CS1321 Homework 4i*

Jeffrey D. Oldham

2000 Feb 27

Contents

This homework is optional and not for credit. It will not be graded and need not be submitted. It is strongly recommended, however, that you do the homework anyway to get additional practice in defining a C++ class since defining C++ classes is an important skill.

1 Reading

Start reading chapter 4.

2 Problem Statement

You are to implement a class for mathematical vectors, as discussed in class. In mathematics, a *vector* is an n-tuple of real numbers, which we could write as $(x_1 \ldots x_n)$. n is called the length of the vector. Many operations are defined on vectors including the following.

- Addition of two vectors of length n produces another vector of length n:

$$(x_1 \ldots x_n) + (y_1 \ldots y_n) = (x_1 + y_1 \ldots x_n + y_n)$$

- Subtraction of two vectors of length n produces another vector of length n:

$$(x_1 \ldots x_n) - (y_1 \ldots y_n) = (x_1 - y_1 \ldots x_n - y_n)$$

- Scalar multiplication of a vector of length n and a real number a produces a vector of length n:

$$a \cdot (x_1 \ldots x_n) = (a \cdot x_1 \ldots a \cdot x_n)$$

- The *dot product* of two vectors of length n produces a real number:

$$(x_1 \ldots x_n) \cdot (y_1 \ldots y_n) = x_1 \cdot y_1 + \ldots + x_n \cdot y_n$$

- Two vectors are considered to be equal if they are identical, i.e., they have the same length and corresponding elements are equal.

The objective in this assignment is to define a C++ class `MVector` to represent mathematical vectors. Here is an example of using this class. Our strategy for implementing `MVector` is to represent a mathematical vector by a C++ STL `vector` of `double`s. We will need to write functions to support the following operations on `MVector` objects (using syntax as illustrated in the example).

- Construct a zero-length `MVector`, an `MVector` of a specified length, or an `MVector` that is a copy of another `MVector`.
- Get or change the value of a specified element of an `MVector`.
- Get or change the size of an `MVector`.
- Form a new `MVector` by adding two `MVector`s.
- Form a new `MVector` by subtracting on `MVector` from another `MVector`.
- Increment the current `MVector` by adding to it another `MVector`.
- Compute the dot product of two `MVector`s.
- Form a new `MVector` as the scalar product of a number and an `MVector`.
- Scale the current `MVector` by a number a; i.e., modify the current `MVector` such that its new value is the scalar product of a and its old value.
- Compare two `MVector`s for equality and inequality.
- Input numbers into an `MVector` using the `>>` operator.
- Output an `MVector` using the `<<` operator.

We also need to define a type `size_type`.

3 What Files Do I Need?

There is one essential file and one recommended file.

- mvector.h contains the skeleton of the `class MVector` definition. Write your function and friend declarations here. Comments in the code indicate what you need to implement.
- mvector-use.cc uses the `MVector` class. You may want to extend this program to further test your implementation.

To use the code, write a .cc file containing a main function definition. Be sure to `#include "mvector.h"` and put the file in the same directory as `mvector.h`. Then compile the .cc file. For example,

```
g++ -Wall -pedantic mvector-use.cc -o mvector-use
```

4 Testing

You are encouraged to test your code thoroughly with the example-use program (`mvector-use.cc`), which you may want to extend with additional tests. As currently written, this program will not compile unless you implement all the desired `MVector` functions. If you want to implement and test functions a few at a time, you can do so by commenting out the parts of `mvector-use.cc` that test functions you have not yet implemented.