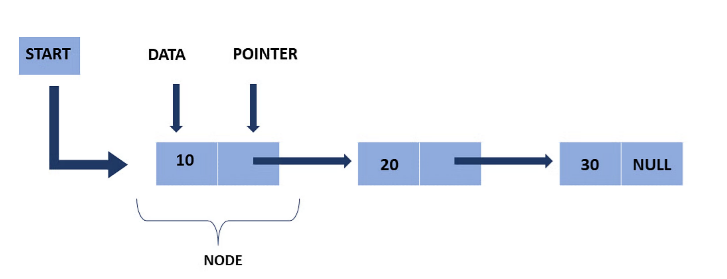
**Strukture podataka – teze**

1. **Singly Linked List**

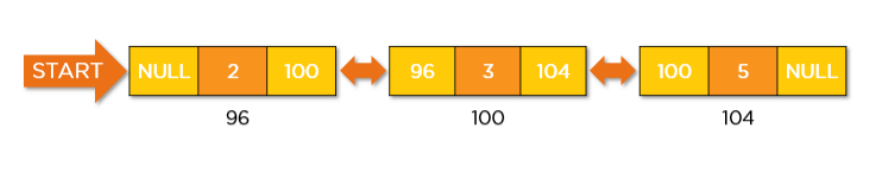
* linear data structure
* handles dynamic data elements
* does not have fixes size
* stores a collection of data elements dunamically
* data element know as NODE
* each NODE has two fields: data + an adress that keeps a reference to the next node (link/pointer)
* last node – null because it will point to no node
* grow and shrink its size, as per requirement – easy to add or remove elements
* NOT WASTE MEMORY SPACE!
* allow elements to be scattered throughout memory



* START pointer – stores the address of the firt node
* END pointer – null pointer
* limits:
  + extra memory needed for pointers (inefficiency)
  + to reach a node, ypu have to go through evry node before it
  + soritng is complex
* Insertions at the beginning – O(1)
* Inserting at the end - O(n) (if you don’t know pointer at last node)
* Search element – O(n) traverse each node
* Access element – O(n) need to follow the chain (Array faster! O(1))
* Deleting element from beginning – O(1)
* Deleting element from middler or end – O(n)
* USE CASE: great for situations where you freaquently add or remove elements ( lists of users for web app)
* Essential Operations:
  + Traversing
  + Insertion
  + Deletion
  + Searching
  + Reverse

1. **Doubly Linked List**

* bidirectional linked list - each node points to both its **next** and **previous** nodes
* deck of cards
* head to tail - traversal in both directions
* easily assign or reassign memory
* for implementation complex data structures (like stacks, binary trees..)



* requires more space for each node because nodes have extra pointer
* insertion, delition SLOWER than singly linked lists
* random stored in memory - need to be accessed sequentially
* USE CASE: for implementation of stacks, queues or chaches