

1 Introduction



Objectives

At the end of the lesson, the student should be able to:

- Identify the different components of a computer
- Know about programming languages and their categories
- Understand the program development life cycle and apply it in problem solving
- Learn the different number systems and their conversions



Introduction

- Computer
 - a machine that performs a variety of tasks according to specific instructions
- Two major components:
 - Hardware
 - tangible part of the computer
 - composed of electronic and mechanical parts
 - Software
 - intangible part of a computer
 - consists of data and the computer programs



Basic Components of a Computer : Hardware

- CPU
 - Central processing unit
 - The processor is the “brain” of the computer
 - It does the fundamental computing within the system
 - Examples: Pentium, Athlon and SPARC.



Basic Components of a Computer : Hardware

- Memory
 - where data and instructions needed by the CPU to do its appointed tasks can be found
 - 2 Types:
 - Main Memory
 - Secondary Memory



Basic Components of a Computer : Hardware

- Main Memory
 - used to hold programs and data, that the processor is actively working with.
 - not used for long-term storage
 - sometimes called the RAM (Random Access Memory).
 - considered as volatile storage – means once the computer is turned off, all information residing in the main memory is erased



Basic Components of a Computer : Hardware

- Secondary Memory
 - used to hold programs and data for long term use.
 - Examples of secondary memory are hard disks and cd-rom.
 - considered as non-volatile storage



Basic Components of a Computer : Hardware

- Comparison between main memory and secondary memory

<i>Main Memory</i>	<i>Secondary Memory</i>	<i>Property</i>
Fast	Slow	Speed
Expensive	Cheap	Price
Low	High	Capacity
Yes	No	Volatile

Basic Components of a Computer : Hardware

- Input and Output Devices
 - allows a computer system to interact with the outside world by moving data into and out of the system
 - Examples:
 - input devices: keyboards, mice and microphones
 - output devices: monitors, printers and speakers



Basic Components of a Computer : Software

- Software
 - a program that a computer uses in order to function
 - kept on some hardware device like a hard disk, but it itself is intangible
 - data that the computer uses can be anything that a program needs
- Programs
 - act like instructions for the processor.



Basic Components of a Computer : Software

- Some Types of Computer Programs
 - Systems Programs
 - Application Programs
 - Compilers



Basic Components of a Computer : Software

- Systems Programs
 - Programs that are needed to keep all the hardware and software systems running together smoothly
 - Examples: Operating Systems like Linux, Windows, Unix, Solaris, MacOS



Basic Components of a Computer : Software

- Application Programs
 - Programs that people use to get their work done
 - Examples: Word Processor, Game programs, Spreadsheets



Basic Components of a Computer : Software

- Compilers
 - Translates computer programs to machine language
 - Machine language
 - Language that your computer understands.



Programming Languages

- Programming Language
 - a standardized communication technique for expressing instructions to a computer
 - Like human languages, each language has its own syntax and grammar
 - There are different types of programming languages that can be used to create programs, but regardless of what language you use, these instructions are translated into machine language that can be understood by computers.



Categories of Programming Languages

- High-level Programming Languages
 - a programming language that is more user-friendly, to some extent platform-independent, and abstract from low-level computer processor operations such as memory accesses
 - A programming statement may be translated into one or several machine instructions by a **compiler**.
 - Examples: Java, C, C++, Basic, Fortran



Categories of Programming Languages

- Low-level Assembly Language
 - Assembly languages are similar to machine languages, but they are much easier to program because they allow a programmer to substitute names for numbers
 - Assembly languages are available for each CPU family, and each assembly instruction is translated into one machine instruction by an assembler program



Categories of Programming Languages

- NOTE:
 - The terms "high-level" and "low-level" are inherently relative.
 - Originally, assembly language was considered low-level and COBOL, C, etc. were considered high-level.
 - Many programmers today might refer to these latter languages as low-level



Program Development Life Cycle

- Basic steps in trying to solve a problem on the computer:
 1. Problem Definition
 2. Problem Analysis
 3. Algorithm design and representation
(Pseudocode or flowchart)
 4. Coding and debugging



1. Problem Definition

- A clearly defined problem is already half the solution.
- Computer programming requires us to define the problem first before we even try to create a solution.
- Let us now define our example problem:
 - “Create a program that will determine the number of times a name occurs in a list.”



2. Problem Analysis

- After the problem has been adequately defined, the simplest and yet the most efficient and effective approach to solve the problem must be formulated.
- Usually, this step involves breaking up the problem into smaller and simpler subproblems.
- Example Problem:
 - Determine the number of times a name occurs in a list
- Input to the program:
 - list of names (let's call this nameList)
 - name to look for (let's call this keyName)
- Output of the program:
 - the number of times the name occurs in a list



3. Algorithm Design and representation

- Algorithm
 - a clear and unambiguous specification of the steps needed to solve a problem.
 - It may be expressed in either :
 - Human language (English, Tagalog)
 - Graphical representation like a flowchart
 - Pseudocode - which is a cross between human language and a programming language



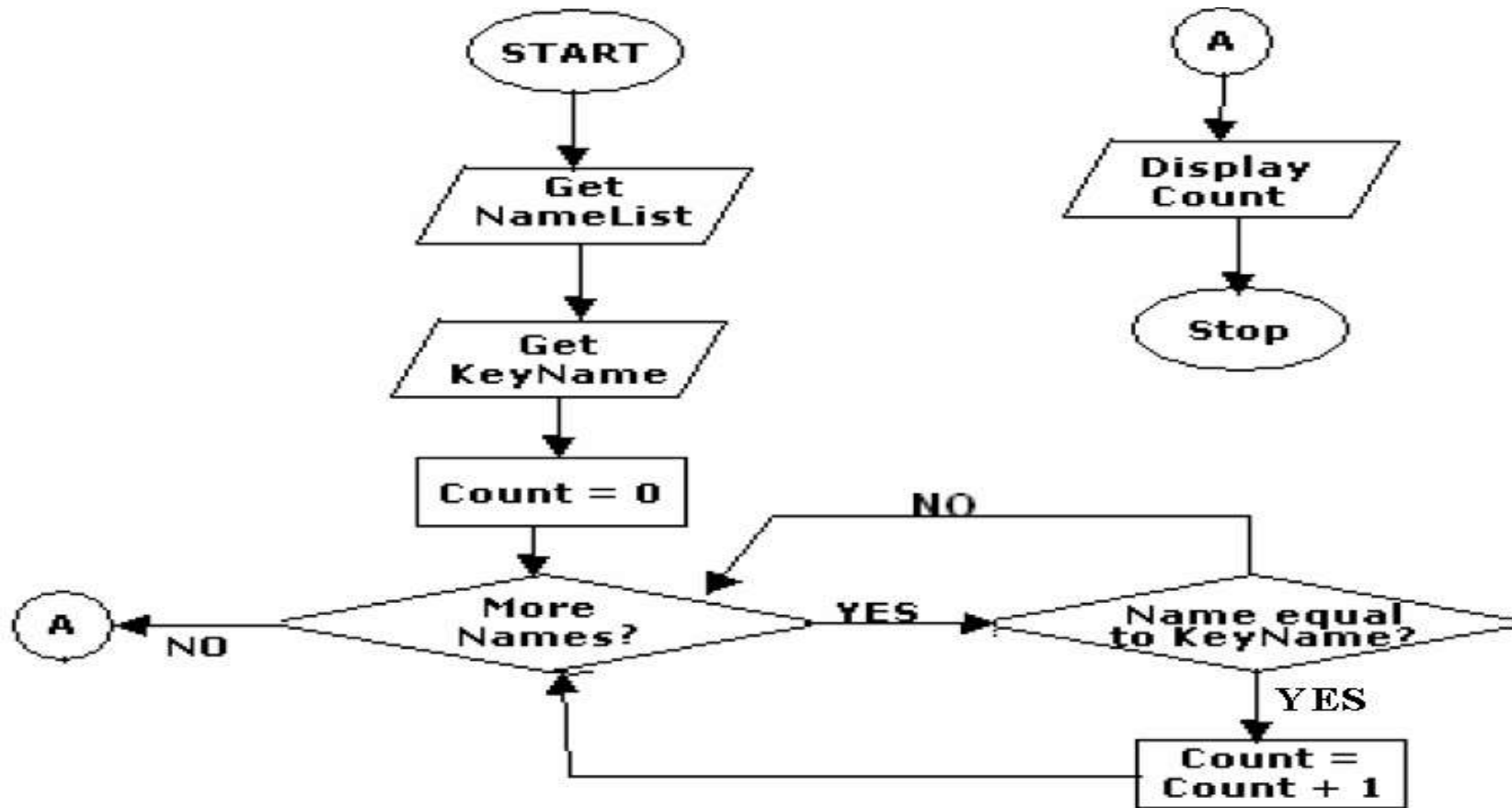
3. Algorithm Design & representation–Human Lang.

- Expressing our solution through Human language:
 1. Get the list of names, let's call this **nameList**
 2. Get the name to look for, let's call this the **keyname**
 3. Compare the **keyname** to each of the names in **nameList**
 4. If the **keyname** is the same with a name in the list, add 1 to the **count**
 5. If all the names have been compared, output the result



3. Algorithm Design and representation - Flowchart

- Expressing our solution through a flowchart:

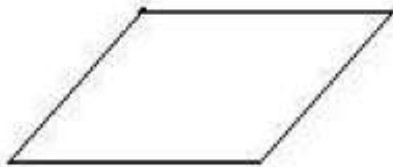


Flowchart Symbols



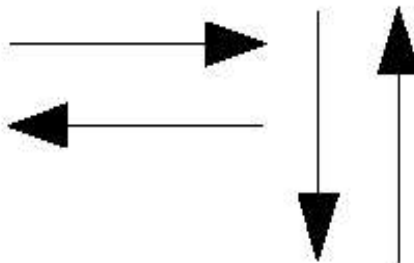
Process Symbol

Represents the process of executing a defined operation or groups of operations that results in a change in value, form, or location of information. Also functions as the default symbol when no other symbol is available.



**Input/Output
(I/O) Symbol**

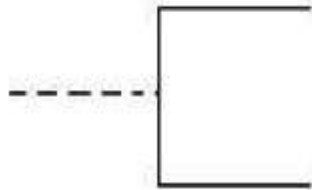
Represents an I/O function, which makes data available for processing (input) or displaying (output) of processed information.



Flowline Symbol

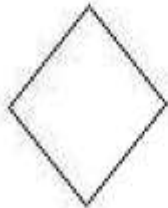
Represents the sequence of available information and executable operations. The lines connect other symbols, and the arrowheads are mandatory only for right-to-left and bottom-to-top flow.

Flowchart Symbols



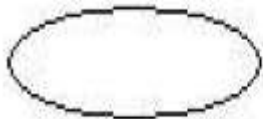
**Annotation
Symbol**

Represents the addition of descriptive information, comments, or explanatory notes as clarification. The vertical line and the broken line may be placed on the left, as shown, or on the right.



Decision Symbol

Represents a decision that determines which of a number of alternative paths is to be followed.



Terminal Symbol

Represents the beginning, the end, or a point of interruption or delay in a program.

Flowchart Symbols



**Connector
Symbol**

Represents any entry from, or exit to, another part of the flowchart. Also serves as an off-page connector.



**Predefined
Process Symbol**

Represents a named process consisting of one or more operations or program steps that are specified elsewhere.

- **NOTE:**
 - These are just guidelines for commonly used symbols in creating flowcharts. You can use any symbols in creating your flowcharts, as long as you are consistent in using them



3. Algorithm Design and representation - Pseudocode

- Expressing our solution through pseudocode:

Let nameList = List of Names

Let keyName = the name to be sought

Let Count = 0

For each name in NameList do the following

 if name == keyName

 Count = Count + 1

Display Count



4. Coding and Debugging

- After constructing the algorithm, it is now possible to create the source code. Using the algorithm as basis, the source code can now be written using the chosen programming language.
- Debugging
 - The process of fixing some errors (bugs) in your program



Types of Errors

- Compile-time error or syntax errors
 - occur if there is a syntax error in the code.
 - The compiler will detect the error and the program won't even compile. At this point, the programmer is unable to form an executable program that a user can run until the error is fixed.
- Runtime Errors
 - Compilers aren't perfect and so can't catch all errors at compile time. This is especially true for logic errors such as infinite loops. This type of error is called runtime error.



Number Systems

- Numbers can be represented in a variety of ways.
- The representation depends on what is called the BASE.
- You write these numbers as:

Number_{base}



Number Systems

- The following are the four most common representations.
 - Decimal (base 10)
 - Commonly used
 - Valid digits are from 0 to 9
 - Example: 126_{10} (normally written as just 126)
 - Binary (base 2)
 - Valid digits are 0 and 1
 - Example: 1111110_2



Number Systems

- The following are the four most common representations.
(continuation)
 - Octal (base 8)
 - Valid digits are from 0 to 7
 - Example: 176_8
 - Hexadecimal (base 16)
 - Valid digits are from 0 to 9 and A to F (or from a to f)
 - Example: $7E_{16}$



Number Systems

- Examples:

<i>Decimal</i>	<i>Binary</i>	<i>Octal</i>	<i>Hexadecimal</i>
126 ₁₀	1111110 ₂	176 ₈	7E ₁₆
11 ₁₀	1011 ₂	13 ₈	B ₁₆

Conversion: Decimal to Binary


- Method:
 - continuously divide the number by 2
 - get the remainder (which is either 0 or 1)
 - get that number as a digit of the binary form of the number
 - get the quotient and divide that number again by 2
 - repeat the whole process until the quotient reaches 0 or 1
 - we then get all the remainders starting from the last remainder, and the result is the binary form of the number
 - NOTE: For the last digit which is already less than the divisor (which is 2) just copy the value to the remainder portion.



Example: Decimal to Binary

For Example:

$$126_{10} = \underline{\quad}_2$$

	Quotient	Remainder	
126 / 2 =	63	0	 Write it this way
63 / 2 =	31	1	
31 / 2 =	15	1	
15 / 2 =	7	1	
7 / 2 =	3	1	
3 / 2 =	1	1	
1 / 2 =		1	

So, writing the remainders from the bottom up, we get the binary number 1111110_2



Conversion: Binary to Decimal

- Method:
 - we multiply the binary digit to "2 raised to the position of the binary number"
 - We then add all the products to get the resulting decimal number.



Example: Binary to Decimal

For Example:

$$1111110_2 = \underline{?}_{10}$$

Position
Binary
Digits

6	5	4	3	2	1	0	
1	1	1	1	1	1	0	
							$0 \times 2^0 = 0$
							$1 \times 2^1 = 2$
							$1 \times 2^2 = 4$
							$1 \times 2^3 = 8$
							$1 \times 2^4 = 16$
							$1 \times 2^5 = 32$
							$1 \times 2^6 = 64$
							TOTAL: 126



Conversion: Decimal to Octal/Hexadecimal

- Method:
 - Converting decimal numbers to Octal or hexadecimal is basically the same as converting decimal to binary.
 - However, instead of having 2 as the divisor, you replace it with 8(for octal) or 16 (for hexadecimal)



Example: Decimal to Octal/Hexadecimal

For Example (Octal):

$$126_{10} = \underline{\quad}_8$$

	Quotient	Remainder	
$126 / 8 =$	15	6	Write it this way ↑
$15 / 8 =$	1	7	
$1 / 8 =$		1	

So, writing the remainders from the bottom up, we get the octal number 176_8

For Example (Hexadecimal):

$$126_{10} = \underline{\quad}_{16}$$

	Quotient	Remainder	
$126 / 16 =$	7	14 (equal to hex digit E)	Write it this way ↑
$7 / 16 =$		7	

So, writing the remainders from the bottom up, we get the hexadecimal number $7E_{16}$

* * *



Conversion: Octal/Hexadecimal to Decimal

- Method:
 - Converting octal or hexadecimal numbers is also the same as converting binary numbers to decimal.
 - To do that, we will just replace the base number 2 with 8 for Octal and 16 for hexadecimal.



Example: Octal/Hexadecimal to Decimal

For Example (Octal):

$$176_8 = \underline{\quad?}_{10}$$

Position	2	1	0	
Octal Digits	1	7	6	
				$6 \times 8^0 = 6$
				$7 \times 8^1 = 56$
				$1 \times 8^2 = 64$
				<hr/>
				TOTAL: 126

For Example (Hexadecimal):

$$7E_{16} = \underline{\quad?}_{10}$$

Position	1	0	
Hex Digits	7	E	
			$14 \times 16^0 = 14$
			$7 \times 16^1 = 112$
			<hr/>
			TOTAL: 126



Conversion: Binary to Octal

- Method:
 - partition the binary number into groups of 3 digits (from right to left)
 - pad it with zeros if the number of digits is not divisible by 3
 - convert each partition into its corresponding octal digit
 - The following is a table showing the binary representation of each octal digit.

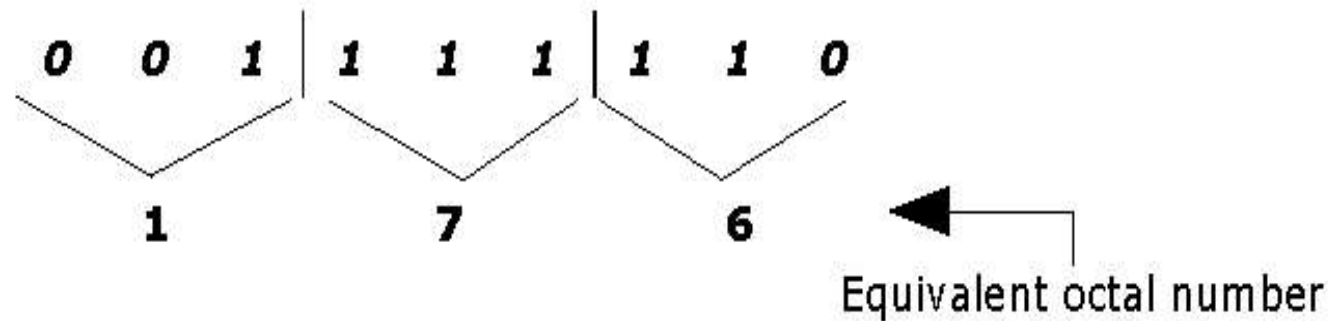
<i>Octal Digit</i>	<i>Binary Representation</i>
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111



Example: Binary to Octal

For Example:

$1111110_2 = \underline{\quad}_8$



Conversion: Octal to Binary

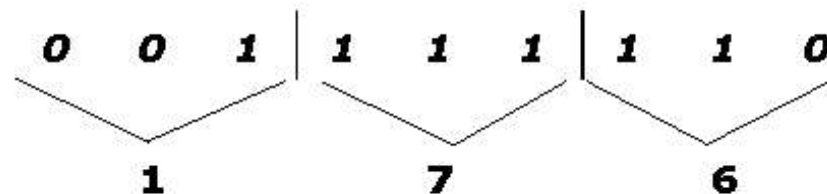
- Method:
 - Converting octal numbers to binary is just the opposite of what is given previously.
 - Simply convert each octal digit into its binary representation (given the table) and concatenate them.
 - The result is the binary representation.



Example: Octal to Binary

Octal Digit	Binary Representation
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

For Example:
 $1111110_2 = ?_8$



←
Equivalent octal number



Conversion: Binary to Hexadecimal

Hexadecimal Digit	Binary Representation
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

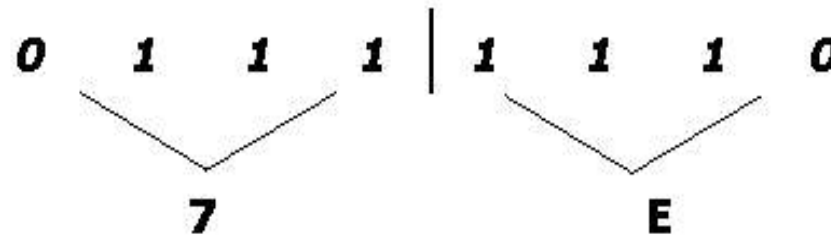
- Method:
 - partition the binary number into groups of 4 digits (from right to left)
 - pad it with zeros if the number of digits is not divisible by 4
 - convert each partition into its corresponding hexadecimal digit
 - The following is a table showing the binary representation of each hexadecimal digit



Example: Binary to Hexadecimal

For Example:

$1111110_2 = \underline{\quad?}_{16}$



←
Equivalent Hexadecimal
number



Conversion: Hexadecimal to Binary

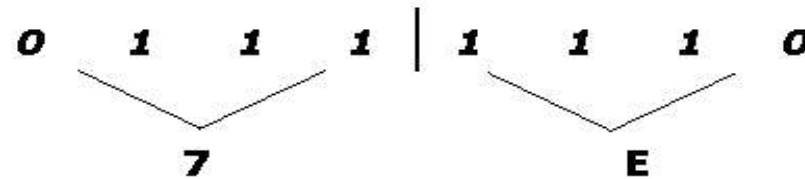
- Method:
 - Converting hexadecimal numbers to binary is just the opposite of what is given previously.
 - Simply convert each hexadecimal digit into its binary representation (given the table) and concatenate them.
 - The result is the binary representation.



Example: Hexadecimal to Binary

Hexadecimal Digit	Binary Representation
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

For Example:
 $1111110_2 = ?_{16}$



←
Equivalent Hexadecimal
number



Summary

- Basic Components of a Computer
 - Hardware
 - Software
- Overview of Computer Programming Languages
 - What is a Programming Language?
 - Categories of Programming Languages
- Program Development Life Cycle
 - 1. Problem Definition
 - 2. Problem Analysis
 - 3. Algorithm Design and representation (human Lang., Flowchart, Pseudocode)
 - 4. Coding and Debugging



Summary

- Types of Errors
 - Compile time errors/syntax errors
 - Runtime errors
- Number Systems
 - Decimal, Hexadecimal, Binary, Octal
 - Conversions

