CIT 593 Homework 11: C – Dynamic Memory & Linked Lists

Due Date: Monday 12/5 @11:59 PM via canvas upload ONLY

In this assignment we will be working with dynamic memory. **Chapter 19** will be very helpful for this assignment, please read it before starting! To work with dynamic memory, we will need access to malloc() and free(); they are part of the standard library in C. There isn't a standard library available for the LC4 compiler, so we will instead use the C-compiler available on the universities server known as ENIAC. Instructions on how to connect and compile a C program on ENIAC are available on canvas under: Files->Resources->Tutorials

*Note: for this assignment you must compile and ensure your programs work on our school server ENIAC. Compilers on your own machine may differ from the one used on ENIAC. Even though something may compile and run on your own local machine, we will be compiling and testing your program **ONLY** on eniac, so it is your responsibility to ensure your code compiles and runs properly there.*

1) **Setting up your C-programming environment:**
   For this assignment you will need to do two things before going on:
   1. To read the eniac connection/compiler **tutorial**
   2. Download the helper files given with this assignment to your local computer

   a) Once you have completed the above, connect to eniac and then type the following:
   ```
   mkdir cit593
   cd cit593
   mkdir hw11
   cd hw11
   ```
   The above will create a folder called "cit593" and another folder inside it called: "hw11"

   b) On your local computer, extract the helper files from the provided ZIP

   c) Transfer the helper files from your local computer to eniac
   Examples of how to do this are shown in the tutorial

   d) Compile the helper files on ENIAC and then run the helper files by typing:
   ```
   clang –o program1 program1.c linked_list.c
   ./program1
   ```
   The above will compile program1.c and linked_list.c and link them to form a single program that you can run called: program1. The statements above will also run the program that you are creating, and you will need to interact with it when it runs!

While authoring functions for linked lists is something done often and much information is on the Internet about doing it, I remind you that taking code from the Internet will most certainly result in you being caught when we run our plagiarism analysis software or simply look at your code. If you cannot do the assignment on your own; come for help, otherwise you will be caught, but worse you will waste a chance to become a great C programmer. *We are not looking for perfect solutions, we are looking for your solutions to the following challenges.*

CIT 593 Homework 11: C – Dynamic Memory & Linked Lists

2) <u>Examine the provided files:</u>

The file: **program1.c**, in this file you will see the "main()" function
The file: **linked_list.h**, in this file you will see the definition of a structure and 8 helper functions declared
The file: **linked_list.c**, in this file you will see the definition (the "meat") of the 8 helper functions that were declared in linked_list.h

Open up the **program1.c** file:
Scroll down to the main() function and examine it:

- A pointer is created, it is intended to always point to the head of a linked list
- Main() calls a function: print_menu() to display a simple menu to the user
- The user may: exit the program, print out the linked list, or add a # to the list
    - *You will need to add more choices as part of the assignment*
- Carefully examine how a "switch" statement works, it is much like several if/else statements, except it is a bit more concise:
  (two useful references: Textbook: 13.5.1 and:
  http://www.tutorialspoint.com/cprogramming/switch_statement_in_c.htm
- If the user chooses to "add a number to the list" the number is then asked what # they would like to add:
  How scanf() works: Textbook: 11.5.4 and:
  http://computer.howstuffworks.com/c7.htm
- After the user types that number, a linked list "helper function" is called
    - The linked list helper functions are in linked_list.c

Open up the **linked_list.h** file:

- In this file you'll see the definition of a single node's structure
    - Notice this is a linked list node that is similar to the example from class, except it will be a "doubly" linked list!
    - Carefully examine the two pointers in the node definition
- You'll also see the names, arguments, and return types of 8 helper functions designed to work with a linked list of the type of node defined at the top of the file
- We call these "helper functions" as they "help" another programmer work with our linked list type
- This file can be "included" by other programs (like program1.c) if a programmer wishes to gain access to these functions
- Since these are only the "declarations" of the helper functions, and not the "definitions" of the functions (aka – the code that makes the functions work), we must actually define the functions somewhere!

CIT 593 Homework 11: C – Dynamic Memory & Linked Lists

Open up the **linked_list.c** file:

- In this file you'll see the definition of the helper functions that were declared in linked_list.h
- You'll notice only 1 function is properly defined, the rest are empty
- You will define the remainder of the functions as part of this assignment
- Carefully examine the "add_to_top()" helper function
    - It takes in two arguments: a pointer to the "head" of an existing linked list
    - As well as a new integer to add to the start of the linked list that is specified by the other argument
    - Its purpose of the function is to add the integer that is passed in, to the head of the existing linked list
    - The function allocates memory on the heap for the new node and attaches this new node, promoting it to be the new head of the linked list
    - Upon completion it returns a pointer to the new head of the linked list
    - This function also serves as a way to create the linked list if it doesn't already exist!

Now…return to **program1.c**:

- Look carefully on line 42 of program1.c
- Notice how it requests an addition to the linked list, by passing in the existing linked list's head pointer and the new integer value it would like added to the list
- But most importantly, notice how the head pointer will be updated after add_to_top() is complete!

CIT 593 Homework 11: C – Dynamic Memory & Linked Lists

3) First things first…update linked_list.c

- Implement the remainder of the linked list helper functions in linked_list.c
    i. The details of their implementation are in the comments in linked_list.c
- As you implement each helper function, COMPILE your code and TEST IT!
    i. DO NOT attempt to write all the helper functions at the same time, compile, and debug from there. That is a terribly strategy that will most certain end in disaster. We will not help you debug your code in that scenario.
- I highly recommend starting with the easiest helper function: print_list()
- After that, I recommend implementing "delete_list()"

4) Updating program1.c

- As you implement each helper function in linked_list.c, add an item to the menu to allow the user of your program (and you) to test the helper function
- So that all student programs can be tested by our graders, when you complete your menu system, it must be in the following order:

```
0 – exit
1 – print the entire linked list
2 – add to the start of the linked list
3 – add to the end of the linked list
4 – search list for a #
```

(this should prompt the user for a number to search for in the list. If the matching node is found it should print out the node # a print "FOUND IT" or "NOT FOUND" as the case may be)

```
5 – sort list
6 – sort list backwards
7 – delete list
```

- To do this you will need to modify print_menu() and certainly the switch() statement cases.

CIT 593 Homework 11: C – Dynamic Memory & Linked Lists

5) Test for memory leaks

- One of the most difficult tasks for new programmers of linked lists is not to leak memory when adding/removing nodes/deleting the entire linked list
    - This means you called "malloc()" but never called free() on the memory that was malloc()'ed. For every call to malloc() you must have a corresponding call to "free()
- It is very difficult to determine if you've made a mistake and not properly free()'ed some memory without debugging tools
- On ENIAC we have a powerful memory debugging tool called: **valgrind**
- You MUST test your program with valgrind prior to submitting it
- If you "leak" memory in your program, you will lose significant points on this assignment
- The starter code for this assignment has a major memory leak in it! Notice, I never implemented "delete_list()"
    - While I call that helper function in main, it certainly doesn't do anything
    - Once you properly implement delete_list() the code won't leak memory any more
- You run valgrind on ENIAC by typing this:

    ```
    valgrind program1
    ```

    add a number to the list…then exit the program

- After you compile the starter code, try running valgrind as I've shown above and you'll get the following error messages:

    ```
    ==3437== HEAP SUMMARY:
    ==3437==     in use at exit: 24 bytes in 1 blocks
    …
    ==3437== LEAK SUMMARY:
    ==3437==    definitely lost: 24 bytes in 1 blocks
    ```

    This means I called malloc() and got 24 bytes on the heap, but I never called "free()" on those same 24 bytes before the program exited! Notice 24 bytes is exactly the amount of memory a **struct_of_ints** requires on the heap!

- While that is a horrible error (that you must fix) other errors may come up as you code, so more details on valgrind are provided here:

    http://valgrind.org/docs/manual/quick-start.html

CIT 593 Homework 11: C – Dynamic Memory & Linked Lists

6) Extra Credit (10 points)

- After you get the entire program working above, create a copy of it called:
  `linked_list2.c, linked_list2.h, program2.c`

- Alter the node structure to contain two pieces of data:

```
int value ;
char* string ;
```

- This new field: **string**, will allow the linked list to contain integers and have a string associated with each integer
- Change the "add_to_top" and "add_to_end" functions to also take in a string
- When a new node is created, allocate memory for the string, and copy the user passed in string to the node
- Update the menu, to not only ask the user for a number, but also for a string
  - You'll need to include the c header file: #include <string.h> to gain access to C functions like: strcpy() which works exactly the same was as your lc4_strcpy()
- You will be given 5 points for getting this to work properly
- You will be given the last 5 points if your program doesn't leak memory!

CIT 593 Homework 11: C – Dynamic Memory & Linked Lists

Directions on how to submit your work:

- Create a single zip file called: **LAST_FIRST_HW#.zip**
- The zip file should contain the files named in this assignment.
- There should not be any sub-directories within your zip file.

As an example, I would turn in the following SINGLE zip file:

FARMER_THOMAS_HW11.zip

This single zip file would contain these files *(italicized file names are for extra credit):*

linked_list.h
linked_list.c
program1.c
linked_list2.h
linked_list2.c
program2.c

You will then upload ONLY 1 file to canvas: **FARMER_THOMAS_HW11.zip**

- DO NOT TURN IN ANY **executable files**!
- Make certain that you submit the latest versions of your code.
- **Make certain that you ran valgrind on your code prior to submission**
- **If you program doesn't compile or work at all, you will receive little or no credit on the assignment, make certain to use late days if you need to at least get the program working!**
- Submitting using any other compression type (.RAR, TAR, GZIP) will be rejected.

Paper/Email submissions will not be accepted for this assignment.