Memory management functions and its usage are handled in Swift language through Automatic reference counting ARC. ARC is used to initialize and deinitialize the system resources thereby releasing memory spaces used by the class instances when the instances are no longer needed. ARC keeps track of information about the relationships between our code instances to manage the memory resources effectively.

**Functions of ARC**

- ARC allocates a chunk of memory to store the information each and every time when a new class instance is created by init.
- Information about the instance type and its values are stored in memory.
- When the class instance is no longer needed it automatically frees the memory space by deinit for further class instance storage and retrieval.
- ARC keeps in track of currently referring class instances properties, constants and variables so that deinit is applied only to those unused instances.
- ARC maintains a 'strong reference' to those class instance property, constants and variables to restrict deallocation when the class instance is currently in use.

**ARC Program**

```swift
class StudDetails {
    var stname: String!
    var mark: Int!

    init(stname: String, mark: Int) {
        self.stname = stname
        self.mark = mark
    }

    deinit {
        println("Deinitialized \(self.stname)")
        println("Deinitialized \(self.mark)")
    }
}

let stname = "swift"
let mark = 98
println(stname)
println(mark)
```

When we run above program using playground, we get following result.

```
swift
98
```

**ARC Strong Reference Cycles Class Instances**

```swift
class studmarks {
    let name: String
    var stud: student?

    init (name: String) {
        println("Initializing: \(name)")
        self.name = name
    }

    deinit {
```
```swift
println("Deallocating: \(self.name)")
}

class student {
    let name: String
    var strname: studmarks?

    init (name: String) {
        println("Initializing: \(name)"
        self.name = name
    }
    deinit {
        println("Deallocating: \(self.name)")
    }
}

var shiba: studmarks?
var mari: student?

shiba = studmarks(name: "Swift")
mari = student(name: "ARC")

shiba!.stud = mari
mari!.strname = shiba
```

When we run above program using playground, we get following result.

```
Initializing: Swift
Initializing: ARC
```

**ARC Weak and Unowned References**

Class type properties has two ways to resolve strong reference cycles:

- Weak References
- Unowned References

These references are used to enable one instance to refer other instances in a reference cycle. Then the instances may refer to each and every instances instead of caring about strong reference cycle. When the user knows that some instance may return 'nil' values we may point that using weak reference. When the instance going to return something rather than nil then declare it with unowned reference.

**Weak Reference Program**

```swift
class module {
    let name: String
    init(name: String) { self.name = name }
    var sub: submodule?
    deinit { println("\(name) Is The Main Module") }
}

class submodule {
    let number: Int
    init(number: Int) { self.number = number }
    weak var topic: module?
    deinit { println("Sub Module with its topic number is \(number)") }
}

var toc: module?
var list: submodule?
```
When we run above program using playground, we get following result.

ARC Is The Main Module
Sub Module with its topic number is 4

**Unowned Reference Program**

class student {
    let name: String
    var section: marks?

    init(name: String) {
        self.name = name
    }

    deinit { println("\(name)") }
}
class marks {
    let marks: Int
    unowned let stname: student

    init(marks: Int, stname: student) {
        self.marks = marks
        self.stname = stname
    }

    deinit { println("Marks Obtained by the student is \(marks)") }
}
var module: student?
module = student(name: "ARC")
module!.section = marks(marks: 98, stname: module!)
module = nil

When we run above program using playground, we get following result.

ARC
Marks Obtained by the student is 98

**Strong Reference Cycles for Closures**

When we assign a closure to the class instance property and to the body of the closure to capture particular instance strong reference cycle can occur. Strong reference to the closure is defined by 'self.someProperty' or 'self.someMethod'. Strong reference cycles are used as reference types for the closures.

class HTMLElement {
    let samplename: String
    let text: String?

    lazy var asHTML: () -> String = {
        if let text = self.text {
            return "<\(self.samplename)><\(text)</\(self.samplename)>"
        } else {
            return "<\(self.samplename) />
        }
    }
}
init(samplename: String, text: String? = nil) {
    self.samplename = samplename
    self.text = text
}

deinit {
    println("\(samplename) is being deinitialized")
}

var paragraph: HTMLElement? = HTMLElement(samplename: "p", text: "Welcome to Closure SRC")
println(paragraph!.asHTML())

When we run above program using playground, we get following result.

<p>Welcome to Closure SRC</p>

**Weak and Unowned References**

When the closure and the instance refer to each other the user may define the capture in a closure as an unowned reference. Then it would not allow the user to deallocate the instance at the same time. When the instance sometime return a 'nil' value define the closure with the weak instance.

```swift
class HTMLElement {
    let module: String
    let text: String?

    lazy var asHTML: () -> String = {
        [unowned self] in
        if let text = self.text {
            return "<\(self.module)>\(text)</\(self.module)>"
        } else {
            return "<\(self.module) />"
        }
    }

    init(module: String, text: String? = nil) {
        self.module = module
        self.text = text
    }

deinit {
    println("\(module) the deinit()")
}
}

var paragraph: HTMLElement? = HTMLElement(module: "Inside", text: "ARC Weak References")
println(paragraph!.asHTML())
paragraph = nil
```

When we run above program using playground, we get following result.

<Inside>ARC Weak References</Inside>

```
Inside the deinit()
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js