```
/*
Name: Amruta Nandargi
Roll No.: PF21
PRN: S1032180391
Problem Statements:
1] Hello World
2] num_threads
3] parallel for
4] sections
5] Sum of array elements
6] atomic derivative
7] Max element of array
*/


#include<stdio.h>
#include<omp.h>
#include<time.h>
#include<stdlib.h>


//HELLO WORLD
/*int main(){
	#pragma omp parallel
	{
		printf("Hello World");
	}
	//system("PAUSE");
	return 0;
}*/
/*OUTPUT
Hello WorldHello World*/



//NUM_THREADS
/*int main(){
	#pragma omp parallel num_threads(20)
	{
		int n=omp_get_thread_num();
		printf("\nThread no.: %d ", n);
	}
	return 0;
}*/
/*OUTPUT
Thread no.: 1
Thread no.: 2
Thread no.: 3
Thread no.: 4
Thread no.: 5
Thread no.: 6
Thread no.: 7
Thread no.: 8
Thread no.: 9
Thread no.: 10
Thread no.: 11
Thread no.: 12
Thread no.: 13
Thread no.: 14
Thread no.: 15
```

```
Thread no.: 16
Thread no.: 17
Thread no.: 0
Thread no.: 18
Thread no.: 19 */



//PARALLEL FOR
/*int main(){

		#pragma omp parallel for

			for(int i=0; i<15; i++)
			{
				printf("\n%d", i);
				int n=omp_get_thread_num();
				printf("\tthread no.: %d", n);
			}


	return 0;
}*/
/*OUTPUT
0	thread no.: 0
1	thread no.: 0
2	thread no.: 0
3	thread no.: 0
4	thread no.: 0
5	thread no.: 0
6	thread no.: 0
7	thread no.: 0
8	thread no.: 1
9	thread no.: 1
10	thread no.: 1
11	thread no.: 1
12	thread no.: 1
13	thread no.: 1
14	thread no.: 1*/


//SECTIONS
/*int main(){

#pragma omp parallel
{
	#pragma omp sections
	{
		#pragma omp section
		{
			int id=omp_get_thread_num();
			printf("\nExecuted by thread number: %d\n", id);
		}
		#pragma omp section
		{
			int id=omp_get_thread_num();
			printf("\nExecuted by thread number: %d\n", id);
		}
	}
}
```

```
return 0;
}*/
/*OUTPUT
Executed by thread number: 0

Executed by thread number: 1
*/



//SUM OF ARRAY ELEMENTS
int main(){

	clock_t ss, sp, es, ep;
	double cpu_time_serial, cpu_time_parallel;

	long long sum=0, na=10000, i=0;
	int* a;

	a= malloc(na * sizeof(int));
	//srand(0);
	for(i =0; i<na; i++){
		a[i]=rand()%10;
	}

	sp=clock();

#pragma omp parallel for shared(sum)
	for (i = 0; i < na; i++) {
#pragma omp critical
		sum = sum + a[i];

	}
	ep=clock();

	printf("Sum of the array elements(parallel): %lld\n", sum);
	sum=0;

	cpu_time_parallel= ((double)(ep-sp))/CLOCKS_PER_SEC;
	printf("Parallel execution time: %lf\n", cpu_time_parallel);

	ss=clock();
	for(i=0; i<na; i++){
		sum=sum+a[i];
	}
	es=clock();

	printf("Sum of the array elements(serial): %lld\n", sum);

	cpu_time_serial= ((double)(es-ss))/CLOCKS_PER_SEC;
	printf("Serial execution time: %f\n", cpu_time_serial);
	free(a);
	return 0;
}
/*OUTPUT
Sum of the array elements(parallel): 10675833115211
Parallel execution time: 0.000721
Sum of the array elements(serial): 10675833115211
Serial execution time: 0.000037
```

```
*/

//ATOMIC DERIVATIVE
/*int main(){
	int count =0;

	#pragma omp parallel num_threads(10)
	{
		#pragma omp atomic
		count++;
	}

	printf("Number of threads: %d", count);

	return 0;
}*/
/*Output:
Number of threads: 10*/

//MAX ELEMENT OF ARRAY
/*int main(){

	clock_t ss, sp, es, ep;
	double cpu_time_serial, cpu_time_parallel;

	long long sum=0, na=100000000, i=0, max1=-1, max=-1;
	int* a;

	a= malloc(na * sizeof(int));
	srand(0);
	for(i =0; i<na; i++){
		a[i]=rand();
	}

	sp=clock();
	#pragma omp parallel for shared(a) private(i) num_threads(2)
	for(i=0; i<na; i++){
		if(a[i]>max){
			max=a[i];
		}
	}
	ep=clock();

	printf("Max element is (parallel): %lld\n", max);
```

```
	cpu_time_parallel= ((double)(ep-sp))/CLOCKS_PER_SEC;
	printf("Parallel execution time: %lf\n", cpu_time_parallel);

	ss=clock();
	for(i=0; i<na; i++){
		if(a[i]>max1){
			max1=a[i];
		}
	}
	es=clock();
	printf("Max element is (serial): %lld\n", max1);
	cpu_time_serial= ((double)(es-ss))/CLOCKS_PER_SEC;
	printf("Serial execution time: %f\n", cpu_time_serial);
	return 0;

}*/
/*OUTPUT:
Max element is (parallel): 2147483611
Parallel execution time: 0.299388
Max element is (serial): 2147483611
Serial execution time: 0.316355

*/
```