1. #!/bin/bash

# Prompt the user for input

read -p "Enter Customer ID: " customer\_id

read -p "Enter Customer Name: " customer\_name

read -p "Enter Units Consumed: " units\_consumed

# Function to calculate the bill based on unit consumption

calculate\_bill() {

local units=$1

local bill=0

if [[ $units -le 199 ]]; then

bill=$((units \* 1.2))

elif [[ $units -le 400 ]]; then

bill=$((199 \* 1.2 + (units - 199) \* 1.5))

elif [[ $units -le 600 ]]; then

bill=$((199 \* 1.2 + 201 \* 1.5 + (units - 400) \* 1.8))

else

bill=$((199 \* 1.2 + 201 \* 1.5 + 200 \* 1.8 + (units - 600) \* 2.0))

fi

echo $bill

}

# Calculate the bill

bill=$(calculate\_bill $units\_consumed)

# Apply a surcharge (replace with your desired surcharge calculation)

surcharge=$(echo "$bill \* 0.1" | bc) # Assuming a 10% surcharge

total\_bill=$(echo "$bill + $surcharge" | bc)

# Display the bill details

echo "Electricity Bill for Customer ID: $customer\_id"

echo "Customer Name: $customer\_name"

echo "Units Consumed: $units\_consumed"

echo "Bill Amount: Ksh $bill"

echo "Surcharge (10%): Ksh $surcharge"

echo "Total Bill Amount: Ksh $total\_bill"

1. #!/bin/bash

# Prompt the user for input

read -p "Enter Employee Name: " employee\_name

read -p "Enter Hours Worked: " hours\_worked

read -p "Enter Rate per Hour (Ksh): " rate\_per\_hour

# Calculate basic pay

basic\_pay=$(echo "$hours\_worked \* $rate\_per\_hour" | bc)

# Function to calculate tax based on basic pay

calculate\_tax() {

local pay=$1

local tax=0

if [[ $pay -gt 70000 ]]; then

tax=$(echo "$pay \* 0.25" | bc)

elif [[ $pay -gt 15000 ]]; then

tax=$(echo "$pay \* 0.15" | bc)

fi

echo $tax

}

# Calculate tax

tax=$(calculate\_tax $basic\_pay)

# Calculate net pay

net\_pay=$(echo "$basic\_pay - $tax" | bc)

# Display employee pay details

echo "Employee Name: $employee\_name"

echo "Basic Pay (Ksh): $basic\_pay"

echo "Tax (Ksh): $tax"

echo "Net Pay (Ksh): $net\_pay"]

1. #include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

int main() {

// File name to open

char filename[] = "message.txt";

// Open the file in write mode with read/write permissions for the user

int file\_descriptor = open(filename, O\_WRONLY | O\_CREAT, 0644);

// Check if opening the file was successful

if (file\_descriptor == -1) {

perror("Failed to open file");

exit(1);

}

// Message to write to the file

char message[] = "Hello World\n"; // Includes a newline character

// Write the message to the file

ssize\_t bytes\_written = write(file\_descriptor, message, sizeof(message));

// Check if writing to the file was successful

if (bytes\_written == -1) {

perror("Failed to write to file");

close(file\_descriptor); // Close the file even on error

exit(1);

}

printf("Successfully wrote %ld bytes to the file.\n", bytes\_written);

// Rewind the file pointer to the beginning for reading

lseek(file\_descriptor, 0, SEEK\_SET);

// Buffer to store the read content

char buffer[100];

// Read content from the file

ssize\_t bytes\_read = read(file\_descriptor, buffer, sizeof(buffer) - 1);

// Check if reading from the file was successful

if (bytes\_read == -1) {

perror("Failed to read from file");

close(file\_descriptor); // Close the file even on error

exit(1);

}

// Null-terminate the buffer to ensure a valid string

buffer[bytes\_read] = '\0';

// Print the read content

printf("Read content from the file:\n%s\n", buffer);

// Close the file

if (close(file\_descriptor) == -1) {

perror("Failed to close file");

exit(1);

}

printf("File closed successfully.\n");

return 0;

}