FACULTY OF ENGINEERING AND TECHONOLOGY

COMPUTER PROGRAMMING

COURSE PROJECT

WATER SUPPLY AND PIPED IRRIGATION DESIGN

BY GROUP SIX

PRESENTED TO MR. MASERUKA BENEDICTO

DATE OF SUBMISSION;

11TH NOVEMBER,2025.

AIM;This report is aimed to partially contribute to the continuous assesment for computer programming course marks.

GROUP SIX MEMBERS

NAME	COURSE	REG NUMBER
OPIO SAM ODWORI	WAR	BU/UP/2024/4454
MUWAZA DEOGRATIUS	PTI	BU/UP/2024/5398
KATUNGUKA EMMANUEL	WAR	BU/UP/2024/1030
JOAN RACHAEL LAGU	WAR	BU/UP/2024/1028
KISAKYE PRISCILA PATIANC	WAR	BU/UP/2024/1034
AYOO LINDA	AMI	BU/UP/2024/5097
WAMBASA GEOFERY	WAR	BU/UP/2024/3259
NAMANYA PRECIOUS	AMI	BU/UP/2024/5255
NANGOLI DERRIK	AMI	BU/UP/2024/3736
APUDA CECILIA LINDI	WAR	BU/UP/2024/3248

DECLARATION

We group six members declare that all the work embodied in this report is of our own efforts and that it was not duplicated from any.

NAMES	SIGNATURE
AYOO LINDA	
OPIO SAMUEL ODWORI	
JOAN RACHAEL LAGU	
KISAKYE PRISCILA PATIENCE	
NAMANYA PRECIOUS	
WAMBASA GEOFREY	
KATUNGUKA EMMANUEL	
MUWAZA DEOGRATIUS	
APUDA CECILIA LINDI	
NANGOLI DERICK	

ACKNOWLEDGEMENT

We thank the Almighty God for the gift of life, knowledge and constant guidance throughout the course of this assignment. We extend our sincere gratitude to our beloved parents for their support in our academics and also our lecturer Mr. Maseruka Benedicto (HOD) for his continuous efforts in ensuring we learn computer programming thoroughly.

We finally thank our fellow group six members for their cooperation and active participation in this assignment through our group leader Wambasa Geofrey.

APPROVAL.

This is to confirm that this report has been prepared and presented by group six without duplication but research.

Date ………………………………………………………………

Signature…………………………………………………………….

DEDICATION

We dedicate this report to all group SIX members, who worked tirelessly to ensure that its completed.

METHODOLOGY

In this assignment, we utilized our knowledge of ALL course modules and graphical user interface to relate real world problems .

.

EXECUTIVE SUMMARRY

- Irrigation system
- Pipe network
- Pump system
- Test script
- Water system element
- Graphical user interface

LIST OF ACRONYMS

W.S……….Water Supply

OBJ……….Objective

IS………….Irrigation system

TABLE OF CONTENTS

Catalog

CHAPTER ONE

BACKGROUND

Water is one of the essential natural resource for human survival, agricultural , production and industrial development.The design of efficient water supply and piped irrigation systems plays a crucial roll in ensuring the sustainable and reliable delivery of these resources for domestic and commercial use.

PROBLEM STATEMENT

Many communities face unreliable water supply due to poor irrigation pipelin design leakages and inadequate infrastructure . This leads to water losses low pressure and limited access to clean water. There is therefore a need to design an efficient piped water system that ensures reliable , safe and continuous water distribution.

OBJECTIVES OF THE PROJECT

To design an efficient and sustainable water supply and irrigation system that ensures reliable delivery of clean water for both domestic and agricultural use, maintain adequate pressure , minimise losses and promote effective management for improved productivity.

SIGNIFICANCE OF THE STUDY

The design of piped water supply and irrigation system is important in improving stardards of living and agricultural output.It ensures continuous access to clean water, efficient water use reduce wastage and supports food securit

CHAPTER TWO:

Question

Amatlab design of water supply and irrigation to mitigate most of the challenges that engineers face during implementions of such projects.

IRRIGATION SYSTEM

```
% IrrigationSystem.m

classdef IrrigationSystem

% Manages the collection of elements and performs the simulation (The main Class).

properties (Access = public)

Elements % Array of WaterSystemElement objects (Encapsulation)

Nodes % Map of node IDs to their elevation/demand properties

end

methods

function obj = IrrigationSystem()

% Constructor

obj.Elements = Pipe.empty(0, 1);

obj.Nodes = containers.Map('KeyType', 'int32', 'ValueType', 'any');

end


% Method to add elements (Encapsulation)

function addElement(obj, element)

% Check for inheritance to ensure correct type (Robustness)

if isa(element, 'WaterSystemElement')

obj.Elements(end+1) = element;

else

error('Element must inherit from WaterSystemElement.');

end

end

% Method to define a node (reservoir, junction, demand point)

function defineNode(obj, node_id, elevation_m, demand_lps)

obj.Nodes(node_id) = struct('Elevation', elevation_m, 'Demand',
demand_lps/1000); % Convert to m^3/s

end

% High-level Simulation Method (Abstraction/System Logic)
```

```
function [flows, heads] = runSimulation(obj)
fprintf('\n Starting Water Network Simulation \n');
% --- Initial Setup ---
node_ids = cell2mat(obj.Nodes.keys);
num_nodes = length(node_ids);
% Initialize heads (e.g., to reservoir head)
heads = ones(num_nodes, 1) * 50;
% Simple iterative head balance (Newton-Raphson/Linear)
MAX_ITER = 10;
TOLERANCE = 0.01; % m of head
for iter = 1:MAX_ITER
max_change = 0;
% 1. Calculate Flows for the current heads (Polymorphism in action!)
flows = zeros(length(obj.Elements), 1);
% 2. Build the System Matrix (Simplified Nodal Balance)
% For simplicity, we'll iterate through nodes and update head
% This is a highly simplified representation of a Hardy Cross or NR solver.
for i = 1:num_nodes
node_id = node_ids(i);
node_data = obj.Nodes(node_id);
% Calculate net flow at the node (Sum(Inflows) - Sum(Outflows) - Demand)
net_flow = -node_data.Demand; % Start with demand as an outflow
% Find connected elements and calculate their flow based on current head
for k = 1:length(obj.Elements)
element = obj.Elements(k);
% Get connected node heads
h_start = heads(node_ids == element.StartNode);
h_end = heads(node_ids == element.EndNode);
if isa(element, 'Pipe')
head_diff = h_start - h_end; % Assume flow is from start to end
Q_k = element.calculateFlow(head_diff);
if element.StartNode == node_id
net_flow = net_flow - Q_k; % Outflow from node
elseif element.EndNode == node_id
```

```
net_flow = net_flow + Q_k; % Inflow to node
end
flows(k) = Q_k; % Store flow magnitude/direction
elseif isa(element, 'Pump')
% Pump flow must be solved simultaneously with system head loss,
% which is complex. For a high-level view, we'll assume a
% desired flow and calculate the required pump head.
% A full implementation would require a true network solver.
if element.StartNode == node_id || element.EndNode == node_id
% Skip for simplified nodal balance
end
end
end
% 3. Update Head (Very simple correction based on flow imbalance)
% This is NOT a real hydraulic solver, but demonstrates the concept.
head_correction = net_flow * 0.5; % '0.5' acts as a damping factor
new_head = heads(i) + head_correction;
change = abs(new_head - heads(i));
max_change = max(max_change, change);
heads(i) = new_head;
end
fprintf('Iteration %d: Max Head Change: %.4f m\n', iter, max_change);
if max_change < TOLERANCE
fprintf('Convergence achieved.\n');
break;
end
end
% Final Output
fprintf('\n Simulation Complete \n');
obj.displaySystemInfo(heads, node_ids);
% Note: For a real simulation, you'd integrate the Pump head gain into the matrix.
end
% Private method for displaying results (Encapsulation)
function displaySystemInfo(obj, heads, node_ids)
```

```
% Display Heads
fprintf('\n Nodal Heads \n');
T_Heads = table(node_ids', heads, 'VariableNames', {'Node_ID', 'Head_m'});
disp(T_Heads);
% Display Element Info (Polymorphism allows us to call displayInfo)
fprintf('\n Element Status\n');
for k = 1:length(obj.Elements)
obj.Elements(k).displayInfo();
if isa(obj.Elements(k), 'Pipe')
% Need to recalculate final flow using final heads
h_start = heads(node_ids == obj.Elements(k).StartNode);
h_end = heads(node_ids == obj.Elements(k).EndNode);
final_Q = obj.Elements(k).calculateFlow(h_start - h_end);
fprintf(' Calculated Flow: %.4f m^3/s (%.1f L/s)\n', final_Q, final_Q*1000);
end
end
end
end
end
```

PIPE NETWORK

```
% Pipe.m
classdef Pipe < WaterSystemElement
% Represents a simple pipe segment.
properties (Access = private)
Diameter_m % Diameter in meters (Encapsulation)
Length_m % Length in meters (Encapsulation)
Roughness_mm % Manning's or Hazen-Williams roughness factor
end
methods
function obj = Pipe(id, start_node, end_node, D, L, R)
% Constructor
obj = obj@WaterSystemElement(id, start_node, end_node); % Call base constructor
```

```
obj.Diameter_m = D;
obj.Length_m = L;
obj.Roughness_mm = R;
end
% Implementation of the abstract method (Polymorphism)
function Q = calculateFlow(obj, head_diff)
% Simplified Hazen-Williams calculation for flow (Q) given Head Loss (h_L).
% h_L = (10.67 * L * Q^1.852) / (C^1.852 * D^4.87)
% We solve for Q given h_L (which is head_diff)
C = 130; % Hazen-Williams C-factor (common for new PVC)
K = (10.67 * obj.Length_m) / (C^1.852 * obj.Diameter_m^4.87);
% Q = (head_diff / K)^(1/1.852)
Q = (abs(head_diff) / K)^(1/1.852);
if head_diff < 0 % Adjust direction based on head difference
Q = -Q;
end
end
% Implementation of the abstract method (Polymorphism)
function displayInfo(obj)
fprintf(' Pipe ID: %d, Length: %.1f m, Diameter: %.2f m\n', ...
obj.getID(), obj.Length_m, obj.Diameter_m);
end
end
end
```

PUMP SYSTEM

```
% Pump.m
classdef Pump < WaterSystemElement
% Represents a pump with a simple head curve.
properties (Access = private)
% Simplified Head Curve: H = A - B*Q^2 (Encapsulation)
A % Max Head (intercept)
B % Coefficient for Q^2
end
methods
```

```
function obj = Pump(id, start_node, end_node, max_head, coeff)

% Constructor

obj = obj@WaterSystemElement(id, start_node, end_node);

obj.A = max_head;

obj.B = coeff;

end

% Implementation of the abstract method (Polymorphism)

function H = calculateHeadGain(obj, Q)

% Calculate the head (H) added by the pump at a given flow (Q)

H = obj.A - obj.B * Q^2;

if H < 0

H = 0; % Pump cannot add negative head

end

end

% The 'calculateFlow' method is complex for a pump and usually solved iteratively

% within the main system model (Polymorphism applied differently here)

function Q_dummy = calculateFlow(~, ~)

warning('Pump flow is determined by system demand and head balance, not simple
head difference.');

Q_dummy = 0;

end

% Implementation of the abstract method (Polymorphism)

function displayInfo(obj)

fprintf(' Pump ID: %d, Max Head (A): %.1f m\n', obj.getID(), obj.A);

end

end

end
```

TEST SCRIPTS

```
% main_design_script.m

clear;

clc;


% Create the Irrigation System Network

system = IrrigationSystem();
```

```
% Define Nodes (Abstraction)
% Node ID, Elevation (m), Demand (L/s)
% Node 1: Reservoir (High Elevation, 0 Demand)
system.defineNode(1, 100, 0);
% Node 2: Intermediate Junction
system.defineNode(2, 90, 0);
% Node 3: Irrigation Sector 1 (Demand Point)
system.defineNode(3, 80, 50);
% Node 4: Irrigation Sector 2 (Demand Point)
system.defineNode(4, 75, 75);


% Add Elements (Inheritance & Encapsulation)

% Pipe 1: Connects Reservoir (1) to Junction (2)
pipe1 = Pipe(101, 1, 2, 0.30, 500, 0.01); % D=0.3m, L=500m
system.addElement(pipe1);

% Pipe 2: Connects Junction (2) to Sector 1 (3)
pipe2 = Pipe(102, 2, 3, 0.20, 300, 0.01); % D=0.2m, L=300m
system.addElement(pipe2);

% Pipe 3: Connects Junction (2) to Sector 2 (4)
pipe3 = Pipe(103, 2, 4, 0.25, 450, 0.01); % D=0.25m, L=450m
system.addElement(pipe3);

% Run the Simulation (Polymorphism and System Abstraction)
[final_flows, final_heads] = system.runSimulation();

% Data Extraction for Plotting
% Get the node IDs in the same order as the final_heads array
node_ids = cell2mat(system.Nodes.keys);
% Get the initial elevations in the corresponding order
```

```
elevations = arrayfun(@(id) system.Nodes(id).Elevation, node_ids);
% Get the Pipe IDs in the same order as the final_flows array
pipe_ids = arrayfun(@(e) e.getID(), system.Elements);



% Plot 1: Nodal Heads vs. Elevation
figure;
% Plot both Elevation and Final Head side-by-side
% FIX: Transpose 'elevations' (elevations') to match the orientation of
'final_heads'
bar(node_ids, [elevations' final_heads]);
title('Nodal Head and Elevation');
xlabel('Node ID');
ylabel('Head (m)');
legend('Elevation', 'Final Head', 'Location', 'northwest');
set(gca, 'XTick', node_ids); % Ensure all Node IDs are displayed
grid on;



% Plot 2: Hydraulic Grade Line (HGL)
figure;
title('Hydraulic Grade Line (HGL) and Pipe Head Loss');
xlabel('Distance Along Path (Conceptual)');
ylabel('Head (m)');
hold on;
grid on;

% Create an array to hold the conceptual network path
path_nodes = [1, 2, 3, 4]; % Example path: Reservoir -> Junction -> Sector 1 & 2



% Simplified: Just plot the head at each node, ordered by a logical path (1-2-3-4)
conceptual_path_head = arrayfun(@(id) final_heads(node_ids == id), path_nodes);
conceptual_path_elevation = arrayfun(@(id) system.Nodes(id).Elevation, path_nodes);
```

```
conceptual_path_x = [0 500 800 500]; % X-coords: Node 1 (0), Node 2 (500), Node 3
(800), Node 4 (500)

% Plot HGL
plot(conceptual_path_x, conceptual_path_head, 'b-o', 'LineWidth', 2,
'MarkerFaceColor', 'b', 'MarkerSize', 8);
% Plot Elevation Line
plot(conceptual_path_x, conceptual_path_elevation, 'k--', 'LineWidth', 1);

% Add labels for the nodes
text(conceptual_path_x, conceptual_path_head, cellstr(num2str(path_nodes')),
'VerticalAlignment', 'bottom');

legend('Hydraulic Grade Line (HGL)', 'Ground Elevation', 'Location', 'best');
hold off;
```

WATER SYSTEM ELEMENT

```
% WaterSystemElement.m
classdef (Abstract) WaterSystemElement < handle & matlab.mixin.Heterogeneous
% An Abstract Base Class defining the required interface for all network
components.
properties (Access = protected)
ID % Unique identifier
StartNode % Node where the element begins
EndNode % Node where the element ends
end
methods (Abstract)
% Required methods (Polymorphism/Abstraction)
calculateFlow(obj, head_difference);
displayInfo(obj);
end
methods
function obj = WaterSystemElement(id, start_node, end_node)
% Constructor
obj.ID = id;
```

```
obj.StartNode = start_node;
obj.EndNode = end_node;
end
function id = getID(obj)
% Encapsulation: Controlled access to ID
id = obj.ID;
end
end
end
```

GRAPHICAL USER INTERFACE

```
function WaterSystemGUII()

% Setup the Main Figure
fig = uifigure('Name', 'Water System Simulator (V2)', 'Position', [100 100 850
650]);

% Input Panel
inputPanel = uipanel(fig, 'Title', 'Simulation Parameters', 'Position', [10 550
830 90]);

% Reservoir Head Input (Node 1)
uilabel(inputPanel, 'Text', 'Reservoir Head (m):', 'Position', [10 35 120 22]);
reservoirHeadEdit = uieditfield(inputPanel, 'numeric', 'Value', 60, ...
'Position', [130 35 80 22]);

% Demand Input (Node 2)
uilabel(inputPanel, 'Text', 'Demand (L/s) at Node 2:', 'Position', [230 35 140
22]);
demandEdit = uieditfield(inputPanel, 'numeric', 'Value', 100, ...
'Position', [370 35 80 22]);
% Pipe Diameter Input (Pipe 101)
uilabel(inputPanel, 'Text', 'Pipe 101 Diameter (m):', 'Position', [470 35 140 22]);
diameterEdit = uieditfield(inputPanel, 'numeric', 'Value', 0.30, ...
```

```
'Position', [610 35 80 22]);


% Run Simulation Button
uibutton(inputPanel, 'Text', 'Run Simulation', ...
'Position', [700 30 120 30], ...
'ButtonPushedFcn', @(~,~) runSimulationCallback(fig, reservoirHeadEdit, demandEdit,
diameterEdit));


% Plotting Axes (FIXED) ---
resultsAxes = uiaxes(fig, 'Position', [10 70 830 470]);
% Set the title and labels using the correct object-oriented syntax
resultsAxes.Title.String = 'Simulation Results: Nodal Heads';
resultsAxes.XLabel.String = 'Node ID';
resultsAxes.YLabel.String = 'Head (m)';
% Store the axes handle
fig.UserData.ResultsAxes = resultsAxes;


% Save Plot Button
uibutton(fig, 'Text', 'Save Plot', ...
'Position', [700 20 140 30], ...
'ButtonPushedFcn', @(~,~) savePlotCallback(fig));
end


function runSimulationCallback(fig, reservoirHeadEdit, demandEdit, diameterEdit)
% Retrieve User Inputs
try
res_head = reservoirHeadEdit.Value;
demand_lps = demandEdit.Value;
pipe_diameter = diameterEdit.Value;
catch
uialert(fig, 'Please enter valid numbers for all input fields.', 'Input Error');
return;
end
% Basic validation
```

```
if res_head <= 0 || demand_lps < 0 || pipe_diameter <= 0
uialert(fig, 'Head, Demand, and Diameter must be positive.', 'Input Error');
return;
end

try
% Setup the Irrigation System
sys = IrrigationSystem();

% Define Nodes:
sys.defineNode(1, res_head, 0);
sys.defineNode(2, 20, demand_lps);
sys.defineNode(3, 10, 0);

% Define Elements:
sys.addElement(Pipe(101, 1, 2, pipe_diameter, 500, 0.05));
sys.addElement(Pipe(102, 2, 3, 0.20, 300, 0.05));
sys.addElement(Pump(201, 3, 1, 30, 0.001));

% Run the Simulation
[~, heads] = sys.runSimulation();
% Plot the Results
resultsAxes = fig.UserData.ResultsAxes;
% Get node IDs and sort them to ensure plotting order matches node ID
node_ids = cell2mat(sys.Nodes.keys);
[sorted_node_ids, sort_idx] = sort(node_ids);
sorted_heads = heads(sort_idx);

cla(resultsAxes); % Clear previous plot
bar(resultsAxes, sorted_node_ids, sorted_heads);
% Add text labels for the head values
for i = 1:length(sorted_node_ids)
text(resultsAxes, sorted_node_ids(i), sorted_heads(i) + 1, ...
sprintf('%.1f', sorted_heads(i)), ...
```

```
'HorizontalAlignment', 'center', 'FontSize', 9);
end
% Update title string
resultsAxes.Title.String = sprintf('Simulated Nodal Heads (Reservoir Head: %.1fm, Demand: %.1fL/s)', res_head, demand_lps);
resultsAxes.XLabel.String = 'Node ID';
resultsAxes.YLabel.String = 'Head (m)';
grid(resultsAxes, 'on');

catch ME
% Display any internal simulation errors in a UI alert
uialert(fig, ['An error occurred during simulation: ', ME.message], 'Simulation Error');
end
end

function savePlotCallback(fig)
resultsAxes = fig.UserData.ResultsAxes;
if isempty(resultsAxes.Children)
uialert(fig, 'No plot data to save. Run the simulation first.', 'Save Error');
return;
end
% Open file dialog
[filename, pathname] = uiputfile({'*.png','PNG Image (*.png)'; ...
'*.eps','Encapsulated PostScript (*.eps)'}, ...
'Save Plot As', 'WaterSystemPlot.png');
if ischar(filename) % Check if user didn't cancel
fullPath = fullfile(pathname, filename);
% Save the content of the axes
try
exportgraphics(resultsAxes, fullPath);
uialert(fig, ['Plot saved successfully to: ', fullPath], 'Save Success');
catch ME
uialert(fig, ['Failed to save plot: ', ME.message], 'Save Error');
end
```

```
end
End
```

PUMP SYSTEM

```
% Pump.m
classdef Pump < WaterSystemElement
% Represents a pump with a simple head curve.
properties (Access = public)
% Simplified Head Curve: H = A - B*Q^2 (Encapsulation)
A % Max Head (intercept)
B % Coefficient for Q^2
end
methods
function obj = Pump(id, start_node, end_node, max_head, coeff)
% Constructor
obj = obj@WaterSystemElement(id, start_node, end_node);
obj.A = max_head;
obj.B = coeff;
end
% Implementation of the abstract method (Polymorphism)
function H = calculateHeadGain(obj, Q)
% Calculate the head (H) added by the pump at a given flow (Q)
H = obj.A - obj.B * Q^2;
if H < 0
H = 0; % Pump cannot add negative head
end
end
% The 'calculateFlow' method is complex for a pump and usually solved iteratively
% within the main system model (Polymorphism applied differently here)
function Q_dummy = calculateFlow(~, ~)
warning('Pump flow is determined by system demand and head balance, not simple
head difference.');
Q_dummy = 0;
end
```

```matlab
% Implementation of the abstract method (Polymorphism)

function displayInfo(obj)

fprintf(' Pump ID: %d, Max Head (A): %.1f m\n', obj.getID(), obj.A);

end

end

end
```

CHAPTER THREE:

3.1 CHALLENGES

- Limited time given for the assignment to be completed.
- Referencing errors at times made the work hectic
- Lack of concentration due to the different course units being handled at the same time

3.2 RECOMMENDATIONS

- We recommend that the lecturer to always give us ample time to complete the assignment.

3.3 CONCLUSION AND LEARNING EXPERIENCE

Upon assignment completion, we really appreciated the MATLAB especially from the introduction part to the coverage. This embedded a real-life application of the software into the different engineering aspects. We gained a deeper rhythm on problem solving for water and piped irrigation system and also the entire coverage of ALL modules . This experience was of utmost importance to all of us.