**ARC (Automatic Reference Counting)**

Automatic Reference Counting (ARC) is used by Swift to track and manage the memory usage of an app. ARC frees up memory space automatically that is used by class instances when they are no longer needed. ARC also uses the freed-up memory for other purposes. With this, class instances won’t take up a lot of space when they’re not needed.

ARC takes in a large chunk of memory to store information about that specific instance. The memory has a hold on information about the instance along with the values of stored properties. ARC would free up memory that is no longer needed so that the memory can be used for other purposes.

Every instance (of a class) has a property known as reference count. If the reference count is greater than 0, the instance would be kept within the memory, whereas vice versa, it will be removed from the memory.

If ARC were to deallocate an instance that is currently in use, it would be impossible to access those instances properties or methods. However, if you try to gain access to that instance, the application is most likely to crash.

ARC tracks properties, constants and variables that are referring to class instances to ensure instances don’t disappear. ARC won’t deallocate instances if there is at least one active reference exists to that instance.

Once a class instance is assigned to a property, constant or variable, this makes a strong reference to the instance. This is known as a strong reference as it keeps a strong firm on the instance and won’t allow it to be deallocated to if the strong reference remains.

The strong reference is a default reference type. When a class instance is assigned to a variable or constant, that makes a strong reference.

Weak reference doesn’t have a strong grip on the instance it refers to. Weak reference prevents strong reference cycles in between class instances. ARC sets a weak reference automatically to a nil when the instance is deallocated.

Unowned is similar to a weak reference. Unowned reference doesn’t keep a strong hold on the instance. This reference always has a value. ARC does not set unowned reference type to a nil. It is defined using a nonoptional type. If a unowned references value is a accessed after the instance has been deallocated, there will be a runtime error.

One concern with ARC is that it doesn’t detect when an object refers to each other cyclically. Objects that aren’t referenced anywhere in the program, may not get deleted when they are still referenced to each other.

ARC is a form of a garbage collector. The difference between garbage collector and ARC is that a garbage collector is a system that permits a programmer to allocate resources without the need of deallocating them. ARC has only one technique. All objects get marked with a count of how many objects maintain a reference to it. Once the reference count reaches zero, the object is deallocated.