SD400: Problem Solving

Lesson 3: Problem Decomposition

Wholeness

- Our focus today is to fine-tune our skills. The exam showed that there were some issues with code reading and taking problems appart, solving the pieces and putting them back together.
- This is similar to the principle that purification leads to progress.

Lesson Objectives

- Breaking a Problem into Parts
- Determine required data
- Determine required processes
- Defining tables
- Integrating the algorithm component
- Implementing all above

Input and Output

```
let input = prompt(“enter name”)
// algorithm …
console.log(name)
```

Exercise

- What is the output:

```
console.log(“Hello! ”);
console.log(“This is a test \n”);
console.log(10 + “5”);
Console.log(2 + 2);
```

Operators

The “+” with a string on one side and a number on another

- Turns the number into a string and concatenates

- Any operation with a string and a number
  - Will turn the string into a number!

Breaking a Problem into Parts

- When problems we've been doing don't get solved, or solved incorrectly it's often because the parts have gotten mixed
- Each part can be solved individually, and then combined relatively easily (little or no interaction with the other parts) to create a total solution.

Example

Write a program that asks how many coin flips you want, and then uses a loop to ‘flip’ (create heads / tails) that amount of times.

There are 3 main parts here:

- Input
- A loop with a counter (repeat)
- ‘a coin flip’

Step 1 – Determine required data

- Before thinking of building algorithm, it's always a great practice to determine the required data.
- It is important since it will give you an insight of what declarations you have to identify – (defining the ingredients)

- Even if you didn't get all required data, it will still be very helpful. Some cases, you will realize that you need to add another variable or more.

Step 2 – Determine the required processes

- Think of all the operations to process in order to achieve the required output
- This is where you will mostly determine the data that is related to the task such as (maxWeight, milesAllowed, taxRate, etc.)

- A good approach for early phases of coding for Step 1 and 2 is to use a **Defining Table**

Defining Table

- A **defining table** is a useful tool to help you better understand a problem before you develop an algorithm to solve it.
- A defining table has three sections: input, processing, and output.
- To create a defining table, simply draw a table with the three sections. Then as you read and re-read the problem, put the parts of the problem into their correct section in the table.

Example

- You work for a large construction company. Your boss has asked you to write a computer program that will read a list of window openings for a building and compute, and output the total cost of all the windows. The window openings are entered in inches with the width first and the height second. The cost of a window is computed by multiplying the area of the window in square feet by $35.

Defining Table

Input	Processing	Output
A list of window openings For each window • width in inches • height in inches	For each window • compute area in sq. ft. • multiply area by $35 • add cost of this window to the total cost	total cost of all windows

Step 3 – Integrate the Algorithm Components

- After the completion of Step 1 & 2 creating a defining table. You can build the algorithm – synthesis phase.
- This is where you form your flowchart and convert it to code.

- All of these phases will be implemented spontaneously when you get the grip of it.

Converting it to code

- A good practice to place the processes in comments, then implementing them with code.

- Following these steps will help you keep track of the code and convert a flowchart smoothly. Even if the flowchart is a thought in your mind.

Main Point 1

- The key to solving a big / complex problem is identifying the smaller / simpler parts that it is made of, solving them and joining them together.
- Neither the parts or the joining should be complex.
- Complexity indicates trying multiple things in one. Identifying them, solving them separately, and then joining them will be the solution.

Implementation

- The following slides will be examples to understand how to break down a problem and build an algorithm.

- You are not required now to know all the required knowledge for the following tasks.

- You shou cover all of it by the end of this course.

Example

- Write a program that will ask the user for the radius of a circle and calculate the area

Example

- Write a program that will ask the user to enter the first and last name and print out the initials

Example

- Write a program that will ask the user to input a number ***n***, and flip a coin ***n*** times and count how many heads occured.

Example

- Write a program that ask the user to enter a ***target*** to search in a list of names. The output should be the number of occurrences of that target.

Main Point 2

- Each flowchart element has a 1 to 1 correspondence to code.
- Being able to correctly interpret / read code is important as you'll often be working with code that people wrote.