20BCE1550
Samridh Anand Paatni
CSE4001 Lab 05
OMP Synchronization Constructs

Q1.

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

#define N 10

#define OMP_NUM_THREADS 4

/*
1. Write your own code snippet to demonstrate the following
    a. Barrier
    b. Master
    c. Single
    d. Critical
    e. Ordered
*/

void barrier() {
    printf("\na. Barrier:\n");
    int a[1000], b[1000], i = 0, sum = 0;

    for (i = 0; i < 1000; i++) {
        a[i] = rand() % 100;
        b[i] = rand() % 10;
    }

    #pragma omp parallel private(i)
    {
        for (i = 0; i < 1000; i++) {
            a[i] = a[i] - b[i];
        }

        #pragma omp barrier

        #pragma omp for reduction(+ : sum)
        for (i = 0; i < 1000; i++)
        {
            sum += a[i];
        }
    }
    printf("sum: %d\n", sum);
}
```

```c
void master() {
    printf("\nb. Master\nwithout 'master':\n");
    #pragma omp parallel
    {
        printf("hello, from thread %d\n", omp_get_thread_num());
    }

    printf("\nwith master:\n");
    #pragma omp parallel
    {
        #pragma omp master
        {
            printf("hello, from thread %d\n", omp_get_thread_num());
        }
    }
}

void single() {
    printf("\nc. Single:\n");
    int a=0, b=0;
    #pragma omp parallel num_threads(4)
    {
        #pragma omp single
        a++;
        #pragma omp critical
        b++;
    }
    printf("single: %d | critical: %d\nsingle runs once, critical is run once per
thread\n", a, b);
}

void critical() {
    printf("\nd. Critical:\n");

    int i; int max; int a[N];
    for (i = 0; i < N; i++) {
        a[i] = rand();
        printf(
            "a[%d] = %d\tthread no %d\n",
            i,
            a[i],
            omp_get_num_threads()
        );
    }
    max = a[0];
    #pragma omp parallel for
    for (i = 1; i < N; i++) {
        if (a[i] > max) {
            #pragma omp critical
            {
                if (a[i] > max) max = a[i];
            }
        }
```

```
    }
    printf("\nmax = %d\t%d threads\n", max, omp_get_num_threads());
}

void ordered() {
    printf("\ne. Ordered:\nwithout ordered:\n");

    int i = 0;
    int n = 10;

    #pragma omp parallel shared(n) private(i)
    {
        #pragma omp for
        for (i = 0; i < n; i++) {
            printf("thread %d at index %d\n", omp_get_thread_num(), i);
        }
    }

    printf("\nwith ordered:\n");

    #pragma omp parallel shared(n) private(i)
    {
        #pragma omp for ordered
        for (i = 0; i < n; i++) {
            #pragma omp ordered
            {
                printf("thread %d at index %d\n", omp_get_thread_num(), i);
            }
        }
    }
}

int main() {
    barrier();
    master();
    single();
    critical();
    ordered();

    printf("\n");
    return 0;
}
```

Output:

```
gcc q1.c -o q1.out -fopenmp
./q1.out

a. Barrier:
sum: 18404

b. Master
without 'master':
hello, from thread 1
hello, from thread 2
hello, from thread 5
hello, from thread 6
hello, from thread 3
hello, from thread 0
hello, from thread 4
hello, from thread 7

with master:
hello, from thread 0
```

```
c. Single:
single: 1 | critical: 4
single runs once, critical is run once per thread

d. Critical:
a[0] = 184794536          thread no 1
a[1] = 388450127          thread no 1
a[2] = 915736906          thread no 1
a[3] = 101072999          thread no 1
a[4] = 659067697          thread no 1
a[5] = 1777483316         thread no 1
a[6] = 1906889260         thread no 1
a[7] = 113766839          thread no 1
a[8] = 111387570          thread no 1
a[9] = 1883555567         thread no 1

max = 1906889260          1 threads
```

```
e. Ordered:
without ordered:
thread 7 at index 9
thread 1 at index 2
thread 1 at index 3
thread 3 at index 5
thread 4 at index 6
thread 0 at index 0
thread 0 at index 1
thread 5 at index 7
thread 6 at index 8
thread 2 at index 4
```

```
with ordered:
thread 0 at index 0
thread 0 at index 1
thread 1 at index 2
thread 1 at index 3
thread 2 at index 4
thread 3 at index 5
thread 4 at index 6
thread 5 at index 7
thread 6 at index 8
thread 7 at index 9
```

Q2
Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <unistd.h>
#include <time.h>

#define MAX_SLEEP 10

omp_lock_t lock;
int cr = 0;

void reader(int i) {
    #pragma omp critical
    {
        cr++;
        if (cr == 1) {
            omp_set_lock(&lock);
            printf("lock set by reader %d \n", i);
        }
    }

    printf(
        "reader %d (on thread %d) is reading\n",
        i,
        omp_get_thread_num()
    );

    sleep(rand() % MAX_SLEEP);

    #pragma omp critical
    {
        cr--;
        if (cr == 0) {
            omp_unset_lock(&lock);
            printf("lock unset by reader %d\n", i);
        }
    }
}

void writer(int i) {
    omp_set_lock(&lock);
    printf("lock set by writer %d\n", i);

    printf(
        "writer %d (on thread %d) is writing\n",
        i,
        omp_get_thread_num()
    );

    sleep(rand() % MAX_SLEEP);
```

```
    omp_unset_lock(&lock);
    printf("lock unset by writer %d\n", i);
}

int main(int argc, char *argv[]) {
    printf("Readers-writers in parallel\n");

    srand(clock());
    omp_init_lock(&lock);

    #pragma omp parallel sections num_threads(8)
    {
        #pragma omp section
        {
            writer(0);
        }
        #pragma omp section
        {
            reader(0);
        }
        #pragma omp section
        {
            reader(1);
        }
        #pragma omp section
        {
            reader(2);
        }
        #pragma omp section
        {
            writer(1);
        }
        #pragma omp section
        {
            writer(2);
        }
        #pragma omp section
        {
            reader(3);
        }
        #pragma omp section
        {
            reader(4);
        }
    }

    return 0;
}
```

Output:

```
gcc q2.c -o q2.out -fopenmp
./q2.out
Readers-writers in parallel
lock set by writer 0
writer 0 (on thread 2) is writing
lock unset by writer 0
lock set by reader 1
reader 1 (on thread 3) is reading
reader 0 (on thread 0) is reading
reader 3 (on thread 1) is reading
reader 2 (on thread 5) is reading
reader 4 (on thread 7) is reading
lock unset by reader 1
lock set by writer 2
writer 2 (on thread 4) is writing
lock unset by writer 2
lock set by writer 1
writer 1 (on thread 6) is writing
lock unset by writer 1
```