03 - Formal Introduction to OOP

Post-Lab Exercises

Exercise 1

Study the contents of the `EmonLib` library, which is a part of the Open Energy Monitor project (www.openenergymonitor.org). This library consists of a class with certain data members and member functions. Go to the library repository at (https://github.com/openenergymonitor/EmonLib) and download the library files. The main library interface and function definitions are described in `EmonLib.h` and `EmonLib.cpp` files respectively.

Study this class interface and implementation and describe each data member and member function in detail.

Some functions in this class belong to the Arduino microcontroller library (e.g. `analogRead( )`, `millis( )`, etc.), whose description is available in the Arduino library references at (wwww.arduino.cc). You will need to apply your knowledge of AC electrical power and energy in order to make sense of this class

---

Member Variables

Variable Name	Access Specifier	Data Type	Description
`realPower`	public	`double`	Real power consumed during one wave cycle. Reset to zero at the end of each loop. Used in power factor calculation.
`apparentPower`	public	`double`	Product of RMS current and voltage. Used in power factor calculation.
`powerFactor`	public	`double`	Ratio of real power to apparent power, and the overall efficiency of the system.
`Vrms`	public	`double`	RMS Voltage. Used in power calculations.
`Irms`	public	`double`	RMS Current. Used in power calculations.
`inPinV`	private	`unsigned int`	Stores pin number of voltage input pin. Used as argument for `analogRead()` during measurement.
`inPinI`	private	`unsigned int`	Stores pin number of current input pin. Used as argument for `analogRead()` during measurement.
`VCAL`	private	`double`	Calibration coefficient for voltage measurements. Defined differently for each Arduino board.
`ICAL`	private	`double`	Calibration coefficient for current measurements.
`PHASECAL`	private	`double`	Calibration coefficient for voltage phase shift measurements.
`sampleV`	private	`int`	Stores analog voltage input from `inPinV` for further calculations in main loop.

Variable Name	Access Specifier	Data Type	Description
sampleI	private	int	Stores analog current input from inPinI for further calculations in main loop.
lastFilteredV	private	double	Set to previous loop's filteredV at the beginning of each loop. Used to account for phase differences and their effect on voltage.
filteredV	private	double	Removes the offset from the sampled voltage, so a strictly AC value centred about the internal reference voltage is stored for future calculations.
offsetV	private	double	DC voltage offset of the Arduino board.
offsetI	private	double	DC current offset of the Arduino board.
phaseShiftedV	private	double	Tunes voltage reading based on phase differences between filteredV and lastFilteredV.
sqV	private	double	Used in Vrms calculation as intermediate variable.
sumV	private	double	Used in Vrms calculation as intermediate variable.
sqI	private	double	Used in Irms calculation as intermediate variable.
sumI	private	double	Used in Irms calculation as intermediate variable.
instP	private	double	Intermediate variable for post-loop calculations.
sumP	private	double	Intermediate variable for post-loop calculations.
startV	private	int	Instantaneous voltage at start of sampling duration.
sumP	private	double	Intermediate variable for post-loop calculations.
startV	private	int	Instantaneous voltage at start of sampling duration.
lastVCross	private	bool	Used to check how many times the voltage has crossed voltage at start of sampling window.
VCross	private	bool	Used to check how many times the voltage has crossed voltage at start of sampling window. Indirectly helps keep track of the number of wavelengths sampled.
checkVCross	private	bool	True if the sample voltage crosses the starting voltage. Used in conjunction with the lastVCross variable to identify new wave cycles.

Member Functions

Function Name	Access Specifier	Parameters	Description
voltage	public	_inPinV, _VCAL, _PHASECAL	Sets the pin which will be used for voltage measurement. Also initialises calibration constants for phase and voltage measurements, as well as the DC offset voltage.
current	public	_inPinI, _ICAL	Sets the pin which will be used for current measurement. Also initialises calibration constants for current measurement and the Arduino board's offset current.
voltageTX	public	_VCAL, _PHASECAL	Alternative initialiser for voltage measurement pin (and other quantities). To be used if the Arduino board's serial communication pins are used for measurement instead of analog pins. Does not take an _inPinV parameter because in this configuration, the TX pin (2) is used for input by default.
currentTX	public	_channel, _ICAL	Alternative initialiser for current measurement pin (and other quantities). To be used if the Arduino board's serial communication pins are used for measurement instead of analog pins. Does not take an _inPin argument. Sets it by default based on the channel argument (what is channel?).
calcVI	public	crossings, timeout	The Energy Monitor's equivalent of 'main'. First waits for the input voltage and current sinusoids to be close to the reference voltage. crossing and timeout are used to determine the duration of the sampling process. Then samples voltage and current over specified time period. Proceeds to remove their DC offsets to derive purely AC values. Performs RMS calculations on these values in a step-by-step process (square -> mean -> root). Then performs post-loop calculations for instantaneous, average, real powers and power factors before resetting sum of PIV accumulator variables to 0.
calIrms	private	NUMBER_OF_SAMPLES	Samples the instantaneous current, filters the DC offset, and uses the number of samples taken along with calibration constants to return an RMS current for PIV calculations (Why is this a separate function when Vrms is not?)
serialprint	private	–	Outputs results of post-loop power calculations to the Arduino serial monitor, which in turn can display them on a screen or send them to some other output device.
readVcc	private	–	Returns the supply voltage for the current Arduino board arrangement. Uses multiplexers for some reason. Why are you doing this to us, Sir Hassan?

Exercise 2

Separate the interface and implementation of the two classes present in the given programme `exprogram.cpp`. Also describe the following aspects of the class: writing a destructor, the `delete` keyword, and its opposite `new` keyword. Also, describe how this class is similar to the high order array class we wrote in lab.

Person - Header

```
#ifndef Person_h
#define Person_h

class Person
{
private:
    string      firstName;
    string      lastName;
    int         age;
public:
    Person(string last, string first, int a);
    void        displayPerson();
    string      getLast();
}
```

Person - Implementation

```
#include "Person.h"

Person::Person(string last, string first, int a)
{
    lastName = last;
    firstName = first;
    age = a;
}

void Person::displayPerson()
{
    cout << "   Last name: " << lastName;
    cout << ", First name: " << firstName;
    cout << ", Age: " << age << endl;
}

string Person::getLast()
{ return lastName; }
```

ClassDataArray - Header

```
#ifndef DataArray_h
#define DataArray_h
#include "Person.h"
#include <vector>

class ClassDataArray
{
private:
    vector<Person*> v;
    int             nElems;
public:
    ClassDataArray(int max);
    ~ClassDataArray();
    Person* find(string searchName);
    void insert(string last, string first, int age);
    bool remove(string searchName);
    void displayA();
};
#endif /* DataArray_h */
```

ClassDataArray - Implementation

```
#include "DataArray.h"
ClassDataArray::ClassDataArray(int max) : nElems(0)
{ v.resize(max); }


ClassDataArray::~ClassDataArray()
{
    for (int j = 0; j < nElems; j++)
        delete v[j];
}

Person* ClassDataArray::find(string searchName)
{
    int j;
    for (j = 0; j < nElems; j++)
        if (v[j]->getLast() == searchName)
            break;
    if (j == nElems)
        return NULL;
    else
        return v[j];
}

void ClassDataArray::insert(string last, string first, int age)
{
    v[nElems] = new Person(last, first, age);
    nElems++;
}
```

```
bool ClassDataArray::remove(string searchName)
{
    int j;
    for (j = 0; j < nElems; j++)
        if (v[j]->getLast() == searchName)
            break;
    if (j == nElems)
        return false;
    else
    {
        delete v[j];
        for (int k = j; k < nElems; k++)
            v[k] = v[k+1];
        nElems--;
        return true;
    }//end else
}//end remove

void ClassDataArray::displayA()
{
    for (int j = 0; j < nElems; j++)
        v[j]->displayPerson();
}   //end displayA();Array
```

Main

```
#include "Person.h"
#include "DataArray.h"

int main()
{
    int maxSize = 100;                  //array size
    ClassDataArray arr(maxSize);        //array

    arr.insert("Evans", "Patty", 24);   //insert 10 items
    arr.insert("Smith", "Lorraine", 37);
    arr.insert("Yee", "Tom", 43);
    arr.insert("Adams", "Henry", 63);
    arr.insert("Hashimoto", "Sato", 21);
    arr.insert("Stimson", "Henry", 29);
    arr.insert("Velasquez", "Jose", 72);
    arr.insert("Lamarque", "Henry", 54);
    arr.insert("Vang", "Minh", 22);
    arr.insert("Creswell", "Lucinda", 18);

    arr.displayA();                     //display items

    string searchKey = "Stimson";       //search for item
    cout << "Searching for Stimson" << endl;
    Person* found;
    found=arr.find(searchKey);
```

```
if(found != NULL)
    {
        cout << "   Found ";
        found->displayPerson();
    }
    else
        cout << "   Can't find " << searchKey << endl;

    cout << "Deleting Smith, Yee, and Creswell" << endl;
    arr.remove("Smith");                //delete 3 items
    arr.remove("Yee");
    arr.remove("Creswell");

    arr.displayA();                     //display items again
    return 0;
}  //end main()
```

Output

```
   Last name: Evans, First name: Patty, Age: 24
   Last name: Smith, First name: Lorraine, Age: 37
   Last name: Yee, First name: Tom, Age: 43
   Last name: Adams, First name: Henry, Age: 63
   Last name: Hashimoto, First name: Sato, Age: 21
   Last name: Stimson, First name: Henry, Age: 29
   Last name: Velasquez, First name: Jose, Age: 72
   Last name: Lamarque, First name: Henry, Age: 54
   Last name: Vang, First name: Minh, Age: 22
   Last name: Creswell, First name: Lucinda, Age: 18
Searching for Stimson
   Found    Last name: Stimson, First name: Henry, Age: 29
Deleting Smith, Yee, and Creswell
   Last name: Evans, First name: Patty, Age: 24
   Last name: Adams, First name: Henry, Age: 63
   Last name: Hashimoto, First name: Sato, Age: 21
   Last name: Stimson, First name: Henry, Age: 29
   Last name: Velasquez, First name: Jose, Age: 72
   Last name: Lamarque, First name: Henry, Age: 54
   Last name: Vang, First name: Minh, Age: 22
Program ended with exit code: 0
```

Explanations

Destructors

Just as a constructor is used to reserve memory for an object of a class, a destructor is used to free memory occupied by objects. The destructor is a special function that is invoked every time an automatic variable or object goes ‘out of scope’ (no references to the object can be found in the existing programme). Destructors thus ensure a programme is efficient and uses as little memory as possible for only as long as is necessary. Destructors in C++ are written using the class name prefaced with the tilde (~) symbol.

‘new’ and ‘delete’

The new keyword is used to reserve memory for a reference-type object at runtime. The delete object is used to free memory occupied by objects created with the ‘new’ keyword back to the freestore.

In This Class

It was necessary to write a destructor for this class because it uses a vector of references to Person objects as a member variable. Each of these Person objects is created dynamically using the ‘new’ keyword, which means each Person object in the vector of each instance of the ClassDataArray class occupies memory in the freestore. Since these Person objects are not automatic variables, they will not be deleted when they go out of scope. To return the memory occupied by vector of Person variables back to the freestore, each of them needs to be deallocated explicitly with the delete keyword.

This is exactly what the destructor for ClassDataArray does. It parses the entire vector of (references to) Person objects and deallocates them with the delete keyword to return their memory to the freestore. This destructor is invoked implicity as soon as no further references to a ClassDataArray object can be found in main.cpp.