Lecture 1,2

Introduction to subject

Programming Life cycle, Programming methodology

Programming paradigms, Types of programming languages , C++ as a first programming language.

Program Development Life Cycle:

When we want to develop a program using any programming language, we follow a sequence of steps. These steps are called phases in program development. The program development life cycle is a set of steps or phases that are used to develop a program in any programming language. Generally, program development life cycle contains 6 phases, they are as follows….

Problem Definition

Problem Analysis

Algorithm Development

Coding & Documentation

Testing & Debugging

Maintenance

1. Problem Definition

In this phase, we define the problem statement and we decide the boundaries of the problem. In this phase we need to understand the problem statement, what is our requirement, what should be the output of the problem solution. These are defined in this first phase of the program development life cycle.

2. Problem Analysis

In phase 2, we determine the requirements like variables, functions, etc. to solve the problem. That means we gather the required resources to solve the problem defined in the problem definition phase. We also determine the bounds of the solution.

3. Algorithm Development

During this phase, we develop a step by step procedure to solve the problem using the specification given in the previous phase. This phase is very important for program development. That means we write the solution in step by step statements.

4. Coding & Documentation

This phase uses a programming language to write or implement actual programming instructions for the steps defined in the previous phase. In this phase, we construct actual program. That means we write the program to solve the given problem using programming languages like C, C++, Java etc.,

5. Testing & Debugging

During this phase, we check whether the code written in previous step is solving the specified problem or not. That means we test the program whether it is solving the problem for various input data values or not. We also test that whether it is providing the desired output or not.

6. Maintenance

During this phase, the program is actively used by the users. If any enhancements found in this phase, all the phases are to be repeated again to make the enhancements. That means in this phase, the solution (program) is used by the end user. If the user encounters any problem or wants any enhancement, then we need to repeat all the phases from the starting, so that the encountered problem is solved or enhancement is added.

Programming Methodology:



**Bottom-Up and Top-Down Approaches:**

Any problem can be dealt with no ways viz top-down or bottom-up. A simple example is given here to illustrate the concept.

Sorting an array of numbers involves the following :

(a) Comparison

(b) Exchange.

At the top level, an algorithm has to be formulated to carry out sorting using the above operations. Once the algorithm is confirmed, then the algorithms for comparison and exchange

are formulated, before implementation of the entire algorithm. Therefore in this approach one

begins from the top level without bothering about the minute details for implementation to start with.

The bottom-up approach is just the reverse. The lower level tasks are first carried out and are then integrated to provide the solution. In this method lower level structures are carried out. Here the algorithms for exchange and comparison will be formulated before formulating the algorithms for the whole problem.

In any case, dividing the problem into small tasks and then solving each task provides the solution. Therefore, either the top-down or bottom-up methodology has to be adopted for dividing the problem into smaller modules and then solving it. **In the top-down methodology, the overall structure is defined before getting into details, but in the bottom-up approach, the details are worked out first before defining the overall structure.**

**Points about Bottom-Up Approach**

1. In bottom-up design **individual parts of the system are specified in detail**. The parts are then linked together to form larger components, which are in turn linked until a complete system is formed.

2. Bottom-up design **yield programs which are smaller and more agile**. A shorter program doesn’t have to be divided into many components, and fewer components means program which are easier to read or modify.

3. Bottom-up design **promotes code reusability**. When you write two or more programs, many of the utilities you wrote for the first program will also be useful in the succeeding ones. That’s why reusability of code is one of the main benefits of the bottom-up approach.

4. Bottom-up design **makes programs easier to read**.

5. Working bottom-up **helps to clarify your ideas about the design of your program**.

6. Bottom-up programming may allow you for unit testing, but until most of the system comes together none of the system can be tested as a whole, often causing compilations near the end of the project.

7. An example of programming which uses this approaches is C++ and java.

Points about Top-Down Approach

1. In the top-down model **an overview of the system is formulated**, without going into detail for any part of it. **Each part of the system is then refined by designing it more detail**.

2. Each new part **may then be refined again**, defining it in yet more detail until the entire specification is detailed enough to validate the model.

3. Top-down approaches **emphasize planning and a complete understanding of the system**. It is inherent that no coding can begin until a sufficient level of detail has been reached in the design of at least some part of the system.

4. Top-down programming **is a programming style, the mainstay of traditional procedural languages**, in which design begins by specification complex pieces and then dividing them into successively smaller pieces.

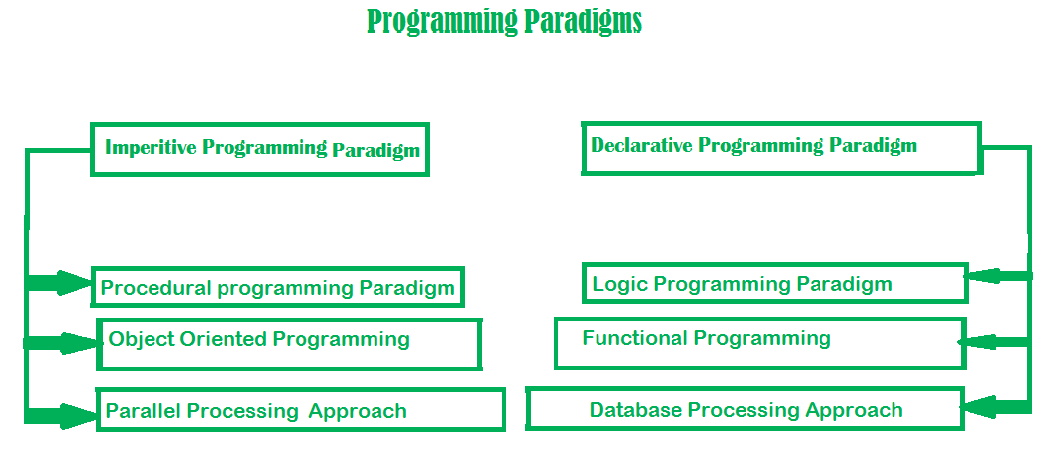
5. The technique for writing a program using top-down methods is **to write a main procedure** that have been coded the program is done.

6. Top-down programming **may complicate testing**, since nothing executable will even exit until near the end of the project.

7. An example of programming which uses this approach is C and Pascal.

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 consists 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 three broad categories: Procedural, OOP and parallel processing. These paradigms are as follows:

1. **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.

1. **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:**

1. Data security
2. Inheritance
3. Code reusability
4. Flexible and abstraction is also present
5. **Parallel processing approach –**

Parallel processing is the processing of program instructions by dividing them among multiple processors. A parallel processing system posses many numbers of processor with the objective of running a program in less time by dividing them. This approach seems to be like divide and conquer. Examples are NESL (one of the oldest one) and C/C++ also supports because of some library function.

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 code is actually doing. It just declare 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.**

Types of Programming Languages

Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate translation steps.

Machine Languages

Any computer can directly understand only its own machine language (also called machine code), defined by its hardware architecture. Machine languages generally consist of numbers

(ultimately reduced to 1s and 0s). Such languages are cumbersome for humans.

Assembly Languages

Programming in machine language was simply too slow and tedious for most programmers. Instead, they began using English-like abbreviations to represent elementary operations. These

abbreviations formed the basis of assembly languages. Translator programs called assemblers were developed to convert assembly language programs to machine language. Although assembly-language code is clearer to humans, it’s incomprehensible to computers until translated to machine language.

High-Level Languages

To speed up the programming process further, high-level languages were developed in which single statements could be written to accomplish substantial tasks. High-level languages, such as C, C++, Java, C#, Swift and Visual Basic, allow you to write instructions that look more like everyday English and contain commonly used mathematical notations. Translator programs called compilers convert high-level language programs into machine language. The process of compiling a large high-level language program into machine language can take a considerable amount of computer time. Interpreter programs were developed to execute high-level language programs directly (without the need for compilation), although more slowly than compiled programs. Scripting languages such as the popular web languages JavaScript and PHP are processed by interpreters.

C++ as a first programming language.

C++ evolved from C, which is available for most computers and is hardware independent. With careful design, it’s possible to write C programs that are portable to most computers. C++, an extension of C, was developed by Bjarne Stroustrup in 1979 at Bell Laboratories. Originally called “C with Classes,” it was renamed to C++ in the early 1980s. C++ provides a number of features that “spruce up” the C language, but more importantly, it provides capabilities for object-oriented programming that were inspired by the Simula simulation programming language.

C++ Standard Library

C++ programs consist of pieces called classes and functions. You can program each piece yourself, but most C++ programmers take advantage of the rich collections of classes and functions in the C++ Standard Library. Thus, there are really two parts to learning the C++ “world.” The first is learning the C++ language itself (often referred to as the “core language”); the second is learning how to use the classes and functions in the C++ Standard Library