1. C++ exercise

1. Write a program in C++ which swaps the values of two variables not using third variable.

```
#include <iostream>
using namespace std;
int main()
{
    int a,b;
    cin>>a>>b;

    // your code here

    cout<<"a="<<a<<"b="<<b<<endl;
    return 0;
}
```

2. There are four different points on a plane: A(x1, y1), B(x2, y2), C(x3, y3) and D(x4, y4). Write a C++ program to check whether two straight lines AB and CD are orthogonal or not.

```
#include <iostream>
using namespace std;
int main()
{
    double x[4],y[4];
    for(int i=0; i<4; i++)
    {
        cin >> x[i] >> y[i];
    }

    // your code here

return 0;
}
```

3. Recursion: Print Fibonacci Series F_n

Fibonacci Series: $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$.

(1) Print Fibonacci Series F_n using recursion in C++.

(2) Print Fibonacci Series F_n using **dynamic programming** in C++. (When you need to print Fibonacci Series **up to** n number of terms?, dynamic programming can save you lots of time.)

(3) Use `microbenchmark()` in R to compare their speed.

```
#include<iostream>
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]
using namespace arma;
using namespace std;
```

```
// [[Rcpp::export]]
int fib(int x)
{
    // ....
}

// [[Rcpp::export]]
int fib_dp(int n)
{
    //...
}
```

4. Generates a uniform random integer

(1) Write a function `rand7` in C++ which generates a uniform random integer in the range 1 to 7.

(2) Then write a function `rand10` in C++ using `rand7` which generates a uniform random integer in the range 1 to 10.

Check your result in R using histogram.

```
int rand7()
{
  // your code here
}

int rand10() {
    // your code here
    // you should only use rand7()
}
```

```
library(Rcpp)
sourceCpp("yourcpp.cpp")

N = 10000
result = rep(0,10000)
for (i in 1:N) {
  result[i] = rand10()
}

library(ggplot2)
ggplot() + geom_bar(aes(result))
```

2. Armadillo

0. Armadillo Matrix Classes

> http://arma.sourceforge.net/docs.html

- Basic knowledge:
  - Basically a **Template**: `Mat<type>`
  - `type` is one of
    - `float, double, std::complex<float>, std::complex<double>, short, int, long`
    - unsigned versions of `short, int, long`. For example, `uword, sword`.
  - Some classes:
    - `mat = Mat<double>`
    - `umat = Mat<uword>`
- submatrix views
  - A collection of member functions of *Mat*, *Col* and *Row* classes that provide read/write access to submatrix views.
  - contiguous views for matrix X:
    - `X.col( col_number )`, `X.row( row_number )`
    - `X.submat( first_row, first_col, last_row, last_col )`
    - `X( span(first_row, last_row), span(first_col, last_col) )`
    - `X( span(first_row, last_row), col_number )`
    - `X( row_number, span(first_col, last_col) )`
  - non-contiguous views for matrix or vector X:
    - `X.cols( vector_of_column_indices )`,`X.rows( vector_of_row_indices )`
    - `X.submat( vector_of_row_indices, vector_of_column_indices )`
    - `X( vector_of_row_indices, vector_of_column_indices )`
- member functions
  - `svd`: Singular value decomposition
  - `chol`: Cholesky decomposition
  - `eig_sym`: Eigen decomposition of dense symmetric/hermitian matrix X
  - `solve`: Solve a dense system of linear equations
  - `as_scalar`: convert 1x1 matrix to pure scalar
- member variables
  - `.n_rows`
  - `.n_cols`
  - `.n_elem`
- generated vectors / matrices
- ....