Amrita Vishwa Vidyapeetham

TIFAC-CORE in Cyber Security

20CYS312 - Principles of Programming Languages
Assignment-01: Exploring Programming Paradigms

Nithin S

21st January, 2024

Introduction

What is Programming Paradigm?

Programming paradigms are different ways or styles in which a given program or programming language can be organized. Every paradigm has certain features, structures, and beliefs about how typical programming issues need to be solved. The difficulty of using a paradigm differs depending on the language.

Why Paradigm is needed and How it is Useful?

Programming paradigms dictate how programmers should think about and structure their code.

They influence the way programs are written, the techniques used to solve problems, and the overall design philosophy. Every paradigm has advantages and disadvantages, and a program's effectiveness, maintainability, and scalability can all be significantly impacted by selecting the best paradigm.

Paradigms are used by a number of programming languages, but in order to do so, they must follow a method or approach. Paradigms are not meant to be mutually exclusive; a single program can feature multiple paradigms.An overview of programming languages and their paradigmatic approach is given in Figure 1.

Why Many Paradigms?:

- Similar to the subject of why there are numerous programming languages, one may wonder why there are numerous programming paradigms.
- It makes sense to employ different paradigms for different kinds of real-world problems since different paradigms are better suited for different kinds of challenges.
- Additionally, new paradigms may emerge over time as the field of programming evolves and new techniques are developed.
- Thus, before selecting a programming language, we shall select the programming paradigm that best fits each challenge.

History of Programming Paradigms:

1. "The evolution of modern programming languages is closely coupled with the development (and formalization) of the concept of data type".

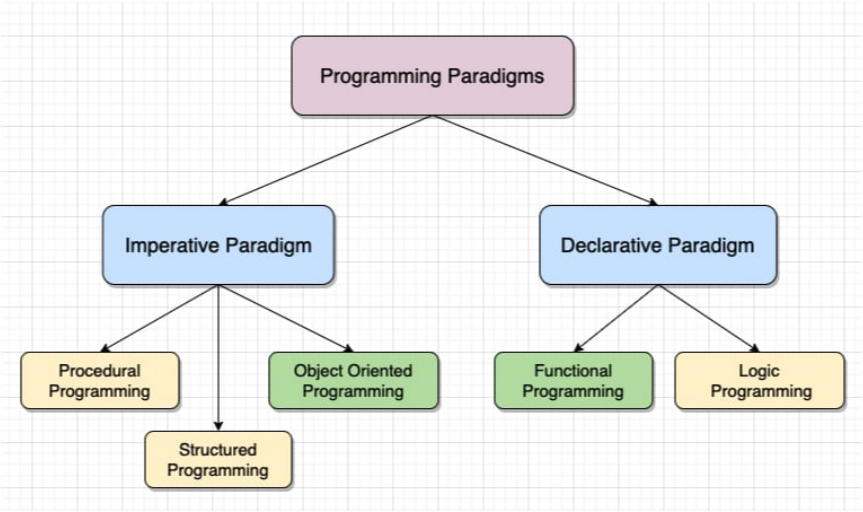

Figure 1: Types of Programming Paradigms

2. ”This evolution of languages also has been heavily influenced by progress in the theory of computing, which has led to a formal understanding of the semantics of statements, modules, abstract data types, and processes”.

3. “Object-oriented programming developed as the dominant programming methodology during the mid-1980s, isn’t a revolution in informatics but rather a long evolutionary process aiming to improve the SP [structured programming] efficiency for complex system development”.

But there is another idea. The purpose of writing a program is an algorithmic solution of a problem. However, there may be several solutions. The best solution is chosen in accordance with some criterion related to the time complexity of the problem solution and the complexity of the program structure.

A natural criterion for choosing the structure of a program is the length of its text (number of lines): the shorter the program, the more readable and easier to debug. One of the ways to shorten the text of a program is to use the go to statement. However, small but unstructured programs required a lot of time for maintenance. Therefore, at present, the following is used as a criterion: "program structure should be such as to anticipate its adaptation and modification"

Constraints imposed by the programming paradigms:

Each programming paradigm introduces constraints in programming. The constraints put the programmer in some framework that does not allow him to use the capabilities of programming languages, which contradict the paradigm, but reduce the complexity of program development.

1 Paradigm 1: Object Oriented Programming Paradigm

What is Object Oriented Programming?

OOP is defined as a programming paradigm built on the concept of objects, i.e., a set of data contained in fields, and code, indicating procedures – instead of the usual logic-based system.

OOP approach identifies classes of objects that share a close relationship with each other related to methods with which they are associated.

It arranges a computer program into fundamental, reusable code structures called "classes." Then, new and distinct objects with related functions are created by reusing these classes.

In OOP, objects are the fundamental building blocks. They encapsulate data (attributes or properties) and behavior (methods or functions) into a single unit. Objects can communicate with each other by invoking methods or accessing properties, enabling interaction and collaboration.

When to use Object Oriented Programming?

We should go for Object Oriented Programming (OOP) in the following scenarios:

When we are going to be performing few operations on lots of different variants which have common behavior. In other words, when we have more things with few operations.

Examples of OOP:

- **Pure OOP languages:**
  - Ruby
  - Scala
  - JADE
  - Emerald
- **Programming languages designed primarily for OOP:**
  - Java
  - Python
  - C++
- **Other programming languages that pair with OOP:**
  - Visual Basic .NET
  - PHP
  - JavaScript

The key concepts in object-oriented programming are:

1. **Class:** It is a blueprint or template for creating objects. It defines the properties and methods that objects of that class will have.For example, if we have a class called "Car," objects of this class will represent individual cars, and the class will define the common characteristics and behaviors that all cars share.
2. **Objects:**Objects are instances of classes. They are created from a class blueprint and have their own state (values of attributes) and behavior (methods to perform actions).Each object can have different values for its attributes while still following the structure defined by the class.
3. **Inheritance:** It allows the creation of new classes based on existing classes. It enables the reuse of code and the creation of class hierarchies. A subclass (derived class) inherits the attributes and methods of its superclass (base class or parent class) and can extend or modify their behavior. Inheritance promotes code reusability and supports the "is-a" relationship.Here are some common types of inheritance with corresponding C++ examples:

(a) **Single Inheritance:** Single inheritance involves a derived class inheriting properties and behaviors from a single base class. It forms a direct hierarchy where a derived class extends the base class.

(b) **Multiple Inheritance:** Multiple inheritance allows a derived class to inherit properties and behaviors from multiple base classes. This means that a derived class can have multiple direct base classes, combining their features into a single derived class.

(c) **Multilevel Inheritance:** Multilevel inheritance occurs when a derived class inherits from another derived class. It creates a chain or hierarchy of classes, where each derived class further extends the features of its parent class.

(d) **Hierarchical Inheritance:** Hierarchical inheritance involves multiple derived classes inheriting from a single base class. It creates a hierarchy of classes, where each derived class has its specific features while sharing the common features of the base class.

(e) **Hybrid (Virtual) Inheritance:** Hybrid inheritance combines multiple inheritance with either single or multilevel inheritance. It allows a class to inherit from multiple base classes while avoiding the issues of ambiguity that can arise due to multiple inheritance.

4. **Polymorphism:**It allows objects of different classes to be treated as objects of a common superclass and they can take on more than one form. It is achieved through method overriding and method overloading. Two Types: Compile time and run time. The program will determine which meaning or usage is necessary for each execution of that object from a parent class, reducing the need to duplicate code. A child class is then created, which extends the functionality of the parent class. Polymorphism allows different types of objects to pass through the same interface.

   There are two main types of polymorphism:

   (a) **Compile-time Polymorphism (Static Polymorphism):** Compile-time polymorphism refers to the ability of a programming language to select different functions or methods at compile-time based on the arguments provided or the types of the objects involved. This is achieved through function overloading and operator overloading.

   - Function overloading allows multiple functions with the same name but different parameter lists to coexist in the same scope. The appropriate function is selected based on the arguments passed during the function call.
   - Operator overloading allows operators such as +, -, *, /, etc., to be overloaded for different classes. This enables customized behavior for operators depending on the operands' types.

   (b) **Runtime Polymorphism (Dynamic Polymorphism):** Runtime polymorphism allows objects of different classes to be treated as objects of a common superclass. This is achieved through inheritance and virtual functions.

   - Inheritance establishes an "is-a" relationship between classes, where a derived class inherits properties and behaviors from a base class. The derived class can be used wherever the base class is expected.
   - Virtual functions are functions defined in the base class and overridden in the derived class. They allow dynamic dispatch, which means that the appropriate function to call is determined at runtime based on the actual type of the object.

5. **Abstraction:** It focuses on defining the essential characteristics and behavior of an object, while hiding the implementation details. The derived class can have its functionality extended. This concept can help developers more easily make additional changes or additions over time.

6. **Encapsulation:**It is the practice of hiding the internal details of an object and exposing only the necessary information and functionality. Access to object attributes and methods can be controlled through access modifiers. It helps maintain data integrity and provides a clean interface for interacting with objects.

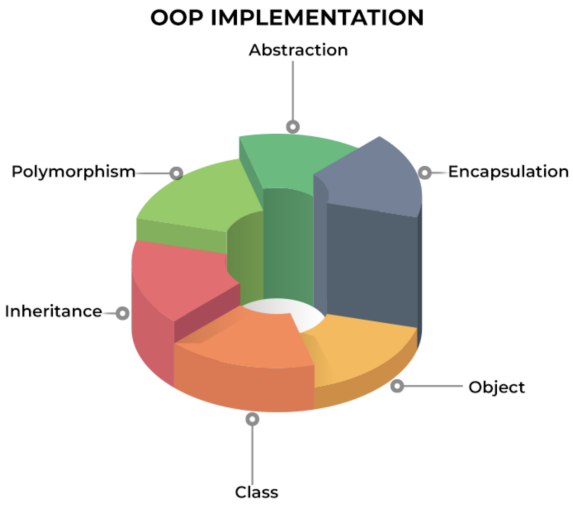

Figure 2: Implementation of OOP

The key Principles of OOP

The principles of Object-Oriented Programming (OOP) guide the design and implementation of software using the object-oriented paradigm. These principles help developers create modular, maintainable, and flexible code. Here are the key principles of OOP other than encapsulation, Inheritance, Polymorphism and abstraction:

1. **Modularity:** Modularity is the principle of breaking down complex systems into smaller, self-contained modules or objects. Each module focuses on a specific functionality or aspect of the system. Modularity promotes code organization, ease of maintenance, and reusability by allowing independent development, testing, and integration of modules.

2. **Composition:** Composition is the principle of creating complex objects by combining simpler objects or components. It enables objects to be composed of other objects, creating a "has-a" relationship. Composition promotes code reuse, flexibility, and modularity by allowing objects to collaborate and delegate tasks to other objects.

3. **Single Responsibility Principle (SRP):** SRP states that a class or module should have only one reason to change. It emphasizes separating concerns and ensuring that each class or module has a single responsibility. This principle enhances code maintainability, readability, and reduces the impact of changes.

4. **Open/Closed Principle (OCP):** The OCP states that software entities (classes, modules, functions) should be open for extension but closed for modification. It encourages designing code that can be easily extended with new functionality without modifying existing code. This principle promotes code stability, modularity, and supports the concept of interfaces and abstract classes.

A simple program that demonstrates all the concepts of OOP:

OOP program that demonstrates concepts, including class, inheritance, polymorphism, abstraction, and encapsulation, through a program that calculates and prints the area of geometric shapes (circle and square).

```
from abc import ABC, abstractmethod

# Abstraction (using abstract base class)
class Shape(ABC):
```

```
    @abstractmethod
    def area(self):
        pass

# Encapsulation
class Circle(Shape):
    def __init__(self, radius):
        self.__radius = radius

    def area(self):
        return 3.14 * self.__radius * self.__radius

# Inheritance
class Square(Shape):
    def __init__(self, side_length):
        self.__side_length = side_length

    def area(self):
        return self.__side_length * self.__side_length

# Polymorphism
def print_area(shape):
    print("Area:", shape.area())

# Creating objects and demonstrating OOP concepts
circle = Circle(5)
square = Square(4)

print("Circle:")
print_area(circle)

print("\nSquare:")
print_area(square)
```

This program defines an abstract class Shape with an abstract method area. It then creates two concrete classes Circle and Square that inherit from Shape and implement the area method. The print area function demonstrates polymorphism by accepting any object of type Shape and calling its area method without knowing its specific subclass. Encapsulation is demonstrated by using private attributes in the Circle and Square classes. Abstraction is achieved through the abstract base class Shape.

Advantages of Object Oriented Programming

Despite the rise of various programming models, OOP remains popular in DevOps. Let's look at the main advantages of OOP:

1. **Enables Code Reusability:** The idea of inheritance is one of the critical concepts offered by object-oriented programming. A class's attributes can be passed down through inheritance, eliminating the need for duplication of effort. Doing this prevents the problems associated with repeatedly writing the same code.
2. **Increases Productivity in Software Development:** We can create programs from pre-written, interconnected modules rather than having to start from scratch, which would save time and increase productivity.
3. **Reinforces Security:** The concept of data abstraction in OOPS allows only a small amount of data to be displayed to the user, which is one of OOP's strong points.

4. **Results in Flexible Code:** One advantage of OOP is polymorphism, which allows a piece of code to exist in more than one version.

5. **Productivity of Software Development Increased and Cost of Development Lowered:** Let us look at a simple example. In a market, a person will act like a customer; in a school, a person will act like a student; and in a home, a person will act like a son or daughter. Here, the same person exhibits various behaviors depending on the environment.

Let us look at a simple example. In a market, a person will act like a customer; in a school, a person will act like a student; and in a home, a person will act like a son or daughter. Here, the same person exhibits various behaviors depending on the environment.

Advantages	**Disadvantages**
- We can reuse the code multiple times using class. - Inherit the class to subclass for data redundancy. - It is easy to maintain and modify. - It maintains the security of data. - Low-cost development	- Size is larger than other programs. - It required a lot of effort to created. - It is slower than other programs. - It is not suitable for some sorts of problems

Table 1: Comparison table for Object Oriented Programming

Criticism of Object Oriented Programming

The object-oriented programming model has been criticized by developers for multiple reasons. The largest concern is that OOP overemphasizes the data component of software development and does not focus enough on computation or algorithms. Additionally, OOP code may be more complicated to write and take longer to compile.

Alternative methods to OOP include:

- **Functional programming:** This includes languages such as Erlang and Scala, which are used for telecommunications and fault-tolerant systems.
- **Structured or modular programming:** This includes languages such as PHP and C#.
- **Imperative programming:** This alternative to OOP focuses on function rather than models and includes C++ and Java.
- **Declarative programming:** This programming method involves statements on what the task or desired outcome is but not how to achieve it. Languages include Prolog and Lisp.
- **Logical programming:** This method, which is based mostly on formal logic and uses languages such as Prolog, contains a set of sentences that express facts or rules about a problem domain. It focuses on tasks that can benefit from rule-based logical queries.

Language for Paradigm 1: C#

C# (C-Sharp) is a modern, general-purpose programming language developed by Microsoft. It can be used to perform a wide range of tasks and objectives that span over a variety of professions.

C-Sharp is primarily used on the Windows .NET framework.It supports Object Oriented, Imperative programming, Functional Programming paradigms.

C# was originally designed to rival Java. Judging by the quick rise to popularity and the positive response from both new and seasoned developers, it's safe to say that goal has been achieved.

When was C# created?

With its close to twenty years of age, C# is a newer member of the programming family when compared to more established languages like Python and PHP. The language was created in 2000 by Anders Hejlsberg, a Danish software engineer with a track record of well-known inventions, who worked at Microsoft.

Anders has contributed to the development of a few robust programming languages and tools, such as Microsoft's TypeScript and Delphi, which might serve as a good substitute for Turbo Pascal.

Just behind Java and JavaScript, C# was placed 4th on the PYPL Popularity of Programming Language Index as of November 2022. This index was created using data gathered from Google searches for tutorials on various programming languages.

Additionally, C# is frequently included among the top 10 programming languages according to the TIOBE Index, a report whose data is compiled from a number of well-known search engines, including Google, YouTube, and Bing.

What is C# used for?

C# may be used to construct a wide range of programmes and applications, including games, cloud-based services, mobile and desktop apps, websites, enterprise software, and cloud-based apps. This is similar to other general-purpose programming languages. Many, many games. Despite its remarkable versatility, C# is most typically employed in three areas.

1. **Windows Programs:**
    - Since Microsoft developed C# for Windows, the majority of apps developed with it are Windows desktop programs.
    - The ideal application use case for C# is creating programs tailored to the architecture of the Microsoft platform, as C# applications depend on the Windows.NET framework to run well.
2. **C# in Video Games:**
    - One of the greatest programming languages for games is C#. Popular games like Rimworld using the Unity Game Engine are developed using this language.
    - Unity is the most widely used game engine, and over one-third of the best and most played games in the industry were created using Unity. C# can be used on almost any modern mobile device or console through cross-platform technologies like Xamarin, integrating seamlessly with the Unity engine.
3. **C# for Creating Websites:**
    - C# is frequently used to create open-source software and polished, dynamic websites on the.NET framework.
    - You can still utilize C# to develop a fully functional website even if you're not a fan of the Microsoft architecture. The language's object-oriented design makes it popular for creating highly effective, easily scalable, and maintainable websites.

ECMA and ISO have approved C# as a standard. CLI (Common Language Infrastructure) is the target audience for C#. The runtime environment and executable code are described in the CLI specification.

C# Program to Display Cost of Rectanlge plot using Inheritance:

```
using System;
    class Rectangle
    {
        protected double length;
        protected double width;
        public Rectangle(double l, double w)
        {
            length = l;
            width = w;
        }
        public double GetArea()
        {
            return length * width;
        }
        public void Display()
        {
            Console.WriteLine("Length: {0}", length);
            Console.WriteLine("Width: {0}", width);
            Console.WriteLine("Area: {0}", GetArea());
        }
    }
    class Tabletop : Rectangle
    {
        private double cost;
        public Tabletop(double l, double w)
            : base(l, w)
        { }
        public double costcal()
        {
            double cost;
            cost = GetArea() * 70;
            return cost;
        }
        public void Display()
        {
            base.Display();
            Console.WriteLine("Cost: {0}", costcal());
        }
    }
    class CalRectangle
    {
        static void Main(string[] args)
        {
            Tabletop t = new Tabletop(7.5, 8.03);
            t.Display();
            Console.ReadLine();
        }
    }
```

This program calculates and displays the cost of a rectangular plot and its tabletop, showcasing the use

of inheritance in object-oriented programming. The Tabletop class inherits from Rectangle, introducing a method to calculate the cost based on the area. The main class creates an instance and displays the dimensions and cost.

Advantages of C# Programming:

1. **Time Efficiency:**
   - Perhaps the greatest advantage is how much time you can save by using C# instead of a different programming language.
   - Being that C# is statically typed and easy to read, users can expect to spend less time scouring their scripts for tiny errors that disrupt the function of the application.
   - C# also emphasizes simplicity and efficiency, so programmers can spend less time writing complicated stacks of code that are repeatedly used throughout the project. Top it all off with an extensive memory bank and you've got a time-effective language that can easily reduce labor hours and help you meet tight deadlines without tossing back that third cup of coffee at 2:00am.
2. **Easy to Learn:**
   - Not only may you save time on project development, but learning C# will take less time than learning other, more complex programming languages.
   - C# has a relatively short learning curve for newcomers because of its features and simplicity.
   - This language is an excellent method to get started in the profession and gives ambitious developers a pleasant way to learn programming without feeling overwhelmed or frustrated.
3. **Scalable and Easy to Maintain:**
   - The programming language C# is incredibly maintainable and scalable. C# programmes are more easier to modify and maintain than programmes created in other languages because static codes have precise requirements that must be followed. As a result, C# programmes are consistently reliable.
4. **Great Community:**
   - The value of a supportive community that you can rely on in the realm of coding and programming cannot be emphasised.
   - Programming languages are not a service or platform that offers convenient IT support or a dedicated help line. Programmers are dependent on the assistance of their peers who share their struggles and frustrations.

Apps made with C-sharp are : Microsoft Visual Studio,Paint.NET, FlashDevelop, NMath, Pinta, OpenRA.

Key characteristics and features of C-Sharp that align with OOP

- **Classes:** Blueprints for creating objects with properties (data) and methods (behavior).
- **Objects:** Instances of classes, representing real-world entities or concepts.
- **Inheritance:** Allows new classes (subclasses) to inherit properties and methods from existing classes (base classes), promoting code reuse and extensibility.
- **Polymorphism:** Enables objects of different classes to be treated as the same type, allowing for flexible and adaptable code.
- **Encapsulation:** Bundles data and methods within classes, protecting data integrity and promoting modularity.
- **Abstraction:** Hides implementation details and exposes only essential features, simplifying interfaces and making code more manageable.

Paradigm 2: Imperative Programming Paradigm

What is Imperative Programming?

Imperative programming is the process of giving clearly-defined sequence of instructions to a computer to follow in a predetermined order. The reason it's termed "imperative" is that, as programmers, we specify exactly what the machine must do, and how. Examples of Languages:- Java, C, C++ etc..

Programs in this paradigm consist of a series of statements that modify the program state.

It is based on the notion of changing program state through a series of statements that specify how to manipulate variables and data structures.

Imperative programming languages are very specific, and operation is system-oriented. On the one hand, the code is easy to understand; on the other hand, many lines of source text are required to describe what can be achieved with a fraction of the commands using declarative programming languages.

Imperative programming languages are characterized by their instructive nature and, thus, require significantly more lines of code to express what can be described with just a few instructions in the declarative style. Say you want a chocolate coffee. The imperative approach to do might look like this:

- Brew coffee.
- Melt chocolate.
- Mix melted chocolate into coffee.
- Stir well.
- Add sugar or sweetener.

Actual code example: Let's say we want to calculate the sum of first 10 natural numbers.

```
#include <stdio.h>

int main() {
    int sum = 0;

    for (int i = 1; i <= 10; i++) {
        sum += i;
    }

    printf("Sum of the first 10 natural numbers: %d\n", sum);

    return 0;
}.
```

Here we're telling our program to iterate through 1 to 10 and add all the values to get the result and print it as the sum of all elements.

We're being detailed and specific in our instructions, and that's what imperative programming stands for.

Key Features and Concepts of Imperative Programming:

1. **State and Variables:**
    - **Description:** Imperative programming involves managing and altering the program's state by utilizing variables. These variables serve as containers capable of holding values that can be dynamically changed during the course of program execution.
    - **Explanation:** The ability to work with mutable variables allows programmers to track and manipulate the state of the program as it progresses.

2. **Assignment Statements:**
    - **Description:** Imperative programs make use of assignment statements to update the values held by variables. These statements play a crucial role in assigning new values to variables or modifying their existing values.
    - **Explanation:** Through assignment statements, developers can manipulate and control the flow of data within the program, facilitating dynamic changes in variable values.
3. **Control Flow:**
    - **Description:** Imperative programming employs control flow structures to dictate the sequence in which statements are executed. This includes the use of loops (e.g., for, while) for repetitive tasks and conditionals (e.g., if-else) for decision-making processes.
    - **Explanation:** Control flow mechanisms enable the program to adapt and respond to different scenarios, allowing for efficient and flexible execution of statements.
4. **Procedures and Subroutines:**
    - **Description:** Imperative programming encourages the organization of code into reusable procedures or subroutines. These are modular blocks of code that can be called and executed at various points within the program.
    - **Explanation:** The use of procedures promotes code modularity, reusability, and maintainability, as common tasks can be encapsulated into distinct, callable units.
5. **Mutable State:**
    - **Description:** In imperative programming, the program's state is subject to modification during execution. This implies that variables can be altered, leading to side effects that influence the behavior of the program.
    - **Explanation:** The ability to modify state allows for dynamic adaptation of variables, although it introduces considerations related to potential side effects that may impact program logic.

Advantages and Disadvantages of Imperative Programming:

- There are a lot of computer languages in use today that are based on the imperative programming paradigm.
- This is due, in part, to the fact that the method is the original form of programming. However, there are still some useful benefits to the imperative paradigm despite the existence of competing paradigms.
- Because the code may be read as a step-by-step tutorial, the languages are comparatively simple to learn. As such, the first language that programmers often learn in their training is an imperative language.
- Easily readable content is essential for daily activities. In the end, it shouldn't be possible for one employee to handle application optimisation and maintenance alone; multiple staff members should be able to handle these tasks without too much trouble, even if they haven't built the code from beginning.
- Procedural programming has the drawback of rapidly increasing the amount of code required to handle increasingly complicated issues. The volume causes it to become complex, but it is still easy to read.
- Because execution and programming are not as clearly separated as they are in the declarative style, further interventions may result in unintentional errors. In pure imperative programming, extensions are also more challenging to develop than in declarative language, where methods are available for adding them individually.

Advantages	Disadvantages
- Relatively easy to read and learn. - Conceptual model (solution path) is very easy for beginners to understand. - Specific application characteristics may be taken into consideration.	- The code grows incredibly long and confusing very fast. - Higher risk of errors when editing. - Because of system-oriented programming, application development is impeded by maintenance. - Extension and optimization are more challenging.

Table 2: Comparison table for Imperative Programming

Evolution of Imperative Programming:

The main objective of the imperative programming paradigm evolution is to decrease expenses of program development as well as maintenance. Programming paradigms shift one another when the old programming paradigm makes developing of a program system of new complexity level impossible.

1. Early Machine Languages (1940s-1950s):
   - First imperative languages, directly controlling hardware.
   - Limited abstraction and maintainability.
2. Procedural Programming (1950s-1960s):
   - Introduction of procedures for code organization and reusability.
   - Key languages: FORTRAN, ALGOL, COBOL.
3. Structured Programming (1960s-1970s):
   - Focus on control flow structures (if-else, loops) for readability.
   - Key languages: Pascal, C.
4. Modular Programming (1970s-1980s):
   - Emphasis on breaking down programs into independent modules for reusability.
   - Key languages: Modula-2, Ada.
5. Modern Imperative Programming (1980s-present):
   - Blends with other paradigms like OOP and functional programming.
   - Focus on readability, maintainability, and hardware efficiency.
   - Key languages: Go, Rust.

Language for Paradigm 2: FORTRAN

Fortran is a general purpose programming language, mainly intended for mathematical computations in science applications. It is developed by John Backus for IBM to remove the obstacles presented by machine code in the creation of complex programs. Fortran is an acronym for FORmula TRANslation. Fortran(1950) was the first high-level programming language. The work on Fortran started in the 1950's at IBM and there have been many versions since. The most common Fortran version today is still Fortran 77, although Fortran 90 is growing in popularity.

Along with declarations, expressions, and statements, it supported arrays, subroutines, "do" loops.

Why learn FORTRAN?

It is the dominant programming language used in scientific applications. So it is important for physics (or engineering) students to learn Fortran code. Fortran is the most enduring computer programming language in history. One of the main reasons Fortran has survived and will survive is software inertia. Once a company has spent many people-years and perhaps millions of dollars on a software product, it is unlikely to try to translate the software to a different language. Reliable software translation is a very difficult task and there's 40 years of Fortran code to replace!

Portability:

A major advantage Fortran has is that it is standardized by ANSI (American National Standards Institute) and ISO (International Standards Organization). Consequently, if your program is written in ANSI Fortran 77 then it will run on any computer that has a Fortran 77 compiler. Thus, Fortran programs are portable across computer platforms.

A Fortran program generally consists of a main program (or driver) and possibly several subprograms(or procedures or subroutines). The STRUCTURE of the main program is:

```
program name
declarations
statements
stop
end
```

FORTRAN 77 BASICS: A Fortran program is just a sequence of lines of text. The code has to follow a certain syntax to be a valid Fortran program. We will now look at a simple example where we calculate the area of a circle:

```
      PROGRAM CIRCLE
      REAL R, AREA
C THIS PROGRAM READS A REAL NUMBER R AND PRINTS
C THE AREA OF A CIRCLE WITH RADIUS R.
      WRITE (*,*) 'GIVE RADIUS R:'
      READ (*,*) R
      AREA = 3.14159*R*R
      WRITE (*,*) 'AREA = ', AREA
      STOP
      END
```

The lines that begin with a "c" are comments and have no purpose other than to make the program more readable for humans. Now we see that this follows the imperative programming paradigm as the instructions are clearly defined in a sequence.

FORTRAN 90+ Version:

```
      program circle
                real :: r, area
```

```
            ! This program reads a real number r and prints
            ! the area of a circle with radius r.

            write (*,*) 'Give radius r:'
            read (*,*) r
            area = 3.14159 * r * r
            write (*,*) 'Area = ', area
            stop
end program circle
```

This is similar as the Fortran 77 but the end statement should specify the program name.

What is FORTRAN used for?

Science and engineering are the two fields that embraced computing the earliest and utilise Fortran the most. These include computational fluid dynamics, applied arithmetic, statistics, economics, and numerical weather and ocean prediction. High performance computing is dominated by the language Fortran, which is used to benchmark the world's fastest supercomputers.

While Fortran does embody some features of other programming paradigms, it primarily adheres to the imperative programming paradigm. Here's a breakdown:

Imperative paradigm: Core features: Explicit instructions, mutable variables, state change, control flow constructs, procedures.

Fortran's alignment: Fortran heavily relies on these features, directly telling the computer what to do step-by-step, modifying variables, controlling execution flow, and organizing code into procedures. Other potential paradigms and Fortran's relationship:

1. Object-oriented programming (OOP): While Fortran has incorporated some OOP elements like classes and modules, it doesn't fully embrace the core OOP principles like inheritance, polymorphism, and encapsulation. It remains primarily imperative in its code structure and execution flow.

2. Declarative programming: There are limited declarative aspects in Fortran, mainly in the form of array initialization syntax. However, the overall program execution still follows the imperative approach of directing the computer's actions step-by-step.

3. Functional programming: Functional programming focuses on immutable data and pure functions, while Fortran uses mutability and side effects extensively. These fundamental differences preclude classifying Fortran as functional. Therefore, although Fortran might exhibit some influences from other paradigms, its core design, features, and focus on explicit instructions firmly place it within the realm of imperative programming.

Characteristics and Features of Fortran or How does it embody IMPERATIVE principles:

1. Emphasis on Statements and Explicit Instructions:

   Fortran code consists of statements that directly tell the computer what to do. These statements include assignments (`a = b + c`), function calls, conditional branches (`IF/ELSE`), and loops (`DO WHILE`). The order of statements in the code determines the program's flow and logic. The programmer explicitly controls the sequence of execution.

2. Mutable Variables and State Change:

   Fortran variables can have their values changed during program execution. This allows for data manipulation, modification of memory state, and tracking changing information within the program. The ability to mutate variables is a fundamental characteristic of imperative programming, allowing for dynamic computation and manipulation of data.

3. Control Flow Constructs and Algorithmic Steps:

   Fortran provides various control flow structures like `IF/ELSE` and `DO` loops to control the program's execution flow based on conditions and repetitive tasks. These constructs enable the expression of precise algorithms involving well-defined steps and computations, a core principle of the imperative paradigm.

4. Procedural Organization and Modularization:

   Fortran uses procedures (`SUBROUTINES` and `FUNCTIONS`) to break down large programs into reusable modules, enhancing code organization and maintainability. These procedures encapsulate specific tasks and can be called upon as needed within the program, reflecting the imperative focus on breaking down problems into smaller, sequential steps.

5. Focus on Scientific Computation:

   Fortran primarily caters to scientific and engineering applications where precise numerical algorithms and computations are crucial. The imperative paradigm, with its emphasis on precise instructions and control flow, aligns well with the need for step-by-step calculations and data manipulation in these domains.

Real-world problem domains where FORTRAN excels:

1. **Scientific Computing:**
   - **Numerical computations:**
     - Solving complex mathematical equations (linear algebra, differential equations, numerical integration).
     - Modeling physical systems (fluid dynamics, weather forecasting, molecular dynamics).
   - **Scientific simulations:**
     - Astrophysics, climate modeling, quantum mechanics, computational chemistry.
2. **Engineering Applications:**
   - **Aerospace and mechanical engineering:**
     - Structural analysis, fluid dynamics simulations, heat transfer calculations.
   - **Electrical engineering:**
     - Circuit analysis, power systems modeling, control systems design.
   - **Civil engineering:**
     - Structural design, finite element analysis, hydraulic simulations.
3. **High-Performance Computing (HPC):**
   - **Supercomputers and clusters:**
     - Computationally intensive tasks in science and engineering.
     - Weather forecasting, climate research, drug discovery, nuclear simulations.
4. **Data Analysis and Visualization:**
   - **Scientific data processing:**
     - Handling large datasets, statistical analysis, generating visualizations.
   - **Data visualization libraries:**
     - Plotting graphs and charts for scientific research.
5. **Legacy Code Maintenance:**
   - **Existing scientific and engineering codebases:**
     - Large body of FORTRAN code still in use, requiring maintenance and updates.

Analysis

Analysis of OOP and Imperative Programming Paradigms

Strengths

*OOP (C#):

- **Code organization and reusability:** Classes and objects group related data and operations, promoting modularity and code reuse.
- **Abstraction and encapsulation:** Hiding implementation details makes code cleaner and easier to maintain.
- **Polymorphism and inheritance:** Enables flexible and adaptive code through shared behavior and specialization.
- **Suitability for complex systems:** Effectively models real-world entities and relationships in complex software.
- **Rich development environment:** C# integrates with Visual Studio, offering strong tooling and debugging capabilities.

Imperative (Fortran):

- **Efficiency and control:** Imperative approach directly controls program flow, maximizing efficiency for numerical algorithms.
- **High-precision calculations:** Fortran supports high-precision floating-point numbers, critical for scientific computing.
- **Extensive libraries:** Wide range of libraries specifically designed for scientific and engineering domains.
- **Legacy compatibility:** Large amount of existing scientific and engineering code written in Fortran.
- **Simple syntax and learning curve:** Imperative concepts are often easier to grasp for beginners.

Weaknesses

OOP (C#):

- **Complexity overhead:** Overuse of OOP principles can lead to complex code structures and design challenges.
- **Performance considerations:** OOP abstractions can sometimes add overhead and impact performance.
- **Steeper learning curve:** OOP concepts take more time to understand and master compared to imperative programming.

Imperative (Fortran):

- **Maintainability concerns:** Large, code-centric programs can be difficult to maintain due to tight coupling and lack of abstraction.
- **Readability challenges:** Imperative code can become cluttered and harder to understand for complex tasks.
- **Limited applicability:** Mainly optimized for scientific and engineering domains, less suitable for broader software development.

Notable Features

OOP (C#):

- **Classes and objects:** The core building blocks of OOP, encapsulating data and behavior.
- **Inheritance:** Allows sub-classes to inherit properties and methods from parent classes.
- **Polymorphism:** Enables objects of different types to be treated in a similar way.
- **Interfaces:** Define contracts for behavior without implementation details.
- **Generics:** Provide type-safe collections and reusable code without code duplication.

Imperative (Fortran):

- **Procedures and functions:** Reusable blocks of code for modularity.
- **Control flow structures:** if, while, for loops control program execution.
- **Arrays and matrices:** Efficient data structures for numerical calculations.
- **High-precision floating-point arithmetic:** Handles complex scientific computations with accuracy.
- **Domain-specific libraries:** Extensive libraries for scientific and engineering domains like BLAS and LAPACK.

Future trends and considerations for OOP (C#) and Imperative (FORTRAN) paradigms

OOP (C#):

Trends:

- Focus on cloud-native development: C#'s integration with Azure will strengthen its role in cloud-based applications.
- Cross-platform expansion: .NET 5/6 and beyond will drive further cross-platform capabilities, reaching more platforms and devices.
- Integration with AI and machine learning: C# will become increasingly important for building intelligent systems using ML.NET and other frameworks.
- Growth in game development: Unity's popularity will continue to fuel C#'s use in game development across platforms.
- Adoption of functional programming features: C# will likely incorporate more functional programming elements for concise and expressive code.

Considerations:

- Balancing complexity and maintainability: Overuse of OOP can lead to complex code structures; careful design is crucial for maintainability.
- Performance optimization: OOP abstractions can sometimes impact performance; understanding overhead and optimization techniques is essential.

Imperative (Fortran):

Trends:

- Continued dominance in scientific computing: Fortran will remain a key language for numerical computations and simulations.
- Collaboration with other languages: Fortran will increasingly interact with Python and other languages for data analysis and visualization.
- Adaptation to modern hardware: Fortran compilers will evolve to exploit newer processor architectures and parallel computing capabilities.
- Integration with cloud environments: Fortran will find use in cloud-based scientific computing platforms.

Considerations:

- Modernization challenges: Updating legacy Fortran code can be complex and time-consuming.
- Attracting new developers: Fortran's popularity among younger developers is relatively low; efforts to modernize and promote the language are needed.
- Competition from newer languages: Julia and Python are gaining traction in scientific computing, posing challenges to Fortran's dominance.

Comparison

Object Oriented Programming (OOP) is a super set of Imperative Programming. It follows all characteristics of IP with some extra features. Those extra features are:

- Everything is an Object.
- Each Object contains Some Data Fields and Methods.
- OOPs Concepts: Abstraction, Encapsulation, Inheritance, and Polymorphism.

Similarities

- Both define programs as sequences of instructions.
- Both use variables to store and manipulate data.
- Both can be used to solve a wide range of problems.

Differences:

1. Focus:
    - Imperative: How instructions change program state.
    - OOP: Data and operations bundled in objects
2. Code organization:
    - Imperative: Procedural, step-by-step.
    - OOP: Modular, object-oriented.
3. Data access
    - Imperative: Open access to variables.
    - OOP: Controlled access through object methods.
4. Code Reusability
    - Imperative: Limited through functions.
    - OOP: High through inheritance and polymorphism

Suitable Problems:

- Imperative: Procedural tasks, step-by-step calculations, simple algorithms.
- OOP: Real-world scenarios with complex entities and relationships, simulations, graphical user interfaces.

Which is better?

- There's no one-size-fits-all answer. Choose the paradigm that:
- **Matches the problem complexity:** OOP for complex, interrelated entities; imperative for simpler, procedural tasks.
- **Promotes code maintainability and reusability:** OOP offers greater modularity and reuse potential.
- **Suits your team's expertise and preferences:** Different paradigms have different learning curves.

Challenges Faced

Changing between languages such as Fortran and C# requires getting used to new grammar structures. C# syntax differs greatly from imperative Fortran syntax in that it emphasises object-oriented ideas. These adjustments were particularaly challenging.

One challenge was translating theoretical information into real-world situations. I found it challenging to apply OOP concepts in C# because the language differed slightly from C++.

Full study of programming paradigms and related languages is necessary for a full report. It took time and effort to compile information on the historical background, characteristics, strengths, and shortcomings.

It takes serious thought to address these unique advantages and difficulties that each language presents as well as to comprehend their applicability in a variety of contexts.

A thorough report requires efficient time management. It was difficult to find a balance between researching, drafting, and revising.

However understanding about the paradigms was a very good basement knowledge and will be helpful in learing a new language further in the future. Also publishing this online would be very helpful for other people to easily understand without taking a lot of time.

Conclusion

We have made the following conlusions.

1. We have analyzed what a paradigm is and why we need to study it, exploring its applications in real-world problems. Concluding that each programming paradigm introduces constraints, shaping programming within a specific framework. These constraints, while limiting some capabilities, reduce the complexity of program development. The importance of programming paradigms lies in dictating code structuring, problem-solving techniques, and design philosophy.

2. We have concluded that the evolution of programming paradigms was influenced by the search for new ways to solve real-world problems. Each paradigm introduces constraints, shaping programming within a specific framework.

3. We have learned the implementation of all object-oriented paradigms and their types, understanding the differences between each of them. The advantages of OOP and their disadvantages provide us knowledge to decide whether to choose them to solve a problem.

4. Despite the fact that C# is not commonly known to everyone, it is developed by Microsoft and is used for the .NET framework. We have also analyzed other usages of C# and provided a sample program to demonstrate the difference between C# and C++.

5. We have analyzed the Imperative paradigm as the process of giving instructions to a computer and provided sample programs. Key features of IP include control flow, procedures and subroutines, and mutable state. We traced the evolution of Imperative Programming from early machine languages to modern-day imperative programming languages.

6. We have concluded that Fortran is a general-purpose programming language mainly intended for mathematical computations and supports Imperative, OOP, declarative, and functional paradigms. Different versions of FORTRAN cater to various programming needs.

7. We have analyzed the real-world problem domains where C# and FORTRAN languages excel, along with future trends:

   (a) C# trends include a focus on cloud-native development, cross-platform expansion, integration with AI and machine learning, growth in game development, and adoption of functional programming features. Considerations involve balancing complexity and maintainability, as well as performance optimization.

(b) Fortran maintains dominance in scientific computing, collaborates with Python, adapts to modern hardware, and integrates with cloud environments. Challenges include modernization, attracting new developers, and facing competition from newer languages.

References

- https://www.freecodecamp.org/news/an-introduction-to-programming-paradigms/#what-is-a-programming-
- https://www.spiceworks.com/tech/devops/articles/object-oriented-programming/
- https://deb.ugc.ac.in/Uploads/SelfLearning/HEI-Exempted-U-0748/HEI-Exempted-U-0748_SelfLearning_20231026135924.pdf
- https://www.inscc.utah.edu/~krueger/6150/Fortran77_tutorial.pdf
- https://en.wikipedia.org/wiki/Imperative_programming
- https://iopscience.iop.org/article/10.1088/1757-899X/714/1/012001/pdf
- https://en.wikipedia.org/wiki/Modular_programming
- https://www.youtube.com/watch?v=__2UgFNYgf8
- https://www.pluralsight.com/blog/software-development/everything-you-need-to-know-about-c
- https://medium.com/@Ariobarxan/what-is-a-programming-paradigm-ec6c5879952b
- https://www.digitalocean.com/community/tutorials/functional-imperative-object-oriented-programming
- https://www.javatpoint.com/c-sharp-tutorial
- https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP#:~:text=Object%2Doriented%20programming%20(OOP)%20is%20a%20computer%20programming%20model,has%20unique%20attributes%20and%20behavior
- https://en.wikipedia.org/wiki/History_of_programming_languages
- https://www.sanfoundry.com/csharp-program-cost-rectangle-inheritance/
- https://fortran-lang.org/