APS145 Applied Problem Solving & Computational Thinking

Computational Thinking

Programmers are different...

- Programmers must <u>think differently</u> in order to produce software and design systems – not everyone can make this mindset adjustment!
- Programmers must harness the <u>computational thinking</u> approach to problem solving
- Computational thinking ultimately paves the path to providing a <u>solution</u> that both, a <u>computer</u>, and a <u>human</u> can understand

Problem Solving - Understand the Problem!

- Before doing anything, you must <u>UNDERSTAND</u> the problem
- <u>Carefully read</u> the documentation that states the problem (several times if need be)
- Identify any uncertainties or questions you might have
 - Never assume anything get answers to things you are unclear about
- Do not proceed to create a "solution" to something you don't 100% fully understand

There are 4 main parts to the "Computational Thinking" approach to problem solving:

- 1. Decomposition
- 2. Pattern recognition & data representation
- Abstraction
- 4. Algorithms

1. Decomposition

- 2. Pattern recognition & data representation
- 3. Abstraction
- 4. Algorithms

 The process of breaking down complex problems into more manageable parts

 Even "simple" problems are more involved than you initially perceive

• **Example**: Describe your normal/typical morning?

Some example highlights

- Wake-up
- Shower
- Brush teeth
- Get dressed
- Eat breakfast
- Go to work/school

- There is much more going on in each of those points
- Take "Brush Teeth" as an example.
- This can be broken down into further considerations:
 - What tooth brush should be used?
 - What toothpaste should be used?
 - How long should you brush for?
 - How hard do you brush?
 - What part of your mouth do you start at?
 - Etc....

- This stage of computational thinking is vital for determining and identifying the scope of your solution
- This stage also significantly helps determine what you need to focus on to building a solution (also helps answer the question "Where do I begin?")
- This stage deserves a lot of focus, time, and attention since it defines the boundaries of your solution

- 1. Decomposition
- 2. Pattern recognition & data representation
- 3. Abstraction
- 4. Algorithms

- The pattern recognition and data representation stage is about identifying commonality in process and data.
- Reviewing the decomposed smaller problems will very likely reveal patterns and common data structures (attributes)
- Identifying patterns is important as it:
 - Simplifies the solution and makes it more efficient
 - Lends itself to reusability of logic for similar tasks
 - Reduces repetition and redundancy
- **Example**: Describe the task of baking a cake.

Some example highlights

- What kind of cake?
- What ingredients are needed?
- How much of each ingredient?
- When should the ingredient be added?
- How many people is this for?
- How long should it be baked?
- What temperature should the oven be set to?

- Consider the preceding parts when baking DIFFERENT types of cakes – Can you identify things that are common?
 - Ingredients are required in a specific amount...
 - Ingredients are added at specific times/sequence...
 - Each cake will be baked for a specific duration...
 - Each cake will be baked at a specific temperature...
 - Each cake will serve a specific number of people...
- Further review of this broken-down pattern can reveal more patterns. In this case a data structure for an "ingredient" could be defined as having:
 - Description/name
 - Amount/quantity

- This stage also deserves a lot of review and consideration as it will reveal and identify common behaviour and attributes
- When/if this can be done, you will simplify the solution by making parts reusable that can be applied to other similar problems (eliminating redundancies)
 - Ex: Reusable behaviour (a process) or data representation (attributes)

- 1. Decomposition
- 2. Pattern recognition & data representation
- 3. Abstraction
- 4. Algorithms

3. Abstraction

 This stage concentrates on generalizing the identified characteristics of a problem and/or pattern

Abstraction is about filtering out specific details to form an idea or concept

• In most cases this addresses specific data "values" which are inconsequential to the concept itself

3. Abstraction

Consider the cake example....

- It doesn't matter if the cake needs to be baked for 50 minutes or 90 minutes...
 - The <u>amount</u> or <u>data value</u> of how long it needs to be bake **doesn't change** <u>what</u> needs to be done
- What's important is that it needs baking for some defined period of time

3. Abstraction

- Abstraction declutters characteristics of a problem and solution of unnecessary details
- Abstraction leads to creating a generalized model and ensures the solution addresses the problem
 - If filtering of specific details is not done, you can inadvertently produce a solution that doesn't address the problem (ex: All cakes must be baked at 190 °C)

- 1. Decomposition
- 2. Pattern recognition & data representation
- 3. Abstraction
- 4. Algorithms

- The last stage involves algorithms representing a master plan
- The plan is a set of very specific step-by-step instructions that can be carried out to solve a problem
- The sequencing of the instructions is in the order in which they must occur
- There is always a defined starting point and ending point
- Algorithms can refer to other algorithms when necessary for larger more complicated problems

- It is <u>VITAL</u> to incorporate <u>UNIT-TESTING</u> during the assembly of the algorithm(s).
- Testing will reveal incomplete or missing logic
- When incomplete or missing logic is identified, you must revisit the other stages to review and reassess the effect it may have on other aspects of the defined problem and working solution
- The last step is to perform an <u>OVERALL TEST</u> of the entire solution (algorithm/s) to ensure the sequencing is correct and the problem is in fact solved

Algorithms are typically represented in two ways:

- Pseudo code
- Flowchart

Pseudo Code

- Best used for computer programmers
- Provides the framework of the program without the specific syntax of the language
- Significantly reduces time to code the logic when a detailed set of instructions is already prepared

NOTE: In IPC, it is expected you produce this before coding!

Flowchart

- Best used to communicate process to non-technical business persons and peers
- Labelling should use simple and concise language to reach a broader audience
- Often used to explode a very specific small section of logic to help visualize a task