CS6771 Assignment Two 2017

Euclidean Vector Class Library

Due: 11:59pm Tuesday 5 September 2017

Worth: 10%

1. Aims:

- Familiarity with C++ Classes
- Constructors
- Destructor
- Uniform initialisation
- Copy Control
- Function Overloading
- Operator Overloading
- Friends
- (Simple) Dynamic Memory Management
- Seperation of Interface from Implementation

Write a Euclidean Vector Class Library in C++, with its interface given in `EuclideanVector.h` and its implementation in `EuclideanVector.cpp`. For assessment purposes, your `EuclideanVector.cpp` will be compiled with a series of assessment test cases that #include your EuclideanVector.h.

Euclidean Vectors are commonly used in physics to represent directed quantities. Your Euclidean Vector class should be dimension-agnostic, that is, it should be able to be constructed and operate on data of arbitrary dimensionality. **The magnitude in each dimension will always be a double.** For example:

```
evec::EuclideanVector a(1);        // a Euclidean Vector in 1 dimension, with default
magnitude 0.0.
evec::EuclideanVector b(2,4.0);    // a Euclidean Vector in 2 dimensions with magnitude 4.0 in
both dimensions

std::list<double> l;
l.push_back(5.0);
l.push_back(6.5);
l.push_back(7.0);
evec::EuclideanVector c(l.begin(),l.end());     // a Euclidean Vector in 3 dimensions
constructed from a list of magnitudes
```

2. International Representation

You absolutely must use dynamically allocated memory to store a Euclidean vector where appropriate. Use of an STL container (except obviously in type conversion) will result in a zero. Please make sure you use `new` and `delete` Therefore, the compiler-generated copy-control members will not provide the correct semantics. You must provide your own implementations for these copy-control member functions.

3. Constructors

Your design should allow Euclidean vectors to be defined in the following ways:

Constructor	Description and Hints	Examples

Constructor	A constructor that takes the number of dimensions (as an unsigned int) but no magnitudes, sets the magnitude in each dimension as 0.0. Hint: you may want to make this a delegating constructor to the next constructor below.
	This is the default constructor, with the default value being 1.
		`(1) EuclideanVector a(1);`
		`(2) unsigned int i {3};`
		`EuclideanVector b(i);`
		`(3) EuclidenVector c; // or c{}`
		`// same as EuclidenVector c(1);`

Constructor	A constructor that takes the number of dimensions (as an unsigned int) and initialises the magnitude in each dimension as the second argument (a double)	`(1) EuclideanVector a(2,4.0);`
		`(2) unsigned int x {3};`
		`double y {3.24};`
		`EuclideanVector b(x,y);`

Constructor	A constructor (or constructors) that takes the start and end of an iterator and works out the required dimensions, and sets the magnitude in each dimension according to the iterated values. The iterators will be from std::vector or std::list. Hint: a function template may help. Hint 2: the compiler prefers calling normal functions over templated functions, even if it's an exact match for the template	`(1) std::list l;`
		`EuclideanVector a{l.begin(),l.end()};`
		`(2) std::vector v;`
		`EuclideanVector b{v.begin(),v.end()};`

Constructor	An initialiser-list constructor that creates an Euclidean vector from a list of double values. See Pages 220 -- 224 of the text.	`EuclideanVector a {1,2,3,4};`

Constructor	A copy constructor	`EuclideanVector aCopy{a};`
Constructor	A move constructor	`EuclideanVector aMove{std::move(a)};`

For all constructors, you may assume that the arguments supplied by the user are correct (as is the case for the STL containers).

Please make sure that your constructors work correctly. Otherwise, we have no way to construct any Eucliden vector to test your solution.

3. Destructor

Due to the use of dynamic memory allocation in your constructors, you must provide a destructor that deallocates the memory acquired by the constructors. You should ensure that your implementation does not leak memory. You will be penalised for an improperly implemented destructor.
Please use the compiler flag ``-fsanitize=address" to enable AddressSanitizer , a fast memory error detector. Memory access instructions are instrumented to detect out-of-bounds and use-after-free bugs.

4. Operations

Your design must support the following public (member) operations performed on Euclidean vectors:

Operation	Description and Hints	Examples
Copy Assignment	A copy assignment operator overload	`EuclideanVector a;` `a = b;`
Move Assignment	A move assignment operator	`EuclideanVector a;` `a = std::move(b);`
getNumDimensions()	Returns an unsigned int containing the number of dimensions in a particular vector	`a.getNumDimensions();`
get(unsigned int)	Returns a double, the value of the magnitude in the dimension given as the function parameter	`a.get(1);`
getEuclideanNorm()	Returns the Euclidean norm of the vector as a double. The Euclidean norm is the square root of the sum of the squares of the magnitudes in each dimension. E.g, for the vector [1 2 3] the Euclidean norm is sqrt(1*1 + 2*2 + 3*3) = 3.74. Hint: the Euclidean norm should only be calculated when required and ideally should be cached if required again. Hint 2: mutable data members from lecture 3.1 may come in handy	`a.getEuclideanNorm();`
createUnitVector()	Returns an Euclidean vector that is the unit vector of *this vector. The magnitude for each dimension in the unit vector is the original vector's magnitude divided by the Euclidean norm.	`a.createUnitVector();`
operator[]	Allows to get and set the value in a given dimension of the Euclidean Vector. Hint: you may need two overloaded functions to achieve this requirement.	`double a {b[1]};` `b[2] = 3.0;`
operator+=	For adding vectors of the same dimension.	`a += b;`
operator-=	For subtracting vectors of the same dimension.	`a -= b;`
operator*=	For scalar multiplication, e.g. [1 2] * 3 = [3 6]	`a *= 3;`
operator/=	For scalar division, e.g. [3 6] / 2 = [1.5 3]	`a /= 4;`

Type Conversion	Operators for type casting to a `std::vector` and a `std::list`	`EucilideanVector a;`
		`std::vector<double> vf = a;`
		`std::list<double> lf = a;`

You should not modify or augment the public interface provided.

5. Nonmember functions

In addition to the operations indicated earlier, the following operations should be supported as nonmember functions. Note that these operations don't modify any of the given operands.

operator==	True if the two vectors are equal in the number of dimensions and the magnitude in each dimension is equal.	`a == b;`
operator!=	The opposite of ==	`a != b;`
operator+	For adding vectors of the same dimension.	`a = b + c;`
operator-	For substracting vectors of the same dimension.	`a = b - c;`
operator*	For dot-product multiplication, returns a double. E.g., [1 2] * [3 4] = 1 * 3 + 2 * 4 = 11	`double c {a * b};`
operator*	For scalar multiplication, e.g. [1 2] * 3 = 3 * [1 2] = [3 6]. Hint: you'll obviously need two methods, as the scalar can be either side of the vector.	`(1) a = b * 3;` `(2) a = 3 * b;`
operator/	For scalar division, e.g. [3 6] / 2 = [1.5 3]	`a = b / 4;`
operator<<	Prints out the magnitude in each dimension of the Euclidean Vector (surrounded by [and]), e.g. for a 3-dimensional vector: [1 2 3]	`std::cout << a;`

6. Private Members

Remember you are required to store an Euclidean vector in dynamically allocated memory. Otherwise you may introduce whatever private members you feel are necessary for your implementation. This includes private member functions.

7. Move Semantics

An Euclidean vector, once moved (by your move constructor or move assignment operator), must be left in a valid state. In principle, a moved-from object can be written into but not read from. For marking purposes, please put a moved-from Euclidean vector in such a valid state that calling the output operator << on it will result in the output

```
[]
```

with nothing inside the brackets.

8. Getting Started

There is no starter code for this second assignment.

- Create EuclideanVector.h and EuclideanVector.cpp
- Place the class declaration and definition inside the namespace evec
- Ensure the name of your class is EuclideanVector
- Make sure you include Header Guards in EuclideanVector.h
- You will need to figure out the function prototypes from the above descriptions
- Your EuclideanVector.cpp should not include a main method (your class will be compiled against a series of test files that include a main method)
- Your code should not read or write any files, or print anything to the screen unless called to do so through the overloaded << overloaded from a test file which #include "EuclideanVector.h"
- It is vital that the << operator is overloaded correctly for printing out your euclidean vector. This function must be a friend and the function prototype should look like this:`std::ostream& operator<<(std::ostream &os, const EuclideanVector &v);`

Note: If you are unsure about vector mathematics look in the library for a text book on linear algebra. Additionally, the book: Bourg, David M. *Physics for Game Developers* (2001) O'Reilly Media, provides a good overview (and was an inspiration for this assignment).

9. Sample Test Case (EuclideanVectorTester.cpp)

Testing for this assignment will be done via a number of test cases written as C++ programs that compile against your library. The following program is an example of what these test cases will look like.

```
#include <iostream>
#include <vector>
#include <list>

#include "EuclideanVector.h"

int main() {

        evec::EuclideanVector a(2);

        std::list<double> l {1,2,3};
        evec::EuclideanVector b{l.begin(),l.end()};

        std::vector<double> v2 {4,5,6,7};
        evec::EuclideanVector c{v2.begin(),v2.end()};

        std::vector<double> a1 {5,4,3,2,1};
        evec::EuclideanVector d{a1.begin(),a1.end()};

        std::list<double> a2 {9,0,8,6,7};
        evec::EuclideanVector e{a2.begin(),a2.end()};

        // use the copy constructor
        evec::EuclideanVector f{e};

        std::cout << a.getNumDimensions() << ": " << a << std::endl;
        std::cout << "D1:" << b.get(1) << " " << b << std::endl;
        std::cout << c << " Euclidean Norm = " << c.getEuclideanNorm() << std::endl;
        std::cout << d << " Unit Vector: " << d.createUnitVector() << " L = " <<
d.createUnitVector().getEuclideanNorm() << std::endl;
        std::cout << e << std::endl;
        std::cout << f << std::endl;
```

```
        // test the move constructor
        evec::EuclideanVector g = std::move(f);
        std::cout << g << std::endl;
        std::cout << f << std::endl;

        // try operator overloading
        e += d;
        std::cout << e << std::endl;

        evec::EuclideanVector h = e - g;
        std::cout << h << std::endl;

        // test scalar multiplication
        h *= 2;
        std::cout << h << std::endl;

        evec::EuclideanVector j = b / 2;
        std::cout << j << std::endl;

        std::cout << "dot product = " << j * b << std::endl;

        if (g == (e - d)) std::cout << "true" << std::endl;
        if (j != b ) std::cout << "false" << std::endl;

        j[0] = 1;
        std::cout << j << std::endl;

        // type cast from EuclideanVector to a std::vector
        std::vector<double> vj = j;

        // type cast from EuclideanVector to a std::vector
        std::list<double> lj = j;

        for (auto d : lj) {
                std::cout << d << std::endl;
        }

        // list initialisation
        evec::EuclideanVector k {1, 2, 3};
        std::cout << k << std::endl;
}
```

The correct output is:

```
2: [0 0]
D1:2 [1 2 3]
[4 5 6 7] Euclidean Norm = 11.225
[5 4 3 2 1] Unit Vector: [0.6742 0.53936 0.40452 0.26968 0.13484] L = 1
[9 0 8 6 7]
[9 0 8 6 7]
[9 0 8 6 7]
[]
[14 4 11 8 8]
[5 4 3 2 1]
[10 8 6 4 2]
[0.5 1 1.5]
dot product = 7
true
false
[1 1 1.5]
1
1
1.5
[1 2 3]
```

As illustrated in this example, the first element in a Euclidean vector has a position of 0 not 1, just like in the case of `std::vector`.

10. Tips:

- Your code should be const correct.
- Use C++14 style and methods as much as possible.
- The lecture slides and tutorials have many code snippets that you may find helpful.

- Your code should be well documented, clearly describing how each function operates.
- To calculate a square root you may need to `#include <cmath>`
- Do not use other libraries (e.g., boost).
- The reference solution is around 400 lines including generous comments.

11. Testing

Place all your code in files called `EuclideanVector.h` and `EuclideanVector.cpp`

Ensure it compiles on the CSE machines using the following sample makefile

```
all: EuclideanVectorTester

EuclideanVectorTester: EuclideanVectorTester.o EuclideanVector.o
        g++ -fsanitize=address EuclideanVectorTester.o EuclideanVector.o -o
EuclideanVectorTester

EuclideanVectorTester.o: EuclideanVectorTester.cpp EuclideanVector.h
        g++ -std=c++14 -Wall -Werror -O2 -fsanitize=address -c EuclideanVectorTester.cpp

EuclideanVector.o: EuclideanVector.cpp EuclideanVector.h
        g++ -std=c++14 -Wall -Werror -O2 -fsanitize=address -c EuclideanVector.cpp

clean:
        rm *o EuclideanVectorTester
```

To compile your code type:

```
make
```

To run your code type:

```
./EuclideanVectorTester
```

You should create your own test cases to check the full functionality of your code against the specifications.

12. Marking

Your submission will be given a mark out of 100 with 70% an automarked performance component for output correctness and 30% a manually marked component for code style and quality.

As this is a third-year course we expect that your code will be well formatted, documented and structured. We also expect that you will use standard formatting and naming conventions.

Note, if you write in C or use C types (e.g. union) or #define macros you will lose performance marks as well as style marks.

A number of test cases will be used to mark your solution. To pass a test case, your solution must produce exactly the same output as the reference solution. The results from both will be compared by using the linux tool diff.

13. Submission Instructions

Copy your code to your CSE account and make sure it compiles without any errors or warnings. Then run your test cases. If all is well then submit using the command:

```
give cs6771 ass2 EuclideanVector.cpp EuclideanVector.h
```

Note you do not need to submit a makefile or other test files, we will supply a makefile and test cases for each test.

Late submissions will be penalised unless you have legitimate reasons for an extension which is arranged before the due date. **Any submission after the due date will attract a reduction of 20% per day to your individual mark.** A day is defined as a 24-hour day and includes weekends and holidays. No submissions will be accepted more than three days after the due date.

Plagiarism constitutes serious academic misconduct and will not be tolerated.

Further details about lateness and plagiarism can be found in the Course Introduction.