Programming Fundamentals I

Chapter 1: *An Overview of Computers and Programming Languages*

Dr. Adriana Badulescu

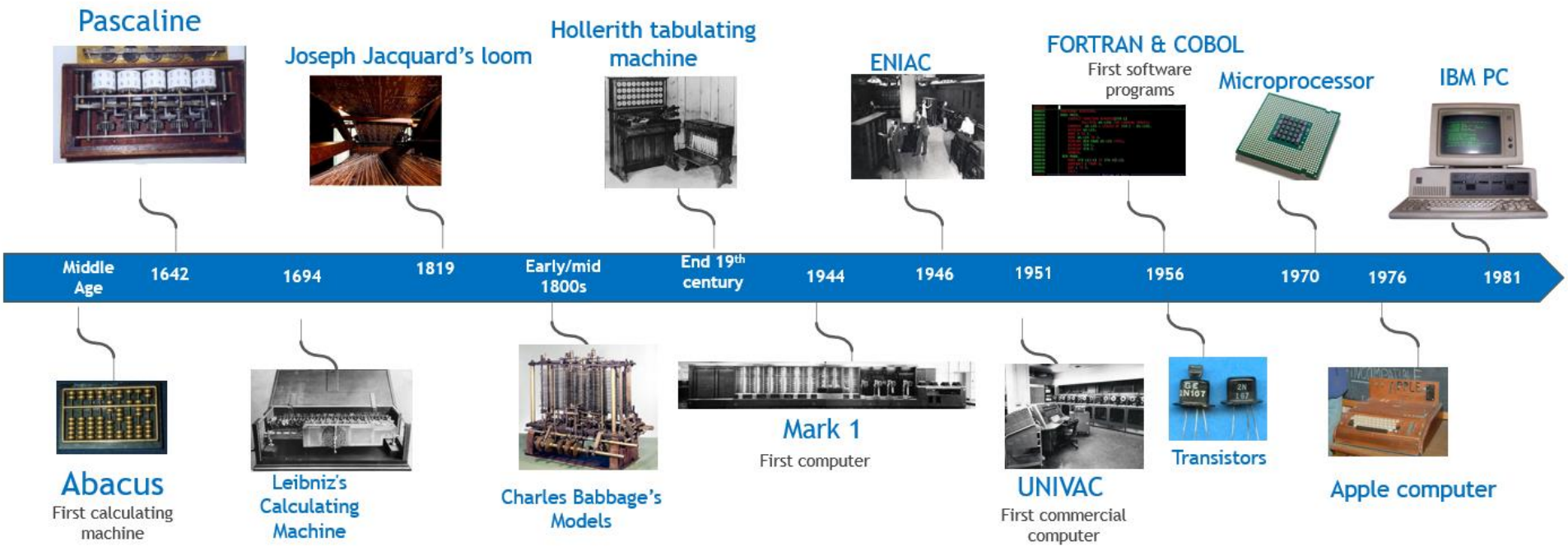
Objectives

- Learn about computer and programming major milestones
- Learn about different types of computers
- Explore the hardware and software components of a computer
- Learn about the language of a computer
- Learn about the evolution of programming languages
- Examine high-level programming languages
- Become aware of structured design and object-oriented design programming methodologies
- Become aware of Standard C++ and its history
- Understand the programming process

Introduction

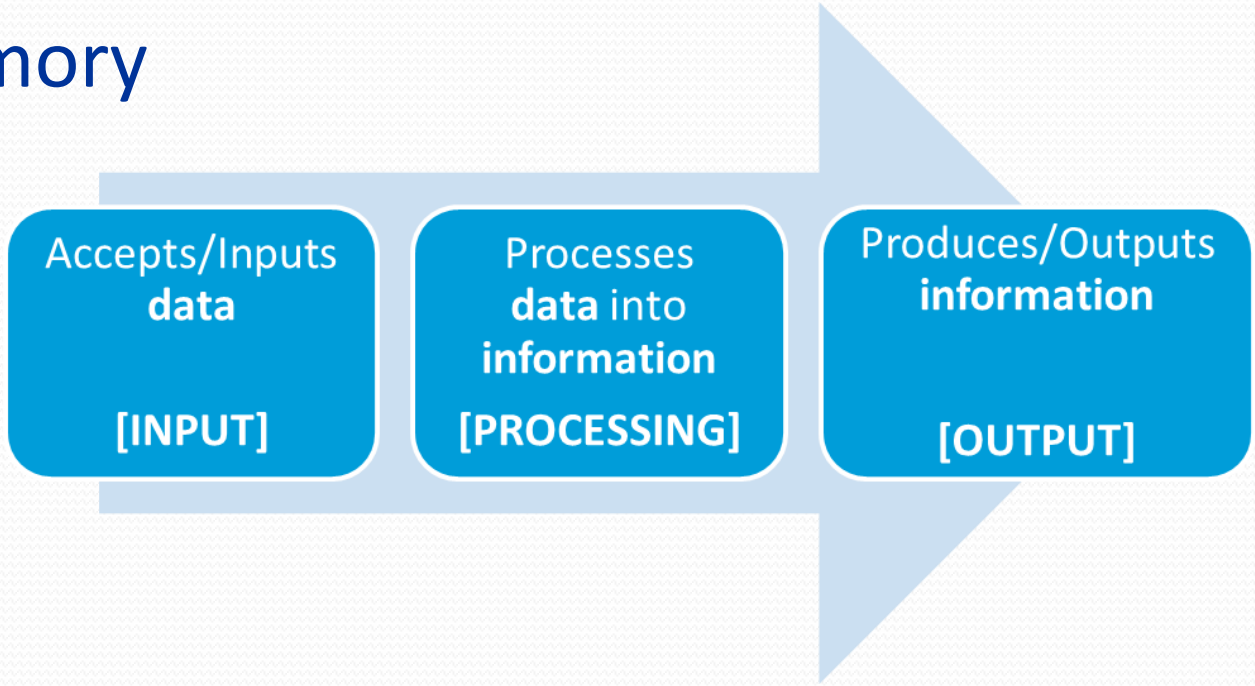
- Without software, the computer is useless
- Software developed with programming languages
  - C++ is a programming language
- C++ suited for a wide variety of programming tasks
- Before programming, it is useful to understand terminology and computer components

A Brief Overview of the History of Computers



Computer Components and Operations

- A **computer** is an electronic device, operating under the control of instructions stored in its own memory



Information Processing Cycle

Computer Components and Operations

- **INPUT**
  - Hardware devices, such as keyboards and mice, perform input operations
  - **Data,** or facts, enter the computer system through input devices
- **PROCESSING**
  - Processing data items may involve organizing them, checking them for accuracy, or performing mathematical operations on them
  - The hardware that performs these tasks is the **central processing unit**, or **CPU**
- **OUTPUT**
  - Sending information resulting from processing to a printer or monitor so people can view, interpret, and use the results
  - Can be stored on disks or flash media for later retrieval

Categories/Types of Computers

Micro computers/ personal computers

Desktop Computer

Notebook/Laptop Computer

Tablet PC

Handheld Computer

PDA

Smart Phone

Game Console

Embedded computer

Server

Midsize computers

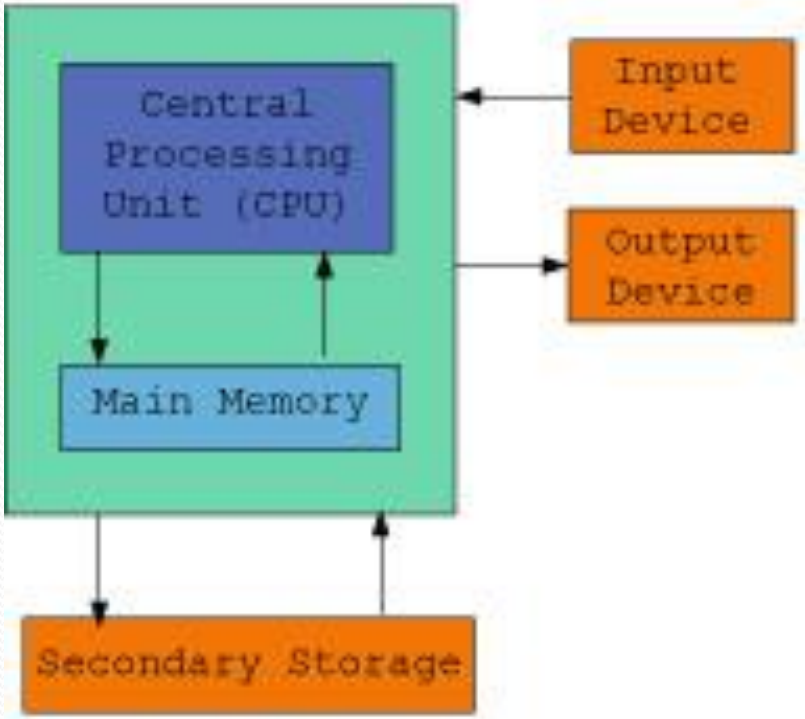
Mainframe Computer

Supercomputer

Mainframes computers

Elements of a Computer System

- Hardware
  - Central Processing Unit (CPU)
  - Main memory (primary storage): RAM
  - Secondary storage
  - Input/Output devices
- Software

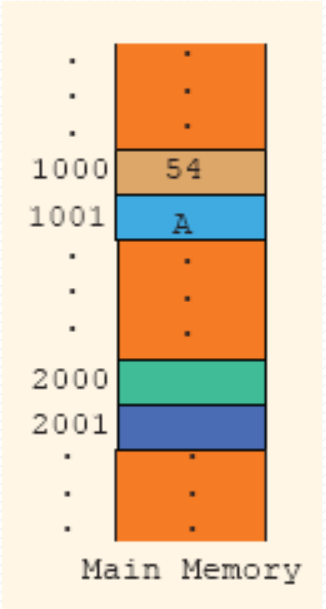


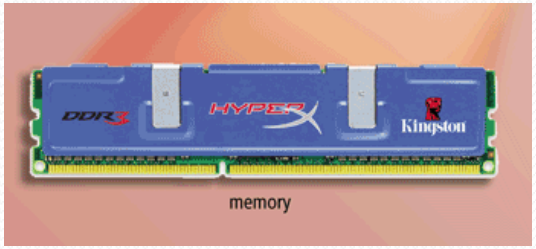
Central Processing Unit

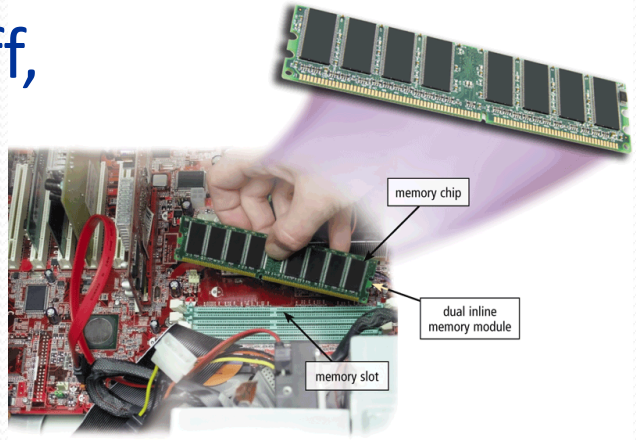
- The **processor** or the **central processing unit** (**CPU**), interprets and carries out the basic instructions that operate a computer (Control Unit) and arithmetic and logical operations (Arithmetic Logic Unit)
- Brain of the computer
- One of the most expensive piece of hardware
- Multiple Cores allows for parallel processing

Main Memory

- **Random access memory (RAM)**
  - Directly connected to the CPU
  - All programs must be loaded into main memory before they can be executed
  - All data must be brought into main memory before it can be manipulated
  - When computer power is turned off, everything in main memory is lost





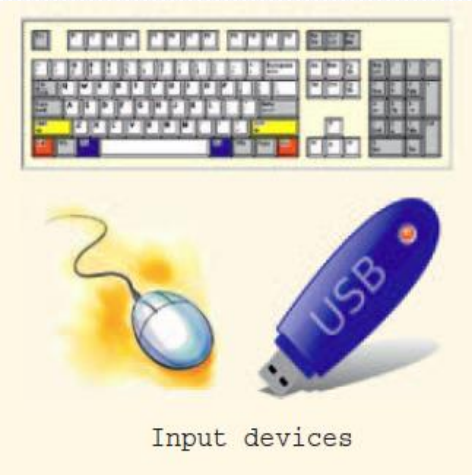


Secondary Storage

- **Secondary storage**: device that stores information permanently
- Examples of secondary storage:
  - Hard disks
  - Flash drives
  - Floppy disks
  - Zip disks
  - CD-ROMs
  - Tapes

Input/Output Devices

- **Input devices** feed data and programs into computers
  - Keyboard
  - Mouse
  - Secondary storage
- **Output devices** display results
  - Monitor
  - Printer
  - Secondary storage



Input devices

Output devices

Computer Software

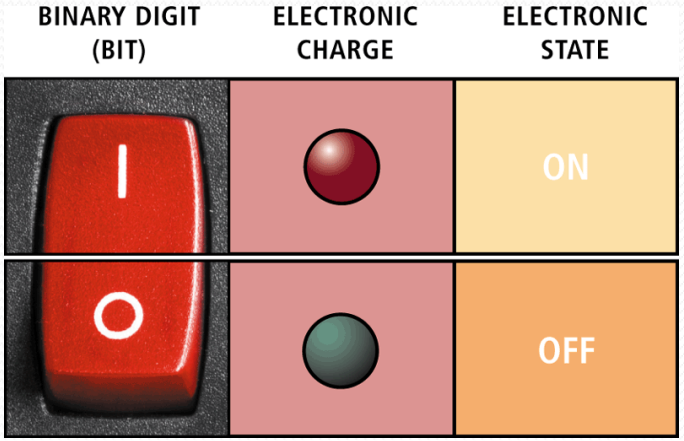
- **Software**, also called a **program**, consists of a series of instructions that tells the computer what tasks to perform and how to perform them
  - **System software -** programs that coordinates all activities among computer hardware devices and allow the user to perform maintenance-type tasks usually related to managing a computer, its devices or its programs
    - Example: operating system, disk defragmenter, etc
  - **Application software** - programs that perform a specific task and are designed to make users more productive
    - Example: word processor, programming languages, games, etc.

Operating Systems

- An **operating system (OS)** is a set of programs containing instructions that work together to coordinate all the activities among computer hardware resources

The Language of a Computers

- A computer is an electronic device
- **Analog signals**: continuous wave forms
- **Digital signals**: represent information as a sequence of 0s and 1s (binary code or binary number)
- **Machine language**: language of a computer
- **Binary digit (bit)**: the digit 0 or 1
- **Byte**: a sequence of eight bits



The Language of a Computers

- **ASCII** (American Standard Code for Information Interchange)
  - the most widely used coding scheme to represent data
  - 128 characters (0 to 127), extended version: 256 characters
- **EBCDIC** (Extended Binary Coded Decimal Interchange Code)
  - Used by IBM
  - 256 characters (1 byte)
- **Unicode**
  - 65536 characters (2 bytes)
  - capable of representing all world's languages

The Language of a Computers

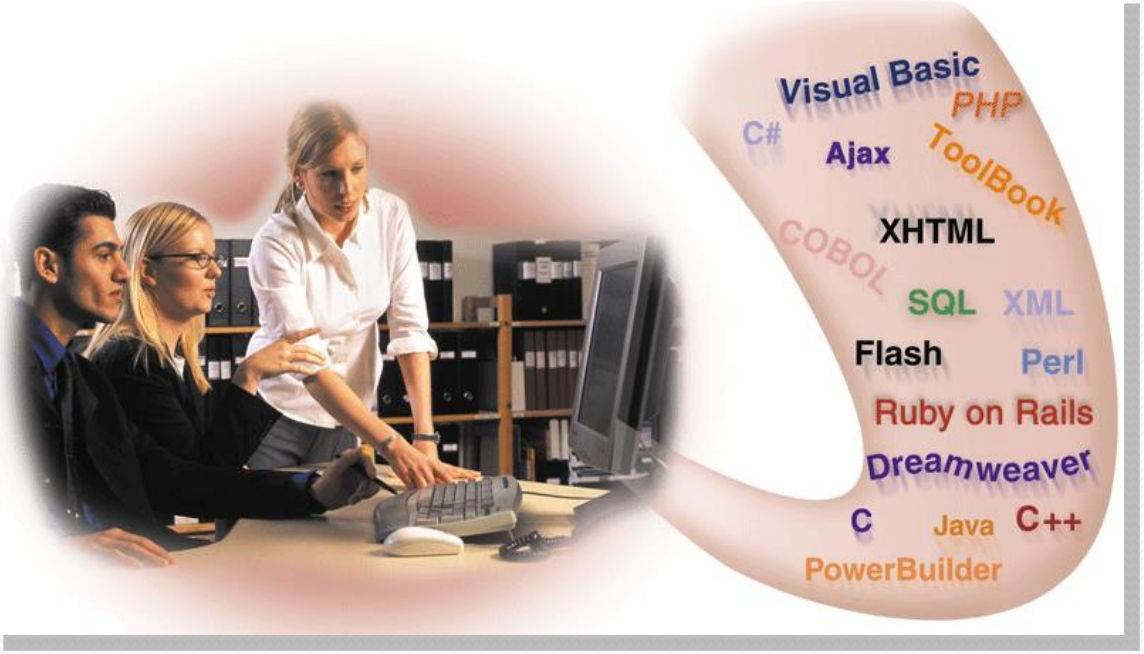
Symbol	ASCII Decimal	ASCII Binary	ASCII Hexadecimal	Extended ASCII	EBCIDIC	UNICODE Hexadecimal
A	65	01000001	41		193	0041
Z	90	01011010	5A		233	005A
a	97	01100001	61		129	0061
z	122	01111010	7A		169	007A
0	48	00110000	30		240	0030
9	57	00111001	39		249	0039
~	126	01111110	7E		161	007E
á				160		00E1
ș						0219
α						03B1
∞						221E
ئ						0626

Storage / Memory Sizes

Unit	Symbol	Bits/Bytes	Approximation
Byte		8 bits	
Kilobyte	KB	2^{10} bytes = 1024 bytes	10^3 bytes = 1,000 bytes
Megabyte	MB	1024 KB = 2^{20} bytes = 1,048,576 bytes	10^6 bytes = 1,000,000 bytes
Gigabyte	GB	1024 MB = 2^{30} bytes = 1,073,741,824 bytes	10^9 bytes = 1,000,000,000 bytes
Terabyte	TB	1024 GB = 2^{40} bytes = 1,099,511,627,776 bytes	10^{12} bytes = 1,000,000,000,000 bytes
Petabyte	PB	1024 TB = 2^{50} bytes = 1,125,899,906,842,624 bytes	10^{15} bytes = 1,000,000,000,000,000 bytes
Exabyte	EB	1024 PB = 2^{60} bytes = 1,152,921,504,606,846,976 bytes	10^{18} bytes = 1,000,000,000,000,000,000 bytes
Zettabyte	ZB	1024 EB = 2^{70} bytes = 1,180,591,620,717,411,303,424 bytes	10^{21} bytes = 1,000,000,000,000,000,000,000 bytes

Computer Programs

- **Computer program** - series of instructions that directs computer to perform tasks
- **Programming language** - used to communicate instructions to a computer



Programming Languages

Low-level language

- **Machine-dependent** runs only on one type of computer
- Machine and assembly languages are low-level
- Programmers need to know the instruction set of the microprocessor
- First-generation language (1GL)
- Second-generation language (2GL)

High-level language

- Often **machine-independent** can run on many different types of computers and operating systems
- Programs (source code) are develop faster, have fewer errors, and are easier to read
- Third-generation language (3GL)
- Fourth-generation language (4GL)
- Fifth-generation language (5GL)

Machine Language

- Machine language
  - Only language computer directly recognizes
  - Uses a series of binary digits (1s and 0s) with a combination of numbers and letters that represent binary digits (binary code)
  - Early computers were programmed in machine language
  - To calculate the `wages` from `rate` and `hours`:

```
100100 — Load
010001 — Memory address to load from
100110 — Multiply
010010 — Memory address to get number to multiply with
100010 — Store
010011 — Memory address to store the result
```

Assembly Language

- **Assembly language**
  - Instructions made up of symbolic instruction codes, meaningful abbreviations and codes
  - Source program contains code to be converted to machine language by an **assembler**

Machine Language	Assembly Language
100100	LOAD
100010	STOR
100110	MULT
100101	ADD
100011	SUB

- To calculate the `wages` from `rate` and `hours`:

```
LOAD  rate
MULT   hours
STOR   wages
```

Procedural Languages

- Programmer writes instructions (**source program**) that tell computer what to accomplish and how to do it
- Uses series of English-like words to write instructions
- Often called **third-generation language (3GL)**
- To calculate the `wages` from `rate` and `hours`:
  - `wages = rate * hours;`
- Converting the source program into machine language
  - **Compiler -** translates an entire program before executing it
  - **Interpreter** - converts and executes one code statement at a time

Examples:

C, C++, Pascal, Basic

Procedural Languages

- Early procedural languages had line numbers and “goto” statements
  - **Spaghetti code**
- Each line would translate to several lines of assembly or machine code

```
      IF (IA) 777, 777, 701
  701 IF (IB) 777, 777, 702
  702 IF (IC) 777, 777, 703
  703 IF (IA+IB-IC) 777,777,704
  704 IF (IA+IC-IB) 777,777,705
  705 IF (IB+IC-IA) 777,777,799
  777 STOP 1
C USING HERON'S FORMULA WE CALCULATE THE
C AREA OF THE TRIANGLE
  799 S = FLOATF (IA + IB + IC) / 2.0
      AREA = SQRT( S * (S - FLOATF(IA)) * (S -
FLOATF(IB)) *
     +     (S - FLOATF(IC)))
      WRITE OUTPUT TAPE 6, 601, IA, IB, IC, AREA
  601 FORMAT (4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H
AREA= ,F10.2,
     +        13H SQUARE UNITS)
      STOP
      END
```

Examples: FORTRAN, COBOL, Basic

Structured Languages

- Use structured design - dividing a problem into smaller subproblems
- The structured design approach is also called top-down (or bottom-up) design,, stepwise refinement, or modular programming
  - No more need for GOTO or line numbers (spaghetti code)
- Introduce new concepts like data structure and function

- Data structures:

```
CircleData: Radius
RectangleData: Width, Height
```

- Functions:

```
CalculateCircleArea (CircleData)
CalculateRectangleArea
   (RectangleData)
```

- Main code:

```
If shape is circle
then
call CalculateCircleArea (shape)
else
Call CalculateRectangleArea (shape)
```

Examples:
C,
Pascal

Object-Oriented Programming (OOP) Languages

- Used to implement object-oriented design (OOD)
- Object is a single unit component that contains data and operations on data
- Major benefit is ability to reuse existing objects
- The object-oriented design is used with structured design

```
Class CircleClass
{
  Radius
  CalculateArea()
}

Class RectangleClass
{
  Width, Height
  CalculateArea()
}
CircleClass ShapeObject;
ShapeObject.CalculateArea();
```

Examples:
C++,
Java,
C#,
Visual Basic

Nonprocedural Programming Languages

- **Nonprocedural Programming Language**
  The programmer writes English-like instructions or interacts with a visual environment to retrieve data from files or a database
- **Program Development Tools**
  User-friendly environment designed to assist both programmers and users in creating programs

Example: SQL

Visual Programming Languages

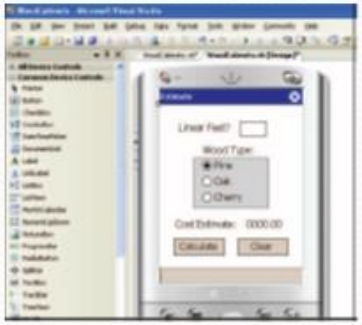
- Visual programming environment (VPE) allows developers to drag and drop objects to build programs
- Visual Programming Languages provide visual or graphical interface for creating source code
- Event-driven - checks for and responds to set of events (actions to which programs can respond to)

Examples:
Java,
FLEX,
C#,
Visual C++,
Visual C#,
Visual Basic,
Delphi

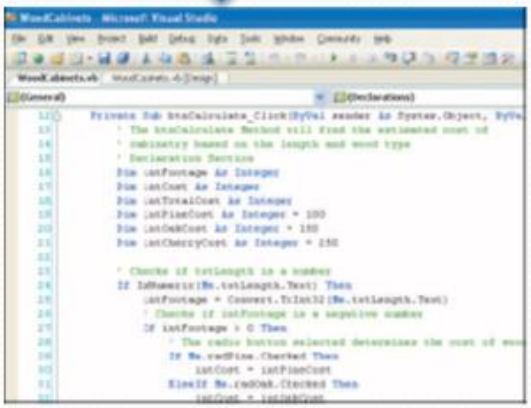
Visual Programming Languages

- Visual Studio
  - .NET is set of technologies that allows program to run on Internet
  - Comprised of Visual Basic, Visual C++, Visual C#, and Visual J#

Step 1. The developer designs the user interface.



Step 2. The developer assigns properties to each object on the form.



Step 3. The developer writes code to define the action of each command button.

Step 4. The developer tests the program.

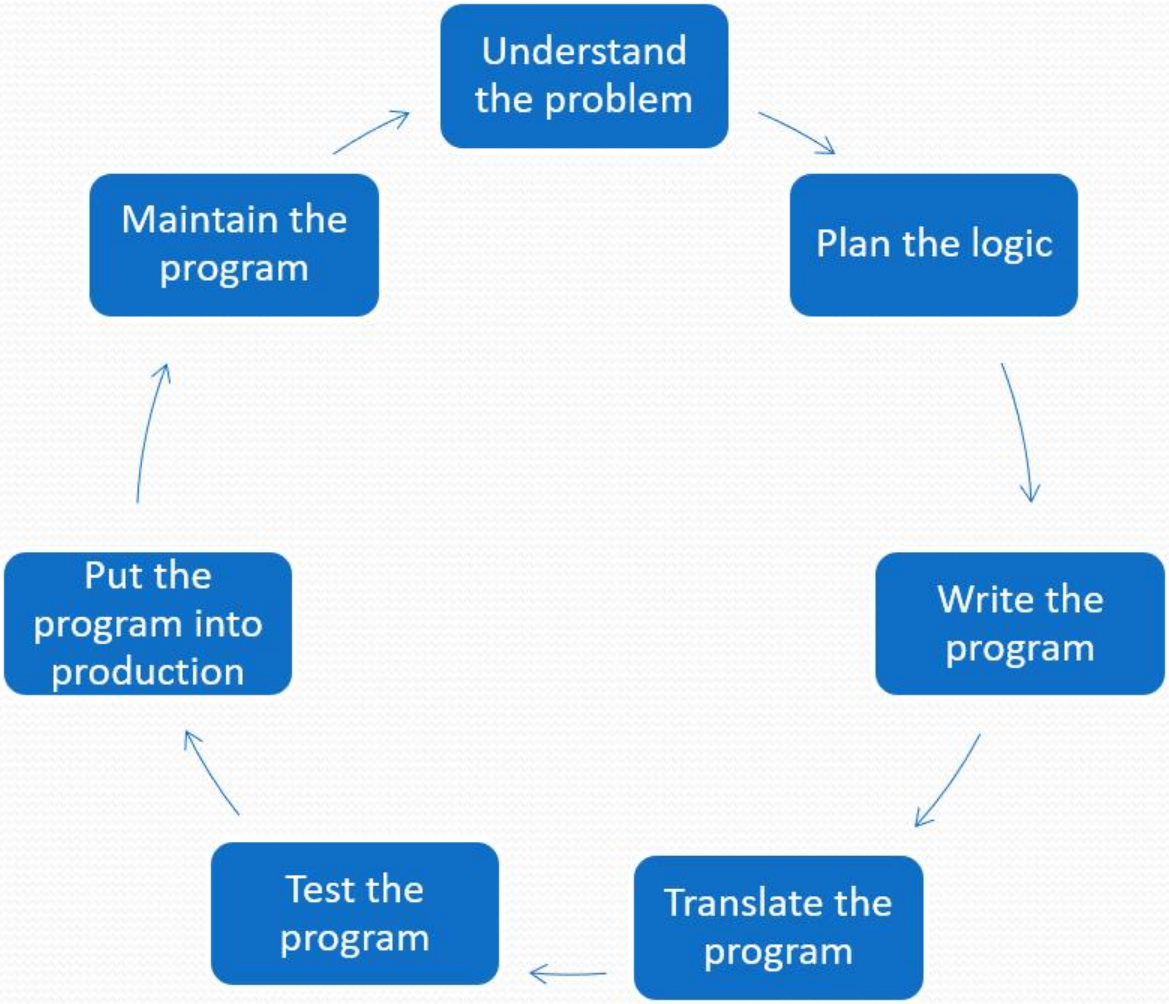
History of C++ Programming Language

- C++ evolved from C
- C++ designed by Bjarne Stroustrup at Bell Laboratories in early 1980s
- C++ programs were not always portable from one compiler to another
- In mid-1998, ANSI/ISO C++ language standards were approved

Understanding the Programming Process

- Programming is a process of **problem solving**
- A problem-solving technique:
  - Analyze the problem
  - Outline the problem requirements
  - Design steps (algorithm) to solve the problem
- **Algorithm** – a step-by-step problem-solving process that reaches a solution in a finite amount of time

Understanding the Programming Process



The Program Development Cycle

The Problem Analysis–Coding–Execution Cycle

1. Analyze the problem
   - Outline the problem
   - Outline the problem requirements
   - Design steps (algorithm) to solve the problem
2. Implement the algorithm
   - Implement the algorithm in code
   - Verify that the algorithm works
3. Maintain the program
   - Use and modify the program if the problem domain changes

The Problem Analysis–Coding–Execution Cycle

- Thoroughly understand the problem
- Understand problem requirements (input, output, interface, data manipulation)
- If the problem is complex, divide it into subproblems and analyze each subproblem as above
- If problem was broken into subproblems, design algorithms for each subproblem
- Check the correctness of algorithm using sample test data. Some mathematical analysis might be required.
- Once the algorithm is designed and correctness verified, write the equivalent code in high-level language

The Problem Analysis–Coding–Execution Cycle

- Enter the program using text editor and run code through compiler
- If compiler generates errors, look at code and remove errors. Run code again through compiler
- If there are no syntax errors, the compiler generates equivalent machine code
- Linker links machine code with system resources
- Once compiled and linked, loader can place program into main memory for execution
- The final step is to execute the program
- Compiler guarantees that the program follows the rules of the language. Does not guarantee that the program will run correctly

Example 1

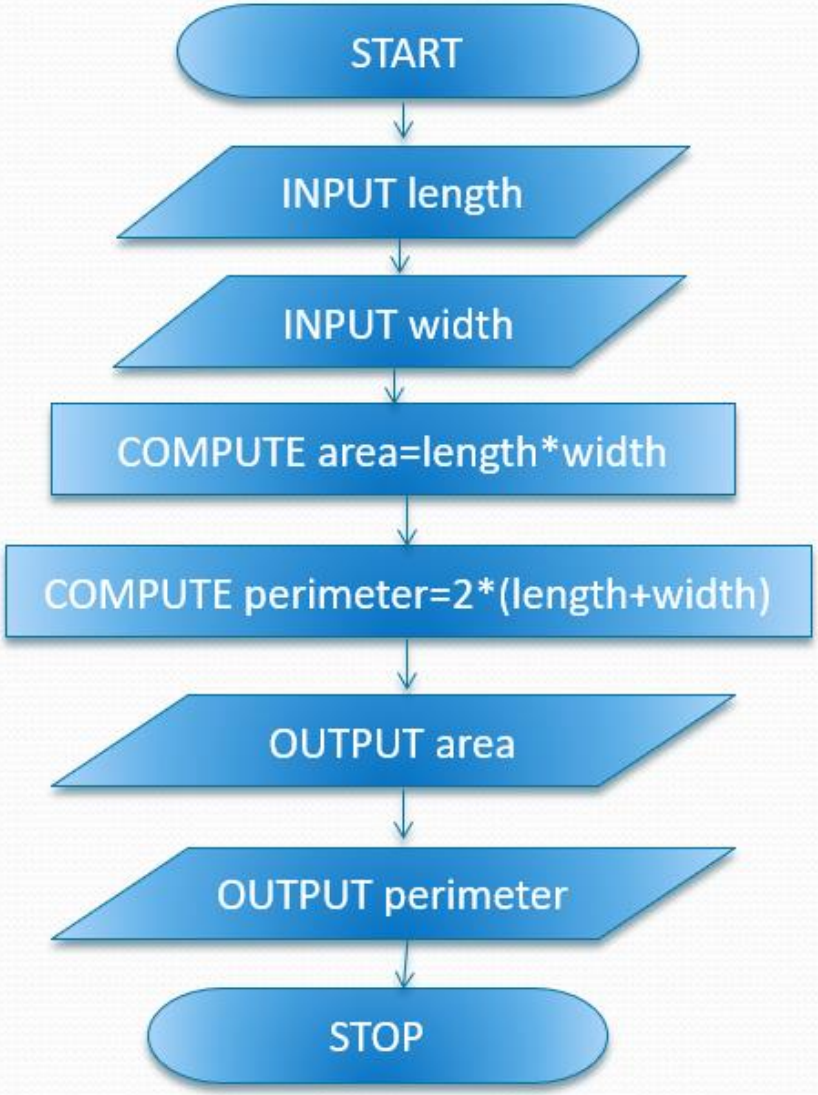
- **Problem**:
  - Design an algorithm to find the perimeter and area of a rectangle

  - The perimeter and area of the rectangle are given by the following formulas:

```
perimeter = 2 * (length + width)
area = length * width
```

Example 1

- Flowchart

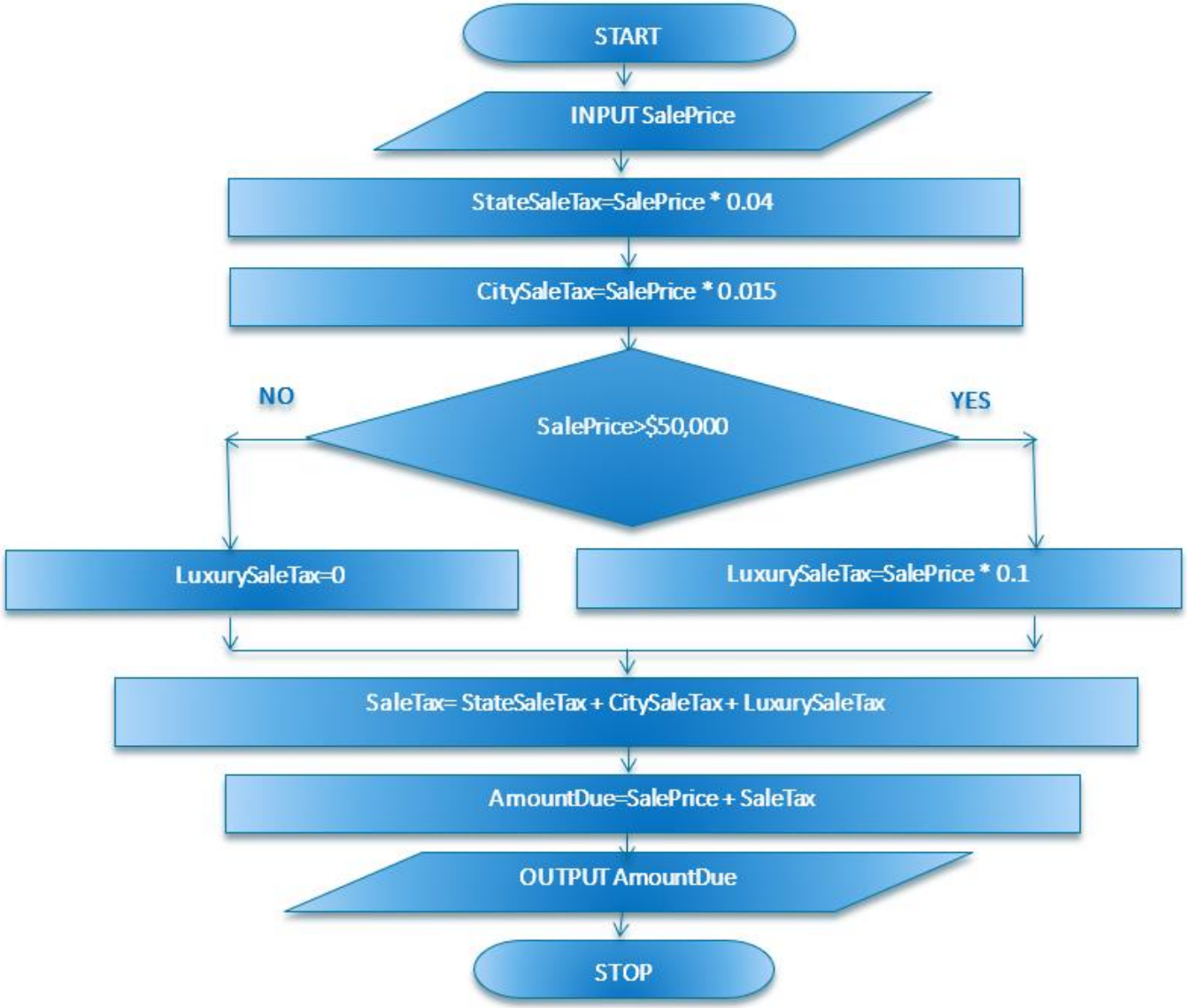


Example 2

- **Problem**
  - Write a program that calculates the sale taxes and the sale price of an item sold in a particular state.
  - The sales tax is calculated as follows:
    - the state sale tax is 4%,
    - the city tax is 1.5% and
    - the luxury tax (for luxury items ; e.g. >=$50,000) is 10%

Example 2

- Flowchart

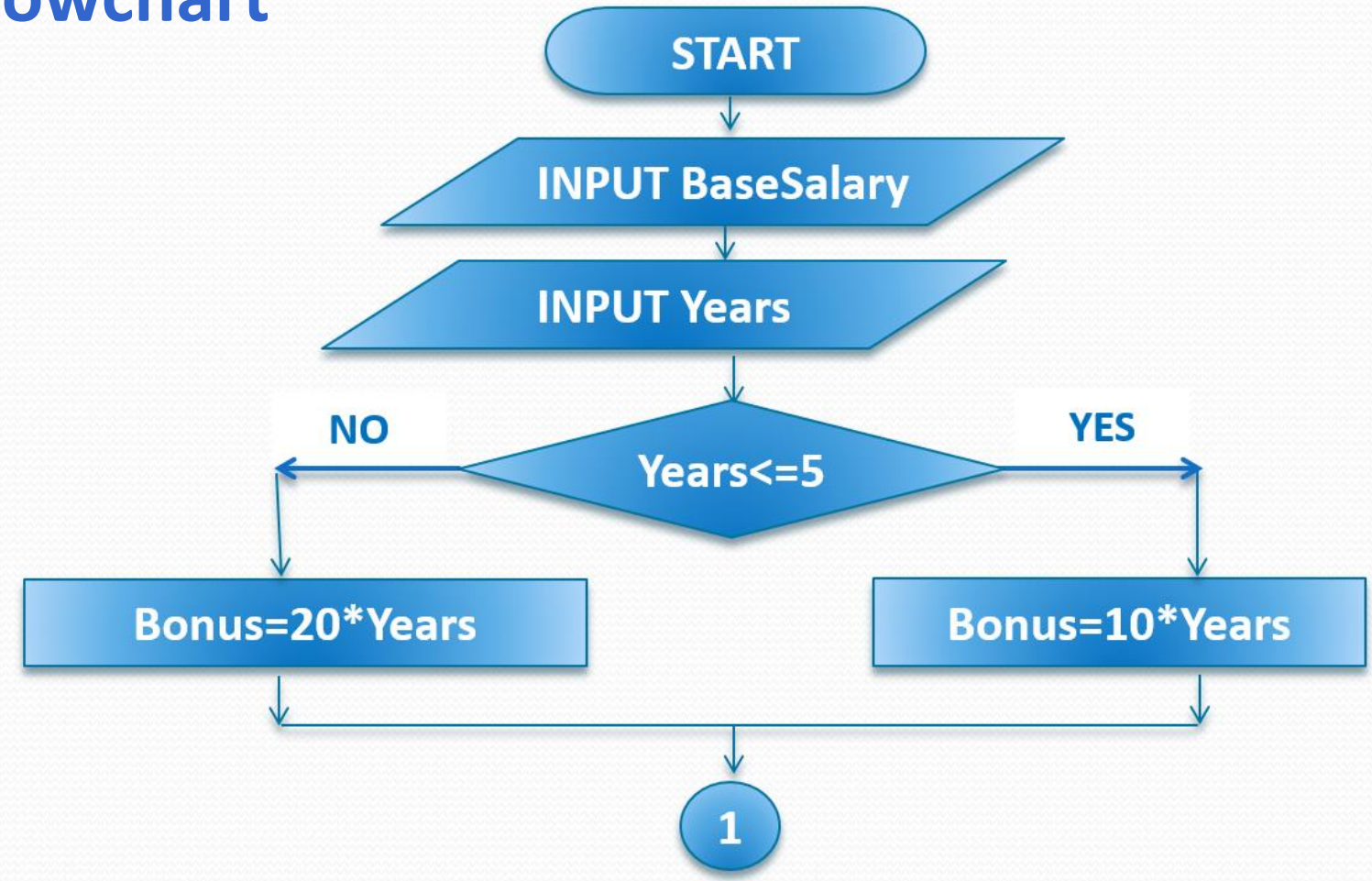


Example 3

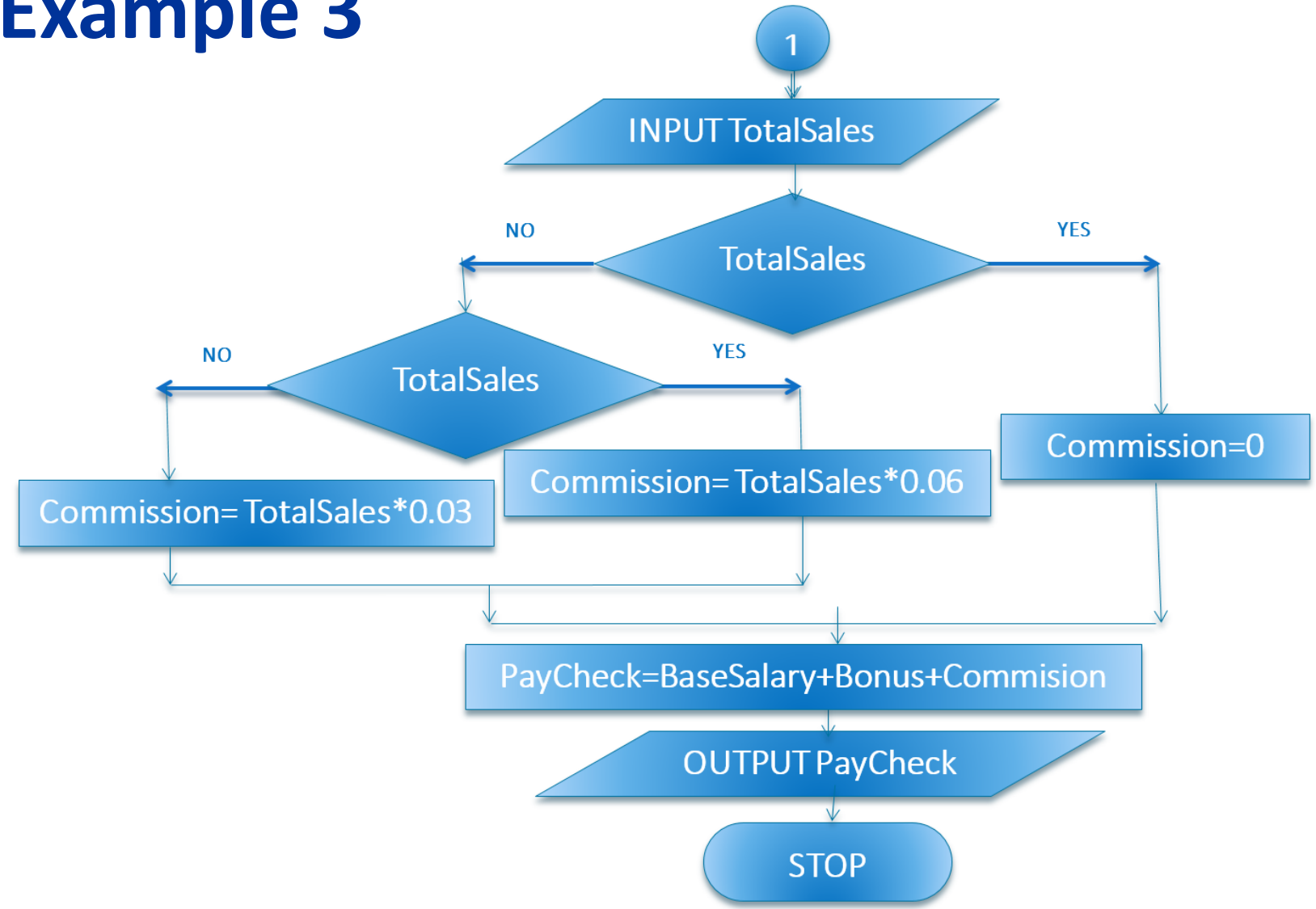
- **Problem**
  - Write a program that calculates the monthly paycheck of a salesperson at a local department store
    - Every salesperson has a base salary
    - Salesperson receives $10 bonus for each year worked if he/she has been with the store for five or less years
    - The bonus is $20 for each year that he or she has worked there if over 5 years
    - If total sales for the month are $5,000-$10,000, he or she receives a 3% commission
    - If total sales for the month are at least $10,000, he or she receives a 6% commission

Example 3

- Flowchart



Example 3

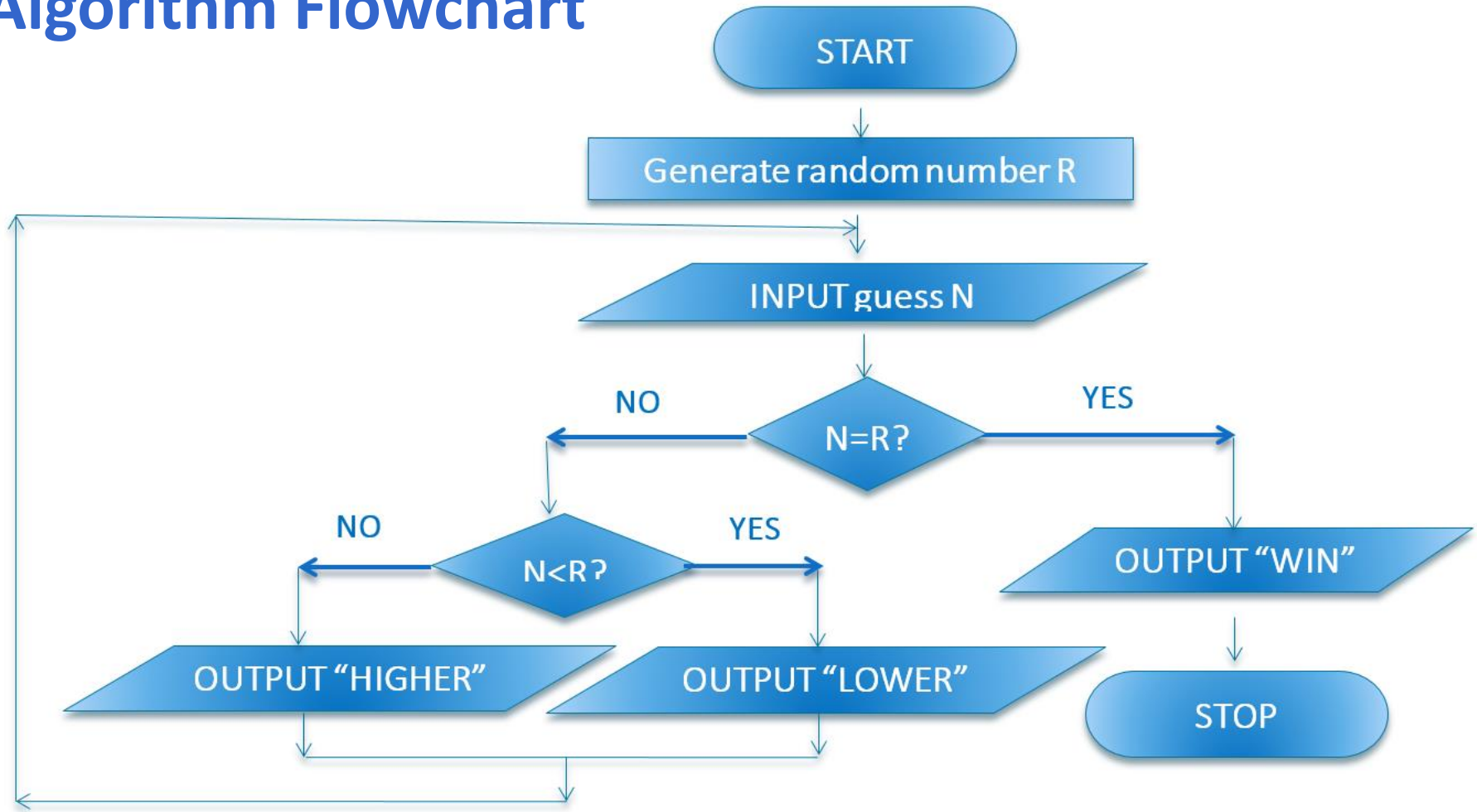

1
INPUT TotalSales
NO
TotalSales
YES
NO
TotalSales
YES
Commission=0
Commission= TotalSales*0.06
Commission= TotalSales*0.03
PayCheck=BaseSalary+Bonus+Commision
OUTPUT PayCheck
STOP

Example 4

- **Problem**
  - Design an algorithm to play a number-guessing game
  - Generate a random integer between 0 and 100. Then, prompt the player to guess the number.
  - If the guessed number is the random number, the players win.
  - If the guessed number is less that the random number output “Your guess is lower than the number”
  - If the guessed number is higher that the random number output “Your guess is higher than the number”

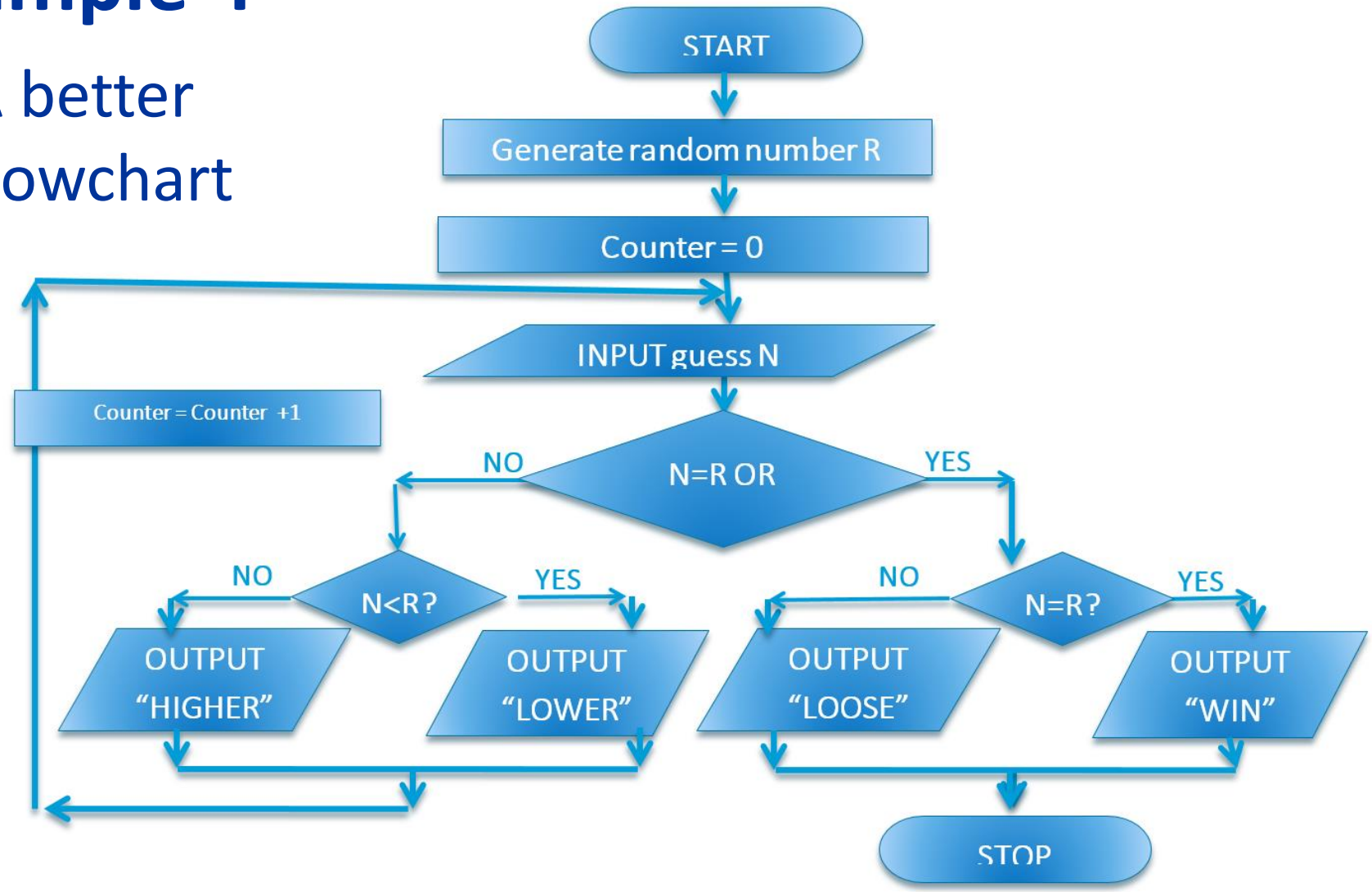
Example 4

- Algorithm Flowchart



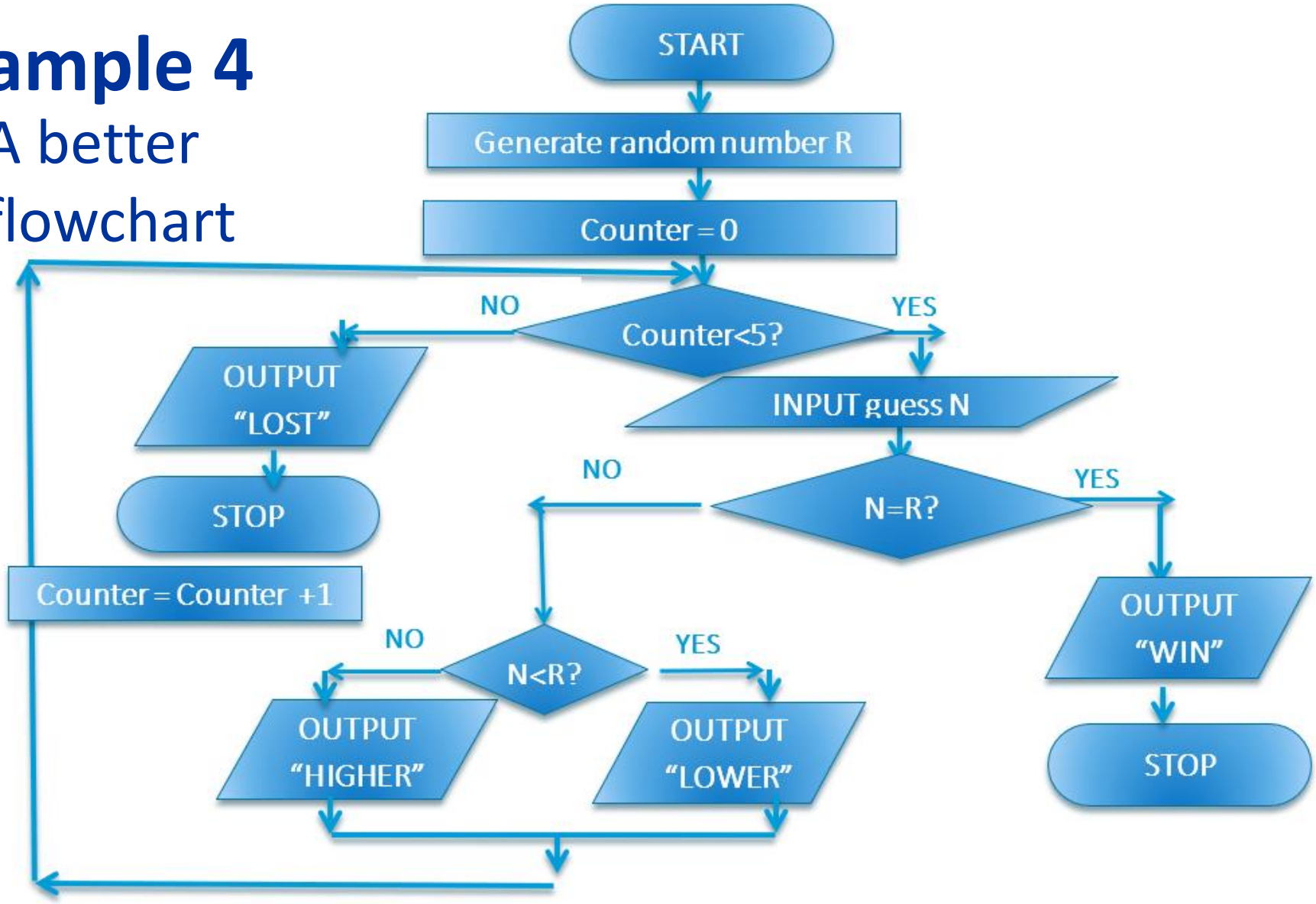
Example 4

- A better flowchart



Example 4

- A better flowchart



Example 5

- **Problem**
  - Design an algorithm to calculate the letter grade and the class average for each student from a class of 10 students that has taken five tests and each test is worth 100 points
  - Data consists of students' names and their test scores

Example 5

	$Test_1$	$Test_2$	$Test_3$	$Test_4$	$Test_5$	Total Test Points	Average Test Score	Letter Grade
$Student_1$								
$Student_2$								
$Student_3$								
$Student_4$								
$Student_5$								
$Student_6$								
$Student_7$								
$Student_8$								
$Student_9$								
$Student_{10}$								
AVERAGE								

$= Test_1 + Test_2 + Test_3 + Test_4 + Test_5 = \sum_{k=1}^{5} Test_k$

$= \frac{TotalTestPoints}{5}$

= ***A*** if 90<=*AverageTestScore*
= ***B*** if 80<=*AverageTestScore*<90
= ***C*** if 70<=*AverageTestScore*<80
= ***D*** if 60<=*AverageTestScore*<70
= ***F*** if *AverageTestScore*<60

$= \frac{AverageTestScore_1 + \cdots ++ AverageTestScore_{10}}{10}$

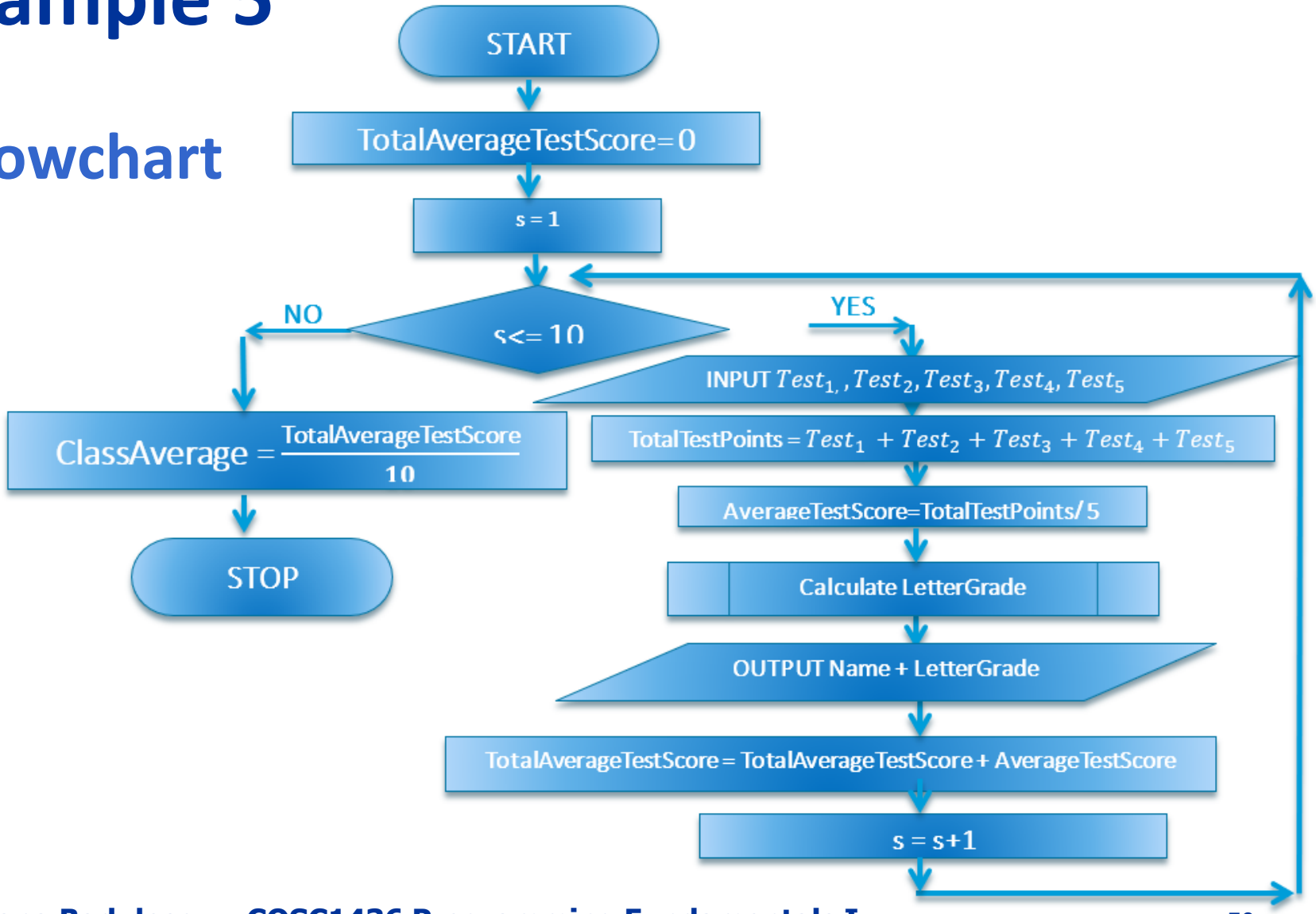
Example 5

- ***TotalTestPoints***= $Test_1 + Test_2 + Test_3 + Test_4 + Test_5$
- ***AverageTestScore*** = $\frac{TotalTestPoints}{5}$
- ***LetterGrade*** =
  - ***A*** if 90<=*AverageTestScore*
  - ***B*** if 80<=*AverageTestScore*<90
  - ***C*** if 70<=*AverageTestScore*<80
  - ***D*** if 60<=*AverageTestScore*<70
  - ***F*** if *AverageTestScore*<60
- ***ClassAverage***

= $\frac{AverageTestScore_1 + \cdots + AverageTestScore_{10}}{10}$

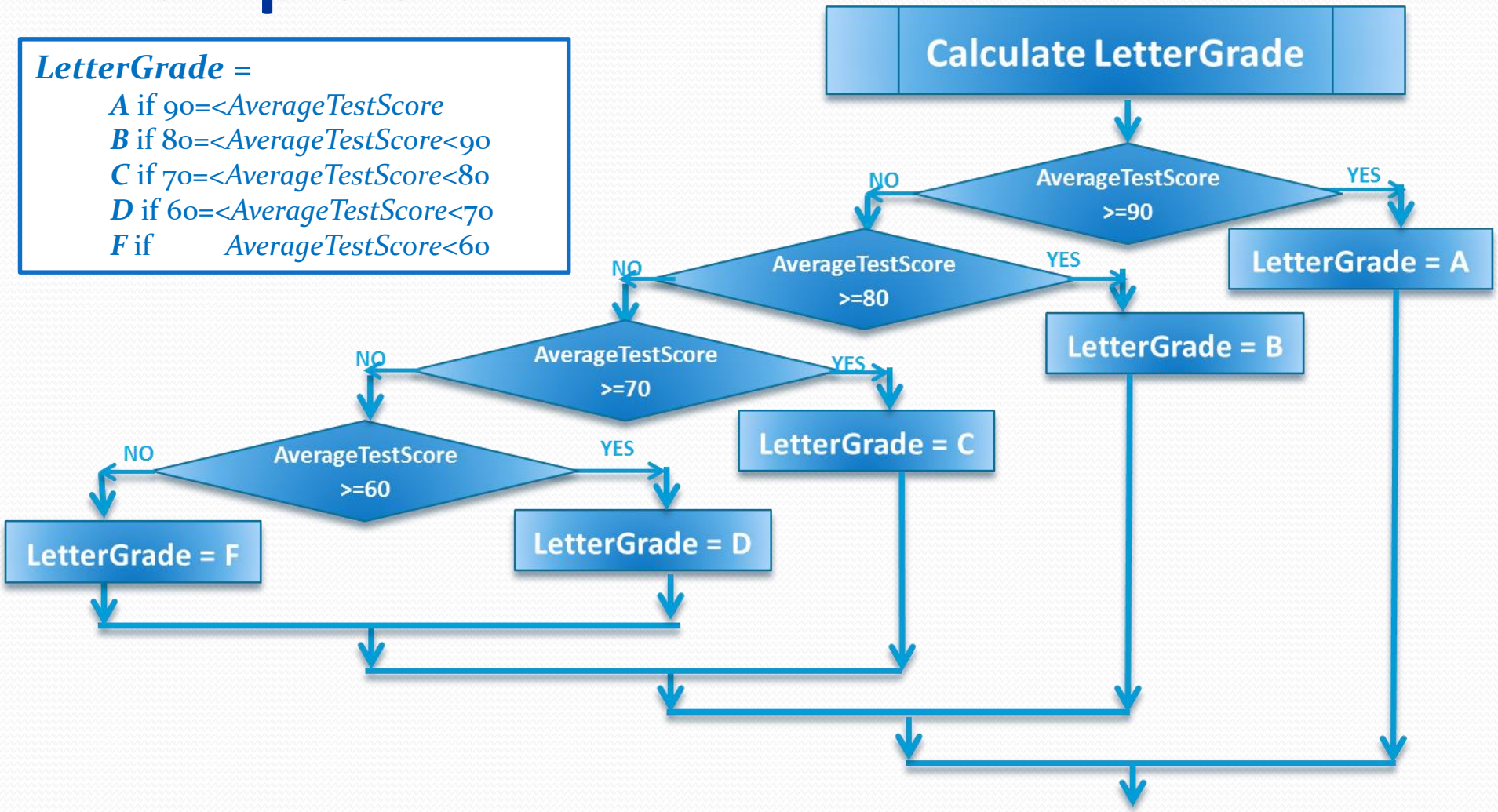
Example 5

- Flowchart



Example 5

LetterGrade =
A if 90=<*AverageTestScore*
B if 80=<*AverageTestScore*<90
C if 70=<*AverageTestScore*<80
D if 60=<*AverageTestScore*<70
F if *AverageTestScore*<60



Summary

- History of Computers and Programming
- Information Processing Cycle
- Types of Computers and Programming Language
- Computer Software
- Language of computers
- Types of Programming languages
- History of C++ Programming Language
- Programming Process and Problem-Solving Technique
- Examples and Exercises