```
/* ********************************************************
   **********************     Task 9    ********************
   ********************************************************  */

typedef void (*slave_func_t) (void);

// The functions to be executed by the workers.
static slave_func_t slave_funcs[5]={NULL,NULL,NULL,NULL,NULL};
// The number of available (non-working) worker threads.
static int available=5;

// Further synchronisation data structures.
/* Your work here. (1) */

// ---------------------------------------------------------------------

/** Wait until work is available, and execute the work function.
    \param id The id of the calling thread.
*/
static void wait_until_work(int id){
    /* Your work here. (2) */
}

/** Notify the master thread that it can assign work to a thread.
    \param id The thread which has finished its work.
*/
static void mark_as_ready(int id){
    /* Your work here. (3) */
}

/** The main thread function for the 5 slave threads.
    Branches as needed to slave functions.
    \param id_pointer Pointer to int, carrying an id (0..4).
    \return Irrelevant.
*/
static void* thread_func(void* id_pointer){
    int id=*(int*)(id_pointer);
    while(true){
        wait_until_work(id);
        mark_as_ready(id);
    }
    return NULL;
}

// ---------------------------------------------------------------------

/** Get the next function that should be assigned to a slave thread.
    Implemented somewhere else.
    \return The next function to be executed.
*/
slave_func_t get_next_func();

/** Assign work to a slave thread. Wait until one is available, if necessary.
    \param work The work to be assigned.
*/
static void set_next_work(slave_func_t work){
    /* Your work here. (4) */
}

/** The master thread, branching to the helper functions. */
static void master_thread(){
    while(true){
        slave_func_t next_func = get_next_func();
        set_next_work(next_func);
    }
}
```

```c
/* *********************************************************
   *********************   Aufgabe 9   ********************
   *********************************************************  */

typedef void (*slave_func_t) (void);

// Die Funktionen, die von den Arbeiter-Threads auszufuehren sind.
static slave_func_t slave_funcs[5]={NULL,NULL,NULL,NULL,NULL};
// Anzahl der zur Verfuegung stehenden (nicht arbeitenden) Arbeiter-Threads.
static int available=5;

// Weitere Synchronisationsvariablen.
/* Hier vervollstaendigen. (1) */

// ------------------------------------------------------------------

/** Warte, bis eine Arbeitsfunktion zugewiesen wurde, und fuehre sie aus.
    \param id Die id des aufrufenden Threads.
*/
static void wait_until_work(int id){
    /* Hier vervollstaendigen. (2) */
}

/** Benachrichtige den Master-Thread, dass Arbeiter-Thread eine Arbeit beendet hat.
    \param id Der Thread, der nun verfuegbar ist.
*/
static void mark_as_ready(int id){
    /* Hier vervollstaendigen. (3) */
}

/** Hauptfunktion der 5 Arbeiter-Threads.
    Verzweigt wie vorgegeben zu den Hilfsfunktionen.
    \param id_pointer Zeiger auf int mit einer ID (0..4).
    \return Irrelevant.
*/
static void* thread_func(void* id_pointer){
    int id=*(int*)(id_pointer);
    while(true){
        wait_until_work(id);
        mark_as_ready(id);
    }
    return NULL;
}

// ------------------------------------------------------------------

/** Hole Zeiger auf die Funktion, die als naechstes auszufuehren ist.
    Ist extern implementiert.
    \return Die als naechstes auszufuehrende Funktion.
*/
slave_func_t get_next_func();

/** Uebergebe eine Funktion an einen Arbeiterthread.
    Wartet, falls noetig, bis einer frei ist.
    \param work Die auszufuehrende Funktion.
*/
static void set_next_work(slave_func_t work){
    /* Hier vervollstaendigen. (4) */
}

/** Der Master-Threads. Verzweigt wie vorgegeben zu den Hilfsfunktionen. */
static void master_thread(){
    while(true){
        slave_func_t next_func = get_next_func();
        set_next_work(next_func);
    }
}
```