CSE 3100 Systems Programming Fall 2022

Exam #1

This is a 110 minutes exam. *Please read every question carefully.*
Please read the following list of important items on exam policy and make sure you understand it.

- The Makefile for the exam can be downloaded from HuskyCT.
- During the exam, you cannot communicate with and/or obtain help from people other than TAs and instructors on exam questions. Particularly, you cannot use any messaging applications.
- During the exam, you can only access data/files on HuskyCT, codingrooms, your Virtual Machine, and files on your own laptop and on the lab computer you use. You cannot access data/files on other computers/servers. You cannot use your phone.
- There are three (3) questions on the exam. You must submit your code on HuskyCT. We only grade submissions we have received on HuskyCT.
- You must return the exam paper before leaving the exam room.
- After the exam, you cannot discuss/disclose exam problems and/or share your code with students who have not taken the exam.

I pledge my honor that I have not violated and will not violate the exam policy of this course and the Student Conduct Code during this examination.

Signature: ________________________________ Date: _______________________

Printed Name: _____________________________ NetID: _______________________

Lab Section: _________________

Note that a Makefile is provided and can be downloaded from HuskyCT. Please submit your .c code in the HuskyCT exam.

Problem 1. (30 points) Palindromic numbers
A palindromic number is a number (such as 16461) that remains the same when its digits are reversed. First, we implement the function palindrome to test whether an unsigned long integer n is a palindromic number. The function returns 1 if that is the case and 0 otherwise.

```
int palindrome(unsigned long n)
{

}
```

Next, we need to find a unsigned long integer p that is NOT a palindromic number but p^3 is a palindromic number. Specifically, we implement the following function, which returns the smallest such number p.

```
unsigned long pCube()
{

}
```

Problem 2. (30 points) Last digit count
In this problem we write code to obtain the frequencies of the last digit among n unsigned integers. For example, if we have the following unsigned integers,

```
11, 111, 20, 34, 455, 6778, 700, 8, 9, 99, 999,
```

the frequencies of the last digits are:

```
0: 2, 1: 2, 2: 0, 3: 0, 4: 1, 5: 1, 6: 0, 7: 0, 8: 2, 9: 3.
```

Here 9, 99, 999 all ends with the digit 9, therefore the frequency for digit 9 is 3.
Specifically, we need to write the following function for this purpose.

```
void lastDigitCount(unsigned *a, unsigned n, unsigned freq[10])
{
}
```

This function will count the frequencies of the last digits among all the unsigned integers in the array a[]. There are n numbers in the array as indicated by the argument n. Moreover, these frequencies will be stored in the array freq[10]. Here freq[0] stores the frequency of 0, freq[1] stores the frequency of 1, etc. The main function is given and should not be changed, as shown below.

```
//Do not change the main function
int main(int argc, char *argv[])
{
    assert(argc == 2);
    unsigned n = atoi(argv[1]);

    assert(n>=1 && n<=10000);
    unsigned a[n], i, s = 0;

    for(i=0; i<n; i++)
    {
        s += 2*i+1;
        a[i] = s;
    }
    unsigned freq[10];
    lastDigitCount(a, n, freq);
    for(i=0; i<10; i++)
        printf("%u: %u\n", i, freq[i]);
    return 0;
}
```

Below is an example output.

```
./q2 10000
0: 1000
1: 2000
2: 0
3: 0
4: 2000
5: 1000
6: 2000
7: 0
8: 0
9: 2000
```

Problem 3. (40 points) Doubly linked list reversal
In this problem, we write code to implement a reverse_list function to reverse an existing doubly linked list.

Each node in a doubly linked list contains an extra pointer, typically called the previous pointer, together with the next pointer and data which are there in the singly linked list.

In order to implement a doubly link list, we introduce the node structure as follows.

```
typedef struct node_tag {
    struct node_tag * prev; // A pointer to the previous node
    int    v;          // data
    struct node_tag * next; // A pointer to the next node
} node;
```

To facilitate adding a node to the tail of a doubly linked list, we not only use a pointer to keep track of the head of the doubly linked list, we also use a pointer to keep track of the tail of the doubly linked list, as indicated in the following line of code in the main function.

```
node *p, *head=NULL, *tail=NULL;
```

In fig 1, we show an example of a doubly linked list and the corresponding reversed doubly linked list. The following functions are already implemented.

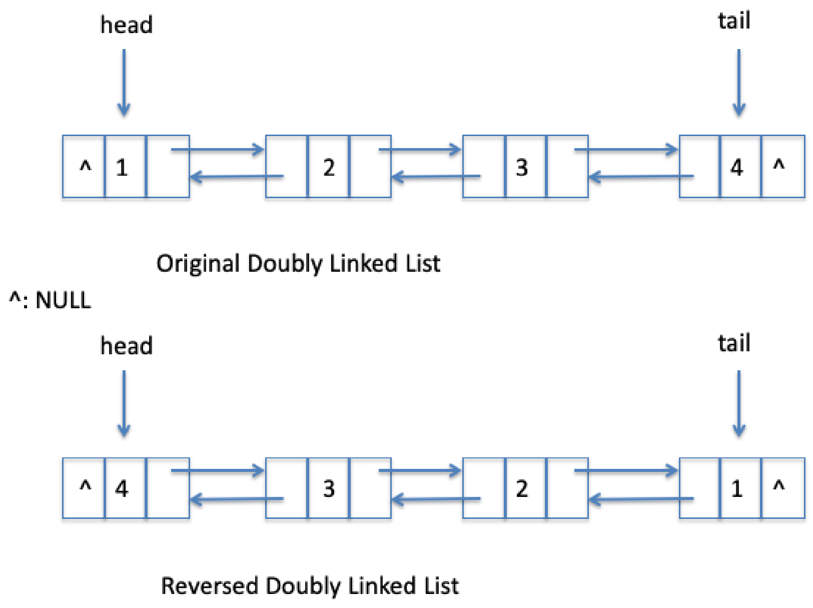

Figure 1: A doubly linked list and the reversed doubly linked list

```
void add_last(node **head, node **tail, node *newnode);

node * remove_first(node **head, node **tail);

void print_list(node *head);
```

The purpose of these functions are self-evident.

What is left to be implemented are the following two functions.

```
void reverse_list(node **head, node **tail);

void free_all(node **head, node **tail);
```

The function reverse_list reverses a existing doubly linked list specified by the two arguments.

The function free_all delete all the nodes from the doubly linked list and free all the associated memory. The doubly linked list is specified by the two function arguments.

Below is the main() function of the program; note how the two functions we need to implement are used.

```
//Do not change the main function
int main(int argc, char * argv[])
{
    assert(argc == 2);
    int n = atoi(argv[1]);

    node *p, *head=NULL, *tail=NULL;

    for(int i=0; i<n; i++)
    {
        p = create_node(i);
        add_last(&head, &tail, p);
    }
    print_list(head);
    reverse_list(&head, &tail);
        print_list(head);
    free_all(&head, &tail);
    return 0;
}
```

Below are some typical outputs.

```
$ ./q3 10
0 1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3 2 1 0
$ ./q3 20
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```