OCR Computer Science A Level

1.2.4 Types of Programming Language

Intermediate Notes

Specification:

1.2.4 a)

- **Programming paradigms**
  - Need for these paradigms
  - Characteristics of these paradigms

1.2.4 b)

- **Procedural languages**

1.2.4 c)

- **Assembly language**
  - Following LMC programs
  - Writing LMC programs

1.2.4 d)

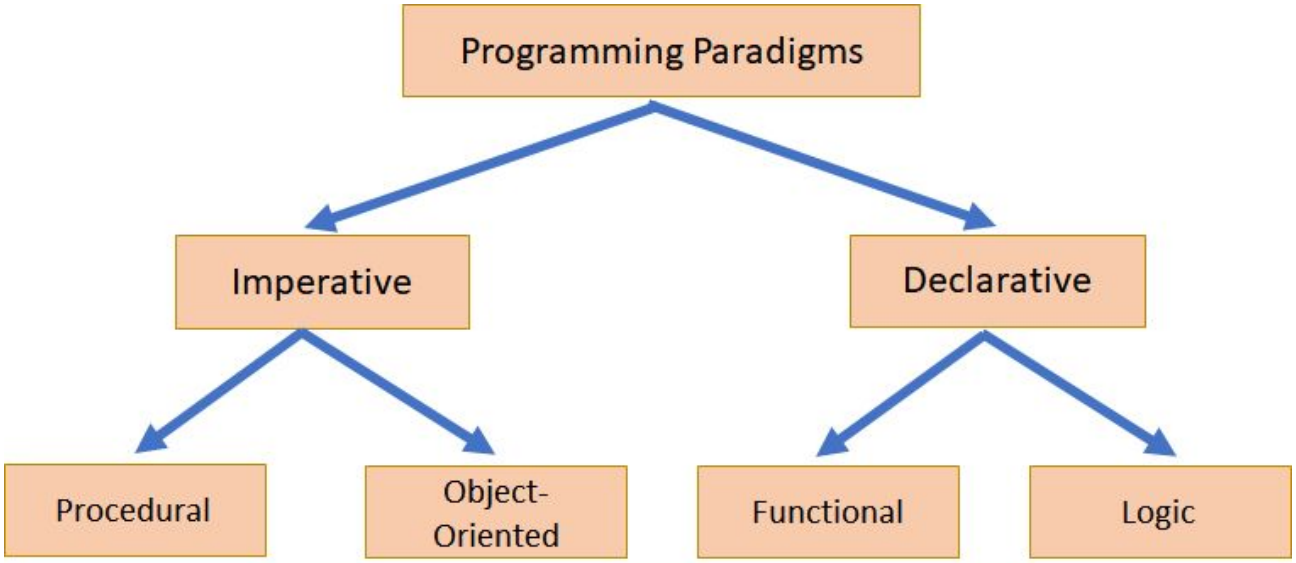
- **Modes of addressing memory**
  - Intermediate, Direct, Indirect, Indexed

1.2.4. e)

- **Object-oriented languages**
  - Classes
  - Objects
  - Methods
  - Attributes
  - Inheritance
  - Encapsulation
  - Polymorphism

Programming Paradigms

Programming paradigms are different approaches to using a programming language to solve a problem. They are split into two broad categories - imperative and declarative - which can be broken down further into more specific paradigms. The paradigm used depends on the type of problem that needs solving.



Imperative

Imperative programming paradigms use code that clearly specifies the actions to be performed.

Procedural

Procedural programming is one of the most widely-used paradigms as it can be applied to a wide range of problems and is easy to write and interpret. This type of programming uses a sequence of instructions, often contained within procedures. These instructions are carried out in a step-by-step manner.

Synoptic Link

You will encounter the programming constructs discussed here again in **2.2**.

Object-Oriented

Object-oriented programming (referred to as OOP) is another popular paradigm as it is applicable to problems which can be broken into reusable components with similar characteristics. OOP is built on objects formed from classes which have attributes and methods. OOP focuses on making programs that are reusable and easy to update and maintain.

Synoptic Link

Declarative languages and OOP are built around the principle of abstraction, which is explored in 2.1.

Declarative

Declarative programming states the desired result and the programming language determines how best to obtain the result. The details about how it is obtained are abstracted from the user. Declarative programming is common in expert systems and artificial intelligence.

Functional

Functional programming uses functions, which form the core of the program. Programs are made up of function calls, often combined within each other. Functional programming is closely linked to mathematics.

Logic

Logic languages use code which defines a set of facts and rules based on the problem. Queries are used to find answers to problems.

Procedural Language

Procedural programming is used widely in software development as it is simple to implement and applicable to most problems. However, it is not possible to solve all kinds of problems with procedural languages or may be inefficient to do so.
Procedural languages use traditional data types which are built into the language and also provides data structures. Structured programming is a popular subsection of procedural programming in which the control flow is given by four main programming structures:

- Sequence
  Code is executed line-by-line, from top to bottom.
- Selection
  A certain block of code is run if a specific condition is met, using IF statements.
- Iteration
  A block of code is executed a certain number of times or while a condition is met. Iteration uses FOR, WHILE or REPEAT UNTIL loops.
- Recursion
  Functions are expressed in terms of themselves. Functions are executed until a certain condition known as a base case is met.

Assembly Language

Assembly language is a low level language that is the next level up from machine code.

Assembly language uses mnemonics, which makes it easier to use than direct machine code. Each mnemonic is an abbreviation for a machine code instruction and is represented by a numeric code. However, the commands that assembly language uses are processor-specific.

Each line in assembly language is equivalent to one line of machine code.
Below is a list of the mnemonics you need to be aware of and be able to use:

Mnemonic	Instruction	Function
ADD	Add	Add the value at the given memory address to the value in the Accumulator
SUB	Subtract	Subtract the value at the given memory address from the value in the Accumulator
STA	Store	Store the value in the Accumulator at the given memory address
LDA	Load	Load the value at the given memory address into the Accumulator
INP	Input	Allows the user to input a value which will be held in the Accumulator
OUT	Output	Prints the value currently held in the Accumulator
HLT	Halt	Stops the program at that line, preventing the rest of the code from executing.
DAT	Data	Creates a flag with a label at which data is stored.
BRZ	Branch if zero	Branches to a given address if the value in the Accumulator is zero. This is a conditional branch.
BRP	Branch if positive	Branches to a given address if the value in the Accumulator is positive. This is a conditional branch.
BRA	Branch always	Branches to a given address no matter the value in the Accumulator. This is an unconditional branch.

Below is an example of an LMC program which returns the remainder, called the modulus, when num1 is divided by num2.

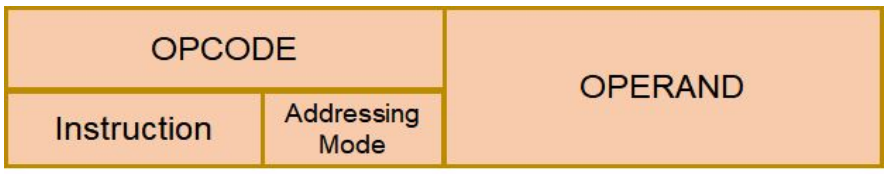
> **Note**
>
> In a high-level language, this program would be represented as a single instruction - MOD.

```
          INP
          STA num1
          INP
          STA num2
          LDA num1
positive  STA num1          // branches to the 'positive' flag,
          SUB num2          subtracting num2 while the result
          BRP positive      of num1 minus num2 is positive
          LDA num1
          OUT
          HLT
    num1  DAT
    num2  DAT
```

Modes of Addressing Memory

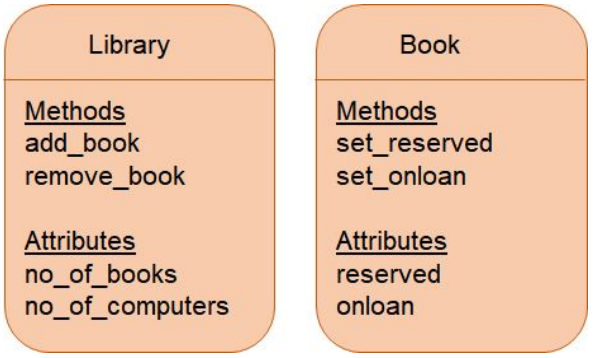
Machine code instructions are made up of two parts, the opcode and operand. The opcode specifies the instruction to be performed and the addressing mode. The operand holds a value related to the data on which the instruction is to be performed. The addressing mode specifies how the operand should be interpreted.
Addressing modes are used to allow for a much greater number of data storage locations.
There are four addressing modes you need to know:

- Immediate Addressing
  The operand is the actual value upon which the instruction is to be performed, represented in binary,
- Direct Addressing
  The operand gives the address which holds the value upon which the instruction is to be performed. Direct addressing is used in LMC.
- Indirect Addressing
  The operand gives the address of a register which holds another address, where the data is located.
- Indexed Addressing
  An index register is used, which stores a certain value. The address of the operand is determined by adding the operand to the index register.



Object Oriented Language

Object-oriented languages are built around the idea of reusable classes. A class is a template for an object and defines the state and behaviour of an object. State is given by attributes which give an object's properties. Behaviour is defined by the methods associated with a class, which describe the actions it can perform.
Classes can be used to create objects by a process called instantiation. An object is a particular instance of a class, and a class can be used to create multiple objects.
A class is associated with an entity. For example, take a class called 'Library'. It could have attributes 'number_of_books', 'number_of_computers' and methods 'add_book' and 'remove_book' amongst others. Similarly, 'Book' could also be a class.



set_reserved and set_onloan are a special type of method called Setters. A setter is a method that sets the value of a particular attribute. In this example, 'set_reserved' would set the attribute 'Reserved' to 'True' if someone was to reserve that book. A getter is another special method used in OOP which retrieves the value of a given attribute.

Getters and setters ensure attributes cannot be directly accessed and edited by users. This is called encapsulation, in which attributes are declared as private so can only be altered by public methods.
Every class must also have a constructor method, called 'new'. A constructor allows a new object to be created.

Below is part of the pseudocode for the 'Book' class described above:

```
class Book:
     private reserved
     private onLoan
     private author
     private title
     public procedure new(title,author,reserved,onLoan)
          title = givenTitle
          author= givenAuthor
          reserved = givenReserved
          onLoan = givenOnLoan
     endprocedure
     public function set_reserved()
          reserved=True
     end function
Endclass
```

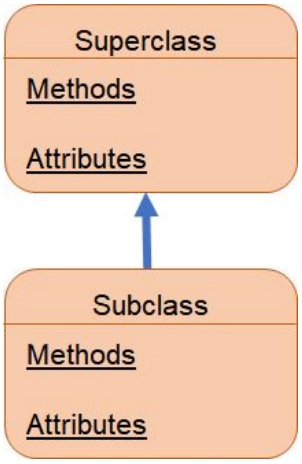
This is how a constructor is used to create a new object called 'myBook' from the 'Book' class:

```
myBook = new Book('Great Expectations', 'Charles Dickens','False', False')
```

When a book is reserved, the following code would be used to call the setter function:

```
myBook.set_reserved()
```

Another property of object-oriented programming is inheritance. When a class inherits from another, the subclass will possess all of the methods and attributes of the superclass and can also have its own additional properties. This feature of OOP allows programmers to effectively reuse certain components and properties while making some changes.



Inheritance would be expressed as:

```
class Biography inherits Book
```

Polymorphism is a property of OOP that means objects can behave differently depending on their class. There are two categories of polymorphism: overriding and overloading.
Overriding is redefining a method within a subclass and altering the code so that it functions differently and produces a different output.
Overloading is passing in different parameters into a method.

Advantages

- OOP allows for a high level of reusability, which makes it useful for projects where there are multiple, similar components.
- Classes can also be used across multiple projects.
- Encapsulation makes the code more reliable by protecting attributes from being directly accessed.
- OOP requires advance planning and a thorough design can produce a higher-quality piece of software with fewer vulnerabilities.
- The modular structure used in OOP makes it easy to maintain and update.
- Once classes have been created and tested, they can be reused as a black box which saves time and effort.

Disadvantages

- This is a different style of programming and so requires an alternative style of thinking.
- OOP is not suited to all types of problems and can sometimes produce a longer, more inefficient program.
- Generally unsuitable for smaller problems.