# UNIT 1
# PROGRAMMING LANGUAGE

LH – 10HRS

PRESENTED BY:

**ER. SHARAT MAHARJAN**

C PROGRAMMING

# CONTENTS (LH – 10HRS)

1.1 Introduction to Programming Language
1.2 Types of Programming Language
1.3 Language processor
1.4 Program errors
1.5 Features of Good Program
1.6 Different Programming Paradigm
1.7 Software Development Model
1.8 Program Development Life Cycle
1.9 System Design Tools

# 1.1 Introduction to Programming Language

- If one wants to communicate with a person to do something, he will tell him in a language that he understands. Similarly, if one want to perform some task using a computer, one has to instruct the computer in a language that the computer understands.
- The languages which are used to instruct the computer to perform certain tasks are called computer programming languages.
- Thus, a programming language is a set of rules that provides a way of instructing the computer to perform certain operations.
- Currently there are many programming languages in world to write computer programs.
- A computer program is a set of instructions written in a specific programming language that a computer can interpret and execute.

# 1.2 Types of Programming Language

**Types of Programming Languages**

There are **two types** of programming languages, which can be categorized into the following ways:

**1. Low level language**

a) Machine language (1GL-1st Generation Language)

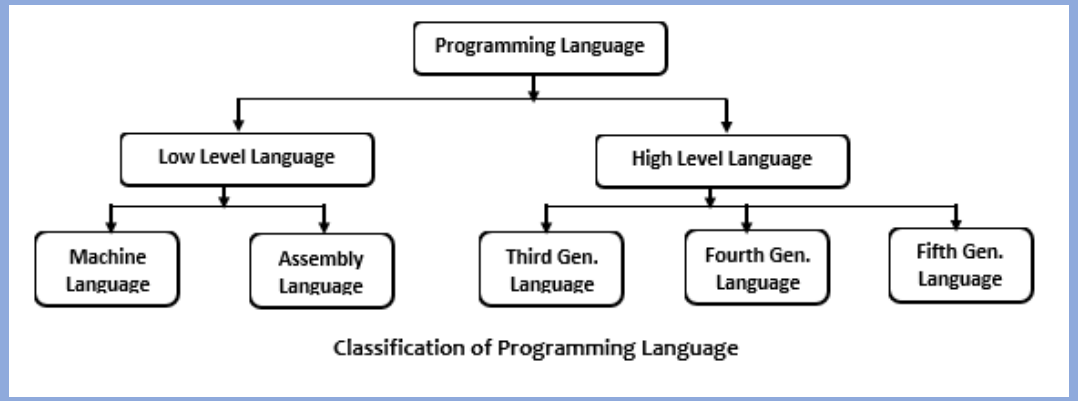
b) Assembly language (2GL)

**2. High level language**

a) Procedural-Oriented language (3GL)

b) Problem-Oriented language (4GL)

c) Natural language (5GL)



Classification of Programming Language

## 1. Low level language:

This language is the most understandable language used by computer to perform its operations. It can be further categorized into:

### a) Machine Language (1GL)

Machine language consists of strings of binary numbers (i.e. 0s and 1s) and it is the only one language, the processor directly understands. Machine language has an Merits of very fast execution speed and efficient use of primary memory.

**Advantages:**

- It is directly understood by the processor so has faster execution time since the programs written in this language need not to be tanslated.
- It doesn’t need larger memory.

**Disadvantages:**

- It is very difficult to program using 1GL since all the instructions are to be represented by 0s and 1s.
- Use of this language makes programming time consuming.
- It is difficult to find error and to debug.
- It can be used by experts only.

## b) Assembly Language (2GL):

Assembly language is also known as low-level language because to design a program programmer requires detailed knowledge of hardware specification. This language uses **mnemonics code** (symbolic operation code like '**ADD**' for addition) in place of 0s and 1s. The program is converted into machine code by assembler. The resulting program is referred to as an object code.

**Advantages:**

- It is makes programming easier than 1GL since it uses **mnemonics code** for programming. e.g.; **ADD** for addition, **SUB** for subtraction, **DIV** for division, etc.
- It makes programming process faster.
- Error can be identified much easily compared to 1GL.
- It is easier to debug than machine language.

**Disadvantages:**

- Programs written in this language is not directly understandable by computer so translators should be used.
- It is hardware dependent language so programmers are forced to think in terms of computer's architecture rather than to the problem being solved.
- Being machine dependent language, programs written in this language are very less or not portable.
- Programmers must know its mnemonics codes to perform any task.

## 2. High level language:

Instructions of this language closely resembles to human language or English like words. It uses mathematical notations to perform the task. The high level language is easier to learn. It requires less time to write and is easier to maintain the errors. The high level language is converted into machine language by one of the two different languages translator programs; **interpreter or compiler.**

High level language can be further categorized as:

### a) Procedural-Oriented language (3GL):

Procedural Programming is a methodology for modeling the problem being solved, by determining the steps and the order of those steps that must be followed in order to reach a desired outcome or specific program state. These languages are designed to express the logic and the procedure of a problem to be solved. It includes languages such as C, FORTAN, etc.

**Advantages:**

- Because of their flexibility, procedural languages are able to solve a variety of problems.
- Programmer does not need to think in term of computer architecture which makes them focused on the problem.
- Programs written in this language are portable.

**Disadvantages:**

- It is easier but needs higher processor and larger memory.
- It needs to be translated therefore its execution time is more.

## b) Problem-Oriented language (4GL):

It allows the users to specify what the output should be, without describing all the details of how the data should be manipulated to produce the result. This is one step ahead from 3GL. These are result oriented and include database query language.

E.g.: C#, PHP, etc.

The objectives of 4GL are to:

- Increase the speed of developing programs.
- Minimize user's effort to obtain information from computer.
- Reduce errors while writing programs.

**Advantages:**

- Programmer need not to think about the procedure of the program. So, programming is much easier.

**Disadvantages:**

- It is easier but needs higher processor and larger memory.
- It needs to be translated therefore its execution time is more.

## c) Natural language (5GL):

Natural language are still in developing stage where we could write statements that would look like normal sentences.

**Advantages:**

- Easy to program.
- Since, the program uses normal sentences, they are easy to understand.
- The programs designed using 5GL will have artificial intelligence (AI).
- The programs would be much more interactive and interesting.

**Disadvantages:**

- It is slower than previous generation language as it should be completely translated into binary code which is a tedious task.
- Highly advanced and expensive electronic devices are required to run programs developed in 5GL. Therefore, it is an expensive approach.

# 1.3 Language processor

- Mostly, high-level languages like C, C++, Java, and more are used to write the programs, called source code. These source codes need to translate into machine language to be executed because they cannot be executed directly by the computer. Hence, a special translator system, a language processor, is used to convert source code into machine language.
- A language processor is a special type of software program that has the potential to translate the program codes into machine codes.
- Languages such as C and Fortran have language processors, which are generally used to perform tasks like processing source code to object code or machine code.

The language processors can be any of the following three types:

## 1. Compiler:

- The language processor that reads the complete source program written in high level language as a whole in one pass and translates it into an equivalent machine code is called Compiler.
- Example: C, C++, Java etc.
- In a compiler, the source code is translated to object code successfully if it is free from errors. The compiler specifies the errors at the end of compilation with line numbers when there are any errors in the source codes. The errors must be removed before the compiler can successfully recompile the source code again.

## 2. Assembler:

- It is used to translate the program written in Assembly language into machine code.
- The source program is an input of assembler that contains assembly language instructions. The output generated by assembler is the object code or machine code understandable by the computer.


## 3. Interpreter:

- It is the language translator that translates the statement of source program written in high level language into machine code line by line and executes it immediately before moving on to the next line.
- If there is an error in the statement, interpreter displays an error message.
- Example: Python, Perl etc.

| Compiler | Interpreter |
| --- | --- |
| 1. It is a program which converts the entire source code of a program into executable machine code at once. | 1. It is a program which converts the source program into machine code line by line. |
| 2. It takes large amount of time to analyze the entire source code but the overall execution time of the program is comparatively faster. | 2. It takes less amount of time to analyze the source code but the overall execution time of the program is slower. |
| 3. It generates the error message only after scanning the whole program, so debugging is comparatively hard. | 3. Its debugging is easier as it continues translating the program until the error is met. |
| 4. It generates intermediate object code. | 4. No intermediate object code is generated. |
| 5. Examples: C, C++, Java | 5. Examples: Python, Perl |

# 1.4 Program errors

When developing **programs** there are three types of error that can occur:

1. syntax errors
2. logic errors
3. runtime errors

**1. Syntax errors**

A syntax error occurs when the code given does not follow the syntax rules of the programming language. Examples include:

- misspelling a statement, e.g. writing pint instead of print
- using a variable before it has been declared
- missing brackets, e.g. opening a bracket, but not closing it

A program cannot run if it has syntax errors. Any such errors must be fixed first. A good integrated development environment (IDE) usually points out any syntax errors to the programmer.

## 2. Logic errors:

A logic error is an error in the way a program works. The program can run but does not do what it is expected to do.

Logic errors can be caused by the programmer:

- incorrectly using logical operators, e.g. expecting a program to stop when the value of a variable reaches 5, but using <5 instead of <=5
- incorrectly using Boolean operators
- unintentionally creating a situation where an infinite loop may occur
- incorrectly using brackets in calculations
- unintentionally using the same variable name at different points in the program for different purposes

A program with a syntax error will not run. A program with a logic error will run but it will not perform as expected.

## 3. Runtime errors:

- A runtime error is an error that takes place during the running of a program.
- An example is writing a program that tries to access the sixth item in an array that only contains five items. A runtime error is likely to crash the program.

# 1.5 Features of Good Program

- A proper and correct instructions should be provided to the computer so that it can provide the desired output.
- Hence, a program should be developed in such a way that it ensures proper functionality of the computer.
- In addition, a program should be written in such a manner that it is easier to understand the underlying logic.

A good computer program should have following characteristics:

1. Portability:

It refers to the ability of an application to run on different platforms (operating systems) with or without minimal changes. portability is measured by how a software application can be transferred from one computer environment to another without failure. A program is said to be more portable if it is easily adopted in different computer systems.

2. Readability:

The program should be written in such a way that it makes other programmers or users to follow the logic of the program without much effort. If a program is written structurally, it helps the programmers to understand their own program in a better way.

3. Efficiency:

Program is said to be more efficient if it takes least amount of memory and processing time and is easily converted to machine language. The program efficiency is also high if it has high speed during runtime execution of the program.

4. Flexibility:

The program should be written in such a manner that it allows to add new features without changing the existing module. The majority of the projects are developed for a specific period and they require modifications from time to time. It should always be ready to meet new requirements.

## 5. Cost Effectiveness:

Cost Effectiveness is the key to measure the program quality. Cost must be measured over the life of the program and must include both cost and human cost of producing these programs.

## 6. Reliable:

The user's actual needs will change from time-to-time, so the program is said to be reliable if it works smoothly in every version. It is measured as reliable if it gives same performance in all simple to complex conditions.

## 7. Maintainability:

It is the process of fixing program errors and improving the program. A maintainable software allows us to quickly and easily fix a bug, increase usability and performance, add new features, make changes to support multiple platforms etc.

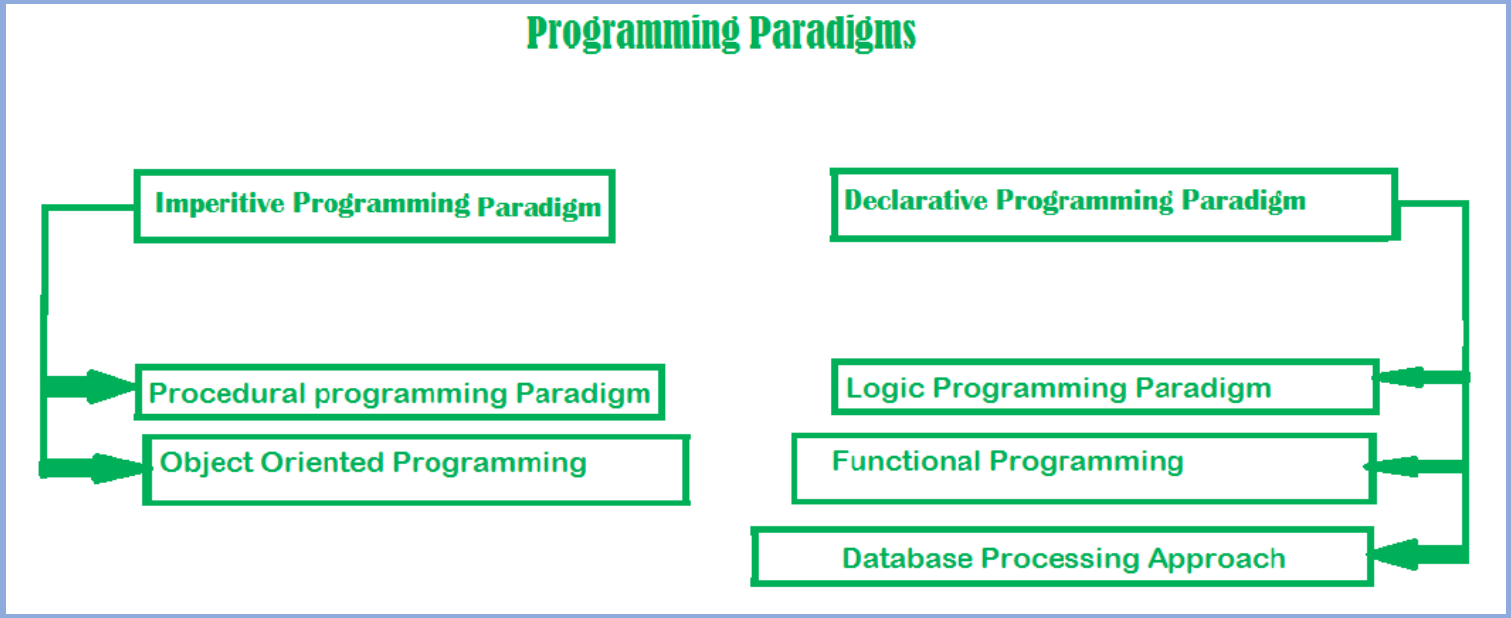
## 8. Documentation:

It is one of the most important components of an application development. A well-documented application is also useful for other programmers because even in the absence of the author, they can understand it.

# 1.6 Different Programming Paradigm

**Paradigm** can also be termed as method to solve some problem or do some task. Programming paradigm is an approach to solve problem using some programming language or also we can say it is a method to solve a problem using tools and techniques that are available to us following some approach. There are lots for programming language that are known but all of them need to follow some strategy when they are implemented and this methodology/strategy is paradigms. Apart from varieties of programming language there are lots of paradigms to fulfil each and every demand.

They are discussed below:




**1. Imperative programming paradigm:** It is one of the oldest programming paradigm. It features close relation to machine architecture. It is based on Von Neumann architecture. It works by changing the program state through assignment statements. It performs step by step task by changing state. The main focus is on how to achieve the goal. The paradigm consist of several statements and after execution of all the result is stored.

**Advantage:**

1. Very simple to implement
2. It contains loops, variables etc.

**Disadvantage:**

1. Complex problem cannot be solved
2. Less efficient and less productive
3. Parallel programming is not possible

Imperative programming is divided into two broad categories: Procedural and OOP. These paradigms are as follows:

a. **Procedural programming paradigm –** This paradigm emphasizes on procedure in terms of under lying machine model. There is no difference in between procedural and imperative approach. It has the ability to reuse the code and it was boon at that time when it was in use because of its reusability.

b. **Object oriented programming –** The program is written as a collection of classes and object which are meant for communication. The smallest and basic entity is object and all kind of computation is performed on the objects only. More emphasis is on data rather procedure. It can handle almost all kind of real life problems which are today in scenario.

**Advantages:**

- Data security
- Inheritance
- Code reusability
- Flexible and abstraction is also present

**2. Declarative programming paradigm:** It is divided as Logic, Functional, Database. In computer science the *declarative programming* is a style of building programs that expresses logic of computation without talking about its control flow. It often considers programs as theories of some logic. It may simplify writing parallel programs. The focus is on what needs to be done rather how it should be done basically emphasize on what code is actually doing. It just declares the result we want rather how it has be produced. This is the only difference between imperative (how to do) and declarative (what to do) programming paradigms. Getting into deeper we would see logic, functional and database.

a. **Logic programming paradigms –** It can be termed as abstract model of computation. It would solve logical problems like puzzles, series etc. In logic programming we have a knowledge base which we know before and along with the question and knowledge base which is given to machine, it produces result. In normal programming languages, such concept of knowledge base is not available but while using the concept of artificial intelligence, machine learning we have some models like Perception model which is using the same mechanism. In logical programming the main emphasize is on knowledge base and the problem. The execution of the program is very much like proof of mathematical statement, e.g., Prolog

b. **Functional programming paradigms –** The functional programming paradigms has its roots in mathematics and it is language independent. The key principal of this paradigms is the execution of series of mathematical functions. The central model for the abstraction is the function which are meant for some specific computation and not the data structure. Data are loosely coupled to functions. The function hide their implementation. Function can be replaced with their values without changing the meaning of the program. Some of the languages like Perl, JavaScript mostly uses this paradigm.

c. **Database/Data driven programming approach –** This programming methodology is based on data and its movement. Program statements are defined by data rather than hard-coding a series of steps. A database program is the heart of a business information system and provides file creation, data entry, update, query and reporting functions. There are several programming languages that are developed mostly for database application. For example SQL. It is applied to streams of structured data, for filtering, transforming, aggregating (such as computing statistics), or calling other programs. So it has its own wide application.

# 1.7 Software Development Model

A software process model is an abstraction of the software development process. The models specify the stages and order of a process. So, think of this as a representation of the **order of activities** of the process and the **sequence** in which they are performed.

**A model will define the following:**

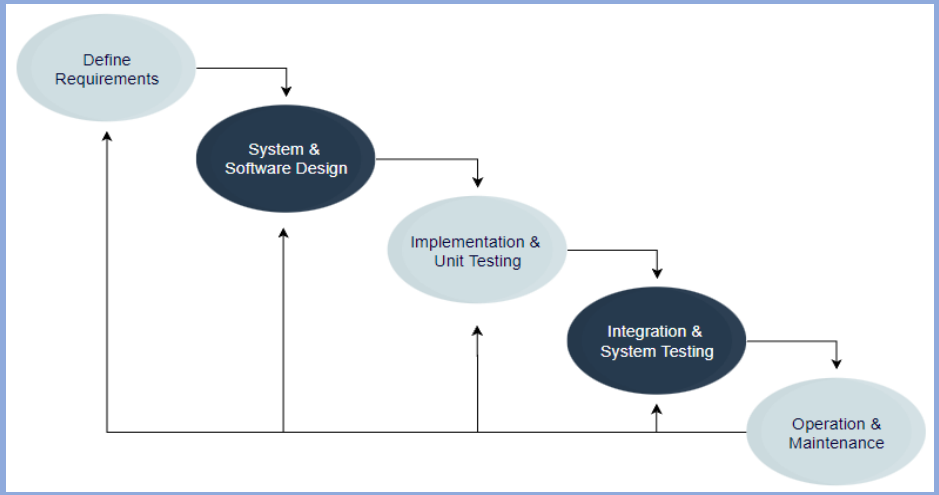
- The tasks to be performed
- The input and output of each task
- The pre and post conditions for each task
- The flow and sequence of each task

# 1. Waterfall Model:

- The waterfall model is a **sequential, plan driven-process** where you must plan and schedule all your activities before starting the project. Each activity in the waterfall model is represented as a separate phase arranged in linear order.

It has the following phases:

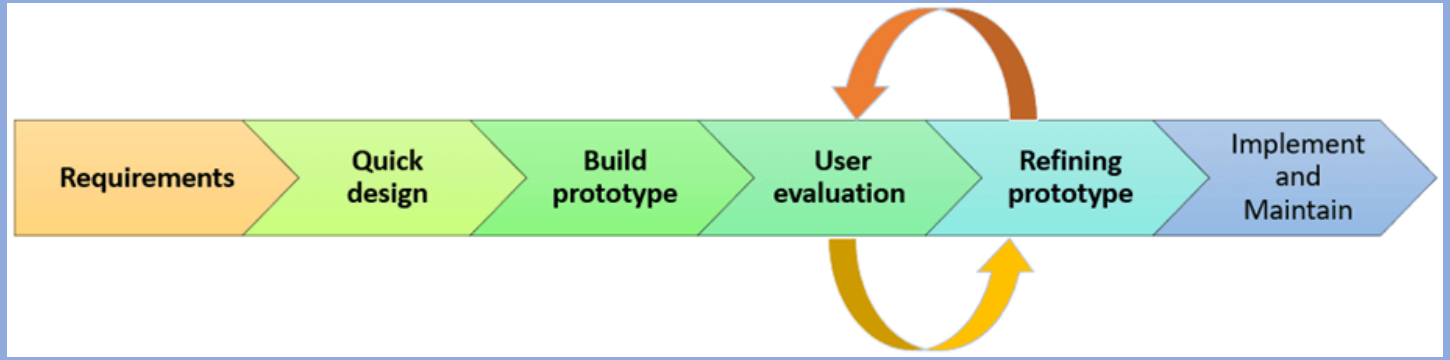
- Requirements
- Design
- Implementation
- Testing
- Deployment
- Maintenance




- Each of these phases produces one or more documents that need to be approved before the next phase begins. However, in practice, these phases are very likely to overlap and may feed information to one another.
- The waterfall model is easy to understand and follow. It doesn't require a lot of customer involvement after the specification is done. Since it's inflexible, it can't adapt to changes. There is no way to see or try the software until the last phase.
- The waterfall model has a rigid structure, so it should be used in cases where the requirements are understood completely and unlikely to radically change.

## 2. Prototyping Model:

**Prototyping Model** is a software development model in which prototype is built, tested, and reworked until an acceptable prototype is achieved. It also creates base to produce the final system or software. It works best in scenarios where the project's requirements are not known in detail. It is an iterative, trial and error method which takes place between developer and client.




Prototyping Model has following six SDLC phases as follow:

**Step 1: Requirements gathering and analysis**

- A prototyping model starts with requirement analysis. In this phase, the requirements of the system are defined in detail. During the process, the users of the system are interviewed to know what is their expectation from the system.

**Step 2: Quick design**

- The second phase is a preliminary design or a quick design. In this stage, a simple design of the system is created. However, it is not a complete design. It gives a brief idea of the system to the user. The quick design helps in developing the prototype.

**Step 3: Build a Prototype**

- In this phase, an actual prototype is designed based on the information gathered from quick design. It is a small working model of the required system.

**Step 4: Initial user evaluation**

- In this stage, the proposed system is presented to the client for an initial evaluation. It helps to find out the strength and weakness of the working model. Comment and suggestion are collected from the customer and provided to the developer.

**Step 5: Refining prototype**

- If the user is not happy with the current prototype, you need to refine the prototype according to the user's feedback and suggestions.
- This phase will not over until all the requirements specified by the user are met. Once the user is satisfied with the developed prototype, a final system is developed based on the approved final prototype.

**Step 6: Implement Product and Maintain**

- Once the final system is developed based on the final prototype, it is thoroughly tested and deployed to production. The system undergoes routine maintenance for minimizing downtime and prevent large-scale failures.

**Advantages of the Prototyping Model**

- Users are actively involved in development. Therefore, errors can be detected in the initial stage of the software development process.
- Missing functionality can be identified, which helps to reduce the risk of failure as Prototyping is also considered as a risk reduction activity.
- Helps team member to communicate effectively
- Customer satisfaction exists because the customer can feel the product at a very early stage.
- There will be hardly any chance of software rejection.

**Disadvantages of the Prototyping Model**

- Prototyping is a slow and time taking process.
- The cost of developing a prototype is a total waste as the prototype is ultimately thrown away.
- Prototyping may encourage excessive change requests.
- Some times customers may not be willing to participate in the iteration cycle for the longer time duration.
- There may be far too many variations in software requirements when each time the prototype is evaluated by the customer.

## 3. Spiral Model:

The spiral model, initially proposed by Boehm, is an evolutionary software process model that couples the iterative feature of prototyping with the controlled and systematic aspects of the linear sequential model. It implements the potential for rapid development of new versions of the software. Using the spiral model, the software is developed in a series of incremental releases. During the early iterations, the additional release may be a paper model or prototype. During later iterations, more and more complete versions of the engineered system are produced.

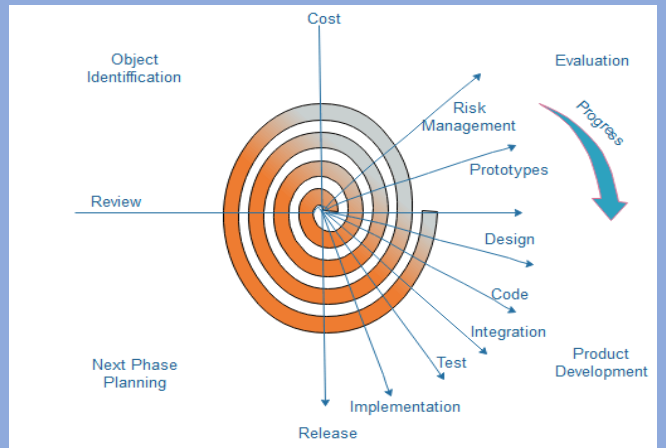



*Fig. Spiral Model*

**Each cycle in the spiral is divided into four parts:**

- **Objective setting:** Each cycle in the spiral starts with the identification of purpose for that cycle, the various alternatives that are possible for achieving the targets, and the constraints that exists.
- **Risk Assessment and reduction:** The next phase in the cycle is to calculate these various alternatives based on the goals and constraints. The focus of evaluation in this stage is located on the risk perception for the project.
- **Development and validation:** The next phase is to develop strategies that resolve uncertainties and risks. This process may include activities such as benchmarking, simulation, and prototyping.
- **Planning:** Finally, the next step is planned. The project is reviewed, and a choice made whether to continue with a further period of the spiral. If it is determined to keep, plans are drawn up for the next step of the project.

**Advantages**

- High amount of risk analysis
- Useful for large and mission-critical projects.

**Disadvantages**

- Can be a costly model to use.
- Risk analysis needed highly particular expertise
- Doesn't work well for smaller projects.

# 1.8 Program Development Life Cycle

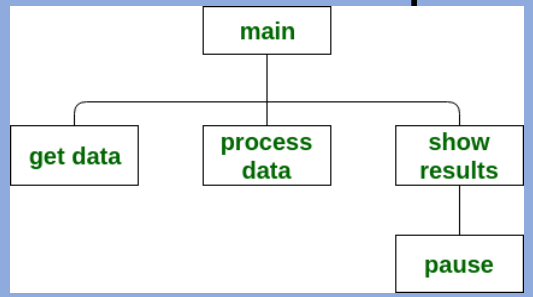
- **Program Development Life Cycle (PDLC)** is a systematic way of developing quality software. It provides an organized plan for breaking down the task of program development into manageable chunks, each of which must be successfully completed before moving on to the next phase.
- The program development process is divided into the steps discussed below:

1. **Defining the Problem –** The first step is to define the problem. In major software projects, this is a job for system analyst, who provides the results of their work to programmers in the form of a *program specification*. The program specification defines the data used in program, the processing that should take place while finding a solution, the format of the output and the user interface.

2. **Designing the Program –** Program design starts by focusing on the main goal that the program is trying to achieve and then breaking the program into manageable components, each of which contributes to this goal. This approach of program design is called *top-bottom program design* or *modular programming*. The first step involve identifying *main routine*, which is the one of program's major activity. From that point, programmers try to divide the various components of the main routine into smaller parts called *modules*. For each module, programmer draws a conceptual plan using an appropriate program design tool to visualize how the module will do its assign job.

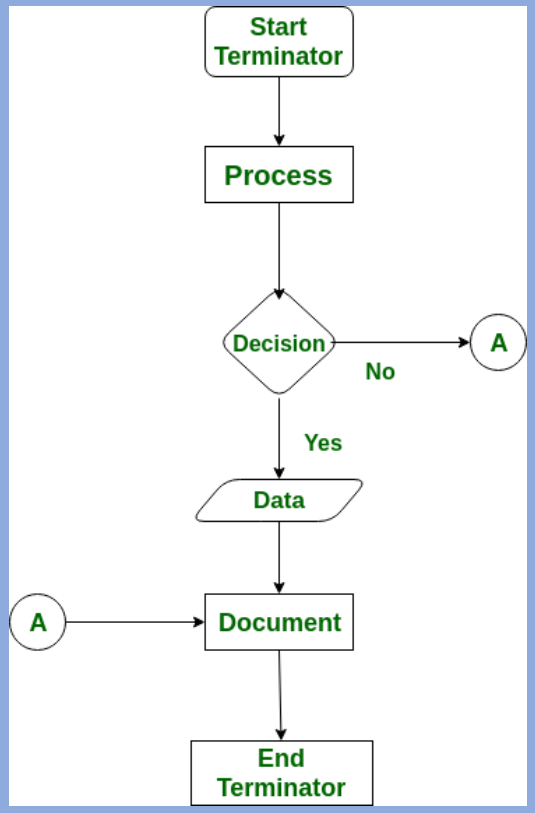
- **Program Design Tools:** The various program design tools are described below:

a. **Structure Charts** – A *structure chart*, also called *Hierarchy chart*, show top-down design of program. Each box in the structure chart indicates a task that program must accomplish. The Top module, called the *Main module* or *Control module*. For example:



b. **Algorithms –** An *algorithm* is a step-by-step description of how to arrive at a solution in the most easiest way. Algorithms are not restricted to computer world only. In fact, we use them in everyday life.

c. **Flowcharts –** A *flowchart* is a diagram that shows the logic of the program. For example:



d. **Decision tables –** A *Decision table* is a special kind of table, which is divided into four parts by a pair of horizontal and vertical lines.

e. **Pseudocode –** A *pseudocode* is another tool to describe the way to arrive at a solution. They are different from algorithm by the fact that they are expressed in program language like constructs.

3. **Coding the Program –** Coding the program means translating an algorithm into specific programming language. The technique of programming using only well defined control structures is known as *Structured programming*. Programmer must follow the language rules, violation of any rule causes *error*. These errors must be eliminated before going to the next step.

4. **Testing and Debugging the Program –** After removal of syntax errors, the program will execute. However, the output of the program may not be correct. This is because of logical error in the program. A logical error is a mistake that the programmer made while designing the solution to a problem. So the programmer must find and correct logical errors by carefully examining the program output using *Test data*. Syntax error and Logical error are collectively known as *Bugs*. The process of identifying errors and eliminating them is known as *Debugging*.

5. **Documenting the Program –** After testing, the software project is almost complete. The *structure charts, pseudocodes, flowcharts* and *decision tables* developed during the design phase become documentation for others who are associated with the software project. This phase ends by writing a manual that provides an overview of the program's functionality, tutorials for the beginner, in-depth explanations of major program features, reference documentation of all program commands and a thorough description of the error messages generated by the program.

6. **Deploying and Maintaining the Program –** In the final phase, the program is deployed (installed) at the user's site. Here also, the program is kept under watch till the user gives a green signal to it. Even after the software is completed, it needs to be maintained and evaluated regularly. In software maintenance, the programming team fixes program errors and updates the software.

# 1.9 System Design Tools

System design tools play an important role in system development. It is similar to designing the blueprint of a house before actual construction begins.

1. **Data Flow Diagram (DFD)**

- A data flow diagram is a tool that describes the flow of data through a system and works or processing performed by that system. It shows the flow of data from external entities into the system of how the data moves from one process to enter and its logical storage.

2. **Context Diagram**

- Context Diagram is a diagram that represents the actors outside a system that interact with the system. It is useful in system design to represent external factors interacting with the system itself so that system requirement and constraints can be studied very easily. It is used to document the scope for a system.

## 3. Decision Table

- A decision table allows us to identify the exact course of actions for given conditions. A decision table provides unambiguous decisions. Leading to good program design. It is a precise way to model complicated logic. It consist of three parts Conditions, Actions, and Rules.

## 4. Decision Tree

- Decision tree also does more or less the same job as decision table, except that it follows the tree structure and each of the node denote conditions. Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

# THANK YOU FOR YOUR ATTENTION