Homework 4
COMP 3220

Brandon Hurler 9/22/2014

Due: September 22nd, Monday by 11:59PM
Please submit as a PDF or WORD document using Canvas

(1. 10pts) Question 6.4 (page 314) from the textbook.

a. Tombstones are pointers to the heap-dynamic variable. The actual pointer variable points only at tombstones and never to heap-dynamic variables. When a heap-dynamic variable is de-allocated, the tombstone remains but is set to nil, indicating that the heap-dynamic variable no longer exists. This prevents a pointer from ever pointing to a de-allocated variable, through which we can infer that any pointer which points to a nil tombstone is in error. However, because tombstones are never de-allocated, their storage is never reclaimed making them inefficient in terms of space. Tombstones are also time inefficient. Every access to a heap dynamic variable through a tombstone requires an additional level of indirection, requiring an additional machine cycle on most computers.

The locks-and-keys approach has pointer values, an integer key and address. The heap-dynamic variables are represented as the storage for the variable, plus a header cell that stores an integer lock value. When a heap-dynamic variable is allocated, a lock value is created and placed both in the lock cell of the heap-dynamic variable and in the key cell of the pointer that is specified in the call to new. Every access to the dereferenced pointer compares the key value of the pointer to the lock value in the heap-dynamic variable. If they match, the access is legal; otherwise the access is treated as a run-time error. Any copies of the pointer value to other pointers must copy the key value. Therefore, any number of pointers can reference a given heap-dynamic variable. When a heap-dynamic variable is de-allocated, its lock value is cleared to an illegal lock value. If a pointer other than the one specified in the dispose is dereferenced, it will retain its address value, but its key value will no longer match the lock, and access will be denied. This method is better than tombstones in terms of being able to recover the lost storage space; however, additional space must be allocated for the ordered pairs to be stored. Also, this method may not be as safe as tombstones since the keys are user generated and could be improperly done.

(2. 30pts) Question 6.10 (page 314) from the textbook.

Let us name the subscript ranges of the three dimensions as follows: min(1), min(2), min(3), max(1), max(2), and max(3). We can then let the sizes of the subscript ranges be size(1), size(2), and size(3) and assume the element size is 1.

a. Row Major Order

i. location(a[i, j, k]) =

(address of a[min(1), min(2), min(3)]) + ((i - min(1)) * size(3) + (j - min(2))) * size(2) + (k - min(3))

b. Column Major Order

    i. location(a[i, j, k]) = (address of a[min(1), min(2), min(3)]) + ((k - min(3)) * size(1) + (j - min(2))) * size(2) + (i - min(1))

(3. 20pts) Question 6.13 (page 314) from the textbook.

a. Java's reference pointers are self-managed and safer than C++'s. This has its own pros and cons; in C++ pointers can become dangling or the information stored could be dereferenced and lost, becoming garbage. The benefit to C++'s reference pointers is that they can perform pointer arithmetic and be more efficient. The major differences are that C++'s pointers are more versatile; however they need to be explicitly managed in terms of garbage collection and pointer allocation and deallocation, whereas the pointers are automatically handled in Java.

(4. 20pts) Question 6.14 (page 314) from the textbook.

a. With the choice to not include the pointers of C++, Java gained implicit deallocation, or garbage collection. This allowed Java to be safer since it no longer had the chance of having garbage or dangling pointers due to human error. However, Java lost a lot of the flexibility and versatility that the C++ pointers offered.

(5. 20pts) Question 6.15 (page 314) from the textbook.

a. Pros: Java's implementation of implicit heap storage recovery avoids dangling pointers and memory leaks associated with uncollected garbage that is allowed by the explicit heap storage recovery of C++ since all of the pointers and allocated space does not need to be explicitly managed.

b. Cons: Java's implementation of implicit heap storage recovery is more time consuming as memory is frequently being checked to see if previously allocated space is now available for garbage collection. In C++, this cost is explicitly managed and does not have as many unnecessary memory checks for garbage collection, allowing for faster time.