

MARMARA

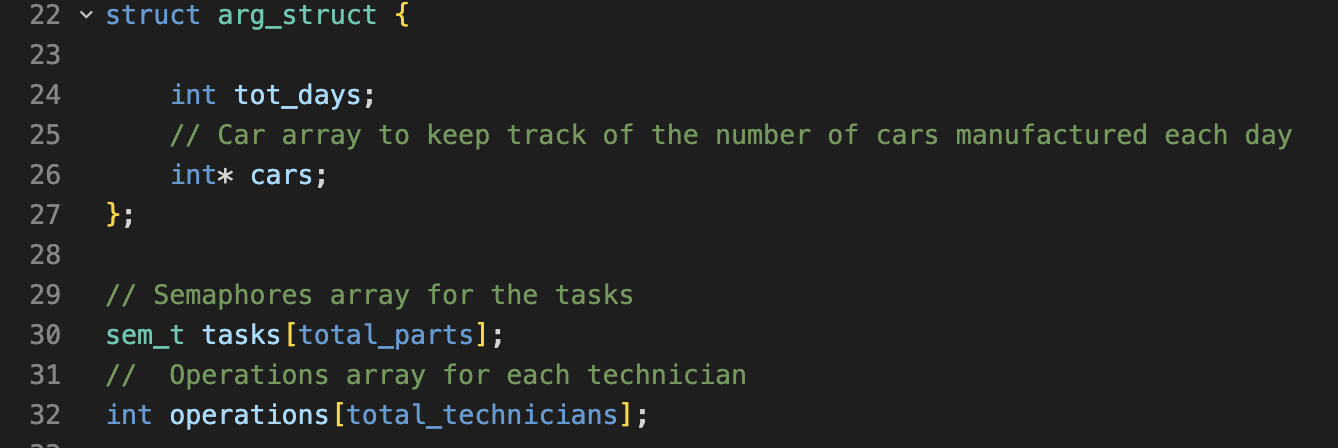
UNIVERSITY

Course Project Report

Marmara University, Advanced Unix Programming

Fatih Genç - 150119905

**Define section:**

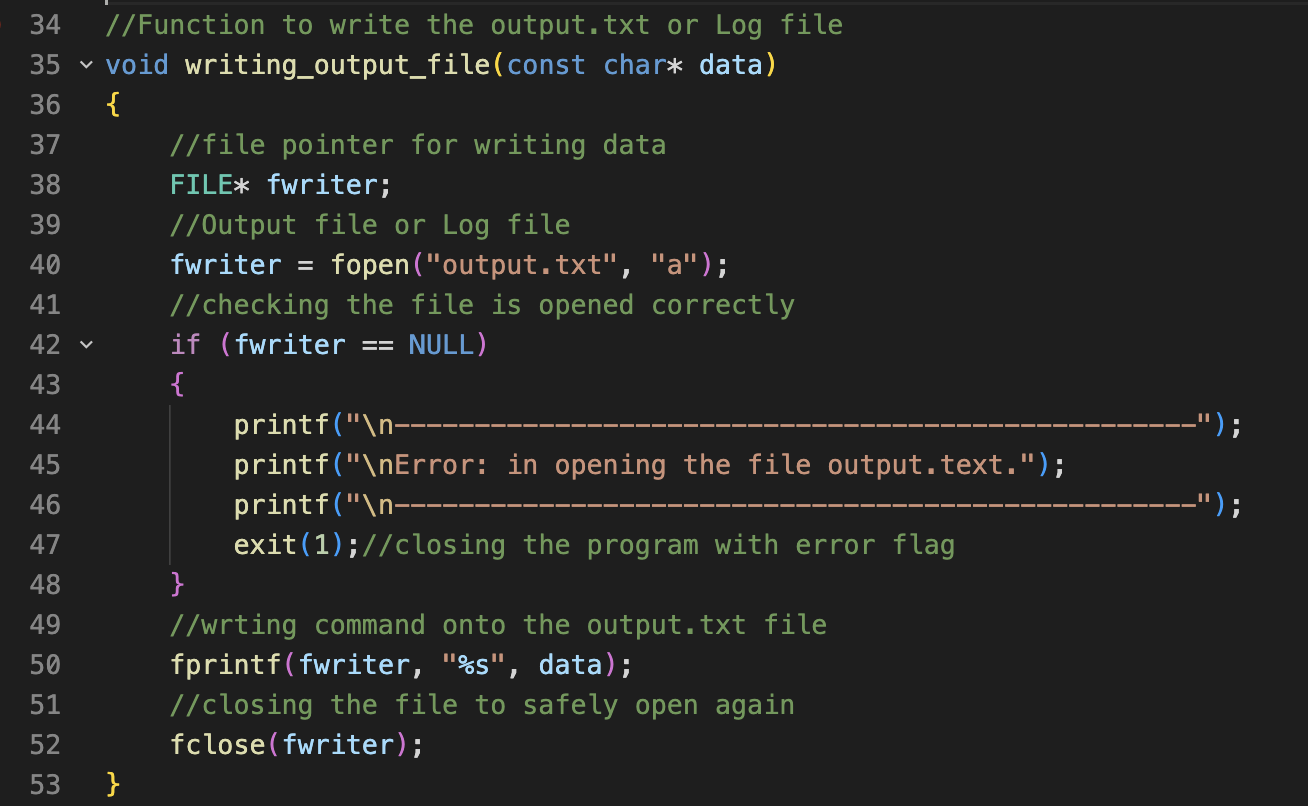


arg\_struct has two members:

**tot\_days**, an integer variable that likely holds the number of days that the program will run or it could be the number of days for which the manufacturing details are being recorded.

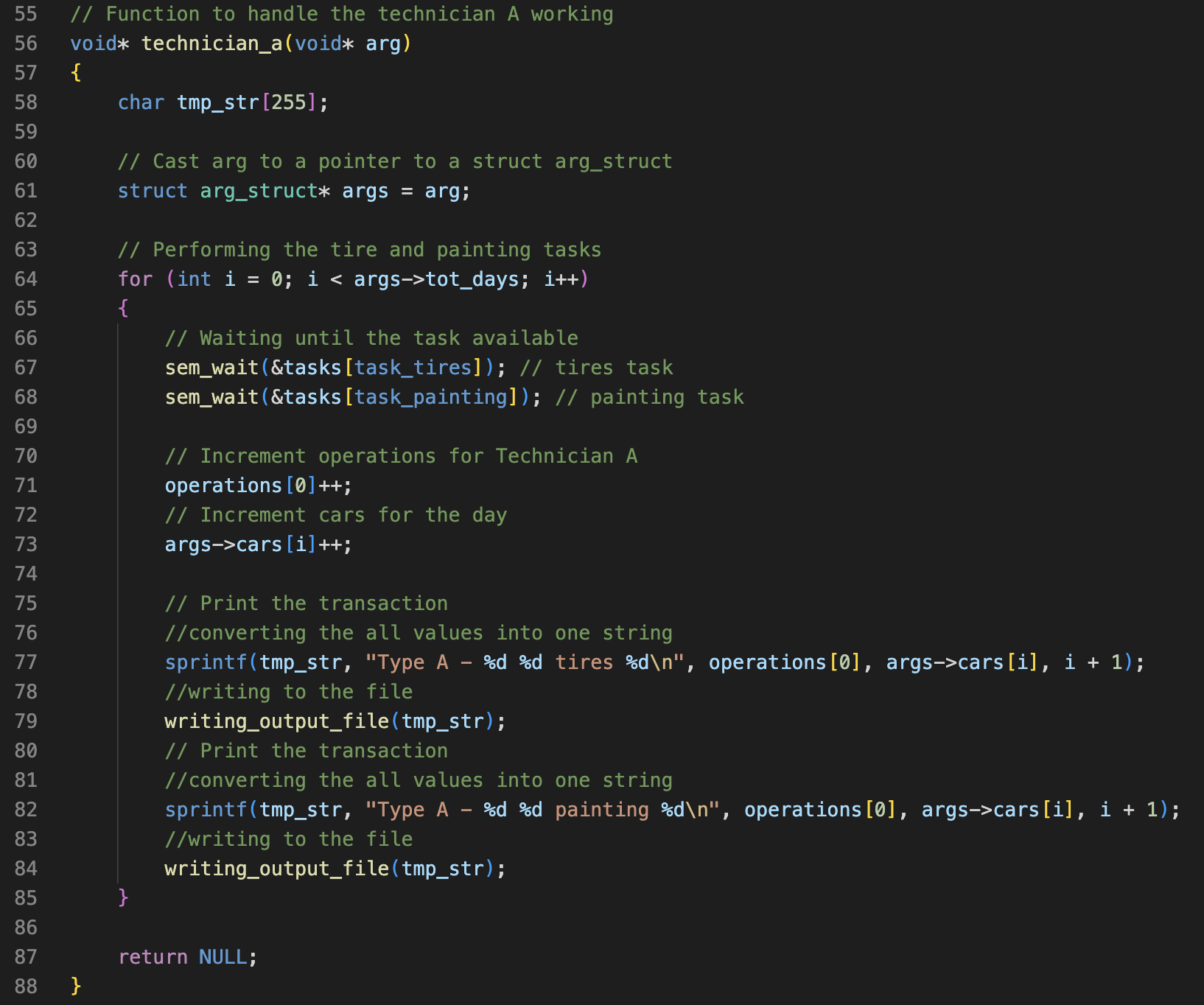
**cars**, a pointer to an integer, that is likely being used to keep track of the number of cars manufactured each day. It's an array of size tot\_days.

**Output File:**

****

This code defines a function that opens the "output.txt" file in append mode, and write the passed data, it also check for any error during file opening, and exit the program if not able to open file, and close the file after writing the data.

**Define a function technician\_a:**

The function begins by defining a temporary string variable tmp\_str with a maximum length of 255 characters. Then, it takes the argument passed to the function, which is a pointer to a struct arg\_struct, and assigns it to a variable named args.

It then enters a loop that runs for **args->tot\_days** iterations, on each iteration:

The function decrements the semaphore value of the **tasks[task\_tires]** by 1 using **sem\_wait**, which is a semaphore function that tells the calling thread to wait until the semaphore value is greater than zero.

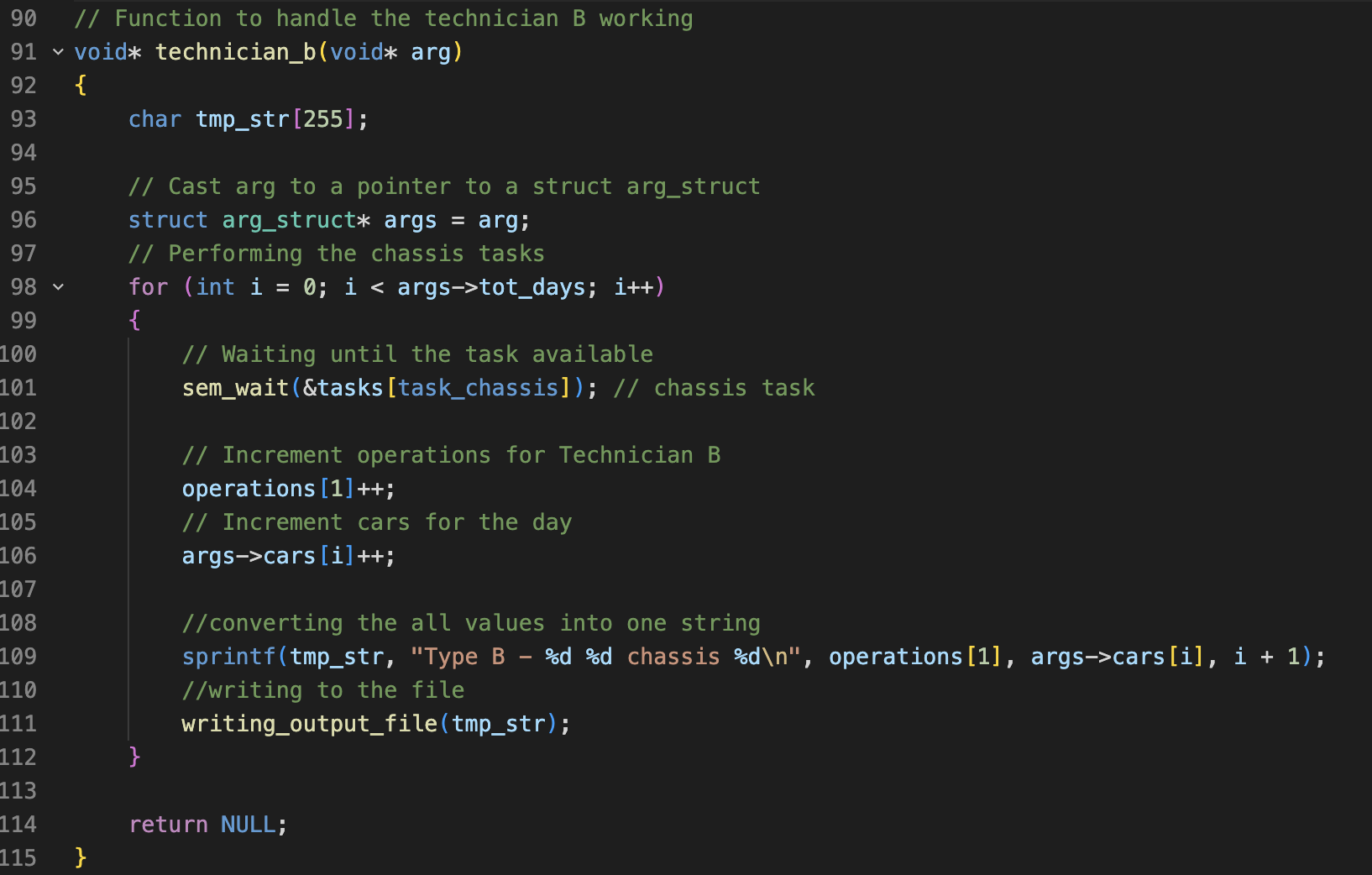
The function decrements the semaphore value of the **tasks[task\_painting]** by 1 using **sem\_wait**, which is a semaphore function that tells the calling thread to wait until the semaphore value is greater than zero.

The **operations[0]** value is incremented by one, possibly indicating that an operation has been completed The value of **args->cars[i]** is incremented by one.

Then, the function creates a string **tmp\_str** by using **sprintf** function, which formats the string and writes the output to the variable **tmp\_str** and calling **writing\_output\_file** function with **tmp\_str** as the argument. This output string is likely to be a log message indicating that a technician has performed a task, such as "Type A - 1 1 Tires 1”.

Finally, another sprintf and writing output file called with similar format as before.

**Define a function technician\_b:**



The function begins by defining a temporary string variable tmp\_str with a maximum length of 255 characters. It then casts the input argument arg to a pointer to a struct called arg\_struct and assigns it to the variable args.

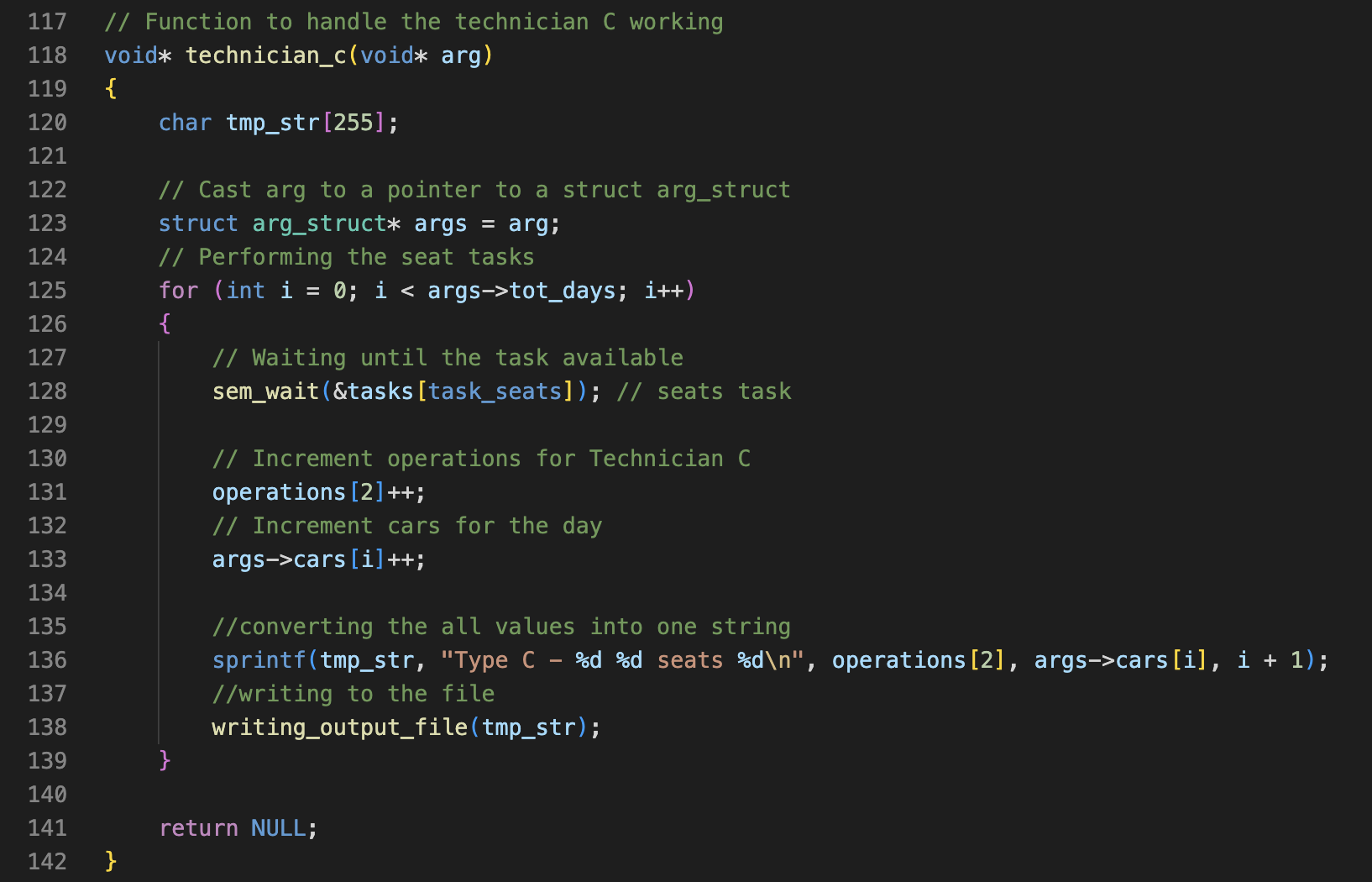
The function then enters a for loop that iterates **tot\_days** times. On each iteration of the loop, it calls the **sem\_wait** function on the **tasks[task\_chassis]** semaphore. This semaphore is used as a synchronization mechanism to ensure that multiple threads are not accessing the same resource simultaneously.

The function then increments the **operations[1]** variable, which is presumably being used to keep track of the number of operations that the technician has performed. It also increments the **cars[i]** variable, which is presumably being used to keep track of the number of cars that have been serviced on the **i**-th day.

The function then creates a string by using the sprintf function, which formats the values of **operations[1], cars[i]**, and i + 1 into the **tmp\_str** variable. It ends by writing the **tmp\_str** variable to a file by calling the **writing\_output\_file** function. The function then exits.

This function is meant to run in a separate thread and work on the "chassis" task, incrementing operation and car counters, and writing the output to a file, which could be useful for a log or any future analysis.

**Define a function technician\_c:**



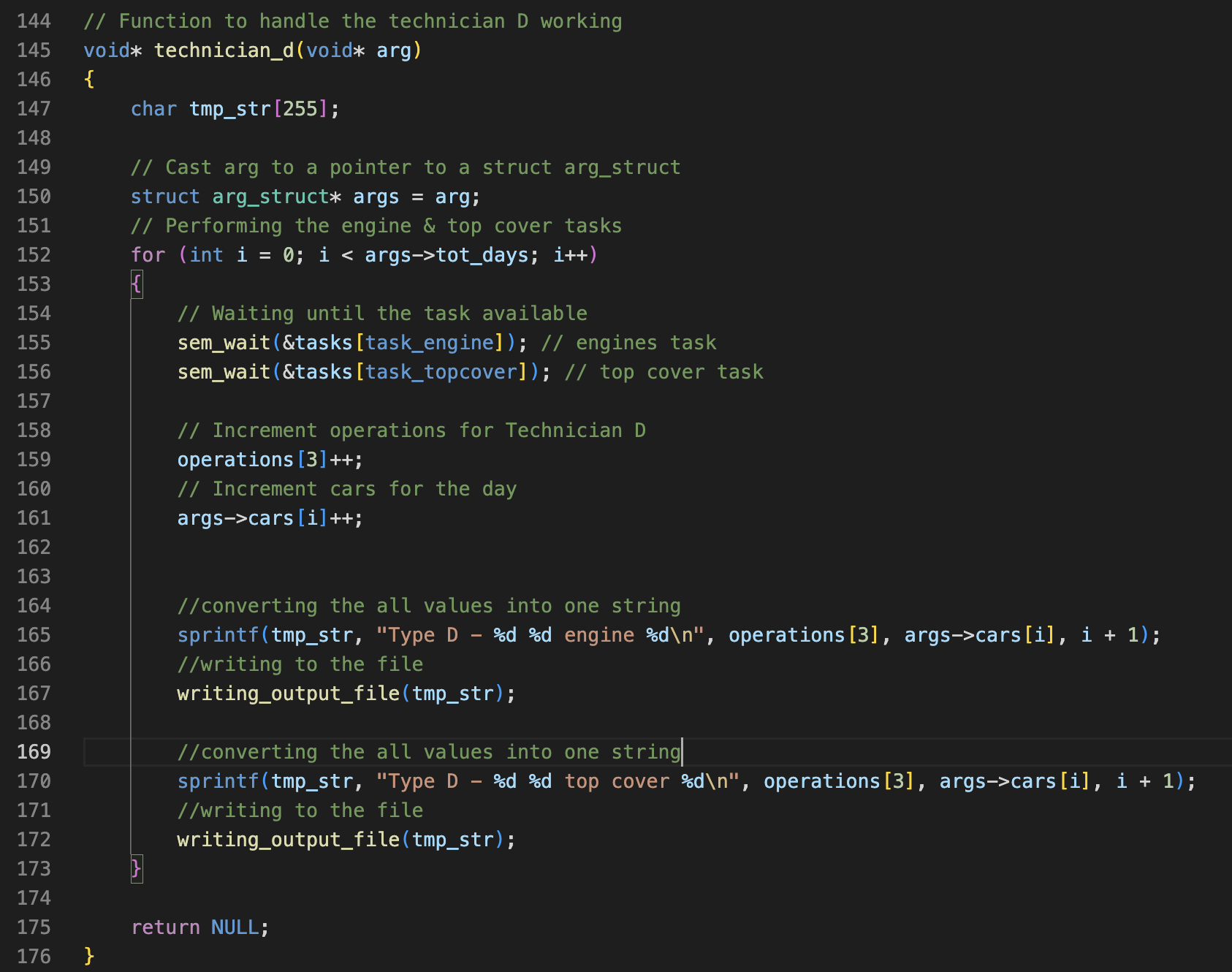
It starts by defining a character array called **tmp\_str** of size 255 and then casts the input argument arg to a pointer to a struct called **arg\_struct** and assigns it to the variable args.

The function then enters a for loop that iterates **tot\_days** times. On each iteration of the loop, it calls the **sem\_wait** function on the **tasks[task\_seats]** semaphore. Like the previous function, this semaphore is used as a synchronization mechanism to ensure that multiple threads are not accessing the same resource simultaneously.

The function then increments the **operations[2]** variable, which is presumably being used to keep track of the number of operations that the technician has performed. It also increments the **cars[i]** variable, which is presumably being used to keep track of the number of cars that have been serviced on the **i**-th day.

The function then creates a string by using the sprintf function, which formats the values of **operations[2]**, **cars[i]**, and **i** + 1 into the **tmp\_str** variable. It ends by writing the **tmp\_str** variable to a file by calling the **writing\_output\_file** function. The function then exits.

This function is meant to run in a separate thread and work on the "seats" task, incrementing operation and car counters, and writing the output to a file, which could be useful for a log or any future analysis.

**Define a function technician\_d:**

It starts by defining a character array called **tmp\_str** of size 255 and then casts the input argument **arg** to a pointer to a struct called **arg\_struct** and assigns it to the variable **args**.

The function then enters a for loop that iterates **tot\_days** times. On each iteration of the loop, it calls the **sem\_wait** function on the **tasks[task\_engine]** and **tasks[task\_topcover]** semaphores. Like the previous function, these semaphores are used as a synchronization mechanism to ensure that multiple threads are not accessing the same resource simultaneously.

The function then increments the **operations[3]** variable, which is presumably being used to keep track of the number of operations that the technician has performed. It also increments the **cars[i]** variable, which is presumably being used to keep track of the number of cars that have been serviced on the **i**-th day.

The function then creates a string by using the sprintf function, which formats the values of **operations[3]**, **cars[i]**, and **i + 1** into the **tmp\_str** variable, it also writes the **tmp\_str** variable to a file by calling the **writing\_output\_file** function twice, once for the engine task and once for the top cover task, as this technician is working on both tasks.

This function is meant to run in a separate thread and work on the "engine" and "top cover" tasks, incrementing operation and car counters, and writing the output to a file, which could be useful for a log or any future analysis.

Int Main Function:

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

The first thing the program does is some important declarations. It initializes an integer "i" to 0, declares an array of characters called "tmp\_str" with a capacity of 255, declares four more integers "pt\_a", "pt\_b", "pt\_c", "pt\_d", and an integer "tot\_days", and an integer array "ops\_per\_day" which is initialized with total\_parts.

Then, it uses the remove function to delete an existing file called "output.txt". This is done in case the program has been run previously and an old output file needs to be overwritten.

The program then opens an input file called "input.txt" in read mode and checks if the file was opened successfully. If the file could not be opened, the program prints an error message and exits with an error flag.

If the file was opened successfully, the program reads the number of technicians of each type (A, B, C, D) and the total number of days the simulation should run for from the input file using the fscanf function. Then, it reads the number of operations per day for each task, using a for loop and the fscanf function. Finally, the program closes the input file.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

The next section of the code initializes a struct called "arg\_struct" which contains two members: "tot\_days" and "cars". It sets "tot\_days" to the value read from the input file, and initializes the "cars" member as a pointer to an array of integers using malloc function with the size of tot\_days.

It also initializes the "operations" array to 0, using a for loop and sets the "cars" array to 0 as well.

Then it initializes the semaphores for every task by iterating over the array "tasks" and using the sem\_init function for each element of the array. The first argument to sem\_init is the address of the semaphore, the second argument is 0, and the third argument is the number of operations per day for the corresponding task.

Then the function writes a header row to the output file "output.txt" using a function called "writing\_output\_file".

The program creates a thread for each technician and runs the simulation for a certain number of days (equal to tot\_days), using a while loop. The loop creates a thread for each technician (A, B, C, D) using the pthread\_create function. And it passes the address of the struct arg\_struct as an argument to each thread.

It then uses the sleep function to pause the program for 3 seconds(representing 1 day) before continuing the loop. The loop will continue until the total number of days specified in the input file have been reached.

It also prints the current day that passed

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

After all the threads have completed execution, the program will print the number of operations performed by each technician using a for loop. It uses the sprintf function to convert the values into a single string, tmp\_str. Then the program uses the "writing\_output\_file" function to write the values of tmp\_str to the output file.

It also prints the number of cars manufactured each day by iterating over the "args.cars" array and using the same method, converting the values into a single string, tmp\_str and writing it to the output file.

It will then print a message indicating that the program has ended successfully and the user can check the output file for the results, and the program returns 0, indicating that it has completed execution without any errors.

**Example run:**

