Matthew Browning
Dr. Li
Comp 2710
30 March 2020

Exam 2

1. **Tower of Hanoi (40 points)**
   1. **Please implement a "moveDisk" function in a given TowerOfHanoi_plain.cpp (20 points)**
      Separate file
   2. **Assume moving one single disk each time costs one second, how many seconds are spent if there are 3 disks, 8 disks, 10 disks. (6 points. 2 points each)**
      With 3 disks it will take 7 seconds.
      With 8 disks it will take 255 seconds.
      With 10 disks it will take 1023 seconds.
   3. **How many seconds are spent if we have N disks. (5 points)**
      Given the recursive algorithm for solving the Tower of Hanoi problem, to solve N disks you will need (2^n) - 1 seconds.
   4. **Since this is a 24-hour exam, how many disks can you successfully move from Peg A to Peg C? Don't guess it. List your brief calculations. (5 points)**
      Okay, time to break out the calculator!
      Since each disk transition takes 1 second, it is obvious that the first step we need to take is seeing how many seconds that we are working with.
      $$60*60*24 = 86{,}400\text{seconds/day}$$
      Now that we know how many seconds, we can set that equal to the equal seen in part 3 (seconds = (2^n) - 1).
      $$(2^\wedge n) - 1 = 86400$$
      Then we solve.
      $$2^\wedge n = 86401$$
      $$\log_2 86401 = n$$
      $$n = 16.3987$$
      Now, since we can't move 39.87% of a disk to Peg C, just round this down to 16.
   5. **Running your own solution-- TowerOfHanoi_plain.cpp, how many seconds does AU server spend on 16 disks? (4 points)**
      It takes 1 second to move 16 disks. I promise that is what it says.

2. **Singly-Linked Lists (48 points)**
    1. **Please define each node of Tree from Fig. 1 (12 points)**

```
struct Node {
        int data;
        struct Node*left;
        struct Node *right;

        Node(int data) {
                this -> data = data;
                this -> left = NULL;
                this -> right = NULL;
        }
};
```

    2. **Please print 5 numbers with Inorder (12 points)**

```
// Algorithm Inorder(tree)
//1. Traverse the left subtree, i.e., call Inorder(left-subtree)
//2. Visit the root.
//3. Traverse the right subtree, i.e., call Inorder(right-subtree)

void printInorder(struct Node* node) {
        if (node == NULL)
                return;

        /* first recur on left child */
        printInorder(node -> left);
        /* then print the data of node */
        cout << "Node data: " << node -> data << endl;
        /* now recur on right child */
        printInorder(node -> right);
}
```

3. **Please print 5 numbers with Preorder (12 points)**

```
//Algorithm Preorder(tree)
//1. Visit the root.
//2. Traverse the left subtree, i.e., call Preorder(left-subtree)
//3. Traverse the right subtree, i.e., call Preorder(right-subtree)

void printPreorder(struct Node* node) {
        if (node == NULL)
                return;

        /* first print data of node */
        cout << “Node data: ” << node -> data << endl;
        /* then recur on left subtree */
        printPreorder(node -> left);
        /* now recur on right subtree */
        printPreorder(node -> right);
}
```

4. **Please print 5 numbers with Postorder (12 points)**

```
//Algorithm Postorder(tree)
//1. Traverse the left subtree, i.e., call Postorder(left-subtree)
//2. Traverse the right subtree, i.e., call Postorder(right-subtree)
//3. Visit the root.

void printPostorder(struct Node* node) {
        if (node == NULL)
                return;

        // first recur on left subtree
        printPostorder(node -> left);
        // then recur on right subtree
        printPostorder(node -> right);
        // now deal with the node
        cout << “Node data: ” << node -> data << endl;
}
```

3. **Please fill out blanks to reverse a doubly link list with a Node struct (12 points)**

```
struct Node {
        int data;
        struct Node *next;
        struct Node *prev;
};

void reverse(Node **head) {
        Node *temp = NULL;
        Node *current = *head;

        /* swap next and prev for all nodes of a doubly linked list */
        while (current != NULL) {
                temp = current -> prev;
                current -> prev = current -> next;
                current -> next = temp;
                current = current -> prev;
        }

         /* Before changing the head, check for the cases like empty list and list
        with only one node */
         if (temp != NULL)
                *head = temp -> prev;
}
```