Lecture 13

- Linked Lists - II

`getnode()` and `freenode()` Operations

```
p = getnode()
```

Deleting the front node from the AVAIL list if not empty

Steps:

```
If(AVAIL == NULL)then
    print("Memory Overflow")
    exit()
Else
    p = AVAIL
    AVAIL = Link(AVAIL)
```

```
freenode(p)
```

Inserting unused node at the front of the AVAIL list

Steps:

```
Link(p) = AVAIL
AVAIL = p
```

Copying a Single Linked List

We can create a copy of a Single Linked List by copying content of each node into the newly allocated node.

Input: HEADER is the pointer to the header node of the master list.
Output: HEAD is the pointer to the duplicate list.

Steps:

Do It Yourself!

for diagram refer notebook

this not full code

```
p=link(header1);
q=link(header2);
while(p1=NULL) do
data(q)=data(p);
q=link(q);
p=link(p);
end while
q=NULL;
```

Merging Two Single Linked Lists Into One

We can merge two lists into one – see the diagram.

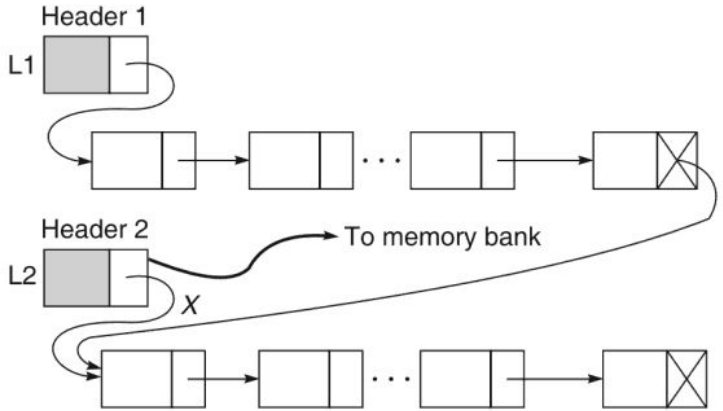
Input: HEADER1 and HEADER2 are the pointers to the header nodes of the two lists to be merged.
Output: HEADER is the pointer to the merged list.

This is called concatenating this not merge .

Steps:

Do It Yourself!

```
p=link(header);
while (link(p)!=NULL) do
p=link(p);
end while;
q=link(header2);
link(p)=q;
```



Search for an Element in a Single Linked List

We can search for an element into the list – we did a similar task earlier while inserting after or deleting a node with the DATA as KEY.

Input: KEY is the item to be searched.
Output: LOCATION, the pointer to a node where the KEY belongs to or an error message.

Steps:

Do It Yourself!

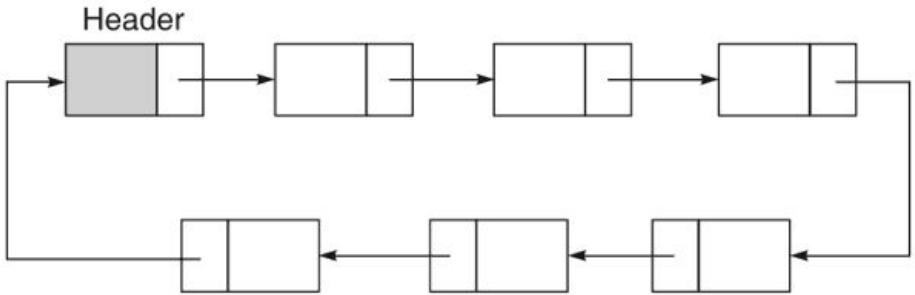
```
p=link(header);
index=0;
while(p!=NULL  and  data(p)!=key) do
p=p->link;
index++;
end while;
if (p==NULL) do
"element does not present in this link list";
else do
"element is present in this list at %index"
```

Circular Linked List

If the last node of a singly linked list points to the header node instead of being NULL, we refer to this list as a Circular Linked List.

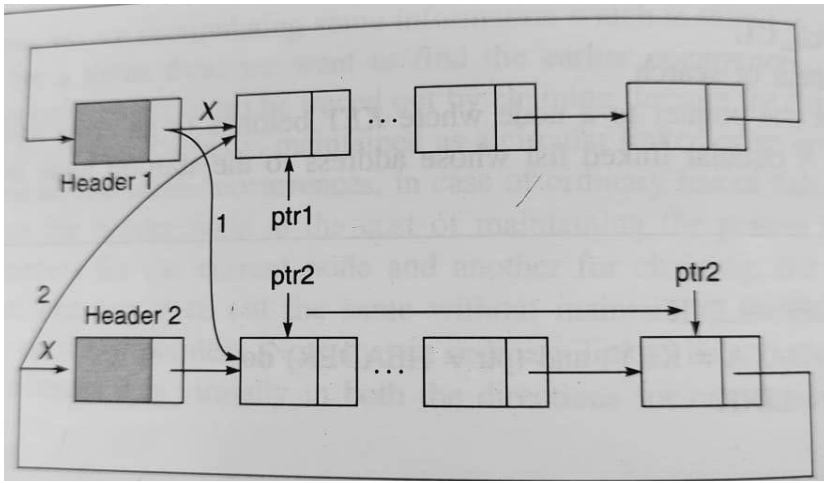
In circular linked list,

- every member node is accessible from any node
- end of the list is the HEADER node only.



Circular Linked List

Some operations can more easily be implemented with a circular linked list than the single linked list, such as, merging, splitting, etc.



Circular Linked List

Circular list has advantages over Singly lists, still has a few drawbacks –

- One cannot traverse such a list backwards
- If only a pointer to a node is given, it cannot be deleted without traversing to that node.
  if ptr2 is not given that

Solution is **Doubly Linked Lists**.

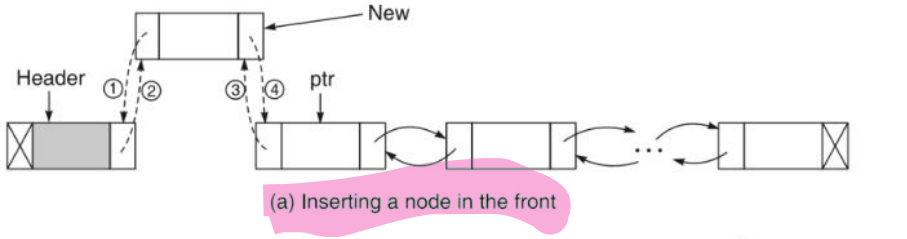
for better understanding of circular linked list prefer note or video

Doubly Linked List

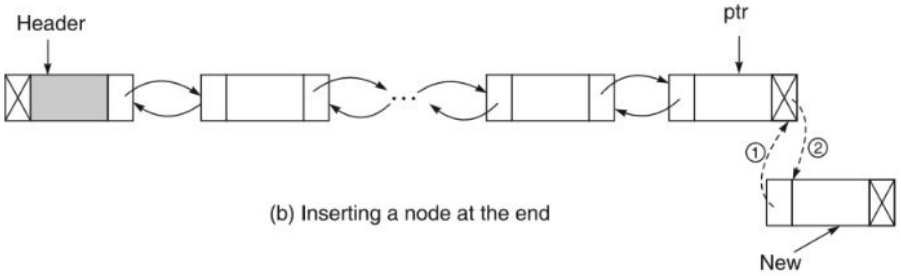
In Doubly Linked List -

- Each node contains two pointers: **Left** and **Right**.
- The former points to a **node prior to the current node** and the latter points to the **one after it**.
- Doubly linked list can be linear, circular, with or without the header node.

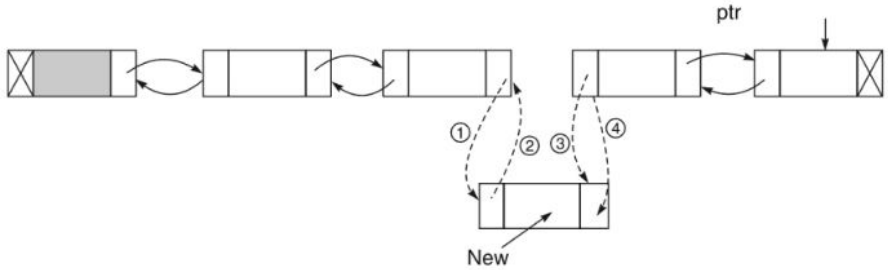
Inserting a Node into a Doubly Linked List



(a) Inserting a node in the front



(b) Inserting a node at the end



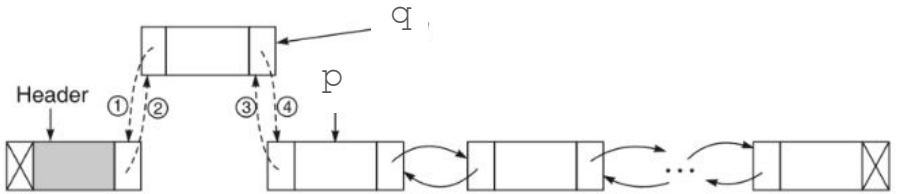
(c) Inserting a node at any intermediate position

Inserting a Node into a Doubly Linked List

Input: HEADER – the header node of the list and X – the data of the node to be inserted
Output: A linear doubly linked list after inserting a node **with data X.**

Steps to Insert at the Front:

```
p = Right(HEADER)
q = getnode()
If(q != NULL)then
    Left(q) = HEADER      // 1
    Right(HEADER) = q     // 2
    Right(q) = p          // 3
    Left(p) = q           // 4
    Data(q) = X
Else
    print("Memory Underflow!")
EndIf
Stop
```



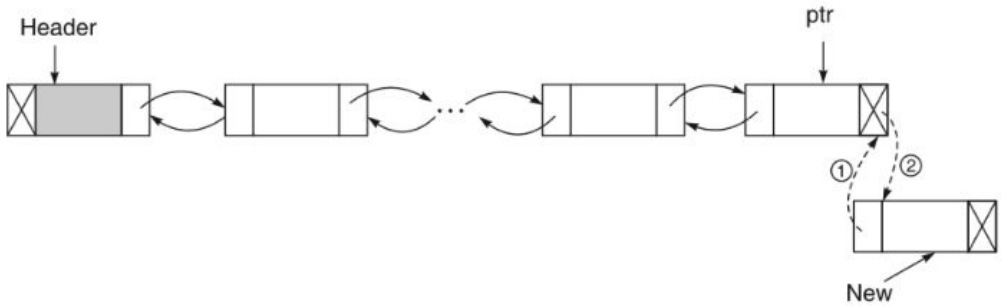
Inserting a Node into a Doubly Linked List

Input: HEADER – the header node of the list and X – the data of the node to be inserted
Output: A linear doubly linked list after inserting a node **with data X.**

Steps to Insert at the End:

```
Do it Yourself!
```

dlinklist1.cpp

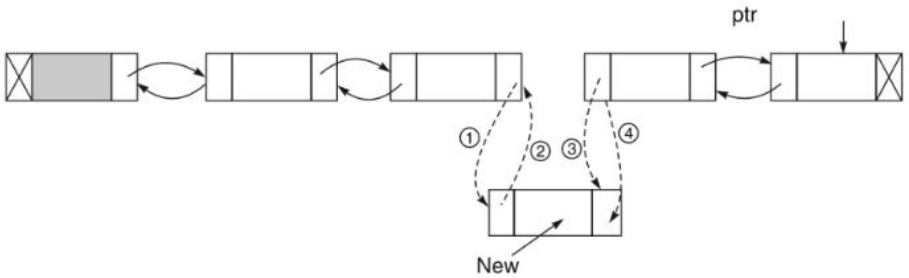


Inserting a Node into a Doubly Linked List

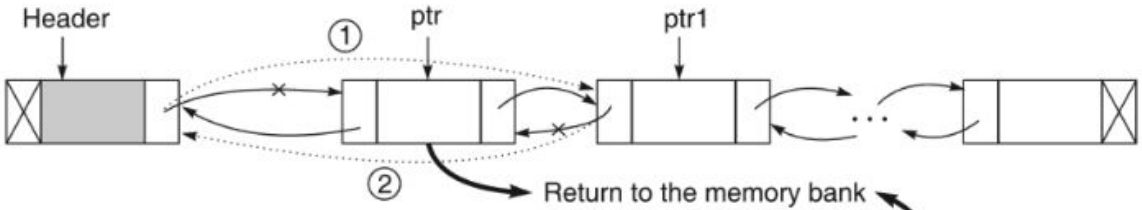
Input: HEADER – the header node of the list; X – the data of the node to be inserted; KEY – the data content of the node after which new node is to be inserted.
Output: A linear doubly linked list after inserting a node **with data X.**

Steps to Insert at any Position:

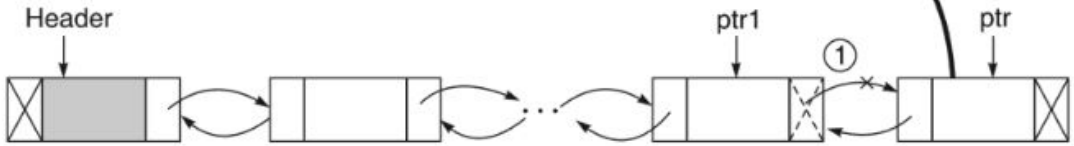
```
Do it Yourself!
```



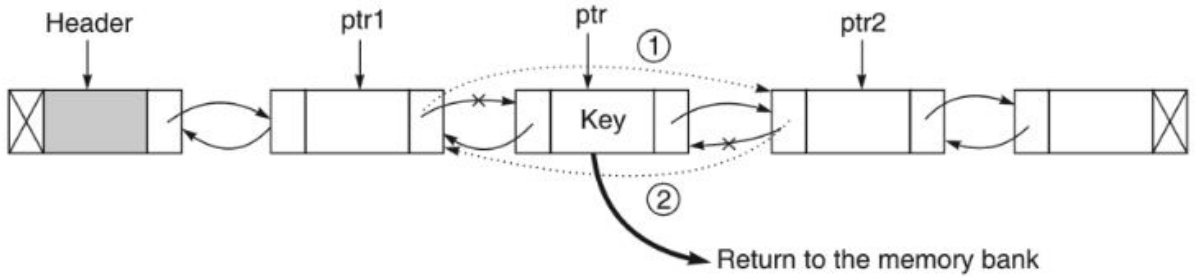
Deleting a Node from a Doubly Linked List



(a) Deleting the node placed at the front



(b) Deleting the node placed at the end



(c) Deleting a node from any intermediate position

Deleting a Node from a Doubly Linked List

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

Steps to delete a node at the front:

```
p = Right(HEADER)
If(p == NULL)then
    print("The List is Empty")
    exit()
Else
    q = Right(p)
    Right(HEADER) = q          //1
    If(q != NULL)then
        Left(q) = HEADER       //2
    EndIf
    freenode(p)
EndIf
Stop
```

Circular Doubly Linked List

In circular doubly linked list, the last node of a linear doubly linked list connected to the right of the header.

- All the basic operations can be performed similar to the linear doubly linked list.

Exercise

- Write an algorithm to sort a circular doubly linked list. exercise part->d_c_linklist.cpp
- How to use Linked lists to represent Sparse Matrices more efficiently? exercise part->sparse.cpp
- How to use Linked lists to represent polynomials? poly.cpp

FOR ALL CODES OF SINGLE , DOUBLY AND CIRCULAR PREFER lab-05 CODES.

2.In Linked list we store only non-zero values so in this we don't required to much memory.

Next Lecture

- Stacks