```
== numerics ==
  int  int8  int16  int32  int64
  uint uint8 uint16 uint32 uint64 uintptr

  float32 float64

  math.Max<T> math.Min<T>, eg. math.MaxInt32

== words ==
  string
- []rune(str) : convert to a []rune

  byte // alias for uint8
- buffers will typically pass one in to write to
  rune // alias for int32 & unicode codepoint
- '' literals

== maps ==
  make(map[<key>]<T>)
  if v, exists := aMap[val]; exists { ... }
- iteration is not a guaranteed order

== slices ==
  make([]<T>, len, cap)
  s[index:len]
  append([]<T>, <T>)
  copy(dst, src []<T>) // returns # of ele copied, shorter
  sort.Slice(s, func(i, j int) bool { return s[i] < s[j] })

== Queue ==
FIFO : just use a slice, append and pop from beginning
  a = append(a, x)
  x, a = a[0], a[1:]

LIFO : just use a slice, append and pop
  a = append(a, x)
  x = a[len(a)-1]     // may need to be zero'd for gc
  a = a[:len(a)-1]

== heap (or PriorityQueue) ==
- import "container/heap"
- implement the interface : type <Name> []int
  Len() Less() Swap() Push(<T>) Pop() <T>
  `heap.Init()`, `heap.Push()`, `heap.Pop()`
```

```
== linked list ==
__recursion impl__              |      __loop impl__
func main(head *Node) *Node {   |  func main(head *Node) *Node {
  return traverse(head)         |      if head == nil {
}                               |        return head
                                |      }
func traverse(n *Node) *Node {  |
  if n == nil {                 |      cur, next := head, head.Next
    return nil                  |      for next != nil {
  }                             |        if next.Val == 7 {
  // recursively drop a node    |          next = next.Next
  if n.Val == 7 {               |          cur.Next = next
    return traverse(n.Next)     |        } else {
  }                             |          cur = next
                                |          next = next.Next
  // otherwise traverse         |        }
  n.Next = traverse(n.Next)     |      }
  return n                      |      return head
}                               |  }


== Binary Trees ==
__recursion dfs__               |      __loop dfs__
func main(root *Node) bool {    |   func main(root *Node) bool {
  return dfs(root)              |      n := root
}                               |
                                |      for n != nil {
// dfs to search for 7          |        if n.Val == 7 {
func dfs(n *Node) bool {        |          return true
  if n == nil {                 |        }
    return false                |        if n.Val > 7 {
  }                             |          n = n.Right
                                |        } else {
  if n.Val == 7 {               |          n = n.Left
    return true                 |        }
  }                             |      }
  if n.Val > 7 {                |      return false
    return dfs(n.Right)         |   }
  }                             |
  return dfs(n.Left)            |
}                               |

__loop bfs__
- FIFO queue of L + R nodes and pop for traverse order
```