CS300 Final Review Questions

This is not a complete list of questions and topics, but a good sampling of questions that will help you study for the final. I strongly advise you to work through every single question.

- Review each of your old Exams.
- Review each in-class Lab.
- Review each programming assignment.
- Review each set of notes and the questions/problems embedded in the notes.
- Make sure you can program generically with void * data types and function pointers

```
typedef struct NODE *NODE_PTR;
typedef struct NODE
        {
          char data;
          NODE_PTR psNext;
        } NODE;
```

```
typedef struct BT_NODE *BT_NODE_PTR
typedef struct BT_NODE
        {
          int data;
          BT_NODE_PTR psLeftChild;
          BT_NODE_PTR psRightChild;
        } BT_NODE;
```

1) The values A, B, C, D are inserted into a queue maintained as a circular list. Draw a picture of the resulting queue after all elements have been inserted.

2) The queue described in 1) is maintained with a single pointer of type NODE_PTR. Write a function qDequeue that returns the data value from the queue deleting the queue element from the queue.

3) Using the list routines from list.h, define a data structure for a stack that is maintained using the list routines.

4) Using your data structure in 3), create a stack and write routines stkCreate, stkSize, stkIsFull, and stkPush. What other data structures can easily be implemented with a list?

4.5) Assume the list and stack ADTs from problems 3) and 4) have been implemented. You now have actual datatypes List and Stack. You are to create a new datatype called Queue that uses two Stack variables to implement the Queue.

a) Write the declaration for the Queue datatype.

```
typedef struct Queue
{
  Stack sStack1, sStack2;
} Queue;
```

b) Write qInit.
c) Write qEnqueue.
d) What is the computing complexity of qEnqueue? Why?
e) Does the Makefile for Queue have any dependencies on List? Why or why not? Draw the dependency graph for this Queue implementation.

5) Assume that we have a new data structure for a circular queue maintained in an array as follows:

`typedef struct Q_ELEMENT`
`        {`
`          char name[32];`
`          int age;`
`        } Q_ELEMENT;`
	`typedef struct QUEUE *QUEUE_PTR`
	`typedef struct QUEUE`
	`        {`
	`          Q_ELEMENT data[100];`
	`          int qFront, qRear;`
	`          int size;`
	`        } QUEUE;`

Write the functions cqCreate, cqIsFull, and cqEnqueue.

7) Show what a call would look like for the functions described in 2) and 5).

8) What is the computing complexity for the enqueue operation in 5)?

9) Insert the following values into a BST: 40, 30, 35, 60, 80, 70, 32, 25, 27.

10) What is the worst-case computing complexity for searching a: a) BST b) ordered array c) unordered array d) ordered list e) unordered list.

11) What is the worst-case computing complexity for inserting into a: a) BST b) ordered array c) unordered array d) ordered list e) unordered list.

12) The following functions were written to find a key in a BST. Does each function work? If not, find all errors.

```
BT_NODE_PTR bstFindKey (const BT_NODE_PTR psBSTRoot, int key)
{
  BT_NODE_PTR psTemp = psBSTRoot;

  while (NULL != psTemp)
  {
    if (key == psTemp)
    {
      return psTemp;
    }
    else
    {
      bstFindKey (psTemp->psLeftChild, key);
      bstFindKey (psTemp->psRightChild, key);
    }
  }

  return NULL;
}
```

```
BT_NODE_PTR bstFindKey (const BT_NODE_PTR psBSTRoot, int key)
{
  BT_NODE_PTR psTemp = psBSTRoot;

  if (key != psTemp->data)
  {
    bstFindKey (psTemp, key);

    if (psTemp->data > key)
    {
      psTemp = psTemp->psLeftChild;
    }
    else
    {
      psTemp = psTemp->psRightChild;
    }
  }

  if (key == psTemp->data)
  {
    return psTemp;
  }
  else
  {
    return NULL;
  }
}
```

13) Write a function btCountNodes that returns the number of nodes in a Binary Tree. What does a call to your function look like?

14) Write a function btLargest that returns the largest value in a: a) BST b) BT. What does a call for each function look like?

15) Write a function lstIsEqual that accepts two list pointers of type NODE_PTR and returns TRUE if the two lists are the same; otherwise, FALSE is returned.

16) Review hash tables including: a) hash methods b) collision handling techniques, c) the concepts of primary and secondary clustering

17) What are the advantages of generic programming?

18) Make sure you understand the specifics of makefiles, pointers, handles, dynamic memory, activation records, the heap.

HASH TABLES

19) Use Open Address where f(i) = i as the collision handling technique to insert the follow values into a hash

table of length 11. The hash function is (N % 11).

Values: 11, 1, 0, 34, 43, 6, 32, 13, 12, 22

Highlight any primary clusters that arise.

20) Use Open Address where f(i) = i^3 as the collision handling technique to insert the following values into a hash table of length 11. The hash function is (N % 11).

Values: 11, 1, 0, 34, 43, 6, 32, 13, 12, 22

Highlight any primary clusters that arise.

21) Use Chaining as the collision handling technique to insert the follow values into a hash table of length 11. The hash function is (N % 11).

Values: 11, 1, 0, 34, 43, 6, 32, 13, 12, 22

22) What is the average access time for each element in 19?

23) What is the average access time for each element in 20?

24) What is the average access time for each element in 21?

25) There is a built-in quick sort function in C as follows:

```c
void qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *,
const void*));
```

- **base** - pointer to the first element of the array.
- **nitems** - number of items in the array
- **size** - size of each element in bytes
- **compar** - compare function that compares two integers

a) Describe each piece of the compar function in the above prototype.

b) Create an array of 100 integers filled with random values.

c) Write the appropriate compar function to help sort the array of integers in increasing order.

d) Show the call to qsort that sorts the array of integers in increasing order passing in your compar function.