CSE 15
Introduction to Data Structures
Final Review Problems

Study the review problems from previous midterm reviews, previous midterms, and study any posted solutions.

1. Fill in the definition of the C function below called `printPreOrder()`. This function will, given the root `N` of a binary search tree based on the Node structure below, print out the keys according to a pre-order tree traversal. Write similar functions called `printInOrder()` and `printPostOrder()` that print in-order and post-order tree traversals respectively.

```
typedef struct NodeObj{
   int key;
   struct NodeObj* left;
   struct NodeObj* right;
} NodeObj;

typedef NodeObj* Node;

void printPreOrder(Node N){









}
```

2. Draw the Binary Search Tree resulting from inserting the keys: 5 8 3 4 6 1 9 2 7 (in that order) into an initially empty tree. Draw another BST that results from deleting the keys 5 1 7 (in that order) from the previous tree. Show the output of functions `printInOrder()`, `printPreOrder()` and `printPostOrder()` from problem 1 when run on the root of this tree.

3. Assuming the existence of the `NodeObj` and `Node` types defined in problem 1, and given the Node constructor and destructor defined below, trace function `main()` and **draw the binary search tree that results**. Write C instructions that will "manually" perform the following operations in succession: insert the key 1, insert the key 3, delete the key 5. **Draw the resulting binary search tree**. Complete the definition of the recursive C function `void freePostOrder(Node R)` below that will free all nodes in any tree based on the above `Node` type, then use it to free the entire tree at the end of function `main()`.

```
Node newNode(int k){
   Node N = malloc(sizeof(NodeObj));
   N->key = k;
   N->left = N->right = NULL;
   return N;
}

void freeNode(Node* pN){
   if(pN!=NULL && *pN!=NULL){
      free(*pN);
      *pN = NULL;
   }
}

void freePostOrder(Node R){









}

int main(){
   Node root = newNode(5);
   root->left = newNode(2);
   root->right = newNode(7);
   root->left->right = newNode(4);
   root->right->left = newNode(6);
   root->right->right = newNode(8);
```



```
   return EXIT_SUCCESS;
}
```

4. Write a C function with prototype `char* cat(char* s1, char* s2, char* s3)` that takes three C strings `s1` , `s2` and `s3` (i.e. null `'\0'` terminated char arrays), allocates a char array from heap memory of sufficient length to store the concatenation of the three strings (including a terminating null '\0' at the end), copies the contents of `s1` , `s2` and `s3` into that array (including the terminating null '\0'), then returns a pointer to the allocated memory. You may not use functions from the `string.h` library to accomplish the above tasks, in particular, you must manually determine the length of the strings `s1` , `s2` and `s3` by searching for their terminating null characters.

5. Write a C function heading `void sortWords(char** W, int n)` that sorts its array argument `W` in alphabetical order. Assume that `W` is an array of length *n* whose elements are C strings (null-terminated char arrays). Do this using the algorithms
   a. Bubble Sort.
   b. Selection Sort.
   c. Insertion Sort.
   d. Merge Sort.
   e. Quick Sort.
   f. Counting Sort.

6. The keys 28, 5, 15, 19, 10, 17, 33, 12, 20, 6, 9, 23, 34, 22 and 21 are to be inserted in order into a hash table of length 9.
   a. Suppose that collisions are to be resolved by chaining, with insertions performed at the head of each list. The hash function is $h(k) = k \ \boldsymbol{mod}\ 9$. Show the state of the hash table after all insertions are performed, i.e. draw the array and the linked lists which result.
   b. Referring to part (a), what is the load factor α? How many collisions are there?
   c. Show the result of inserting the same keys into a hash table of length 9 where collisions are resolved by open addressing, with hash function: $h(k,i) = (k \ \boldsymbol{mod}\ 9 + 5i) \ \boldsymbol{mod}\ 9$. Note that in this case there are more keys than slots, so you won't be able to insert all the keys. Just insert the keys in the given order until there is no more room in the table. Again draw a picture of the array and its contents.

7. The keys 34, 25, 79, 56, 6 are to be inserted into a hash table of length 11, where collisions will be resolved by open addressing. The hash function is $h(k,i) = (k \ \boldsymbol{mod}\ 11 + \ i \cdot (1 + k \ \textbf{mod}\ 10)) \ \boldsymbol{mod}\ 11$.
   a. Calculate the probe sequence of each of the above keys.
   b. Write a program in C that calculates and prints out the probe sequences you computed in (a).

8. Given the `NodeObj` structure and `Node` reference below, write a *recursive* C function with heading `int sumList(Node H)` that returns the sum of all the items in a linked list headed by H. An empty list is headed by `NULL` and has sum 0.

```
typedef struct NodeObj{
   int item;
   struct NodeObj* next;
} NodeObj;
typedef NodeObj* Node;

int sumList(Node H){
   // your code goes here



}
```

9. Given the `NodeObj` structure and `Node` reference from problem 9 above, write the following C functions:
   a. A constructor that returns a reference to a new `NodeObj` allocated from heap memory with its item field set to *x* and its `next` field set to `NULL`.

```
Node newNode(int x){
   // your code goes here




}
```

   b. A destructor that frees the heap memory associated with a `Node`, and sets its reference to `NULL`.

```
void freeNode(Node* pN){
   // your code goes here




}
```

   c. A *recursive* function with heading `void clearList(Node H)` that frees all heap memory associated with a linked list headed by `H`.

```
void clearList(Node H){
   // your code goes here







}
```

10. Adapt the BSTSort algorithm to operate on C strings. The algorithm can be found at

    /Examples/Lecture/ADT-Examples/DictionaryADT/BSTExamples/BSTsort.c