CS 300 Exam 3 Review

VOCABULARY Expect to write some code, read some code, explain concepts Read all your slides (even ones we skipped), Read the example code. Read your project code. Re-do the Exam 1 & 2 review!

What is a handle?

Which is bigger, a handle, a void*, or an int*? Justify your answer.

Draw a picture of memory when the code is at the marked line in the code. Make sure you provide the address and value for each variable in the program. Assume the stack starts at address 100,000 and the heap starts at address 20,000.

```
void practice(int x, int *pInt, int **hInt)
{
   int localInt, *pLocalInt, **hLocalInt;

  localInt = *pInt;
  *hInt = pInt;
  x++;
  pLocalInt = &localInt;
  // MARKED LINE
}

int main()
{
  int a = 9 , b = 10 , c = -3;
  int *pMainPtr;

  pMainPtr = & c;

  practice(a, pMainPtr,  & pMainPtr);
}
```

What is a deep copy? Why does a struct that contains a void* complicate a deep copy?

Why would you want to use a hash table to store data as opposed to 1) an array or 2) a list?

What operations on a hashtable to we care about the speed of?

Why would you want to use a queue as opposed to 1) a list or 2) a stack?

What properties do we want from a hash function used in a hash table? Justify each property.

Explain what a collision is during hashing. [1]

Why is it difficult to prevent a collision from ever happening?

Describe how linear probing can lead to poor access times for a Hash Table.

What scenario leads to the worst case performance for a hash table that using chaining?

Hash the keys M6, G7, Q21, Y77, R19, Z18, and F2 using the hash formula h(Kn) = n mod 7 with the following collision handling technique: linear probing, chaining
Compute the average number of probes to find an arbitrary key for each method

The tool gpref (http://www.gnu.org/software/gperf/) will generate C code that produces a perfect hash function. However, gpref requires that you provide every key you will ever hash. Why is this necessary?

An ordered list is pointed to by a pointer psList. The ordering is characters from smallest ASCII value to the largest ASCII value. Can we search this list for a specific character and return a pointer to the node, if the character exists, in O(log2N) time? Why or why not?

Using the List ADT, how would you implement the Stack functions?

A data file numbers.txt contains an unknown number of integers, one per line. Write a C program that opens the file and finds the average of all numbers read. The executable and data file both exist in the same directory. Further, the name of the data file is passed in through argv.

How does mid-square hashing differ from division hashing when determining the hash table size?

Declare a function pointer variable that can point to a function with the prototype:
int review(char c, int val);
Set that variable to point to review. Call review through that function pointer with the arguments ‘X’, 99. Print the return value.

In your Makefile, why does queue.o depend on pqueue.h but not list.h?

1 Thanks to Doug Ryan for most of this review!

In your Makefile, why does queue.o not depend on pqueue.c?

If your code crashed in memcpy(), how would you go about finding the cause of the error?

Could you build your Priority Queue on top of your Queue (the opposite of what we did in class)? What are the advantages and disadvantages of this opposite method?

Why did pqTerminate() need to do more work than just call lstTerminate()?

Did queueTerminate() need to do more than just call pqTerminate()? Why or why not?

What do we mean when we say Big Oh evaluates growth of the runtime but not actual runtime?

What is the computing complexity (Big Oh) of the following function. Be sure to define what n is.

```
int runtime(int array[], int size)
{
   int retVal = 0;

   for(int i = 0 ; i< size/2; i++)
   {
       for(int k = 0; k<size ; k++)
       {
         retVal += (array[k] + array[i]);
       }
   }
   for(int m = 0 ; m < 2*size; m++)
   {
       retVal -= array[m % size];
   }
   return retVal;
}
```

What issue(s) does the following code have?

```
void allocData(int **hData)
{
	*hData = malloc(sizeof(int));
	**hData = 8;
}

void alsoAllocData(int *pData)
{
	pData = malloc(sizeof(int));
	*pData = 9;
}

int main()
{
	int x;
	int *pInt;
	int *pAnotherPtr;


	pInt = & x;

	allocData(&pInt);

	pInt = malloc(sizeof(int));

	allocData(&pInt);

	alsoAllocData(pAnotherPtr);
}
```