UNIVERSITI TEKNOLOGI MALAYSIA

PROGRAMMING TECHNIQUE II

(SCSJ1023)

SEMESTER 2 2024/2025

GROUP PROJECT

PLANT WATERING REMINDER SYSTEM

ABDURRAFIQ BIN ZAKARIA (A24CS0031)

DANIEL IMAN HAQIMIE BIN YUSOFF (A24CS0063)

FATHURRAHMAN HAKIM BIN SULIMAN (A24CS0071)

1/SECRH

SECTION 05

Lecturer:

MADAM LIZAWATI MI YUSUF

27th JUNE 2025

The work in this page has been done by: Hakim	**Date: 20/6/2025**

SECTION A: PROJECT DESCRIPTION

Green-thumbed or not, you won’t ever miss a day to water your plants with our **Plant Watering Reminder System**. The system focuses on providing convenience to its users, reminding them of when to water their plants. The best part is, it’s entirely **customisable and flexible**, depending on the type of the plant, allowing plant owners to **set their own timings**. The system does already have *two predetermined plant types (Flowering, Succulent)*. However, plants, just like us humans, have their own features that make them unique, so we, the development team, tried to keep an open-mind on that fact and made it as flexible as we could to cater to a wide field of plants.

The system aims to provide users with a simple and consistent way to keep track of one of the most important parts of the plant care routine, watering the plant. Users are capable of registering themselves into the system, allowing them to add and configure details about the plants they own. This includes plant, plant type (Flowering, Succulent and Other) and the frequency of watering for the plant. After the info is entered, the plant is saved in an array that belongs to the User, and that specific user only. The system is now prepared to keep track of when each plant has to be watered and was last watered.

For the scope of the system, it’s capable of supporting multiple users and stores user-specific plant data, meaning users hold their own plant data without disrupting other users’ plant data, for better organisation. It aims to be scalable and maintainable, making it easy to introduce new plant types or features in future versions.

Workflow:

The system kicks off by letting users register or log in with their username. Once inside, users can add plants by entering the plant name, type (Flowering, Succulent, or Other), and how often it needs watering. Each user has their own personal plant list, so there's no data clash between users.

Every plant added has a countdown (daysLeft) that tracks when it needs watering again. As each day progresses (simulated using the proceedDay() function), the system automatically updates all plant schedules. When a plant's countdown hits zero, it shows up in the user’s schedule as “due for watering.”

Users can then go to their schedule, check which plants need attention, and mark them as watered, this resets the countdown. The system also allows viewing reports, checking overall progress, and seeing whether all plants have been watered for the day.

All of this is managed using a combination of classes (User, Plant, Reminder, Schedule, etc.) to ensure the program stays clean and manageable, even as more users or plants are added.

OO Concepts:

Our system is built with Object-Oriented Programming (OOP) at its core to make it organized, flexible, and easy to maintain.

- Encapsulation is used throughout the system by grouping data (attributes) and behaviors (methods) into classes like Plant, User, and Reminder. For example, each User manages their own list of plants privately, no other user can mess with it.

- Inheritance is applied in the Plant class and its child classes (Flowering, Succulent, and Other). These child classes inherit basic plant behavior and allow us to customize the watering frequency or type if needed without rewriting everything.

- Polymorphism lets us call methods like getPlantType() or getWateringFrequency() without worrying about which exact plant type we're dealing with. Each plant class responds in its own way, keeping the system flexible and extendable.

- Abstraction helps us hide the complex logic behind simple actions. For example, when a user waters a plant, they just call markAsWatered() and the system handles updating days left, flags, and schedules behind the scenes.

Together, these concepts make the system more than just functional, they make it smart, clean, and future-proof. If we want to add new plant types, features, or improve the reminder system, we can do it without breaking everything else.

The work in this page has been done by: Hakim	**Date: 26/06/2025**

SECTION B: CLASS DIAGRAMS

Class Plant:

Attributes	Description
name	Plant name
PlantType	Type of plant (Succulent, Flowering, Other)
wateringFrequency	How often (in days) a plant is watered
daysLeft	Days left until next watering
wateredToday	Marking the plant as watered or not
Methods	**Description**
virtual getPlantType()	Allows polymorphism in child classes and ease of deciding plant type.
virtual getWateringFrequency()	Allows polymorphism in child classes and ease of deciding watering frequency
markAsWatered()	Sets wateredToday=true, resets daysLeft to wateringFrequency.
proceedDay()	Decrements daysLeft if wateredToday=false; resets wateredToday.
getReport()	Returns a summary string (e.g., "Rose: Water in 2 days").
getName()	Getter for the plant’s name.
setName()	Setter for the plant’s name.
isWateredToday()	Checks if the plant was watered
getDaysLeft()	Returns days until next watering

Class Flowering, Succulent, Other:

Methods	**Description**
getPlantType()	Returns the specific plant type ("Flowering", "Succulent", or "Other")
getWateringFrequency()	Returns the default watering interval for the plant type (e.g., 3 days for Succulents)

Class Schedule:

Attributes	**Description**
plant	Reference to the Plant object being tracked
lastWateredDay	Date when the plant was last watered
nextWateredDay	Calculated date for next watering
Methods	**Description**
updateSchedule()	Recalculates nextWateredDay based on watering frequency
isDue()	Returns true if current date > nextWateredDay
dayUntilNext()	Returns remaining days until next watering

Class User:

Attributes	**Description**
username	Unique display name for the user (e.g., "PlantLover42")
userID	System-generated unique identifier automatically
vector<Plant> plants	Dynamic list of all plants owned by this user
Methods	**Description**
addPlant()	Creates and adds a new Plant to the user's collection
viewSchedule()	Displays all plants with their watering due dates
markPlantAsWatered()	Marks a specific plant as watered and updates its schedule
viewReport()	Generates a summary of all plants' watering statuses
allPlantsWatered()	Returns true if all plants were watered today
proceedDay()	Advances time by 1 day for all plants (updates watering counters)
getUsername()	Returns the user's display name
getUserID()	Returns the user's unique ID
getPlantCount()	Returns the total number of plants in the collection
getPlant()	Retrieves a specific Plant object by index

Class Reminder:

Attributes	**Description**
vector<User> users	Maintains a list of all registered user accounts in the system
Methods	**Description**
addUser()	Creates a new user account and adds it to the users collection
switchUser()	Changes the active user session (for multi-user support)

showUserMenu()	Displays the main interactive menu with user options
showPlantMenu()	Renders plant management options for the current user
proceedDay()	Advances time by 1 day for all users' plants
handlePlantMenu()	Processes user selections from the plant menu (e.g., add/water plants)
run()	Starts the main application loop and manages overall program flow

Class Validator:

Methods	Description
validateFreq(int)	Checks if a watering frequency is valid. Returns true if valid, false if invalid.
validateMenuChoice(int, int, int)	Verifies if a user's menu selection is within the valid range (between min and max). Returns true if valid, false if out of range.

The work in this page has been done by: Rafiq	**Date: 26/06/2025**

SECTION C: SOURCE CODE AND USER MANUAL

All the Source Code in this GitHub link:
UTM/Plant Watering Reminder System | PTII Project at main · abdurrafiqz0304/UTM

User Manual:

1. System Overview

A C++ application to manage plant watering schedules. Supports multiple users and plant types (Succulent, Flowering, Other).

2. Getting Started

2.1 How to Run

1. **Compile:**

```
g++ plant.cpp -o plant_reminder
```

2. **Execute:**

```
./plant_reminder
```

2.2 First-Time Setup

- Enter a username when prompted
- System auto-generates your User ID

3. Main Features

3.1 Adding Plants

1. **Go to: Configure Plants > Add New Plant**
2. **Enter:**
   - Plant name (e.g., "Snake Plant")
   - Type:
     - 1 Succulent (waters every 3 days)
     - 2 Flowering (waters daily)
     - 3 Other (custom settings)

3.2 **Daily Use**

Action	**How To**
Check schedule	Configure Plants > View Schedule
Mark as watered	Select plant > Mark Watered
Advance day	Configure Plants > Proceed Day

4. **User Management**

- Switch between users via Current User menu
- Each user has separate plant lists

5. **Troubleshooting**

- **Invalid input:** System will prompt to re-enter
- **No plants showing:** Confirm user account is selected

6. **Support**

Source code:
[https://github.com/abdurrafiqz0304/UTM/tree/main/Plant%20Watering%20Reminder%20System%20%7C%20PTII%20Project]

APPENDIX A

Plant.cpp

```cpp
// ABDURRAFIQ BIN ZAKARIA A24CS0031
// DANIEL IMAN HAQIMIE BIN YUSUFF A24CS0063
// FATHURRAHMAN HAKIM BIN SULIMAN A24CS0071

#include <iostream>
#include "Reminder.hpp"
using namespace std;

int main() {

    Reminder re;
    re.run();
}
```

Plant.hpp

```
#ifndef PLANT_HPP
#define PLANT_HPP
#include "valid.hpp"

#include <iostream>
#include <string>
#include <cstring>
using namespace std;

class Plant {
    protected:
        string name;
        string plantType;
        int wateringFrequency;
        int daysLeft;
        bool wateredToday;
    public:
        Plant(string name=" ", string type=" ", int freq=0) : name(name),
plantType(type), wateringFrequency(freq) {
            daysLeft = freq;
            wateredToday = 0;
        }
        Plant(const Plant& other) = default;

        virtual string getPlantType() const { return plantType;}
        virtual int getWateringFrequency() const {return wateringFrequency;}
        void markAsWatered() {
            daysLeft = wateringFrequency;
            wateredToday = true;
        }

        void proceedDay() {
            if (daysLeft > 0) {
                daysLeft--;
            } else {
                daysLeft = getWateringFrequency();
            }
        }
        string getReport() const {
            string outputReport;

            outputReport = "Watering Report:";
            outputReport += "\n";
            outputReport += string(25, '-');
            outputReport += "\n";
            outputReport += "Plant: ";
            outputReport += name;
            outputReport += "Type: \n";
            outputReport += plantType;
            outputReport += "\n";

            if (wateredToday) {
                outputReport += "Has already been watered recently!\n Days
until next watering: ";
                outputReport += to_string(daysLeft);
            } else {
                outputReport += "Has not been watered yet today! \n Please
water the plant, thank you.";
            }
            return outputReport;
        }

        string getName() const { return name;}

        void setName(string n) { name = n; }

        bool isWateredToday() const { return wateredToday;}

        int getDaysLeft() const{ return daysLeft;}
    };

#endif
```

PlantType.hpp

```cpp
#ifndef PLANT_TYPE_HPP
#define PLANT_TYPE_HPP
#include "Plant.hpp"
#include "valid.hpp"

#include <iostream>
#include <string>
#include <cstring>
using namespace std;

class Succulent : public Plant {
    public:
        string getPlantType()const override {
            return "Succulent";
        }
        int getWateringFrequency() const override {
            return 3;
        }
};

class Flowering : public Plant {
    public:
        string getPlantType()const override {
            return "Flowering";
        }
        int getWateringFrequency() const override {
            return 1;
        }
};


class Other : public Plant {
    public:
        string getPlantType() {
            cout << "Enter plant type: ";
            string type;
            getline(cin, type);
            return type;
        }
        int getWateringFrequency() const override{
            cout << "Enter watering frequency (days): ";
            int freq;
            cin >> freq;
            Validator v;
            v.validateFreq(freq);
            return freq;
        }
};

#endif
```

Reminder.hpp

```
#ifndef REMINDER_HPP
#define REMINDER_HPP
#include "User.hpp"
#include "Plant.hpp"
#include "PlantType.hpp"
#include "Schedule.hpp"
#include "valid.hpp"

#include <iostream>
#include <vector>
#include <string>
#include <cstring>
using namespace std;

class Reminder {
    private:
        vector<User> users;
        int currentUserIndex = 1;
        public:
        void addUser() {
            string usernames;
            cout << "Enter your username: ";
            getline(cin, usernames);

            User addUser(usernames);
            users.push_back(addUser);

            cout << "\nUser " << usernames << " has been added as User " <<
currentUserIndex++ << ".\n";
            //currentUserIndex = users.size() - 1;
        }
        void switchUser() {
            int index=0;

            if (users.empty()) {
                cout << "No users available.\n";
                return;
            }
            cout << "\nChoose your user:\n";
            for (size_t i = 0; i < users.size(); i++) {
            cout << i + 1 << ") " << users[i].getUsername() << "\n";
            }

            cout << "Please insert using the given order of numbers:\nUser No
: ";

            cin >> index;

            int choice;
            while (true) {
                cout << "Enter user number: ";
                cin >> choice;
                    if (choice >= 1 && choice <=
static_cast<int>(users.size())) {
                        currentUserIndex = choice - 1;
                        cout << "Switched to user: " <<
users[currentUserIndex].getUsername() << "\n";
                    break;
                    }
            cout << "Invalid input. Try again.\n";
        }
        }
        void showUserMenu() {
            cout << "\n==USER MENU==\n";
            cout << "1. Add New User\n";
            cout << "2. Current User\n";
            cout << "3. Configure Plants\n";
            cout << "4. Exit System\n";
            cout << "Enter your choice: ";
        }
        void showPlantMenu() {
            cout << "\n==PLANT MENU==\n";
            cout << "1. Add New Plant\n";
            cout << "2. View Watering Schedule\n";
            cout << "3. Mark Plant as Watered\n";
            cout << "4. View Watering Report\n";
            cout << "5. Proceed Day\n";
            cout << "6. Exit\n";
            cout << "Enter your choice: ";
        }
        void proceedDay() {
            if (currentUserIndex != -1) {
                users[currentUserIndex].proceedDay();
            } else {
                cout << "No user selected.\n";
            }
        }

        void handlePlantMenu() {
            int choice;
            while (true) {
                Validator v;
                showPlantMenu();
                cin >> choice;
                cin.ignore();

                if (cin.fail()) {
                    cout << "Invalid input. Please enter a number." << endl;
                    cin.clear();
                    while (cin.get() != '\n'); // flush junk
                    continue; // skip rest of the loop
                }

                try {
                    v.validateMenuChoice(choice, 1, 6);  // throws if out of
range

                    switch (choice) {
                        case 1: users[currentUserIndex].addPlant();
break;
                        case 2: users[currentUserIndex].viewSchedule();
break;
                        case 3: users[currentUserIndex].markPlantAsWatered();
break;
                        case 4: users[currentUserIndex].viewReport();
break;
                        case 5: users[currentUserIndex].proceedDay();
break;
                        case 6: return;  // Exit plant menu
                    }
                }
                catch (const char* errMsg){
```

```
                cout << "Invalid option. Try again.\n";
            }
        }
    }

    void run() {
        Validator v;
        addUser();

        int choice;

        while (true){
            showUserMenu();
            cin >> choice;
            cin.ignore();

            if (cin.fail()) {
                cout << "Invalid input. Please enter a number." << endl;
                cin.clear();
                while (cin.get() != '\n'); // flush junk
                continue; // skip rest of the loop
            }

            try {
                v.validateMenuChoice(choice, 1, 4);  // throws if out of
range

                switch (choice){
                    case 1: addUser(); break;
                    case 2: switchUser(); break;
                    case 3: handlePlantMenu(); break;
                    case 4: cout << "Exiting the Plant Watering Reminder
System, thank you for time!\n";
                        system ("pause");
                        return;
                }
            }
            catch (const char* errMsg) {
                cout << "Invalid option. Try again.\n";
            }
        }
    }
```

```
};


#endif
```

Schedule.hpp

```
#ifndef SCHEDULE_HPP
#define SCHEDULE_HPP
#include "Plant.hpp"
#include "valid.hpp"

#include <iostream>
#include <string>
#include <cstring>
using namespace std;

class Schedule {
    private:
        Plant* plant;
        int lastWateredDay, nextWateredDay;
    public:
        void updateSchedule() {
            //plant = &plant;
            lastWateredDay = plant->getWateringFrequency() -
plant->getDaysLeft();
        }
    bool isDue() {
        if(plant->getDaysLeft() == plant->getWateringFrequency()) {
            return 1;
        } {
            return false;
        }
    }
    int dayUntilNext() {
        return plant->getDaysLeft();
    }
};

#endif
```

User.hpp

```
#ifndef USER_HPP
#define USER_HPP
#include "Plant.hpp"
#include "PlantType.hpp"
#include "valid.hpp"

#include <iostream>
#include <vector>
#include <string>
#include <cstring>
using namespace std;

class User {
    private:
        string username;
        int userID;
        vector<Plant> plants;
    public:
        User(string name=" ", int id=1) : username(name), userID(id) {}

        // Add this public method to your User class
void debugDisplayPlants() const {
    cout << "\n=== Current Plants in Vector ===\n";
    if (plants.empty()) {
        cout << "No plants stored yet!\n";
        return;
    }

    for (int i = 0; i < plants.size(); i++) {
        cout << "Plant #" << i+1 << ":\n"
             << "  Name: " << plants[i].getName() << "\n"
             << "  Type: " << plants[i].getPlantType() << "\n"
             << "  Water Freq: " << plants[i].getWateringFrequency() << "
days\n";
    };
}

        void addPlant() {
            Validator v;
            string tempName; int tempFreq;
            int plantTypeChoice;
            cout << "Enter plant name: ";
            getline(cin, tempName);  // Reads entire line (spaces allowed)

            // Get plant type
            cout << "Select plant type:\n"
                 << "1. Succulent\n"
                 << "2. Flowering\n"
                 << "3. Other\n"
                 << "Enter your choice: ";
            cin >> plantTypeChoice; cin.ignore();

            Plant tempPlant;

            switch (plantTypeChoice){
                case 1: {Succulent s;
                    tempPlant = Plant(tempName, s.getPlantType(),
s.getWateringFrequency());
                        break;
                }
                case 2: {Flowering f;
                    tempPlant = Plant(tempName, f.getPlantType(),
f.getWateringFrequency());
                        break;
                }
                case 3: {Other o;
                    tempPlant = Plant(tempName, o.getPlantType(),
o.getWateringFrequency());
                        break;
                }
            }

            cout << "Plant added successfully!\n";

            // Get watering frequency (with input validation)
            plants.push_back(tempPlant);

            debugDisplayPlants();
        }
        void viewSchedule() {
            cout << "Watering Schedule:\n";

            for (auto i = plants.begin(); i!=plants.end(); i++){
                cout << "- ";
                cout << i->getName() << "(every " << i->getWateringFrequency()
<< " day(s))\n";
            }
        }
        void markPlantAsWatered() {
            int index = 0;

            if (index >= 0 && index < plants.size() &&
plants[index].isWateredToday() == false){
                plants[index].markAsWatered();
                cout << plants[index].getName() << "has been watered!";
            } else {
                cout << "Has already been watered.";
            }
        }
        void viewReport() const {
                for (int i = 0; i < plants.size(); i++) {
                    cout << plants[i].getReport() << "\n";
            }
        }
        bool allPlantsWatered() {
            for (auto i = plants.begin(); i!=plants.end(); i++){
                if (!(i->isWateredToday())) {
                    return false;
                }
            }
            return 1;
        }
        void proceedDay() {
            cout << "Day has proceeded\n";

            for (auto i = plants.begin(); i!=plants.end(); i++) {
                if (i->getDaysLeft() == 0)
                    cout << "Please water " << i->getName() << "today\n";
                else
                    cout << i->getName() << " is " << i->getDaysLeft() << "
day(s) away from needing to be watered.\n";
```

```cpp
            }
        }
        string getUsername() {
            return username;
        }
        int getUserID() {
            return userID;
        }
        int getPlantCount() {
            return plants.size();
        }
        Plant* getPlant(int index){
            if (index >= 0 && index < plants.size()) {
                return &plants[index];
            } else {
                return nullptr; // For safety if invalid ----> default
            }
        }
};

#endif
```

valid.hpp

```
#ifndef VALID_HPP
#define VALID_HPP

#include <iostream>
#include <string>
#include <cstring>
using namespace std;

class Validator {
public:
    void validateFreq(int freq) {
        if (freq <= 0) {
            throw "Frequency must be greater than 0.";
        }
    }

    void validateMenuChoice(int choice, int min, int max) {
        if (choice < min || choice > max) {
            throw "Menu choice must be within valid range.";
        }
    }
};

#endif
```