מחשוב מקבילי ומבוזר

תרגיל #1

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

- Parallelize the following code:

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


#define HEAVY 1000
#define SIZE 40
#define RADIUS 10

#define FILE_NAME "points.txt"
// This function simulates heavy computations,
// its run time depends on x and y values
// DO NOT change this function!!
double heavy(int x, int y) {
	int i, loop;
	double sum = 0;
	if (sqrt((x - 0.25 * SIZE) * (x - 0.25 * SIZE) + (y - 0.75 * SIZE) * (y - 0.75 *
SIZE)) < RADIUS)
		loop = 5 * x * y;
	else
		loop = abs(x-y) + x;
	for (i = 0; i < loop * HEAVY; i++)
		sum += sin(exp(cos((double)i / HEAVY)))/HEAVY;
	return  sum;
}

// Reads a number of points from the file.
// The first line contains a number of points defined.
// Following lines contain two integers each - point coordinates x, y
int *readFromFile(const char *fileName,  int *numberOfPoints) {
	FILE* fp;
	int* points;

	// Open file for reading points
	if ((fp = fopen(fileName, "r")) == 0) {
		printf("cannot open file %s for reading\n", fileName);
		exit(0);
	}

	// Number of points
	fscanf(fp, "%d", numberOfPoints);

	// Allocate array of points end Read data from the file
	points = (int*)malloc(2 * *numberOfPoints * sizeof(int));
	if (points == NULL) {
		printf("Problem to allocate memotry\n");
```

```
			exit(0);
		}
		for (int i = 0;   i < *numberOfPoints; i++) {
			fscanf(fp, "%d %d", &points[2*i], &points[2*i + 1]);
		}

		fclose(fp);

		return points;
}


// Sequencial code to be parallelized
int main(int argc, char* argv[]) {
		double answer = 0;
		int numberOfPoints;
		int *points;

		// Read points from the file
		points = readFromFile(FILE_NAME, &numberOfPoints);

		// Perform heavy sequencial computation
		for (int i = 0;    i < numberOfPoints;   i++)
			answer += heavy(points[2 * i], points[2 * i + 1]);
		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 Pool** Approach for parallel solution

2. Run, measure execution time, 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. It is considered as a “black box” , meaning that your solution is not based on understanding what kind of computation is made and how long it may run for specific parameters x and y.

Solution type	Number of Slaves	Execution time	Explain the result
Sequential Solution			
Static Task Pool	2		
Static Task Pool	4		
Static Task Pool	10		
Dynamic Task Pool	2		
Dynamic Task Pool	4		
Dynamic Task Pool	10		

Grading Policy:

- **10 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.
- **70 points** – for proper implementation of the requirements.
- **20 points** – for final results explanation and for time measurement.

Important:

- The Homework has to be tested under Ubuntu OS in VLAB with compilation and running from Terminal.
- The Homework must be delivered in time. It may be performed in pairs. *Only one member of pair* submits the solution through the Moodle.
- The whole solution must be zipped and named as

111111111_222222222.zip

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

בהצלחה!