Computer Applications (EPE1040)
Smart Meter Group Project 2024-2025

SEC	ID	Name
2	91240589	لؤى علي حسن علي حسن
2	91241228	ريم الرشيد احمد الحاج
2	91241032	مايكل مرقص يواقم عطاالله
2	91240573	كريم عصام سيد محمود حسن الخطيب
3	91240713	محمود محسن محمود حسن

Filehandling.h

Purpose:

-Declare variables to store current, voltage and time.

-Declare functions to input and output csv file data.

Filehandling.cpp

Well, starting with the files we did 2 file handling files obviously the .h for declarations and .cpp for definitions. We made 2 functions one to input data from a file and the other to save the output data in a file named by the user.

-Readfile function:

Purpose:

-It inputs data from a file and store it in vectors.

Explanation:

We first started with clearing each vector since we are using only 3 vectors globally throughout our program so it would be a mess if we called this function to input data into those 3 vectors twice without clearing them first.

After that all it does is just using getline to input the data we didn't of course use the traditional way to do it (fin>>time>>volt>>current) as of the csv files nature that separates values by commas which is non defined by (fin) since (fin) expects a white space to separate between values and store it correctly in variables so once it encounters the first comma the program will fail so we solved that problem by using the getline method we get the whole line and then store it in a variable the we used stringstream to convert the string to a stream of characters to facilitate parsing certain sections of the string stream string we obviously set the delimiter to be the comma so it parses the first value and stops once it encounters the comma then we stored that value in a variable then converted it to a double using stod() once we did that we stored our values in 3 vectors (time, voltage, current) those are the vectors that we used in our entire program throughout our functions.

-Outputdata function:

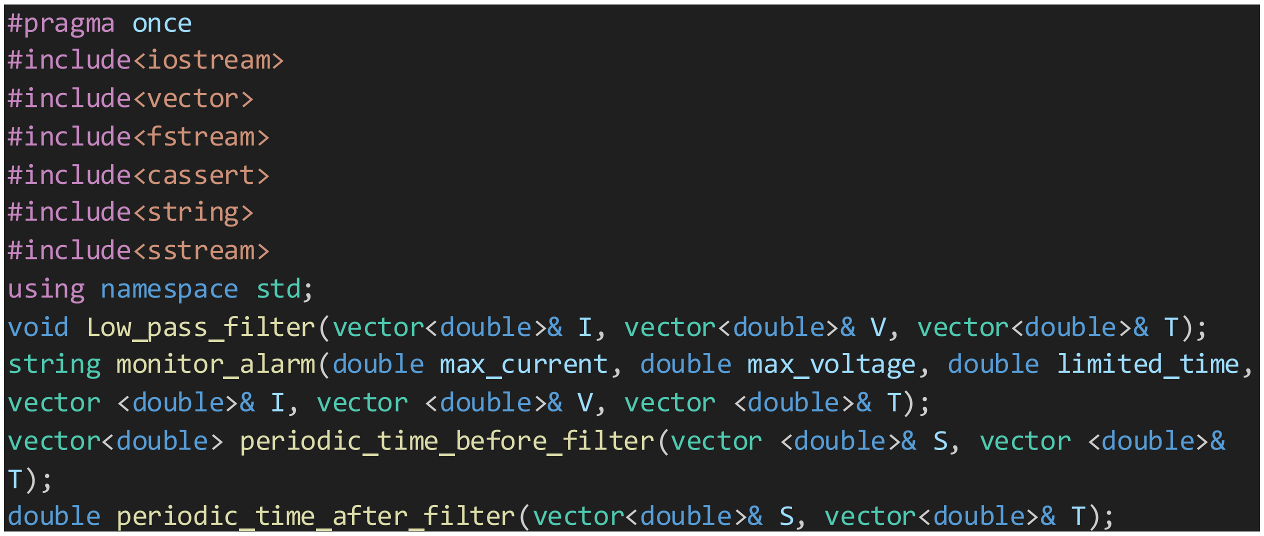
Purpose:

It takes the output data of our program and saves it in csv file named by the user.

Explanation:

for the output function that was much simpler we just stored our output data in variables and then created a function that takes these data and output it into a csv file we used the common method of course by using (ofstream fout) then we outputted a comma between each value since we are going to output the data into a csv file.

Alarm.h



```
#pragma once
#include<iostream>
#include<vector>
#include<fstream>
#include<cassert>
#include<string>
#include<sstream>
using namespace std;
void Low_pass_filter(vector<double>& I, vector<double>& V, vector<double>& T);
string monitor_alarm(double max_current, double max_voltage, double limited_time,
vector <double>& I, vector <double>& V, vector <double>& T);
vector<double> periodic_time_before_filter(vector <double>& S, vector <double>&
T);
double periodic_time_after_filter(vector<double>& S, vector<double>& T);
```

The main purpose

-declare the functions related to the alarm functionality of the program.

Alarm.cpp

-Low pass filter:

void Low_pass_filter(vector<double>& I, vector<double>& V,vector<double>& T)

Purpose:

Filter out noise and unnecessary fluctuations from the Current and Voltage signals by applying a low-pass filter to smoothen them.

Explanation:

```
    vector<double> Filtered_Volt;
    vector<double> Filtered_I;

    const double cutoff_angular_frequency = 2 * 3.14 * 50.0;
    double Tsampling = T[2] - T[1];
```

Firstly, two vectors for the filtered Voltage and Current are declared so that we can store the filtered values inside them before assigning them.

A constant double variable for cut-off angular frequency is then created to store angular frequency of the cutoff frequency, which we'd set to 50 Hz since it's the frequency Egypt's supply is set on.

Another variable for to store a time sample is created which just consists of the difference between any 2 continuous elements of time vector.

```
    Filtered_Volt.resize(V.size());
    Filtered_I.resize(V.size());
    Filtered_Volt[0] = V[0];
    Filtered_I[0] = I[0];
```

The first two lines in the above code snippet are used to set the size of the vectors containing the filtered Voltage and Current readings. Note that both vectors have the same size as they share the same number of readings, so we just used V.size() twice for simplicity's sake.

The other half of the code snippet was made to initialize the first elements of the filtered vectors.

```
    for (int i = 0; i < V.size() - 1; i++)
    {
        Filtered_Volt[i + 1] = (Filtered_Volt[i] + (cutoff_angular_frequency *
Tsampling * V[i + 1])) / (1 + (cutoff_angular_frequency * Tsampling));
        Filtered_I[i + 1] = (Filtered_I[i] + (cutoff_angular_frequency *
Tsampling * I[i + 1])) / (1 + (cutoff_angular_frequency * Tsampling));
    }
    V = Filtered_Volt;
    I = Filtered_I;
```

For-loop is opened from I to the V.size()-1 as to avoid accessing the next element that's outside of bounds. The following formula is then applied on both vectors.

$$\boldsymbol{Filtered}_{\boldsymbol{Signal}[i+1]} = \frac{\boldsymbol{Filtered}_{\boldsymbol{Signal}[i]} + \left(\boldsymbol{cutoff\ Angular\ Frequency} \times \boldsymbol{Time\ Sample} \times \boldsymbol{unfiltered}_{\boldsymbol{Signal}[i+1]}\right)}{1 + (\boldsymbol{cutoff\ Angular\ Frequency} \times \boldsymbol{Time\ Sample})}$$

This equation makes use of the filtered value before the current element and the unfiltered value after the current element. The main purpose of this equation is to give more weight to lower frequency values and decrease that of higher, thus reducing fluctuations and noise.

Finally, after the Filtered Voltage and Current vectors are populated. We replace the Finally, original values of the main Voltage and Current vectors with the filtered value, as the functions passes the main vectors by reference. Therefore, it doesn't create a copy and replaces them instead.

Note:

Originally (at first), we approached the filter with Simple Moving Average equation in mind

```
for (int i = 0; i < V.size(); i++)
{ double sumV = 0; double sumI = 0; int counter = 0;
for (int j = i - 2; j <= i + 2; j++)
{
    if(j < V.size() && j>=0)
    {
        sumV+=V[j]; sumI += I[j]; counter += 1;
    }
}
Filtered_V.push_back(sumV / counter);
Filtered_I.push_back(sumI / counter);
}
```

Basically, we would get the average of the summation of the present value and the 2 values after and before it and set it to the present value. Counter was used as to ensure that the boundary values do not get summed with elements that use index that's outside of the 2 loop boundaries. That would've ensured that all points have the same weightage but was the idea was abandoned for the above one in the main function.

-Periodic time before filter:

vector<double> periodic_time_before_filter(vector <double>& S, vector <double>& T)

purpose:

we need to get the periodic time before filtration to use this function in function alarm

Explanation:

we declare a vector peak time that stores the peak times as a vector of type double and a vector periodic time of type double that stores periodic times as a vector of type double.

we use first for loop to collect peak_times as a vector by using (if statement) that checks if the value of s[i] > s[i+i] && s[i] > s[i_1] . If the condition is true, we store the time at this position to the vector peak_time as (peak_time.push_back(T[i]);).

Now, we have a vector of periodic times and we use it to get the periodic time as a vector of type double to store periodic times as a vector for using this vector in function alarm.

to get periodic time as a vector that stores periodic times, we need to use for loop to collect periodic times in a vector form, to get periodic time at position [i] : get difference between two peak_time and store it in vector periodic_time at position[i].

-Monitor alarm:

string monitor_alarm(double max_current, double max_voltage, double limited_time, vector <double>& I, vector <double>& V, vector <double>& T)

purpose:

First std::ostringstream oss is used to construct strings by combining various data elements, much like writing to a file or console. The resulting string can then be retrieved using the str () method.

Explanation:

we take three arguments of type double : (max_current : the maximum current that the current shouldn't exceeds it, max_voltaget: the maximum voltage that the voltage shouldn't exceeds it , limited_time) and takes two arguments vectors current & voltage of type double.

we declare max_frequency = 50.5 & mini_frequency = 49.5 and then we use for loop to check the current vector by using (if statement) which it has any value exceeds the max_current.

if the statement is true at position [i], we use second for loop to check if the time if it exceeds the limited_time by checking the value of current at position [i+1] > max current, if the statement is true we take a counter called sumi+ = T[l] - T[i_1], if sumi > limited time then it will give me **"warning: the current exceeded the maximum allowed current for more than the allowed time ".**

Same procedure for voltage.

for frequency if either signal doesn't bypass the max & the mini frequency , we use (for loop) with if statement to check if (((1 / periodic_time_v[i]) > max_frequency) || ((1 / periodic_time_v[i]) < mini_frequency))) the condition it means when ((1 / periodic_time_v[i]) it equals frequency of the signal without making a vector to get the frequency as a vector form by using 1/periodic_time_v[i]) is bigger than max_frequency or less than mini_frequency then the condition is true it will give me " warning: the frequency of (signal name) bypassed the limited frequency".

-Periodic time after filter:

double periodic_time_after_filter(vector<double>& S, vector<double>& T)

Purpose:

Get the periodic time of the new filtered values.

Explanation:

First, we declare a vector of type double to store peak_time s as a vector. We make passing by reference of vector s to store the peak times in this vector.

we make for loop to collect peak-time s and use if statement: s[i] > s[i+1] && s[i] >s[i_1] which checks if the value of current at postion [i] is bigger than the value of s at positions [i +1] and [i_1].

f the statement is true we add the value of the time at position [i] as a vector T[i] to the vector periodic_time.
Finally, we put the periodic_time is equal to the difference between any two consecutive positions like (peak_time [i+1] - peak_time [i])

we take the periodic time for example difference between peak time at positions peak_time [4] - peak_time [3].

Calc.h

```
#pragma once
#include<iostream>
#include<vector>
#include<fstream>
#include<cassert>
#include<string>
#include<sstream>
using namespace std;
double energycalc(vector<double>& I, vector<double>& V, vector<double>& T);
double calculateTariff(double totenergy, vector<double>& T);
double calcTaxes(double totenergy, vector<double>& T);
double PowerCalc(vector <double>& I, vector <double>& V, vector <double>& T);
double PowerFactorCalc(vector <double>& I, vector <double>& V, vector <double>&
T);
```

Main purpose

-Declare functions to calculate energy, total cost and power.

Calc.cpp

-Energy Calculation:

The energycalc function calculates the total energy consumption based on sampled current, voltage, and time data. The process involves:

1. Time Sampling Intervals:

```
double deltaT = (T[i] - T[i - 1]) / (3600.0); //Time interval in hours
```

For each time interval, the duration is calculated in hours by converting seconds to hours.

2. Power Calculation:
The power at each time step is computed using

```
double power = V[i] * I[i];
```

3. Energy Calculation:
The energy consumed during each interval is determined using

```
double energy = power * deltaT * 0.001;
```

converting the result to kilowatt hours (kWh) [multiplying by 0.001 to convert it to Kilo].

4. Accumulation:

The total energy consumption is accumulated over all intervals and returned as the result using for loop.

```
return totenergy; // Return total energy consumed
```

This function was simple and straightforward, I didn’t face any issues.

The function returned the total energy in kilowatt hours [even though it was easy it is effective].

-Tariff Calculation:

The calculateTariff function was the most challenging part (for me). It calculates the total cost of energy consumption based on predefined tariff that based on consumption levels (شرائح الاستهلاك).

In the first, we wrote a version of the function designed for high-voltage systems, including peak and off-peak calculations. However, this was unsuitable for household energy tariff (they do not depend on peak and off-peak hours) so the function needed adjustments.

The energy data was initially in kWh but had to be converted to monthly consumption. This involved calculating the total time from the T vector and dividing it by the number of seconds in a 30-day month. To convert the energy in kWh/month, which is essential for comparing against tariff [to determine the suitable شريحة] .

We wrote that conversion using variable S to make it more readable and reusable in the code. We struggled to figure out the conversion units at first, but always converting problem into smaller parts help to solve it.

First, we calculated the total time by subtracting the first timestamp from the last, then we converted the result to months.

The variable s represents the time in months, calculated by converting the total time from seconds to months.

30*60*60*24 represents the total number of seconds in a month.

```
double totaltime = T[T.size() - 1] - T[0];
double s = totaltime / (30 * 60 * 60 * 24);
```

Then, we converted the energy from kWh to kWh/month.

[This line calculated the number of seconds in a month and multiply it by the total energy that calculated in kWh].

```
double Menergy = (totenergy / totaltime) * 30 * 60 * 60 * 24;
```

After that we turned it again to kwh because the cost was in Pt / kWh not Pt/(kWh/month) using the variable s

```
double s = totaltime / (30 * 60 * 60 * 24);
```

Then it was just for loop to calculate the total cost then added taxes to it [I used the tax function in tariff calculation function]

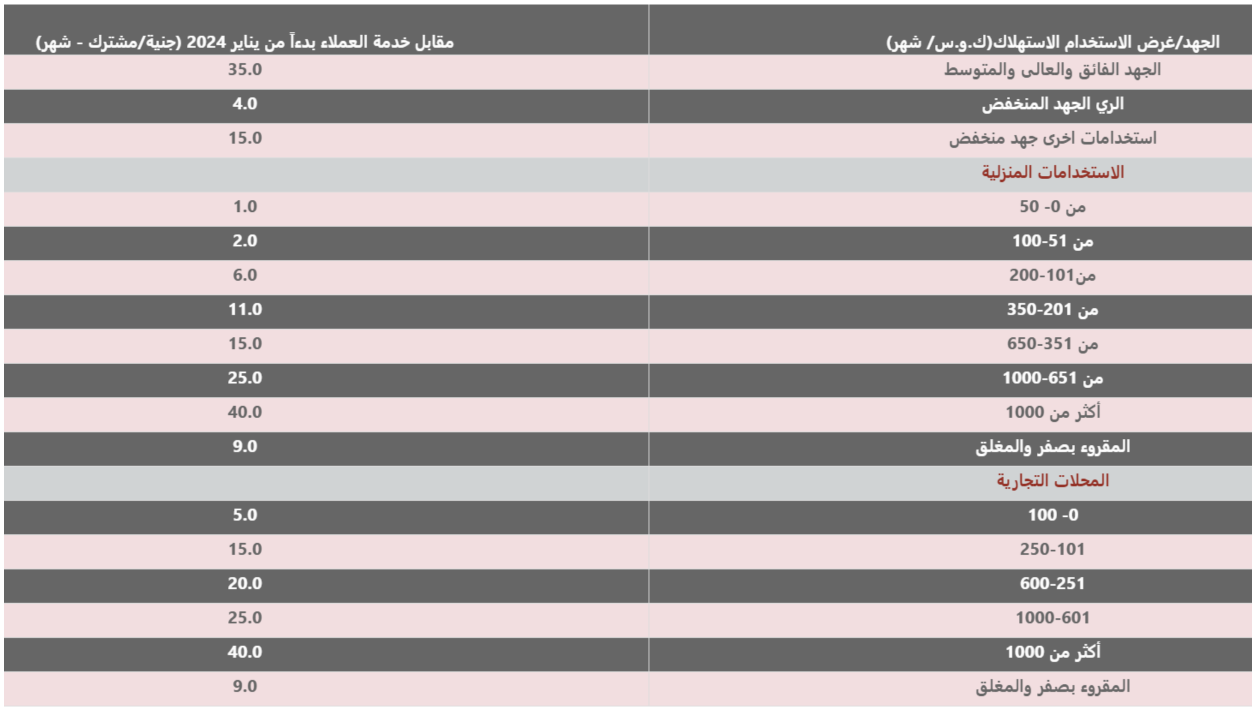
```
double tax = calcTaxes(totenergy,T);
```

-Tax Function:

calcTaxes function calculates the taxes based on kWh/month consumption so again here we converted energy from kWh to kWh/month [it was straightforward as energy calculation function after we solved the problem of conversion units].

```
double totaltime = T[T.size() - 1] - T[0];
totenergy = (totenergy/totaltime)*30*60*60*24;
```

We calculated it using for loop based on this table



مقابل خدمة العملاء بدءاً من يناير 2024 (جنية/مشترك - شهر)	الجهد/غرض الاستخدام الاستهلاك(ك.و.س/ شهر)
35.0	الجهد الفائق والعالى والمتوسط
4.0	الري الجهد المنخفض
15.0	استخدامات اخرى جهد منخفض
	الاستخدامات المنزلية
1.0	من 0- 50
2.0	من 51-100
6.0	من101-200
11.0	من 201-350
15.0	من 351-650
25.0	من 651-1000
40.0	أكثر من 1000
9.0	المقروء بصفر والمغلق
	المحلات التجارية
5.0	0- 100
15.0	101-250
20.0	251-600
25.0	601-1000
40.0	أكثر من 1000
9.0	المقروء بصفر والمغلق

Those set of functions effectively calculate energy consumption, tariff and taxes for household in Egypt.

-The consumed power:

This function is simple compared to the other functions, so as we know the consumed power is the average power, and to calculate the average there are a lot of forms, one of the forms we were thinking to use was these:

$$P_{\text{avg}} = \frac{\sum_{i=1}^{N} P_i \cdot \Delta t}{T_{\text{total}}}$$

This formula wasn't the easiest, but it was very reliable .To simplify this formula, we can Separate it to two parts:

A) power calculation:

```
for (int i = 0; i < num; i++) {
        power.push_back(I.at(i) * V.at(i));
    }
```

To calculate the instantaneous power for any time you multiply the current by the voltage at that time, so we made a new vector called power to store the instantaneous for the whole load, we made a for loop to multiply the current and the voltage and store the result at the new vector.

B) Average power calculation:

Now we have a vector of powers, we can use that vector, so first we get the time interval by simply get the difference between the time of the next power and the current power.

```
    for (int i = 0; i < T.size() - 1; i++) {
        sump += power.at(i) * (T.at(i + 1) - T.at(i));

    }
    double avg_power = sump / totaltime;
```

At the beginning of this function, we calculate the total time

```
double totaltime = T.at(num - 1) - T.at(0);
```

Finally, we return this power divided by the total time to the user.

```
    double avg_power = sump / totaltime;
    return avg_power;
```

-The Power factor:

As we know to calculate the power factor, we divide the real power by the apparent power.

So, this function contents of three parts:

A) real power:

As we know the real power consumed in any load is the average power, so after we wrote a function in the previos part to calculate the average part, we can simply save time and call that function instead of rewriting a now code to calculate it again.

```
    double realpower = PowerCalc(I, V, T);
```

B) apparent power:

First, we need to calculate the rms values to be able to calculate the apparent power.

We use a for loop to calculate the RMS values.

```
    double sumv = 0;
    double sumi = 0;
    for (int i = 0; i < I.size(); i++) {
        sumv = sumv + V.at(i) * V.at(i);
        sumi = sumi + I.at(i) * I.at(i);
    }
    double Vrms = sqrt(sumv / V.size());
    double Irms = sqrt(sumi / I.size());

    double app_power = Vrms * Irms;
```

The for loop calculate the summation of the current and the voltage squared, then it takes the square root for that value.

Finally, we can multiply the RMS values of the current and voltage the get the apparent power.

C) power factor:

Last but not least, after we calculate both the real and apparent power, we can simply divide the real power by the apparent power to get the power factor.

```
    double powerfactor = realpower / app_power;
    return powerfactor;
```

MyForm.h & MyForm.cpp

-Purpose:

Design and implement a GUI to simplifies user interaction with the house meter program by organizing operations into buttons and input fields (to enable alarm monitoring, data calculations, and file handling operations directly through interface).

GUI

In the gui we started by deciding the design first. we came up with 5 text boxes and 3 button at the starting by the text boxes we created two main text boxes one is used to enter the input file name so that we take it ti use it in functions and one to type in it the output file name one of the problems we face is our function take an (std string) as an argument where in the gui the textboxes give us a (system::String^) so we had to convert it into std string we used the (msclr/marshal_cppstd.h) library to do that we first created a context then used it to convert (system::String^) to string and store it in a variable named file to use it in the function (readfile) so then we created two buttons one for the normal calculations to get our data and other one to set and get the load details in a class .

Button 1:

So we started by calling the (readfile) function So i explained earlier how we used it already then we started by calling the rest of the functions to get our output and store it in variables we then converted these variables to strings and stored it all in one string then we created a message box to show us that string that we saved in it all the output and a question asking the user if he wants to save the output data in a file or not this is where the output file textbox comes to the picture if the user wants to save the output data in a file he should first enter the name of the outputfile in the textbox so that the program search for tgat file and save the output in it or if there isnt a one created already the program would create one with that name and save the data in it

Button 2:

Basically, just like the first button but we used here the class we created a class first then used set and get functions to set the load details then get it and convert it to strings then store them in just one string and then again outputting these details in a message box for the user

Button 3:

Finally the alarm button this button has 3 textboxes for itself in order to enter the maximum limited current and voltage and the time limit of them exceeding the limit so we take these values from the textboxes and convert it to doubles then pass it to the alarm function as arguments so that the alarm function uses them then we output a message in message box saying if the one of the values exceed its limit one of the problems we faced us that thus function has cout statements for example a cout statement if the voltage exceeds a certain limit etc... so we didn't know how to take these statements specifically and store it in a string variable to use it in a message box so we had to create a stringstream and store in it every cout statement that comes out of our function then return that stream then we store it in a variable named message we then use .c_str() to convert that string to a pointer to a null terminated string (c style string) the we construct a system::String^ using this c style string and put it in the message box we used also gcnew for automatic dynamic memory allocation.

Background:

To set the Background color, the following line of code was added under the MyForm comment, where the form settings are located inside the MyForm.h file.

```
    this->BackColor = System::Drawing::Color::LightBlue;
```

To illustrate:

```
    this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
    this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
    this->BackColor = System::Drawing::Color::LightBlue;
```

The line accesses member called BackColor inside the form's class and we assign to it LightBlue, the background color, which belongs to System::Drawing::Color::

Logo:

```
// pictureBox1
this->pictureBox1->Anchor =
static_cast<System::Windows::Forms::AnchorStyles>((System::Windo
ws::Forms::AnchorStyles::Bottom |
System::Windows::Forms::AnchorStyles::Right));
this->pictureBox1->Image =
(cli::safe_cast<System::Drawing::Image^>(resources-
>GetObject(L"pictureBox1.Image")));
this->pictureBox1->Location = System::Drawing::Point(847, 438);
this->pictureBox1->Name = L"pictureBox1";
this->pictureBox1->Size = System::Drawing::Size(131, 109);
this->pictureBox1->SizeMode =
System::Windows::Forms::PictureBoxSizeMode::Zoom;
this->pictureBox1->TabIndex = 29;
```

```
this->pictureBox1->TabStop = false;
this->pictureBox1->Click += gcnew System::EventHandler(this,
&MyForm::pictureBox1_Click)
```

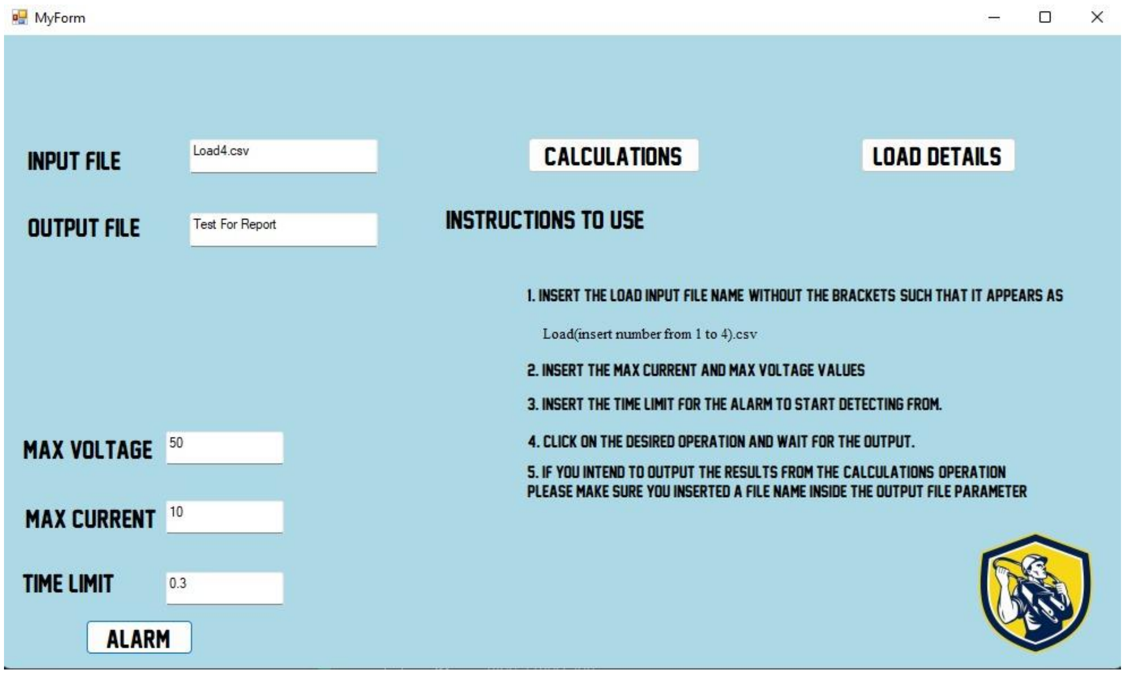
Setting the logo was mostly through the form design settings, which translated to the above piece of code in the header file, including the picture settings and properties such as size, name of picture box and anchoring of the picture.

The Problem:

Initially when we inserted the image, an error would occur where the build couldn't be completed, halting our project. After a bunch of research into the problem online, we discovered that we had to embed a resource file, specifically inside the project's .vcxproj text file, which resolved the problem smoothly.

The output when running the program:

The user will enter the name of input file ,output file and write the conditions he/she wants for the alarm (Max voltage, Max current and Time limit).



MyForm

INPUT FILE: Load4.csv

OUTPUT FILE: Test For Report

MAX VOLTAGE: 50

MAX CURRENT: 10

TIME LIMIT: 0.3

INSTRUCTIONS TO USE

1. INSERT THE LOAD INPUT FILE NAME WITHOUT THE BRACKETS SUCH THAT IT APPEARS AS

   Load(insert number from 1 to 4).csv

2. INSERT THE MAX CURRENT AND MAX VOLTAGE VALUES

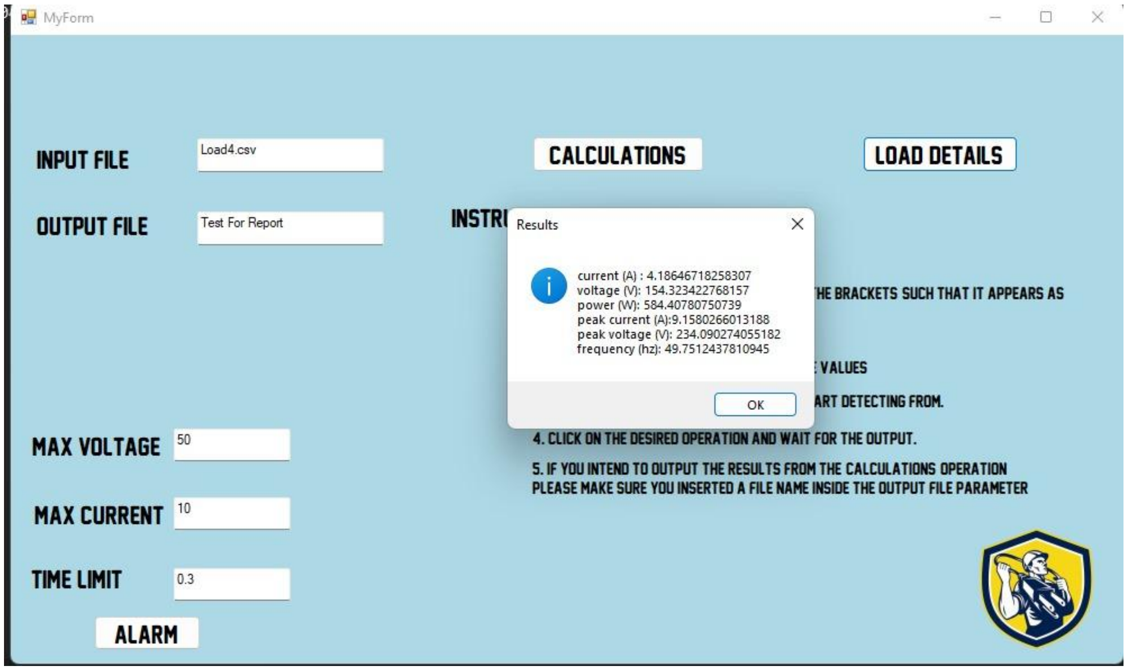
3. INSERT THE TIME LIMIT FOR THE ALARM TO START DETECTING FROM.

4. CLICK ON THE DESIRED OPERATION AND WAIT FOR THE OUTPUT.

5. IF YOU INTEND TO OUTPUT THE RESULTS FROM THE CALCULATIONS OPERATION PLEASE MAKE SURE YOU INSERTED A FILE NAME INSIDE THE OUTPUT FILE PARAMETER

Load Details:

The output when the user clicks on load details button:



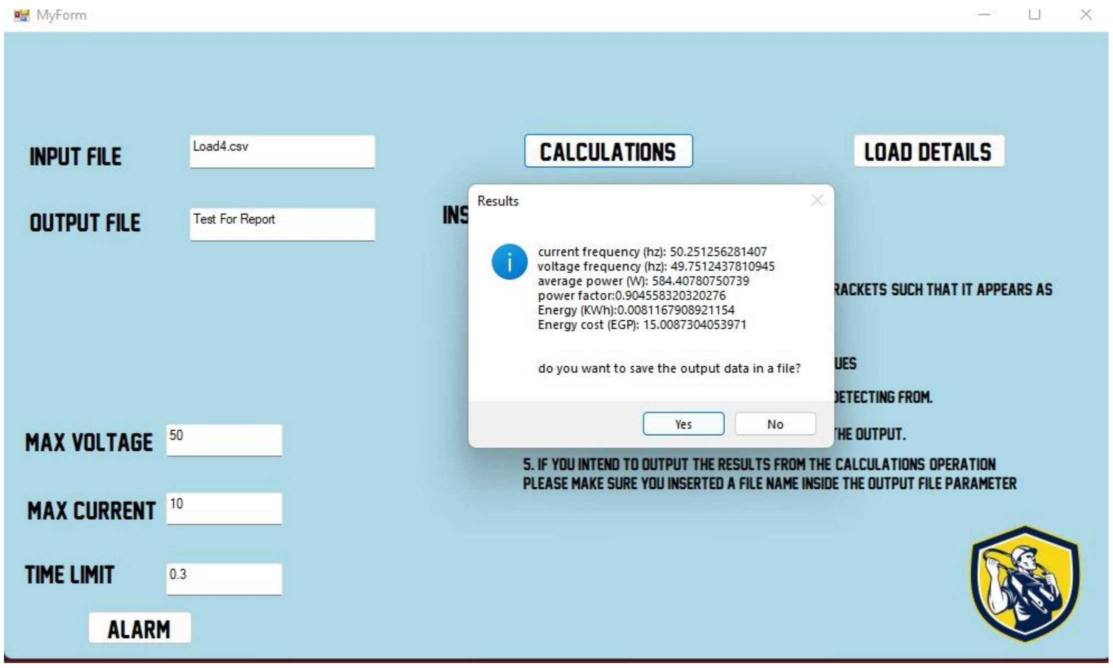
Results

```
current (A) : 4.18646718258307
voltage (V): 154.323422768157
power (W): 584.40780750739
peak current (A):9.1580266013188
peak voltage (V): 234.090274055182
frequency (hz): 49.7512437810945
```

Calculations:

The output when the user clicks on calculations button

The cost included the taxes:



MyForm

INPUT FILE: Load4.csv

OUTPUT FILE: Test For Report

MAX VOLTAGE: 50

MAX CURRENT: 10

TIME LIMIT: 0.3

Results

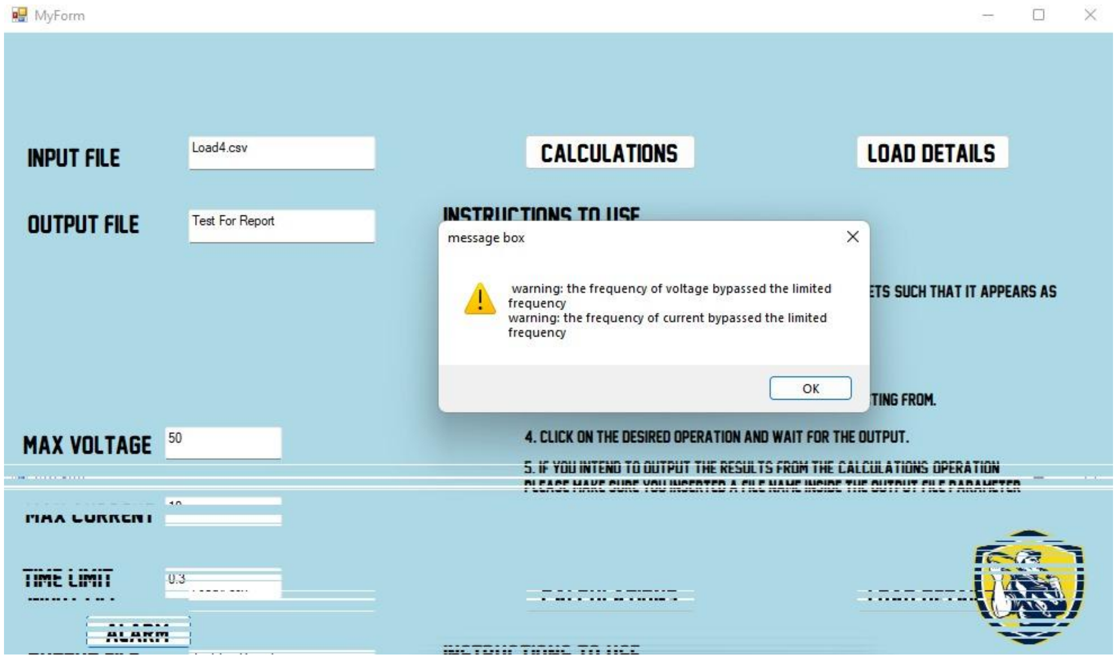
```
current frequency (hz): 50.251256281407
voltage frequency (hz): 49.7512437810945
average power (W): 584.40780750739
power factor:0.904558320320276
Energy (KWh):0.0081167908921154
Energy cost (EGP): 15.0087304053971

do you want to save the output data in a file?
```

5. IF YOU INTEND TO OUTPUT THE RESULTS FROM THE CALCULATIONS OPERATION PLEASE MAKE SURE YOU INSERTED A FILE NAME INSIDE THE OUTPUT FILE PARAMETER

Alarm:

The output when the user clicks on alarm button:



MyForm

INPUT FILE: Load4.csv

OUTPUT FILE: Test For Report

MAX VOLTAGE: 50

MAX CURRENT: 10

TIME LIMIT: 0.3

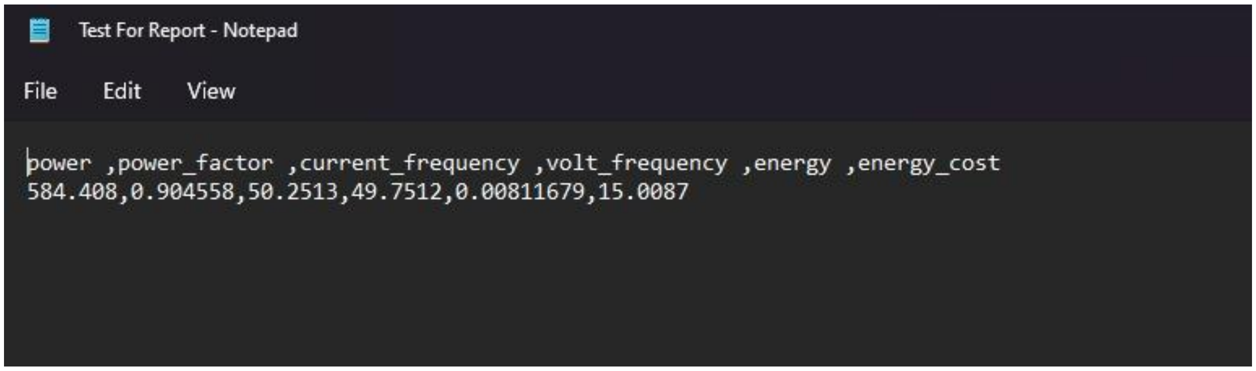
message box

```
warning: the frequency of voltage bypassed the limited
frequency
warning: the frequency of current bypassed the limited
frequency
```

4. CLICK ON THE DESIRED OPERATION AND WAIT FOR THE OUTPUT.

5. IF YOU INTEND TO OUTPUT THE RESULTS FROM THE CALCULATIONS OPERATION PLEASE MAKE SURE YOU INSERTED A FILE NAME INSIDE THE OUTPUT FILE PARAMETER

The output file:



```
power ,power_factor ,current_frequency ,volt_frequency ,energy ,energy_cost
584.408,0.904558,50.2513,49.7512,0.00811679,15.0087
```