The Software Development Life Cycle

Contents of Lecture:

- Computer Programming
- The Software Development Life Cycle (SDLC)
- Constants and variables

Computer Programming

- Programmers do not sit down and start writing code right away when trying to make a computer program.
- Instead, they follow an **organized plan** or **methodology** that breaks the process or problem into a series of tasks.

The Software Development Life Cycle (SDLC)

- SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to **develop**, **maintain**, **replace** or **enhance** specific software.

- Here are six of the most common SDLC methodologies:
  - ✓ Waterfall Model
  - ✓ V-Shaped Model
  - ✓ Iterative Model
  - ✓ Spiral Model
  - ✓ Big Bang Model
  - ✓ Agile Model

- Here are the 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

- This stage is the formal definition of the task.
- It includes:
  - The specification of inputs and outputs processing requirements
  - System constraints
  - Error handling methods.

- This step is very critical for the completion of a satisfactory program.
- It is **impossible** to solve a problem by using a computer, **without** a **clear understanding** and **identification** of the problem.
- Inadequate identification of problem leads to poor performance of the system.

- If programmer does not spend enough time at this stage, programmer may find that the program written my fails to solve the real problem.

- A programmer is usually given a task in the form of a problem. Before a program can be designed to solve a particular problem, the problem must be well and clearly defined first in terms of its input and output requirements.
  - ✓ Because a clearly defined problem is already half the solution.

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 sub-problems.

3. Algorithm design and representation (Pseudocode or flowchart)

- Once our problem is clearly defined, we can now set to finding a solution.
- In computer programming, it is normally required to express our solution in a step-by-step manner.

- **Definition:**
  - ✓ An **Algorithm** is a clear and unambiguous specification of the steps needed to solve a problem.

  - ✓ It may be expressed in either:
    - Human language (English, Arabic)
    - Through a graphical representation like a flowchart
    - Through a pseudocode, pseudocode is a cross between human language and a programming language.

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.

- Most of the time, after the programmer has written the program, the program isn't 100% working right away.
- The programmer has to add some fixes to the program in case of errors (also called bugs) that occurs in the program. This process is called debugging.

- There are two types of errors that a programmer will encounter along the way:
  - Compile-time error
  - Runtime error.

1) Compile-Time 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 that a user can run until the error is fixed.

- Forgetting a semi-colon at the end of a statement or misspelling a certain command, for example, is a compile-time error. It's something the compiler can detect as an error.

2) Runtime error:

- A runtime error is a program error that occurs while the program is running
- Compilers aren't perfect and so can't catch all errors at compile time.

- **Example**: want to add two numbers write number1 – number2 instate of writing number1 + number2.
- Also. This is especially true for logic errors such as infinite loops.

- Other types of run-time errors are when an incorrect value is computed, the wrong thing happens, etc.

Example:

- In order to understand the basic steps in solving a problem on a computer, let us define a single problem that we will solve step-by-step as we discuss the problem solving methodologies in detail.

- **Problem Definition:**
  "Create a program that will know how much older one person is than another."

- **Problem Analysis:**
  - ✓ Input to the program:

  - ✓ Output of the program:

- **Algorithm design and representation (Pseudocode or flowchart):**

- **Coding and debugging:**

Constants and variables

- Constants and variables are two entities which are used to store information (technically called data) in your program. Data has values.

- **Definitions:**
  - **Constant** is a value that never changes as the instructions in a program are followed. Constants can be any type of data.

  - **Variable** is a value that does change as the instructions in a program are followed. Variables can be any type of data.

- Both constants and variables are given names; we use the name of the constant or variable to refer to the locations rather than the memory address.

- For this unit, the main reason why we use these names is to make the program clearer.
- You should always use names that are most meaningful to the end user (the person who is going to use the system you design) or to the problem, this is very important as it helps to make the program more understandable.

- There are four rules to name constants and variables as follows:
  1. Name a constant or variable according to what it represents.
  2. Do not use spaces in a constant or variable name.
  3. Start a constant or variable name with a letter, not a number.
  4. Do not use a dash in a name (the computer will think it a subtraction sign), an underscore is fine.