

0478 Syllabus Paper 2 Review Notes

created by **Hardy Wen**

Overview

- 前段时间做完了Paper1的Review Notes发现效果还不错，来过一下Paper2。23年的Paper2与前几年大改版，差了很多。新Syllabus二卷的形式跟一卷差不了太多，也是一问一答，具体考察如下：

Paper 2 – Algorithms, Programming and Logic

Written paper, 1 hour 45 minutes, 75 marks

This question paper consists of short-answer and structured questions set on Topics 7–10 of the subject content.

All questions are compulsory, and candidates answer on the question paper.

The questions require candidates to have practical programming experience.

Knowledge of programming language syntax is not examined; in all cases the logic is more important than the syntax.

- 因为Paper2的新颖性，所以刷往年的PP无法直观的感受真正Paper2的形式。这时候Specimen Papers就显得尤为重要了：
 - SpecimenA: [Specimen \(2023\) QP - Paper 2 CAIE Computer Science GCSE \(physicsandmathstutor.com\)](https://www.physicsandmathstutor.com/gcse/computer-science/specimen-papers/paper-2/)
 - SpecimenB: [Cambridge IGCSE 0478 Computer Science specimen paper 2B for examination from 2023 \(cambridgeinternational.org\)](https://www.cambridgeinternational.org/0478/specimen/paper-2/)
- 而Paper2会考察到许多编程方面的内容，例如在最后的会有一道15分的场景问题

Scenario question

The final question in Paper 2 is a 15-mark unseen scenario question.

Candidates will be required to write an algorithm using pseudocode or program code for the context provided.

It is expected that candidates should spend 30 minutes answering this question.

- 而编程方面最重要的是你的代码逻辑，对于语法方面的考察会较弱一点，所以如果考场上特别的语法记不清了可以不用太担心（这一点在第一章配图中有提到）。
- CIE要求的伪代码的语法规范、不同的Logic gate, Flowchart的图案可以在Syllabus中找到[595424-2023-2025-syllabus.pdf \(cambridgeinternational.org\)](https://www.cambridgeinternational.org/0478/specimen/paper-2/)
- 由于Paper2考察实践的内容较多，本份文件中不会涉及到具体的语法（如Python要怎么写，Logic gates要怎么画，具体的算法要怎么用Python来实现，Flow Chart要怎么画）。文件中会提到你需要掌握的技能，而具体的语法可以到网上学习（如[Python3 教程 | 菜鸟教程 \(runoob.com\)](https://www.runoob.com/python3-tutorial/)，看Syllabus，或者看我之前写的资料（朋友圈有发合集）。
- 本份文件会着重讲到一些理论性的Programming concepts（也就是Paper2要背的部分），祝大家一起加油！

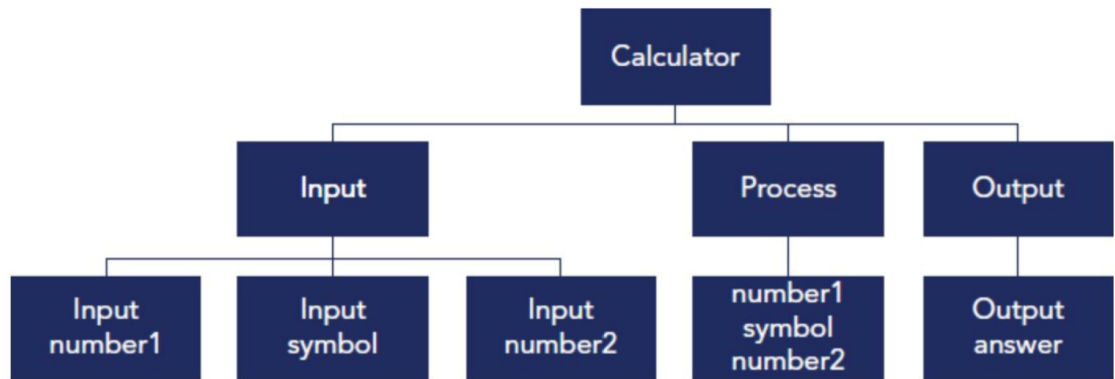
7. Algorithm Design and Problem-Solving

Development Lifecycle

- What is meant by **the program development life cycle**?
 - a series of structured steps that are followed to produce a system
 - It **includes** (you need to **identify** each stage and the tasks associated with each stage)
- 1. **Analysis**
 - abstraction: looking at a problem **in general, intuitively**, instead of looking at specific terms
 - decomposition of the problem: the **breakdown** of problems into subproblems to be tackled one by one
 - identification of the problem and requirements: **identifying** what is the problem and what is required
- 2. **Design**
 - decomposition: the **breakdown** of systems into subsystems to be developed one by one
 - structure diagrams, flowcharts, and pseudocodes: to develop general ideas about how to code the system
 - need to know how to draw the diagrams and write in pseudocode
- 3. **Coding**
 - write program code: developing the system
 - iterative testing: testing throughout the development of the program, often modules by modules
- 4. **Testing**
 - testing program code with the use of test data: testing the whole program to see whether it should receive data within the allowed range and reject data out of the range

Decomposition

- How are computer systems **made up**?
 - **every computer system is made up of sub-systems**, which are made up of further **sub-systems**
 - How can a system be **decomposed**?
 - a system can be decomposed into **inputs, processes, outputs, storage**
 - to visualize the decomposition, use **structure diagrams**
 - a **tree-like** diagram that contains **input, process, and output**
1. you identify what are the inputs and outputs of the system
 2. identify what calculations are needed to draw the diagram
 3. Draw the diagram



Algorithms

- How to **evaluate the purpose** of a given algorithm?
 1. stating what the algorithm is **inputting and outputting**
 2. describing the **processes involved**
- What are the common algorithms? (具体algorithm的示例在书本7.7小节，截图放在这里有点占位置不好排版:())
 - Linear search: 遍历每一个列表里的元素 看那个元素是不是要找的值 (做对比)
 - Bubble sort: 反复遍历列表让如果下一位比当前位的数大/小 就交换位置 要用两层循环
 - Totalling: 初始化一个变量total = 0, 遍历列表里的每一个元素并把把值加到total
 - Counting: 初始化一个变量count = 0, 遍历列表里的每一个元素, 如果元素符合判断条件则count = count + 1
 - Finding Maximum: 初始化一个极小值maxnum = -9999999, 遍历列表里的每一个元素, 如果元素大于maxnum则maxnum等于那个元素, 这样不停迭代更新
 - Finding Minimum: 初始化一个极大值minnum = 9999999, 遍历列表里的每一个元素, 如果元素小于minnum则minnum等于那个元素, 这样不停迭代更新

Validation and Verification

- Why do we need **validation checks**?
 - to test if the data entered is **possible / reasonable / sensible** and **within set bounds**, done by the program
 - 程序判断说用户输入的数据是不是合法的, 满不满足程序的要求
 - **Types of validation checks**

Types	Descriptions
Range Check	Makes sure that the value of the data is between a specific boundary
Length Check	Makes sure that the number of characters is within a set digit
Type Check	Makes sure that the data is from a correct data type
Presence Check	Makes sure that the data has been entered
Format Check	Makes sure that the data meets a specific order/format
Check Digit	Makes sure that the data entered is correct by comparing the check digit value

- Why do we need **verification checks**?
 - to test if the data input is **the same** as the data **that was intended to be input**, done by the user
 - 用户自己判断说自己输入的数据是不是正确的，譬如说自己输入的用户名是不是对的
 - **Types of verification checks**
 1. **visual check**: comparing the data entered with the original side by side 就是用眼睛做对比 😊
 2. **double entry check**: when the same data is entered twice, and the computer will check whether there are any differences in these two times (譬如设置密码的时候就会要求输入两次密码)

Test data

- What are the different types of **test data**?
 - **Normal data**: data that the program should accept
 - **Abnormal data**: data that the program should not accept
 - **Extreme data**: the largest/smallest acceptable value
 - **Boundary data**: the largest/smallest acceptable value and the corresponding smallest/largest rejected value
 - therefore, extreme data is included in boundary data

8. Programming

Programming concepts

- What are some **basic data types**?
 - integer, real (**float** in Python), char, string, Boolean
- What are some **basic programming concepts**?
 - it is important to understand the concepts below, but not important to memorize their definitions
 - **Variable**: a quantity that **can be altered** throughout the execution of the program

- used to store **single items** of data that can be accessed **through the identifier** (variable name)
- **Constant:** a quantity that **cannot be altered** throughout the execution of the program
 - used to store **single items** of data that can be accessed **through the identifier** (constant name)
- **Sequence:** programs carry out instructions in a **sequence**; instructions are executed in order, one after another
 - order of steps in a task
- **Selection:** choosing a path through a program
 - `if` and `case` statements
- **Iteration:** repetition of a sequence of steps in a program
 - **count-controlled loops:** `FOR...`
 - **pre-condition loops:** `WHILE...`
 - **post-condition loops:** `REPEAT... UNTIL...`
- **Totalling:** the process of keeping a running total of values in a program
- **Counting:** the process of finding the number of values in a container, such as an array
- **String handling:** the manipulation of string in a program
 - **length:** `LENGTH(text)` in pseudocode, `len(text)` in Python
 - **substring:** `SUBSTRING(text, start, number of characters)` in pseudocode, `text[start:end]` in Python
 - the first character of the string can be zero or one
 - **upper:** `UCASE(text)` in pseudocode, `text.upper()` in Python
 - **lower:** `LCASE(text)` in pseudocode, `text.lower()` in Python
- What are **operators**?
 - **Arithmetic:** for arithmetic calculations
 - `+`, `-`, `*`, `/`, `^` (`**` in Python for powering), `MOD()` (`%` in Python), `DIV()` (`//` in Python)
 - **Logical:** for logical operations
 - `=`, `<`, `<=`, `>`, `>=`, `<>` (`!=` in Python)
 - **Boolean:** for boolean operations
 - `AND`, `OR`, `NOT` (use lower-case letters in Python)
- What are **nested statements**?
 - a construct (selection or iteration) that is **inside** another construct
 - for example, to iterate through the 2D array,

```

1  for row in range(num_rows):
2      for column in range(num_columns):
3          array[row][column] = 0

```

- What do we need to know about **subroutines**?

- **Procedure**: a subroutine that **does not** return a value to the program that called it
- **Function**: a subroutine that **does return** a value to the program that called it
- **Parameter**: a value that is **sent to a subroutine**
- **Local variables**: the variable that can only be accessed in the subroutine it is declared within
- **Global variables**: the variable that can be accessed from any part of the program
- **Library routines**: pre-written subroutines that can be called within a program
 - MOD, DIV, ROUND, RANDOM
 - `RANDOM()` in pseudocode **returns a random number between 0 and 1 inclusive**
 - so, to return a whole number between -1 and 1,

```
1 | randomNumber <- ROUND( RANDOM() * 2, 0) - 1
```

- To get a random value in Python,

```
1 | import random # import the library
2 | randomNumber = random.randint(1,100) # call the library routine, a
   | random number between 1 and 100 inclusive
```

Maintainable Program

- How to make a program **maintainable**?

- Use **meaningful identifiers**
 - for **variables, constants, arrays, procedures and functions**
 1. easier for others to understand
 2. easier for you to maintain the program later
 3. when you've written a long program you don't need to remember the messy names and the corresponding value that they store, you can just remember the natural language which describes the data as the identifier is meaningful
- Use **comments** (`//` in pseudocode, `#` in Python)
 1. helps others to understand the code, as comments provide explanatory information about the source code
 2. helps you to understand the code later
 3. comments can also be used to note some bugs or ideas that you can look back and fix
- Use **functions or procedures**
 1. avoid repetition of commands, so the code will become easier to read
 2. save programming time

Arrays

- What are the **uses of arrays**?
 - to store a large collection of values
 - to reduce the need of declaring multiple variables with various identifier, and therefore reduce codes
- What are the **features of arrays**?
 - can only store items **of a single datatype**
 - stores items **in contiguous memory locations**
 - if it is a static array, the size of the array is initialized at the start and **cannot be resized** throughout running the program

File handling

- Why do we need **to store data in a file**?
 - data is not lost when the computer is switched off // data is stored permanently
 - data can be used by more than one program or reused when a program is run again
 - data can be backed up or archived
 - data can be transported from one place / system to another

Databases

- What are the **datatypes of SQL**?
 - text/alphanumeric, character, Boolean, integer, real, data/time
- Why do we need a **primary key**?
 - a **unique identifier** for a specific record in the data table

Boolean Logic

- What are the logic gates?
 - NOT, AND, OR, NAND, NOR, XOR