```c
/*
 * Purpose: Demonstrate and time matrix multiplication on the CPU
 *
 * Date and time: 04/09/2014
 * Last modified: 03/16/2016
 * Author: Inanc Senocak
 *
 * to compile: gcc -O2 -lcblas -o CPU.exe cpu_matrixMultiply.c
 * to execute: ./CPU <m> <n> <k>
 *
 */

#include "timer.h"
#include <cblas.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/resource.h>
#include <time.h>

typedef double REAL;

void printMatrix(REAL *matrix, const int nrow, const int ncol)
{
        int i, j, idx;

        for (j = 0; j < nrow; j++) {
                for (i = 0; i < ncol; i++) {
                        idx = i + j * ncol;
                        printf("%8.2f ; ", matrix[idx]);
                }
                printf("\n");
        }
        printf("\n");
}

void InitializeMatrices(REAL *a, REAL *b, const int M, const int N, const int K)
{
        int i, j, idx;

        // initialize matrices a & b
        for (j = 0; j < M; j++) {
                for (i = 0; i < K; i++) {
                        idx    = i + j * K;
                        a[idx] = (REAL) idx;
                }
        }

        for (j = 0; j < K; j++) {
                for (i = 0; i < N; i++) {
                        idx    = i + j * N;
                        b[idx] = (REAL) idx;
                }
        }
}

void matrixMultiply(REAL *a, REAL *b, REAL *c, const int M, const int N, const int K)
{
        // this function does the following matrix multiplication c = a * b
        // a(m x k); b(k x n); c(m x n)

        int  i, j, idk, idx;
        REAL sum = 0.f;
        // multiply the matrices C=A*B
        for (i = 0; i < N; i++) {
                for (j = 0; j < M; j++) {
                        for (idk = 0; idk < K; idk++) {
                                sum += a[idk + j * K] * b[i + idk * N];
                        }
                        c[i + j * N] = sum;
                        sum          = 0.f;
                }
        }
}

void my_ddot(REAL *A, REAL *B, REAL *C, const int M, const int N, const int K)
{
        int i, j;
        for (j = 0; j < M; j++) {
                for (i = 0; i < N; i++) {
                        C[i + j * N] = cblas_ddot(K, A + j * K, 1, B + i, N);
                }
        }
}

double my_daxpy(REAL *A, REAL *B, REAL *C, const int M, const int N, const int K)
```



```c
{
        int i, idk;
        for (i = 0; i < N; i++) {
                for (idk = 0; idk < K; idk++) {
                        cblas_daxpy(M, B[i + idk * N], A + idk, K, C + i, N);
                }
        }
}

int main(int argc, char *argv[])
{
        if (argc < 3) {
                perror("Command-line usage: executableName <m> <k> <n>");
                exit(1);
        }

        int M = atof(argv[1]);
        int K = atof(argv[2]);
        int N = atof(argv[3]);

        REAL *a = (REAL *) calloc(M * K, sizeof(*a));
        REAL *b = (REAL *) calloc(K * N, sizeof(*b));

        REAL *c = (REAL *) calloc(M * N, sizeof(*c)); // Used for CPU
        REAL *d = (REAL *) calloc(M * N, sizeof(*d)); // Used for DDOT
        REAL *e = (REAL *) calloc(M * N, sizeof(*e)); // Used for DAXPY

        InitializeMatrices(a, b, M, N, K);

        double startCPU, finishCPU, elapsedTimeCPU;
        GET_TIME(startCPU);
        matrixMultiply(a, b, c, M, N, K);
        GET_TIME(finishCPU);
        elapsedTimeCPU = finishCPU - startCPU;

        printf("=====CPU=====\n");
        printf("CPU C[2] = %3.1f\n", c[2]);
        //    printMatrix(c, M, N);
        printf("elapsed wall time (CPU) = %.6f microseconds\n", elapsedTimeCPU * 1.0e6);
        printf("\n");

        double startDDOT, finishDDOT, elapsedTimeDDOT;
        GET_TIME(startDDOT);
        my_ddot(a, b, d, M, N, K);
        GET_TIME(finishDDOT);
        elapsedTimeDDOT = finishDDOT - startDDOT;

        printf("=====DDOT()=====\n");
        printf("DDOT d[2] = %3.1f\n", d[2]);
        //    printMatrix(d, M, N);
        printf("elapsed wall time (DDOT) = %.6f microseconds\n", elapsedTimeDDOT * 1.0e6);
        printf("\n");

        double startDAXPY, finishDAXPY, elapsedTimeDAXPY;
        GET_TIME(startDAXPY);
        my_daxpy(a, b, e, M, N, K);
        GET_TIME(finishDAXPY);
        elapsedTimeDAXPY = finishDAXPY - startDAXPY;

        printf("=====DAXPY()=====\n");
        printf("DAXPY e[2] = %3.1f\n",  e[2]);
        //    printMatrix(e, M, N);
        printf("elapsed wall time (DAXPY) = %.6f microseconds\n", elapsedTimeDAXPY * 1.0e6);
        printf("\n");

        // Deallocating Memory
        free(a);
        a = NULL;
        free(b);
        b = NULL;
        free(c);
        c = NULL;
        free(d);
        d = NULL;
        free(e);
        e = NULL;
        return (EXIT_SUCCESS);
}
```