



# SWT 11012 - Fundamentals of Programming

## Introduction

---

**SL. Abdul Haleem**

MSc in IT, PGD in IT, B.Sc .Sp. (Hons) in Computer .Sc., NDRT(UniVoTec)., MCS(SL)., MSLAAS , MCSTA(ACM)

**Senior Lecturer / ICT**

**Department of Information & Communication Technology**

**Faculty of Technology**

**South Eastern University of Sri Lanka**

**2025**

# Lecture Plan

- Academic Calendar
- Time table
- Lecturer in Charge : S.L. Abdul Haleem
- Regulations & Norms for Theory Lecture
- Regulations & Norms for Practical Laboratory

# Course Examination

- Credit Size 02
- Total Lectures 30 hrs
- Lecture Hall
- Academic Time table
- Attendance 80 %
  - Minimum Present 24 hrs.
  - Maximum Absent 06 hrs.
- Lecturer in Charge
- Regulations & Norms for Theory Lecture
- Regulations & Norms for Practical Laboratory
- Any other ??

# Overview of Computer, Programming and Problem Solving

# Lecture One Outline

- Overview of Computer, Programming and Problem Solving
- What is Programming?
- Programming Life-Cycle Phases
- Programming Tools
- Statements Structures
- Sample problem
- Assignment : Submit on VLE

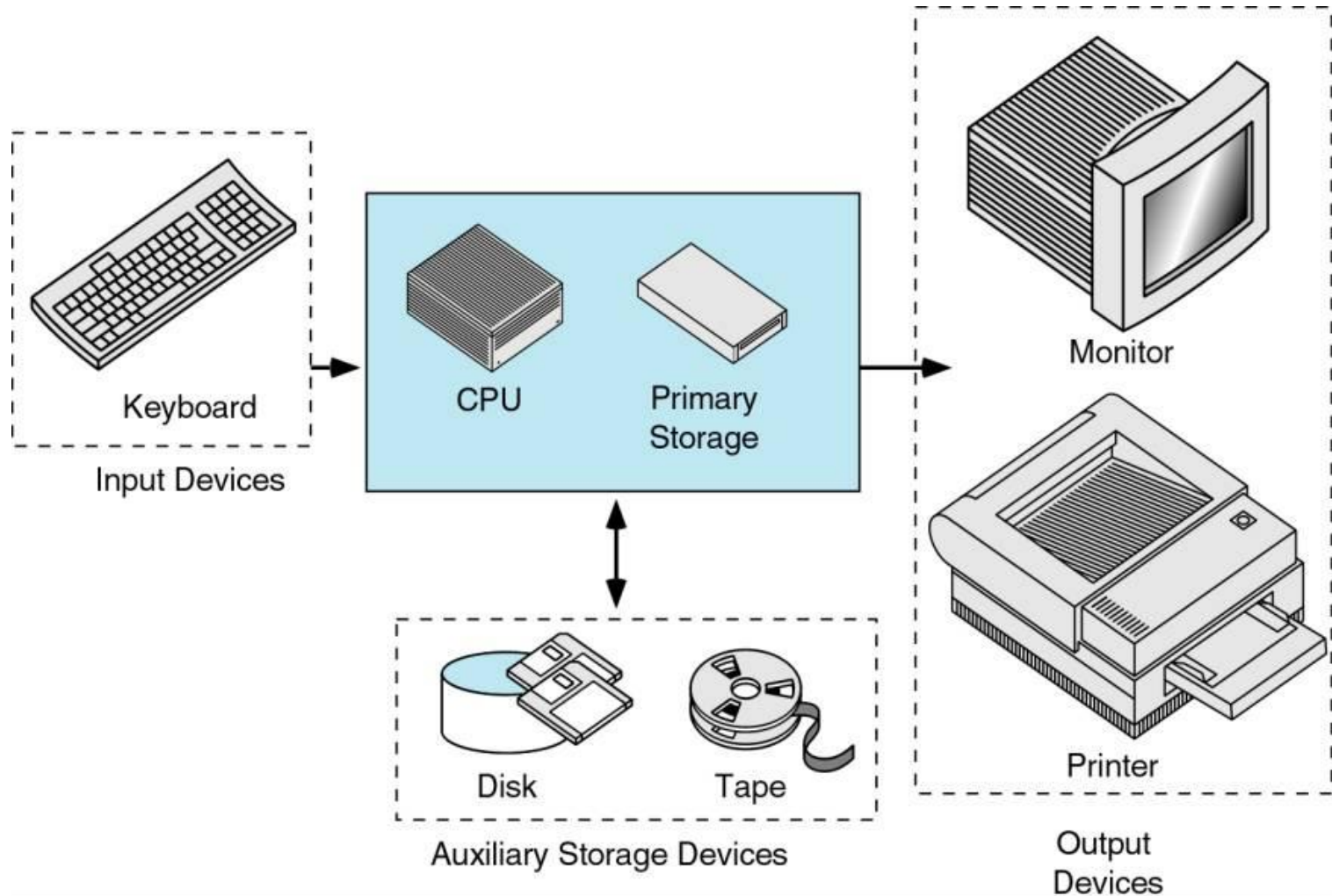
# Background

## What is a Computer?

- Device capable of performing computations and enacting logical decisions
- Perform these tasks millions and even billions of times faster than people can
- Computers process data through sets of instructions called *computer programs*
- Programs guide the computer through actions as specified by people called *computer programmers*; For this course, that will be YOU.

# Background

## Computer Components



# Communicating with the Computer

1. Machine language – low level languages, hard for humans to understand.
2. C – high level language, understood by humans, consists of instructions.



# Programming and Complicated Tasks

1. Tasks are broken down into instructions that can be expressed by a programming language
2. A program is a sequence of instructions
3. Programs can consist of only a few instructions or millions of lines of instructions

# All Programs Have in Common

1. Take data and manipulate it to produce a result
2. Input - Process - Output



- a) Input - from files, the keyboard, or other input device
- b) Output - usually to the monitor, a printer, or a file

# Performing a Task on the Computer

1. Determine **Output**
2. Identify **Input**
3. Determine **Process** necessary to turn given **Input** into desired **Output**

# Problem-Solving: Approach Like Algebra Problem

- How fast is a car traveling if it goes 50 miles in 2 hours?
  1. **Output:** a number giving the speed in miles per hour
  2. **Input:** the distance and time the car has travelled
  3. **Process:**  $\text{speed} = \text{distance} / \text{time}$

# Program Planning (1 of 2)

- A recipe is a good example of a plan
- Ingredients and amounts are determined by what you want to bake
- Ingredients are input
- The way you combine them is the processing
- What is baked is the output

## Program Planning (2 of 2)

- Always have a plan before trying to write a program.
- The more complicated the problem, the more complex the plan must be.
- Planning and testing before coding saves time.

# Terminology

1. **Programmer:** the person who solves the problem and writes the instructions for the computer
2. **User :** any person who uses the program written by the programmer
3. **Zero-Based Numbering :**
  1. In computer programming items are usually counted beginning with 0 instead of 1.
  2. That is, 0th , 1st, 2nd , ...
4. **A computer program may also be called:**
  1. Project
  2. Application
  3. Solution

# What is Programming?

- **Given a well-defined problem:**
  1. Find an algorithm to solve a problem.
  2. Express that algorithm in a way that the computer can execute it.



# Programming Life Cycle Phases

## 1. Analyze the problem.

- This involves identifying the data you have to work with it, the desired results, and any additional requirements or constraints on the solution.

## 2. Design the algorithm to solve the problem.

- An algorithm is a step-by-step procedure for solving a problem in a finite amount of time.

## 3. Implement the algorithm.

- Each algorithm is converted into one or more steps in a programming language. This process is called **CODING**.

## 4. Test and verify the completed program.

- Run the program several times using different sets of data, making sure that it works correctly for every situation in the algorithm .
- if it does not work correctly, then you must find out what is wrong with your program or algorithm and fix it--this is called **DEBUGGING**

## 5. Maintain and update the program.

- maintenance begins when your program is put into use and accounts for the majority of effort on most programs.
- **MODIFY** the program to meet changing requirements or correct errors that show up in using it.

# Algorithm (1 of 2)

A step-by-step series of instructions for solving a problem (a recipe is an example of an algorithm).

## Problem Solving Example :

- How many stamps should you use when mailing a letter?
- One rule of thumb is to use one stamp for every five sheets of paper or fraction thereof.

## Algorithm (2 of 2)

1. Request the number of sheets of paper; call it Sheets. **(input)**
2. Divide Sheets by 5. **(processing)**
3. Round the quotient up to the next highest whole number; call it Stamps. **(processing)**
4. Reply with the number Stamps. **(output)**

# Programming Tools





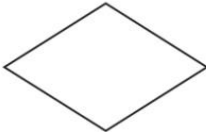


Three tools are used to convert **algorithms** into computer programs:

1. **Flowcharts** : Graphically depicts the logical steps to carry out a task and shows how the steps relate to each other.
2. **Pseudocode** : Uses English-like phrases with some terms to outline the program.
3. **Hierarchy Chart** : Shows how the different parts of a program relate to each other.
4. **Direction of Numbered N Y C Streets Algorithm**
5. **Class Average Algorithm**

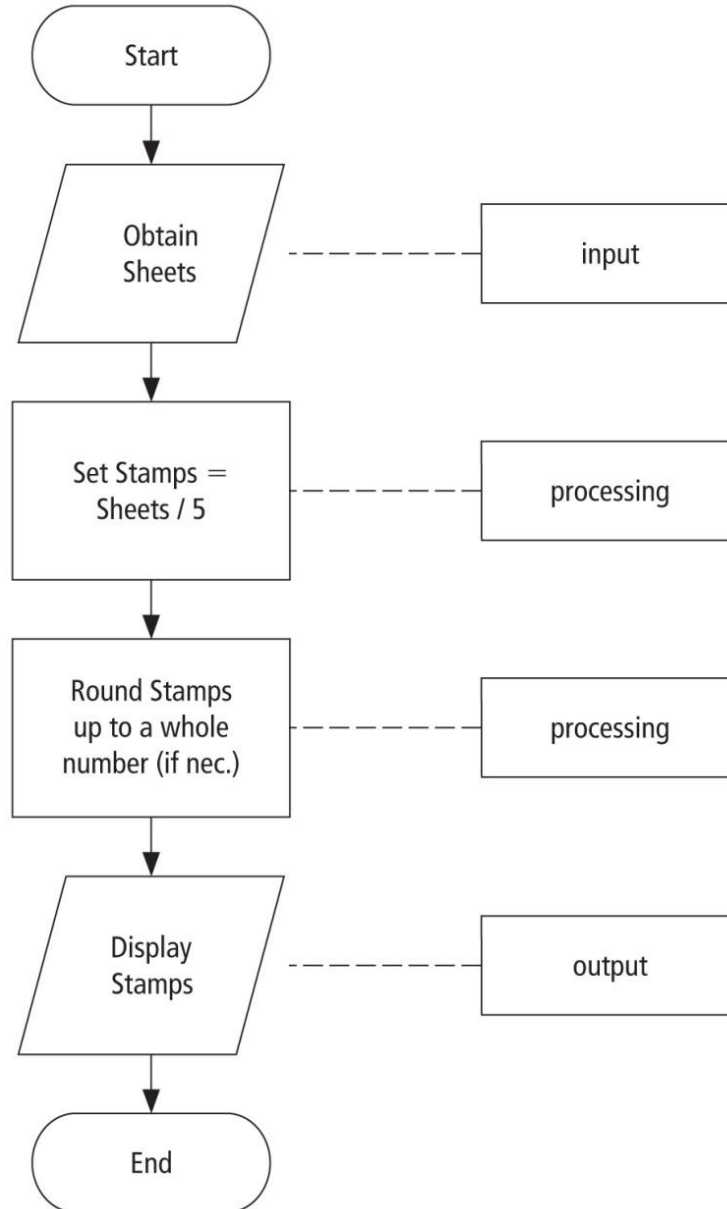
# Flowcharts (1 of 3)

Graphically depict the logical steps to carry out a task and show how the steps relate to each other.

# Flowchart : Symbols

Symbol	Name	Meaning
	<i>Flowline</i>	Used to connect symbols and indicate the flow of logic.
	<i>Terminal</i>	Used to represent the beginning (Start) or the end (End) of a task.
	<i>Input/Output</i>	Used for input and output operations. The data to be input or output is described in the parallelogram.
	<i>Processing</i>	Used for arithmetic and data-manipulation operations. The instructions are listed inside the symbol.
	<i>Decision</i>	Used for any logic or comparison operations. Unlike the input/output and processing symbols, which have one entry and one exit flowline, the decision symbol has one entry and two exit paths. The path chosen depends on whether the answer to a question is “yes” or “no.”
	<i>Connector</i>	Used to join different flowlines.
	<i>Annotation</i>	Used to provide additional information about another flowchart symbol.

# Flowchart : Example



# Pseudocode (1 of 3)

Uses English-like phrases with some terms to outline the task.



# Pseudocode Example

1. Determine the proper number of stamps for a letter
2. Read Sheets (**input**)
3. Set the number of stamps to  $\text{Sheets} / 5$  (**processing**)
4. Round the number of stamps up to the next whole number (**processing**)
5. Display the number of stamps (**output**)

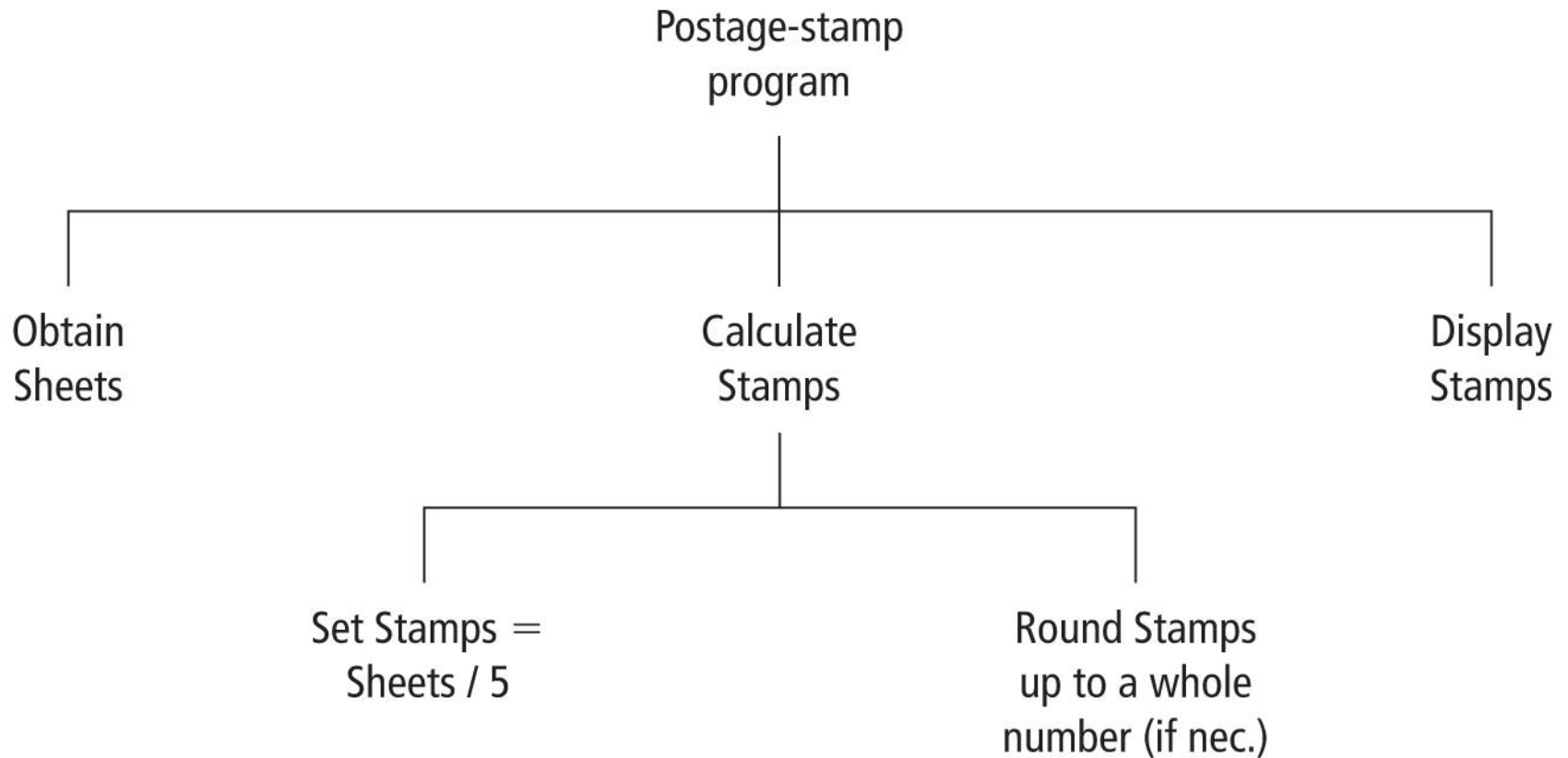
# Hierarchy Chart (1 of 3)

- Shows how the different parts of a program relate to each other

Hierarchy charts are also called

- structure charts
- H I P O (Hierarchy plus Input-Process-Output) charts
- top-down charts
- V T O C (Visual Table of Contents) charts

# Hierarchy Charts Example



# Divide-and-Conquer Method

- Used in problem solving – take a large problem and break it into smaller problems
- Solve the small problems first

# Statement Structures

- **Sequence** - execute instructions from one line to the next without skipping over any lines
- **Decision** - if the answer to a question is “Yes” then one group of instructions is executed. If the answer is “No,” then another is executed
- **Looping** - a series of instructions are executed repeatedly
- **Subprogram** is used to break the program into smaller units

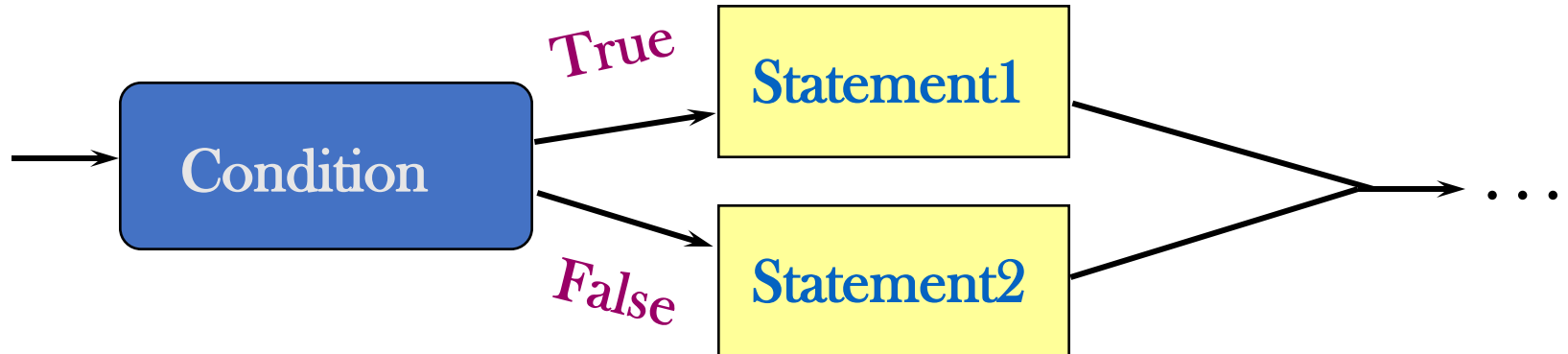
# Statements Sequences



# Control structures

## Selection (Branching)

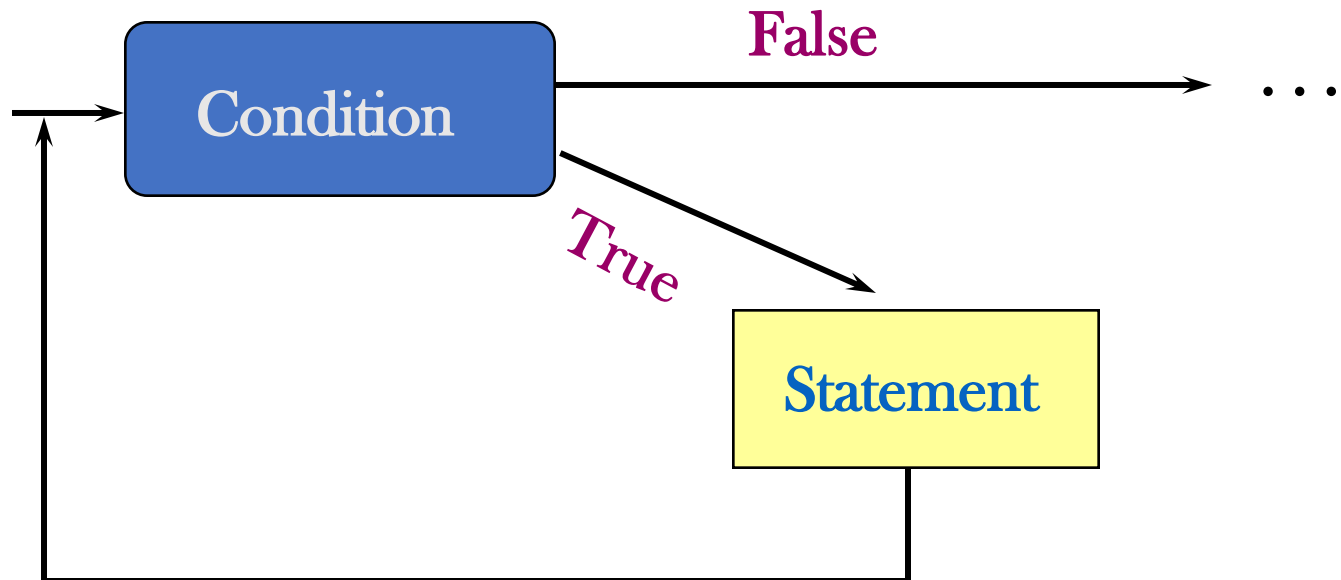
**IF Condition THEN Statement1 ELSE Statement2**



# Statements

## Loop (Repetition)

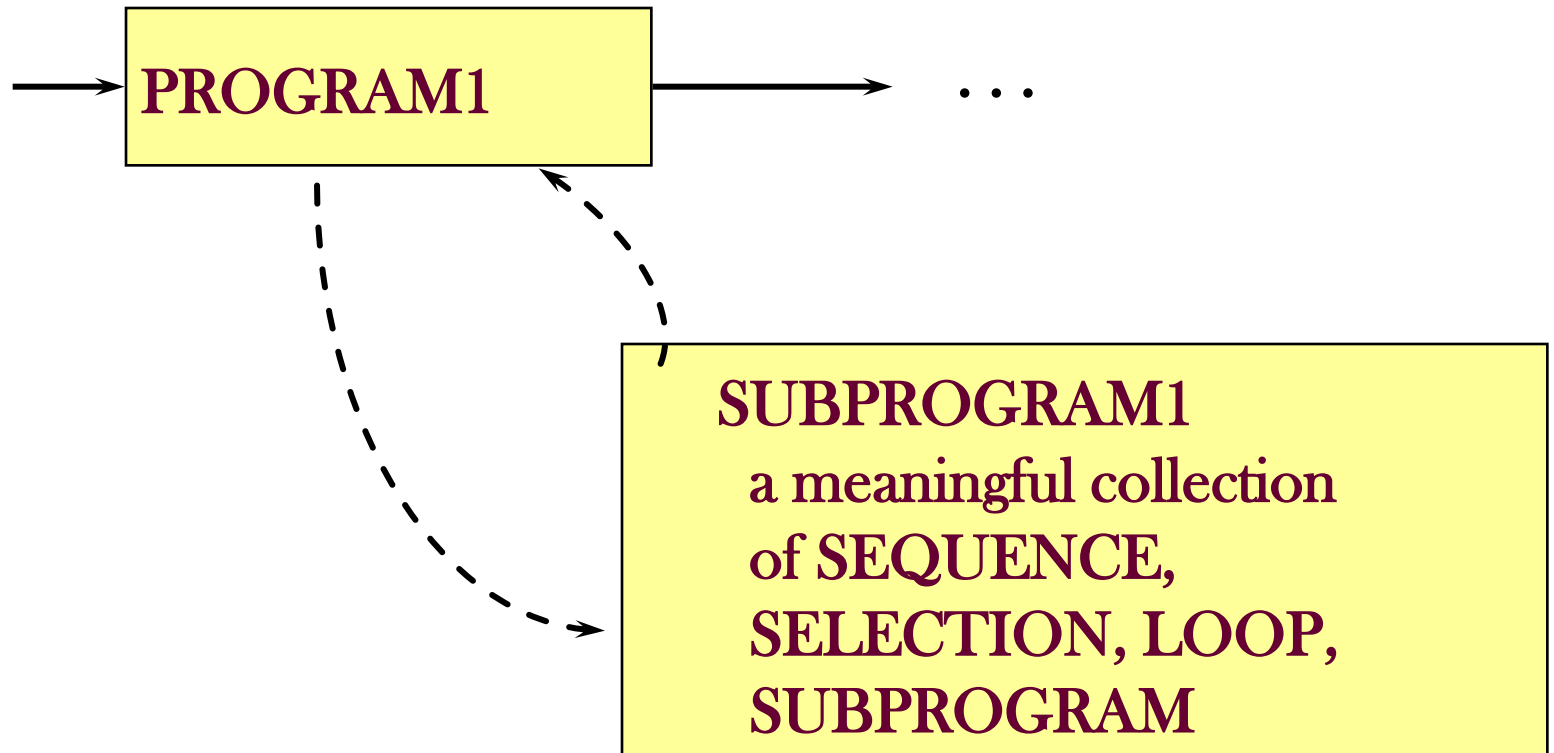
**WHILE** Condition Statement1





# Statements

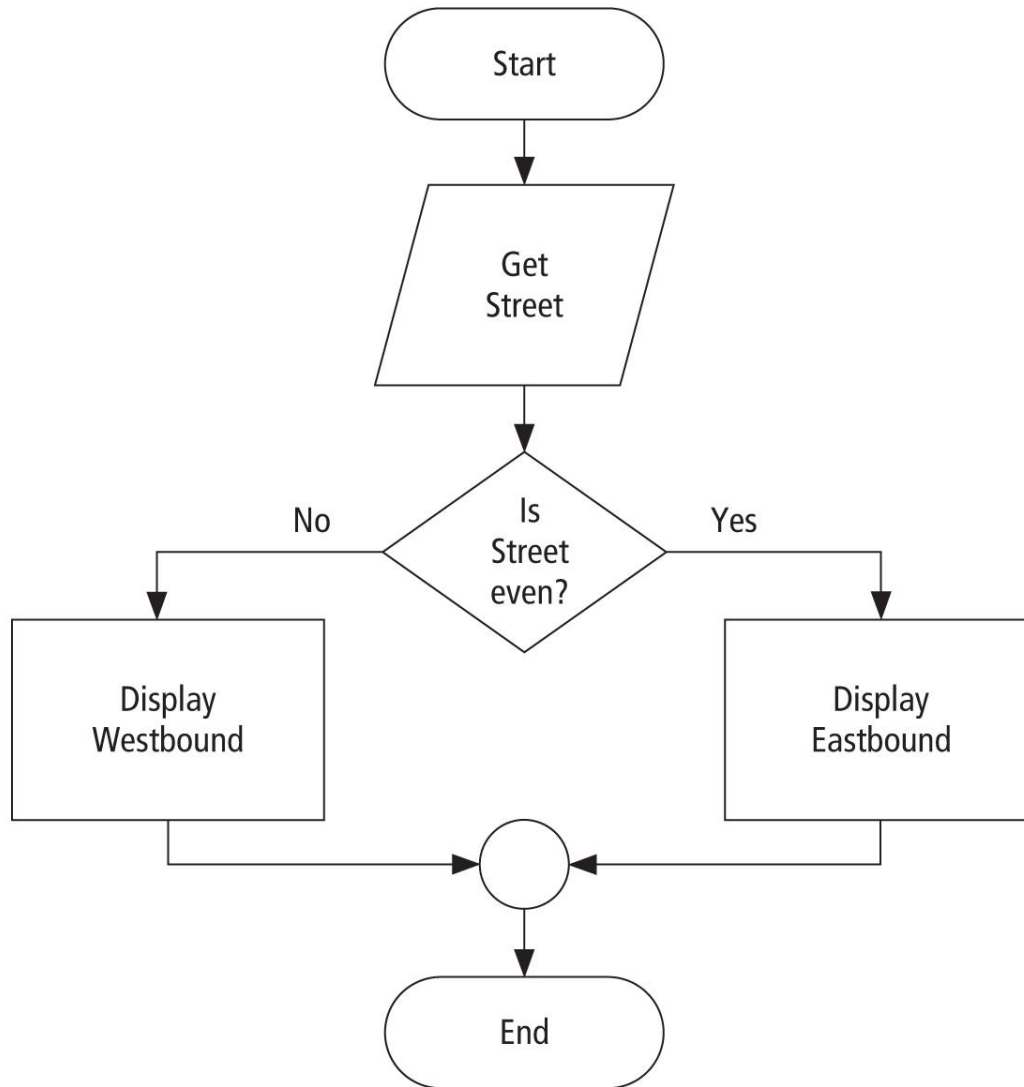
## Subprogram (Function)



# Direction of Numbered N Y C Streets Algorithm

- **Problem:** Given a street number of a one-way street in New York City, decide the direction of the street, either eastbound or westbound
- **Discussion:** in New York City even numbered streets are Eastbound, odd numbered streets are Westbound

## Flowcharts (2 of 3)



## Pseudocode (2 of 3)

- **Program:** Determine the direction of a numbered N Y C street

    Get street

    If street is even Then

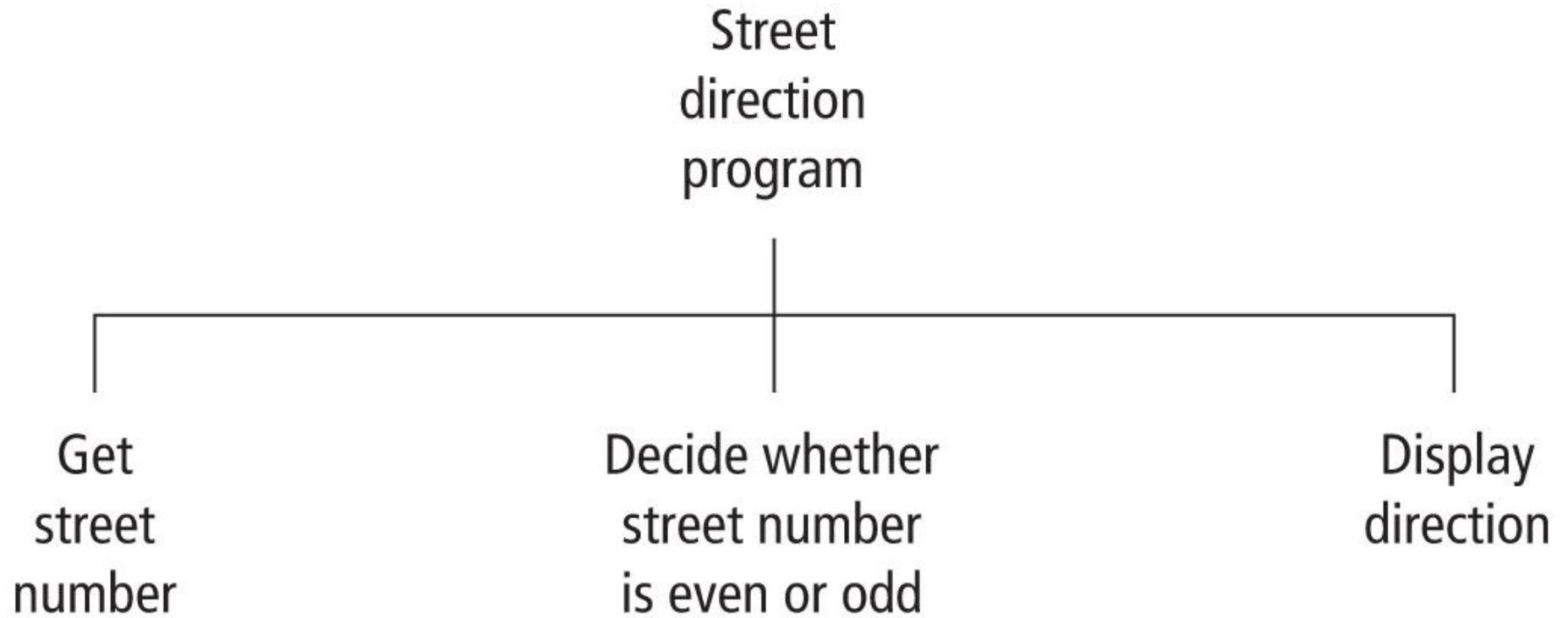
        Display Eastbound

    Else

        Display Westbound

    End If

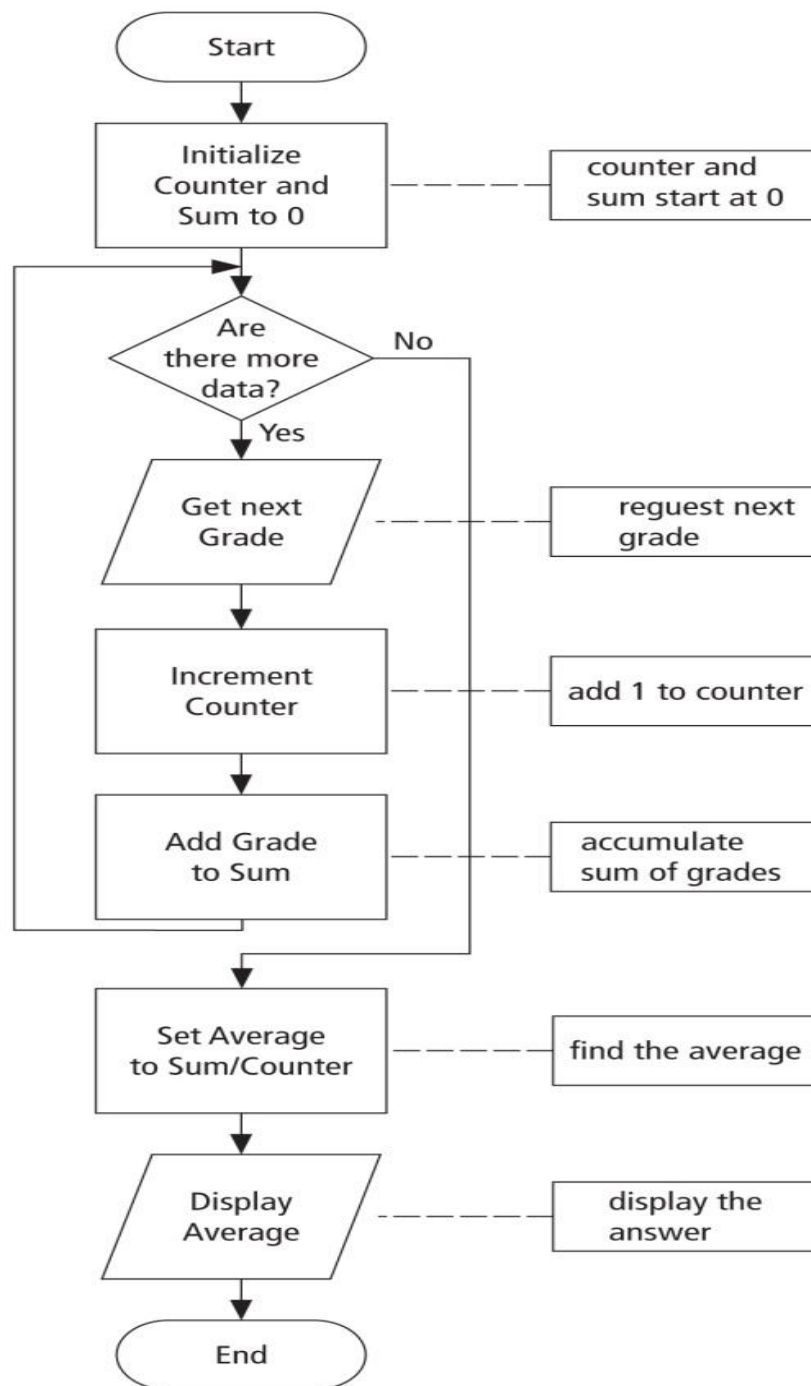
## Hierarchy Chart (2 of 3)



# Class Average Algorithm

- **Problem:** Calculate and report the average grade for a class
- **Discussion:** The average grade equals the sum of all grades divided by the number of students
  - **Input:** Student grades
  - **Processing:** Find sum of the grades; count number of students; calculate average
  - **Output:** Average grade

# Flowchart (3 of 3)



## Pseudocode (3 of 3)

**Program:** Determine average grade of a class

Initialize Counter and Sum to 0

Do While there are more data

    Get the next Grade

    Add the Grade to the Sum

    Increment the Counter

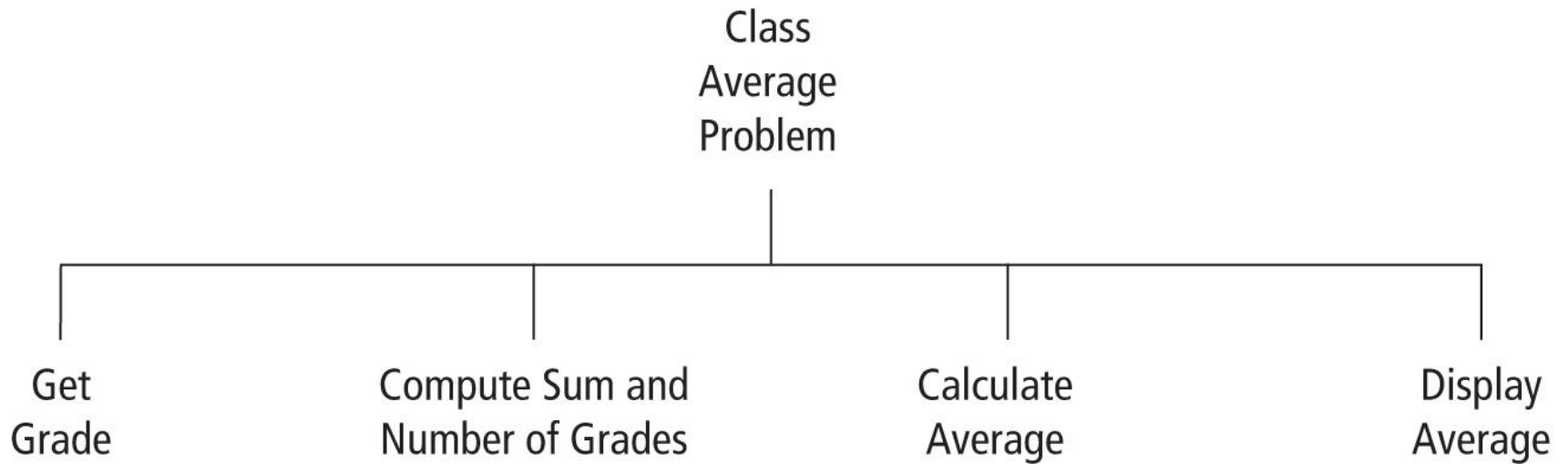
Loop

Compute  $\text{Average} = \text{Sum} / \text{Counter}$

Display Average



# Hierarchy Chart (3 of 3)



# Comments

- When tracing a flowchart, begin at the start symbol and follow the flow lines to the end symbol.
- Testing an algorithm at the flowchart stage is known as **desk checking**.
- Flowcharts, pseudocode, and hierarchy charts are program planning tools that are independent of the language being used.

# Tips and Tricks of Flowcharts

- Flowcharts are time-consuming to write and difficult to update
- For this reason, professional programmers are more likely to favor pseudocode and hierarchy charts
- Because flowcharts so clearly illustrate the logical flow of programs, they are a valuable tool in the education of programmers

# Sample Problem

## Company payroll case study

A small company needs an interactive program to figure its weekly payroll.

- The payroll clerk will input data for each employee, and each employee's wages.
- Display the total wages for the week on the screen.

# Sample Problem

A programmer needs an algorithm to determine an employee's weekly wages. How would the calculations be done by hand?

# Algorithm for Company Payroll Program

- initialize total company payroll to 0.0
- repeat this process for each employee:
  1. Get the employee's ID empNum
  2. Get the employee's hourly payRate
  3. Get the hours worked this week
  4. Calculate this week's wages
  5. Add wages to total company payroll
- write total company payroll on screen

# Weekly Wages, in General

If hours are more than 40.0, then

$$\text{wages} = (40.0 * \text{payRate}) + (\text{hours} - 40.0) * 1.5 * \text{payRate}$$

otherwise,

$$\text{wages} = \text{hours} * \text{payRate}$$

# Questions

Draw the flowchart, pseudocode and hierarchy chart for the above problems.