HOMEWORK #2:

Pointer Armada.

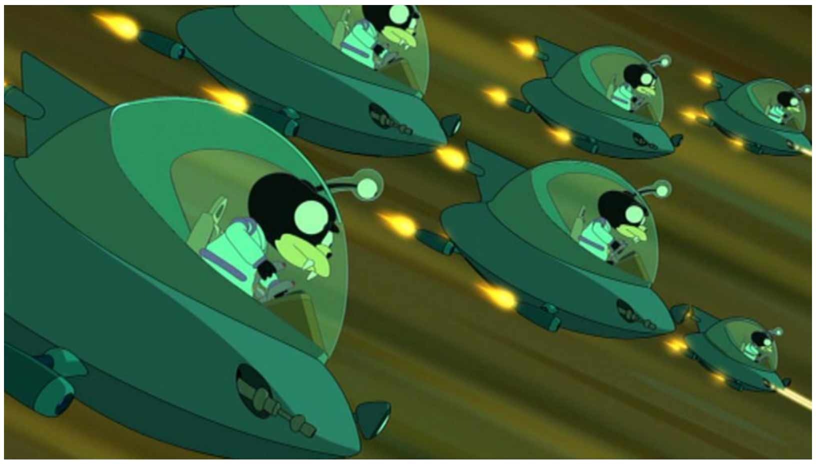
Due Date: Monday, November the 1th, 11:59:59 PM

For this assignment, you need to submit a file called '`linkedlist.hpp`'.
Remember to put your **name** and **section** at the top of your program file.

Problem:

You were so successful with the arraylist software module, that you have now been licensing it to planets all around the galaxy. However, The Nibblonian Council from planet Eternium finds your array implementation an example of the wastefulness of the human species. You really want to sell something to Nibblonian Council, as they will licence many copies of a "**List Abstract Data Type**" for the software in their new cute ship armada.

Implement a **Linked-List** version of the List ADT, and show the Nibblonians that you can both save memory and wrangle a whole herd of wild pointers. A '`linkedlist.h`' file has been provided for you to the wishes and whims of the great Nibblonian Council.



Kitten[1] class attack ships.

[1] Actual designation, check https://en.wikipedia.org/wiki/Futurama

Testing:

Use the provided tester files to check if your implementation is working correctly.

- The program ‘`tester.cpp`’ uses the LinkedList class and the intended output is ‘`testeroutput.txt`’.

In this assignment, you **will be penalized** for memory leaks. To test for leaks, you can use the "Valgrind" tool available in the UNIX systems:
After compiling with `fg++ tester.cpp -o tester.ex`
run your program with the command `valgrind --leak-check=full ./tester.ex`

REMEMBER: Your '`linkedlist.hpp`' should **NOT** '`#include <linkedlist.h>`'

Submission:

Submit your assignment by placing all code in this assignment's git repository in the course's GitLab server, [link] . (You should have a repository setup sometime next week).
Your program will be evaluated and graded on the **Computer Science department's Linux machines** so your program needs to be compatible with the current system, compilers and environment.

Useful Hints:

1. **Carefully** read the comments of each member function.
2. Write down an algorithm for a function before you start coding it.
3. Develop your member functions **one at a time**, starting from the simplest ones.
   Move to the next function only after the previous one has been tested.
   Trying to code the whole class and then remove the bugs may prove to be too big a task.
4. Suggestion: Implement ‘`operator<<`' and ‘`insert_front()`’ first.
5. When a function that needs to return something encounters an error, return `NULL`.
6. There is an auxiliary **constructor** already written in the header file, which is handy for the implementation of the insertion functions.