



CSCI 11032/CTEC 11052 – Structured Programming I

Dr. Sidath R. Liyanage
email: sidath@kln.ac.lk

Faculty of Computing and Technology
University of Kelaniya

Programming Concepts and Introduction to Algorithms

LEARNING OUTCOMES

- By the end of this lesson you should be able to
 - Define a program/an algorithm
 - Describe the steps to develop a program
 - Understand the need for a professional approach to the design of software
 - Understand the advantages and disadvantages of the variety of tools which assist in the process of program design
 - Describe some programming paradigms
 - Write an algorithm for a small problem

What is a Program?

- A *program* is a *collection* of computer *instructions* or *operations* which are grouped together in a logical manner to accomplish a given task.

What is an Algorithm?

- An *algorithm* refers to a detailed solution to a problem.
- The solution is outlined in a manner in which the steps are clearly defined.

Steps to Developing a Program

1. Define the problem.
2. Outline the solution.
3. Develop the outline into an algorithm.
4. Test the algorithm for correctness.
5. Code the algorithm into a specific programming language.
6. Run the program on the computer.
7. Document and maintain the program

1. Define The Problem

At this stage, what is concentrated on is looking at the problem and identifying the following items:-

- *inputs* to the system
- *outputs* of the system
- general *processing* steps to produce the output.

The above three items can be presented in a *defining diagram*.

2. Outline The Solution

During this stage, certain details are *identified* from the problem by analysing it further, such as:-

- major processing tasks involved.
- major subtasks.
- major control structures.
- major variables or record structures used.
- Main line logic.

3. Develop The Algorithm

- At this stage, a detailed step by step algorithm is written out.
- The algorithm is the *design* of the program.
- The solution can be written using either one of the following two tools:-
 - pseudocodes
 - flowcharts

4. Test Algorithm For Correctness

- Once a solution is developed, the solution must be tested for any drawbacks or problems.
- Testing refers to the *process* of *checking* the algorithm and *removing* any possible errors within it.
- Algorithm checking involves the use of a *desk check table*.

5. Code the Algorithm

- This stage involves *converting* the algorithm into the *actual* program using a *specific* programming language

6. Run the Program

- This stage involves the actual *running* of the *program* on the computer system.
- Problems may be encountered here, whereby they are to be rectified

7. Document and Maintain the program

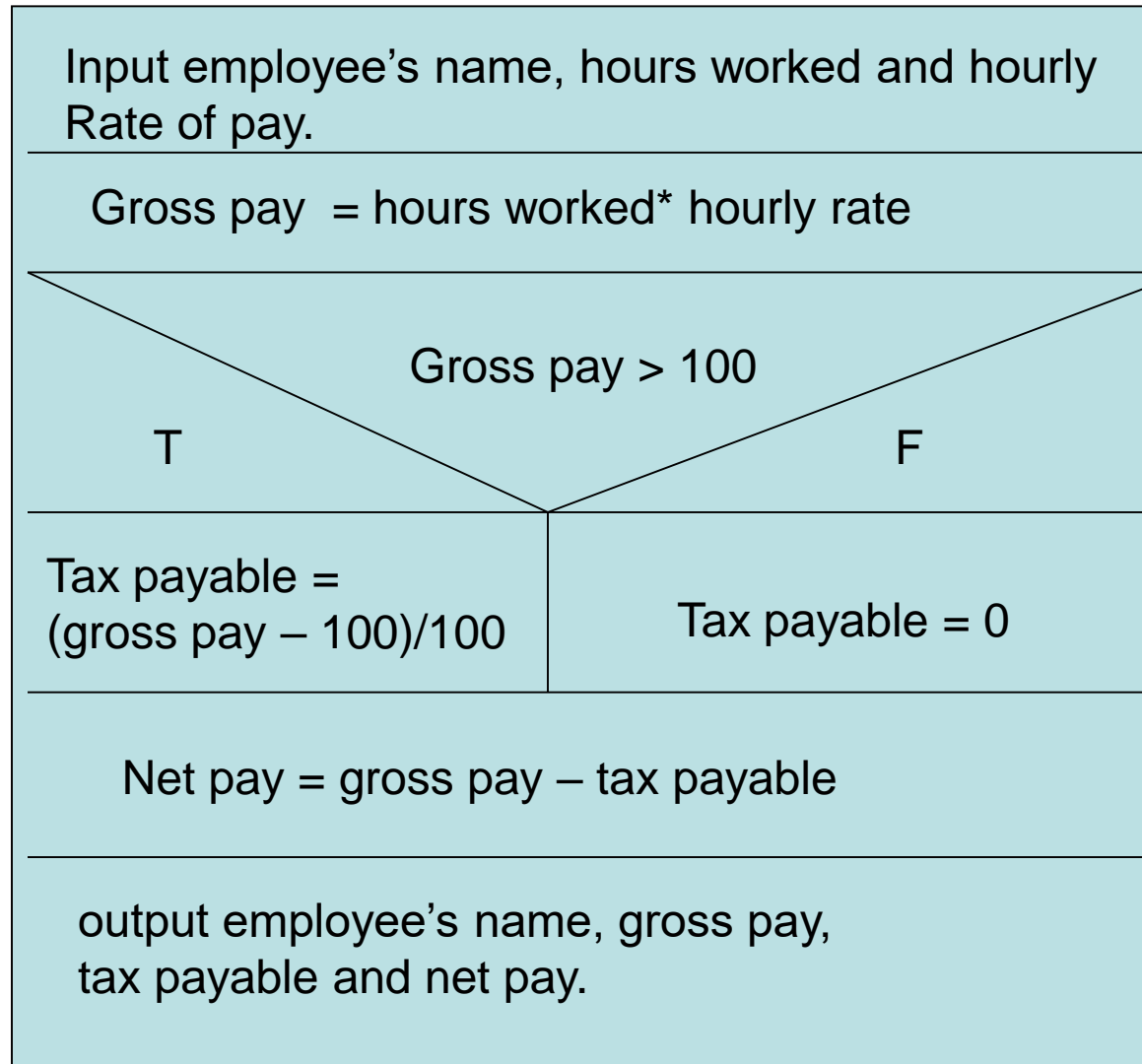
- At this stage, if the program developed is successful in solving the problem, complete documentation must be *compiled* which contains details produced at *all* stages of the program development cycle.
- Although this stage is listed as the *last* stage, documentation is an *on-going* process which starts from the *very beginning* of the program development stage.

Tools for program design

- ***It must be concise.***
- ***It must be unambiguous.***
- ***It must be capable of machine execution***
- ***It must promote elegance in the solution***

Tools to Solve Problems

- ***Nassi-Shneiderman (NS) diagrams***



- ***Pseudocode***

begin

input EmployeeName, HoursWorked, HourlyRate

 GrossPay=HoursWorked +HourlyRate

if GrossPay >100 **then**

 TaxPayable = (GrossPay-100)/10

else

 TaxPayable=0

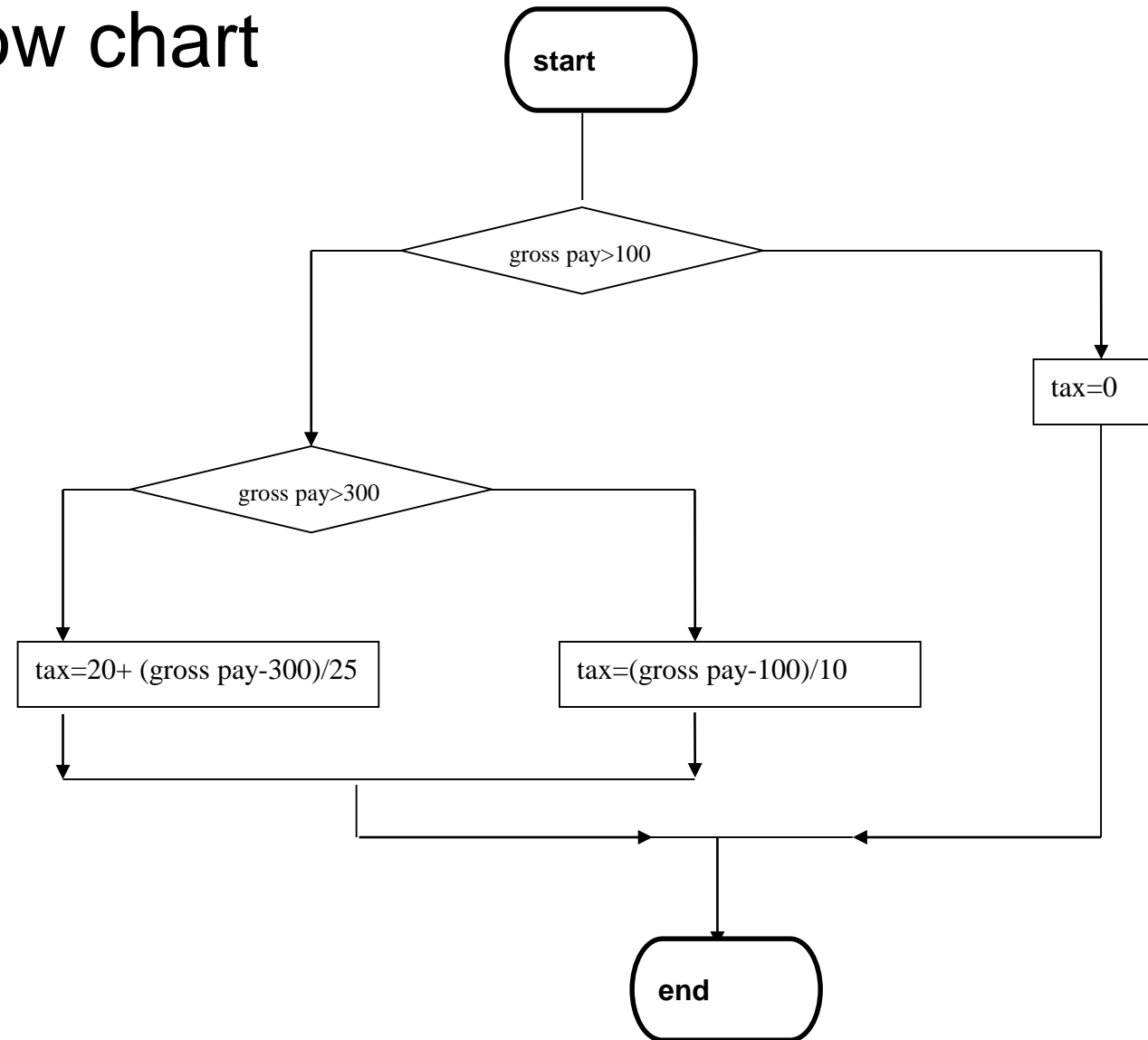
endif

 NetPay=GrossPay-Taxpayable

display EmployeeName, GrossPay,
 TaxPayable, NetPay

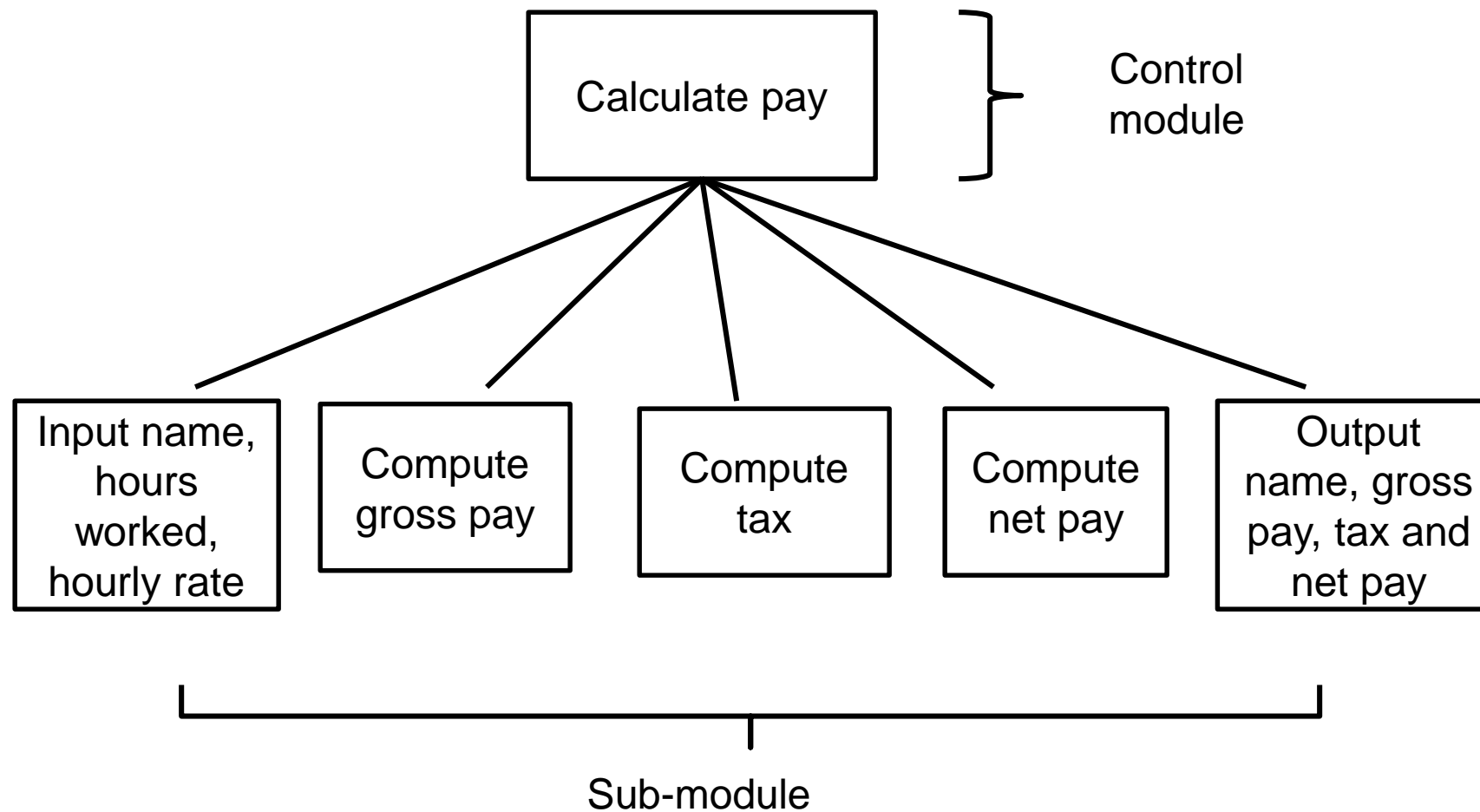
end

- Flow chart



Structure Chart

- Is a top-down decomposition of a program.
- Pictorial representation as a tree structure
- Shows the breakdown of a system to its lowest manageable parts
- Shows the functional flow of the program
- Shows a logical breakdown of a problem into different steps



A Comparison of Design Tools

- **Flowcharts**

For:

- the flow of logic is easy to follow.
- many people find diagrams easier to follow than text.
- good for simple procedures.
- acts as documentation for program flow

Against:

- because arrows can be drawn from anywhere to anywhere, the flow of logic can become convoluted unless strict discipline is enforced.
- cumbersome for complex problems
- difficult for computerized construction and amendment without using drawing tools
- poor in illustrating program structure
- no provision for data definition and scoping.

- **NS Diagrams**

For:

- essentially the same benefits as flowcharts with the advantage that the absence of arrows ensures strict adherence to structured programming concepts.

Against:

- the same restrictions as for flowcharts.
- the real case against NS diagrams is that nobody really uses them as design tools;

- **Pseudocode**

For:

- approximates the format of a programming language
- easy to translate into a program.
- Easy to modify and update.
- allows for the definition and scoping of data
- Flexible and does not require the programmer to memorize any symbols
- universally accepted as a basis of algorithm specification in the absence of a specific programming language

Against:

- Some people find text harder to understand than a diagram
- No standard specifications for developing pseudocodes
- For complex problems may become quite lengthy

Structure chart

- For
 - Modularity improves system maintainability
 - Provides a means for transition from analysis to design
 - Provides a synchronous hierarchy of modules
- Against
 - Does not work well for asynchronous processes
 - Could be too large to be effectively understood with large programs.

Some Programming Paradigm

- *Imperative*: Programming with an explicit sequence of commands that update state.
 - commands show *how* the computation takes place, step by step. Each step affects the global **state** of the computation

- *Structured*: Programming with clean, goto-free, nested control structures.
 - is a kind of imperative programming where control flow is defined by nested loops, conditionals, and subroutines, rather than via gotos. Variables are generally local to blocks
 - has three control constructs **sequence**, **selection** and **iteration**.

- *Declarative*: Programming by specifying the result you want, not how to get it.
 - the programmer states only *what* the result should look like, **not** how to obtain it.

- *Object-Oriented*: Programming by defining objects that send messages to each other. Objects have their own internal (encapsulated) state and public interfaces.
 - based on the sending of *messages* to objects. Objects respond to messages by performing operations, generally called *methods*. Messages can have arguments.

- *Logic* (Rule-based): Programming by specifying a set of facts and rules. An engine infers the answers to questions.
 - programs are built by setting up relations that specify *facts* and inference *rules*, and asking whether or not something is true

An Example

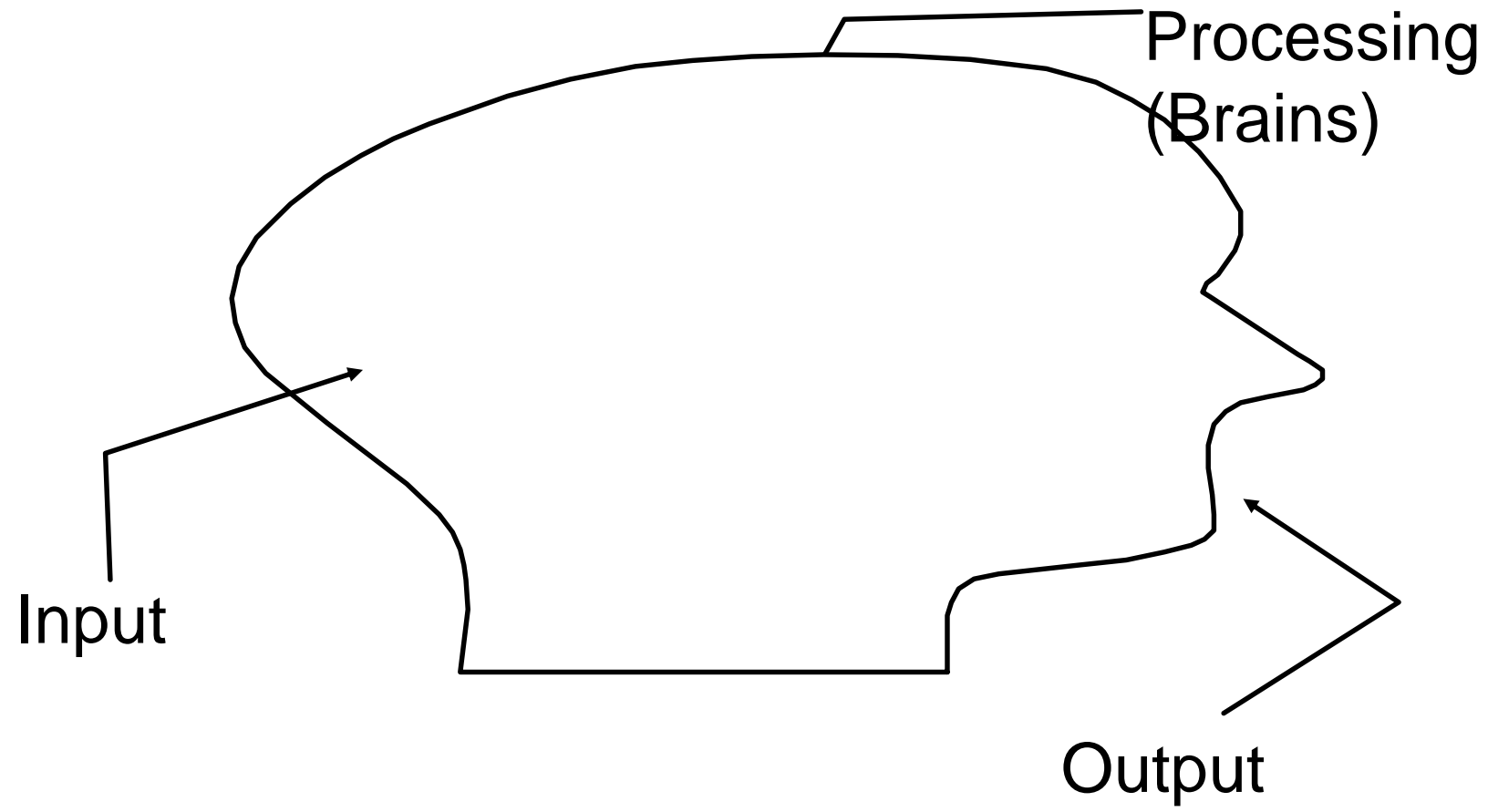
- **Problem**
- You are required to design an algorithm to calculate the *addition of two values*.

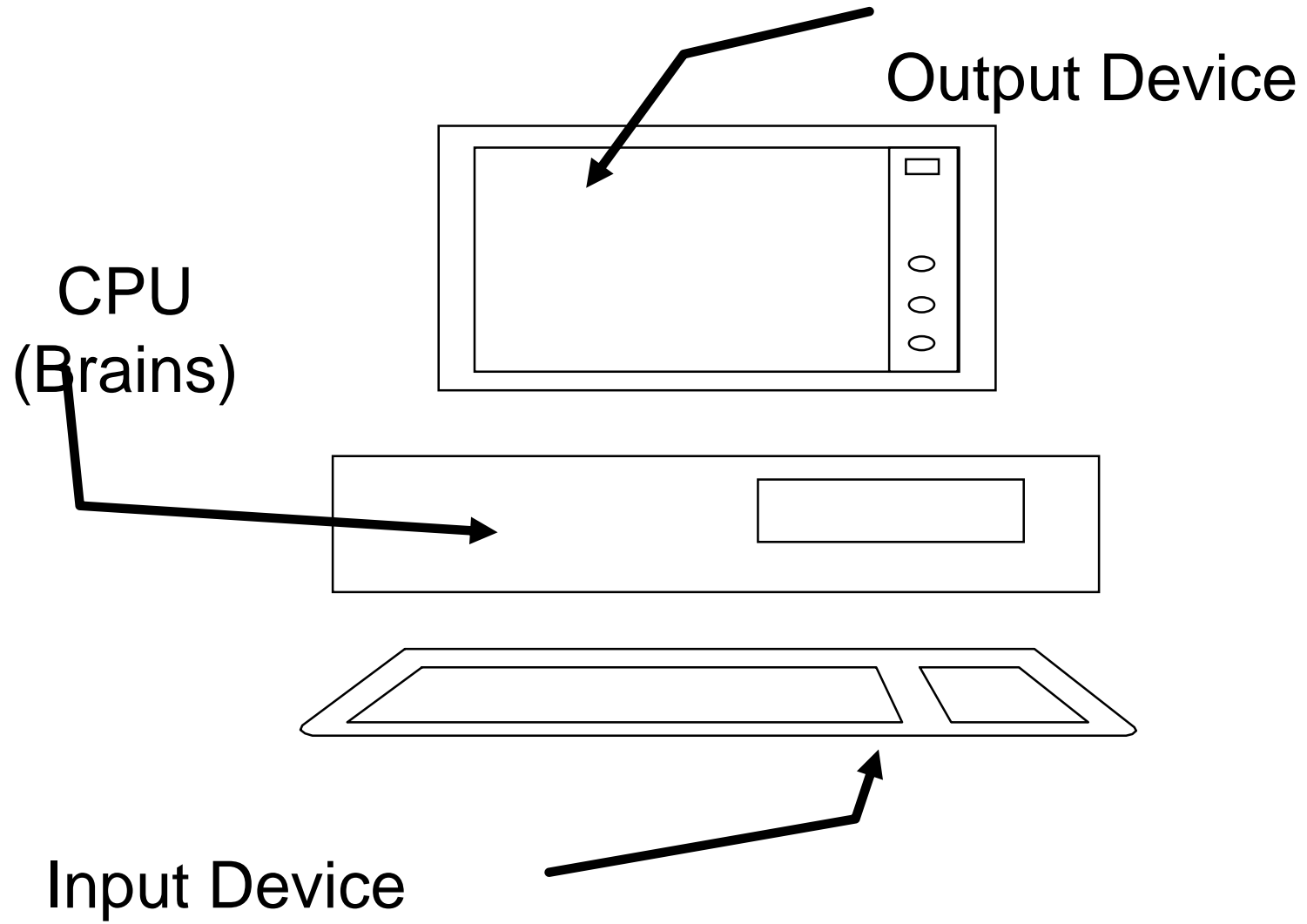
Problem Solving

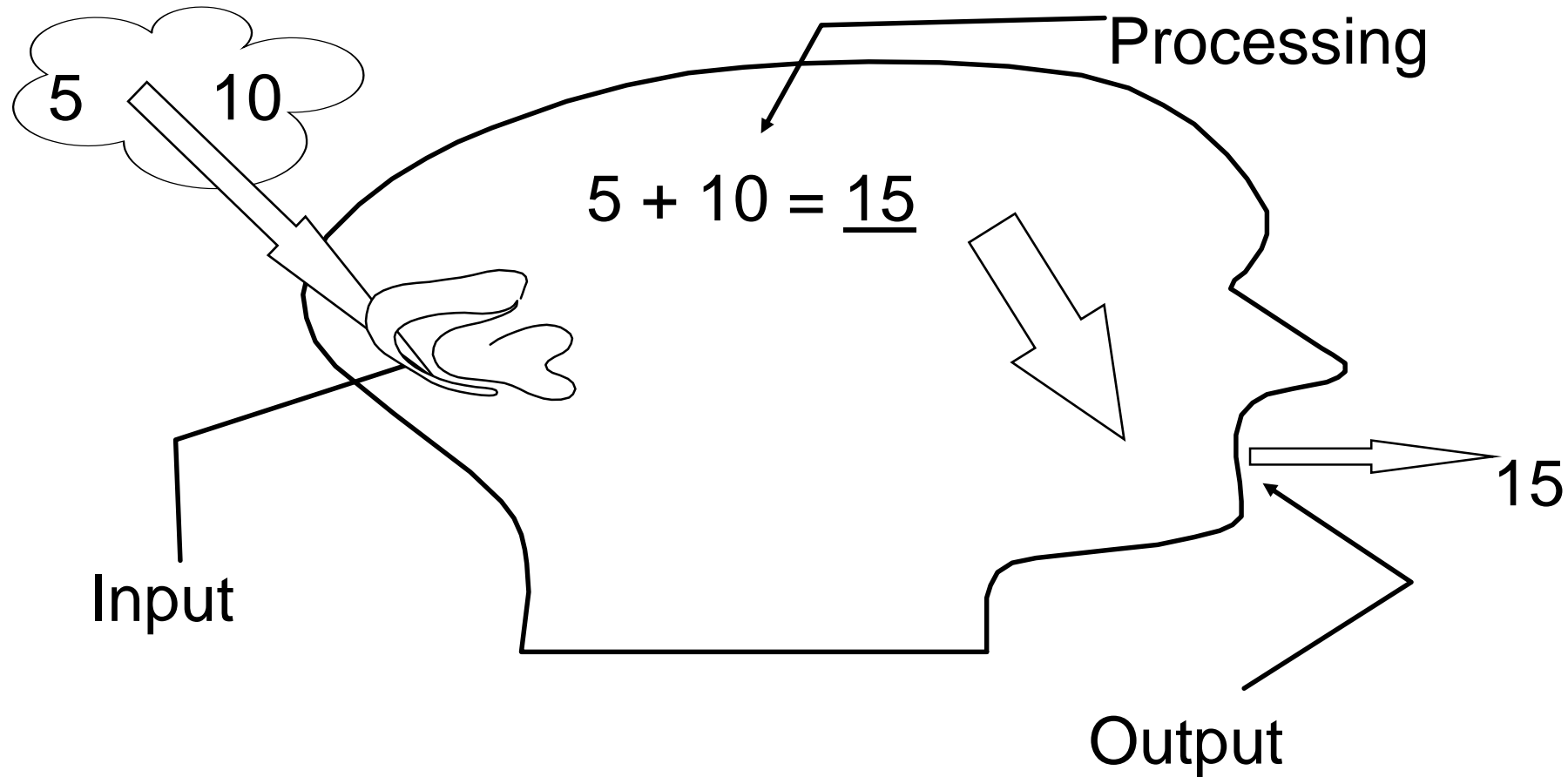
- To grapple with this problem, we have to understand the problem from the human perspective

A question to ask yourself is this,

“How Would You Calculate the Sum of Two Values?”







Let us assume we are interested in calculating the sum of 5 and 10.

Now that we have an *exact idea* about how the problem is solved, let us represent this in a clearer manner, using the *defining diagram*.

Input	Processing	Output
Value1 Value2		Sum

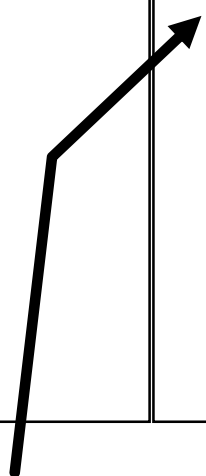
The next step is to *identify* the actual *processing steps* required to convert the *input* to become the *output*.

Input	Processing	Output
Value1 Value2	1) <u>Input</u> Value1, Value2 2) <u>Calculate</u> Sum 3) <u>Display</u> Sum	Sum

Algorithm Development

Once the defining diagram has been developed, the next logical step is to develop the *algorithm* (which is much more detailed).

Input	Processing	Output
Value1 Value2	1) Input Value1, Value2 2) Calculate Sum 3) Display Sum	Sum



The developed processing steps have to be *more detailed* in the algorithm.

The basic mathematical operators used in algorithms are as follows:-

+ addition

- subtraction

***** multiplication

/ division

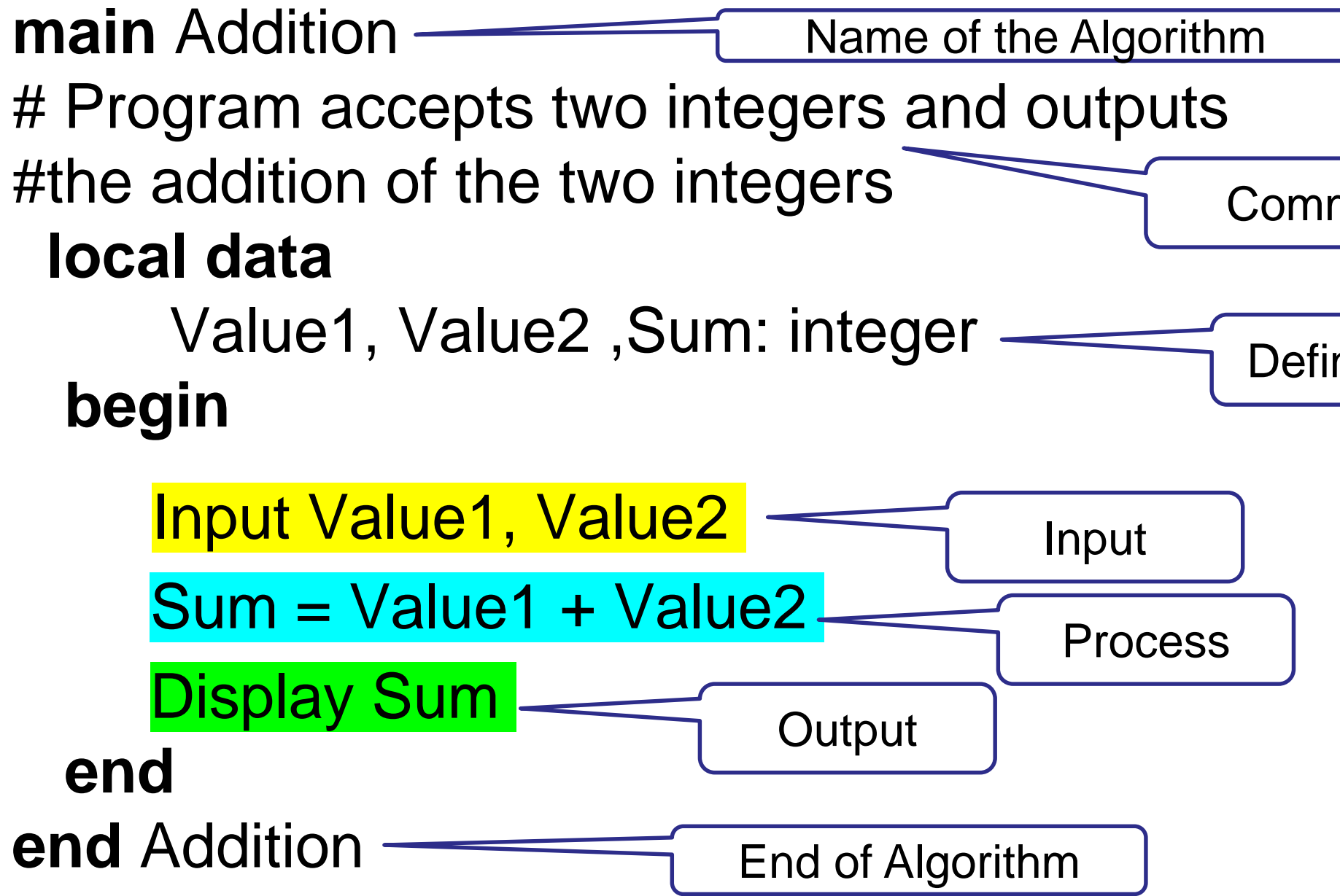
= assignment

^ exponentiation (raising to a power)

div integer division

mod remainder of integer division

()brackets for grouping calculations



main Addition

Program accepts two integers and outputs
#the addition of the two integers

local data

Value1, Value2, Sum : integer

begin

Display "Enter two values"

Input Value1, Value2

Sum = Value1 + Value2

Display "The addition is :", Sum

end

end Addition

Interaction with
the user

Interaction with
the user

Format of the main algorithm:

main GiveaName

Comments explaining the purpose

of the program

local data

Define the variables

begin

The statements are written here

This is the Main body

end

end GiveaName

Data naming

- Naming

Data names may be of any number of alphanumeric characters, the first of which must be alphabetic. They are case insensitive. **Total total** are all same.

Suggested method – multipart names have each part commence with an upper case letter. Eg. **TotalStudentTaxPayable**

Activity:

Identify the naming convention in 'C'

You should discuss this.

Explicit data definition

requires all data items to be defined prior to their use in a procedural statement. $\text{Total} = \text{Total} + \text{Counter}$

the data items Total and Counter must already have been defined within the program.

Example

local data

number1, number2, Sum : integer

- **Implicit Data Definition**

Programming languages which allow implicit definition of data items register each data name as it is encountered within the procedural statements.

While this may seem to save the programmer the trouble of defining data items in advance, it is extremely dangerous.

Data types

- Numeric data

- Integers

Integers are whole numbers and may normally be positive or negative within a certain range of values.

Counter: integer

- Real numbers

Real or floating point numbers are fractional values, and are normally able to hold greater magnitudes on both positive and negative scales than integers.

Average: real

- **Character Data**

- Character data is often referred to as text or string or alphanumeric data. It consists of digits, letters or symbols represented by the internal coding system of the computer, e.g. ASCII.

Message :string

- Character data may be a single character

Flag : character

- **Boolean Data**

- Boolean data items are used as status indicators or flags, and may contain only one of two possible values: True or False.

EndOfFile : Boolean

Data Usage

- **Variables.** Variables are data items, of any type, whose contents may change in value as the program executes. For example, in the statement:

Counter=Counter+1

- **Constants:** constants are either literal values or data items whose contents do not change as the program executes
- Declaring a named data type to be a constant.
Use **value**

pi : real value 3.142

Data Scope

- **Global:** Global data is accessible by a number of procedures/subroutine/modules.

global data

- **Local:** Local data is data defined within a procedure for access by that procedure only. It provides a means of keeping data private within any procedure thereby preventing its access and possible corruption by other procedures.

local data

- Consider the following algorithm and answer the given questions.

main TryThis

Program accepts two integers and outputs
#the substratction of the two integers

local data

number 1, number2 : integer

begin

Display “Enter two numbers

Input number 1, number2

Multiply = number 1 ** number2

Display “The answer is:”, Multilpy

end

end TryThis

1. Identify the errors and correct it.
2. What is the name of the algorithm?
3. List the variables.
4. List the types of the variables.
5. Are the variables local or global?
6. Identify the input, process and output.

- Problem1

You are required to design an algorithm to calculate the *subtraction of two values*.

Problem 1: Area of a square

A program is required to accept from the keyboard, the measurement of side of a square. The program is to output to the screen the area of the square.

Summary of main points

- Program/algorithm
- Steps in designing a program
- Types of tools used
- Advantages/disadvantages
- Programming paradigms
- Write an algorithm for a small problem