Candidate number: 138086

Code Comments

You have split your code into two modules with associated header files. Note that the #endif needs to go at the very end of the header file, otherwise you're going to have multiple declarations of your function prototypes, for example. Note also that in a function prototype, you do not have to include the variable names, just the declaration of type. In runOneSimulation.c, this starts with a whole lot of stuff that really ought to be in the header file, e.g. the declaration of the node type, definition of the light structure, the vehicle structure and all those other #includes should really go in runOneSimulation.h. When you malloc memory to a pointer, you verify that it was successful. If it is not successful, you return a one as the function value. But if it is successful, if this thing runs, then you don't return any value, which you must. And in fact, you don't check that return values after you have called the function. What you should do is you should produce an error message that goes to a stderr and you should exit the program with an error status code. You have a useful comment at the head of each function saying roughly what it does. Again, in remove first node, you return a value of one essentially as some sort of error, but you don't return any other value if it does not. And that I suspect that means you're just not testing it, which in this particular case is fine. You do free up items that were removed from the queue. Nice function to update the lights. You're using the GSL random number generator, but you are seeding it every single time you call for a random number. This program probably runs so quickly that you are always getting the same number every single time you call it. Again, you check the malloc was successful, but really, rather than returning an empty instance, you want to kill your program because at this point it probably cannot do anything else because I haven't got enough memory to do anything. Looking at the main while loop, if you have gone more than 500 iterations, you check to see if the queues are empty and if they are, then you break. It would probably quite a lot easier just to put this in the while condition. Checks if the lights need changing, and if they don't, add to the left and right and remove from the left and right. That is a neat piece of code, that is good. By the time you come out of the while loop, your queues should be empty, and therefore you can just free them, which you have, and you then return your various values to the calling function. In the lectures, I reiterated multiple times that it is a very strong convention, that the main function should always come first. Looking at the main function, before you look at argv, you need to check the argc is telling you that you have sufficient values passed in. Otherwise these values could be complete rubbish. So you run your simulations. You update your results and then you return those to the main function. You have got a separate function called display that outputs all the value, that is very neat. I like that. I am fairly sure that your calculations of averages in update res function are not right. If you have had 98 runs and you're adding in another run, you cannot average them by adding them together and dividing them by two. What you have to do is: (oldAverage*98 + newValue)/99. Anyway, a neatly laid out program in a slightly obscuring format.

Report

Well done on getting rid of any memory leaks (and putting the valgrind result in the report!). A nice set of experiments with good use of graphs. Please do not include your userid in the document (page 4, directory name) as this is supposed to be anonymous marking.

Marks: 78%