John Carroll
jcc0044
902521946

Homework 4

6.4) Compare the tombstone and lock-and-key methods of avoiding dangling pointers, from the points of view of safety and implementation cost.

Ans:

Tombstones are a safe way of handling dangling pointers because pointers are never allowed to be invalid. The way tombstones work is every heap dynamic variable includes a special cell, called a tombstone, which is a pointer to the heap-dynamic variable. If a variable is deleted, the tombstone is set to null, and will represent that there is no data at that point in the linked list. The tombstone method can slow programs down, and be very costly. It requires time to check the state of the tombstone and it can never be deleted after it is created.

The lock-and-key method is not as safe as the tombstone method, because the key values are typically user generated. In this method pointer values are set as ordered pairs consisting of an integer key and address. When a variable is allocated a lock variable is created. If the values are equal then access is granted. If the variable is deal located, the key is changed to a different value so as not to allow access. This approach can also be costly because of memory usage to store the ordered pairs.

6.10) Multidimensional arrays can be stored in row major order, as in C++, or in column major order, as in Fortran. Develop the access functions for both of these arrangements for three-dimensional arrays.

Ans:

Let the subscript ranges of the three dimensions be named min(1), min(2), min(3), max(1), max(2), and max(3). Let the sizes of the subscript ranges be size(1), size(2), and size(3). Assume the element size is 1.

<u>Row Major Order:</u>

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))

<u>Column Major Order:</u>

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))

6.13) Analyze and write a comparison of using C++ pointers and Java reference variables to refer to fixed heap-dynamic variables. Use safety and convenience as the primary considerations in the comparison.

Ans:

Pointers in C++ are more versatile, but not as safe as Java references variables. This is because there is no protection from dangling pointer or lost heap-dynamic variable problems. Due to human error pointers can end up being dangling pointers, because they must be explicitly allocated and de-allocated. Java's reference variables are automatically de-allocated upon deletion. This means Java's heap-dynamic variables are safer.

6.14) Write a short discussion of what was lost and what was gained in Java's designers' decision to not include the pointers of C++.

Ans:

By not allowing pointers in Java, it keeps human error down which could result in errors (such as bad pointer arithmetic), but it also slows down any given program. Java does not allow dangling pointers or lost heap-dynamic variable issues. In addition to the losses, from not including pointers comes the freedom to pointer arithmetic.

6.15) What are the arguments for and against Java's implicit heap storage recovery, when compared with the explicit heap storage recovery required in C++? Consider real-time systems.

Ans:

Java's implicit heap storage recovery or "garbage collection" eliminates the creation of dangling pointers through explicit de-allocation operations, such as delete. The disadvantage of implicit heap storage recovery is that it allows for memory leakage which is the execution time cost of doing the recovery, often when it is not even necessary (there is no shortage of heap storage). Java can be slower than C++ because of this. This is also an arguing point for programmers' preferences as some would prefer to allocate and de-allocate memory themselves, and some would prefer the system to do it.