CS220 - Computer System II
Combined Assignment 8 and 9 (200 points)

Due: 11/10/2017, 11:59pm

1 Instructions

For this assignment:

- Answer the questions individually. Group effort is not allowed.
- Reading: `http://c-faq.com`, K&R C, chapter 1-6.
- You are allowed to use resources from the internet as long as you refer them in your repository README file.
- You will note that none of the files in the repository have been populated except the Makefile (and even that only specifies the targets you must implement and includes our testing Makefile into your Makefile). As we are in the last half of the semester, you should be able to do these tasks with much less guidance.

2 Questions

2.1 sort_nodes

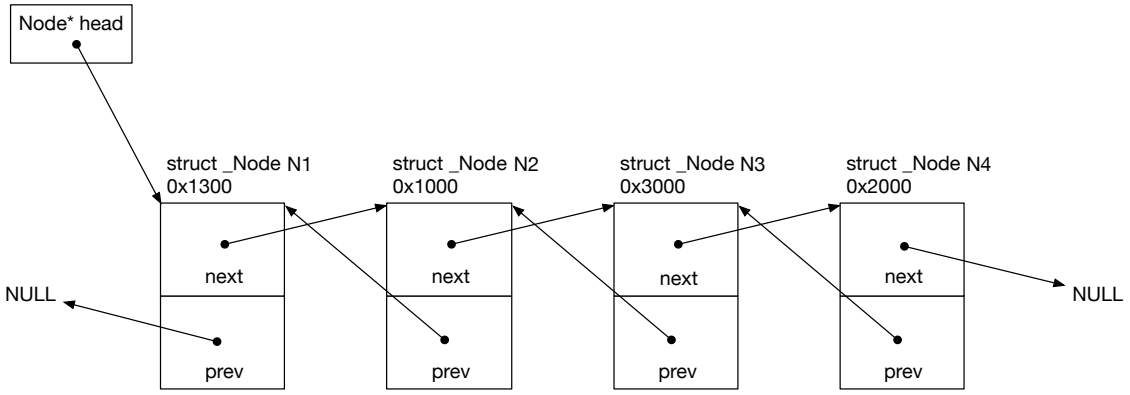
(80 points) You are given structure `struct Node` below, where `next` points to the next node in the list (or NULL if it is the last node in the list), and `prev` presents to the previous node in the list (or NULL if it is the first node).(see Figure 1):

```
1  struct _Node {
   struct _Node *next;
3  struct _Node *prev;
   };
5  typedef struct _Node Node;
```

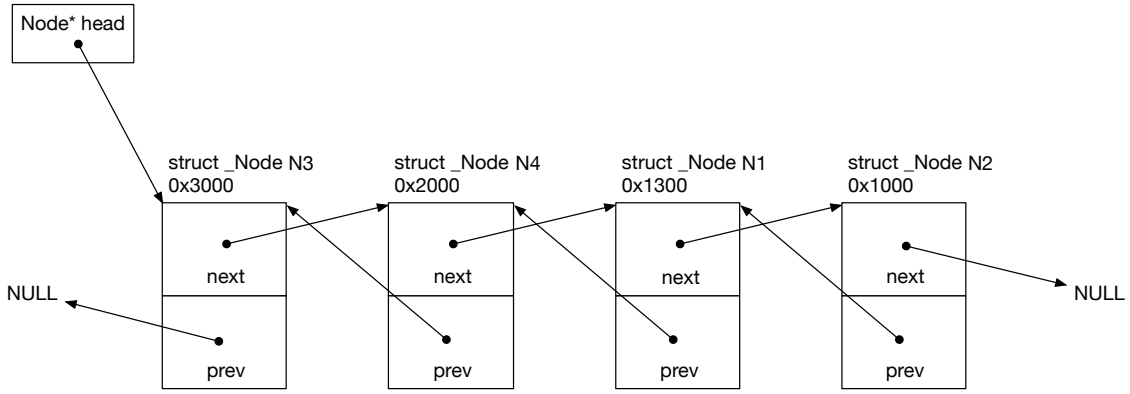
Implement a function `Node *sort_nodes(Node *head)` that returns head of list such that: $\&node1 > \&n2 > \&n3 > ... > \&last\_node$ You will submit **sorter.c** and **node.h**. **sorter.c** will contain the `sort_nodes` function. **node.h** will contain the definition of the `Node` structure and the prototype for the `sort_nodes` function.

Figure 1: Lists before and after sorting. Arrows going left to right indicate "next" pointers, and arrows going right to left indicate the "prev" pointers.

Input



Output



2.1.1 Hints and Notes

- You should mix nodes from stack (local variables), heap (dynamic variables) and global variables to generate a list comprising of nodes of varying addresses.
- Although you are free to use any sorting algorithm, it might be easier to use selection sort. In simple words, selection sort is where you iterate through the list to select the largest element and move it to the beginning of the list. (`https://en.wikipedia.org/wiki/Selection_sort`).
- You are guaranteed that your function will be tested against lists that contain at least 3 elements. So, you are not required to handle lists of smaller size.
- You are not allowed to make copies of the nodes. You are required to sort them by altering the `next` and `prev` pointers.

2.2 Estimator

(60 points) You are to write a program to estimate the throughput of `nop` instructions. That is, you are to find the number of `nop` instructions that the processor can execute in 1 second. Your program must be called **estimator** ands its implementation is in **estimator.c**. It should print a single number in scientific notation indicating the number of `nop` instructions the processor can execute in 1 second with a trailing newline *and nothing else*. This number of instructions should reflect the average of at least 5 trials.

2.2.1 Hints and Notes

Using the timer code given in the Labs, run an assembly function comprising of 100 `nop` instruction a large number of times (e.g., `(unsigned int) 0xffffffff`), and run another assembly function that contains 0 `nop` instructions (only a `ret` instruction) the same number of times. In both cases, measure the time consumed over at least 5 trials. Compute the average number of `nop` instructions the processor executes per second. You can expect at least 2 x 10^9 `nop`'s per second.

2.3 Toggle

(60 points) You are to implement the following function in 64 bit assembly in file **toggle.S**, with the function prototype in **toggle.h**.

```
unsigned long toggle_bit(unsigned long num, unsigned long bit_num);
```

The function accepts an unsigned long `num` and toggles (A 0 bit changes to 1 and a 1 bit changes to 0) the bit at `bit_num` position ($0 \leq$ bit_num ≤ 64, where 0 is the least significant bit and 63 is the most significant bit).

3 Submission

Submit the 7-digit SHA hash as a student comment to a MyCourses submission.