Programming test C++ from Haption

Hamza MAMECHE

July 21, 2021

1 The problem

We want to develop a function which right-multiplies a matrix with a vector, and fills the results into another vector given as parameter. The objective is to implement the code in the control software of a robot, with limited resources in terms of memory and computing power.

The function code shall be safe and efficient, i.e.:

- Verify the type of its parameters, including the relative size of the matrix and vectors

- Be reentrant

- Avoid any memory allocation

The following constraints are to be taken into account:

- The code shall be written in ISO C++ 2014 standard, using classes but as plain as possible

- It shall be delivered with unitary test functions

- It shall be commented in English

- The floating-point variables shall be single-precision

- The code should be platform-independent, as far as possible

- The code should avoid the use of third-party libraries, including the C++ Standard Library (except for "<iostream>")

2 My take

I wrote the function "**void** vecmultmat(**const** T vector[N], **const** T matrix[M][R], T outvector[L])" as a static function and wrap it in a templated class "**class** MatrixMult" with two static printing functions: "**void** printmatrix(**const** T matrix[N][M])" and "**void** printvector(**const** T vector[N])". As you can see I used arrays for vectors and double arrays for matrices which justify the use of the template parameters to take in the type and size of the arrays:
"**template**<**typename** T = **float**, size_t N = 1, size_t M = 1, size_t R = 1, size_t L = 1>".

The function perform $x * A = y$ with the respective dimensions $1XN \ * \ MXR = 1XL$, checks if $N == M \ and \ R == L$ and print the results.

- For type checking, the function parameters are casted at function call from pointers to arrays with the provides type and dimensions, note that using raw arrays, the user is responsible for providing correct template parameters as it can not be checked inside the function.

- As for the constraint to not use C++ libraries and using raw arrays as our data structure, it is not straightforward to police the parameters types to be numerical so for a demonstration I used template specialization to cast away the case of calling the function with "**char**", so a not straightforward way to only accept numerical parameters in this settings is to use template specialization for all **int**#_t and uint#_t plus **float** as special cases with the provided code and provide the general case with "Datatype must be numerical!".

In the main function, we test the function with simple cases:

```
============Test1: providing a compatible operands============
Vector =
        [ 1 2 ]
Matrix =
        [ 1 2 3 ]
        [ 4 5 6 ]
Vector*Matrix =
        [ 9 12 15 ]
============Test2: providing a uncompatible operands============
Incompatible sizes! Make sure to enter the template paramaters correctly
============Test3: multiplying a vector by the identity matrix============
Vector =
        [ 20 2 -10 ]
Matrix =
        [ 1 0 0 ]
        [ 0 1 0 ]
        [ 0 0 1 ]
Vector*Matrix =
        [ 20 2 -10 ]
============Test4: the vector and the matrix are of char type============
Datatype must be numerical!
```