OCR Computer Science AS Level

1.2.3 Introduction to Programming

Advanced Notes

Specification:

1.2.3 a)

- **Procedural programming language techniques:**
  - Program flow
  - Variables and constants
  - Procedures and functions
  - Arithmetic, Boolean and assignment operators
  - String handling
  - File handling

1.2.3 b)

- **Assembly language**
  - Following and writing simple LMC programs

Procedural programming language techniques

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

Program Flow

Structured programming is a popular subsection of procedural programming in which the program flow is given by three 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, ELSE IF and ELSE 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.

Procedural programming is suited to problems that can easily be expressed as a series of instructions using the constructs described above.

Variables and Constants

Variables are named locations in memory where data is stored. The contents of this location can be changed while the program is being executed. Variables must be explicitly declared in a statement before use. In some programming languages, variables must be assigned to a certain data type but otherwise the language assumes the data type of the value.

Variables are assigned using the = sign, as shown below:

```
name = Ellen
sides = 3
```

The = used here is called an assignment operator.

Constants are also named locations in memory, but the value of a constant cannot be edited by the program during execution. Constants are used for values that do not need to be changed, such as the value of pi or the current rate of VAT. Declaring a value as a

constant also prevents it from being accidentally changed. When writing code, constants are often capitalised, as shown below:

```
PI = 3.14159
VAT = 20
```

Procedures and Functions

Procedures and functions are both named blocks of code that perform a specific task. While procedures do not have to return a value, functions must always return a value. Procedures can return multiple values whereas a function must return one, single value. Procedures are typically given data as parameters for manipulation while functions commonly make use of local variables.

The subroutine below is an example of a function as it always returns a value of either True or False regardless of the input.

```
function isEven(number):
     if number MOD 2 = 0:
          return True
     else:
          return False
end function
```

Arithmetic, Boolean and assignment operators

A set of arithmetic operators are used to carry out mathematical functions within programs. These include +, -. * and /. There are several special symbols used to perform additional functions:

`**` is used for exponentiation which is when a number is raised to a power.
`2**4` gives 16.

`DIV or //` calculates the whole number of times a number goes into another. This is called integer division.
`50 DIV 7` gives 7.

`MOD or %` is used to find the remainder when a number is divided by another.
`50 MOD 7` gives 1.

Relational operators are used to make comparisons between two values and produce a result of either True or False. These include >, < and = alongside greater than and equal to and less than and equal to signs which are denoted as >= and <= respectively.

One additional operator is the 'not equal to' operator which is often used as part of conditional statements, as shown below:

```
if result != keyword:
    Print 'not found'
```

`==` is used to check whether one value is identical to another.

These can be combined with Boolean operators to check whether multiple conditions are met within a single statement. Boolean operators include AND, OR and NOT. The code below shows a statement which will return true if num2 is greater than num1 and is not an even number:

Synoptic Link

You will learn more about Boolean logic in 1.4.3.

```
if num2 > num1 AND num2 MOD 2 != 0:
    return True;
```

AND will return true only if all of the values being compared are true. OR will return true if at least one of the values forming the statement returns True.

String handling

There are various operations that can be performed on strings and that you need to be aware of.

Note

Don't forget strings are zero-indexed! This means the first character is considered to be at the 0th position.

To get the length of a string:

```
stringname.length
```

```
text="physics and maths tutor"
```

`text.length` will produce 23.

To get a substring (a section within a string):

```
stringname.subString(startingPosition, numberOfCharacters)
```

```
text="physics and maths tutor"
print text.substring(2,4)
```

will produce 'ysic'.

File handling

In addition to manipulating strings, you need to be able to use pseudocode to handle files.

To open a file to read:

```
myFile = openRead("filename.txt")
```

To read a line from a file:

```
fileContent = myFile.readLine()
```

To close a file:

```
myFile.close()
```

To open a file to write:

```
myFile = openWrite("nameoffile.txt")
```

To write a line to a file:

```
myFile.writeLine("Physics and Maths Tutor")
```

The end of the file is given by:

```
endOfFile()
```

Assembly Language

Assembly language is the next level up from machine code and is part of a family of low level languages. This is converted to machine code using an assembler when it is executed.

Assembly language uses mnemonics rather than binary, which makes it easier to use than direct machine code. Each mnemonic is represented by a numeric code. However, the commands used by assembly language are processor-specific as they directly interact with the CPU's special purpose registers. This allows for direct interaction with hardware so is useful in embedded systems. Typically, each instruction in assembly language is equivalent to almost 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
```