Dynamic Memory in L3 SP23

Contents

Dynamic Memory in L3 SP23 1

Reading Balls – allocation 1

Clearing the lists 2

The Linked List Allocations and Diagnostics 2

Reading Balls – allocation

Our bits are read in to a structure. That structure will have the bits and all of the position and velocity data. Input can have one statically allocated ball structure, but that is a temporary holding place. Lab 3 will dynamically allocate ball structures to put on the list. So when input reads a ball, it will hand the ball data off to code that copies it to dynamically allocated space.

The dynamic memory code will live in its own file. Any function that calls an allocator or that calls free will be in this file.

The dynamic allocation code will need to use malloc or calloc to create space for the structure that holds the bits. Such code will **always** check the return value to make sure that space was allocated.

If allocation fails, the code must not crash. The error recovery strategy we will use is that we will be unable to do anything with the ball we read in, but we will go back to reading as long as the file has data. Our code won't attempt to change the situation. It will attempt to go on with the next ball if there is one and it will attempt to run the sim if any balls made it onto the table. Real-world production code needs error handling strategies better than simply crashing. Our method is a hand wave to the idea that we are doing error recovery.

If dynamic allocation succeeded, the ball structure will need to have its simulation structure pointer set along with the bits and initial position data. Our standard load message from lab 2 gets generated at this time as well.

The allocation function must keep a static int that counts the number of times allocation succeeds. That int gets incremented every time allocation succeeds. Since it is declared static, it retains its value between function calls. In TEXT mode, it will print a DIAGNOSTIC message with the updated count. Be sure ***not*** to use a global variable for this. The text output on piazza shows these messages.

```
Loaded 35 6 ball at  -2.00000,   1.50000 -90.00000 deg  90.00000 ips
DIAGNOSTIC: 1 balls allocated
DIAGNOSTIC: 1 nodes allocated.
Launched B5 6 ball at  -2.00000,   1.50000 at   0.00000, -90.00000
```

See the output document for the required diagnostic messages.

Clearing the lists

When the simulation terminates and final output has been done, the dynamic data needs to be freed. Your dynamic memory file will have code that handles freeing the memory for a single ball, but it doesn't want to know how to clear lists. See the output document for what gets printed.

Each of the two lists will need a deleteSome call to get any ball structures off of them. Thos starts a chain of calls that end up calling into your dynamic memory function that frees the ball. That function will be the only code that is allowed to call free. That function must also keep a static int, but in this case it counts how many times it has been called to free a ball structure. And like the allocation code, in TEXT mode it will generate a DIAGNOSTIC message showing the free count. Again, the sample output shows this message.

```
DIAGNOSTIC: 4 balls freed
```

When this is finished, the count of allocations and the count of frees must match!

Here is a final output sample:

```
Deleted 0 balls from in play list.
DIAGNOSTIC: 1 balls freed
DIAGNOSTIC: 5 nodes freed.
DIAGNOSTIC: 2 balls freed
DIAGNOSTIC: 6 nodes freed.
DIAGNOSTIC: 3 balls freed
DIAGNOSTIC: 7 nodes freed.
DIAGNOSTIC: 4 balls freed
DIAGNOSTIC: 8 nodes freed.
Deleted 4 balls from off table list.
```

Lab 3 code generates the "deleted" lines and the "ball" messages. The linked list library generates the "node" messages.

The Linked List Allocations and Diagnostics

The list allocates and frees <u>node</u> structures. In TEXT mode it also generates diagnostic messages with the counts. In a correct lab, each ball structure is on each of the lists in succession, so the number of nodes allocated and freed should be twice the number of bit structures allocated and freed.

You do not have to write any code to generate linked list diagnostic messages, they are built in to the library code of insert and deleteSome. See the output just above. Each ball lives in the in-play list before it moves to the off-table list, so each ball causes 2 nodes to be used, meaning 4 balls need 8 nodes.