CS300 Final Review Questions[1]

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 and exam reviews
- 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. **Zybook 13.5**

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 qCreate.
c) Write qEnqueue and qDequeue
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.

[1] Thanks to Doug Ryan for most of these questions!

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 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) Binary Tree (not sorted). 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.

15.1) Write a function bstIsEqual that accepts two Binary Search Trees and returns TRUE if the two trees are the same; otherwise, FALSE is returned. The same means contains the name values (not the trees are structured identically). You may use any auxiliary data structure you wish to help you.

15.2) Write a function bstIsIdentical that accepts two Binary Search Trees and returns TRUE if the two trees are structurally the same; otherwise, FALSE is returned. Structurally the same means the same value in each tree has the same parent and same children. You may use any auxiliary data structure you wish to help you.

15.5) Write a function to find the Nth smallest item stored in a BST. For example, the 1st smallest item is the smallest item in the BST. The 2nd smallest item is the next item in ascending order. You may use any auxiliary data structure you wish to help you.

15.75) Insert the following items into a B-Tree or order 4: 1, 100, 2, 200, 3, 400, 5, 500, 0, 1000

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:

```
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.

Review all of the Exams, previous Review Sheets, Assignments, Notes, Quizzes, and enigmatic utterances from class.

What factors would lead you to use a Queue rather than a Stack?
What are the advantages of using a Stack backed by an array? What are the disadvantages? What are the advantages of using a Stack backed by a Linked List?

Why would you ever write a function that accepts a handle as a parameter?
Why does lstCreate() need a pointer and not a handle?

Why is encapsulation a good idea? Explain how our projects this semester demonstrated encapsulation.

If you wanted to let your sibling use your GenericDynamicList, which file(s) would be the minimum you would have to give them: list.h, list.c, or list.o. (assume your sibling is compiling on the same Linux VM you are using).

What is a memory leak? Write code to demonstrate one.
What is a dangling pointer? Write code to demonstrate one.
Of the two memory errors above, memory leak and dangling pointer, which is more likely to cause a runtime error? Why?

TRUE FALSE 0xF0 & 0x0F
TRUE FALSE 0xF0 | 0x0F
TRUE FALSE 116 == 0x74

What is the Big Oh runtime of the following function? Explain why Big Oh complexity cannot predict the wall clock time of a function below. Be as specific as possible. Hint: give two data sets that have the same N but very different wall-clock run times.

```
int silly(int array[], int size)
{
   int i, k;
   int sum = 0;
   for( i = 0; i < size ; ++i)
   {
       sum += array[i];
       if( array[i] % 3 == 0)
       {
           k =0;
           while( k < 3 && k < size)
           {
                 sum += array[k++];
           }
      }
    }
    return sum;
}
```

Does a hash table with chaining have primary clusters?

What is a Perfect Hash?

Why does the following fail, and when does it fail?

```
void* pMystery;
int value = 9;

pMystery = &value;
printf("%d ", *pMystery);
```

Imagine the GenericDynamicList stored the size of memory each void* pData pointed to as shown below. This way, the user would not need to know the size of the data stored in each ListElement. The user could put chars and ints in the same List and the List could tell which Elements contained 1 byte and which contained 4 bytes.

```
typedef struct ListElement
{
        void *pData;
        int size;
        ListElementPtr psNext;
} ListElement;
```

How would the prototypes (return value and parameter list) for lstPeek() and lstInsertBefore() change?

Write each of these two functions.

What problems might arise if the user puts both ints and floats into the same list? Hint: sizeof(int), sizeof(float).

Write a function, int btPrintReverse(BT_NODE_PTR psRoot); that will print the values in the tree in reverse order.

Write a function, float btAvg; that will return the average of all of the ints in the tree rooted at psRoot. You get to define the parameters.

B-Trees
Why use a B-Tree rather than a BST? Rather than a Hashtable? Rather than a Linked List?

Why use a fan-out of greater than 2 for a B-Tree? Define fan-out.

Given a B-Tree of order 4, insert the value 100:

1	99	2000

Linux:
Practice the common Linux commands we used: ls, mkdir, cd, ssh, wget, scp

Valgrind: Friend or enemy?

What does git push do? Why is this necessary?

Explain why you can replace recursion with a loop and stack in BSTSumAllValue() which sums all the values in a BST. What does the stack store? How does the recursive version store the data you explicitly store in the stack?

Write a function, int sumVowelDistance(char *szMsg), which determines the sum of distances between each character in the string and the vowel closest to that character in the alphabet. For instance, the result of sumVowelDistance("data") is: d-e = 1, a-a=0, t-o = 5, a-a = 0 for a total of 6. Use appropriate data structures to make your code efficient.

Find and fix at least two errors (I would not have a "find the errors" problem on the exam).

```
int openAndRead(FILE *pFile, char *szName)
{
  int v;

  pFile = fopen(szName, "r");
  fscanf(pFile, "%d", v);

  return v;
}

int readInt(FILE *pFile)
{
  int z;
  fscanf(*pFile, "%d", z);
  return z;
}

int main()
{
  FILE *pFile;

  printf("%d\n", openAndRead(pFile, "data.txt"));
  printf("%d\n", readInt(pFile));
  fclose(pFile);
  return 0;
}
```

```
GenericDynamicList/bin/:

-rw-r--r-- 1 chadd users 16968 Nov 20 14:33 list.o

GenericDynamicList/include/:

-rw-r--r-- 1 chadd users 8577 Sep 26 10:47 list.h

GenericDynamicList/src/:

-rw-r--r-- 1 chadd users 11886 Nov 21 15:55 list.c
-rw-r--r-- 1 chadd users  4452 Sep 19 11:06 listdriver.c

GenericDynamicPriorityQ/bin:

-rwxr-xr-x 1 chadd users 32736 Nov 20 14:33 pqdriver1
-rw-r--r-- 1 chadd users  9952 Nov 20 14:33 pqdriver1.o
-rwxr-xr-x 1 chadd users 33176 Nov 20 14:33 pqMemTest
-rw-r--r-- 1 chadd users 10616 Nov 20 14:33 pqMemTest.o
-rwxr-xr-x 1 chadd users 32680 Nov 20 14:33 pqueuedriver
-rw-r--r-- 1 chadd users  9464 Nov 20 14:33 pqueuedriver.o
-rw-r--r-- 1 chadd users 13216 Nov 20 14:33 pqueue.o

GenericDynamicPriorityQ/include:

-rw-r--r-- 1 chadd users 5148 Sep 27 11:44 pqueue.h

GenericDynamicPriorityQ/src:

-rw-r--r-- 1 chadd users 7379 Sep 27 11:44 pqueue.c
-rw-r--r-- 1 chadd users 4744 Oct 17 13:00 pqueuedriver.c

GenericDynamicPriorityQ/testdrivers:

-rw-r--r-- 1 chadd users 4966 Oct 18 11:35 pqdriver1.c
-rw-r--r-- 1 chadd users 6174 Oct 18 11:35 pqMemTest.c
```

Makefiles!

Given the directory structure, files, and time stamps above, and the Makefiles on the next page, answer the following questions. Use proper Makefile terminology.

1. If the user types **make** in the directory GenericDynamicPriorityQ, which **gcc commands** will get executed and in what order?

2. For each gcc command run above, which pieces of data were used by make to decide to run that command? Be as complete and as specific as possible.

3. What should the clean rule look like for GenericDynamicPriorityQ?

4. What do each of the characters "-rw-r--r--" before list.o mean?

5. Assume step 1. above was successful. If the user types **make clean** in the GenericDynamicList directory and then types make in the directory GenericDynamicPriorityQ, which **gcc commands** will get executed and in what order?

6. Define dependency and target. Is clean in the Makefile for List a dependency or target?
   Why is the link blank after **clean:** ?

```
CC=gcc                  # MAKEFILE FOR LIST
CFLAGS=-Wall -g

all: bin/listdriver

bin/listdriver: bin/listdriver.o bin/list.o
        ${CC} ${CFLAGS} bin/listdriver.o bin/list.o -o bin/listdriver

bin/listdriver.o: src/listdriver.c include/list.h
        ${CC} ${CFLAGS} -c src/listdriver.c -o bin/listdriver.o

bin/list.o: src/list.c include/list.h
        ${CC} ${CFLAGS} -c src/list.c -o bin/list.o

clean:
        rm -f bin/*

######           MAKEFILE FOR PRIORITY Q

all: bin/pqueuedriver bin/pqdriver1 bin/pqMemTest

bin/pqueuedriver: bin/pqueuedriver.o bin/pqueue.o \
                          ../GenericDynamicList/bin/list.o
        ${CC} ${CFLAGS} -o bin/pqueuedriver bin/pqueuedriver.o bin/pqueue.o \
        ../GenericDynamicList/bin/list.o

bin/pqueuedriver.o: src/pqueuedriver.c include/pqueue.h
        ${CC} ${CFLAGS} -o bin/pqueuedriver.o -c src/pqueuedriver.c

bin/pqueue.o: include/pqueue.h src/pqueue.c \
                          ../GenericDynamicList/include/list.h
        ${CC} ${CFLAGS} -o bin/pqueue.o -c src/pqueue.c

../GenericDynamicList/bin/list.o: \
                                ../GenericDynamicList/include/list.h \
                                ../GenericDynamicList/src/list.c
        cd ../GenericDynamicList; make bin/list.o

bin/pqdriver1: bin/pqdriver1.o bin/pqueue.o ../GenericDynamicList/bin/list.o
        ${CC} ${CFLAGS} -o bin/pqdriver1 bin/pqdriver1.o bin/pqueue.o \
        ../GenericDynamicList/bin/list.o

bin/pqdriver1.o: testdrivers/pqdriver1.c include/pqueue.h
        ${CC} ${CFLAGS} -o bin/pqdriver1.o -c testdrivers/pqdriver1.c

bin/pqMemTest: bin/pqMemTest.o bin/pqueue.o ../GenericDynamicList/bin/list.o
        ${CC} ${CFLAGS} -o bin/pqMemTest bin/pqMemTest.o bin/pqueue.o \
        ../GenericDynamicList/bin/list.o

bin/pqMemTest.o: testdrivers/pqMemTest.c include/pqueue.h
        ${CC} ${CFLAGS} -o bin/pqMemTest.o -c testdrivers/pqMemTest.c
```