מחשוב מקבילי ומבוזר

תרגיל #1

The purpose of this exercise is to implement a simple application with **Dynamic** and **Static** Task Pool approaches.

Important:

- The homework may be performed in pairs. Only one member of pair submits the solution through the Moodle. The whole project must be zipped and named as **111111111_222222222.zip**

Where **111111111** is ID of the one student and **222222222** is ID of another student

Parallelize the following code:

```
#include <stdio.h>
#include <math.h>

#define HEAVY 100000
#define SHORT 1
#define LONG 10


// This function performs heavy computations,
// its run time depends on x and y values
 double heavy(int x, int y) {
            int i, loop = SHORT;
            double sum = 0;

// Super heavy tasks
       if (x < 3 || y < 3)
            loop = LONG;
       // Heavy calculations
       for(i = 0;  i < loop*HEAVY;  i++)
          sum += cos(exp(sin((double)i/HEAVY)));

     return sum;
}


int main(int argc, char *argv[]) {
            int x, y;
            int N = 20;
            double answer = 0;

            for (x = 0; x < N; x++)
                  for (y = 0; y < N; y++)
                        answer += heavy(x, y);

       printf("answer = %e\n", answer);
}
```

Requirements:

1. Implement two approaches to parallelize the code:
    a. Use **Static Task Pool** approach to solve the problem
    b. Implement **Dynamic Task Poo**l Approach for parallel solution

2. Run, measure execution time and explain the results. The table with the time measurement is to be placed in the separate Word file named **results.doc** in the root directory of the solution.

3. No changes to function heavy are allowed.

Solution type	Number of Slaves	Execution time	Explain the result
Sequential Solution	1		
Static Task Pool	2		
Static Task Pool	4		
Dynamic Task Pool	2		
Dynamic Task Pool	4		
Dynamic Task Pool	20		

Grading Policy:

- **20 points** for code quality:
    a. The code must be divided into small functions (not more than 40 lines of code).
    b. Use meaningful names for variables, functions, files, constants.
    c. Place enough comments to understand the code
    d. No unused lines of code. Don't repeat the code – use functions!
    e. Write README.TXT file if special instructions are needed to run the solution. The file must be in the root folder of the solution.
- **60 points** – for proper implementation of parts 1, 2 of the requirements.

- **20 points** – for final results explanation and for time measurement.

Note:
You may need to link with –lm (the math library)

בהצלחה!