CS 2700 PROGRAMMING & DATA STRUCTURES.

WEEK 4 : LISTS

GOALS FOR THE WEEK

1) ABSTRACT DATA TYPES (ADTS)

2) LIST ADT

3) IMPLEMENTING A LIST ADT

4) APPLICATIONS.

WHAT IS AN ADT / WHY ADTS?

ADT IS A MATHEMATICAL ABSTRACTION OF DATA.

- FOCUS ON WHAT AND NOT ON HOW.
  - example: int data type in C / C++
    supports +, -, mod, /, *

details of implementation are hidden / abstracted

- DEFINING AND USING ADTS enables modular programming

BASIC DATA TYPES

int, float, char

bool (C++)

USER DEFIND DATA TYPES

struct student { //members }

struct node { ... }

struct rectangle { }

ABSTRACT DATA TYPES.

Student ADT : what interface does it provide?

- setRollNo, getRollNo
- setcourses, getcourses, getDept, setdept

LIST ADT

$A_1, A_2, A_3, \ldots A_n$

- general list

A_{i-1}, A_i, A_{i+1}

A_{i-1} ↙ predecessor

A_{i+1} ↘ successor

- Operations on list
  1. add an element
  2. delete an element
  3. search for an element
  4. print list

LIST ADT.

OPERATIONS :

1. SIZE
2. INSERT ELEMENT val → allow duplicates, insert in the end
3. DELETE val → delete all occurences
4. FIND val → true if present, false otherwise
5. PRINT

ARRAY BASED IMPLEMENTATION OF LIST

MAINTAIN AN ARRAY INTERNALLY — HOW BIG?

assume some max size.

free cells in array

INSERT — maintain index at pos to be inserted

O(1)

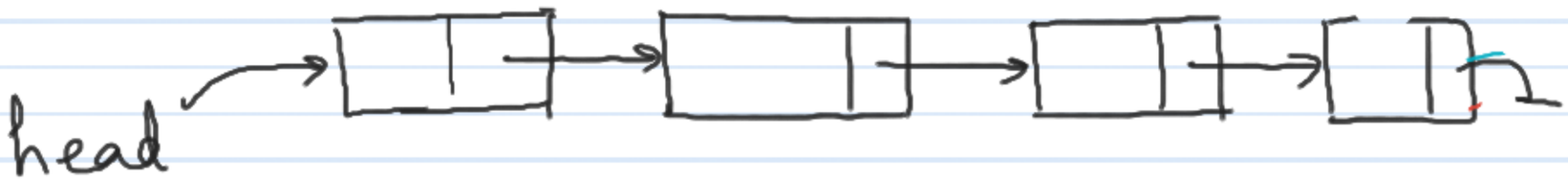
DELETE — when we see element 'e' move all elements to the right of e by 1 post

k occurrences of element e

each delete takes O(n), TOTAL TIME $O(n^2)$

improve this to $\Theta(n)$

LINKED LIST BASED IMPLEMENTATION.



INSERT O(n) since insert happens at end

DELETE O(n) delete needs to go over all the list

FIND O(n)

SIZE O(n) but can be improved to O(1)

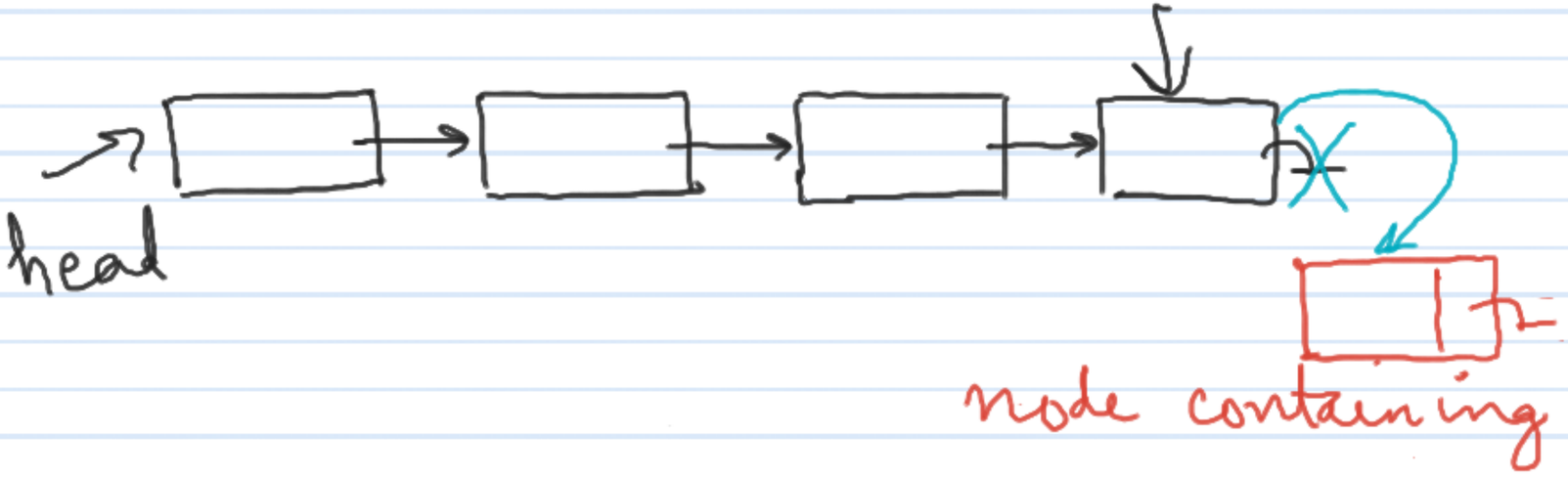
INSERT IN A LINKED LIST

TWO CASES:

(1) INSERT FIRST ELEMENT

head needs to change

(2) INSERT non FIRST ELEMENT

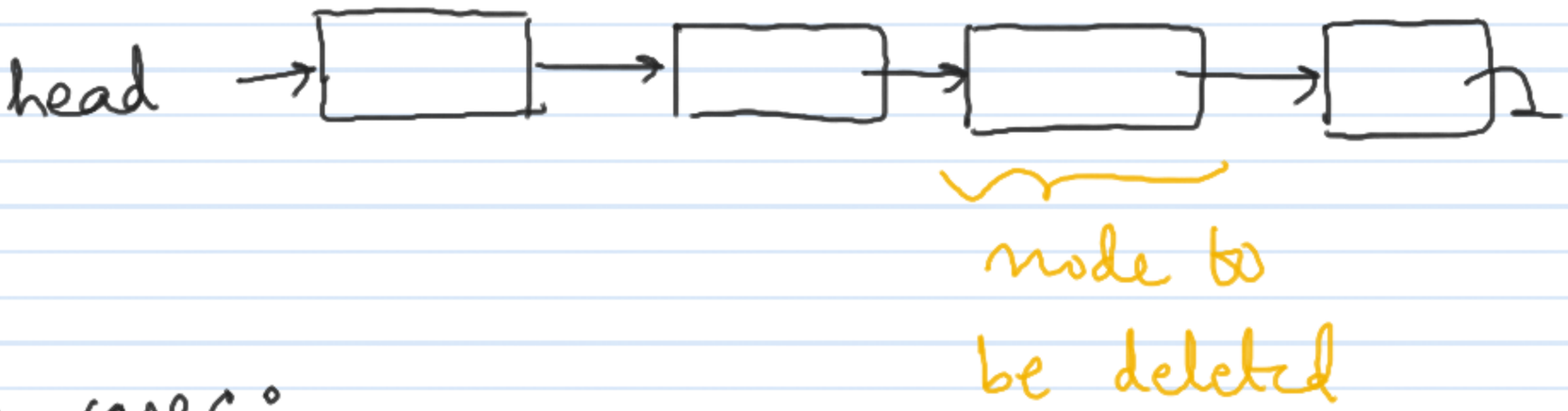


In Both cases:

- we need to get storage on heap (new / malloc)
- set nodes val and nextptr appropriately

These are O(1) operations

DELETE FROM A LINKED LIST.



Two cases:

(1) head needs to be deleted

(2) some internal node needs to be deleted

Maintain a prev pointer, prev == NULL indicates first node needs to be deleted.

Recursive print list

```
void printR ( Node * ptr)
    if (ptr == NULL)
        return;
    else {
        { ptr.print()
          printR(ptr → next);
        }
    }
```

simply prints the node.

} What happens if we switch the order of the 2 statements?

Notes:

(1) If head is "private" then a starter function is needed. (see code)

(2) The running time of the function is $T(n) = T(n-1) + O(1) \approx O(n)$.