Lecture 1: Programming Paradigms

COMP26020 Part 1 (C) Lecture Notes

Pierre Olivier

These notes summarise the important points mentioned in the lectures. They are supposed to be a help for revising and not a way to avoid attending the live lectures and watching the videos. In other words, live lectures and videos may include examinable content that is not present in these notes.

The slides for this lecture are available here:
https://olivierpierre.github.io/comp26020-lectures/01-programming-paradigms.

Videos and recordings of live sessions can be found on the video portal: https://video.manchester.ac.uk/lectures.

In this lecture we discuss the concept of programming paradigm and how it relates to programming languages.

Programming Paradigms

To put it simply a programming paradigm is a **style of programming**. This style defines how the programmer describes a program within its source code: the data that it uses, as well as the computations manipulating this data. Paradigms are high level categories used to classify programming languages.

Relation to Programming Languages

All programming languages fall within at least one paradigm. What this means is that they make it easy to program in the style defined by the paradigm in question. For example a lot of operations look pretty similar in C++ and in C# because both languages fall within the object-oriented programming paradigm. Also, a lot of languages can be classified within several paradigms. For example, objective caml belongs to the object-oriented as well as functional programming paradigms.

Suitability of Paradigms/Language to Software Engineering Problems

Given a particular software engineering problem, let's say one has to build a particular program, there are some programming paradigms that are much more efficient at solving that problem. For example one cannot really write an operating system with a logical programming language.

JavaScript Example

Let's illustrate the fact that a paradigm defines a programming style with an example. We simply want to multiply by 2 each element of an array. JavaScript allows writing code according to the **imperative** programming paradigm:

```javascript
function mult_by_two_imperative (array) {
  let results = []
  for (let i = 0; i < array.length; i++) {
    results.push(array[i] * 2)
  }
  return results
}
```

The imperative paradigm requires describing step by step all operations performed by the program. So we use a for loop to iterate over all the elements and multiply each by two. In some sense this is very close to what happens on the CPU, which is executing instructions one step at a time.

JavaScript also allows writing code according to the **declarative** paradigm:

```
function mult_by_two_declarative (array) {
  return array.map((item) => item * 2)
}
```

With the declarative paradigm we describe high level operations to be performed on each element of the array: each item is multiplied by two. This is more abstract and also more concise than imperative programming: it's a one-liner

A Programming Paradigm is a Programming Style

The paradigm characterises how the programmer defines the **computations**. It's generally a sequence of statement performing various operations, in various order – for example sequentially or in parallel. These computations can be decomposed into units named functions. The computations can also be described in terms of how the result should look like. The paradigm also defines how to describe the **data** that is manipulated by the computations. It answers questions like what are the basic types, can we define custom data structures by composing these types, can the functions manipulate only a local set of variable or do they have access to a global state.

Picking a Programming Paradigm

Because some paradigms are more efficient than others to solve a given problem, when one needs to develop software, choosing a language (in other words choosing a paradigm) has huge consequences on the efficiency, size, complexity and clarity of the code.