Lecture 12

- Linked Lists

IT205: Data Structures (AY 2023/24 Sem II Sec B) — **Dr. Arpit Rana**

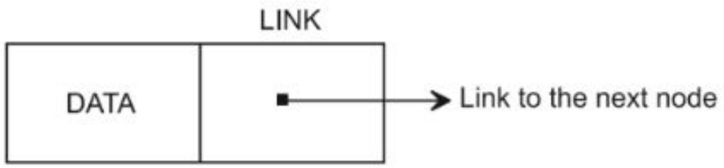
Motivation for Linked Lists

- Array elements are stored in contiguous memory locations –
  - A block of memory required for the array should be allocated before hand
  - Neither can be extended or shrinked – may lead to either wastage or shortage of memory
- Array is known as static data structure.

Linked Lists

A linked list is an ordered collection of finite, homogeneous data elements called nodes where the linear order is maintained by means of links or pointers.

- A node of a linked list consists of two fields –
  - Data: to store the actual information
  - Link: to store a pointer to the next node



Linked Lists

A linked list is an ordered collection of finite, homogeneous data elements called nodes where the linear order is maintained by means of links or pointers.

- Depending on the requirements the pointers are maintained, and accordingly the linked list can be classified into three major groups:
  - Single Linked List
  - Circular Linked List
  - Doubly Linked List

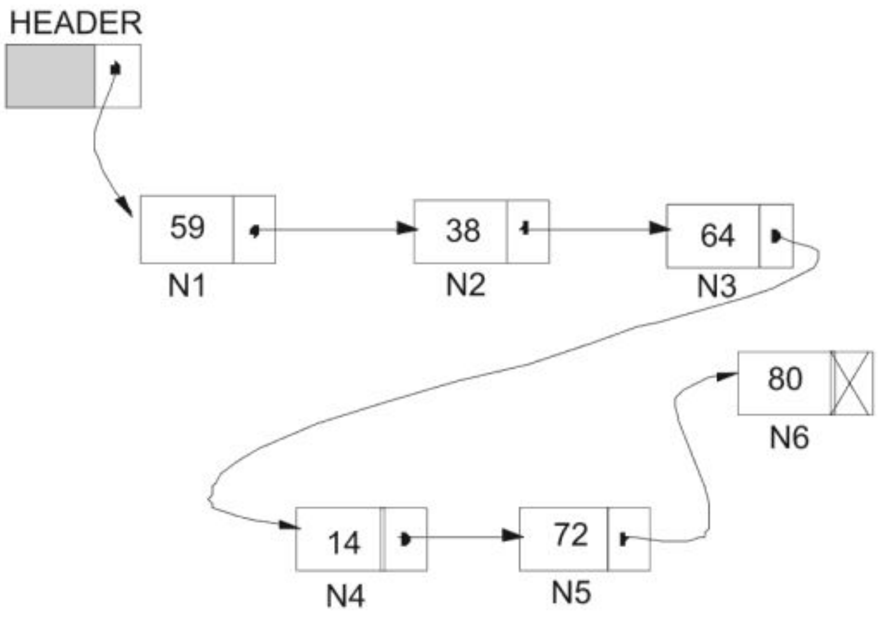
Single Linked List

In a Single Linked List (a.k.a., One-way List), each node contains only one link which points to the subsequent node in the list.

- HEADER is an empty node (data content NULL, or sometimes size of the list) and only used to store a pointer to the first node N1.

- All other nodes: N1, N2, … N6 are constituent nodes in the list.

- The last node of the list contains NULL in its link field.



Representation of Linked List in Memory

There are two ways to represent a linked list in memory –

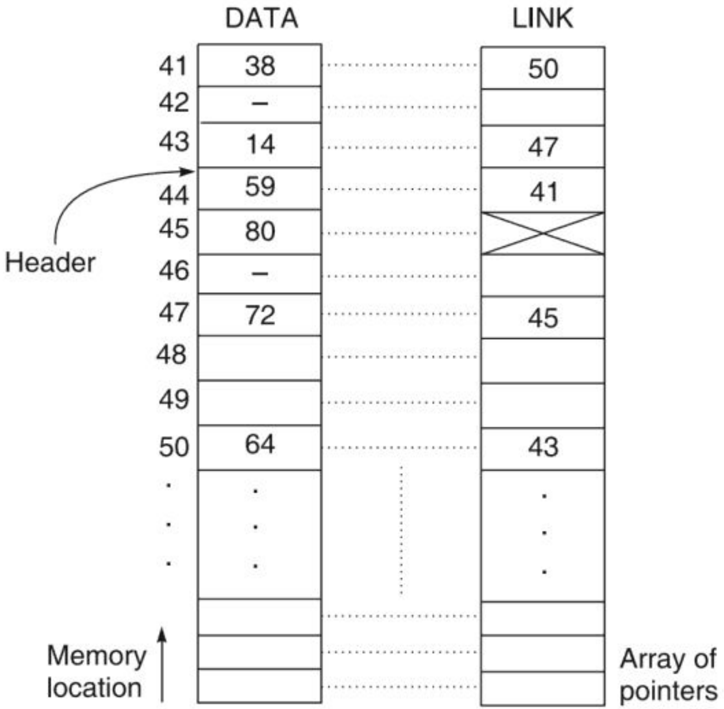
- Static representation using array
- Dynamic representation using free pool of storage

Linked List: Static Representation

In static representation of a single linked list, two arrays of equal size are maintained:

- one array for **data**, and
- the other for **links**

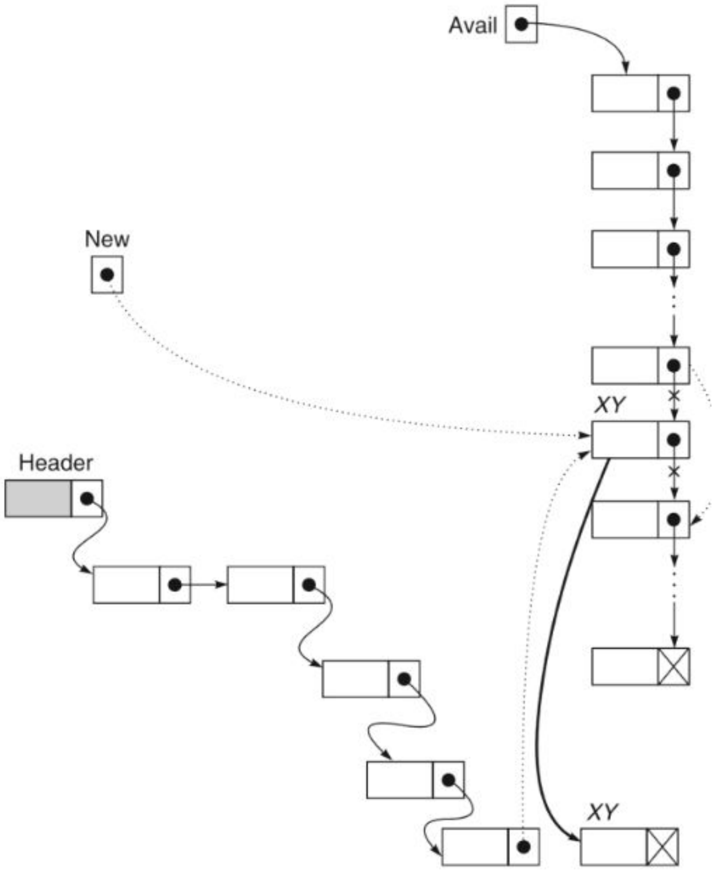
This contradicts the idea of non-contiguous memory allocation.



Linked List: Dynamic Representation

Dynamic representation of Linked List employs dynamic memory management policy.

- **Memory bank:** a collection of free memory spaces
- **Memory manager:** a program that searches for free memory block on request

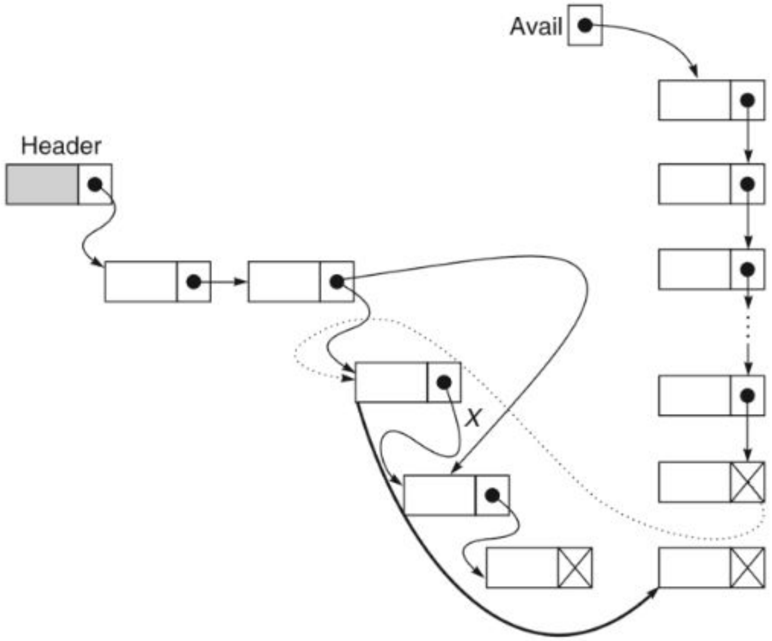


Linked List: Dynamic Representation

Dynamic representation of Linked List employs dynamic memory management policy.

- **Garbage collector**: a program that returns the unused node to the memory bank

Memory allocation and deallocation are carried out by adjusting the pointers only.



Operations on a Single Linked List

for cancatinate linklist5.c

The operations possible on single linked list are as follows.

concatinating and merging are different for better understanding wach video 202 and 203

- **Traversing** the list
- **Inserting** a node into the list
- **Deleting** a node from the list
- **Copying** the list to make a duplicate of it
- **Merging** the linked list with another one to make a larger list   ex3->ex3_4.cpp
- **Searching** for an element in the list

for reversing we take three method's reverse data or links
1.reverse data:we take one array the do thing.you can watch video.video no is 198
2.reverse links: we use sliding pointer method . in this method we take 3 pointer .video no is 199
in first method we need extra array so it space complexity is increased and in second method we take 3 pointer
but most preferable method is second.for large data second method is good.
3.third method is recursion   video no 200

Traversing a Single Linked List

Visiting every node of the list starting from the first node to the last node.

Input: HEADER is the pointer to the header node of the list.
Output: According to the Process() that we apply onto each node.

Steps:

```
p = Link(HEADER)             // p refers to the current node
While(p != NULL) do          // continue till the last node
    Process(p)               // perform Process() on the current node
    p = Link(p)              // move to the next node
EndWhile
Stop
```

in codeslide part file name linklist1.c

O(n) time complexity in loop or recustiona and O(n) space complexity in recursion

Inserting a Node in a Single Linked List

There are various positions where a node can be inserted:

- Inserting at the front (as a first element)
- Inserting at the end (as a last element)
- Inserting at any other position

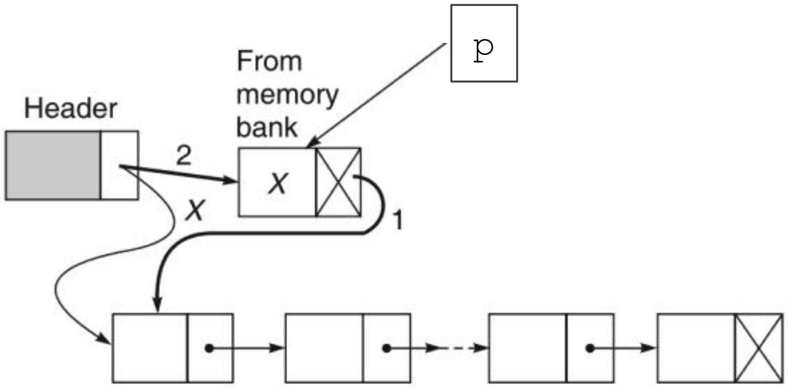
linklist2.c

Inserting a Node in a Single Linked List

Input: HEADER – the header node of the list and X – the data of the node to be inserted
Output: A single linked list with a newly inserted node **at the front of the list**

Steps:

```
p = getnode()
If(p == NULL)then
    print("Memory Underflow")
    exit()
Else
    Data(p) = X
    Link(p) = Link(HEADER)
    Link(HEADER) = p
EndIf
Stop
```



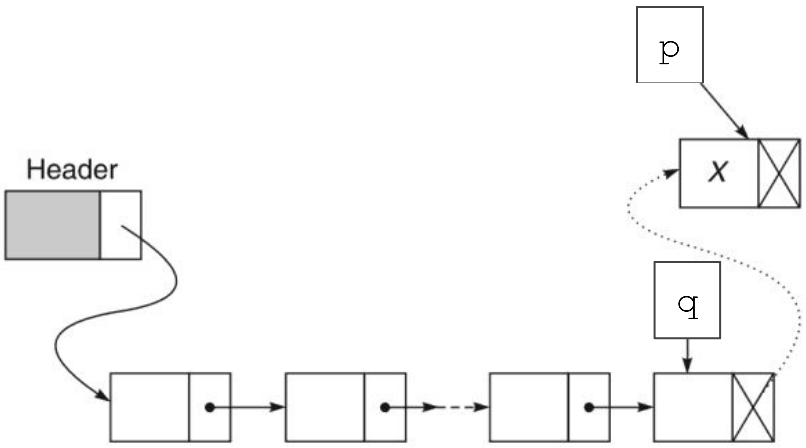
O(1)

Inserting a Node in a Single Linked List

Input: HEADER – the header node of the list and X – the data of the node to be inserted
Output: A single linked list with a newly inserted node **at the end of the list**

Steps:

```
p = getnode()
If(p == NULL)then
    print("Memory Underflow")
    exit()
Else
    q = HEADER
    While(Link(q) != NULL)do
        q = Link(q)
    EndWhile
    Data(p) = X
    Link(q) = p
EndIf
Stop
```



O(n)

if we take pointer as last it pointing on last node then when we want to add new node at last then it should be easy because we should do only last->next=t(new node); so it take constant time

Inserting a Node in a Single Linked List

Input: HEADER – the header node of the list, X – the data of the node to be inserted, and KEY – the data of the key node after which the node has to be inserted
Output: A single linked list with a newly inserted node **after the node with data KEY.**

Steps:

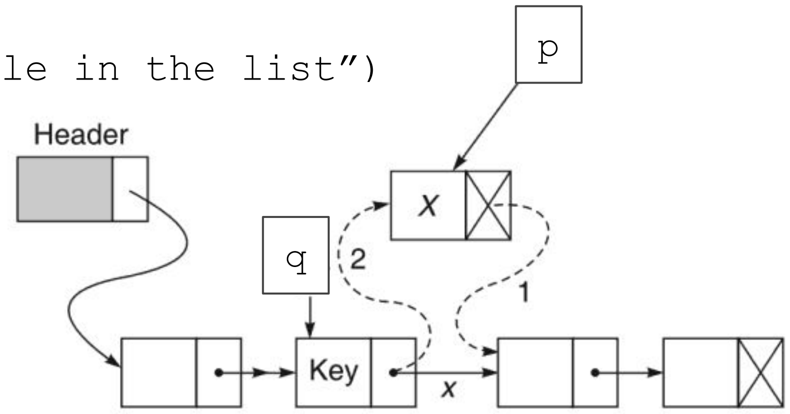
```
p = getnode()
If(p == NULL)then
    print("Memory Underflow")
    exit()
Else
    q = HEADER
    While(Data(q)!= KEY and Link(q) != NULL)do
        q = Link(q)
    EndWhile                              O(n)
    contd...
```

Inserting a Node in a Single Linked List

Input: HEADER – the header node of the list, X – the data of the node to be inserted, and KEY – the data of the key node after which the node has to be inserted
Output: A single linked list with a newly inserted node **after the node with data KEY.**

Steps:

```
        contd...
        If(Link(q) == NULL) then
            print("KEY is not available in the list")
            exit()
        Else
            Data(p) = X
            Link(p) = Link(q)
            Link(q) = p
        EndIf
    EndIf
    Stop
```



Deleting a Node from a Single Linked List

There are various positions from where a node can be deleted:

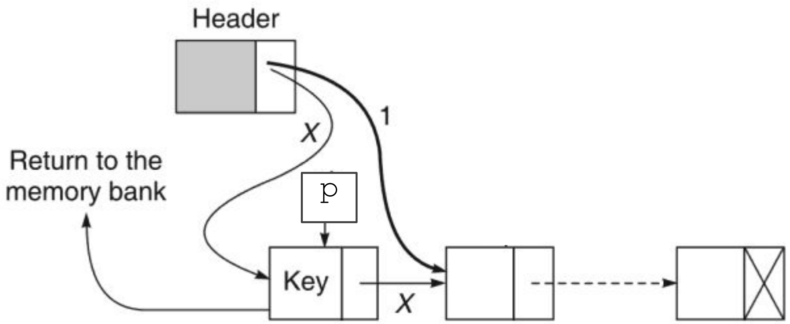
- Deleting from the front of the list (the first element)
- Deleting from the end of the list (the last element)
- Deleting from any other position in the list

Deleting a Node from a Single Linked List

Input: HEADER – the header node of the list
Output: A single linked list after deleting a node **at the front of the list**

Steps:

```
p = Link(HEADER)
If(p == NULL)then
    print("The List is Empty")
    exit()
Else
    Link(HEADER) = Link(p)
    freenode(p)
EndIf
Stop
```



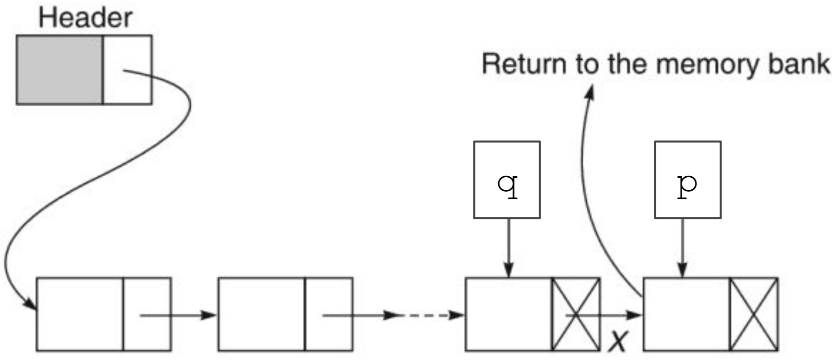
O(1)

Deleting a Node from a Single Linked List

Input: HEADER – the header node of the list
Output: A single linked list after deleting a node **at the end of the list**

Steps:

```
p = HEADER
If(Link(p) == NULL)then
    print("The List is Empty")
    exit()
Else
    q = p; p = Link(p)
    While(p != NULL)do
        q = p
        p = Link(p)
    EndWhile
    Link(q) = NULL
    freenode(p)
EndIf
Stop
```



O(n)

Deleting a Node from a Single Linked List

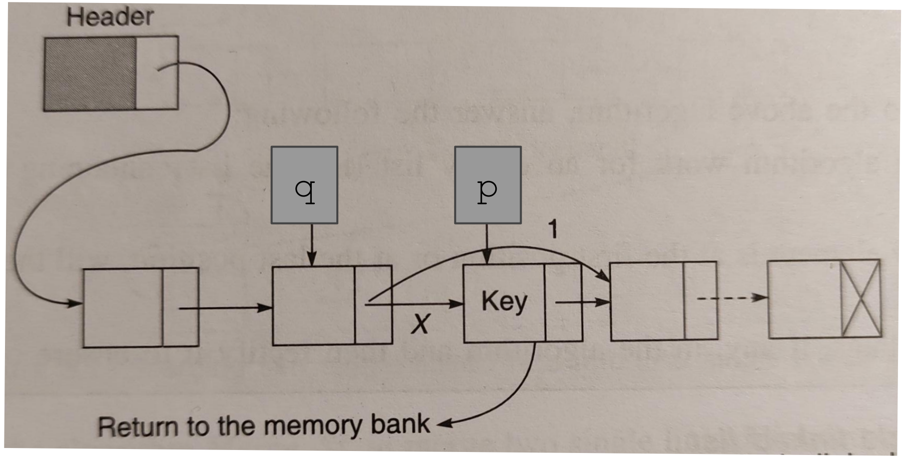
Input: HEADER – the header node of the list and KEY – the data of the key node after which the node has to be inserted
Output: A single linked list after deleting a node **with data KEY.**

Steps:

O(n)

```
q = HEADER; p = Link(HEADER)
While(p != NULL)do
    If(Data(p) != KEY) then
        q = p
        p = Link(p)
    Else
        Link(q) = Link(p)
        freenode(p)
        exit()
    EndIf
EndWhile
```

contd...



Deleting a Node from a Single Linked List

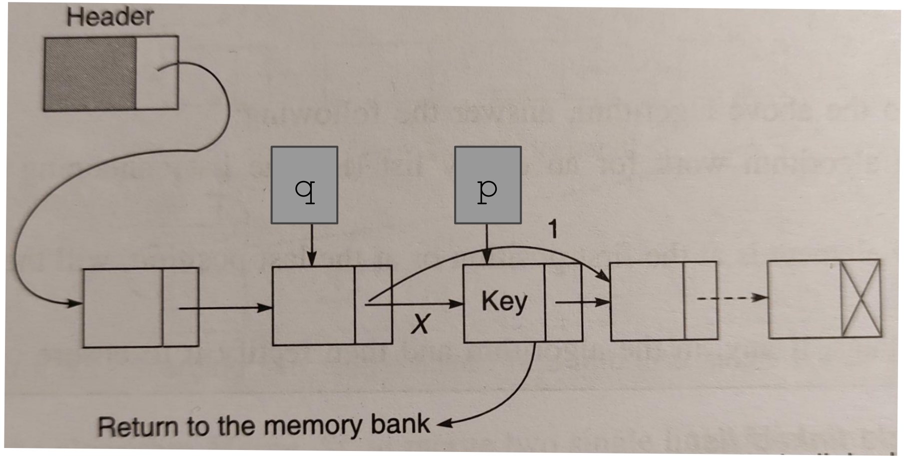
Input: HEADER – the header node of the list, X – the data of the node to be inserted, and KEY – the data of the key node after which the node has to be inserted
Output: A single linked list with a newly inserted node **after the node with data KEY.**

Steps:

contd…

```
If(p == NULL) then
    print("Node with KEY does not exist")
EndIf
Stop
```

linklist2.cpp



Next Lecture

- Linked Lists Contd...