تمرین شماره ۲**برنامه‌نویسی موازی**
نیم‌سال اول **۹۹-۹۸**
زمان آپلود: شنبه ۲۵ آبان (ساعت ۱۲ شب)

نکته: توجه داشته باشید که علاوه بر گزارش نتایج خروجی، Screen Shot از اجرای برنامه را نیز آپلود کنید.

تمرین ۱: برنامه زیر را در نظر بگیرید (فایل exer01_1.cpp). در این برنامه در یک حلقه به یک متغیر مشترک N واحد اضافه می‌شود. در این برنامه یک ساختار موازی با دو ریسمان وجود دارد که عملیات مورد نظر را انجام می‌دهند.

```
#include "stdio.h"
#include "omp.h"

#define  N  100

int main(int argc, char * argv[])
{
	int shared_data = 0;
	int i;

	#pragma omp parallel num_threads(2) shared(shared_data) private(i)
	{
		#pragma omp for
			for (i = 0; i < N; i++)
				shared_data ++;
	}
	printf ("The result is %d\n", shared_data);

	return 0;
}
```

الف- برنامه را با مقدار N = 100 چهار بار اجرا کنید. خروجی برنامه چیست؟ آیا همیشه خروجی یکسان است؟ آیا انتظار مشاهده این خروجی را داشتید؟ پاسخ خود را توضیح دهید.

ب- برنامه را با مقدار N = 100000 چهار بار اجرا کنید. خروجی برنامه چیست؟ آیا انتظار مشاهده این خروجی را داشتید؟ پاسخ خود را توضیح دهید.

ج- موارد **الف** و **ب** را برای برنامه زیر (فایل exer01_2.cpp) تکرار کنید.

```
#include "stdio.h"
#include "omp.h"

#define  N  100

int main(int argc, char * argv[])
{
	int shared_data = 0;
	int i;

	#pragma omp parallel num_threads(2) shared(shared_data) private(i)
	{
		#pragma omp for reduction (+:shared_data)
			for (i = 0; i < N; i++)
				shared_data ++;
	}
	printf ("The result is %d\n", shared_data);

	return 0;
}
```

تمرین ۲: می دانیم که در OpenMP حلقه for به شرطی قابل موازی‌سازی است که تعداد تکرارهای آن از پیش مشخص باشد. این امر سبب می‌شود که نتوان از حلقه‌ی for برای پردازش اعضای یک لیست پیوندی استفاده کرد. در زیر دو راه برای این امر ارائه شده است.

راه اول: در این حالت ابتدا یک آرایه از اشاره‌گر به تمام اعضای لیست ایجاد می‌کنیم، سپس با استفاده از این آرایه تک تک اعضای لیست را پردازش می‌کنیم. دقت کنید که در این حالت با پیمایش لیست، تعداد اعضای لیست هم مشخص می‌شود.

```
ptrNode ptr = head;
while (ptr) {
      list_items [num_elemnts++] = ptr;
      ptr = ptr->next;
}
#pragma omp parallel for num_threads (NO_THREADS)
      for (i = 0; i < num_elemnts; i++)
            list_items[i]->data *= 2;   // Process data item
```

راه دوم: در این حالت از یک ساختار single استفاده می‌کنیم، با استفاده از nowait می‌توان امکان پردازش موازی اعضای لیست را توسط ریسمان‌ها را فراهم کرد.

```
#pragma omp parallel num_threads (NO_THREADS) private(ptr)
{
      ptr = head;
      while (ptr) {
            #pragma omp single nowait
                  ptr->data *= 2;        // Process data item
                  ptr = ptr->next;
      }
}
```

به نظر شما کدام یک از این روش‌ها مناسب‌تر است (از دیدگاه زمان اجرا و حافظه مصرفی)؟ این دو روش در برنامه exer02.cpp آمده است. این برنامه را برای تعداد ریسمان‌های ۲، ۴ و ۸ و برای تعداد اعضای لیست 100000، 1000000 و10000000 اجرا کنید.

تمرین ۳: برنامه exer03.cpp پیاده‌سازی ضرب ماتریس را در دو حالت سریال و موازی نشان می‌دهد. این برنامه را برای ماتریس‌هایی با اندازه ۱۲۸، ۲۵۶، ۵۱۲ و ۱۰۲۴ با تعداد ریسمان‌های ۲، ۴، ۶، ۸، ۱۰ و ۱۶ اجرا کنید. در یک منحنی تسریع پیاده‌سازی موازی را بر حسب تعداد ریسمان‌ها رسم کنید (محور افقی تعداد ریسمان‌ها و محور عمودی تسریع را نشان دهد). در یک جدول میزان efficiency (از تقسیم میزان تسریع بر تعداد ریسمان‌ها به دست می‌آید) را نشان دهید (سطرهای جدول ابعاد ماتریس و ستون‌های جدول تعداد ریسمان‌ها را نشان دهد).

تمرین 4: برنامه exer04_1.cpp یک برنامه سریال را نشان می‌دهد. زمان اجرای این برنامه را به دست آورید. دقت کنید اگر برنامه را در محیط سیستم عامل windows اجرا می‌کنید، تابع timeGetTime از پیش تعریف شده است، بنابراین این تابع باید از متن برنامه حذف شود. برنامه exer04_2.cpp پیاده‌سازی موازی این تابع را با استفاده از OpenMP نشان می‌دهد. این برنامه را با زمانبندی‌های زیر انجام دهید:

- static
- dynamic, 1000
- dynamic, 2000

زمان اجرای موازی این برنامه و میزان تسریع را گزارش کنید (توجه کنید که برنامه موازی ۶ بار اجرا می‌شود (حلقه خارجی)، زمان اجرای متوسط در این ۶ بار تکرار را گزارش کنید). با تغییر کد، زمان اجرای هر ریسمان را به دست آورید. این زمان نشان‌دهنده توازن بار توزیع شده بین ریسمان‌ها است.