Team Embedded Code Review

**Team Name: Cybot Eats Date: 4/30/19**

**Team members participating in the code review**:

Daniel Nikolic and Rithwik Gokhale

**Checklist selected for the code review:**

**➩ Peer Review Checklist: Embedded C Code (P. Koopman, ECE CMU, 2018)**

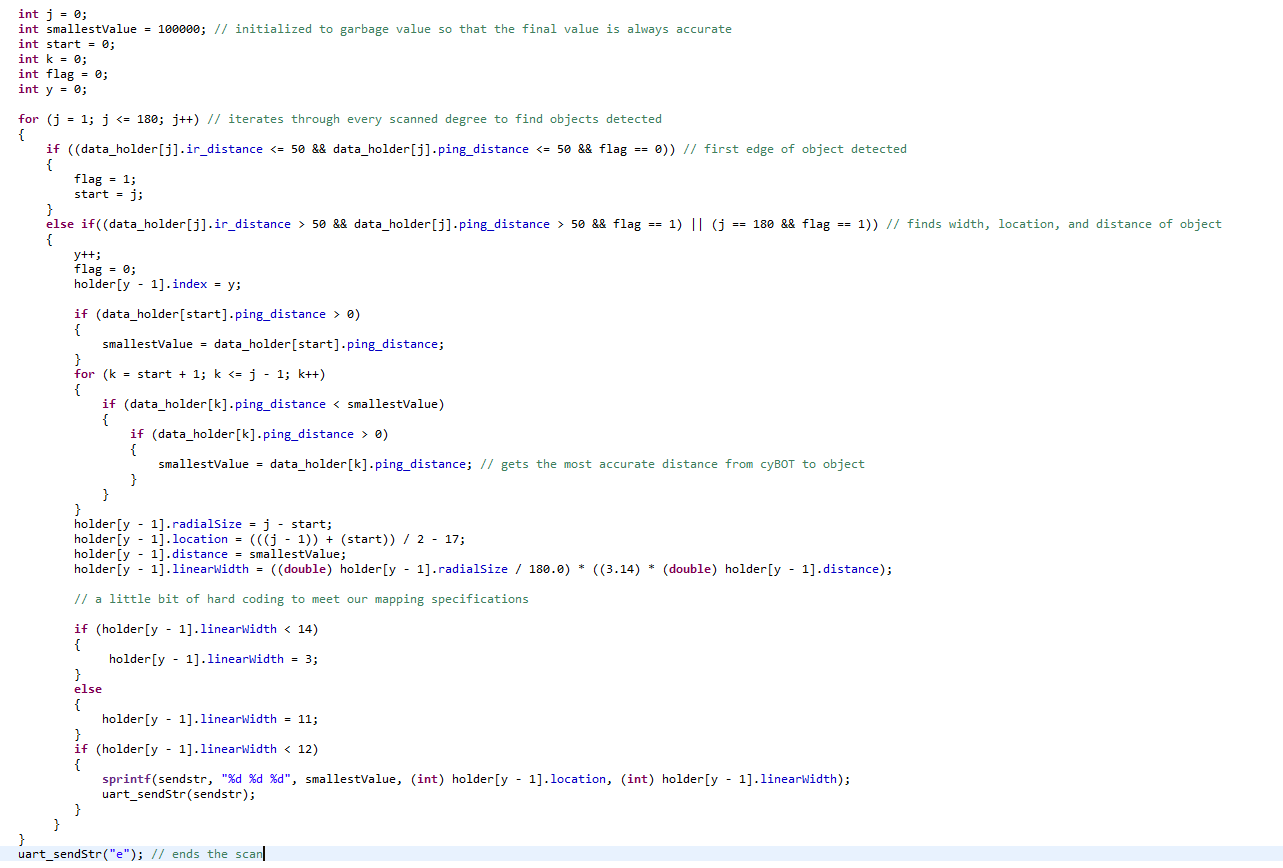
Embedded System Code Review Checklist (G. Khattak & P. Koopman, July 2012)

These checklists are items 3 and 4 in the Code Review resources for this assignment:

* 1-Better Embedded System SW-Embedded System Code Review Checklist.pdf
* 2-PeerReview-selected-slides-CMU.pdf
* 3-peer\_review\_checklist-CMU.pdf
* 4-code\_review\_checklist\_v1\_010.pdf

1. Select 30-40 lines of code developed for your project. Paste/insert/attach the code (or a screenshot of the code) below.

**CODE ON NEXT PAGE**



1. From your selected checklist, identify *perspectives* taken by team members and several specific rules

applied during the code review. **Choose two or three rules focused on per team member** given the limited time available for a short mock review session. Refer to items 1 and 2 in the Code Review resources to learn about perspectives.

Daniel: control flow issues

Rithwik: style issues

Daniel: error handling, no globals - proper scopes, unit testing

Rithwik: commenting, checking for orphans, consistent style

1. Record any issues found related to the specific rules considered.

Daniel: Error handling within the k++ for loop is not perfect because sometimes the smallest value is not accurately pinpointed. The error detection towards the bottom of the code where we hard code the widths works really well with our setup. Moving along, we use no global variables in our code. Most of the variables are stored in structs. Finally, unit testing was useful with this snippet of code. We would usually run in it java to see how our fixes adjust the scan accuracy.

Rithwik: Looking at the snip of code which is in the previous page, one can see all the comments which are present in that sample code. While it is important to have comments in the program, it should be precise and accurate and too much code can also make it more complicated to understand the project. In our snippet of code, it can be seen that there are precise lines of comments which explains the different loops and the if else statements and their purpose. Furthermore, it can be seen that comments have also been written for variables and lines of code which were written to initialize the garbage values so that the final value is accurate.

With regards to orphans in the code, i believe that the above snippet is an accurate representation for the lack of orphans in our program. It can be seen that all the variables which were initialized at the beginning have been used throughout the program. All the lines of code which were temporarily commented out during the project were deleted to ensure that the code looked neat and was easy to understand. Most importantly, it can be seen that all the loops and if-else statements which are present in the code have been written in a manner where they are checking multiple conditions in the same line to avoid repeating code. This will reduce the amount of code which has to be written and furthermore, it is also easier to understand.

Finally, with regards to consistent styling, it can be seen that a lot of attention has been given to the formatting of the code such as bracket and the location where each of the variables were initialized. It is seen here that all the variables where initialized before beginning the program and not in the middle of different lines of codes. So all the global variables were initialized at first. this is important so that the reader can easily understand the code. Sufficient attention was also given to the use of brackets. Each loop and if-else statement had clear brackets to quickly understand where the loops start and end. All these contribute to consistent styling and this will all make it easier to read and understand the code while working on the project but also as a third party reader.