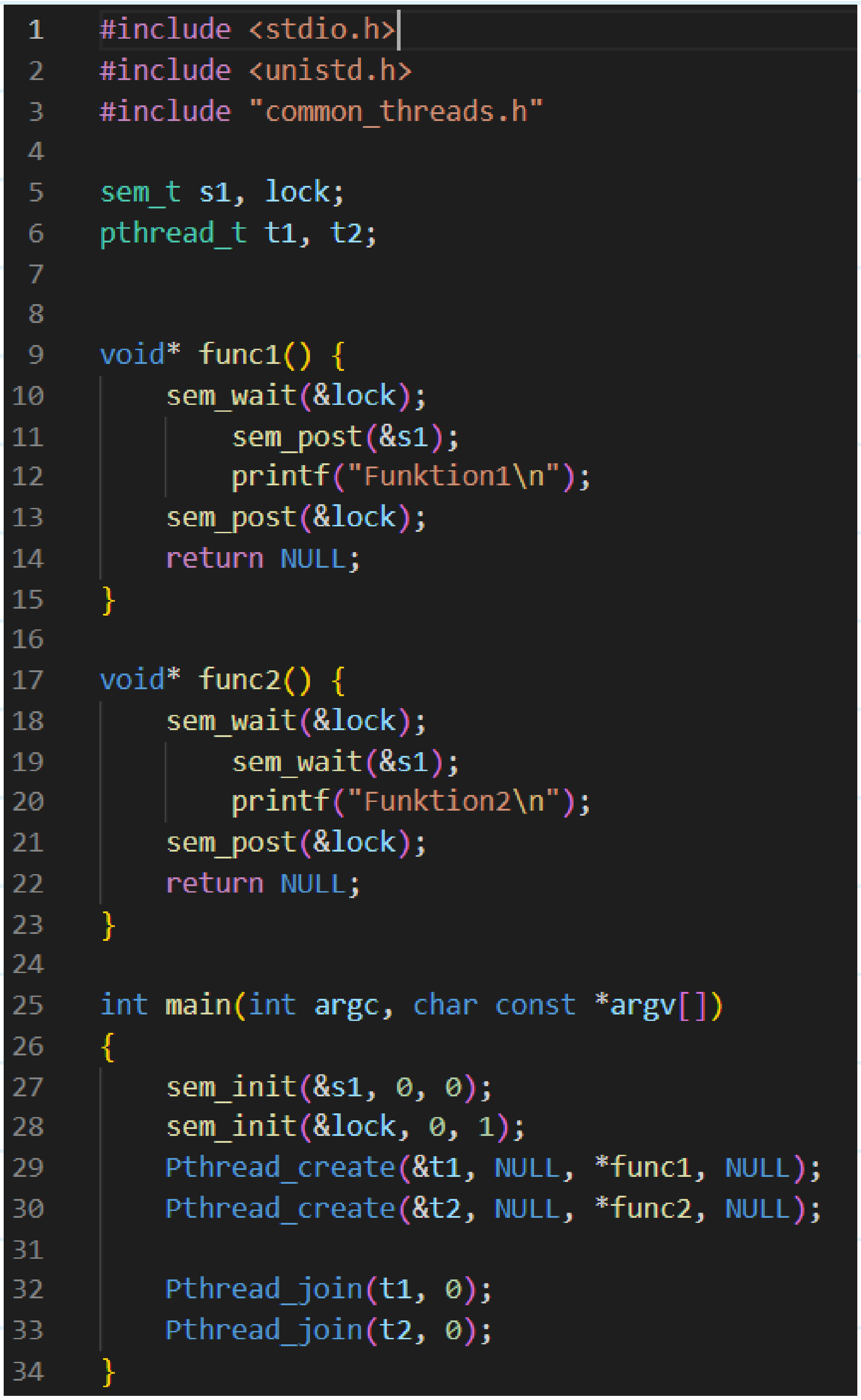
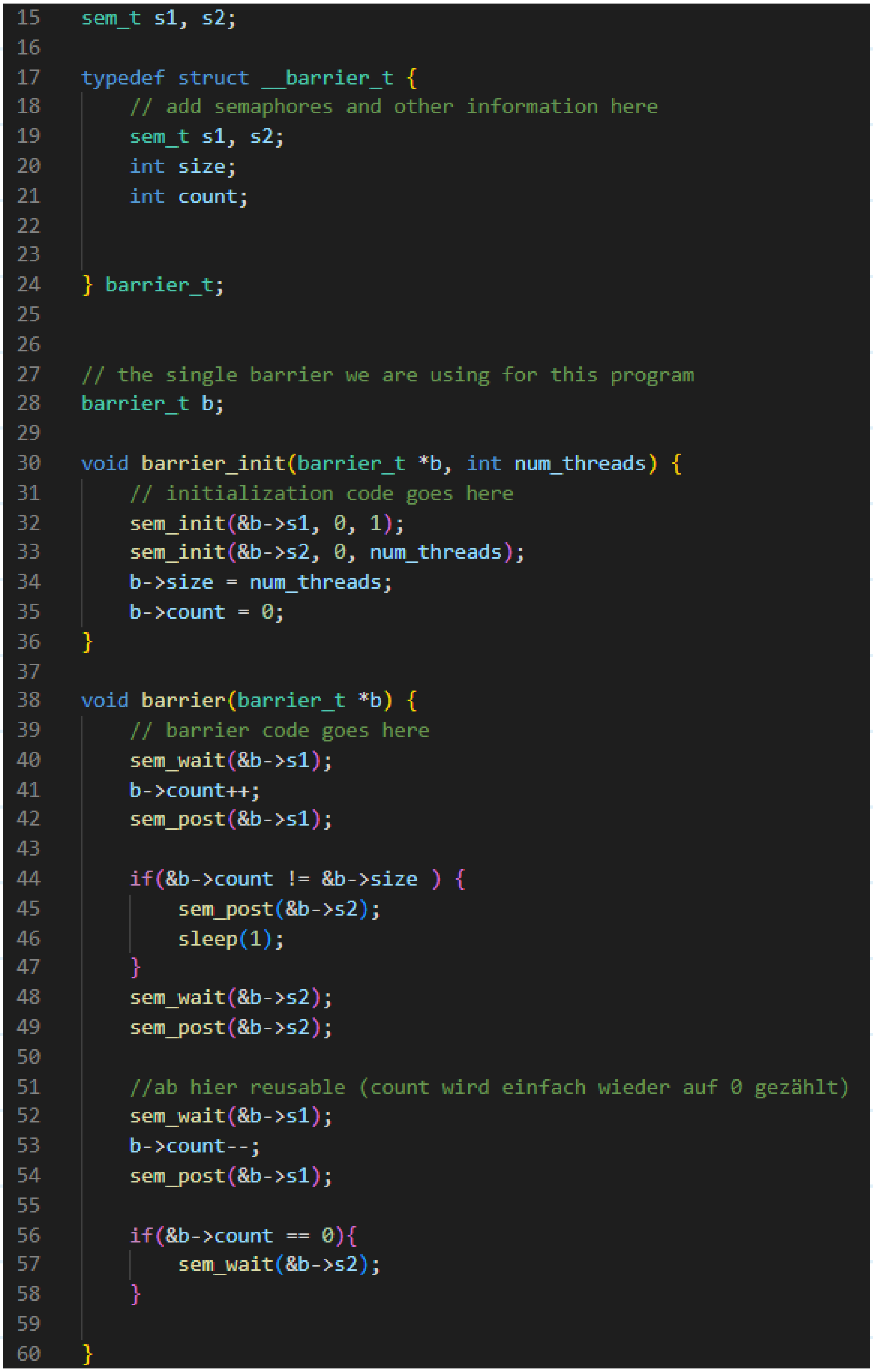
Semaphores



```c
#include <stdio.h>
#include <unistd.h>
#include "common_threads.h"

sem_t s1, lock;
pthread_t t1, t2;


void* func1() {
    sem_wait(&lock);
        sem_post(&s1);
        printf("Funktion1\n");
    sem_post(&lock);
    return NULL;
}

void* func2() {
    sem_wait(&lock);
        sem_wait(&s1);
        printf("Funktion2\n");
    sem_post(&lock);
    return NULL;
}

int main(int argc, char const *argv[])
{
    sem_init(&s1, 0, 0);
    sem_init(&lock, 0, 1);
    Pthread_create(&t1, NULL, *func1, NULL);
    Pthread_create(&t2, NULL, *func2, NULL);

    Pthread_join(t1, 0);
    Pthread_join(t2, 0);
}
```

Rendezvous

Barrier



```c
sem_t s1, s2;

typedef struct __barrier_t {
    // add semaphores and other information here
    sem_t s1, s2;
    int size;
    int count;


} barrier_t;


// the single barrier we are using for this program
barrier_t b;

void barrier_init(barrier_t *b, int num_threads) {
    // initialization code goes here
    sem_init(&b->s1, 0, 1);
    sem_init(&b->s2, 0, num_threads);
    b->size = num_threads;
    b->count = 0;
}

void barrier(barrier_t *b) {
    // barrier code goes here
    sem_wait(&b->s1);
    b->count++;
    sem_post(&b->s1);

    if(&b->count != &b->size ) {
        sem_post(&b->s2);
        sleep(1);
    }
    sem_wait(&b->s2);
    sem_post(&b->s2);

    //ab hier reusable (count wird einfach wieder auf 0 gezählt)
    sem_wait(&b->s1);
    b->count--;
    sem_post(&b->s1);

    if(&b->count == 0){
        sem_wait(&b->s2);
    }

}
```