Unit 1c: Imperative Approach (Procedural)

1 Introduction

As outlined previously the basis of the imperative approach to solving problems is: *'First do this and next do that'*.

We have already know about two ways of implementing an imperative approach: procedural and object oriented. In fact, while they are different approaches, the OO approach incorporates programming concepts originally identified in the procedural paradigm (and adds some specific ideas).

2 Programming Concepts

Before we investigate the two approaches to implementing an imperative solution let's remind ourselves of familiar programming concepts and acknowledge where they came from.

2.1 Variables & Types

As initially described in the Von Neumann architecture, internal memory is used to record the current state of the program as it executes i.e. data values; additionally, we need a way of remembering user input. Variables are used to identify memory locations where values can be stored, accessed and updated.

The number of memory locations required to hold a particular variable value is determined by the type of the variable i.e. a number typically requires less storage than a string.

We will discuss types in much more detail later in the module.

2.2 Statements

A statement is a single piece of executable code designed to progress the state changes required to move the program from an initial state to desired results.

Statements can basically be categorized as:

- Assignment – setting/changing a variable's value
- IO – acknowledging data from an imput device e.g. keyboard; or, sending information to an output device e.g. screen
- Expressions – performing a 'calculation'

This is sufficient if a program is required to simply execute statements from the first to the last in order. However, typically we require a program to deal with alternatives or repeat part of the process. Addressing these issues allowed the development of more flexible and complex software.

Initially the issue was resolved by including a jump statement (GOTO) which allowed a program to jump back to previous statements or jump forward; and a test statement to determine whether the jump should be executed (IF).

3 Structured Programming

As programs became larger and more complex the dangers inherent in this solution – spaghetti code – were discussed in an open letter in 1968 by Edsger Dijkstra:

'Go To Statement Considered Harmful'.

Dijkstra introduced the term "structured programming" which describes a better, clearer solution to the requirements of more complex software.

The basic principle is that any problem with a logical solution only required three ideas for organizing the solution: sequence; selection, and, iteration.

While initially there was some opposition to the adoption of this simpler approach, it became the dominant way of writing software in the 70s and 80s. So much so that Niklaus Wirth identified Stepwise Refinement as a way of designing a solution by starting with an overall description of the solution and, at each stage, refining the solution as a more detailed sequence of statements, a choice of alternative paths, and, a block of statements to be repeated.

The selection (IF, CASE) and repetition (FOR, WHILE, UNTIL) control structures became an integral part of modern programming languages.

4 Procedural Programming

As a single High Level Language statement is essentially an abstraction of multiple machine code instructions, it became clear that a facility to group statements under a single name and use the name as a statement call/execute that group of statements when they were required was desirable. This concept of a procedure (routine, function) allowed software to be designed as separate units.

The focus of a procedural approach is to break down a programming task into a collection of variables, data structures, and subroutines. This relies on procedures that operate on data, behaviour, and the data being separate; any given procedure might be called at any point during a program's execution, including by other procedures or itself.

Handles complexity through *top down* design (functional decomposition) - start with a problem (procedure) and then systematically break the problem down into sub problems (sub procedures), until sub problems are straightforward enough to be solved by the corresponding sub procedure.

Conceptually this is quite different from OO approach: with Procedural you think about what something *does* and what it *does it to*; with OO you think about what something *is* and how it can be *used*.

5 Modular Design/Programming

A modular approach to designing a solution a problem was developed in the 60s and involves identifying separate units of concern which address part of the problem solution and are combined to create the overall solution. Ideally programming languages provide a greater facility than a procedure/function – a module which can be separately translated and executed and can be reused across multiple applications. This leads to the idea of a library of reusable code.

6 Summary

The combination of modular & structured design and implementation ideas utilised in procedural programming became the dominant approach to software development until the late 80s when object oriented ideas came to the forefront.