11.10
OCaml uses a combination generational/reference counting garbage collection scheme. There are two
generations, the first of which is contained in a partition of the heap called the “minor heap” and the
second generation is moved to the other partition called the “major heap”. Automatic garbage
collection is only called at the time of an allocation request when the minor heap is full.

11.11
Generational garbage collection is a method of automatic memory management which cleans objects in
an order depending on their age. The design of this method banks on the idea that a typical program
generally has some objects which are frequently allocated and immediately disposed of, and some
objects which tend to hang around much longer during the life cycle of the application. If we can bet
that an object which has lived for a certain threshold of time will likely live on even longer, then it
makes sense to avoid checking that object as often as we check the brand new allocations.

The method of determining which objects may be disposed of varies among implementations of
generational garbage collection. As I mentioned above, OCaml uses reference count to determine this.
Of course, this method has the same disadvantage as the typical reference counting garbage collection
scheme in that we face the possibility of memory leaks from circular references. This problem is
avoided by implementations which determine the disposability of an object based on reachability.
These methods look for reference paths from active, in-memory objects to objects in the heap. If no
trace of references can be made to a memory allocation from an in-memory object, the garbage
collector determines that it may be cleaned. This way, even if a circular reference exists, the objects
involved will still be wiped from memory once they are no longer in use.

Though we can avoid the overhead of keeping a pointer counter for every memory allocation, the
generational garbage collection scheme does have some disadvantages. For example, the aging of
objects is sort of a shot in the dark and could potentially use up a lot of unneeded memory for longer
periods of time than needed if the we age objects shortly before they are due to expire. Also, there is a
certain amount of memory allocated for older generations which may never be needed if all of the
objects created in a program are disposed of quickly. So it is possible that we could run out of space,
still have plenty of space available in the higher generations, and be stuck waiting on the garbage
collector to piece through the lowest generation.

I got most of my information from Wikipedia, the OCaml Tutorial (ocaml-tutorial.org) and an MSDN