HW 01

Part I : Matrix

a. Write a class Column_Major_Matrix that has a member all_column which is of type vector<vector<T>>
b. Write a class Row_Major_Matrix that has a member all_row which is of type vector<vector<T>>
c. Provide a constructor for each class that takes arguments to specify the dimensions (e.g., `Column_Major_Matrix<int> cc1 (1000, 1000);`), and fills up all elements by randomly generated values of type T.
d. Provide getter/setter function to access each column and row by an index.
e. Overload copy/assignment and move copy/assignment operator to allow the following in the main function:

```
Column_Major_Matrix<int> cc1 (1000, 1000);
Row_Major_Matrix<int> rr1( 1000, 1000);
Column_Major_Matrix<int> cc2 (cc1);
Row_Major_Matrix<int> rr2 = (rr1);
Column_Major_Matrix<int> cc3 = std::move( cc2 );
Row_Major_Matrix<int> rr3 = std::move( rr2 );
```

f. Overload operator* in Row_Major_Matrix to allow calculation of the product of a Row_Major_Matrix instance to a Column_Major_Matrix instance, and return the resultant product as a Row_Major_Matrix.
g. Overload operator* in Column_Major_Matrix to allow calculation of the product of a Column_Major_Matrix instance to a Row_Major_Matrix instance, and return the resultant product as a Column_Major_Matrix.

h. Write type conversion operators (i.e., operator Row_Major_Matrix() and operator Column_Major_Matrix()) to allow implicit type conversion between Row_Major_Matrix and Column_Major_Matrix. Show it works by:

```
Column_Major_Matrix<int> cc (55, 1000);
Row_Major_Matrix<int> rr (1000, 66);
Row_Major_Matrix<int> rr = cc*rr;
```

i. Overload operator% to use exactly 10 threads to multiplex the multiplication, and use std::chrono to show the speedup w/ and w/o multithreading.

Part II : Thread pool

1. Design a thread pool class with following features:
   A. Allow users to send jobs into the pool
   B. Allow any kind of callable objects as jobs
   C. Maintain a job queue to store unfinished jobs
      i. Hint: element type: std::function/std::bind or package_task
   D. Have 5 threads always waiting for new jobs. Each thread will keep a record of total running time throughout the lifespan of the thread.
   E. Threads are terminated(joined) only when the thread pool is destructed. The total running time of each thread will be shown on the screen upon destruction along with the std::thread::id.
   F. Use condition variable and mutex to notify threads to do works

2. Write one function (named print_1), which can generate a random integer number and then print out '1' if the number is an odd number otherwise '0'. Note that cout is also a shared resource.

3. Write a print_2 functor, which simply prints "2" on the screen. Use conditional variable to ensure that print_2 functor can only be executed when there is no more print_1 job to be executed.

4. In main, first send 496 functions and then 4 functors into the pool.

```
#include<queue>
#include<functional>
#include<iostream>
Void add()
{
	std::cerr<<“1”<<std::endl;
}

struct ADD
{
	void operator()()
	{
		std::cerr<<”2”<<std::endl;
	}
};

int main(void)
{
	ADD a;
	std::queue< std::function<void(void)> > jobs;
	jobs.push( std::bind(add) );
	jobs.push( std::bind( std::bind(a) ) );
	jobs.push( std::bind(a) );

	while(!jobs.empty() )
	{
		jobs.front()();
		jobs.pop();
	}
	return 0;
}
```