A2 (48 marks)

***Focus**: C pointers and functions, intro to OpenMP*

Q1. [12 marks] Write a C function that sequentially adds 2 vectors A and B as $C[i] = A[i] + B[i]$. Use this function header: `void addVec(int* C, int* A, int* B, int size)`, where size is the number of elements in a vector. You should ***not*** create any additional local variables in the function (hint: use pointers and pointer expressions).
Test your function with two vectors, each having 50 million integers, all initialized to 0 (hint: `calloc`). Note that memory allocation for A, B, and C is done within your test code (the `main` function). Print the first 10 elements of C as well as the total time the function took to complete. Report the results as a comment in your code.

Sample run 1: (successful)

```
0 0 0 0 0 0 0 0 0 0
Execution time: 281.0 ms
```

Sample run 2: (unsuccessful memory allocation)

```
Not enough memory.
```

Marking: +6 for the function, +5 for the test code (2 marks for error checking), +1 for the comment)

Q2. [12 marks] Repeat question Q1 with a function that receives pointers to A and B, allocates memory space for C, and returns a pointer to the memory holding the computed output. Use the header `int* addVec2(int* A, int* B, int size)`. The function should return NULL if memory allocation for C fails. You can only create **two** local variables of the type int* within the function addVec2. Your test code should produce output similar to the sample runs in Q1.

Marking: +6 for the function (2 for error checking), +5 for the test code(2 for error checking), +1 for the comment

Q3. [4 marks] **Intro to OpenMP**: Start the Eclipse, create an OpenMP project, and copy and paste the code below into a new source file. Build and run. Check everything works as expected, if not fix as appropriate. Copy the output into a text file named Q3.txt.

```
int main(int argc, char *argv[]) {
	int numThreads, tid;
	/* This creates a team of threads; each thread has own copy of variables  */
	#pragma omp parallel private(numThreads, tid)
	{
		tid = omp_get_thread_num();
		printf("Hello World from thread number %d\n", tid);

		/* The following is executed by the master thread only (tid=0) */
		if (tid == 0) {
			numThreads = omp_get_num_threads();
			printf("Number of threads is %d\n", numThreads);
		}
	}
	return 0;
}
```

Marking: +4 for running code

Q4. [20 marks] Develop a C function that returns an `int*` pointer to a vector A initialized as $A[i] = i$, or returns NULL if the function fails.

- Write the sequential code for the function. Use the header:

```
int* vecCreate (int size)
```

- Write the OpenMP implementation of the function. Use the header:

```
int* vecCreateOpenMP(int size, int num_thread)
```

  Here, you will try to speed up the vector initialization by dividing the work among several threads where each thread will initialize a segment of the vector. For example, if num_thread = 2 and size = 10, thread 0 will initialize elements 0 to 4 and thread 1 will initialize elements 5 to 9. Using `#pragma omp parallel num_threads( num_thread )`. Your code should be similar to Version1 of the Trapezoid Area Calculation example from the lecture notes. Your function should only work if the vector size is divisible by the number of threads. If not, your function should display an error message and return NULL.

Test both functions with a 50-million element vector (and 4 threads for the second function). Your test code should print the value of the last element in the vector along with the time taken for each function to complete. Report the results as a comment in your code.

Sample run 1: (successful)

```
Using serial code
v[49999999] = 49999999
Time: 144.00 ms

Using OpenMP with 4 threads:
v[49999999] = 49999999
Time: 59.00 ms
```

Sample run 2: (num_thread not divisible by size)

```
Using serial code
v[49999999] = 49999999
Time: 144.00 ms

Using OpenMP with 3 threads: Error: number of threads must be divisible by vector size.
```

Sample run 3: (unsuccessful memory allocation)

```
Not enough memory.
Not enough memory.
```

Marking: +5 for serial code, +10 for parallel code, +5 for the test code. Up to -4 marks for not handling errors.

Submission Instructions

For this assignment, you need to do the following:

1- *Programming questions*: Create one C file for *each* programming question and write your answer inside that file. Your files should have the same name as the question number (e.g., Q1.c)

2- Non-programming questions:
   a. If there are any discussion/essay questions related to a programming question, write your answers as comments at the end of your code for that question.
   b. For all other non-programming questions (i.e., not related to any programming question), write your answers to all of them in *one* Word document file,

3- After solving all questions, compress all your files into one zip folder and give a name to the zipped file that matches your ID (e.g., 1234567.zip).

4- Submit the zipped file to Blackboard Connect.

Note that you can resubmit an assignment, but the new submission overwrites the old submission and receives a new timestamp.