

**PROJECT REPORT**

**SUBJECT:**

* **OOP**

**SUBMITTED TO:**

* **Sir Nadeem**

**SUBMITTED BY:**

* **AIMAN WASIM (230201118)**
* **DUA NASEEM (230201054)**

**DEPARTMENT & BATCH:**

* **CS-04-B**
* **Home Automation System Report**

**Introduction**

This C++ program simulates a home automation system that manages multiple devices. The system is capable of dynamically updating the status and settings of these devices based on the current time and temperature. The key functionalities include:

1. Adding various types of devices.
2. Displaying the status of each device.
3. Adjusting settings of devices based on type.
4. Dynamically controlling device behavior based on time and temperature.

**Main Features and Functionalities**

**Time and Temperature Based Control**

The program monitors the current time and temperature to control the devices. For example:

* Lights are turned on at night (between 7 PM and 6 AM).
* Air conditioners are adjusted and turned on if the temperature exceeds 40°C.
* Specific devices are activated at certain times, such as the washing machine at 8 AM and the oven at 7 AM.

**Device Management**

The system can manage up to 10 devices. It supports different types of devices:

* **Light** with brightness control.
* **Air Conditioner** with temperature setting.
* **TV** with volume control.
* **Clock** displaying the current time.
* **Fan**, **Oven**, and **Washing Machine** with basic on/off functionality.

**Class Structure**

**Base Class: Device**

This is an abstract class representing a generic device with common attributes and methods:

* name: The name of the device.
* isOn: A boolean indicating whether the device is on or off.
* Virtual methods for turning the device on/off, getting its status, and setting configurations.

**Derived Classes**

Each specific device type is derived from the Device class and implements its unique characteristics:

* **Light**: Manages brightness.
* **Air Conditioner**: Manages temperature.
* **TV**: Manages volume.
* **Clock**: Displays the current time.
* **Fan**, **Oven**, and **Washing Machine**: Provide basic on/off functionality.

**Controller Class: HomeAutomationController**

This class manages all the devices and contains methods to:

* Add a new device.
* Get the status of all devices.
* Set device settings based on type.
* Update devices based on time and temperature.

**Detailed Implementation**

**Time Formatting**

// Function to format time

string formatTime(time\_t time) {

    struct tm\* tm\_info = localtime(&time);

    char buffer[26];

    strftime(buffer, 26, "%Y-%m-%d %H:%M:%S", tm\_info);

    return string(buffer);

}

This function formats the given time to a human-readable string.

**Night Time Check**

// Function to check if it is night (let's say night is between 7 PM and 6 AM)

bool isNightTime(struct tm\* localTime) {

    int hour = localTime->tm\_hour;

    return (hour >= 19 || hour < 6);

}

Determines if the current time falls within the night period (7 PM to 6 AM).

**Greeting Based on Time**

// Function to display greeting based on the current time

void displayGreeting(struct tm\* localTime) {

    int hour = localTime->tm\_hour;

    if (hour >= 4 && hour < 12) {

        cout << "Good Morning!" << endl;

    } else if (hour >= 12 && hour < 17) {

        cout << "Good Afternoon!" << endl;

    } else if (hour >= 17 && hour < 21) {

        cout << "Good Evening!" << endl;

    } else {

        cout << "Good Night!" << endl;

    }

}

Displays an appropriate greeting based on the current hour.

**Device-Specific Implementations**

Each derived class implements the getStatus method and overrides setSetting as needed. For example, the Light class:

class Light : public Device {

private:

    int brightness;

public:

    Light(string n) : Device(n), brightness(50) {}

    char getType() const override {

        return 'B'; // 'B' for Bulb

    }

    void getStatus() const override {

        cout << "Light " << name << " is " << (isOn ? "on" : "off") << " with brightness " << brightness << "." << endl;

    }

    void setSetting(int setting) override {

        brightness = setting;

    }

};

The Light class includes brightness control and overrides methods to reflect its unique functionality.

**Home Automation Controller**

This class manages the devices and their interactions:

// Controller class to manage all devices

class HomeAutomationController {

private:

    static const int maxDevices = 10;

    Device\* devices[maxDevices];

    int deviceCount;

public:

    HomeAutomationController() : deviceCount(0) {}

    ~HomeAutomationController() {

        for (int i = 0; i < deviceCount; ++i) {

            delete devices[i];

        }

    }

    // Function template to add a device of any type

    template <typename T>

    void addDevice(string name) {

        if (deviceCount < maxDevices) {

            devices[deviceCount++] = new T(name);

        } else {

            cout << "Cannot add more devices, maximum limit reached." << endl;

        }

    }

    void getDeviceStatus() const {

        for (int i = 0; i < deviceCount; ++i) {

            devices[i]->getStatus();

        }

    }

    void setDeviceSetting(char deviceType, int setting) {

        for (int i = 0; i < deviceCount; ++i) {

            if (devices[i]->getType() == deviceType) {

                devices[i]->setSetting(setting);

            }

        }

    }

    // Function to handle dynamic behavior based on time and temperature

    void updateDevicesBasedOnTimeAndTemperature(struct tm\* localTime, int currentTemperature) {

        if (isNightTime(localTime)) {

            for (int i = 0; i < deviceCount; ++i) {

                if (devices[i]->getType() == 'B') {

                    devices[i]->turnOn();

                }

            }

        } else {

            for (int i = 0; i < deviceCount; ++i) {

                if (devices[i]->getType() == 'B') {

                    devices[i]->turnOff();

                }

            }

        }

        for (int i = 0; i < deviceCount; ++i) {

            if (devices[i]->getType() == 'A') {

                AirConditioner\* ac = dynamic\_cast<AirConditioner\*>(devices[i]);

                if (ac) {

                    ac->updateTemperature(currentTemperature);

                    if (currentTemperature > 40) {

                        ac->turnOn();

                    } else {

                        ac->turnOff();

                    }

                }

            }

        }

        // Additional time-based control logic for other devices

        int hour = localTime->tm\_hour;

        int min = localTime->tm\_min;

        int sec = localTime->tm\_sec;

        for (int i = 0; i < deviceCount; ++i) {

            char type = devices[i]->getType();

            if (type == 'W' && hour == 8 && min == 0 && sec < 10) { // Washing machine at 8 AM

                devices[i]->turnOn();

            } else if (type == 'O' && hour == 7 && min == 0 && sec < 10) { // Oven at 7 AM

                devices[i]->turnOn();

            } else if (type == 'A' && hour == 12 && min == 0 && sec < 10) { // AC at 12 PM

                devices[i]->turnOn();

            } else if (type == 'F') { // Fan always on

                devices[i]->turnOn();

            } else if (type == 'B') { // Lights always on

                devices[i]->turnOn();

            }

        }

    }

};

This class includes methods to add devices, retrieve their status, and update their settings based on the time and temperature.

**User Interaction**

The program provides an interactive loop where users can:

1. View the current time and temperature.
2. Enter new times and temperatures to simulate different conditions.
3. Observe how the system dynamically adjusts the status of various devices.

**Main Function**

The main function orchestrates the initialization of devices, time and temperature updates, and user interaction:

int main() {

    HomeAutomationController controller;

    // Adding devices using function template

    controller.addDevice<Light>("Garage Light");

    controller.addDevice<AirConditioner>("Living Room AC");

    controller.addDevice<TV>("Bedroom TV");

    controller.addDevice<Clock>("Wall Clock");

    controller.addDevice<Fan>("Living Room Fan");

    controller.addDevice<Oven>("Kitchen Oven");

    controller.addDevice<WashingMachine>("Laundry Washing Machine");

    atomic<bool> running(true);

    time\_t currentTime = time(0);

    int currentTemperature = 25; // Default temperature

    while (running) {

        system("cls"); // clear the console on Windows, use "clear" for Linux/Mac

        displayCurrentTime(currentTime);

        displayCurrentTemperature(currentTemperature);

        struct tm\* localTime = localtime(&currentTime);

        displayGreeting(localTime);

        controller.updateDevicesBasedOnTimeAndTemperature(localTime, currentTemperature);

        controller.getDeviceStatus();

        // Displaying the time change message

        cout << "Time after 10 seconds will be: " << formatTime(currentTime + 10) << endl;

        // Increment the current time by 10 seconds

        currentTime += 10;

        cout << "Enter new time (HH MM SS) or 'quit' to exit: ";

        string input;

        getline(cin, input);

        if (input == "quit") {

            running = false;

            break;

        } else {

            int hh, mm, ss;

            if (sscanf(input.c\_str(), "%d %d %d", &hh, &mm, &ss) == 3) {

                localTime->tm\_hour = hh;

                localTime->tm\_min = mm;

                localTime->tm\_sec = ss;

                currentTime = mktime(localTime);

            }

        }

        cout << "Enter new temperature or 'skip' to continue: ";

        getline(cin, input);

        if (input == "skip") {

            // Simulate waiting for the next iteration

            for (int i = 0; i < 10; ++i) {

                // Simulating a delay without this\_thread::sleep\_for

                for (volatile int j = 0; j < 100000000; ++j);

            }

            continue;

        } else {

            int temp;

            if (sscanf(input.c\_str(), "%d", &temp) == 1) {

                currentTemperature = temp;

            }

        }

        // Simulate waiting for the next iteration

        for (int i = 0; i < 10; ++i) {

            // Simulating a delay without this\_thread::sleep\_for

            for (volatile int j = 0; j < 100000000; ++j);

        }

    }

    return 0;

}

**Conclusion**

This home automation system demonstrates a practical application of object-oriented programming in C++. It efficiently manages various types of devices and dynamically adjusts their settings based on time and temperature inputs, providing a user-friendly interface for real-time interaction and control.

**THE END**