

# OCR Computer Science A Level

## 2.2.1 Programming Techniques

### Concise Notes



## **Specification:**

### **2.2.1 a)**

- **Programming constructs**
  - Sequence
  - Iteration
  - Branching

### **2.2.1 b)**

- **Recursion**
  - How it can be used
  - How it compares to an iterative approach

### **2.2.1 c)**

- **Global and local variables**

### **2.2.1 d)**

- **Modularity, functions and procedures**
  - Parameter passing by value
  - Parameter passing by reference

### **2.2.1 e)**

- **Use of an IDE to develop / debug a program**

### **2.2.1 f)**

- **Use of object-oriented techniques**



## Programming Constructs

- **Sequence**
  - Code is executed **line-by-line**, from top to bottom
- **Branching**
  - Particular block of code is run **if a specific condition is met**
  - Uses IF and ELSE statements
- **Iteration**
  - Can be either:
    - Count-controlled
      - Block of code executed a **certain number of times**
    - Condition-controlled
      - Block of code is executed **while a condition is met**
  - Uses FOR, WHILE or REPEAT UNTIL loops

## Recursion

- Programming construct in which a **subroutine calls itself** during its execution
- Continues until a **stopping condition** is met

Advantages	Disadvantages
Can be represented in fewer lines of code	Risk of stack overflow if memory runs out
Easier to express some functions recursively eg. Factorial	Difficult to trace

## Global and Local Variables

- Variables can be defined with either global or local scope
- **Scope** is the **section of code in which the variable can be accessed**
- A local variable within a subroutine takes precedence over a global variable with the same name

### Local Variables

- Can only be **accessed within the subroutine in which they were defined**
- Multiple local variables with the same name can exist in different subroutines
- Are deleted once subroutine ends
- Using local variables ensures subroutines are **self-contained**



## Global Variables

- Can be **accessed across the whole program**
- Useful for values that need to be used by multiple parts of the program
- Danger of being **unintentionally edited**
- Not deleted until program terminates, so **require more memory**

## **Modularity, Functions and Procedures**

- Modular programming is a technique used to **split large, complex programs into smaller, self-contained modules**
- Easier to **divide tasks between a team** and manage projects
- Simplifies the process of testing and maintenance, as each component can be **dealt with individually**
- Improves **reusability** of components

## Top-down Design/ Stepwise Refinement

- Technique used to modularise programs
- Problem is **broken down into sub-problems**, until each is represented as an **individual, self-contained module which performs a certain task**
- Modules form blocks of code called **subroutines**

## Functions and Procedures

- Both **named blocks of code that perform a specific task**
- Procedures do not have to return a value
- Functions must **always return a single value**
- Parameters can be passed into a subroutine either **by value** or **by reference**

## Passing by Value

- A **copy of the value** is passed to the subroutine and discarded at the end
- Its value outside of the subroutine remains unaffected

## Passing by Reference

- **Address of parameter** is given to the subroutine
- Value of the parameter will be **updated at the given address**

In exam questions, you should assume parameters are passed by value unless you are told otherwise. The following format will be used:

```
function multiply(x:byVal, y:byRef)
```



## Use of an IDE (Integrated Development Environment)

- **Program** which provides a **set of tools** to make it easier for programmers to **write, develop and debug code**
- Features of IDEs include:
  - Stepping
  - Variable watch
  - Breakpoint
  - Source code editor
  - Debugging tools

## Use of object-oriented techniques

- Object-oriented languages are built around the idea of classes
- A **class** is a **template for an object**
- Classes define the **state and behaviour of an object**
- Object state is given by **attributes** while behaviour is defined by **methods**
- An **object** is a **particular instance of a class**
- **Attributes cannot be directly edited** in a technique called **encapsulation**
- Top-down design applies encapsulation to modules, which are also built to be self-contained and reusable

