Project 10, Program Design

1. (70 points) Modify project 9 by adding and modifying the following functions:

   1) Add a delete function in `request.c` that delete a request. The function should delete a request from the list by classroom and child's first and last name. Classroom and name will be entered by the user. The function should have the following prototype:

      ```
      struct request* delete_from_list(struct request *list);
      ```

      Don't forget to add the function prototype to the header file; modify the main function in `waiting_list.c` to add `'d'` for delete option in the menu and it calls the delete function when the `'d'` option is selected.

   2) Modify the append_to_list function so a request is inserted into **an ordered list** by classroom and then the child's last name and the list remains ordered after the insertion. For example, a request for `Ashley Meyers` in `Infants` classroom should be before a request for `Justin Lewis` in `Toddlers` in the list; a request for `Elizabeth Winfield` in `Toddlers` classroom should be after a request for `Ashely Meyers` in `Toddlers` classroom in the list. The order of the requests does not matter if multiple requests have the same classroom and last names.

2. (30 points) Library function `qsort` uses quick sort algorithm to sort an array of any type. Modify project 7 so that it uses quick sort to sort the array to find out the top 5 products. The file input and output should stay the same. Instead of a sorting function you wrote for the program, use the quick sort library function and implement the comparison function.

3. (Extra Credit: 20 points) Modify the append_to_list function in request.c. A request is inserted into an ordered list also by first name after being ordered by classroom and last name. For example, a request for `Elizabeth Meyers` in `Toddlers` classroom should be after a request for `Ashely Meyers` in `Toddlers` classroom in the list.

   The extra credit part should be kept in separate files and a separate folder from the other files for project 10.

Total points: 100 (70 points for part 1 and 30 points for part 2)

1. A program that does not compile will result in a zero.
2. Runtime error and compilation warning 5%
3. Commenting and style 15%
4. Functionality 80%:
   a. **Implementation meets the requirement.**
   b. **Using the malloc and free function properly.**

Before you submit

1. (part 1) Compile with makefile. Be sure it compiles on ***student cluster*** with no errors and no warnings.

2. (part 1) Test your program with script *try_list2*. *try_list2* assumes the executable is named *waiting_list.*

   chmod +x try_list2
   ./try_list2

3. (part 2) Compile your program with the following command:

   gcc –Wall top5_products_2.c

4. (part 2) Test your program.

   chmod +x try_top5
   ./try_top5

5. Your source files should be read & write protected. Change file permission on Unix using chmod 600.

6. Submit all the source files, header files, and makefile for part 1 and *top5_products_2.c* and *fruits_vegetables.txt* (for grading purpose) for part 2 on Canvas.

(Extra Credit: 20 points) Before you submit:

1. Test your program with script. *try_list3* assumes the executable is named *book_requests*

   chmod +x try_list3
   ./try_list3

2. Your source files should be read & write protected. Change file permission on Unix using chmod 600.

3. Submit all the source files, header files, and makefile for the extra credit problem in a **separate** zipped folder under project 10 submission.

Programming Style Guidelines

The major purpose of programming style guidelines is to make programs easy to read and understand. Good programming style helps make it possible for a person knowledgeable in the application area to quickly read a program and understand how it works.

1. Your program should begin with a comment that briefly summarizes what it does. This comment should also include your **name**.
2. In most cases, a function should have a brief comment above its definition describing what it does. Other than that, comments should be written only *needed* in order for a reader to understand what is happening.
3. Information to include in the comment for a function: name of the function, purpose of the function, meaning of each parameter, description of return value (if any), description of side effects (if any, such as modifying external variables)
4. Variable names and function names should be sufficiently descriptive that a knowledgeable reader can easily understand what the variable means and what the function does. If this is not possible, comments should be added to make the meaning clear.
5. Use consistent indentation to emphasize block structure.
6. Full line comments inside function bodies should conform to the indentation of the code where they appear.
7. Macro definitions (#define) should be used for defining symbolic names for numeric constants. For example: `#define PI 3.141592`
8. Use names of moderate length for variables. Most names should be between 2 and 12 letters long.

Use underscores to make compound names easier to read: `tot_vol` or `total_volumn` is clearer than totalvolu