Linked-lists

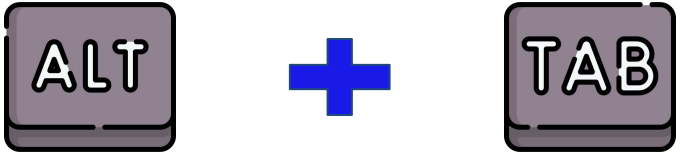
By: Yehia M. Abu Eita

Outlines

- Introduction
- Types of linked-lists
- Insert, delete and print
- Linked-lists applications
- How to implement a linked-list

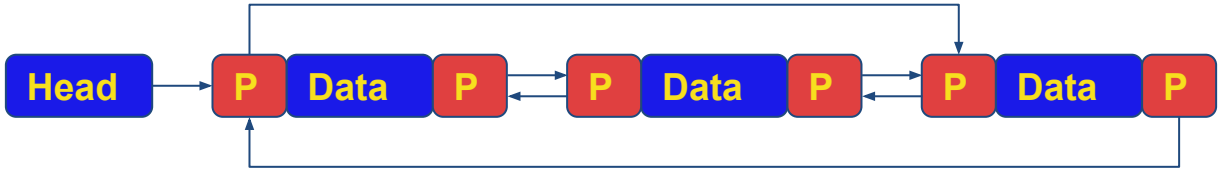
Introduction

- Linked-list is a linear data structure that **connect** data nodes together **without** a need of **contiguous** storing into memory.
- A real-life example of a linked-list is **Alt+Tab** in windows.
- **Any linked-list is characterized by its head and tail.**



Types of linked-lists

- Singly Linked-list
- Circular Linked-list
- Doubly Linked-list
- Circular Doubly Linked-list

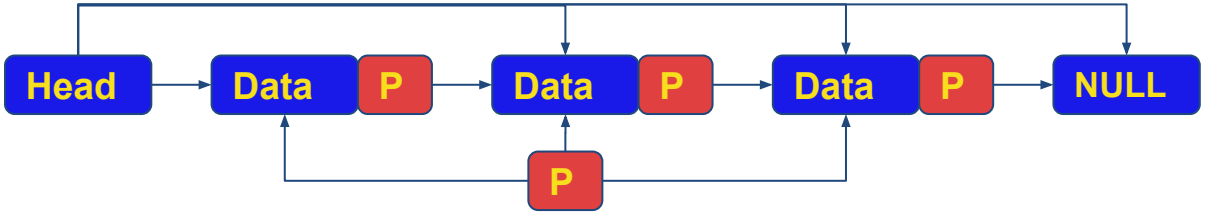


Insert, delete and print

- **Inserting** may be into **head**, **tail** or **any position**.
- **Deleting** may be from **head**, **tail** or any **position**.
- A **linear linked-list node** is represented by a **structure** with at **two members**, **data** and **pointer to the next node**.
- An linked-list is represented by a structure with **two members, head pointer and size**. It is **empty** when its **head** points to **NULL**.
- Linked-lists take place into **heap**.

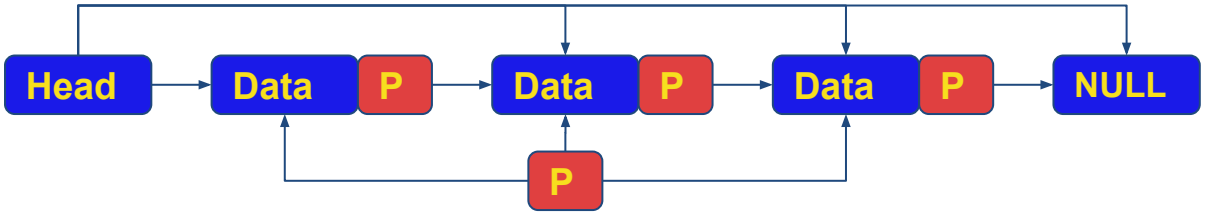
Insert, delete and print

- **Steps to insert to head**:
  - **Allocate list node into heap.**
  - **Store data into node.**
  - **Make the node pointer points to what the head is pointing to.**
  - **Make the head pointer points to the created node.**



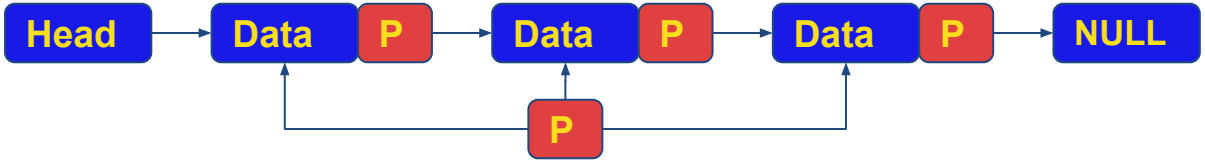
Insert, delete and print

- **Steps to delete from head**:
  - Check if the list is empty.
  - If is empty, print the list is empty error.
  - If is not empty, store the address of the head node to a temp pointer.
  - Make the head pointer points to what the head node next pointer points to.
  - Free the allocated memory pointed by the temp pointer.



Insert, delete and print

- **Steps to print list**:
  - Check if the list is empty.
  - If is empty, print list is empty error.
  - If is not empty, create a temp pointer that points at the head node.
  - Print node data.
  - Change the temp pointer to point to the next node.
  - If is it the last node/tail, print node data and stop.



Linked-lists applications

- It is used to implement stacks and queues.
- It is used in notepad to perform undo or redo or deleting functions.
- We can think of its use in a photo viewer for having look at photos continuously in a slide show.
- Used in switching between applications and programs (Alt + Tab) in the Operating system.

How to implement a linked-list

- Declare a **global or local variable** that defines the **linked-list head**, create a structure that defines a list node with **two members**, data, and pointer to next node.
- Implement **Insert** and **delete** functions as the **main functions** of the list.
- Implement **isEmpty**, **isFull**, **isTail**, and **printList** functions as a **helper** and **utility** functions.
- You can also implement some advanced functions like, **listSearch**, **listSort** and **listReverse**.

How to implement a linked-list

- Use the following **prototypes** as a guide to implement a linked-list:
  - `typedef struct node{int data; struct node *next;}ST_listNode_t; // Node Type`
  - `typedef struct list{ST_listNode_t *head; int listSize;}ST_list_t; // List Type`
  - `void createEmptyList(ST_list_t *list); // Setting list's head to NULL and size to 0`
  - `int insertToList(ST_list_t *list, int position, int data);`
  - `int deleteFromList(ST_list_t *list, int position);`
  - `int printList(ST_list_t *list);`
  - `int getNodeData(ST_list_t *list, int nodeNumber, int *nodeData);`
  - `int isTail(ST_listNode_t node);`
  - `int isEmpty(ST_list_t *list);`

Summary

- Now you familiar with the linked-list data structure.
- Remember that linked-lists data are allocated into heap.
- Remember that before deleting a node take a copy of its address in order to free correctly.
- Remember that linked-lists can implement stack and queue.