**[Homework](http://vlm1.uta.edu/~alex/courses/2320/homework/homework.html) - Homework 3**

Submit to [Blackboard](http://elearn.uta.edu/) before the deadline. You will be able to revise your answers until the deadline with no penalty.

Topics: Lists (single-linked), drawing the relevant nodes and the readjustment of links.

**Task 2 (10 pts.) - Time complexity**

For each function discussed in Task 1, specify the time complexity (in Theta notation) that can be attained for that function, using lists **as defined** in list\_hw.c.   
You must give the complexity of the functions that you wrote, as you wrote them. For example if your implementation is inefficient you should give the run time of your implementation (not that of some other efficient implementation).   
Also if your function calls a helper function that you wrote or if it calls some other already provided list function, you should take in consideration the time complexity of that function as well in your answer.

Remember that the time complexity describes the worst-case behavior.   
A brief justification is sufficient. You do NOT need to use the table method.   
Also note that these points will not be distributed equally per function, but given as a total. Therefore if any of the above functions requires a more complicated time analysis it will get more points than the others.   
Put your answer in the 2320\_H3.pdf file.

**Task 3 (10 pts)**

See [2320\_H3.pdf](http://vlm1.uta.edu/~alex/courses/2320/homework/hw3_code/2320_H3.pdf) for this problem.   
Here is a [docx](http://vlm1.uta.edu/~alex/courses/2320/homework/hw3_code/2320_H3.docx) version of the document, for your convenience. If you are using this, save it as a pdf when finished and submit the pdf.

**Task 1 (80 pts.) - Single-linked list, extend interface.**

Summary: For this task You are given 3 files (list\_hw.h, list\_hw.c, and instructor\_client.c). Download them, and compile and run them on omega as given. They should run fine. File list\_hw.c has 5 stubs (place-holders) for functions that you will have to implement. When compiling with the original list\_hw.c, it will call this functions, but they will not do anything. After you implement them, they should do what is required. You will also have to test your functions thoroughly and report the result of those tests in the 2320\_H3.pdf file.

Provided files:

* [list\_hw.c](http://vlm1.uta.edu/~alex/courses/2320/homework/hw3_code/list_hw.c) - The function stubs (where you will write your code) are at the end.
* [list\_hw.h](http://vlm1.uta.edu/~alex/courses/2320/homework/hw3_code/list_hw.h) - header file. Do not modify it.
* [instructor\_client.c](http://vlm1.uta.edu/~alex/courses/2320/homework/hw3_code/instructor_client.c) - client program. Do not modify it.
* [sample run](http://vlm1.uta.edu/~alex/courses/2320/homework/hw3_code/sample_run.html) - See the expected behavior of your program and compilation instructions here.
* [2320\_H3.pdf](http://vlm1.uta.edu/~alex/courses/2320/homework/hw3_code/2320_H3.pdf) - Fill-in the written answers for all tasks in here. Required test cases for Task 1 are also here.   
  Here is a [docx](http://vlm1.uta.edu/~alex/courses/2320/homework/hw3_code/2320_H3.docx) version of the document, for your convenience. If you are using this, save it as a pdf when finished and submit the pdf.

Below are given 5 functions that you must implement in list\_hw.c and the requirements that your code must meet (e.g. no memory errors, keep the function signature, etc.). The list\_hw.c file already has stubs for these functions. You just need to go in there and put the actual code for them.

1. (15 points) **list sublist(list A, list pos\_list)**. This function returns a list that contains the values that appear in list A at positions given in pos\_list
   * Requirement 1: the values should appear in the result in the order given by pos\_list. For example if   
     A: 15 -> 100 -> 7 -> 5 -> 100 -> 7 -> 30 and   
     pos\_list : 3 -> 0 -> 6 -> 4   
     The resulting list will be A[3] -> A[0] -> A[6] -> A[4] , which gives values: 5 -> 15 -> 30 -> 100.
   * Requirement 2: the result should be a deep copy, so that any future changes in list A cannot possibly affect the result, and any future changes in the result cannot possibly affect list A. (List A should remain as it was after building the sublist.)
   * Requirement 3: DO not copy the nodes in an array and then build a list from an array. When found the node in A, make a new node with the same value and insert it in the result list.
   * Requirement 4: DO not make a copy of a list and change the data there. Create new nodes and insert them at the end.
   * The list pos\_list CAN have repetitions. E.g.:   
     list A: 10 -> 5   
     pos\_list: 0->0->1->-0->1->1->0.   
     produces list: 10->10->5->10->5->5->10   
     implicitly the size of A is unrelated to the size of pos\_list.
2. (10 points) **void deleteOccurrences(list A, int V)**. This function should delete (and free) all links in list A that contain value V. For example if   
   A: 15 -> 100 -> 7 -> 5 -> 100 -> 7 -> 30 and we delete value 7, list A will become:   
   A: 15 -> 100 -> 5 -> 100 -> 30
3. (13 points)**void swapFirstThird(list A)**. This function should swap the first and the third nodes in list A by adjusting the links, not copying the items. For this function you need to:
   * Write the code in list\_hw.c
   * Give both the data and the result for the testing it in the table from the 2320\_H3.pdf document.
   * Make a drawing like the one did in class to show how the links are adjusted. In order to make it easy to correct the drawing, INCLUDE A COPY OF **void swapFirstThird(list A)**NEXT TO THE DRAWING.

Special cases:

* + If the list has only one item, no swap is needed.
  + If the list has 2 items, they should be swapped: if A is 3 -> 7 , it becomes 7 -> 3.

1. (22 points) **void moveAllMaxAtEnd(list A)**. This function should move ALL **the nodes** containing the maximum value in the list, to the end of the list.
   * Requirement: You must **move** the original nodes (by updating the links), NOT delete a node and make a new node with a copy of the data and insert at the end. (See how the moved node has same memory address in the sample run.)
   * For example if   
     A: 15 -> 100 -> 5 -> 100 -> 30 and call this function, list A will become:   
     A: 15 -> 5 -> 30 -> 100 -> 100
2. (20 points) **run\_student\_tests()** - your code to test ALL your functions. This code should also not have memory errors.
3. You must keep the signature for these functions as shown here. We will use test code that will call your functions expecting this format. - NO CREDIT WILL BE GIVEN IF THE CODE DOES NOT COMPILE DUE TO CHANGE IN FUNCTION SIGNATURES OR SYNTAX ERRORS.
4. You cannot use arrays to make working with lists easier. E.g., do not copy the data in an array and then access the numbers from the array. The function "arrayToList"provided in instructors\_client is using an array and that is fine. It is safe for you to use this function. It is not bypassing the list manipulation needed for the homework, it is just making it easier to create a hardcoded list.
5. You must thoroughly test your functions. Put your answers in [2320\_H3.pdf](http://vlm1.uta.edu/~alex/courses/2320/homework/hw3_code/2320_H3.pdf) file indicating if you function passes or fails that test.
   * Sample test case file: [data\_1.txt](http://vlm1.uta.edu/~alex/courses/2320/homework/hw3_code/data_1.txt).
   * Your function "failed a test case" if it did not have code to deal with that case and as a result of that, it either crashed or it produced an unexpected effect, e.g.: it used a pointer that was not initialized, or it accessed the wrong data.
6. Your program should not have any memory leaks or memory related errors. 33% penalty if any memory errors are present even if the program 'computes the correct results'.   
   **If you are using a function already provided in the list\_hw.c file and that results in a memory error, you need to address it: either do not use the provided function (write your own) or call the provided function only for cases that it handles correctly).** (You should still NOT modify any of the provided functions.)   
   You can check for memory leaks using the Valgrind program on omega. To run your code with it and get a report of memory leaks at the end, instead of:
8. ./instr

run:

valgrind --leak-check=yes ./instr

will show a report at the end similar to:

==1210==

==1210== HEAP SUMMARY:

==1210== in use at exit: 0 bytes in 0 blocks

==1210== total heap usage: 142 allocs, 142 frees, 2,272 bytes allocated

==1210==

==1210== All heap blocks were freed -- no leaks are possible

==1210==

==1210== For counts of detected and suppressed errors, rerun with: -v

==1210== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)

-----------------------------------------------------------------

In the above report, on the last line:

* + You must make sure you get *0 errors from 0 contexts*.
  + You do not need to handle suppressed errors. E.g. it is fine if you get: *(suppressed: 4 from 4)*

Here are examples of a memory errors reported by Valgrind. Your program should NOT have any of these:

* + memory leak:
  + ==27047== 400 (16 direct, 384 indirect) bytes in 1 blocks are definitely lost in loss record 3 of 3
  + invalid pointer (memory) use:
  + ==9814== Invalid write of size 1
  + ==9814== at 0x804841E: main (example2.c:6)

==9814== Address 0x1BA3607A is 0 bytes after a block of size 10 alloc'd

Read more about using Valgrind and understanding the errors on [this page](http://www.cprogramming.com/debugging/valgrind.html) from Cprogramming.com.   
See the [Code & Programming Requirements](http://vlm1.uta.edu/~alex/courses/2320/code.html) page for instructions on how to compile and run in order to see LINE NUMBERS for the memory errors that Valgrind reports.

**How to submit**

15 points penalty for wrong filenames, name of folder, type of archive (must be ZIP).   
The assignment should be submitted via [Blackboard](http://elearn.uta.edu/). Place all your files in a folder called HW3. Zip that folder (it will produce a file called HW3.zip). Submit HW3.zip. No other forms of compression accepted, contact the instructor or TA if you do not know how to produce .zip files. The zipped directory should contain the following documents:

* **2320\_H3.pdf**. A PDF file containing your solutions for each task. **Include your name and student ID at the top of this document. It may be printed and graded on paper.**   
  2320\_H3.pdf should contain ALL THE COMMANDS you used to COMPILE AND RUN your code. If missing: 10 points penalty per assignment and the assignment will not be graded until you give that information to the TA.
* **list\_hw.c**   
  DO NOT SUBMIT (and do not modify): instructor\_client.c ,list\_hw.h.

Programs must have the .c extensions and must run on omega.uta.edu (when compiled with the list\_hw.h and instructor\_client.c).   
  
**PARTIAL CREDIT :** Partial credit will NOT be given for programs that have compilation or runtime errors. You are responsible for debugging your own code.   
  
**IMPORTANT:** Pay close attention to all specifications on this page, including file names and submission format. Even in cases where your answers are correct, points will be taken off liberally for non-compliance with the instructions given on this page (such as wrong file names, wrong compression format for the submitted code, and so on). The reason is that non-compliance with the instructions makes the grading process significantly (and unnecessarily) more time consuming. Contact the instructor or TA if you have any questions.

Back to [Homework](http://vlm1.uta.edu/~alex/courses/2320/homework/homework.html) page .