Lab 3

Q1.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *print_message_function(void *ptr);

int main() {
    pthread_t thread1, thread2;
    char *message1 = "Thread 1";
    char *message2 = "Thread 2";
    int iret1, iret2;

    /* Create independent threads each of which will execute function */
    iret1 = pthread_create(&thread1, NULL, print_message_function, (void*) message1);
    iret2 = pthread_create(&thread2, NULL, print_message_function, (void*) message2);

    /* Wait till threads are complete before main continues. */
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    printf("Thread 1 returns: %d\n", iret1);
    printf("Thread 2 returns: %d\n", iret2);

    exit(0);
}

void *print_message_function(void *ptr) {
    char *message;
    message = (char *) ptr;
    printf("%s\n", message);
    return NULL;
}
```

Q2.

The line `iret1 = pthread_create(&thread1, NULL, print_message_function, (void*) message1);` creates a new thread that executes the `print_message_function` with `message1` as its argument. The `&thread1` stores the thread's ID, `NULL` specifies default thread attributes, and `iret1` captures the return value of `pthread_create`, indicating success (`0`) or failure (error code). This allows the program to run `print_message_function` concurrently in a separate thread.