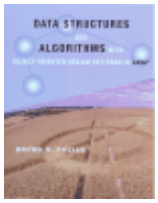

Stop-and-Copy Garbage Collection

The section describes a garbage collection approach that collects garbage *and* defragments the heap called *stop-and-copy* . When using the stop-and-copy garbage collection algorithm, the heap is divided into two separate regions. At any point in time, all dynamically allocated object instances reside in only one of the two regions--the active region. The other, inactive region is unoccupied.

When the memory in the active region is exhausted, the program is suspended and the garbage-collection algorithm is invoked. The stop-and-copy algorithm copies all of the live objects from the active region to the inactive region. As each object is copied, all references contained in that object are updated to reflect the new locations of the referenced objects.

After the copying is completed, the active and inactive regions exchange their roles. Since the stop-and-copy algorithm copies only the live objects, the garbage objects are left behind. In effect, the storage occupied by the garbage is reclaimed all at once when the active region becomes inactive.

As the stop-and-copy algorithm copies the live objects from the active region to the inactive region, it stores the objects in contiguous memory locations. Thus, the stop-and-copy algorithm automatically defragments the heap. This is the main advantage of the stop-and-copy approach over the mark-and-sweep algorithm described in the preceding section.

The costs of the stop-and-copy algorithm are twofold: First, the algorithm requires that *all* live objects be copied every time garbage collection is invoked. If an application program has a large memory footprint, the time required to copy all objects can be quite significant. A second cost associated with stop-and-copy is the fact that it requires twice as much memory as the program actually uses. When garbage collection is finished, at least half of the memory space is unused.