Introduction :

Water management involves strategies and technologies to optimize water use, prevent waste, and ensure sustainability. Water overflow controller in residential contexts, it focuses on efficient distribution, quality monitoring, and conservation.

The Water Overflow Controller is a C-based simulation designed to calculate and monitor the time required to fill a water tank. By using flow rate parameters, the system estimates when the tank will be full or half-full and provides a real-time countdown to alert the user when to turn off the motor pump.

1. Mathematical Model Elaboration

The core of your controller is based on Flow Dynamics. In a real-world scenario, the time to fill a container is the Volume (V) divided by the Volumetric Flow Rate (Q).

Flow Rate (Q): This is calculated as the product of the pipe's cross-sectional area (A) and the velocity of the water (v).

Formula: The program calculates t using t = V / (A \times v).

Constants used in your code:

A = 0.00061544 (This represents a pipe with a specific radius).

v = 2.125 (Flow speed in units like meters/second).

V = 1.5 (The capacity of the tank being filled).

Program Architecture

The software is structured as a Finite State Machine (FSM). Depending on the user's input, the system enters a specific state (Full or Half fill) and executes a timed loop.

Logic Flow:

Initialization: Variables for time and constants are defined.

Input Phase: The user sees the total calculated time and enters a choice (1 for full, 2 for half).

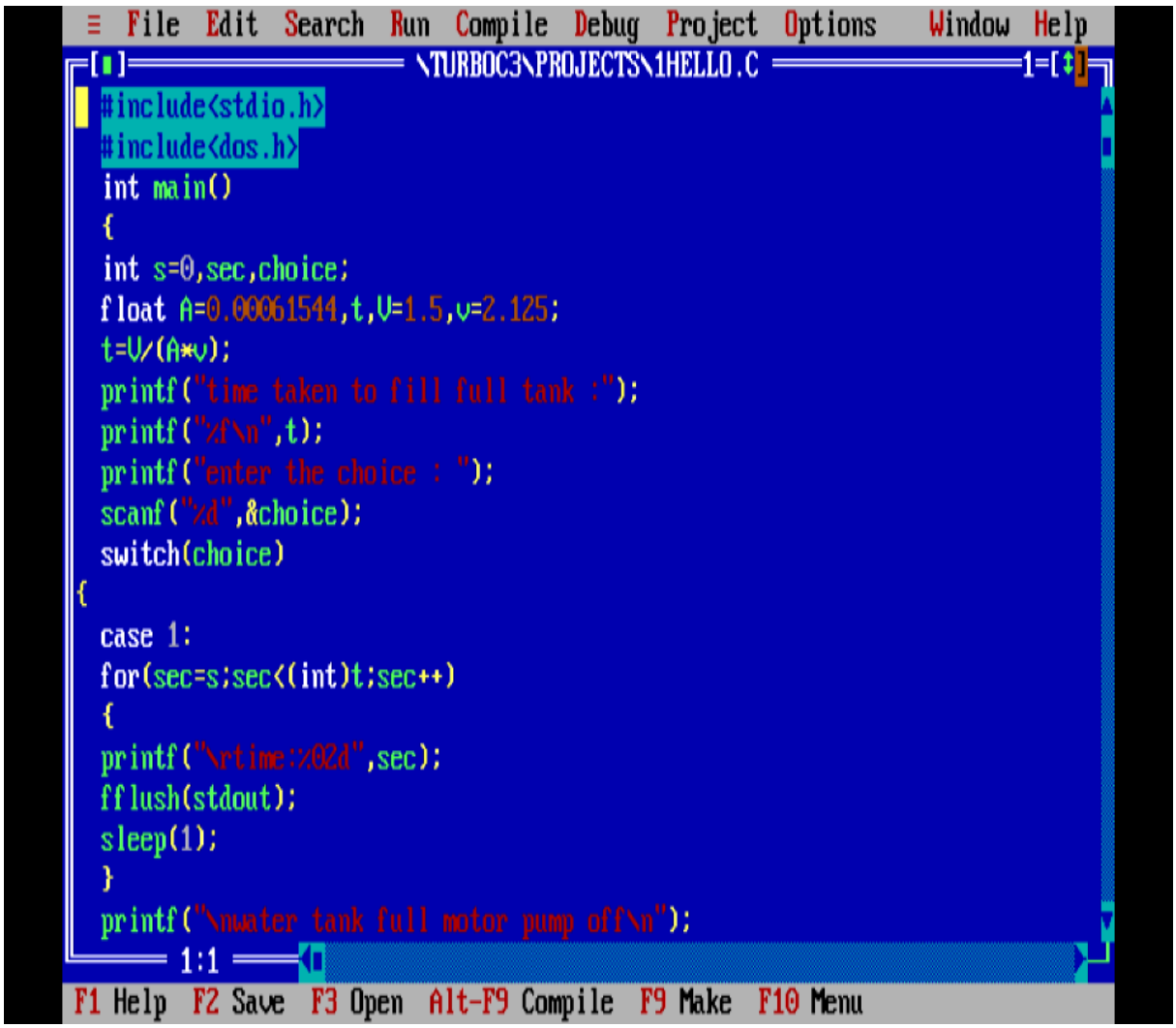
Process Phase (The Loop): * The for loop acts as the clock.

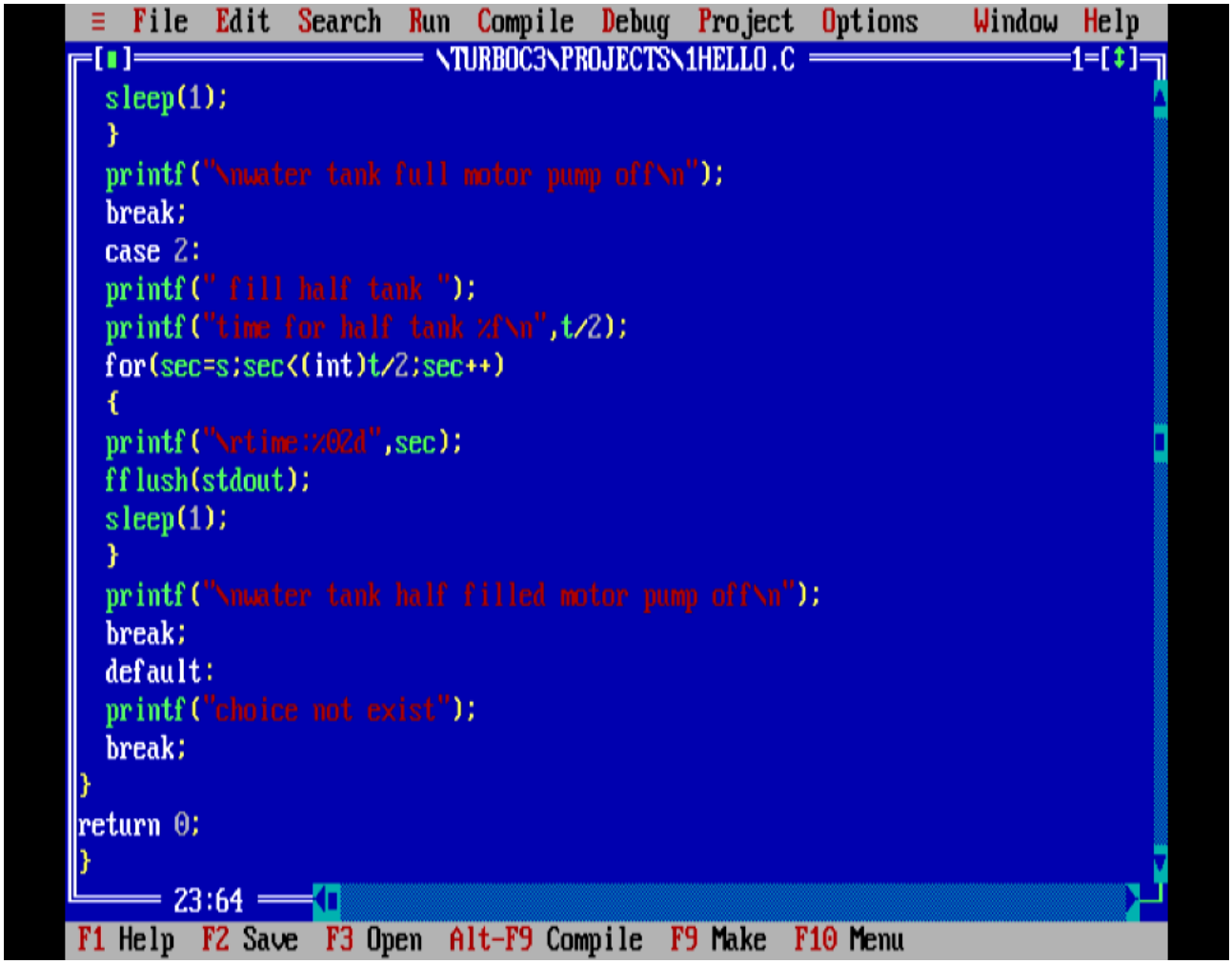
The \r (Carriage Return) in printf("\rtime: %02d", sec) is critical; it moves the

cursor back to the start of the line so the time updates in place rather than printing a long list of numbers.

fflush(stdout) ensures that the output is pushed to the screen immediately, which is necessary in older compilers like Turbo C to show the countdown in real-time.

Completion Phase: Once sec matches the target integer time, the loop breaks and the "Pump Off" message triggers.





Based on the code snippets provided, project : Water Tank Filling Simulator or an automated control logic simulation written in C (using Turbo C++). It calculates the time required to fill a tank based on physical parameters and simulates the countdown.

Here is an analysis of the advantages and disadvantages of the project.

Advantages

Mathematical Precision: Your program uses a specific formula t = V / (A \times v) to calculate filling time. This shows a good integration of physical principles (flow rate and volume) into software logic.

Real-time Simulation: By using sleep(1) and \r (carriage return), you have successfully created a dynamic UI that updates the time on the same line. This is much more user-friendly than printing a new line every second.

Modular Logic: Using a switch-case statement makes the program easy to

expand. You can easily add a "Case 3" for a 75% fill or a "Case 4" for a manual stop.

Low Resource Overhead: Written in C for a DOS environment, this program is extremely lightweight and would run on very low-spec hardware.

Disadvantages

1. Hardcoded Values

Variables like A = 0.00061544, V = 1.5, and v = 2.125 are "hardcoded."

The Issue: If the tank size or pipe diameter changes, you have to rewrite and recompile the code.

The Fix: Use scanf() to let the user input the tank volume and pipe area at runtime.

2. Platform Dependency

The use of <dos.h> and the Turbo C++ environment (shown in your screenshots) is outdated.

The Issue: Modern operating systems (Windows 11, macOS, Linux) do not support dos.h natively.

The Fix: Use standard libraries like <unistd.h> (for sleep) or <windows.h> (for Sleep) to make it cross-platform.

3. Precision Loss in Simulation

In your loop: for(sec=s; sec<(int)t/2; sec++)

The Issue: You are casting a float time to an int. If the calculated time is 573.47 seconds, the loop stops at 573. Over many cycles or larger tanks, these fractional seconds add up to a significant error.

The Fix: Use a more precise timer or handle the remainder after the loop finishes.

4. Lack of Error Handling

The Issue: If a user enters a character instead of a number for the "choice," the program might crash or enter an infinite loop.

The Fix: Add input validation to ensure the user only enters 1 or 2.

Physical Application (Hardware Simulation)

While this is a software simulation, it mimics a Timed Control System.

Current State: The "Motor Pump Off" message represents a signal that, in a hardware-integrated system, would be sent to a Relay Module to physically cut power to a water pump.

Advantage: This method prevents "Sensor Fouling" (where physical water sensors get rusted or dirty) because it relies on mathematical timing rather than physical contact with water.

Analysis of Results

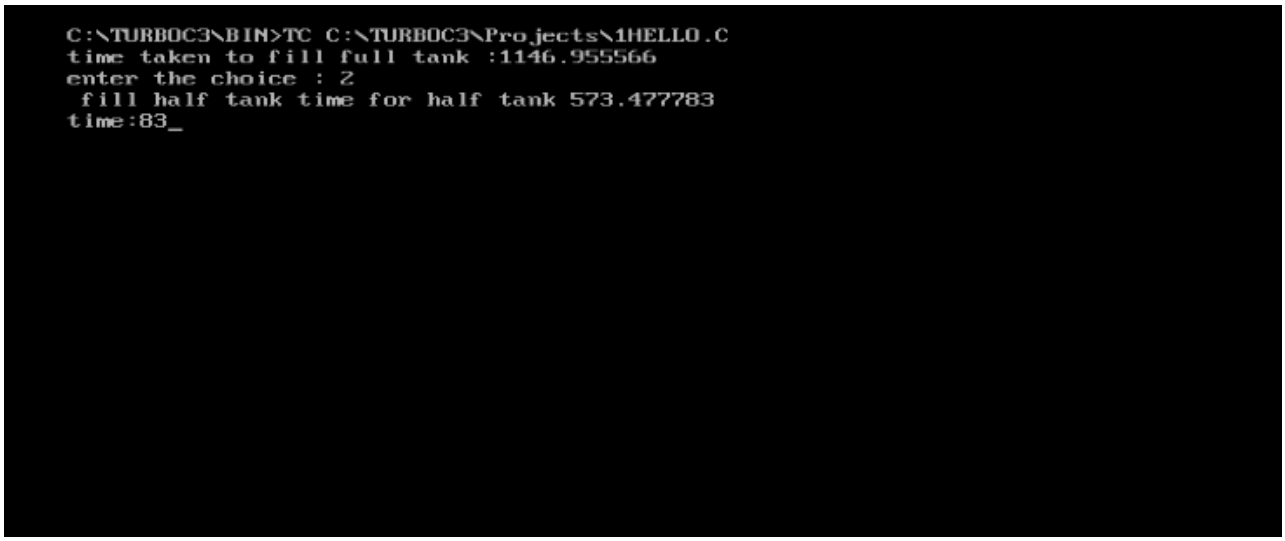
From your execution screen:

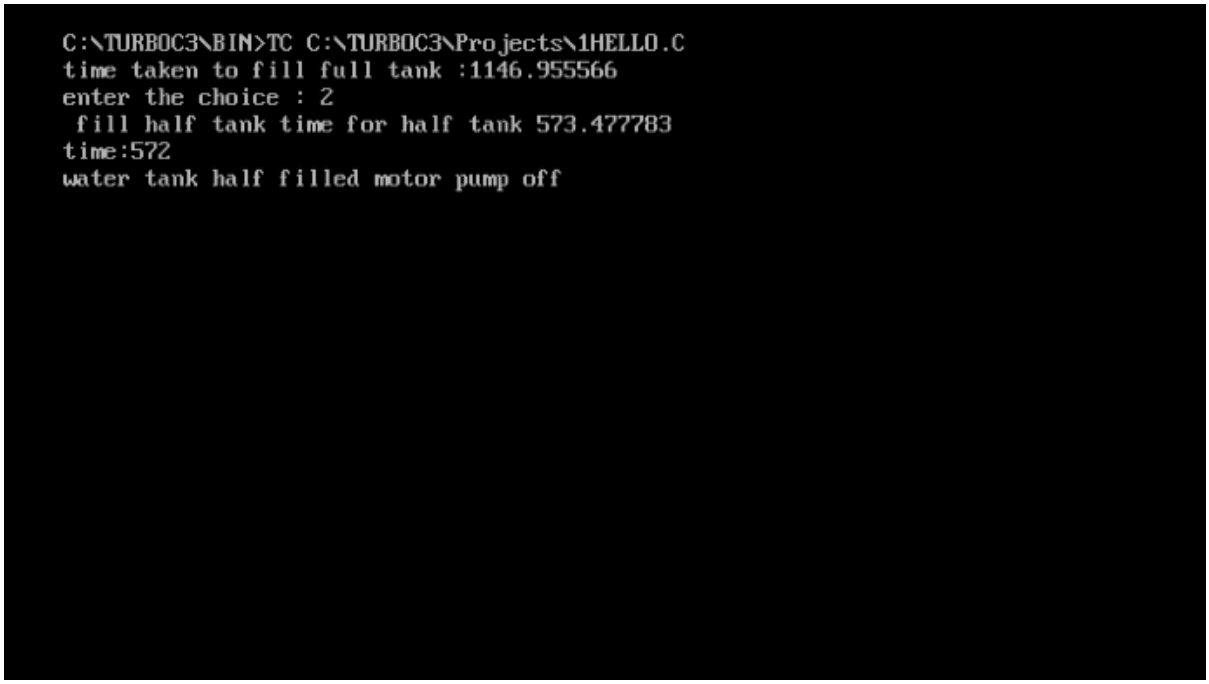
Input: Choice 2 (Half Tank).

Calculated Target: 573.47 seconds.

Behavior: The timer successfully counted up to 572 (the integer limit of the half-time).

Output: The program accurately identified the end of the cycle and issued the "motor pump off" command.





Conclusion

The project successfully demonstrates a automated timing system for water management. It accurately calculates flow time and provides a user-friendly interface to prevent water wastage by alerting the user exactly when the tank reaches the desired level.

C programming mini project

D PAUL ZEBULUN

711525BEC079

ECE – ‘B’