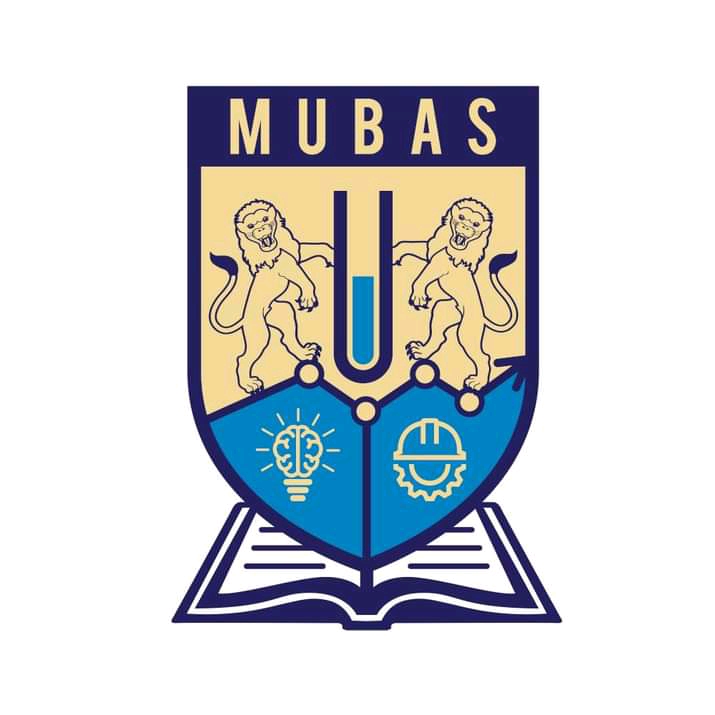
****

**THE MALAWI UNIVERSITY OF BUSINESS AND APPLIED SCIENCES**

**SCHOOL OF SCIENCE AND TECHNOLOGY**

**COMPUTER INFORMATION AND TECHNOLOGY(CIT) DEPARTMENT**

**TO** : DR AMELIA TAYLOR

**FROM :** UPILE CHINGOLO BIT/21/SS/006

**PROGRAMME :** INFORMATION TECHNOLOGY

**MODULE NAME :** ARTIFICIAL INTELLIGENCE

**YEAR OF STUDY :** 4

**ASSIGNMENT :** VACUUM CLEANER AGENT

**DUE DATE** : 13TH FEBRUARY, 2025.

**VACUUM CLEANER DOCUMENTATION**

**TASK 1. EASY**

1. **Functions** 
   * A function vacuum(state) is defined, which takes one parameter, state, representing the cleanliness of two rooms (A and B). Where the state can either be clean or dirty.
2. **Using match Statement for Pattern Matching**
   * The function uses the match statement to compare state against different cases.
   * Each case corresponds to a different room cleanliness condition and determines the vacuum cleaner's actions.
3. **Possible Cases and Actions:**
   * "dirty-dirty": Both A and B are dirty, so it sucks A.
   * "clean-dirty": A is clean, B is dirty, so it moves right and sucks B.
   * "dirty-clean": A is dirty, B is clean, so it moves left and sucks A.
   * "clean-clean": Both rooms are clean, so no action is needed.
4. **Calling the Function:**
   * The function is called with different states, simulating various room conditions.
   * Each function call prints the corresponding action for the given state.

**TASK 2. MEDIUM**

1. **Importing the random Module**
   * The random module is imported to allow random selection of an initial room state.
2. **Defining Possible States**
   * The list states contains four possible cleanliness conditions of two rooms (A and B):
     + "dirty-dirty": Both A and B are dirty.
     + "clean-dirty": A is clean, but B is dirty.
     + "dirty-clean": A is dirty, but B is clean.
     + "clean-clean": Both A and B are clean.
3. **Randomly Selecting an Initial State**
   * The initial state is chosen randomly from the states list using random.choice(states).
   * The selected state is printed to show the starting condition.
4. **While Loop Execution**
   * The vacuum cleaner continues operating until the state becomes "clean-clean".
   * The loop follows a condition-action-update approach based on the current state.
5. **Handling Different Cases Using if-elif Statements**
   * If both rooms are dirty ("dirty-dirty"):
     + The vacuum sucks room A, updates the state to "clean-dirty".
     + Moves right and sucks room B, updating the state to "clean-clean".
   * If room A is clean but B is dirty ("clean-dirty"):
     + The vacuum moves right and sucks B, updating the state to "clean-clean".
   * If room A is dirty but B is clean ("dirty-clean"):
     + The vacuum sucks A, moves right, and updates the state to "clean-clean".
6. **Printing Updated State**
   * After each action, the program prints the updated state.
   * Once "clean-clean" is reached, the loop stops, and a final message is printed:
   * All rooms are clean. Stopping.

**TASK 2**

**1. Importing the random Module**

* The random module is used to randomly set each room’s cleanliness state as either "dirty" or "clean".
* This helps simulate a real-world scenario where some rooms might be dirty and need cleaning.

**2. Defining the Number of Rooms**

* The variable num\_rooms = 5 specifies that there are five rooms in the environment.
* This number can be adjusted to simulate different corridor sizes.

**3. Initializing Room States Randomly**

* A list called rooms is created, where each element is randomly assigned either "dirty" or "clean".
* This is achieved using a list comprehension:
* rooms = [random.choice(["dirty", "clean"]) for \_ in range(num\_rooms)].
* The initial state of all rooms is printed for reference.

**4. Setting the Vacuum's Starting Position**

* The vacuum starts at the first room (index 0) with position = 0.

**5. Running the Cleaning Process with a while Loop**

* The loop continues until all rooms are clean ("dirty" is no longer in rooms).
* Each iteration of the loop represents the vacuum moving from one room to another.

**6. Checking and Cleaning Each Room**

* The vacuum checks if the current room is "dirty".
  + If dirty, it sucks dirt and changes the state to "clean".
  + If already clean, it simply moves to the next room.

**7. Moving Through the Corridor**

* If the vacuum is not in the last room, it moves to the next room and prints "Move to corridor →".
* If the vacuum reaches the last room, it prints "Reached the end of the corridor." and stops moving.

**8. Displaying the Final State**

* After all rooms are cleaned, the program prints the final room states and a stopping message:
* All rooms are clean. Stopping.