Quick revision Notes

Quick revision Notes

Computer science paper 2

The program development life cycle(PDLC):

- **Analysis:**
  - Analysing the problem to be solved by clearly defining it
  - Involves “Abstraction” where unnecessary details to the problem is discarded
  - Involves “Decomposition” to exactly determine the requirements of the program by breaking the problem into smaller pieces
- **Design:**
  - Shows how the program can be developed,By formally documenting it using structure diagrams,flowcharts,pseudocode
- **Coding(And iterative testing):**
  - The program or set of programs are developed,each module of the program is then tested and edited to work as intended before moving onto the next module.
- **Testing:**
  - The program is tested with different types of test data to make sure it is working as intended and solves the initial problem

Test data:

- **Normal test data:**
  - Data that the program should accept,used to check if program accepts and process expected data
- **Abnormal test data:**
  - Data that the program should reject,Used to check if the program rejects data that should not be accepted
- **Extreme test data:**
  - Data that the program should accept,Used to check if the program accepts the lowest and highest values that should be accepted
- **Boundary test data:**
  - Data that the program should accept and reject,Used to check if the program accepts the lowest and highest values that should be accepted and rejects their counterparts above and below the accepted range.

Validation checks:

- **Length checks: Checks for the length of a string/array**
- **Range check: Checks if number is within a specific range**
- **Type check: Checks the variable type of a variable**
- **Presence check: Checks if there is a value present in input**
- **Format check: Checks if the input follows a specific format**
- **Check digits: Checks if the value was entered incorrectly**

Verification checks:

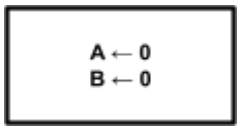
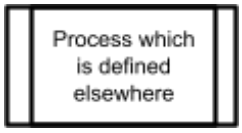
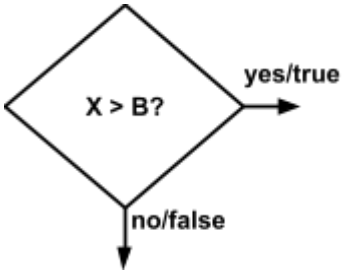
- **Double entry: Input is given 2 times to see if they match**
- **Screen/Visual check: A check by the user ensure the input is valid**

Decomposing a problem:

The component parts of any computer system are:

- **Inputs**
- **Processes**
- **Outputs**
- **Storage**

Flowcharts:

Symbol	Name	Function
	**Terminator** **Begin/End** **Start/Stop (Oval)**	Terminator symbols mark a flowchart's start and end.
	**Flow lines/** **(Arrows)**	Flowchart arrows show direction, usually top-down or left-right.
	**Process** **(Rectangle)**	This symbol shows process details. Functions: **Assignment** – Storing values in variables. **Calculations** – Performing arithmetic operations. **Updating variables** – Modifying values during execution. **Data processing** – Sorting, searching, or transforming data. **Function calls** – Executing predefined procedures.
	**Process Defined Elsewhere/** **Subprogram/** **Subroutine**	This symbol indicates a process defined elsewhere, labeled with its name.
	**Input/Output** **(Parallelogram)**	The same symbol represents both data input and information output.
	**Decision** **(Diamond)**	A decision is used to determine the next action in a process. It is commonly used for selection and iteration (repetition). In flowcharts, a decision symbol always has two output paths, which should be clearly labeled. You can have more than one condition by including AND or OR. Functions: **Conditional checks** (e.g., if age >= 18) **Branching** (Yes/No, True/False) **Loop control** (continue/exit) **Comparisons** (if score > 50) **Logical decisions** (AND/OR conditions)

Operator	Comparison	Example

>	**Greater than**	IF age > 18 THEN
		  OUTPUT "You are eligible to vote."
		ELSE
		  OUTPUT "You are not eligible to vote."
		ENDIF

<	**Less than**	IF temperature < 0 THEN
		  OUTPUT "It is freezing!"
		ENDIF

=	**Equal**	IF password = "admin123" THEN
		  OUTPUT "Access Granted."
		ELSE
		  OUTPUT "Access Denied."
		ENDIF

>=	**Greater than or equal**	IF marks >= 50 THEN
		  OUTPUT "You have passed the exam."
		ELSE
		  OUTPUT "You have failed the exam."
		ENDIF

<=	**Less than or equal**	IF speed <= 80 THEN
		  OUTPUT "You are within the speed limit."
		ELSE
		  OUTPUT "You are exceeding the speed limit!"
		ENDIF

<>	**Not equal**	IF username <> "admin" THEN
		  OUTPUT "You are not an admin user."
		ENDIF
		Explanation: If username is not "admin", it displays a message

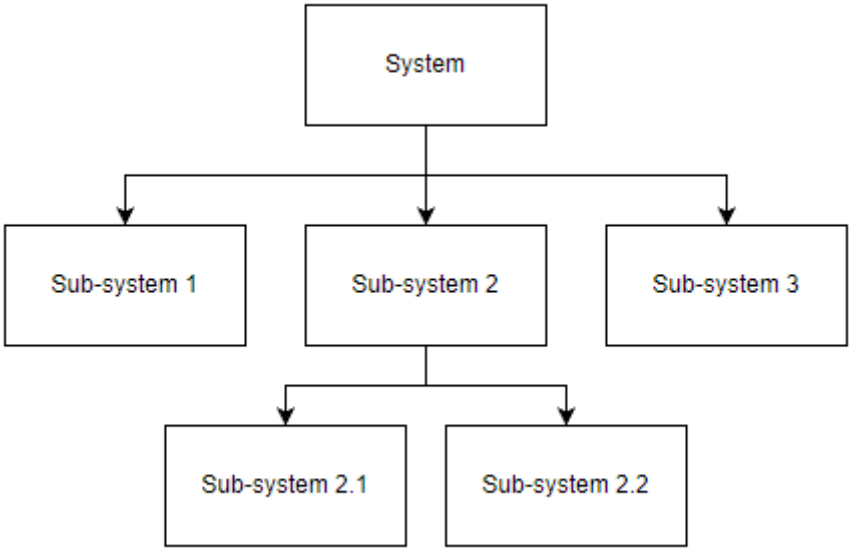
AND	**Both**	IF age >= 18 AND country = "UK" THEN
		  OUTPUT "You can apply for a driving license in the UK."
		ENDIF
		Explanation: The user must be 18 or older AND from the UK to apply for a license.

OR	**Neither**	IF weather = "rainy" OR weather = "snowy" THEN
		  OUTPUT "Take an umbrella."
		ENDIF
		Explanation: If the weather is either rainy or snowy, the program suggests taking an umbrella.

NOT	**Not**	IF NOT (isMember = TRUE) THEN
		  OUTPUT "You need to register first."
		ENDIF
		Explanation: If isMember is not true, it prompts the user to register.

==	**Comparison**	X==3 Is X equal to 3?

Structure diagram:



Pseudocode and Python code:

Code statement	Pseudocode	Python Code
Count control loop	FOR Count ← 1 TO 50 <Code> NEXT Count	for i in range(50): <Code>

Precondition loop	WHILE <Condition> DO
	  <Code>
	ENDWHILE
		while <Condition>:
		  <Code>

Post-condition loop	REPEAT
	  <Code>
	UNTIL <Condition>
		while (True):
		  <Code>
		  if <Condition> :
		  break

Conditional statement	IF <Condition>:
	  THEN
	  <Code>
	  ELIF <Condition>
	  <Code>
	  ELSE
	  <Code>
	ENDIF
		if <Condition>:
		  <Code>
		elif <Condition>:
		  <Code>
		else:
		  <Code>

Input statement	INPUT Number	Number=input(<Prompt>)
Output statement	OUTPUT "you said",Number	print("you said",Number)
Declaration / initialisation	DECLARE Num:INTEGER DECLARE List AS ARRAY[1:20] OF STRING	Num=0 List =["" for i in range(20)]
Counting	Count←Count+1	Count += 1
Totaling	Total←Total+Number	Total += Number

Declaration of procedure	**PROCEDURE <Name>(<Parametre n>:<Parameter n type>)** **<Code>** **ENDPROCEDURE**	**def <Name>(<Parameter n>):** **<Code>**
Declaration of function	**FUNCION <Name>(<Parametre n>:<Parameter n type>) RETURNS <return Type>** **<Code>** **RETURN <Data>** **ENDFUNCTION**	**def <Name>(<Parameter n>):** **<Code>** **return<data>**
Calling procedure/function	**Num ← CALL <Name>(<Parameters>)**	**Num = <Name>(<Parameters>)**
Case of a value	**CASE OF Num** **<Value 1> : <Code>** **<Value 2> : <Code>** **<Value n> : <Code>** **OTHERWISE <Code>** **ENDCASE**	**if <Condition>:** **<Code>** **elif <Condition>:** **<Code>** **else:** **<Code>**
Open/close file	**OPENFILE <File path> FOR <Access type>** **<Code>** **CLOSEFILE(<File path>)**	**Var= open(<File path>,<access type>):** **close(Var)**
Read file	**OPENFILE <File path> FOR READ** **READFILE <Variable to store>** **CLOSEFILE(<File path>)**	**Var= open(<File path>,"r"):** **File_contenets = Var.read()** **close(Var)**
Write file	**OPENFILE <File path> FOR WRITE** **WRITEFILE <Variable with writing data>** **CLOSEFILE(<File path>)**	**Var= open(<File path>,"w"):** **Var.write(<Data>)** **close(Var)**

Name	Sign/function	purpose
add	'+'	Add/concatenate 2 values
subtract	'-'	Subtract 2 values
Divide	'/'	Divide 2 numbers
integer Divide	DIV (or) '//'	Divide 2 numbers and return integer quotient
Modulus	MOD (or) '%'	Divide 2 numbers and return integer remainder
Length	Python: len(<var>) (or) Pseudocode: LENGTH(<var>)	Finds and returns the length of a string,array or dictionary
Uppercase	Python: <str>.upper() (or) Pseudocode: UCASE(<str>)	Returns a version of the string with all characters in upper case
Lowercase	Python: <str>.lower() (or) Pseudocode: LCASE(<str>)	Returns a version of the string with all characters in lower case
Substring	Python: <Str>[<Start char pos> : <End char pos>] (or) SUBSTRING(<str>,<start char pos>,<length>)	Returns a part of the string that is specified

Common definitions:

- **Constant**: Used to store values that do not change throughout the program;Ex: pi.
- **Variables**: Used to store values that do change throughout the program or in runtime;Ex: Looping variables.
- **Functions**: A group of programming statements under a defined name that can be called repeatedly through the program with the defined name,the function returns a value back to the program it was called in;Ex: a function to do integer division.
- **Procedures**: A group of programming statements under a defined name that can be called repeatedly through the program with the defined name,the procedure does not return a value back to the program it was called in.
- **Local variables**: A local variable is a variable that can only be used in the module of code it was defined in,it does not have a special syntax to initialize it.
- **Global variables**: A global variable is a variable that can be used anywhere in the code it was defined in,it does have a special syntax to initialise it.

Database data types:

Data type	Example	Description
Text	“Hello world” (OR) ‘Hello 1234’	A group or ‘string’ of characters
Character	“M” (OR) ‘4’	A single character
Boolean	TRUE (OR) FALSE	A value that can either be {true,1,yes} or {false,0,no}
Integer	9	A whole number
Real	20.35	A decimal value
Date/Time	“22/11/2024” (or) “22:43:10.33”	A date or time value

Records and fields:

Records are the rows

Fields are the columns

Create a table:

Format:	EXAMPLE:
CREATE TABLE <table name>**(** <Field Name> <Field Type> **PRIMARY KEY,** <Field Name> <Field Type> **,** <Field Name> <Field Type> **);**	**CREATE TABLE** Students **(** StudentID **TEXT NOT NULL PRIMARY KEY,** StudentName **TEXT NOT NULL,** StudentGrade **INTEGER NOT NULL,** StudentDOB **DATE NOT NULL,** StudentAvgMarks **REAL NOT NULL,** StudentGender **CHARACTER NOT NULL,** IsStudentTopper **BOOLEAN NOT NULL);**

Insert values into table:

INSERT INTO <table name> **VALUES (**<value1>**,**<value 2>**,**…<value n>**);**

Inserts a record of all values given for each field into the specified table name

Output from database:

SELECT <fields> **FROM** <table name> **WHERE** <condition> **ORDER BY** <field> **DESC** ;

[Text in blue]: Select the values of a field from a specified table, this part of the statement is necessary.

Ex: **SELECT** code, price **FROM** Items {returns fields codes and price from table items}

[Text in orange]: Selects the values from the field only if the condition is true, not necessary for the statement to work.

Ex: **SELECT** * **FROM** Items **WHERE** Price **>** 250 {returns the whole record of any item with a price over 250}

[Text in purple]: Returns the values in alphabetical order, if DESC is specified then the order is reversed, not necessary for the statement to work.

Ex: **SELECT** Item_code **FROM** Items **ORDER BY** Item_name {returns items codes by the alphabetical order of its name}

Extras:

SELECT SUM (<Field>**)** {Returns the sum of the values in the field, field should be INT or REAL}

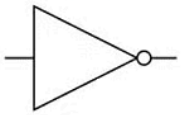
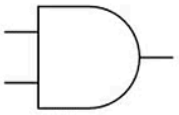
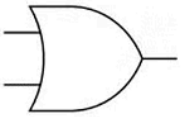
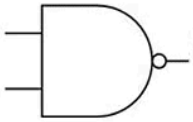
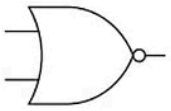
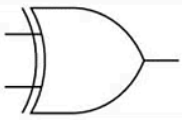
SELECT COUNT (<Field>**)** {Returns the number of records for that field}

Update value in database:

UPDATE <table name> **SET** <field> = <new value> **WHERE** <condition>**;**

Updates the values in the field of a specified table to the new value where the condition is true.
The condition is not necessary.

Logic gates:

Gate	Symbol	Description
NOT		The **NOT** gate takes a **single binary input** and outputs the **opposite** of the input
AND		The **AND** gate takes **two inputs** and produces one output Only **two positive inputs** (1 and 1) will result in a positive **output of 1** If either of the inputs is a 0 the output will be a 0
OR		The OR gate takes **two inputs** and produces one output If **either of the inputs is positive** (1) the **output will be 1**
NAND		A **NAND** gate is a combination of an **AND gate followed by a NOT gate.** If **both inputs are a 1** it will **output a 0.** Any other combination of inputs will result in an output of 1
NOR		A **NOR** gate is a combination of an **OR gate followed by a NOT gate.** If **both inputs are 0** it will output a **1.** Any other combination of inputs will result in an output of 0
XOR		An **XOR** gate (exclusive OR) will **output a** 1 if the **inputs** are **different to one another** (a 1 and a 0)

Python/Pseudocode methods to know:

Method:	**Code:**
1D/2D array handling	#1D array: Var = Array[<pos>] Array.append(<Value>) #2D array: Var = Array[<Row number>,<Column number>] Array[<Row number>].append(<Value>)

Bubble sort	for x in range(len(array)):
	  for i in range(len(array)-1):
	  if array[i] > array[i+1]:
	  array[i] , array[i+1] = array[i+1] , array[i]

Linear search	Var = <Value to find>
	for i in range(len(<Array>)):
	  if array[i]==Var:
	  <Position of Var> = i

Tips and tricks:

- 15 mark Q:
  - In the 15 mark Q add a comment addressing the bullet point before the code for that point is written(example: “#getting wood type from user using integer input”)
  - In the15 mark Q Always add comments before a programming method(e.g. Bubble sort or linear search) stating the method used and the purpose.
  - Allocate half a page for declaration of variables and initialization before moving onto the code in the 15 mark question
  - Add checkboxes next to the bullet points in the 15 mark question.Tick them off when your done with each bullet point
  - Underline the conditions specified in the 15 mark question and refer to them as you write the program

- Error identification and correction Q:
  - In the error identification question always check for syntax errors throughout the code before moving onto the logical errors(At Least 1 error is guaranteed)
  - It is very common to have at least one error due to switched up operators(e.g. “<” instead of “>” ,”←” instead of “=”)

- Pseudocode writing Q:
  - Always recheck the “AND” and “OR” operators in a conditional statement as they are easy to confuse