OCR A Level Computer Science

Your notes

6.3 Thinking Procedurally

Contents

Components of a Problem in Computational Thinking


Your notes

Identifying the Components of a Problem

- In order to create **algorithms** to solve a problem, the important parts of the problem must be identified by **abstracting** away unimportant details and then **decomposing** the problem into smaller **sub-problems**

How do you Identify the Components of a Problem?

- **Abstraction** is the act of **removing unimportant details** from a real problem, such as an object or environment and focusing on the details that will form part of the solution (component).
- **For example**, modelling a garden or football field in a program does not require modelling every single blade of grass. Usually a green mesh or image forms the basis for this representation. Higher **fidelity** simulations such as modern games go a long way to modelling details such as this at the cost of higher processing power however
- Other examples of **abstraction** are:
  - **Saving files**: users do not need to know the underlying complex procedures used to save a file on a hard disk drive or how the operating system represents data using paging and segmentation in RAM
  - **Sending an email**: users do not need to know which protocols are being used, how the data is formatted and what protocol handshaking occurs to send an email
  - **Downloading content**: Similar to sending an email (in reverse), a user does not need to explicitly know how security checks are made using digital certificates, signatures, protocols, etc in order to download content
- Consider a teacher calculating the grades of several classes of students over a year. An example of **identifying components of this problem** would be:
  - How many classes are there?
  - How many students are there?
  - What are the possible grades a student could achieve?
  - How many assessments are there to consider?
- If the teacher were to perform statistical analysis of the grades, **other components** could include:
  - Are the assessments divided into difficulty categories such as easy, medium, hard?
  - Do the number of questions on each assessment matter? Would this influence the grades in any way?

- Do the topics of each assessment matter? Is one topic harder than another? Would this influence the grades?
- Does the format of the questions matter? Do students do better on multichoice questions over written questions?


Your notes

Identifying the Components of a Solution

- Once the **components of the problem** are identified, the matching c**omponents used to solve the problem** can be identified
- In the above example of a teacher calculating the grades of students over the course of year, several key areas have been identified, including the **quantity of classes**, **students**, **assessments** and **grades**
- Thinking **procedurally** involves **procedural abstraction**, which means using a **procedure** to execute sequences of instructions to achieve some goal. In order to solve this problem, a **procedure** must be created.

Examiner Tips and Tricks

It is important to note that "**procedure**" in this context doesn't strictly mean a subroutine that doesn't return a value. A procedure in this context can be viewed simply as a **unit of computation that performs a single task**. It is work noting that a "**function**" is a subroutine that returns a value whereas a procedure does not

- The premise of the above grade calculation problem is simple but requires careful thought on the **technical details** of the solution, namely:
  - **What are the input parameter and output data types?** Integer, real/float, string, boolean, etc?
  - **What data structures may be required?** Array, list, dictionary, linked list, hash table?
  - **Will any libraries be required?** Statistics module, math module?
  - **What format will the output take?** Array, list, dictionary, output string etc?
  - **What data transformations and constructs will be required?** Multiplication, division, for loop, while loop, select case?
  - **Is the efficiency of the algorithm an important factor?** Does the algorithm have to be O(n) or less?
- Once the initial solution components have been identified and at the very least been considered, the **procedure** can then be **decomposed** into more detailed instructions. More complex problems may require larger programs consisting of many **subroutines** that call each other or imported **library subroutines**

Order of Steps in Problem Solving


Your notes

Determining the Order of Steps

- In order to begin creating **subroutines** to **solve a problem**, the overall **order of steps** to carry out must be identified. This is done using **decomposition**

What is Decomposition?

- **Decomposition** is the act of taking a problem and **breaking it down into smaller**, **more understandable** and **more easily solved** sub-problems
- **Top-down design** is one method of **decomposing a problem**

Top-down design

- Top-down design involves **breaking down a problem into major tasks** and **breaking down these major tasks into smaller sub-tasks**. Each sub-task is broken down until each can be solved using a single subroutine or module. Furthermore, each sub-task should be **unable to broken down further**, be easily solved and be clear
- The goal of top-down design is to **structure a program** into small **manageable tasks**. Sometimes these tasks may be delegated to other developers and may need to be coordinated on as a team. It is therefore important that everyone understands the problem and how to solve it
- The **advantages of decomposing a problem into sub-problems** is that each subroutine is much **simpler to test** and **maintain**, especially using **unit testing**. As each subroutine is **self-contained** and **well documented** it is easier to **find errors in code** and **fixing problems** as they arise. It is also **convenient** to **reuse** subroutines as necessary rather than **rewriting** code
- Calculating the student grades for a teachers classes should be done in the following order:
  - Step 1 - Calculate the grade for each assessment
    - For each question, **mark the question** and **store the value**
    - **Sum the value** of each marked question
  - Step 2 - For each student calculate the average grade across all of their assessments
    - **Add up the grade** for each assessment
    - **Divide by the number of assessments**
    - **Store** the result
  - Step 3 - **Repeat** steps 1–2 for each class

Sub-Procedures

Your notes

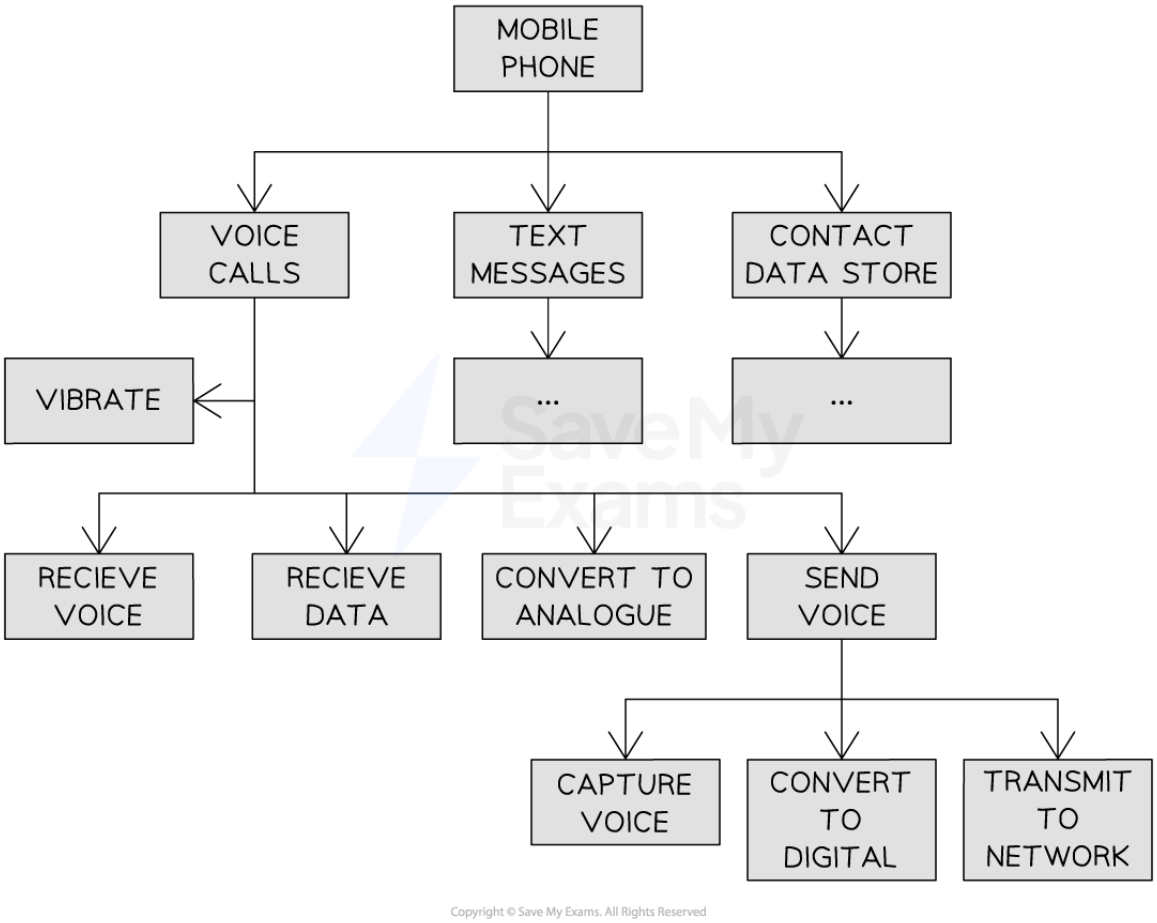
Identify Sub-Procedures Necessary to Solve a Problem

- Problems can be **decomposed** using tables and lists, however larger problems can quickly become cumbersome using this approach so **diagrams are preferred**

What is a Hierarchy chart?

- A **hierarchy chart** is an example of a diagram used to **show problem decomposition**
- Each problem is **divided into multiple subproblems**, which is turn are divided into **further subproblems** until they cannot be divided any further
- **Hierarchy charts** show how module and subroutines **relate** to each other and is **depicted as a tree structure**
- **Hierarchy charts** can be created on **paper** or **digitally** and can also be created **programmatically by software**
- An example of a hierarchy chart is shown below for a sending messages and calls on a mobile phone. Each sub-problem is broken down into smaller sub-problems:

Your notes



Worked Example

Mabel is a software engineer. She is writing a computer game for a client. In the game the main character has to avoid their enemies. This becomes more difficult as the levels of the game increase.

The computer game allows a user to select a character (e.g. name, gender). They can then choose a level for the game (easy, normal, challenging). The user controls their character by moving it left or right. The character can jump using space bar as an input. If the character touches one of the enemies then it loses a life. The character has to make it to the end of the level without losing all their lives.

- **The game is designed in a modular way.**

- **One sub-procedure will handle the user input.**

Describe three other sub-procedures Mabel could create for the given game description.

6 marks

Answer:

Mabel could create a sub-procedure to select a character, including name and gender, etc **[1]**, which gives the user options when choosing a character **[1]**

Mabel could create a sub-procedure to choose a level **[1]** which gives the user a choice of difficulty (easy, medium, challenging) by taking the user input **[1]**

Mabel could create a sub-procedure to allow the character to lose lives **[1]**. If the character has less than 0 lives left then the game ends **[1]**

Examiner Tips and Tricks

Basic input procedures such as moving left and right are not credit worthy but showing the output of a left/right move would be

Your notes