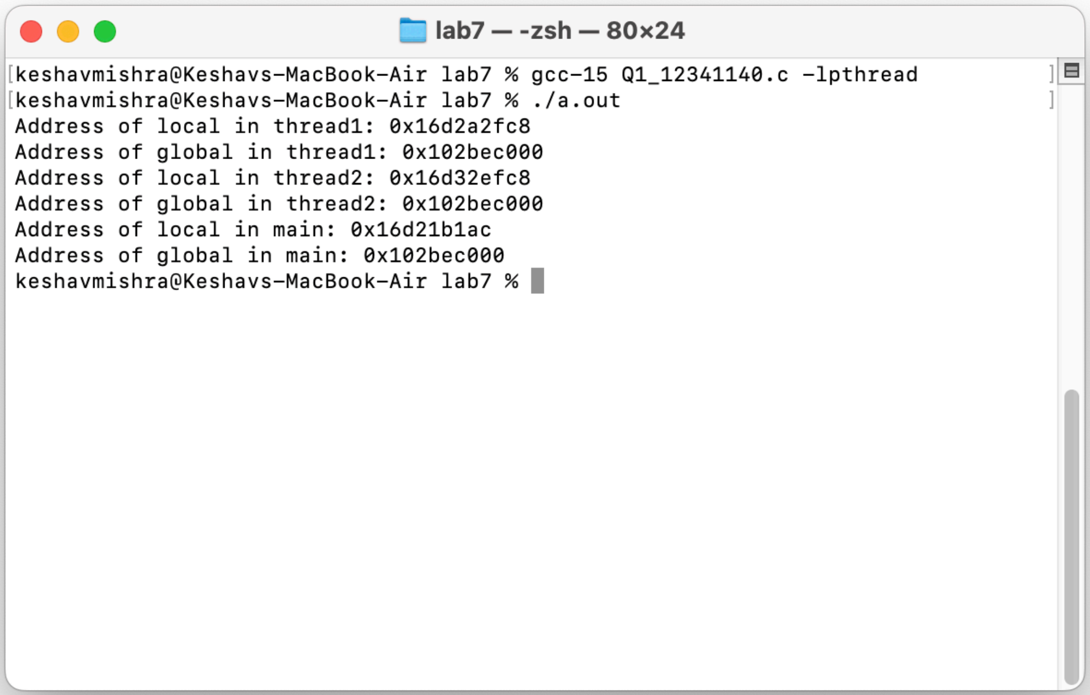
Q1

Code:
```
#include <pthread.h>
#include <stdio.h>

int global;
void* print_id1(void* arg){

   printf("Address of local in thread1: %p\n", (void*)&arg);
   printf("Address of global in thread1: %p\n", (void*)&global);
   return NULL;
}

void* print_id2(void* arg){
   printf("Address of local in thread2: %p\n", (void*)&arg);
   printf("Address of global in thread2: %p\n", (void*)&global);
   return NULL;
}

int main(){
   pthread_t tid1;
   pthread_t tid2;

   int local;

   pthread_create(&tid1,NULL,print_id1,&local);
   pthread_create(&tid2,NULL,print_id2,&local);
   pthread_join(tid1,NULL);
   pthread_join(tid2,NULL);
   printf("Address of local in main: %p\n", (void*)&local);
   printf("Address of global in main: %p\n", (void*)&global);
   return 0;
}
```

Screenshot of output:

}



```
keshavmishra@Keshavs-MacBook-Air lab7 % gcc-15 Q1_12341140.c -lpthread
keshavmishra@Keshavs-MacBook-Air lab7 % ./a.out
Address of local in thread1: 0x16d2a2fc8
Address of global in thread1: 0x102bec000
Address of local in thread2: 0x16d32efc8
Address of global in thread2: 0x102bec000
Address of local in main: 0x16d21b1ac
Address of global in main: 0x102bec000
keshavmishra@Keshavs-MacBook-Air lab7 %
```

Explanation:

Global variables have the same address across all threads because they reside in the shared data segment. Local variables have different addresses in each thread, including the main thread, because each thread has its own stack. Thus, threads share global memory but have separate local memory

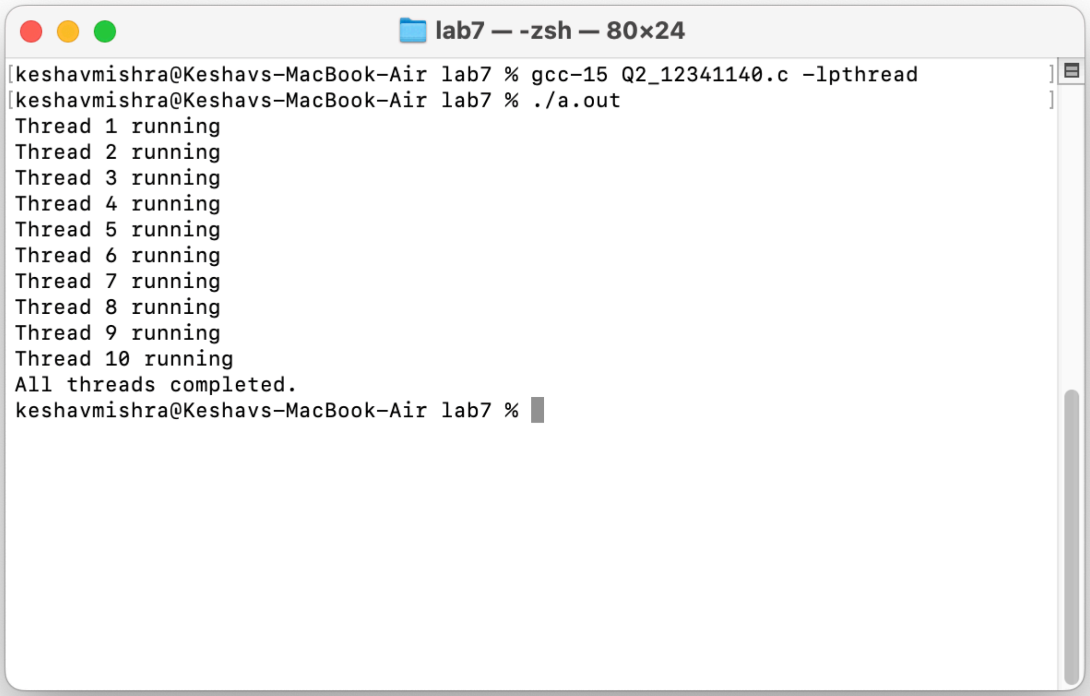
Q2

Code:

```
#include <pthread.h>

#include <stdio.h>

void* thread_func(void* arg) {

    int thread_num = *(int*)arg;

    printf("Thread %d running\n", thread_num);

    return NULL;

}
```

```
int main() {
   pthread_t threads[10];
   int thread_ids[10];
   for (int i = 0; i < 10; i++) {
      thread_ids[i] = i + 1;
      pthread_create(&threads[i], NULL, thread_func, &thread_ids[i]);
   }
   for (int i = 0; i < 10; i++) {
      pthread_join(threads[i], NULL);
   }
   printf("All threads completed.\n");
   return 0;
}
```

Screenshot of Output:



```
keshavmishra@Keshavs-MacBook-Air lab7 % gcc-15 Q2_12341140.c -lpthread
keshavmishra@Keshavs-MacBook-Air lab7 % ./a.out
Thread 1 running
Thread 2 running
Thread 3 running
Thread 4 running
Thread 5 running
Thread 6 running
Thread 7 running
Thread 8 running
Thread 9 running
Thread 10 running
All threads completed.
keshavmishra@Keshavs-MacBook-Air lab7 %
```

Explanation:

The order may not stay the same always, it may be different.

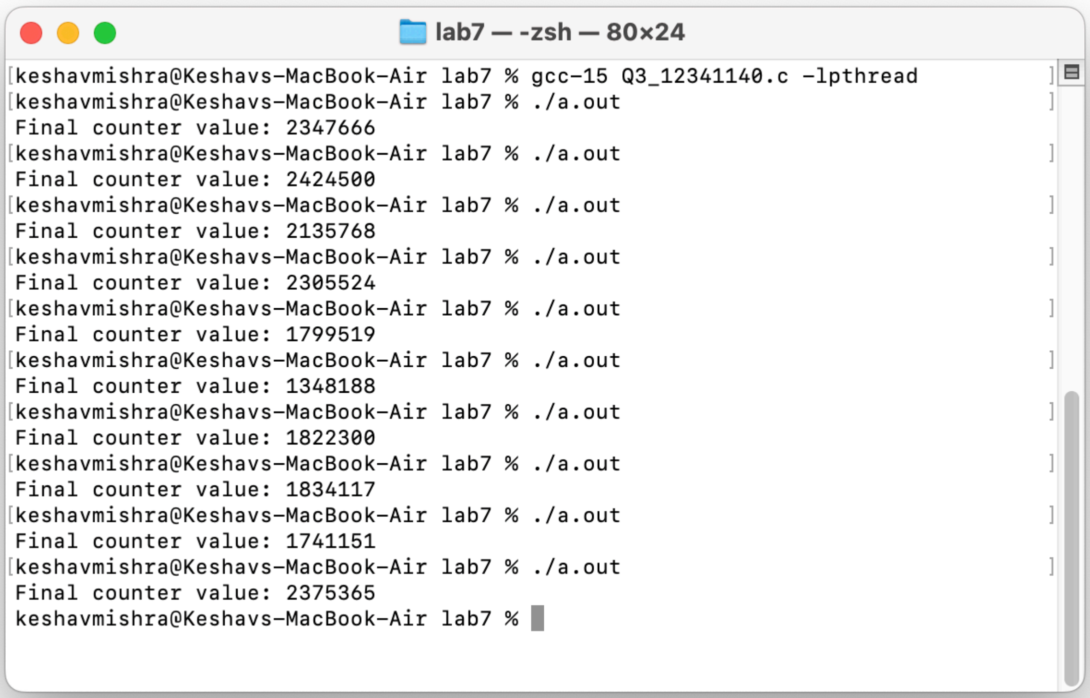
It is because the threads are concurrent processes and the order between them depends on scheduler and impossible for us to tell. Only pthread_join() ensures that main() will wait for threads to finish and then finish.

Q3

Code:

```
#include <pthread.h>

#include <stdio.h>

long long counter = 0;

void* increment_counter(void* arg) {

    for(int i = 0; i < 1e6; i++) {

        counter++;

    }

    return NULL;

}

int main() {

    pthread_t threads[10];

    for (int i = 0; i < 10; i++) {

        pthread_create(&threads[i], NULL, increment_counter, NULL);

    }

    for (int i = 0; i < 10; i++) {

        pthread_join(threads[i], NULL);

    }

    printf("Final counter value: %lld\n", counter);

    return 0;

}
```

Screenshot of Output:



```
keshavmishra@Keshavs-MacBook-Air lab7 % gcc-15 Q3_12341140.c -lpthread
keshavmishra@Keshavs-MacBook-Air lab7 % ./a.out
Final counter value: 2347666
keshavmishra@Keshavs-MacBook-Air lab7 % ./a.out
Final counter value: 2424500
keshavmishra@Keshavs-MacBook-Air lab7 % ./a.out
Final counter value: 2135768
keshavmishra@Keshavs-MacBook-Air lab7 % ./a.out
Final counter value: 2305524
keshavmishra@Keshavs-MacBook-Air lab7 % ./a.out
Final counter value: 1799519
keshavmishra@Keshavs-MacBook-Air lab7 % ./a.out
Final counter value: 1348188
keshavmishra@Keshavs-MacBook-Air lab7 % ./a.out
Final counter value: 1822300
keshavmishra@Keshavs-MacBook-Air lab7 % ./a.out
Final counter value: 1834117
keshavmishra@Keshavs-MacBook-Air lab7 % ./a.out
Final counter value: 1741151
keshavmishra@Keshavs-MacBook-Air lab7 % ./a.out
Final counter value: 2375365
keshavmishra@Keshavs-MacBook-Air lab7 %
```

Explanation:

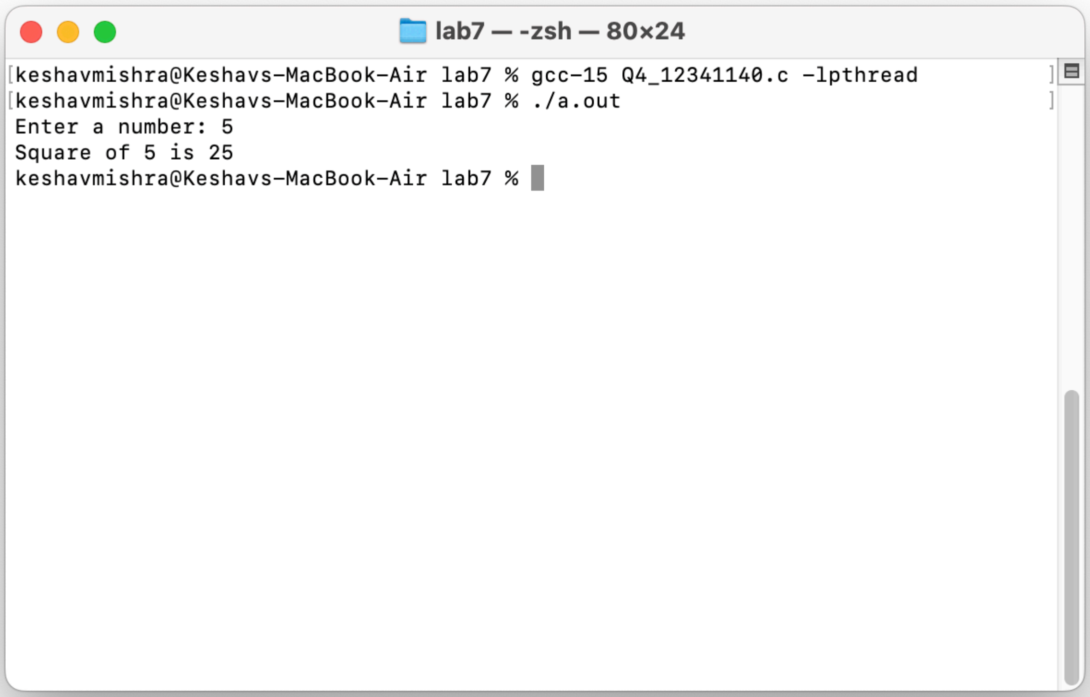
The final value differs because multiple threads are incrementing the shared counter without any synchronization. The operation is not atomic it involves reading the value, adding 1, and writing it back. If two threads perform these steps simultaneously, one increment can be lost, leading to a final value less than expected.
A race condition occurs when two or more threads access and modify shared data concurrently, and the program's outcome depends on the unpredictable order of execution. In this program, the race condition on counter causes some increments to be overwritten or lost, resulting in different outputs.

Q4

Code:

```
#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

void* compute_square(void* arg) {

    int num = *(int*)arg;

    int* square = malloc(sizeof(int));
    *square = num * num;
    return square;
}
int main() {
    pthread_t tid;
    int number;
    printf("Enter a number: ");
    scanf("%d", &number);
    pthread_create(&tid, NULL, compute_square, &number);
    void* result;
    pthread_join(tid, &result);
    int square = *(int*)result;
    printf("Square of %d is %d\n", number, square);
    free(result);
    return 0;
}
```

Screenshot of output:



```
keshavmishra@Keshavs-MacBook-Air lab7 % gcc-15 Q4_12341140.c -lpthread
keshavmishra@Keshavs-MacBook-Air lab7 % ./a.out
Enter a number: 5
Square of 5 is 25
keshavmishra@Keshavs-MacBook-Air lab7 %
```

Explanation:
This program creates a thread to compute the square of a number entered by the user. The thread returns the result via dynamically allocated memory, which the main function retrieves using pthread_join and prints. Finally, the dynamically allocated memory is freed to avoid memory leaks.

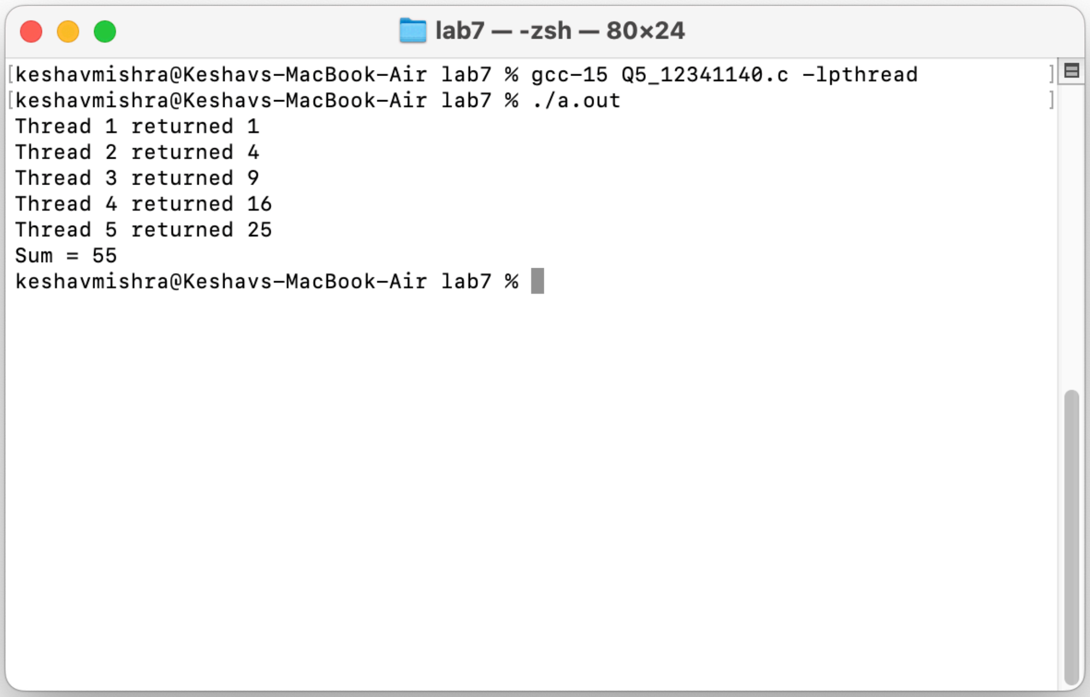
Q5

Code:

```
#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>



void* compute_square(void* arg){

   int thread_num = *(int*)arg;

   int* result = malloc(sizeof(int));

   *result = thread_num * thread_num;

   return result;

}

int main(){

   pthread_t threads[5];

   int thread_ids[5];

   void* thread_result;

   int sum = 0;

   for (int i = 0; i < 5; i++) {

      thread_ids[i] = i + 1;

      pthread_create(&threads[i], NULL, compute_square, &thread_ids[i]);

   }

   for (int i = 0; i < 5; i++) {

      pthread_join(threads[i], &thread_result);

      int square=*(int*)thread_result;
```

```
        printf("Thread %d returned %d\n",i+1,square);

        sum+=square;

        free(thread_result);

    }

    printf("Sum = %d\n",sum);

    return 0;

}
```

Screenshot of output:



```
keshavmishra@Keshavs-MacBook-Air lab7 % gcc-15 Q5_12341140.c -lpthread
keshavmishra@Keshavs-MacBook-Air lab7 % ./a.out
Thread 1 returned 1
Thread 2 returned 4
Thread 3 returned 9
Thread 4 returned 16
Thread 5 returned 25
Sum = 55
keshavmishra@Keshavs-MacBook-Air lab7 %
```

Explanation:
This program creates 5 threads, each computing the square of its thread number and returning it to the main thread.

The main thread collects the results, prints each square, and sums them up to display the total.