CS 300 Exam 2 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 review!

Dynamic Memory/Pointers:

Why is the type void * useful? Why can it be tricky to use?

Given:

```
struct Book
{
   char szTitle[100];
   int pages;
};
```

What value do you expect sizeof(struct Book) to return? Explain how you calculated that value.

How is pass by reference achieved in C? Specify what is provided/expected in the function and in the function call.

How are the * and -> operators related?

Why is it risky to return, from a function, the address of a local variable?

Did our lstPeek(ListPtr, void*, int) create a copy of the data in the ListElement or pass back a pointer to the data in the ListElement? Did lstPeek() contain a call to malloc()? If not, why not? If so, for what purpose?

Draw memory

```
int x = 1, y = 2;
int *pX, *pY;
int **hX, **hY;

pX = &x;
pY= &y;
hX = &pX;
hY = &pY;

// fill in the appropriate format string specifiers (%d, %f, %x, %p, %c, %s) below
printf("                ", x, pX, *pX, hX, *hX, **hX);
// what is the output of the above printf?
```

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);
}
```

Define memory leak. Write a snippet of code that clearly produces a memory leak.

What is a dangling pointer? Write a snippet of code that clearly produces a dangling pointer. How do we guard against such an error?

ADTs:

What does the abstract in ADT mean? What, exactly, is abstracted away? Why do we want to abstract this away? Explain how our two implementations of a Queue (array based and linked list based) demonstrate the notion of “abstract”.

Define a data type (struct) to represent a null terminated string backed by a List. Write a function strCreate() to create the string. Write a function strAppend(StringPtr, char) to append a character to an existing string.

List ADT

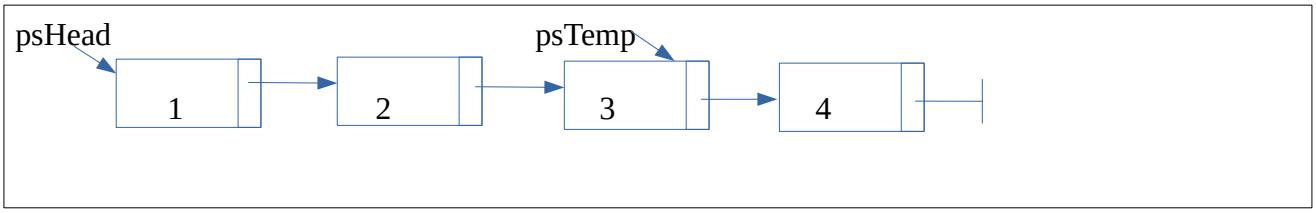
Singly linked list. Doubly linked list,

Write code to use your List project to:
fill a List with 100 random ints (srand()/rand())
search that list to determine if 300 is in the list
delete all the odd integers from the list.

Draw a picture of your List after each of the following function calls: lstCreate(), insertBefore(1), insertAfter(2), insertBefore(0).

Write a function lstSplit(ListPtr original, ListPtr endList) to split the List original in half. The first half of the list must remain in original, the second half of the list is in endList. If the list has an odd number of elements, endList should be one larger than original. Is malloc and/or free required for this function? Why or why not?

Given:



If you want to insert a node after between node 3 and 4, describe exactly how each relevant pointer would need to be updated and in what order so as to not break the chain.

If you only have psHead and psTemp, why is it difficult to insert a node immediately before node 3?

What happens if you set psHead = psHead→psNext?

Redraw the list above as a doubly linked list. If you want to insert a node after between node 2 and 3, describe exactly how each relevant pointer would need to be updated and in what order.

Queue ADT

Array based, Linked list based.

Draw the circular array based queue after each of the following operations:
qCreate()
qEnqueue(1);
qEnqueue(2);
qDequeue();
qEnqueue(3);
qDequeue();

Draw the linked list based queue after each of the following operations:
qCreate()
qEnqueue(1);
qEnqueue(2);
qDequeue();
qEnqueue(3);
qDequeue();

Why did we prefer to be able to write our enqueue() and dequeue() functions without using any loops? How did we achieve this?

What access restrictions are placed on a Queue?

What is meant when someone says “my queue is backed by an array?” Is it sensible to say “my queue is backed by a stack?” Why or why not?

Is it sensible to say “my stack is backed by a queue?” Why or why not?

Complexity

What is the Big-Oh runtime of lstTerminate()? What is n in this situation?

What is the Big-Oh runtime of pqTerminate()? What is n in this situation?

What does it mean that our runtime analysis is based on growth of the runtime?

What is the Big-Oh runtime of the following code?

```
void nonsense(int *aIntArray, int arraySize)
{
   int i, k;

   for(i=0;i<arraySize/2;i++)
   {
     for(k=0;k<arraySize / 4 ;k++)
     {
       aIntArray[k] = aIntArray[i] *2;
     }
   }
   for(i=0;i < (arraySize * 4);i++)
   {
     aIntArray[ (i+1) % arraySize ] = aIntArray[i % arraySize];
   }

   for(i=0;i<arraySize;i++)
   {
     aIntArray[0] += aIntArray[i];
   }
}
```

Other Topics

What is the difference between static and dynamic memory? Give an example of each. When is a good time to use each? What are the disadvantages of each? What are the advantages of each?

What does the Valgrind error “invalid write” indicate?

Write a function that accepts a ListPtr and returns a pointer to an exact copy of the list that was passed in. Assume ListElement contains an int rather than a void*.

Write a function that accepts a ListPtr and returns a pointer to an exact copy of the list that was passed in. This time, ListElement contains a void* as in your project. What problems could arise from using this function? How might you solve these problems (you may modify List, ListElement, add functions, add parameters, whatever you need).

TRUE FALSE I am excited for the TRUE FALSE questions on the exam.