# Instructions

This assignment will test your knowledge and skills in applying the Solving Problem Process taught during weeks 1 and 2 lectures and tutorials and it consists of three parts. The tasks

involved are understanding the problem’s business rules, tackling the challenge by implementing the problem-solving process, and proposing a suitable solution using nothing else than common sense. As such, the assignment requires you to integrate and synthesise what you have learnt so far in this unit to design and create a proper working solution.

## Background: Automated Pet Feeder System

A local animal shelter is looking for a low-cost, programmable automated pet feeder that can:

* Dispense food for cats and dogs at scheduled times.
* Monitor whether food has been consumed or the amount of food that has been consumed.
* Alert staff if there’s an issue (e.g., no food dispensed, food not eaten).

They want a solution that could eventually be implemented using low-cost components (like a servo motor and sensors), but your task is to design and simulate the logic and behaviour of the system first.

# Your Challenge PART 1: On the Solving Problem Process

Using the Integrated Problem-Solving Process, you will:

1. Analyse the problem and define its requirements.
2. Organise and describe all the data and inputs.
3. Design an algorithm to control the system.
4. Implement the solution using plain English.
5. Test and refine your solution with example input values.

**Step-by-Step Instructions**

## Step 1: Understand and Define the Problem (Analyse)

* What features must the feeder include?
* What inputs and outputs are needed (e.g., feeding times, sensors)?
* What are possible assumptions or limitations (e.g., limited memory, one type of pet food)?

*Deliverable: Clear problem statement, assumptions, inputs/outputs, and a simple sketch or block diagram of the system.*

|  |
| --- |
| **Analysing the problem:**  We are required to build and program an automated pet feeder for our client (a local animal shelter organisation) with a set of specific requirements which need to be met for our client.   * The project will begin with phase 1, where we will primarily design and simulate the software-based logic and behaviour of the automatic pet feeder system we will be developing. * Phase 2 will only proceed after the development of Phase 1’s logic algorithm (which will be documented in this file). * Phase 2 will primarily entail the hardware-based development of the automatic pet feeder using budget-oriented components which are easy to attain in case of any hardware malfunctions which may require replacements in the future. Such budget costing components may include a servo motor, appropriate sensors, low-cost memory, etc.   **The required features of the automated pet feeder system include:**   * A time-based scheduled distribution of food for both dogs and cats * The system should be able to monitor how much food has or hasn't been consumed by the pets * It should also push out an alert to notify staff if an issue occurs. e.g. if food was not dispensed by the scheduled time, if the food storage unit is empty, pets not consuming the provided food within a certain period of time, system errors, etc.   **Inputs:**   * Current time (via Real Time Clock) * Scheduled feeding times * Weight sensor readings of food bowl * Level sensor readings of food container * Filling food into food container/storage unit   **Outputs:**   * Servo motor dispenser state * Dispensing food at scheduled times * Alerts to staff * Quantity/weight of monitored food * Documented log records   **Assumptions and potential limitations:**   * System must be budget oriented * The feeder has a limitation of only being able to dispense dry pet food for easy maintenance of system due to limitations of design/budget etc. and to eliminate potential risk of electronic components getting wet * Algorithm will require the total weight of the food bowl to subtract the weight of the bowl itself so that only the weight of the actual food within the bowl is counted/considered by weight sensor. * Assumption that weight and level sensors will provide accurate readings. * Only one bowl is used for dispensing food into for each automatic feeder system/unit. * Source of power will be limited to a wired connection which connects to wall outlets, requiring a constant flow of energy for proper functionality and operation. * Built in memory to record event logs * Method of alerts include blinking LED sequence and sound alert to external system where staff usually are.   **System Block Diagram:**  RTC (Current Time) -> Algorithm Control Logic -> Servo Motor (Dispenses Food) - > Event Logging (Records actions) -> Food Level Sensor (Storage Status) -> Weight Sensor (Bowl Readings) - > Staff Alerts (LED + Sound System Push Out Alert)  A diagram of a machine  AI-generated content may be incorrect. |
|  |

## Step 2: Organise and Describe the Data

* List input types (e.g., real-time clock, food level sensor, weight sensor under bowl).
* List expected outputs (e.g., rotate motor, send alert).
* Provide sample values and operational constraints.

*Deliverable: Data table with inputs/outputs and operational parameters.*

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Example Value** |
| Real Time Clock (RTC) time | Input / Variable | Hours : Mins  (24 hours)   * 7:58   (7:58 am)   * 17:32   (5:32 pm) |
| Current Time | Input / Variable | Same format as RTC |
| Pet Feeding Schedule | Constant | Hours : Mins  (24 hours)  Dog Feeding Times:   * 8:00 * 18:00   Cat Feeding Times:   * 8:30 * 18:30 |
| Food Level Sensor (feeder storage capacity) | Variable / Input | Grams   * 100 g (full) * 21 g (low) |
| Weight Sensor (under bowl) | Variable / Input | 40 g (just dispensed), 3 g (after eating) |
| Rotate Servo Motor (dispenser) | Output | Angles (Degrees)   * No dispense = rotate servo motor 0° * Dispense = rotate servo motor 45° |
| Staff Alerts | Output | * LED Blink = OFF * Alert = ON |
| Food Consumption Threshold | Constant | ≥ 85% of dispensed portion must be eaten |
| Event Logging | Output | * Log Event (Event: Food successfully dispensed at 8:00) * Log Event (Event: Capacity of food not adequate at 8:45) |
| Check feeder capacity | Decision | Sufficient (≥ required grams) / Insufficient (alert staff) |
| Check feeding time | Decision | Compare RTC against schedule |
| Check if food successfully dispensed | Decision | Yes / No |
| Check if food eaten | Decision | Yes (≥85%) / No (<85%) |

## Step 3: Plan the Solution (Design the Algorithm)

* Create decision logic for dispensing food (e.g., “At 8AM and 6PM, rotate servo to dispense food”).
* Add logic to detect errors (e.g., “If bowl weight unchanged 10 mins after feeding, send alert”).
* You must create a flowchart to represent your algorithm using [**Draw.io**.](https://app.diagrams.net/) Follow the instructions shown in the appendix of this assignment.

*Deliverable: A flowchart representing your automated pet feeder logic (exported from Draw.io and included in your assignment report). You must also include in your submission the actual Draw.io file.*

*A screenshot of a computer screen

AI-generated content may be incorrect.*

## Step 4: Implement the Solution (Word Coding)

* Translate your logic into a series of consecutive tasks as shown in the lecture and tutorials.
* Keep it simple, write a modular sequence with meaningful variable names and comments.

*Deliverable: Sequence of tasks with suitable explanations.*

|  |
| --- |
| START  System = ON  LOOP continuously (24/7):    Check feeder capacity (current\_FeederCapacity)  IF (current\_FeederCapacity) insufficient  Log event (log\_Capacity)  Alert staff to refill feeder (alertStaff)  Continue checking capacity until sufficient  WAIT 1 second  Check current RTC time (currentRTC)  IF (currentRTC) time = 08:00 OR 08:30 OR 18:00 OR 18:30  Determine animal to feed based on currentRTC:  08:00 → Dispense dog food (servoMotor = 45°)  08:30 → Dispense cat food (servoMotor = 45°)  18:00 → Dispense dog food (servoMotor = 45°)  18:30 → Dispense cat food (servoMotor = 45°)  ELSE  Don’t dispense any food (servoMotor = 0°)  Continue checking current\_FeederCapacity AND currentRTC  WAIT 1 second  Attempt to dispense food  IF dispense successful  Log success (log\_DSuccess)  ELSE  Log failure (log\_DFail)  Alert staff of failure (alertStaff)  Continue trying to dispense food until success or fix from staff  WAIT 1 second      Wait 15 minutes after for food consumption  Measure bowl weight (measureB\_Weight)  IF food consumed >= 85% of dispensed quantity  Log success of consumption (logC\_Success)  ELSE  Log issue (logC\_Issue)  Alert staff that pet not eating properly (alertStaff)    Re-check feeder capacity (current\_FeederCapacity)  Alert staff if refill needed (alertStaff)    END LOOP/Start from beginning again |

## Step 5: Test and Refine the Solution (Debug and Verify)

Test your logic with some sample scenarios. Have a look at the following examples:

* Pet eats as expected
* Pet does not eat
* Food bin is empty
* Compare output with expectations
* Suggest improvements

*Deliverable: Test outputs, discussion of logic, and system refinements.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Sample Scenarios** | **Output Testing** | **Discussion of Logic** | **Refining the System** |
| **Pet consumes food as expected** | At the set time the feeder dispenses food into the bowl. The weight sensor confirms food is there. After 15 minutes the weight has dropped by more than 85%. The system logs the feeding as successful, and no alerts are triggered. | The logic worked exactly as intended in this case. The system was able to dispense food on time, check that it was eaten, and then continue running without problems. | No changes needed here. A possible small improvement would be to record the exact percentage eaten so staff can track feeding patterns more closely. |
| **Pet does not consume food as expected** | Food is dispensed at the correct time and the sensor shows it is in the bowl. After 15 minutes the weight is almost the same, meaning the pet didn’t eat enough. An alert is triggered, and the event is logged. | The system was able to notice that the food had not been eaten, which is what we want. However, sometimes pets skip meals for harmless reasons so this may cause false alarms. | The system could be improved by re-checking later (for example after 30 minutes) or letting staff adjust the percentage threshold to avoid too many false alerts. |
| **Food container empty** | Before feeding time the feeder checks food storage. The level sensor shows not enough food available, so the servo doesn’t try to dispense. An alert goes to staff, and the event is logged. | The logic prevents the motor from trying to dispense when there is no food, which avoids errors. The alert is important to make sure staff can refill it quickly. | A good upgrade would be an early warning system. For example, send an alert when the food is low but not yet empty, so staff have time to refill before a missed meal. |

**Your Challenge PART 2: On the Use of Technology.**

To promote professional reporting and personal development practices, you are now required to use [**GitHub**](https://github.com/) to manage your project files and collaborate efficiently when needed.

## Step 1: Set Up Your Repository

* Create a public or private repository on [GitHub.](https://github.com/)
* Add your tutor and your lecturer as collaborators under *Settings > Collaborators*.
* Use a clear and descriptive repository name, e.g., `pet-feeder-project`.

**Step 2: Organize Your Repository:**

* Create folders for each step, e.g., `/Step1\_Analysis`, `/Step3\_Flowchart`, `/Step4\_Word\_Code`, etc.
* Include a `**README.md**` file describing your project. - Use meaningful comments to document changes.

**Step 3: Document Your Work:**

* Push your flowchart (e.g., exported PNG/PDF) to the repository.
* Include your word-based code under a clearly named file.
* Upload test results and any additional documentation.

**Submission Requirement:**

* Include the GitHub repository link in your report.

|  |
| --- |
| GitHub Repository Link:   * [Abdullah-R-786/I.I.T-AT1-Pet\_Feeder\_Project: This repository is for Intro to Information Technology - Assessment Task 1 - The Automated Pet Feeder Project. In this repository will include folders for different sections of the assignment with their appropriate files within them for marking.](https://github.com/Abdullah-R-786/I.I.T-AT1-Pet_Feeder_Project) |

* Ensure the repository is accessible to your tutor and unit convener by the due date.

# Your Challenge PART 3: On AI Agent Integration

In this part of the assignment, you will explore how Artificial Intelligence (AI) can assist in solving problems, refining logic, and enhancing your assignment’s documentation. **You are encouraged to use an AI agent** such as Microsoft Copilot to support your work.

Use Copilot to assist with at **least two** or more of the following:

1. **Refine your logic or Word Code**: Ask Copilot to review your Step 4 implementation and suggest improvements or identify potential issues.
2. **Generate alternative solutions**: Prompt Copilot to propose different ways to solve the problem or enhance your flowchart logic.
3. **Explore real-world implementation**: Use Copilot to discuss how your system could be built using actual hardware (e.g., Arduino, Raspberry Pi).
4. **Improve documentation**: Ask Copilot to help you write a professional README.md file or summarize your project for presentation.
5. **Reflect on ethics and limitations**: Use Copilot to explore the ethical implications of using AI in automated pet care or discuss the reliability of AI-generated suggestions.

*Deliverable: Write a short reflection (150–250 words) that includes:*

* *Prompts and responses (what you asked and what it responded with).*
* *What insights or improvements it helped you achieve?*
* *How it influenced your final solution or understanding of the problem?*
* *You may include screenshots or excerpts of your interaction with Copilot if relevant.*

1. **(Option 1) Refining pseudocode logic using Copilot to suggest improvements or identify potential issues**

|  |
| --- |
| I prompted Copilot to review my pseudocode for an automatic pet feeder and suggest any improvements or identify potential issues. Copilot responded and began by highlighting the strengths of my design logic, such as its clear structure, effective scheduling logic, and inclusion of key functionalities like error handling, logging, and consumption monitoring.  It then pointed out several areas for improvement, including the placement of the feeder capacity check and the generic control of the servo motor. One of the most insightful suggestions was to add a delay at the end of each loop declaration to prevent CPU overload. This helped me realize the importance of accounting for hardware limitations and system efficiency in real world implementations.  Another valuable recommendation was to refine the servo motor logic. Copilot explained that using simple ON/OFF states was too vague and suggested specifying rotation angles or wrapping the motor control in a function like ‘*dispenseFood(animal)*’. This led me to revise my pseudocode so that the servo motor now operates based on angle changes, opening to dispense food when conditions are met, and remaining closed otherwise.  Overall, Copilot’s feedback helped me improve the robustness and clarity of my algorithm and deepened my understanding of how to design systems that are both logically sound and practically implementable. |

1. **(Option 4) Asking Copilot to provide aid in writing a professional README.md file for GitHub and general project documentation**

|  |
| --- |
| To improve the documentation of my automatic pet feeder project, I asked Copilot to help me write a professional README.md file and summarize the project for presentation purposes. Copilot responded with a well-structured and clear README template that included sections such as project overview, features, usage as well as suggesting to consider real world hardware components, setup instructions, and usage of the system.  One of the most helpful insights was how Copilot emphasized clarity and accessibility in technical writing. It guided me to write from the perspective of someone unfamiliar with the project, which helped me refine my explanations and ensure that the documentation was easy to follow. Copilot also recommended using markdown formatting to improve readability.  I believe this response from Copilot significantly improved the quality of my documentation and helped me present my project documentation more professionally. It influenced my final solution by encouraging me to think about how others would interact with and understand my work, which is an important aspect of real world engineering and collaboration. |

# Submission Requirements and Checklist

## Checklist Part 1

☐Clearly labelled sections for each problem-solving step.

☐System sketch or diagram (Step 1).

☐Data table (Step 2).

☐Flowchart created with Draw.io (Step 3).

☐Written Sequence of Tasks (Step 4).

☐Sample test cases and discussion (Step 5).

## Checklist Part 2

☐GitHub repository link is included.

☐Student’s GitHub is well structured.

**Checklist Part 3**

☐Reflection of 150-250 words.

## Submission instructions

* **This is a SOLO ASSIGNMENT**.
* Allocated marks are shown in the rubric on page 7 of this assignment.
* This assignment requires you to include all supporting files as indicated in the checklists above.
* Create a folder called “uxxxxx\_Assignment1” and drop all your files in there (problemsolving process answers, flowcharts, Word documents, and the like).
* Compress (zip) ALL your files and folders created above in one single file called uxxxxx\_Assignment1.zip. Upload this file on Canvas by the due date using the drop box provided in the corresponding assignment.

# Appendix

## How to Make a Flowchart using Draw.io

1. **Open Draw.io:**

Go to [Open Draw](https://www.drawio.com/) and click **Start** > **Device** (or **Google Drive** if you want to save it online).

1. **Create a New Diagram** o Choose **"Blank Diagram"**

o Name it something like PetFeeder\_Flowchart

## 3. Use Flowchart Symbols

o Drag and drop standard shapes from the **General** or **Flowchart** section:

* **Oval** = Start / End
* **Rectangle** = Process (e.g., "Dispense Food")
* **Diamond** = Decision (e.g., "Is it 8:00 AM?")
* **Arrow** = Connect the shapes logically

1. **Build the Logic** o Start with “System ON”
   * Include key decisions like:
     + “Is it feeding time?”
     + “Is there food in the container?”
     + “Has the pet eaten?” o Add actions: “Dispense Food”, “Wait 10 minutes”, “Send Alert”, etc. o Use arrows to guide the logical flow
2. **Add Labels and Colours**  o Use colours or text formatting to highlight decision points or alerts
   * Keep the layout clean and readable
3. **Save and Export:**
   * Go to **File > Export As > PDF** or **PNG** o Include the exported image in your assignment submission

# Assignment 1 Grading Rubric

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criteria** | **Excellent (HD)** | **Good (DI)** | **Satisfactory (CR)** | **Needs**  **Improvement (NN)** | **Marks** |
| **Part 1:**  **Problem-**  **Solving Process** |  |  |  |  | **/50** |
| Step 1: Problem Analysis | Clear, detailed problem statement with well-defined assumptions and a professional sketch/block diagram. | Mostly clear with minor gaps in assumptions or diagram clarity. | Basic problem statement with limited assumptions or unclear diagram. | Incomplete or unclear analysis. | /10 |
| Step 2: Data Description | Comprehensive input/output table with realistic sample values and constraints. | Mostly complete with minor omissions. | Basic table with limited examples. | Missing or poorly structured data. | /10 |
| Step 3:  Algorithm  Design  (Flowchart) | Accurate, logical flowchart using correct symbols and structure. | Mostly correct with minor logic or formatting issues. | Basic flowchart with some structural flaws. | Incomplete or incorrect flowchart. | /10 |
| Step 4: Word  Code  Implementation | Clear, modular sequence with meaningful variable names and comments. | Mostly clear with minor issues in structure or naming. | Basic sequence with limited clarity. | Disorganized or unclear code. | /10 |
| Step 5: Testing and Refinement | Thorough testing with insightful  discussion and realistic improvements. | Adequate testing with some discussion. | Basic testing with limited reflection. | Minimal or missing testing. | /10 |
| **Part 2: GitHub Documentation** |  |  |  |  | **/30** |
| Repository Setup | Well-organized repo with clear naming and collaborator access. | Mostly organized with minor issues. | Basic setup with limited structure. | Poorly organized or inaccessible repo. | /10 |
| Documentation  & Commit  History | Clear README, structured folders, and meaningful commit messages. | Mostly complete with minor gaps. | Basic documentation and commits. | Minimal or missing documentation. | /10 |
| File Submission & Structure | All required files correctly uploaded and structured. | Minor issues  in file  naming or structure. | Basic submission with some missing elements. | Incomplete or disorganized submission. | /10 |
| **Part 3: AI Agent Integration** |  |  |  |  | **/20** |
| Use of AI Agent (Copilot) | Thoughtful, relevant use of AI with clear impact on solution. | Adequate use with some reflection. | Basic interaction with limited insight. | Minimal or irrelevant use of AI. | /10 |
| Reflection  Summary | Clear, wellwritten reflection showing understanding and learning. | Mostly clear with minor gaps. | Basic reflection with limited depth. | Missing or unclear reflection. | /10 |
| Total |  |  |  |  | /100 |