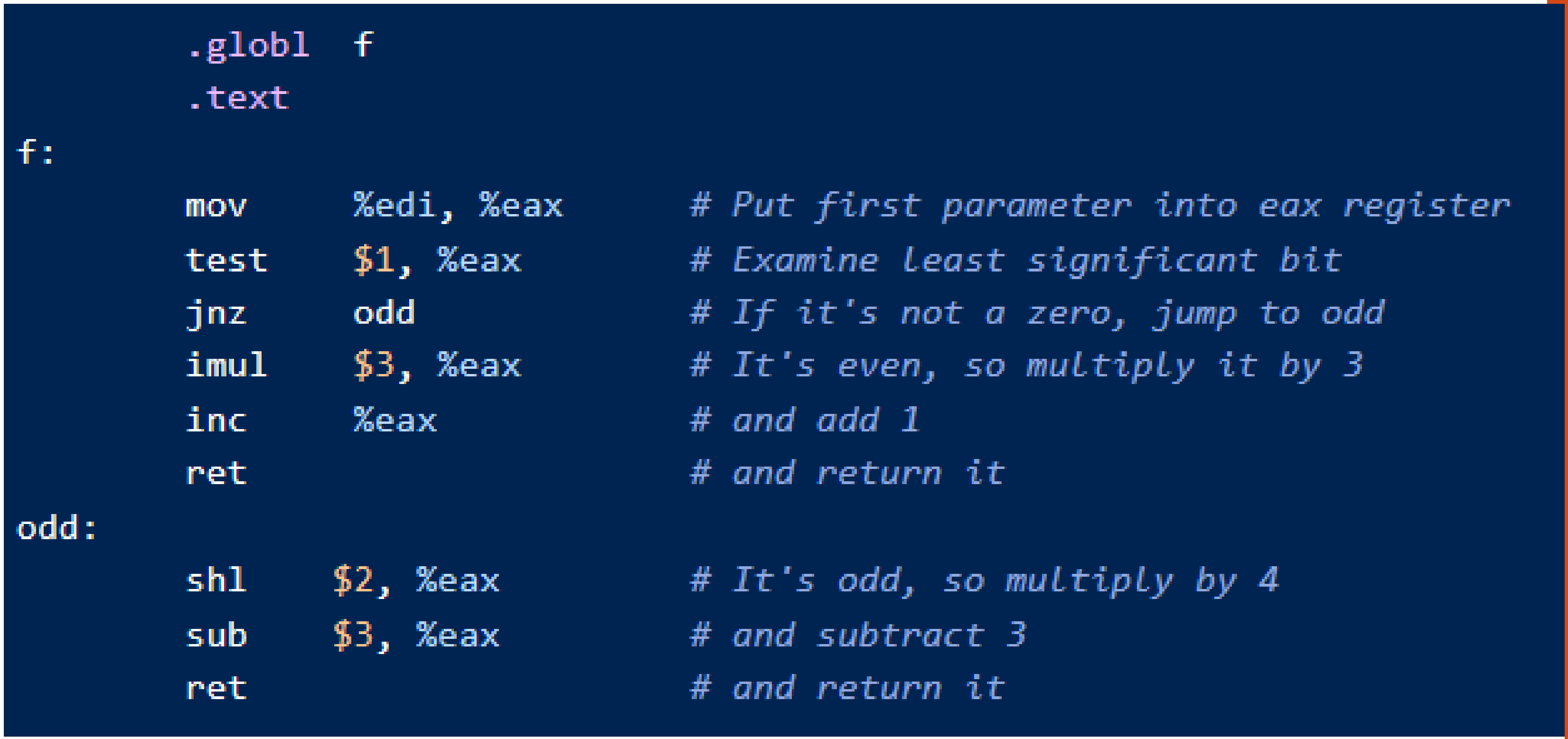
Example:

```
ADD X, Y, Reg1

ADD Reg1, Z, Reg2

STORE Reg2 SUM
```

Assembly Code : Intel 64 architecture using the GAS assembly language.



```
        .globl  f
        .text
f:
        mov     %edi, %eax      # Put first parameter into eax register
        test    $1, %eax        # Examine least significant bit
        jnz     odd             # If it's not a zero, jump to odd
        imul    $3, %eax        # It's even, so multiply it by 3
        inc     %eax            # and add 1
        ret                     # and return it
odd:
        shl    $2, %eax         # It's odd, so multiply by 4
        sub    $3, %eax         # and subtract 3
        ret                     # and return it
```

High Level Programming Language

- Uses syntax resembling combination of mathematical notation and English.
- Easy for humans to understand.
- Not understandable by machines, must be translated using a compiler or an interpreter.
- Programming tools such as integrated programming environment with a debugger are available to aid in programming process.

High Level Programming Languages

Pascal	RESULT := X + Y+ Z;
FORTRAN	RESULT = X + Y + Z
COBOL	COMPUTE RESULT = X + Y + Z.
C	RESULT = X + Y + Z;
C++	RESULT = X + Y + Z;
Ada	RESULT := X + Y + Z;
PL/1	RESULT = X + Y + Z;

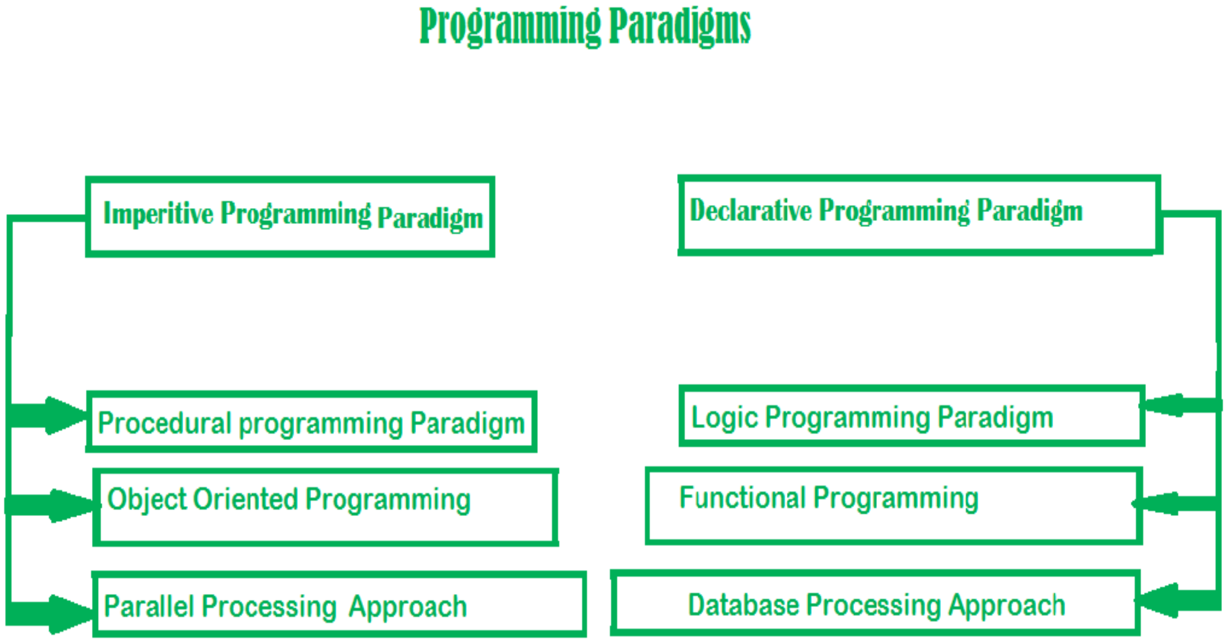
Programming Paradigms

Programming Paradigms

- Programming languages can be categorized into programming paradigms.
- In reality, very few languages are "pure". Most combine features of different paradigms.
- Programming paradigm is a fundamental style of computer programming.
- It serves as a pattern or model for a programming language.
- Paradigms differ in concepts and abstractions used to represent the elements of program.

Principle programming paradigms

- Procedural
- Object-Oriented
- Functional
- Logic
- Concurrent
- Scripting


Programming Paradigms
Imperitive Programming Paradigm
Declarative Programming Paradigm
Procedural programming Paradigm
Object Oriented Programming
Parallel Processing Approach
Logic Programming Paradigm
Functional Programming
Database Processing Approach

Example Languages

- Procedural

  Assembler, Fortran, Cobol, C, etc

- Functional

  Haskell, Erlang, ML, etc

- Logical

  Mercury

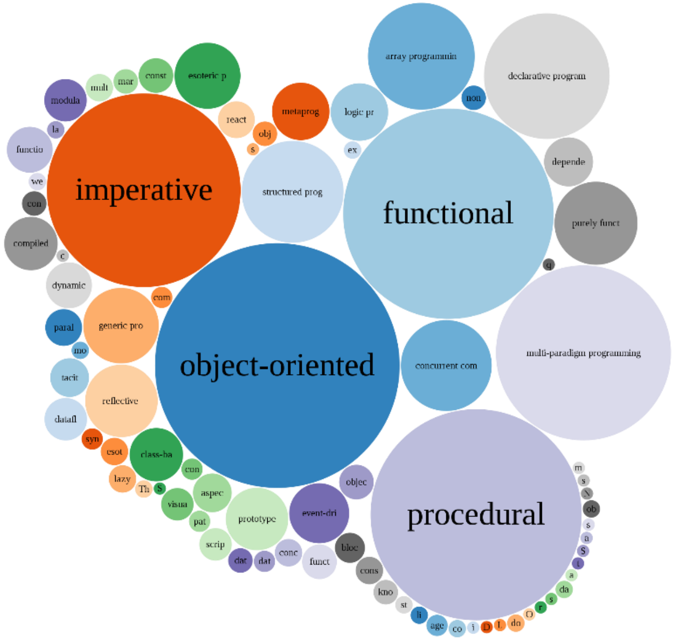
  Prolog

- Object Oriented

  Smalltalk, Java, C# etc.

- Scripting

  SQL, Perl, PHP, etc.



Imperative Programming

- Derived from latin word *imperare* means "to command".
- It is based on commands that update variables in storage.
- It is a programming paradigm that describes computation in terms of statements that change a program state.
- It defines sequences of commands for the computer to perform.

Contd..

- In imperative programming, a name may be assigned to a value and later reassigned to another value.
- A name is tied to two bindings, a binding to a location and to a value.
- The location is called the l-value and the value is called the r-value.

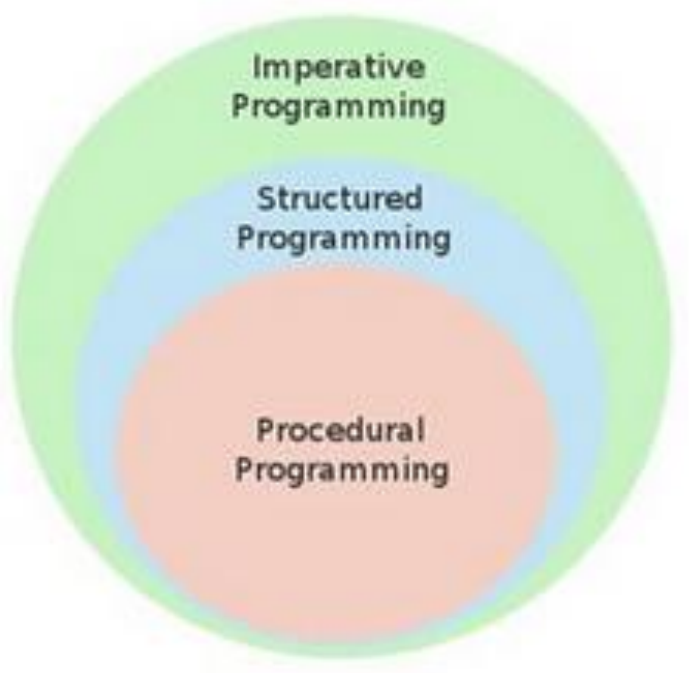
For example,

X = X+2

- Assignment changes the value at a location.
- A program execution generates a sequence of states.

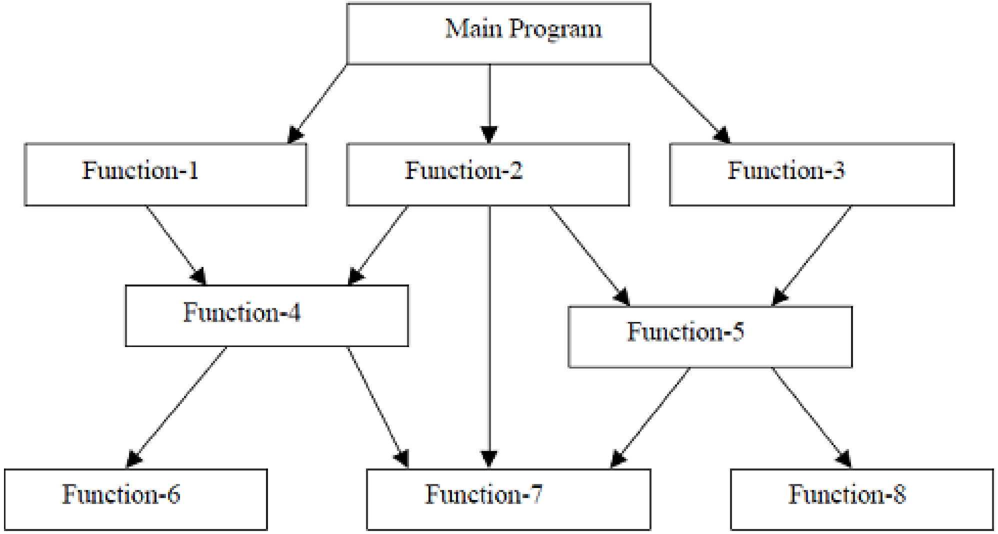
Structured Programming

- Procedural programming is a subset of structured paradigm. C is a Structured Programming Language.
- The goal of structured programming is to provide control structures that make it easier to reason about imperative programs.



Procedural Programming

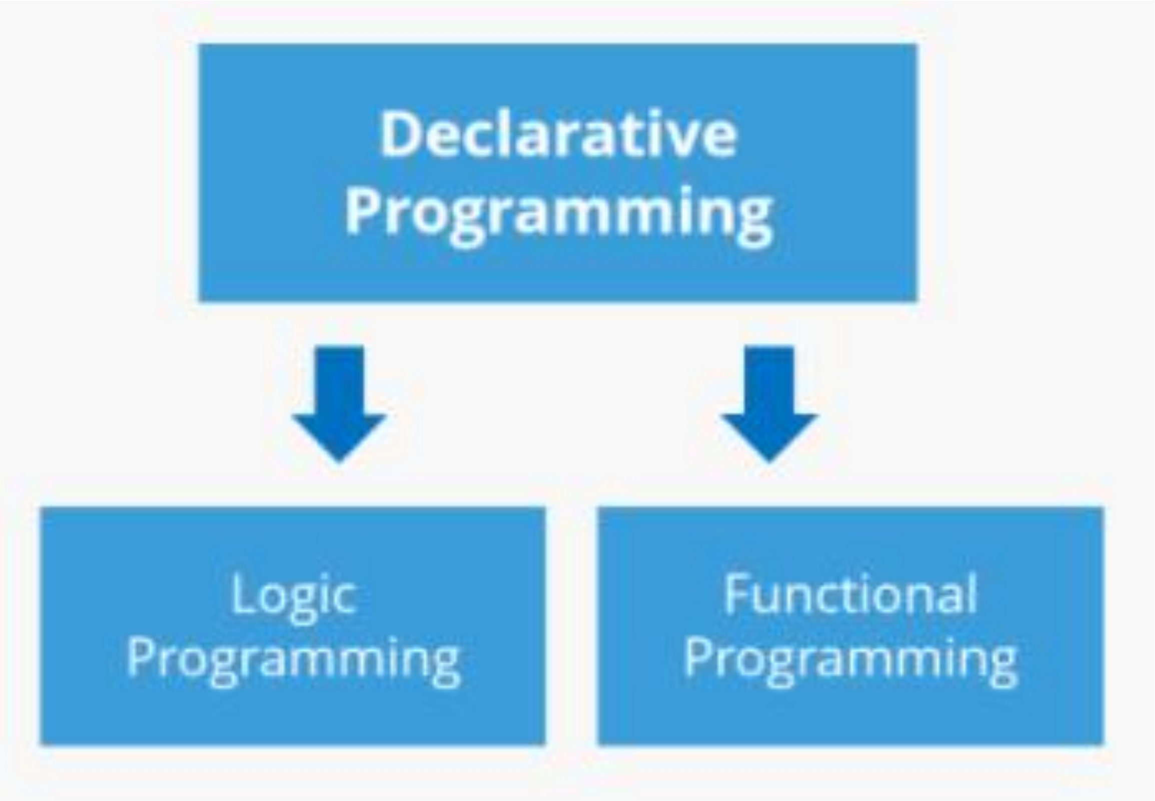
- The program is built from one or more procedures.
- It provides a programmer a means to define precisely each step in the performance of a task.



Structure of procedural oriented programs

Declarative Programming

- Declarative programming is a non-imperative style of programming.
- It does not explicitly list command or steps that need to be carried out to achieve the results.
- It is a style of building the structure and elements of computer programs that expresses the logic of a computation without describing its control flow.


Declarative Programming
Logic Programming
Functional Programming

Functional Programming

- It treats computation as the evaluation of mathematical functions and avoids state and mutable data.
- It emphasizes the application of functions, in contrast to the imperative programming style.
- Functional programming is all about expressions.
- Functions are used as objects in functional programming.
- Functional Programming is about abstraction and reducing complexity.

Example

```
spam = ['pork','ham','spices']
numbers = [1,2,3,4,5]
def eggs(item): return item
map(aFunction, aSequence)
```

it has been famously described:

> *“Functional programming is like describing your problem to a mathematician. Imperative programming is like giving instructions to an idiot.”*

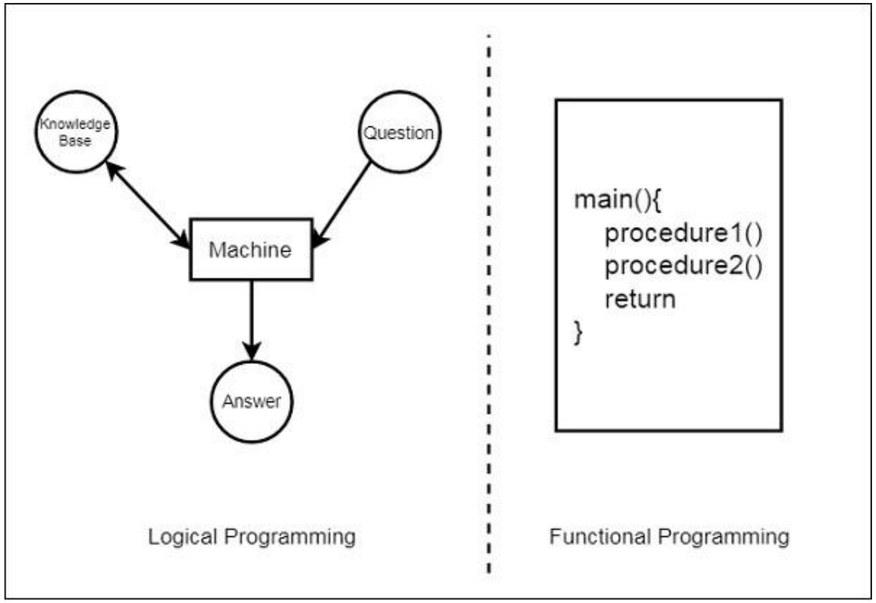
Logic Programming Paradigm

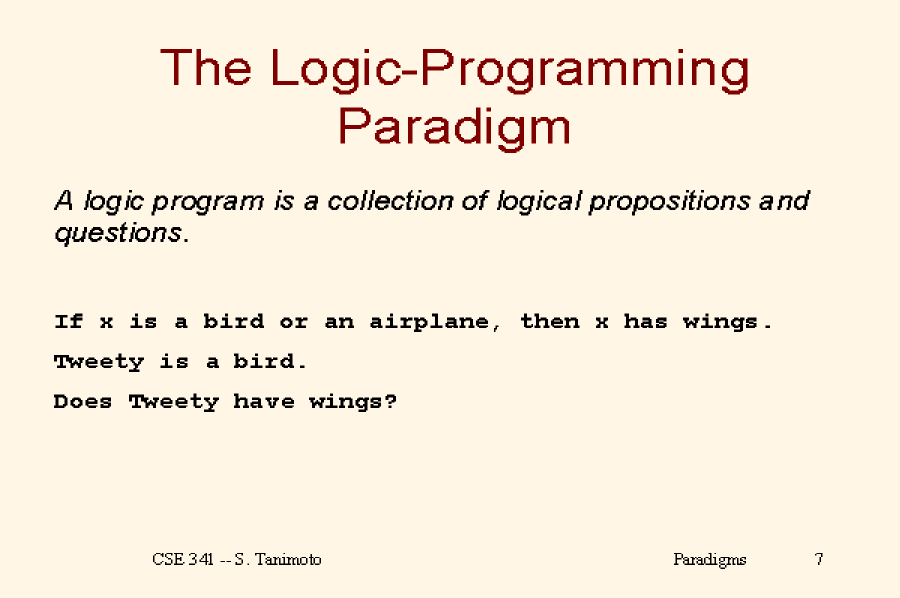
- It is the use of mathematical logic for computer programming.
- The problem-solving task is split between the programmer and theorem-prover.
- To study logic programming means to study proofs.
- It is based upon the fact of a backwards reasoning proof.

Example : If B1 and … and Bn then H.

Prolog

- Prolog is a general purpose logic programming language associated with artificial intelligence and computational linguistics.
- It is based on Facts and Rules.





The Logic-Programming Paradigm

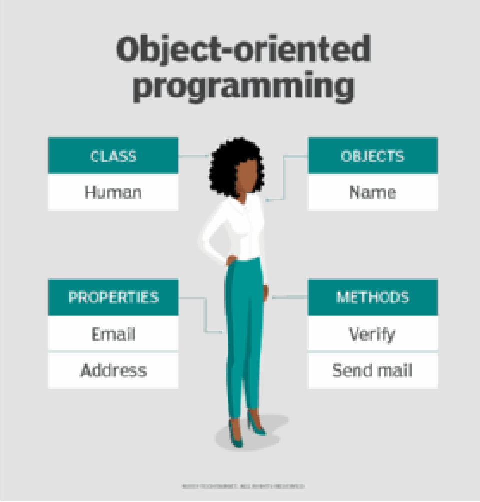
A logic program is a collection of logical propositions and questions.

```
If x is a bird or an airplane, then x has wings.
Tweety is a bird.
Does Tweety have wings?
```

CSE 341 -- S. Tanimoto Paradigms 7

Object Oriented Paradigm

- Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code.
- Data in the form of fields, and code, in the form of procedures.
- A feature of objects is that an object's own procedures can access and often modify the data fields of itself.



Exercise

- What are the differences between these programming paradigms given below.

1. OOP vs Functional Programming
2. OOP vs Procedural Programming

Any
Questi n

THANK YOU… !