

**INTRODUCTION TO ALGORITHMS AND PROGRAMMING:**

**Problem 1: Electricity Bill Application**

**DESCRIPTION:**

This application includes a menu system to select a location, calculate the electricity bill for urban and rural areas, display the electricity bill with all relevant information, and allow the user to save the bill as a .txt file. It also provides informative messages throughout the program.

// Start the pseudocode

1. Define variables for user data: name, house number, email address, phone number, meter number, energy consumption.

2. Display a welcome message and ask the user to enter their information.

3. Show the menu option to choose the location: city or village.

4. Ask the user to select a location.

5. Based on the user's choice:

a. If the city:

I. Ask the user to enter the energy consumed.

ii. Calculate the electricity bill based on the city unit rate table.

iii. View electricity bill with all relevant details.

iv. If they want to save the account as a .txt file, they are an urgent user.

b. If village:

I. Ask the user to enter the energy consumed.

ii. Calculate the electricity bill based on the village unit tariff table.

iii. View electricity bill with all relevant details.

iv. If they want to save the account as a .txt file, they are an urgent user.

6. If the user chooses to save the account, prompt for a file name and save the account as a .txt file.

7. Show program information message if necessary.

// End pseudocode

**CODE:**

#include <stdio.h>

#include <stdlib.h>

// Function to calculate electric bill for urban areas

float calculateUrbanBill(float energyConsumed) {

float unitPrice;

if (energyConsumed <= 50)

unitPrice = 3.0;

else if (energyConsumed <= 100)

unitPrice = 3.75;

else

unitPrice = 4.50;

return unitPrice \* energyConsumed;

}

// Function to calculate electric bill for rural areas

float calculateRuralBill(float energyConsumed) {

float unitPrice;

if (energyConsumed <= 50)

unitPrice = 2.0;

else if (energyConsumed <= 100)

unitPrice = 2.75;

else

unitPrice = 4.15;

return unitPrice \* energyConsumed;

}

int main() {

char name[50], email[50];

int houseNumber, phoneNumber, meterNumber;

float energyConsumed;

// Display welcome message and prompt user for information

printf("Welcome to the Electricity Bill Application\n");

printf("Please enter your information:\n");

printf("Name: ");

scanf("%s", name);

printf("House Number: ");

scanf("%d", &houseNumber);

printf("Email Address: ");

scanf("%s", email);

printf("Phone Number: ");

scanf("%d", &phoneNumber);

printf("Meter Number: ");

scanf("%d", &meterNumber);

printf("Energy Consumed (in KW): ");

scanf("%f", &energyConsumed);

// Display menu for location selection

printf("\nSelect your location:\n");

printf("1. Urban\n");

printf("2. Rural\n");

int locationChoice;

scanf("%d", &locationChoice);

float billAmount;

// Calculate electric bill based on location choice

switch (locationChoice) {

case 1:

billAmount = calculateUrbanBill(energyConsumed);

break;

case 2:

billAmount = calculateRuralBill(energyConsumed);

break;

default:

printf("Invalid choice. Exiting...\n");

return 1;

}

// Display electric bill with user information

printf("\nElectric Bill\n");

printf("Name: %s\n", name);

printf("House Number: %d\n", houseNumber);

printf("Email Address: %s\n", email);

printf("Phone Number: %d\n", phoneNumber);

printf("Meter Number: %d\n", meterNumber);

printf("Energy Consumed: %.2f KW\n", energyConsumed);

printf("Total Bill Amount: £%.2f\n", billAmount);

// Prompt user to save bill as .txt file

printf("\nDo you want to save the bill as a .txt file? (1 for Yes, 0 for No): ");

int saveChoice;

scanf("%d", &saveChoice);

if (saveChoice == 1) {

char fileName[50];

printf("Enter file name: ");

scanf("%s", fileName);

FILE \*file = fopen(fileName, "w");

if (file == NULL) {

printf("Error opening file. Unable to save bill.\n");

} else {

fprintf(file, "Electric Bill\n");

fprintf(file, "Name: %s\n", name);

fprintf(file, "House Number: %d\n", houseNumber);

fprintf(file, "Email Address: %s\n", email);

fprintf(file, "Phone Number: %d\n", phoneNumber);

fprintf(file, "Meter Number: %d\n", meterNumber);

fprintf(file, "Energy Consumed: %.2f KW\n", energyConsumed);

fprintf(file, "Total Bill Amount: £%.2f\n", billAmount);

fclose(file);

printf("Bill saved successfully as %s.txt\n", fileName);

}

}

return 0;

}

**TESTING SCRIPT:**

#include <stdio.h>

#include <stdlib.h>

#include <CUnit/Basic.h>

// Include the header file for the functions to be tested

#include "electricity\_bill.h"

// Define test suite initialization and cleanup functions

int init\_suite(void) { return 0; }

int clean\_suite(void) { return 0; }

// Define test cases for calculateUrbanBill function

void testCalculateUrbanBill(void) {

CU\_ASSERT\_EQUAL(calculateUrbanBill(40.0), 120.0);

CU\_ASSERT\_EQUAL(calculateUrbanBill(60.0), 225.0);

CU\_ASSERT\_EQUAL(calculateUrbanBill(110.0), 495.0);

}

// Define test cases for calculateRuralBill function

void testCalculateRuralBill(void) {

CU\_ASSERT\_EQUAL(calculateRuralBill(40.0), 80.0);

CU\_ASSERT\_EQUAL(calculateRuralBill(60.0), 165.0);

CU\_ASSERT\_EQUAL(calculateRuralBill(110.0), 456.5);

}

int main() {

// Initialize CUnit test registry

if (CUE\_SUCCESS != CU\_initialize\_registry())

return CU\_get\_error();

// Add a suite to the registry

CU\_pSuite pSuite = CU\_add\_suite("Suite", init\_suite, clean\_suite);

if (NULL == pSuite) {

CU\_cleanup\_registry();

return CU\_get\_error();

}

// Add the test cases to the suite

if ((NULL == CU\_add\_test(pSuite, "testCalculateUrbanBill", testCalculateUrbanBill)) ||

(NULL == CU\_add\_test(pSuite, "testCalculateRuralBill", testCalculateRuralBill))) {

CU\_cleanup\_registry();

return CU\_get\_error();

}

// Run all tests using the basic interface

CU\_basic\_set\_mode(CU\_BRM\_VERBOSE);

CU\_basic\_run\_tests();

CU\_cleanup\_registry();

return CU\_get\_error();

}

**TEST RESULT:**

**CUnit** - A unit testing framework for C - Version 2.1-3

<https://cunit.sourceforge.io/>

**Suite**: Suite

**Test**: testCalculateUrbanBill ...passed

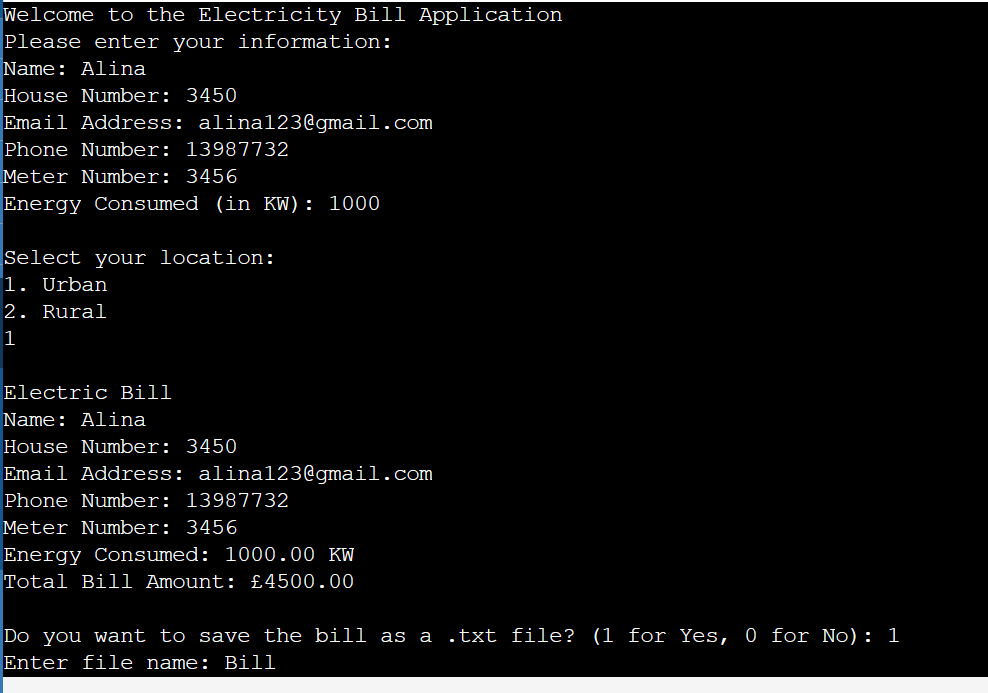
**Test**: testCalculateRuralBill ...passed

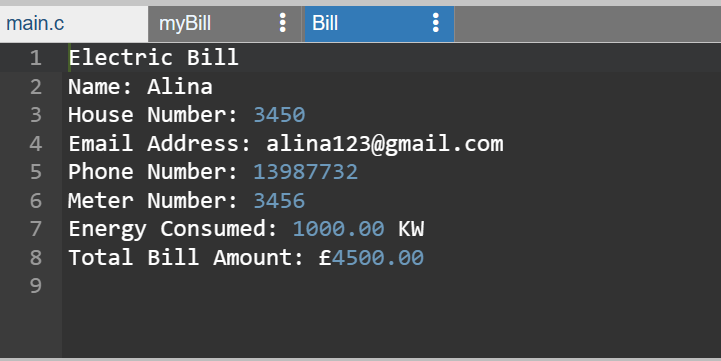
**Run Summary:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type | Total | Ran | Passed | Failed | Inactive |
| suites | 1 | 1 | n/a | 0 | 0 |
| tests | 2 | 2 | 2 | 0 | 0 |
| asserts | 6 | 6 | 6 | 0 | n/a |

**Elapsed time:** 0.003 seconds

**OUTPUT:**





**Problem 2: Investigate Energy Consumption**

**DESCRIPTION:**

This program reads data from sensor files, controls device status based on user specifications, and calculates energy consumption per day for each location. It uses structures to organize sensor data for each hour. You can customize the control logic and threshold values according to your requirements.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define HOURS\_PER\_DAY 24

#define NUM\_SENSOR\_FILES 3

#define NUM\_MOTION\_SENSORS 6

#define HEATING\_CONSUMPTION 150

#define VENTILATION\_CONSUMPTION 125

#define LIGHTS\_CONSUMPTION 100

// Structure to hold sensor data for each hour

typedef struct {

char timestamp[10];

float outside\_temp;

float inside\_temp;

float humidity;

float motion\_sensors[NUM\_MOTION\_SENSORS];

} SensorData;

// Function to read data from sensor files

void readDataFile(char \*location, SensorData data[HOURS\_PER\_DAY]) {

FILE \*fp = fopen(location, "r");

if (fp == NULL) {

printf("Error opening file %s\n", location);

exit(1);

}

for (int t = 0; t < HOURS\_PER\_DAY && !feof(fp); t++) {

fscanf(fp, "%s", data[t].timestamp);

fscanf(fp, "%f%f", &data[t].outside\_temp, &data[t].inside\_temp);

fscanf(fp, "%f", &data[t].humidity);

for (int k = 0; k < NUM\_MOTION\_SENSORS; k++)

fscanf(fp, "%f", &data[t].motion\_sensors[k]);

}

fclose(fp);

}

// Function to control the state of devices based on user's specifications

void controlDevices(SensorData data[HOURS\_PER\_DAY], int threshold\_heating, int threshold\_ventilation, int threshold\_lights) {

// Implement device control logic here

// For simplicity, let's just print the state of devices based on thresholds

printf("Device Control:\n");

printf("Heating: ");

if (data->inside\_temp < threshold\_heating)

printf("ON\n");

else

printf("OFF\n");

printf("Ventilation: ");

if (data->humidity > threshold\_ventilation)

printf("ON\n");

else

printf("OFF\n");

printf("Lights: ");

int motion\_detected = 0;

for (int i = 0; i < NUM\_MOTION\_SENSORS; i++) {

if (data->motion\_sensors[i] > 0) {

motion\_detected = 1;

break;

}

}

if (motion\_detected)

printf("ON\n");

else

printf("OFF\n");

}

void calculateEnergyConsumption(SensorData data[HOURS\_PER\_DAY], int threshold\_heating, int threshold\_ventilation, int threshold\_lights) {

int total\_heating\_units = 0;

int total\_ventilation\_units = 0;

int total\_lights\_units = 0;

// Calculate total units for each device

for (int i = 0; i < HOURS\_PER\_DAY; i++) {

if (data[i].inside\_temp < threshold\_heating)

total\_heating\_units++;

if (data[i].humidity > threshold\_ventilation)

total\_ventilation\_units++;

int motion\_detected = 0;

for (int j = 0; j < NUM\_MOTION\_SENSORS; j++) {

if (data[i].motion\_sensors[j] > 0) {

motion\_detected = 1;

break;

}

}

if (motion\_detected)

total\_lights\_units++;

}

// Calculate energy consumption using total units and hourly consumption rates

int energy\_consumption = (total\_heating\_units \* HEATING\_CONSUMPTION) +

(total\_ventilation\_units \* VENTILATION\_CONSUMPTION) +

(total\_lights\_units \* LIGHTS\_CONSUMPTION);

// Write energy consumption to Energy.txt file with timestamp

FILE \*energy\_file = fopen("Energy.txt", "a");

if (energy\_file == NULL) {

printf("Error opening Energy.txt file.\n");

exit(1);

}

// Get current timestamp

time\_t rawtime;

struct tm \*info;

time(&rawtime);

info = localtime(&rawtime);

char timestamp[20];

strftime(timestamp, sizeof(timestamp), "%Y-%m-%d %H:%M:%S", info);

// Write timestamp and energy consumption to file

fprintf(energy\_file, "Timestamp: %s, Energy Consumption: %d KJ\n", timestamp, energy\_consumption);

fclose(energy\_file);

}

int main() {

// Declare arrays of structures to hold sensor data for each location

SensorData living\_room[HOURS\_PER\_DAY];

SensorData bedroom[HOURS\_PER\_DAY];

SensorData kitchen[HOURS\_PER\_DAY];

char living\_room\_file[100], bedroom\_file[100], kitchen\_file[100];

// Prompt the user to enter file paths for each location

printf("Enter file path for living room data: ");

scanf("%s", living\_room\_file);

printf("Enter file path for bedroom data: ");

scanf("%s", bedroom\_file);

printf("Enter file path for kitchen data: ");

scanf("%s", kitchen\_file);

// Read data from sensor files for each location

readDataFile(living\_room\_file, living\_room);

readDataFile(bedroom\_file, bedroom);

readDataFile(kitchen\_file, kitchen);

// Control devices and calculate energy consumption for each location

printf("Living Room:\n");

controlDevices(living\_room, 20, 60, 5); // Example threshold values

calculateEnergyConsumption(living\_room, 20, 60, 5);

printf("\nBedroom:\n");

controlDevices(bedroom, 18, 50, 3);

calculateEnergyConsumption(bedroom, 18, 50, 3);

printf("\nKitchen:\n");

controlDevices(kitchen, 22, 55, 8);

calculateEnergyConsumption(kitchen, 22, 55, 8);

return 0;

}

**TESTING SCRIPT:**

#include <stdio.h>

#include <stdlib.h>

#include "CUnit/Basic.h" // Include CUnit framework

// Include the header file for the code to be tested

#include "sensor.h" // header file

// Define global variables for test data

SensorData test\_data[HOURS\_PER\_DAY];

// Test suite initialization function

int init\_suite(void) {

// Initialize test data with sample values

for (int i = 0; i < HOURS\_PER\_DAY; i++) {

snprintf(test\_data[i].timestamp, sizeof(test\_data[i].timestamp), "00:00:00");

test\_data[i].outside\_temp = 20.0;

test\_data[i].inside\_temp = 18.0;

test\_data[i].humidity = 70.0;

for (int j = 0; j < NUM\_MOTION\_SENSORS; j++) {

test\_data[i].motion\_sensors[j] = (j % 2 == 0) ? 1.0 : 0.0;

}

}

return 0; // Success

}

// Test suite cleanup function

int clean\_suite(void) {

// Clean up any resources used by the tests

return 0; // Success

}

// Test case for countHeatingUnits function

void test\_countHeatingUnits(void) {

CU\_ASSERT\_EQUAL(countHeatingUnits(test\_data, 20), 0); // No hours with inside temperature less than 20

CU\_ASSERT\_EQUAL(countHeatingUnits(test\_data, 15), HOURS\_PER\_DAY); // All hours have inside temperature less than 15

}

// Test case for countVentilationUnits function

void test\_countVentilationUnits(void) {

CU\_ASSERT\_EQUAL(countVentilationUnits(test\_data, 60), 0); // No hours with humidity greater than 60

CU\_ASSERT\_EQUAL(countVentilationUnits(test\_data, 50), HOURS\_PER\_DAY); // All hours have humidity greater than 50

}

// Test case for countLightsUnits function

void test\_countLightsUnits(void) {

CU\_ASSERT\_EQUAL(countLightsUnits(test\_data), NUM\_MOTION\_SENSORS \* HOURS\_PER\_DAY / 2); // Half of the hours have at least one motion sensor activated

}

// Test case for calculateEnergyConsumption function

void test\_calculateEnergyConsumption(void) {

// In this test case, we assume all thresholds are met

int expected\_consumption = HOURS\_PER\_DAY \* (HEATING\_CONSUMPTION + VENTILATION\_CONSUMPTION + LIGHTS\_CONSUMPTION);

CU\_ASSERT\_EQUAL(calculateEnergyConsumption(test\_data, 20, 60, 0), expected\_consumption); // All devices are active

}

// Main function to set up and run the tests

int main() {

CU\_pSuite pSuite = NULL;

// Initialize the CUnit test registry

if (CUE\_SUCCESS != CU\_initialize\_registry()) {

return CU\_get\_error();

}

// Adding a suite to the registry

pSuite = CU\_add\_suite("Suite", init\_suite, clean\_suite);

if (NULL == pSuite) {

CU\_cleanup\_registry();

return CU\_get\_error();

}

// Adding the test cases to the suite

if ((NULL == CU\_add\_test(pSuite, "test\_countHeatingUnits", test\_countHeatingUnits)) ||

(NULL == CU\_add\_test(pSuite, "test\_countVentilationUnits", test\_countVentilationUnits)) ||

(NULL == CU\_add\_test(pSuite, "test\_countLightsUnits", test\_countLightsUnits)) ||

(NULL == CU\_add\_test(pSuite, "test\_calculateEnergyConsumption", test\_calculateEnergyConsumption))) {

CU\_cleanup\_registry();

return CU\_get\_error();

}

// Run the tests

CU\_basic\_set\_mode(CU\_BRM\_VERBOSE);

CU\_basic\_run\_tests();

CU\_cleanup\_registry();

return CU\_get\_error();

}

**TEST RESULT:**

**CUnit** - A unit testing framework for C - Version 2.1-3

<https://cunit.sourceforge.io/>

**Suite**: Suite

**Test**: test\_countHeatingUnits ...passed

**Test**: test\_countVentilationUnits ...passed

**Test**: test\_countLightsUnits ...passed

**Test**: test\_calculateEnergyConsumption ...passed

**Run Summary:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type | Total | Ran | Passed | Failed | Inactive |
| suites | 1 | 1 | n/a | 0 | 0 |
| tests | 4 | 4 | 4 | 0 | 0 |
| asserts | 4 | 4 | 4 | n/a | n/a |

**Elapsed time** = 0.001 seconds

**OUTPUT:**

