Program #4 – Implement a sophisticated class for mathematical vectors – Due: **Friday, April 4**

Average effort (implement the CANONICAL SET):
- **default CTOR**
- memory-wise **DTOR**
- a **copy CTOR**
- overload **assignment (=)** (Note: if the left-hand-side vector is smaller than the right-hand-side vector you must: make the lhs vector grow() & warn the user that you did this);

Above average effort
- overload **binary +/-** (vectors need *not* be the same size)
- overload all **relational op's** -- `<`, `>`, `<=`, etc (see NOTEs below for how to implement relations);
- getMagnitude()
- overloaded **output (<<)**, e.g., cout << A;
- overloaded **input (>>)**, e.g., cin >> A;

Superior effort
- dot product (`A*B`) and/or cross product (`A^B`);
- **vector** times **scalar**, e.g., `A*4`
- solve a "real" vector problem (in `main()`) that uses your class

Your class *must* be well documented. Include a `testMain.cpp` that documents *how* you are testing (showing me!) the functionality of each addition to the class. You *must* include a README file that explains the state of your software (e.g., this works, this doesn't, etc).

NOTE: Let's agree to **use the magnitude of a vector** when implementing the relational operators. If **v** is a Vector with n entries, the *magnitude* of **v** (generally denoted as **|v|**) is the square root of the sum of squared entries in **v**:

$$\text{magnitude of vector } \mathbf{v} \;=\; |\mathbf{v}| \;=\; \sqrt{\sum_{i=0}^{n-1} v[i]^2}$$

The `operator==` method really should check the *magnitude* of one vector with the *magnitude* of another vector, but more correctly, if the magnitudes are "close enough", right? For example, the two vectors v1 and v2 are not the same size (dimensions) but they have the same magnitude, thus v1 == v2 is true:

four-dimensional vector, v1 = (2, 2, 2, 2) = 4
two-dimensional vector, v2 = (4, 0) = 4

so, `v1 == v2` is `true` in this example above. NOTE: you could implement the `getMagnitude()` method as a <u>private</u> method in your `class Vect`, but asking for the magnitude of a vector is probably something the client (main) might want to do also, thus, implement `magnitude()` as a public method.

Additional points:

(1) our components of a `Vect`(or) should be (dynamically allocated) Real numbers:

```
private:
     double* pVect;
```

(2) you need only three CTORS:

(a) default (no size)

(b) you know the size ahead of time (e.g., 2D or 3D or ...)

e.g.:
```
// A is a 3-D vector, <x,y,z>
Vect  A(3);
```

(c) a copy CTOR

(3) you MUST write `operator[]`

this will let the `main()` client-programmer change the x, y, and z components, e.g.,

```
// change A to have components: x=3, y=2, z=5
A[0] = 3;   // location [0] means x-axis
A[1] = 2;   // location [1] means y-axis
A[2] = 5;   // location [2] means z-axis
```

Note: `A[0] = 3` *really* makes the call: `A.operator[]( 3 );`
which calls the method to set: `this->pVect[0] = 3`

(4) *yes*, we are assuming that we can add/subtract two vectors of unequal dimensions; note that we *always* assume that `[0]` is the x-dimension, `[1]` is the y-dimension, etc. thus, adding a 2-D and a 3-D vector assumes that the "mapping" is handled such that a 2-D vector "gets" a third-z-component with value zero(0) *before* the addition to the 3-D vector occurs