 # DAY-04:

+ Limitations of an array data structure:

- array is "static" i.e. size of an array cannot be either shrink or grow during runtime.
    e.g. int arr[100];
        95 ele's are there
        we want to add new ele at index - 10
        we need to shift all ele's towards its right by 1 pos
- addition & deletion operations on an array are not efficient as these operations takes O(n) time.

- and to overcome these limitations of array data structure "linked list" data structure has been designed.

+ "linked list": it is a collection/list of logically related similar type of elements in which:
    - address of first element always gets stored into a pointer variable referred as "head" pointer, and
    - each element contains data (of any primitive/non-primitive type) and contains addr of its next (as well as prev) element in that list.

- in a linked list element is also called as a "node".
- basically there are two types of linked list:
    1. singly linked list: it is a type of linked list in which each node contains addr of its next node in a list.
        i. singly linear linked list
        ii.singly circular linked list

    2. doubly linked list: it is a type of linked list in which each node contains addr of its next node as well addr of its prev node in that list.
        i. doubly linear linked list
        ii.doubly circular linked list

---

i. singly linear linked list
ii. singly circular linked list
iii. doubly linear linked list
iv. doubly circular linked list

i. "singly linear linked list": it is a linked list in which
    - head always contains addr of first node, if list is not empty
    - each node has two parts:
    1. data part: contains actual data of any primitive/non-primitive type
    2. pointer part(next): contains addr of its next node
    - last node's next part points to NULL.

* "traversal" of a linked list: to visit each node in a list atmost once sequentially from first node till last node.
* "traversal" on an array -- to scan array ele's sequentially from first position to last position.

- basically we can perform addition and deletion operations onto the linked list
1. addition: we can add node into the singly linear linked list by 3 ways:
    1. add node into the list at last position
    2. add node into the list at first position
    3. add node into the list at specific position

1. add node into the list at last position:

    sizeof(int *) = 4 bytes scale factor(int *): 4 bytes
    sizeof(char *) = 4 bytes scale factor(char *): 1 byte
    sizeof(double *) = 4 bytes scale factor(float *): 4 bytes
    sizeof(float *) = 4 bytes scale factor(double *): 8 bytes
    sizeof(node_t *) = 4 bytes scale factor(node_t *): 8 bytes
    sizeof(type *) = 4 bytes - 32 bit compiler
    type may be any primitive/non-primitive

==========

DAY-05:
- SLLL:
    - add node at last position
    - add node at first position
    - add node at specific position
    - display list
    - traversal operation
    - delete node:
    - we can delete node from a linked list (SLLL) by three ways:
        - delete node at first position
        - delete node at last position
        - delete node at specific position

+ limitations of SLLL:
    - in SLLL we cannot access prev node of any node from it.
    - in SLLL we can traverse only on a forward direction
    - add_last() -- O(n)
    - delete_last() -- O(n)
    - in SLLL we cannot revisit any node: to overcome this limitation
    SCLL has been designed.

ii. "singly circular linked list": it is a linked list in which
    - head always contains addr of first node, if list is not empty
    - each node has two parts:
    1. data part: contains actual data of any primitive/non-primitive type
    2. pointer part(next): contains addr of its next node
    - last node's next part points to first node.

exit:
- when any program gets exited that program need to return an exit status to an OS

0 -> successful termination
1 -> erroneous termination: malloc() failed
-1 -> abnormal termination: divide by zero

res = 10/0;