SYED M MADANI

[PROGRAMMING PARADIGMS]

AN OVERVIEW OF PRINCIPAL PROGRAMMING PARADIGMS

Table of Contents

Introduction

Before we look at "**Programming paradigms**", let us go through the basics of "programming languages" and the cause that led to the evolvement of different programming paradigms.

Categories of Programming Languages

There are 3 categories of programming language. They are

- Machine Languages
- Assembly languages
- High-level languages

The first two categories are considered as "low-level languages". Now let us just list down the briefings of each category as to what each means.

Machine Languages

- A machine language program consists of a string of 0s and 1s and hence it's extremely difficult to understand both syntax and semantics of the machine language code (program).

- Any CPU architecture has its own machine language. And hence different "processors" have to be programmed with different machine language programs for the same task which implies that the code written in one machine language is extremely non-portable.

Advantages

- It is extremely efficient.
- Translation of the program code is NOT required.

Disadvantages

- The program (code) is not portable to different machines.
- The machine language is not at all programmer-friendly to program with.

Assembly Languages

- An assembly language program use mnemonics to represent machine instructions.
- Each statement in assembly language corresponds to one statement in machine language.
- There are more than one assembly statements for a statement of a given high-level language.

Examples

- **mov eax, var** will be equivalent to "**1010 0001 0000 0000 0000 0000**" on 8086 machine
- **add eax, var** will be equivalent to "**0000 0011 0000 0110 0000 0000 0000 0010**" on 8086
- High-level statement "**a += b;**" is equivalent to three assembly statements.
  They are

```
mov eax, a
add eax, b
mov a, eax
```

High-level Programming Languages

- A high-level programming language is a programming language with “strong abstractions” from the details of the computer.
- It has language primitives and grammatical rules.
- It has compiler/translator to convert the code into the “Target code”.
- There are three models of execution of modern high-level programming languages
    1) **Interpreted** – This model is where the code is read and executed directly by the Interpreter or Virtual Machine.

    2) **Compiled** – In this model, the high-level code is converted to either machine code/instructions or to an Intermediate language which can be used by an Interpreter or an assembler (if the intermediate language is assembly), and can be compiled with respect to different machine (processor) architectures.

    3) **Translated** – The program code is converted to the program of other programming language whose compilers are then used to execute the translated code.

Advantages

- o High-level language programs are portable as it can be compiled with different machine-specific compilers.
- o They are programmer-friendly.

Disadvantages

- o Most languages are inefficient.
- o Compilation/Translation of program code is required (which is very heavy processing).

Programming Paradigm

Now since the high-level languages have their semantics, the role of "programming paradigm" comes here. Some of these high-level programming languages are specifically designed for use in certain applications. Let us see what "programming paradigm" means.

- A ***programming paradigm*** is an approach of solving problems.
- It is a pattern that serves as a "**school of thoughts**" for programming computers.
- It is a model that affects the way programmers design, organize and write programs.
- Programming paradigms are the results of people's ideas about how programs should be constructed.
- Paradigms differ in ***concepts*** and ***abstractions*** used to represent the elements of a program (objects, functions, variables, constants, etc.) and steps that compose a computation (assigning, evaluation, dataflow, continuation, recursion, etc.).
- Different programming languages follow different approaches of solving a problem (i.e. support a paradigm). Therefore a programming paradigm may consist of many programming languages.

A **programming paradigm** is different from

- **"Programming Technique"** which involves algorithmic idea of solving a problem.
  - Example: **Divide and Conquer**

- **"Programming Style"** which is an elegant way of expressing the solution in a program with a particular programming language.

- **"Programming Culture"** which is a superset which involves paradigms, techniques and styles.

Principal Programming Paradigms

There are many programming paradigms which have been developed but the following are considered to be principal programming paradigms

- **Imperative / Procedural**
- **Functional / Applicative**
- **Object-Oriented**
- **Generic**
- **Logic**
- **Scripting**
- **Concurrent**

Imperative Paradigm

- Imperative or procedural programs consist of **actions to affect the state change** through either assignment operator or side effects. Example: **c = a + b;** OR **add(a, b, &c);**

- A series of statements and variables implies that there is a stage change from one statement the following one which leads to change in program state. This concludes that that "**do this then do that**" is different from "**do that then do this**".

- Program execution involves changing of memory contents of computer continuously.

Some Imperative Programming Languages

- C,
- C++, Java, C#, D, Scala also supports Imperative
- PHP
- Objective-C
- Fortran
- Pascal
- Algol
- Ada
- COBOL, etc.

Advantages

- Low memory utilization by the programs (but changes often).
- Programs written in Imperative programming languages are relatively efficient than programs written using other paradigms.
- Most common form of programming in use today.

Disadvantages

- Difficulty of reasoning about programs i.e. mathematical proofs.
- Difficulty of parallelization.
- Tend to be relatively low-level.

Functional Paradigm

- A program in this paradigm consists of "*functions*" as basic method for abstractions and uses these functions in a similar way as used in Mathematics. Functional programming has its roots in "**Lambda Calculus**"

- There are NO variables in pure functional languages.

- Program execution involves functions calling each other and returning results (functions evaluation is either nested or recursive).

- Functional paradigm avoids
  1) State change, Updates (Mutable objects, variables, functions)
  2) Assignments (Exception: Non-pure functions for Input-Output)
  3) Side effects (Functions don't have side effects to outer environment)

- Evaluation of a function in Functional programming languages doesn't have side effects whereas functions in imperative programming languages do have side effects and state changes.

- For any given "x", f(x) will always be the same value **without any dependency** and **without changing any state** i.e. having NO side effect.

Some Functional Programming Languages

- Haskell, Clojure, F#
- Erlang, OCaml, ML
- Scheme , Common Lisp
- JavaScript & a few other scripting languages also supports it

Advantages

- The syntax is small and clean.
- Best for handling concurrency / parallelism.
- Better support for reasoning programs by mathematical proofs as it is related to Lambda Calculus.
- "Functions" are treated as any other data values and can be used in any data structure, algorithms and can be passed to any other functions.
- Programming in functional programming languages is relatively higher level than imperative programming languages.

Disadvantages

- Difficulty of Input-Output as IO requires variables, assignments, updates, etc.
- Functional programming languages use more memory space than their imperative cousins.

Object-Oriented Paradigm

- A program in this paradigm consists of "***objects***" as abstractions which communicate each other by **sending messages** which is another way of solving a problem with a **higher level of abstraction of real world**.

- ***Example in C++***: Suppose in a snake game, the "snake" is an object and the message to be sent is to "move" in the specified direction according to the key pressed or button clicked.
  Snake my_snake; { Object "my_snake" }
  my_snake.move(DIR_RIGHT); { Message on that object which in turn can call messages to other objects within the "move" }

- Most Object-Oriented languages are "Imperative" but not all.
  - [**Exception:** CLOS (Common Lisp Object System)]

Some Object-Oriented Programming Languages

- Simula67
- C#, C++, Scala, D (supports object-oriented paradigm)
- Ruby
- Java
- Smalltalk
- CLOS (Common Lisp Object System)

Advantages

- The paradigm is conceptually simple.
- Existing library code can be read and understood.
- Increased Productivity.
- Unit testing is relatively easy.
- Better computations. [Example: **Matrix m1, m2, m3;**
  m3 = m1 + 2 * m2;]

Disadvantages

- Can have steep learning curve initially.
- Doing Input-Output can be cumbersome as the "streams" and their "exception objects" have to be handled.
- Sometimes the modules (classes) are too coupled and may not work for different project.
- Learning Design Patterns is a must to be better at Object-Oriented Paradigm.

Generic Paradigm

- Generic programming is a style of computer programming in which **algorithms are written in terms of to-be-specified-later types** that are then instantiated when needed for specific types provided as parameters.

- This approach permits writing ***common functions*** or ***Types*** that differ only in the set of types on which they operate when used, thus **reducing duplication**.

- ***Example in C++:*** Using the "templates" in C++ to create a max() function which can be passed as parameters any type which implements (overloads) operator ">".

```
template <typename T>
T const& max(const T& a, const T& b, const T& c)
{
        return (a > b && a > c ? a : (b > c ? b : c) );
}
```

- Generic programming is commonly used to implement **containers** such as **lists** and **hash tables** and **functions** such as a particular **sorting algorithm** for objects specified in terms of **more general** than a concrete type.

Some Generic Paradigm supporting Programming Languages

- Generics in
  - Ada
  - C#
  - Java
  - Eiffel
  - Haskell
- Templates in C++, D

Advantages

- Libraries can be compatible with each other.
- Duplication of code is reduced.
- Same algorithms implementation code could be used for future without any modification.

Disadvantages

- The paradigm is conceptually complex.
- Code Bloating if care is not taken for proper "type specializations".
- Slow compilation and long compiler error messages.
- Type-safety is lost sometimes.
- A bit inefficient execution of the program written using generic paradigm.

Logic Paradigm

- Logic programming is based on ***predicate logic***. A program in logic paradigm consists of a set of "***predicates***" and "***rules of inferences***".

- Predicates are statements of fact. Example: **Water is wet**.

- Rules of inferences are statements like: **If "x" is human then "x" is mortal** – Implication in this example.

- Predicates and rules of inferences are used to prove the statements that a programmer supplies.

- Logic paradigm is targeted at
  - Theorem-proving
  - Automated reasoning
  - Database applications

Some Logic Programming Languages

- Prolog

Advantages

- Good Mathematical support for reasoning about programs.
- Can lead to concise solutions to problems.

Disadvantages

- Slow execution of programs.
- Difficulty in understanding and debugging large programs (source code).
- System doesn't know the facts that are not its predicates or rule of inference i.e. Limited View.

Scripting Paradigm

- It is **very-high level** both syntactically and conceptually and especially good for non-geeks.
- It is "***Dynamically Typed***".
- It is "***Weakly Typed***". Example: "x" can be assigned value of any type and anytime during execution.
- Scripting paradigm is popular in ***Web development***.

Some Scripting Languages

- JavaScript, TypeScript,
- Python
- Lua
- VBScript
- ActionScript (for Flash programming)
- MaxScript [3dStudioMax]
- MEL [Maya Embedded Language]
- PERL
- UnrealScript [Unreal Game Engine]

Concurrent/Parallel Paradigm

- It deals with ***parallel computing*** with simultaneous execution **with the necessary communication** and **synchronization** between processes/threads.
- Concurrent programming cuts across imperative, object-oriented, and functional paradigms.

Some Concurrent Programming Languages

- Cilk [MIT Research by Dr. Charles Leiserson based on C]
- Erlang
- Elixir

Other Paradigms

Dataflow

- Forced recalculation of formulas is performed when data values change. Example: Spreadsheets.
- It is mostly related to the **dependency of the data values** along involving the formulas.

Visual Programming

- Manipulation of program elements is done graphically rather than by specifying them textually.
- It involves boxes, shapes, connections, directional arrows for program flow, etc.

Some Visual Programming Languages

  - Simulink

Pipeline Programming

- There are no nested function calls.
- There is a simple flow structure with easy to visualize/understand data flow through the program.

Declarative Programming

- Describes actions to be performed. Example: **HTML, CSS, Jade** describes a page but not how to actually display it.

Which Paradigm is "The best" ?

- There is **NO** "**best programming paradigm**".
- Different problems have different solutions and a paradigm can be selected accordingly. There is **NOT** always **one way** to solve (approach) the problem.
- Depending upon the approach of the solution to the problem, that paradigm-supported language shall be selected.
- Most programming languages and their frameworks these days support multiple-paradigms [Imperative, OOP, Functional, Generic, Concurrent, Web Scripting MVC, etc.]. So, any one or more technology i.e. programming language(s) can be combined to achieve a solution.
- We (Programmers) use combination of these paradigms to solve problems (write programs). To master thinking in one paradigm, it requires lots of practice.
- Languages with features from different paradigms are often too complex. Example: C++
- There is **NO** "**Ultimate Programming Language**" and so different programming languages shall be learnt to solve various different sets of problems.

References

[Hpr05]	R. Harper. *Programming Languages: Theory and Practice;* draft. Carnegie Mellon University, October 2005.
[Mit03]	J. C. Mitchell. *Concepts in Programming Languages*. Cambridge University Press, 2003. ISBN 0-521-78098-5
[Bru02]	K. B. Bruce. *Foundations of Object-Oriented Languages – Types and Semantics*, MIT Press, 2002. ISBN 0-265-02523-X
[Pie02]	B. C. Pierce. *Types and Programming Languages,* MIT Press, 2002. ISBN 0-265-16209-1.
[Seb01]	R. W. Sebasta. *Concepts of Programming Languages;* 5th Edition, Addision-Wesley, 2001. ISBN 0-321-33025-0
[Sct00]	M. L. Scott. *Programming Language Pragmatics*, Morgan Kaufmann, 2000. ISBN 1-558-60442-1
[Set96]	Ravi Sethi *Programming Languages: Concepts and Constructs,* 2nd Edition, Addison-Wesley 1996. ISBN-10: 0-201-59065-4