ECE368 Fall 2015 Homework 4

IMPORTANT:

- Do NOT leave your name or Purdue ID on this homework.
- Write your homework security number at the TOP of EACH page.

Read and sign the ***Academic Honesty Statement*** that follows:

"In signing this statement, I hereby certify that the work on this exercise is my own and that I have not copied the work of any other student while completing it. I understand that, if I fail to honor this agreement, I will receive a score of zero for this exercise and will be subject to further disciplinary action."

`Homework security number:`

Please acknowledge any people who have helped you with this homework.

Question	Credits
1	
2	

1. (40 points) The following function `permute()` prints all permutations of the given string `str`. For instance, a call of `permute(0, 2)` should print the following (order does not matter):

```
ABC
ACB
BAC
BCA
CBA
CAB
```

Try to complete `permute()` and briefly describe what does your code do, in English. Hints: 1) consider recursion; 2) you shouldn't write more than 5 lines of code.

```
1  /* A helper function to swap two chars */
2  void swap (char *x, char *y)
3  {
4      char temp;
5      temp = *x;
6      *x = *y;
7      *y = temp;
8  }
9
10 char str[] = "ABC";
11
12 /*
13    This function operates the global variable @str in place.
14    i and n: Starting and ending indice (inclusive) of a substring
         to be permuated.
15 */
16 void permute(int i, int n)
17 {
18    int j;
19    if (i == n)
20      printf("%s\n", str);
21    else {
22        for (j = i; j <= n; j++) {
23             /* YOUR CODE HERE */
24           /* str+i is equivalent to &(str[i]) */
25           swap((str+i), (str+j));
26           permute(i+1, n);
27           swap((str+i), (str+j));  /* backtrack */
28        }
29    }
30 }
```

---

2. (60 points) Write a piece of C code to reverse a singly linked list *in place* (i.e., do not allocate extra memory) and return the new list's head, if successful. For node definition, use the dynamic memory implementation shown in the lecture slides. Briefly describe what does your code do, in English. Hints: 1) pay attention to corner cases; 2) you shouldn't write more than 20 lines of code; 3) recursive or iterative, which way you go?

```
1  /* Iterative */
2  Header_t *reverse(Header_t *header)
3  {
4      /* YOUR CODE HERE */
5      assert(header);
6      Node_t * current =  header->First_Ptr, next, previous = NULL;
7
8      header->Last_Ptr = header->First_Ptr; /* will be new tail */
9
10     while (current) {
11         next = current->Next_Ptr;
12         current->Next_Ptr = previous;
13         previous = current;
14         current = next;
15     }
16
17     /* new head: the one before we run into NULL */
18     header->First_Ptr = previous;
19
20     return header;
21 }
```

```
1  /* Recursive */
2  /* return: the 1st node in the new list */
3  Node_t *do_reverse(Node_t *start)
4  {
5      /* YOUR CODE HERE */
6      Node_t *newstart;
7
8      if (!start)          /* empty list */
9          return NULL;
10
11     if (!start->next_Ptr)        /* only 1 node in the list */
12         return start;
13
14     newstart = do_reverse(start->next_Ptr);
15
16     /* new tail's next_Ptr should point to the old start */
17     start->next_Ptr->next_Ptr = start;
18     start->next_Ptr = NULL;
19
20     return newstart;
21 }
22
23
24 Header_t *reverse(Header_t *header)
25 {
26     /* YOUR CODE HERE */
27     assert(header);
28     header->Last_Ptr = header->First_Ptr;
29     header->First_Ptr = do_reverse(header->First_Ptr);
30
31     return header;
32 }
```