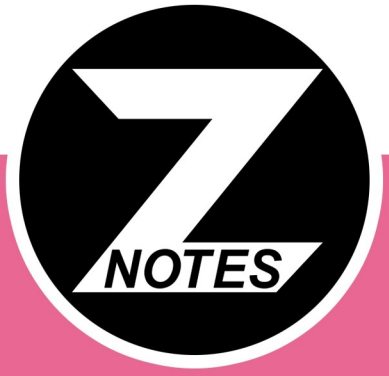
ZNOTES // A-LEVEL SERIES

visit www.znotes.org



Updated to 2017-19 Syllabus

CIE AS-LEVEL COMPUTER SCIENCE 9608

SUMMARIZED NOTES ON THE PRACTICAL SECTION

TABLE OF CONTENTS

1. Algorithm Design & Problem-Solving

Algorithm: a solution to a problem expressed as a sequence of steps

1.1 Identifier Table

- **Identifier:** name given to a variable in order to call it
- An identifier table depicts information about the variable, e.g.

Identifier	Data Type	Description
NumberDrawn	ARRAY[50] : BOOLEAN	FALSE – indicates the number has not yet been drawn TRUE – indicates the number has been drawn
Index	INTEGER	used as the subscript/index for the array

- **Rules for naming identifiers:**
  - Must be unique
  - Spaces must not be used
  - Must begin with a letter of the alphabet
  - Consist only of a mixture of letters and digits and the underscore character '_'
  - Must not be a 'reserved' word – e.g. Print, If, etc.

1.2 Basic Program Operations

- **Assignment:** an instruction in a program that places a value into a specified variable
- **Sequence:** programming statements are executed consequently, as they appear in the program
- **Selection:** control structure in which there is a test to decide if certain instructions are executed
  - **IF selection:** testing 2 possible outcomes
  - **CASE selection:** testing more than 2 outcomes
- **Repetition/Iteration:** control structure in which a group of statements is executed repeatedly
  - **FOR loop:** *unconditional*; executed a set no. of times
  - **WHILE loop:** *conditional*; executed based on condition at start of statements
  - **REPEAT loop:** *conditional*; executed based on condition at end of statements

1.3 Stepwise Refinement

- Process of developing a modular design by splitting a problem into smaller sub-tasks, which themselves are repeatedly split into even smaller sub-tasks until each is just one element of the final program.

1.4 Program Modules

- This refers to a modular program design
- **Subroutines:** self-contained section of code, performing a specific task; part of the main program
- **Procedures:** performs a specific task, no value returned to part of code where called
- **Functions:** performs a specific task, returns a value to part of code where called

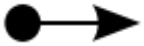
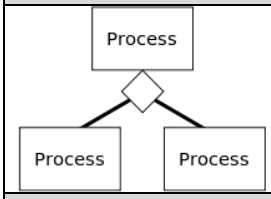
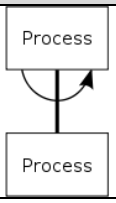
1.5 Logic Statements

Operator	Meaning	Operator	Meaning
<	Less than	>=	Greater/equal
<=	Less than/equal	==	Equal to
>	Greater than	!=	Not equal to

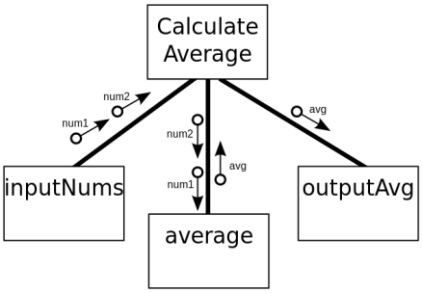
1.6 Structure Charts

- **Purpose:** used in structured programming to arrange program modules, each module represented by a box
- Tree structure visualizes relationships between modules, showing data transfer between modules using arrows.
- Example of a top-down design where a problem (program) is broken into its components.

Rules:

Symbol	Description
Process	
	Represents a programming module e.g. a calculation
Data Couple	
	Data being passed from module to module that needs to be processed
Flag	
	Check data sent to start or stop a process. E.g. check if data sent in the correct format
Selection	
	Condition will be checked and depending on the result, different modules will be executed.
Iteration	
	Implies that module is executed multiple times

Example:



1.7 Corrective Maintenance

- **White-Box testing:** making sample data and running it through a trace table
- **Trace table:** technique used to test algorithms; make sure that no logical errors occur e.g.

Line	largest	x	value[x]> largest	output
1	38	-		
2		2		
3			false	

1.8 Adaptive Maintenance

- Making amendments to:
  - **Parameters:** due to changes in specification
  - **Logic:** to enhance functionality or more faster or both
  - **Design:** to make it more user friendly

2 Data Representation

2.1 Data Types

Integer:

- Positive or negative number; no fractional part
- Held in pure binary for processing and storage
- Some languages differentiate short/long integers (more bytes used to store long integers)

Real:

- Number that contains a decimal point
- Referred to as singles and doubles depending upon number of bytes used to store

Boolean:

- Can store one of only two values; "True" or "False"
- Stored in 1 byte: True = 11111111, False = 00000000

String:

- Combination of alphanumeric characters enclosed in " "
- Each character stored in one byte using ASCII code
- Each character stored in two bytes using Unicode
- Max length of a string limited by available memory.
- Incorrect to store dates or numbers as strings
- Phone no. must be stored as string else initial 0 lost

Character:

- A character is any letter, number, punctuation or space
- Takes up a single unit of storage (usually a byte).

Dates:

- Dates are stored as a 'serial' number
- Equates to the number of seconds since January 1st, 1904 (thus they also contain the time)
- Usually take 8 bytes of storage
- Displayed as dd/mm/yyyy or mm/dd/yyyy

2.2 ASCII Code

- Uses 1 byte to store a character
- 7 bits available to store data and 8th bit is a check digit
- $2^7 = 128$, therefore 128 different values
- ASCII values can take many forms: numbers, letters (capitals and lower case are separate), punctuation, non-printing commands (enter, escape, F1)

2.3 Unicode

- ASCII allows few number of characters; good for English
- Unicode allows others too: Chinese, Greek, Arabic etc.
- Different types of Unicode:
  - UTF-8: compatible with ASCII, variable-width encoding can expand to 16, 24, 32, 40, 42
  - UTF-16: 16-bit, variable-width encoding can expand to 32 bits
  - UTF-32: 32 bit, fixed-width encoding, each character exactly 32 bits

2.4 Arrays

1-Dimensianal (1D) Array	2-Dimensional (2D) Array
declared using a single index, can be represented as a list	declared using two indices, can be represented as a table

Index	Element
[0]	JONES
[1]	ERICSON
[2]	WILLIAMS
[3]	ADAMS
[4]	JOHAL

Indexes	[0]	[1]	[2]
[0]	JONES	F	PENN'S
[1]	ERICSON	M	LAMBOURNE
[2]	WILLIAMS	M	PENN'S
[3]	ADAMS	F	CASWALL'S
[4]	JOHAL	M	SWALLOW'S

- **Pseudocode:**
  - 1-D Array: `A[1:n]`
  - 2-D Array: `A[1:m, 1:n]`

- **Python:**
  - Declaring an array: `a = []`
  - Adding to an array: `a.append(anything)`
  - Length of array i.e. number of elements: `len(a)`
  - Printing an element in a 1D array: print `(a[x])`
  - Printing element in a 2D array: `print(a[row][column])`
  - Printing row in a 2D array: `print(a[row])`
  - Printing column: use for loop and keep adding 1 to the row and keep column same

2.5 Bubble Sort

```
BubbleSort( int a[], int n)
Begin
     for i = 1 to n-1
          sorted = true
          for j = 0 to n-1-i
               if a[j] > a[j+1]
                    temp = a[j]
                    a[j] = a[j+1]
                    a[j+1] = temp
                    sorted = false
          end for
          if sorted
               break from i loop
     end for
End
```

- A FOR loop is set to stop the sort
- Setting a variable 'sorted' to be 'true' at the beginning
- Another FOR loop is set up next in order to search through the array
- An IF is used to see if the first number of the array is greater than the second. If true:
  - First number stored to variable
  - Second number assigned as first number
  - Stored variable assigned to second number
  - Set 'sorted' to 'false' causing loop to start again
- The second FOR loop is count based thus will stop after a specific number of times
- Goes through bigger FOR loop ∴ 'sorted' remains 'true'
- This exits the loop ∴ ending the program

2.6 Linear Search

```
For each item in the list:
     if that item has the desired value:
          stop the search and return the item's location.
Return the item is not in the list
```

- A FOR loop goes through the array
- It compares item in question to those in list using an IF:
  - If item matches with another then search is stopped
  - Also the location where it was found is returned
  - If not found it exits the FOR loop
- Then returns fact that item in question is not in the list

2.5 File Handling

- Files are needed to import contents (from a file) saved in secondary memory into the program, or to save the output of a program (in a file) into secondary memory, so that it is available for future use

Pseudocode:

- Opening a file:
  `OPENFILE <filename> FOR READ/WRITE/APPEND`
- Reading a file:
  `READFILE <filename>`
- Writing a line of text to the file:
  `WRITEFILE <filename>, <string>`
- Closing a file:
  `CLOSEFILE`
- Testing for end of the file:
  `EOF()`

Python:

- Opening a file
  `variable = open("filename", "mode")`
  Where the mode can be:

Mode	Description
r	Opens file for reading only. Pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites file if file exists or creates new file if it doesn't
a	Opens a file for appending. Pointer at end of file if it exists or creates a new file if not

- Reading a file:
  - Read all characters
    `variable.read()`
  - Read each line and store as list
    `variable.readlines()`
- Writing to a file:
  - Write a fixed a sequence of characters to file
    `variable.write("Text")`
  - Write a list of string to file
    `variable.write["line1", "line2", "line3"]`

3 Programming

- Programming is a transferable skill
- **Transferable skill:** skills developed in one situation which can be transferred to another situation.

3.1 Variables

- **Declaring a variable:**
  - Pseudocode: `DECLARE <identifier> : <data type>`
  - Python: no need to declare however must write above as a comment (`#...`)
- **Assigning variables:**

`<identifier> ← <value>` or `<expression>`

`identifier = value` or `expression` or `"string"`

3.2 Selections

'IF' Structure	

`IF <condition>`
`  THEN`
`  <statement(s)>`
`  ELSE`
`  <statement(s)>`
`ENDIF`
	`if expression:`
	`  statement(s)`
	`else:`
	`  statement(s)`

'CASE' Structure	

`CASE OF <identifier>`
`  <value 1>: <statement>`
`  <value 2>: <statement>`
`  ...`
`  OTHERWISE <statement>`
`ENDCASE`
	`if expression:`
	`  statement(s)`
	`elif expression:`
	`  statements(s)`
	`...`
	`else:`
	`  statement(s)`

3.3 Iterations

Count-controlled Loop	

`FOR <identifier> ← <val1> TO <val2> STEP <val3>`
`  <statement(s)>`
`ENDFOR`	

`for x in range(value1, value2):` `  statement(s)`	
Post condition Loop	

`REPEAT`
`  <statement(s)>`
`UNTIL <condition>`
	Not possible in Python
	Use `WHILE` and `IF`

Pre-condition Loop	

`WHILE <condition>`
`  <statement(s)>`
`ENDWHILE`
	`while expression:`
	`  statement(s)`

3.4 Built-in Functions

String/character manipulation:

- Uppercase or lowercase all characters

`("string").upper()` `("string").lower()`

- Finding length of a string

`len("string")`

- Converting:

String to Integer	Integer to String
`int("string")`	`str(integer)`

Random number generator:

`random.randint(a, b)`

where `a` and `b` defines the range

3.5 Benefits of Procedures and Functions:

- Lines of code can be re-used; don't have to be repeated
- Can be tested/improved independently of program
- Easy to share procedures/functions with other programs
- Create routines that can be called like built-in command

3.6 Procedure

Procedure: subroutine that performs a specific task without returning a value

- **Procedure without parameters:**

```
PROCEDURE <identifier>
  <statement(s)>
ENDPROCEDURE
```

```
def identifier():
  statement(s)
```

- When a procedure has a parameter, the function can either pass it by either reference or value
- **Pass by value:** data copied into procedure so variable not changed outside procedure

```
PROCEDURE <identifier> (BYVALUE <param>: <datatype>)
  <statement(s)>
ENDPROCEDURE
def identifier(param):
  statement(s)
```

- **Pass by reference:** link to variable provided so variable changed after going through procedure *(not in Python)*

```
PROCEDURE <identifier> (BYREF <param>: <datatype>)
  <statement(s)>
ENDPROCEDURE
```

- **Calling a procedure:**

`CALL <identifier>()` | `Identifier()`

3.7 Function

Function: subroutine that performs a specific task and returns a value

```
FUNCTION <identifier> (<parameter>: <data type>)
RETURNS <data type>
  <statement(s)>
ENDFUNCTION
def identifier(param):
  statement(s)
  return [expression]
```

4 Software Development

4.1 Program Development Cycle

- **Analyze problem:** define problem, record program specifications and recognize inputs, process, output & UI
- **Design program:** develop logic plan, write algorithm in e.g. pseudocode or flowchart and test solution
- **Code program:** translate algorithm into high level language with comments/remarks and produce user interface with executable processes

- **Test and debug program:** test program using test data, find and correct any errors and ensure results are correct
- **Formalize solution:** review program code, revise internal documentation and create end-user documentation
- **Maintain program:** provide education and support to end-user, correct any bugs and modify if user requests

4.2 Integrated Development Environment

- A software application that allows the creation of a program e.g. Python
- Consists of a source code editor, build automation tools, a debugger

Coding:

- Reserved words are used by it as command prompts
- Listed in the end-user documentation of IDE
- A series of files consisting of preprogrammed-subroutines may also be provided by the IDE

Initial Error Detection:

- The IDE executes the code & initial error detection carried out by compiler/interpreter doing the following:
  - Syntax/Logic Error: before program is run, an error message warns the user about this
  - Runtime Error: run of the program ends in an error

Debugging:

- Single stepping: traces through each line of code and steps into procedures. Allows you to view the effect of each statement on variables
- Breakpoints: set within code; program stops temporarily to check that it is operating correctly up to that point
- Variable dumps (report window): at specific parts of program, variable values shown for comparison

4.3 Types of Errors

Syntax errors:

- When source code does not obey rules of the language
- Compiler generates error messages
- Examples:
  - Misspell identifier when calling it
  - Missing punctuation – colon after if
  - Incorrectly using a built-in function
  - Argument being made does not match data type

Run-time errors:

- Source code compiles to machine code but fails upon execution *(red lines show up in Python)*
- When the program keeps running and you have to kill it manually
- Examples:
  - Division by 0
  - Infinite loop – will not produce error message, program will just not stop until forced to

Logic errors:

- Program works but gives incorrect output
- Examples:
  - Out By One – when '>' is used instead of '>='
  - Misuse of logic operators

4.4 Testing Strategies

Black box testing:

- Use test data for which results already calculated & compare result from program with expected results
- Testing only considers input and output and the code is viewed as being in a 'black box'

White box testing:

- Examine each line of code for correct logic and accuracy.
- May record value of variables after each line of code
- Every possible condition must be tested

Stub testing:

- Stubs are computer programs that act as temporary replacement for a called module and give the same output as the actual product or software.
- Important when code is not completed however must be tested so modules are replaced by stubs

Key:
Pseudocode
Python

CIE AS-LEVEL COMPUTER SCIENCE//9608