OCR Computer Science A Level

2.1.2 Thinking Ahead

Concise Notes
Specification:

2.1.2 a)  
● Identify the inputs and outputs for a given situation.

2.1.2 b)  
● Determine the preconditions for devising a solution to a problem.

2.1.2 c)  
● The nature, benefits and drawbacks of caching

2.1.2 d)  
● The need for reusable program components
Inputs and Outputs

- Designing a solution requires thinking ahead about how the different components of a problem can be handled in the best possible way.
- By thinking ahead, developers can build programs that are easy and intuitive to use.
- All computational problems consist of inputs which are processed to produce an output.
  - Inputs include any data required to solve the problem.
    - These are entered into the system by the user.
  - Outputs are the results that are passed back.
    - Outputs are produced once inputs have been processed.
    - Outputs are essentially the solution to the problem
- You should be able to evaluate the methods using which this data is captured, or relayed back to the user once processed.
  - Consider data structures and data types involved.
  - Consider input and output devices.
- Designers begin by considering the outputs based on the user’s requirements.
- This is used to identify the inputs required and how these need to be processed to achieve these outputs.

Preconditions

- Requirements which must be met before a program can be executed.
  - Can be tested for within the code or included in the documentation accompanying a particular subroutine, library or program.
- Specifying preconditions means that a subroutine expects the arguments passed to it to meet certain criteria.
- Including preconditions within documentation reduces the length and complexity of the program and saves time spent on debugging and maintenance.
- Preconditions make subroutines more reusable.

Reusable Program Components

- Commonly used functions can be packaged into libraries for reuse.
- Teams might create a library of components so they can be reused throughout a project. Reusable components include:
  - Abstract data structures eg. queues and stacks
  - Classes
  - Subroutines eg. functions and procedures
- Problem decomposition is used to identify where previously-developed program components can be reused.
- Reusable components are more reliable than newly-coded components, as they have already been tested.
- They save time, money and resources.
- Components may need to be modified to be compatible with existing software.
- This can be more costly and time-consuming than developing them from scratch.

### Caching

<table>
<thead>
<tr>
<th>A Level only</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Storing instructions or values in cache memory</strong> after they have been used, as they may be used again.</td>
</tr>
<tr>
<td><strong>Saves time</strong> of retrieving instructions from secondary storage again.</td>
</tr>
<tr>
<td>Frequently-accessed web pages are cached so content can be quickly loaded</td>
</tr>
<tr>
<td>This frees up bandwidth for other tasks on a network.</td>
</tr>
<tr>
<td><strong>Prefetching</strong> is when algorithms predict which instructions are likely to soon be fetched and are loaded and stored in cache.</td>
</tr>
<tr>
<td>Thinking ahead means less time is spent waiting for instructions to be fetched.</td>
</tr>
<tr>
<td>Limited by <strong>accuracy of algorithms used</strong>, as data stored in cache is not always used.</td>
</tr>
<tr>
<td>Effectiveness depends on caching algorithm’s ability to manage the cache:</td>
</tr>
<tr>
<td>Larger caches take a long time to search, but smaller <strong>cache sizes limit how much data can be stored.</strong></td>
</tr>
<tr>
<td>Can be <strong>difficult to implement</strong> well.</td>
</tr>
</tbody>
</table>