Exercise 3

Task 3.1 (Exam 2007: OpenMP)**.** Develop parallel versions of the following codes by augmenting them with OpenMP directives. Try to reduce the number of implicit barriers and specify every used variable as `shared` or `private`.

a)
```
for (i=0; i<m; i++){
    y[i]=0;
    for (j=0; j<n; j++){
        y[i]= y[i] + A[i][j]*x[j];
    }
}
```

b)
```
s=0;
for (i=0; i<m; i++){
    y[i]=0;
    for (j=0; j<n; j++){
        y[i]= y[i] + A[i][j]*x[j];
    }
    s += y[i];
}
```

c)
```
y[0]=0;
for (i=1; i<m; i++){
    y[i]=y[i-1];
    for (j=0; j<n; j++){
        y[i]= y[i] + A[i][j]*x[j];
    }
}
```

d) Parallelize the following three loops in a single parallel region and use the `nowait` clause whenever possible.

```
for (i=0; i<m; i++){
    y[i]=0;
    for (j=0; j<n; j++){
        y[i]= y[i] + A[i][j]*x[j];
    }
}
for (i=0; i<m; ++i){
    alpha[i]=A[i][0]*A[i][1];
}
for (i=0; i<m; i++){
    y[i] = y[i]*alpha[i];
}
```

Task 3.2 (Exam 2012: OpenMP)**.** Look at the code in listing 3.1. Function foo1() prints *"foo1"* and creates another task that prints *"foo2"*.

Listing 3.1: OpenMP Taskwait / Barrier

```
1  int main () {
2  #pragma omp parallel num_threads(1)
3    {
4      #pragma omp task
5      {
6        printf("foo1 \n");
7        #pragma omp task
8        {
9          printf("foo2 \n");
10       }
11     }
12     #pragma omp taskwait OR #pragma omp barrier
13     printf("Synchronization done \n");
14   }
15   return 0;
16 }
```

(1) Indicate the difference between using either `#pragma omp taskwait` or `#pragma omp barrier` in line 12 of the code.

(2) With either a `#pragma omp taskwait` or a `#pragma omp barrier` primitive, the program has the following output. Indicate the used variant and justify why the other one could not produce this output:

```
foo1
Synchronization done
foo2
```