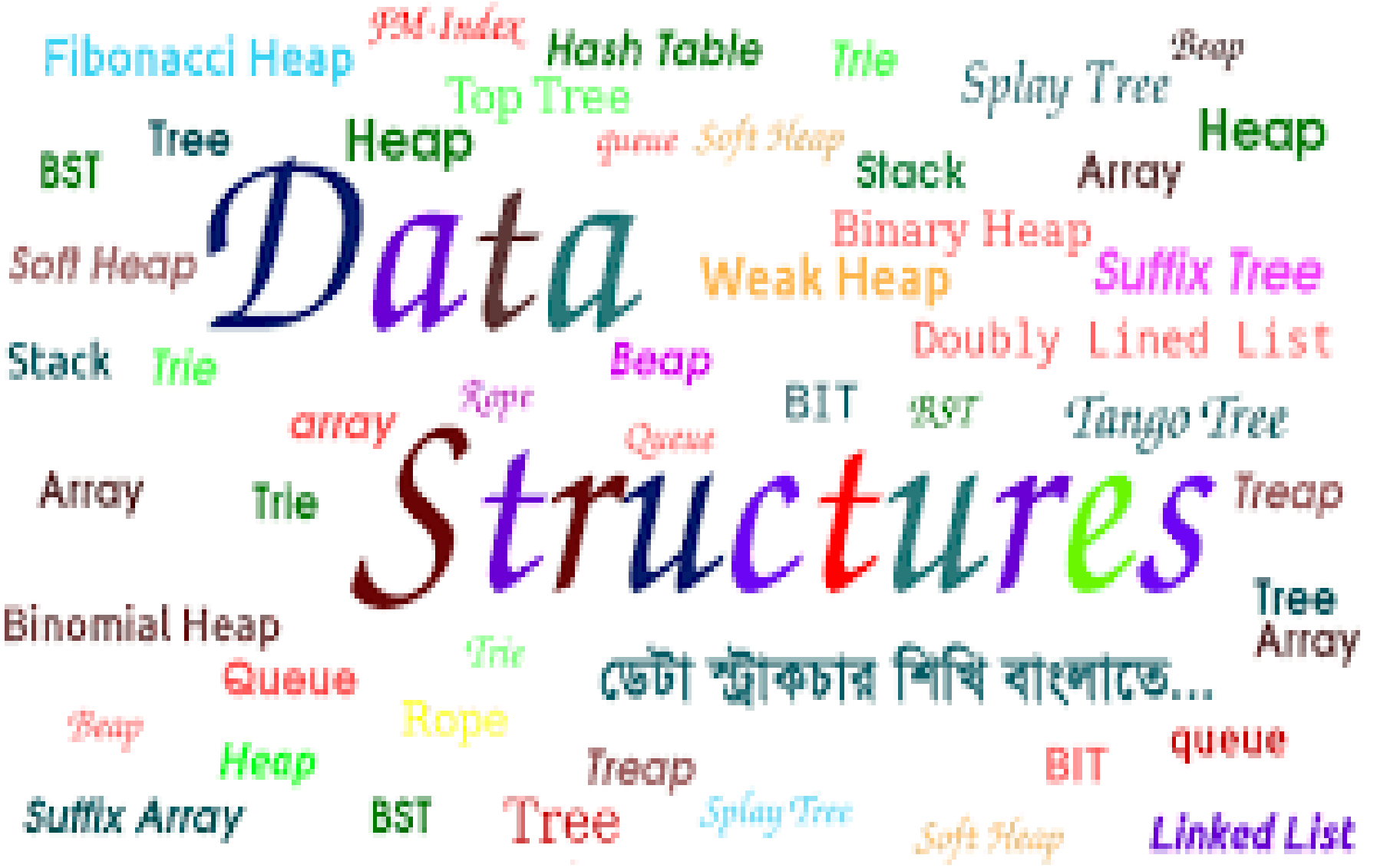
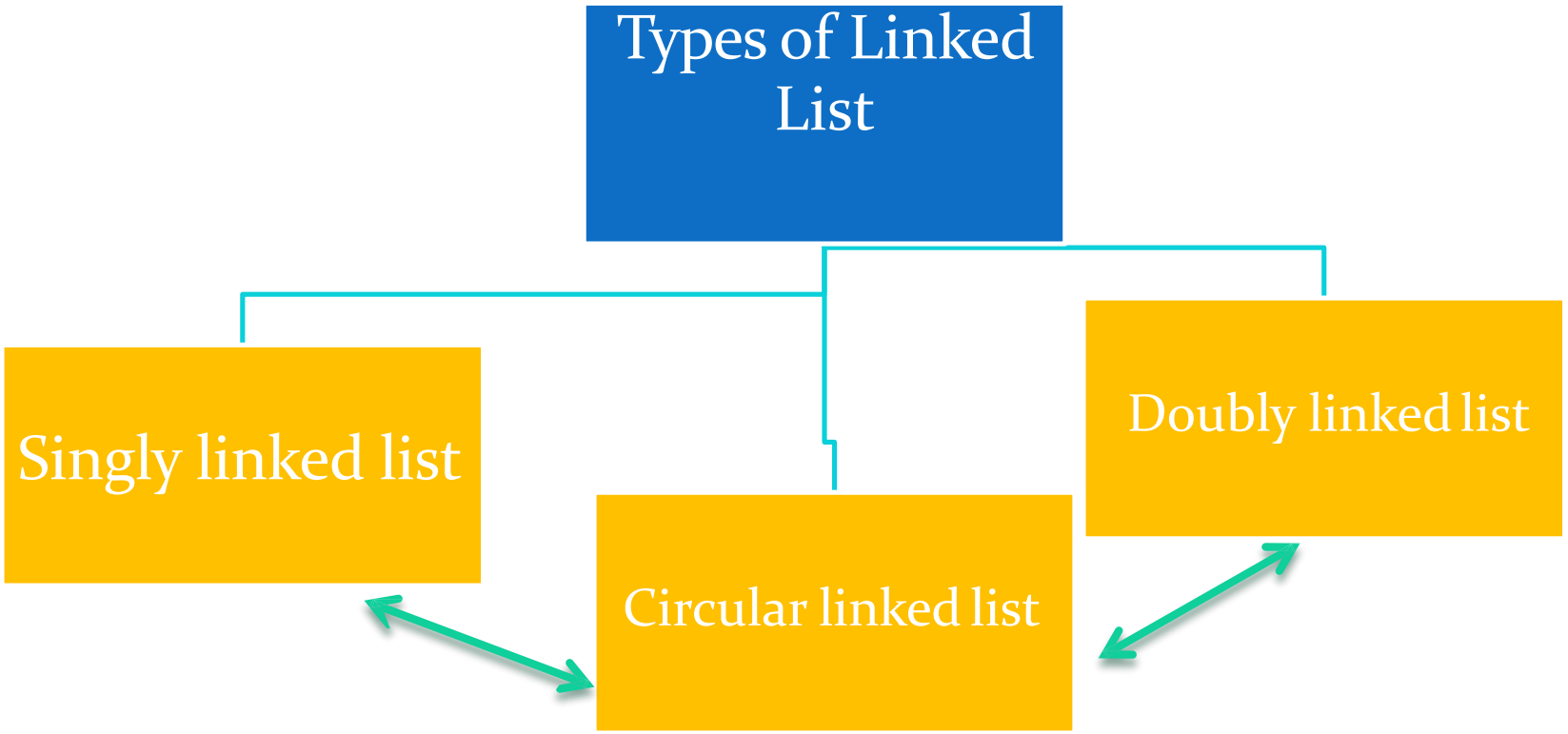

Data
Structures
Fibonacci Heap
FM-Index
Hash Table
Trie
Top Tree
Splay Tree
Beap
Tree
Heap
BST
queue
Soft Heap
Stack
Array
Binary Heap
Soft Heap
Weak Heap
Suffix Tree
Doubly Linked List
Stack
Trie
Beap
Rope
BIT
BST
Tango Tree
array
Queue
Array
Trie
Treap
Binomial Heap
Tree
Array
Trie
ডেটা স্ট্রাকচার শিখি বাংলাতে...
Queue
Beap
Rope
Heap
Treap
BIT
queue
Suffix Array
BST
Tree
Splay Tree
Soft Heap
Linked List

TYPES OF LINKED LIST



Doubly linked list

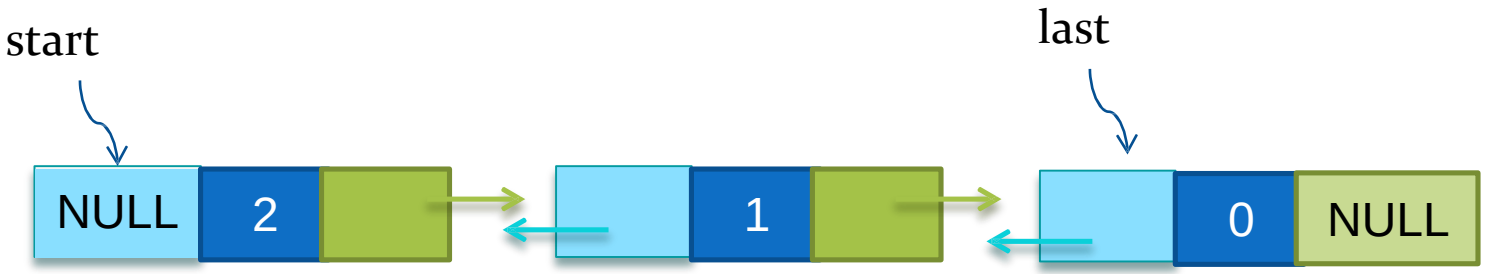
Doubly linked list

- A doubly linked list is one in which all nodes are linked together by multiple links which help in accessing both the successor (next) and predecessor (previous) node for any arbitrary node within the list

- Every nodes in the doubly linked list has three fields: LeftPointer, RightPointer and DATA
- LeftPointer will point to the node in the left side (or previous node) that is LeftPointer will hold the address of the previous node

- RightPointer will point to the node in the right side (or next node) that is RightPointer will hold the address of the next node

Doubly linked list



0x80019

0x80019 0x80021 0x80030

Applications of Doubly linked list

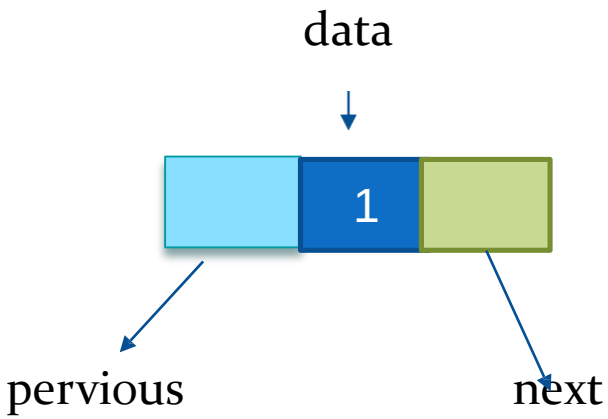
Applications of Doubly linked list can be

- A great way to represent a deck of cards in a game.
- The browser cache which allows you to hit the BACK button (a linked list of URLs)
- Applications that have a Most Recently Used (MRU) list (a linked list of file names)
- Undo functionality in Photoshop or Word (a linked listof state)
- Astack, hash table, and binary tree can be implemented using a doubly linked list.

Representation of doubly linked list

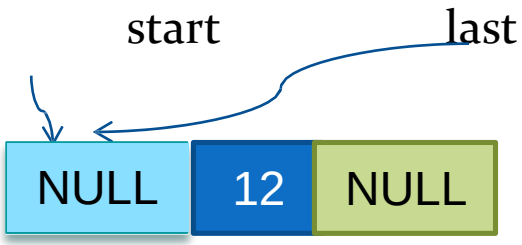
- A node can represent in memoryas

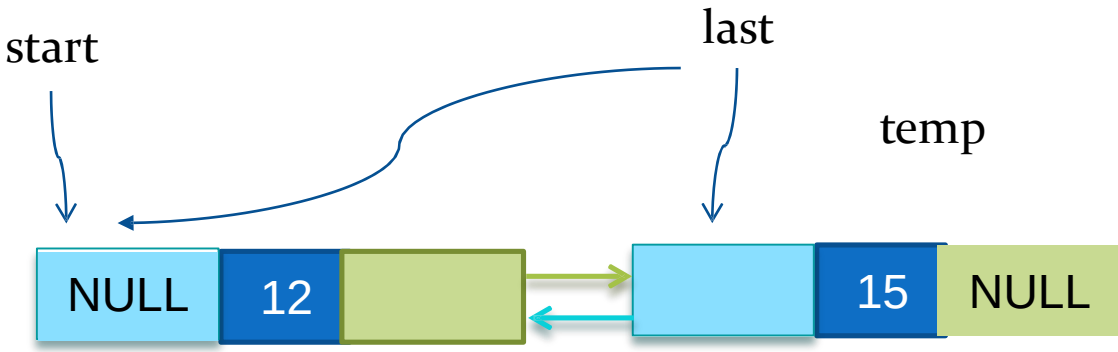
```
struct
{
        int data;
        node * pervious;
        node* next;
};
```



Linked List:add nodes forward

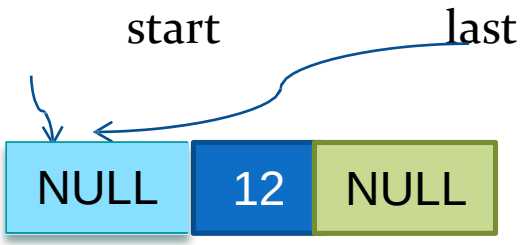
```
void linked_list:: add_node(int value)
    {
        node *temp=new node;
        temp->data=value;
        temp->next=NULL;
        temp->pervious =NULL;
        if (start == NULL)
        {
                start=temp;
                last=temp;
        }
        else
        {
          last->next = temp;

        temp-> pervious=last;
         last = temp;

        }
}
```

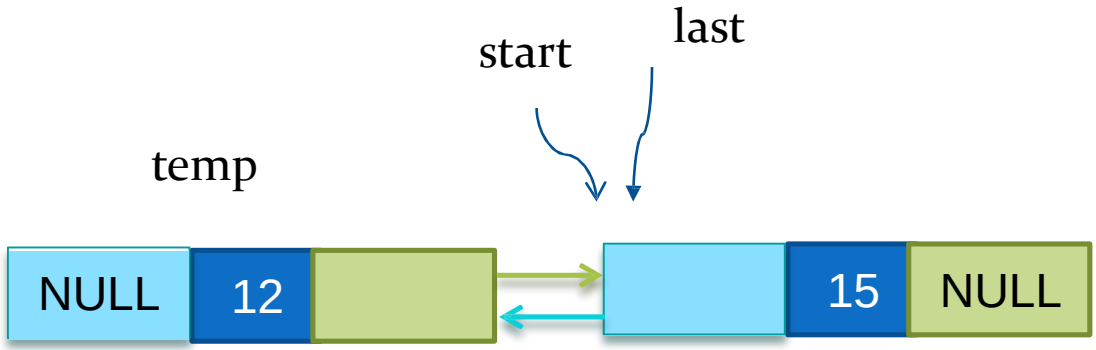




Linked List:add nodes forward

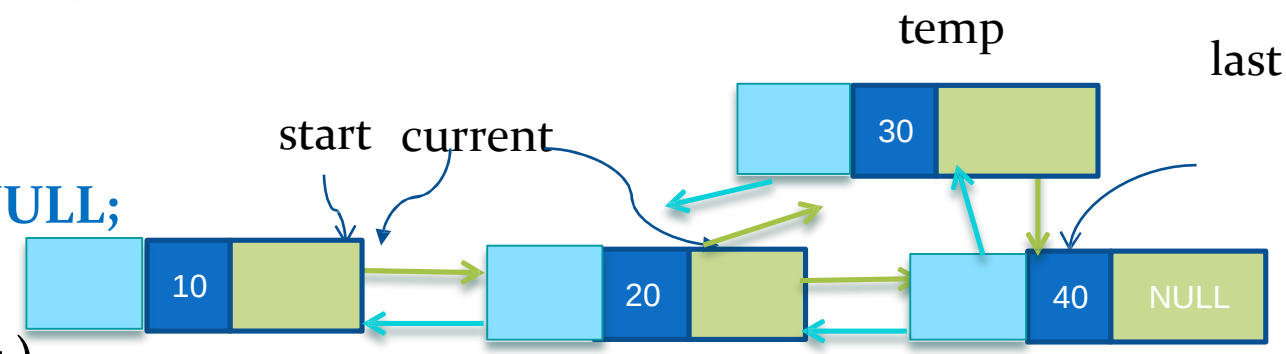
```
void linked_list:: add_first(int value)
    {
        node *temp=new node;
        temp->data=value;
        temp->next=NULL;
        temp->pervious=NULL;
        if (start == NULL)
        {
                start=temp;
               last=temp;
        }
    else{
        temp->next=start;
        start->pervious=temp;
        start=temp; }
    }
```





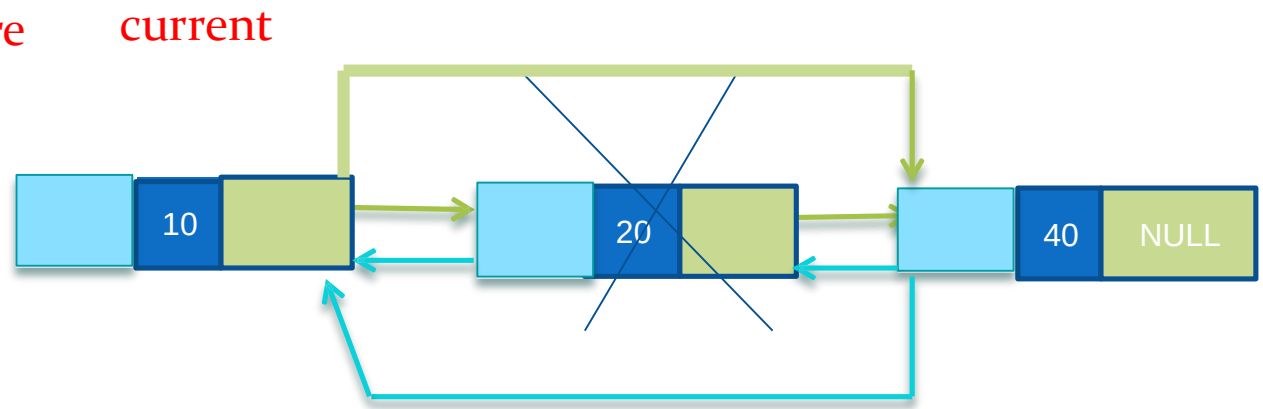
Linked List; insert_position

```
void linked_list::insert_at_position(int pos, int value)
    {node *temp=new node;
    temp->data=value;
    temp->next=NULL;
    temp->pervious=NULL;
node *current=start;
    for(int i=1;i<pos-1;i++)
    {current=current->next;
          if(current==NULL)
     {cout<<" the list has element less than
          "<<pos<< "element ";return;}
    }
    temp->next=current->next;
    temp->pervious=current;
    current->next->pervious=temp;
    current->next=temp;
```



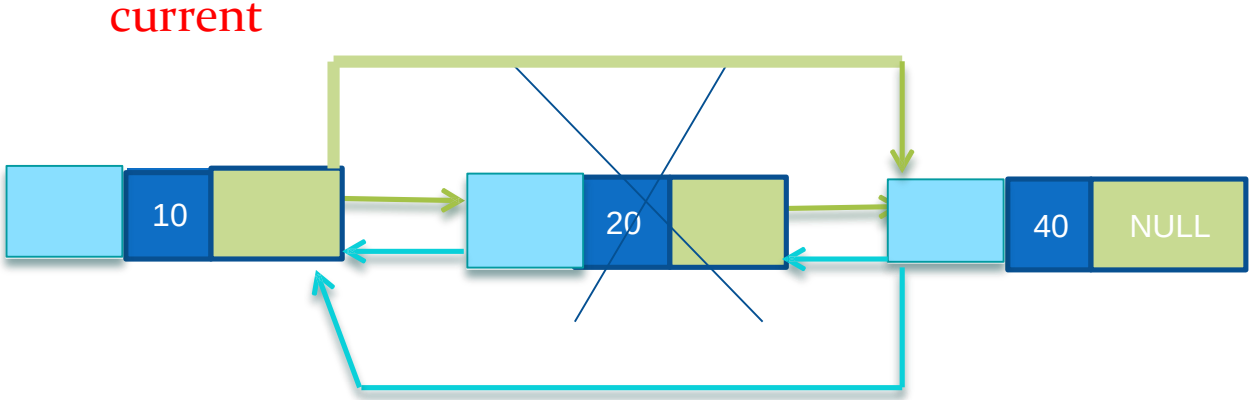
Linked List: delete_ position

```
void linked_list::delete_at_position(int pos)
{
    node *current=start;
    node *pre=start;
    for(int i=1;i<pos;i++)
     { pre=current; current=current->next;
          if(current==NULL) {cout<<" the list has element less than"
                                  <<pos<< " element "; return;}}
     pre->next=current->next;
    current->next->pervious=pre;
    delete current;
}
```



Linked List: delete_ position

```
void linked_list::delete_at_position(int pos)
{
    node *current=start;
    for(int i=1;i<pos;i++)
     {  current=current->next;
          if(current==NULL) {cout<<" the list has element less than"
                                  <<pos<< " element ";return;}}
    current->pervious->next=current->next;
    current->next->pervious=current->pervious;
    delete current;
}
```



Circular linked list

Circular linked list

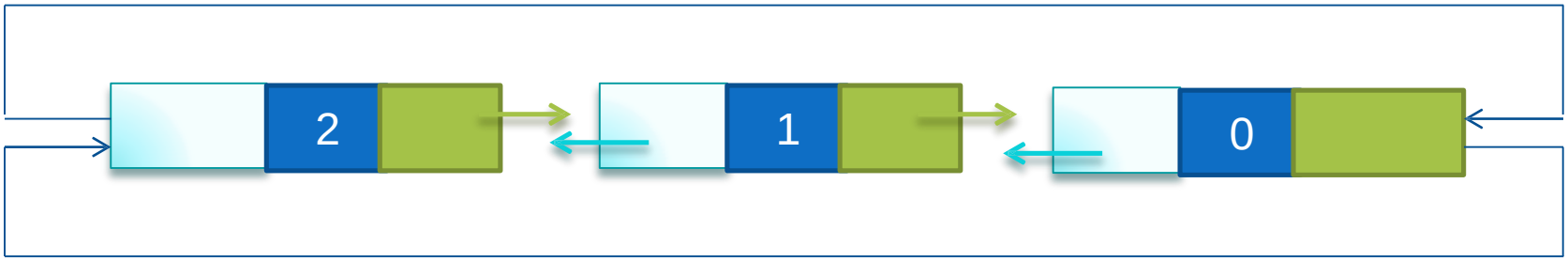
- A circular linked list no end.

- A singly linked list can be made a circular linked list by simply storing the address of the very first node in the linked field of the last node.

- A circular doubly linked list has both the successor pointer and predecessor pointer in circular manner

Circular linked list(cont.)

Circular linked list



Circular Doubly Linked list

Application of circular linked list

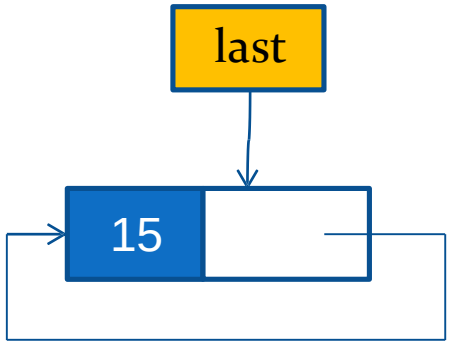
- Operating systems may use it to switch between various running applications in a circular loop.
- Circular Doubly Linked Lists are used for implementation of advanced data structures like Fibonacci Heap
- Multiplayer games uses circular list to swap between players in a loop.

Circular singly linked list

The pointer last points to node last node and last -> next points to first node .

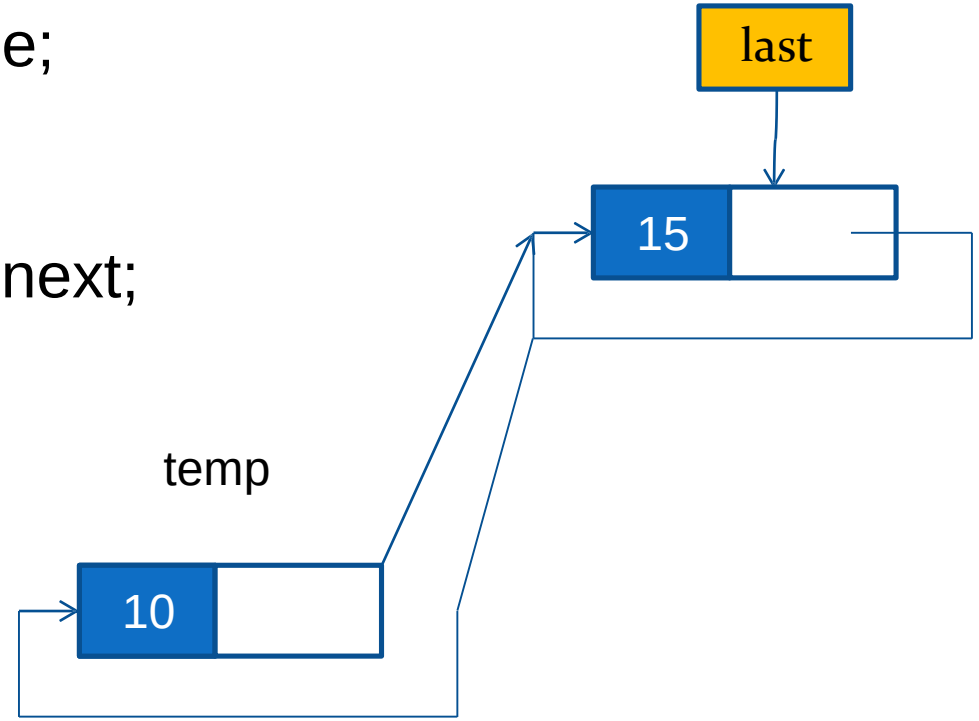
Circular singly linked list: create list

```
void linked_list:: createlist(int value)
  {
                if (last != NULL)
                    return ;

                last=new node;

                last -> data = value

// Note : list was empty. We link single node
// to itself.
                 last -> next = last;
  }
```



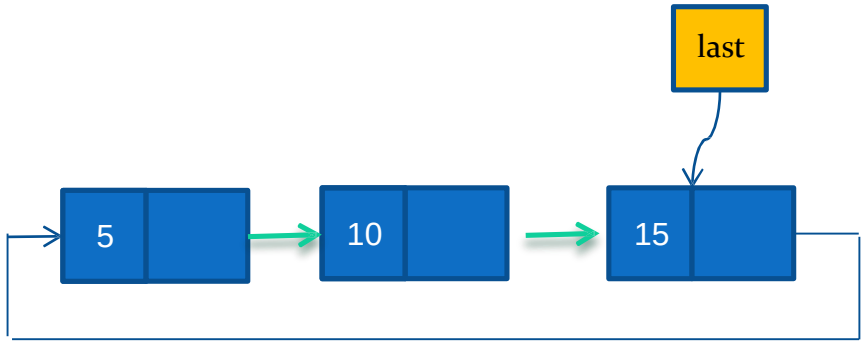
Circular singly linked list: insert first

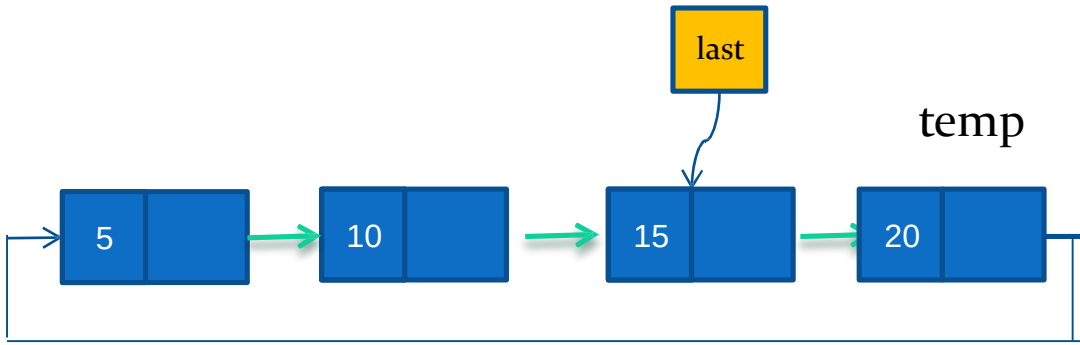
```
void linked_list:: insert_first(int value)
{
        if (last == NULL) createlist(value) ;
// now add required element
        node *temp= new node;
        temp -> data = value;

        temp -> next = last -> next;
        last -> next = temp;


}
```



Circular singly linked list: insert last

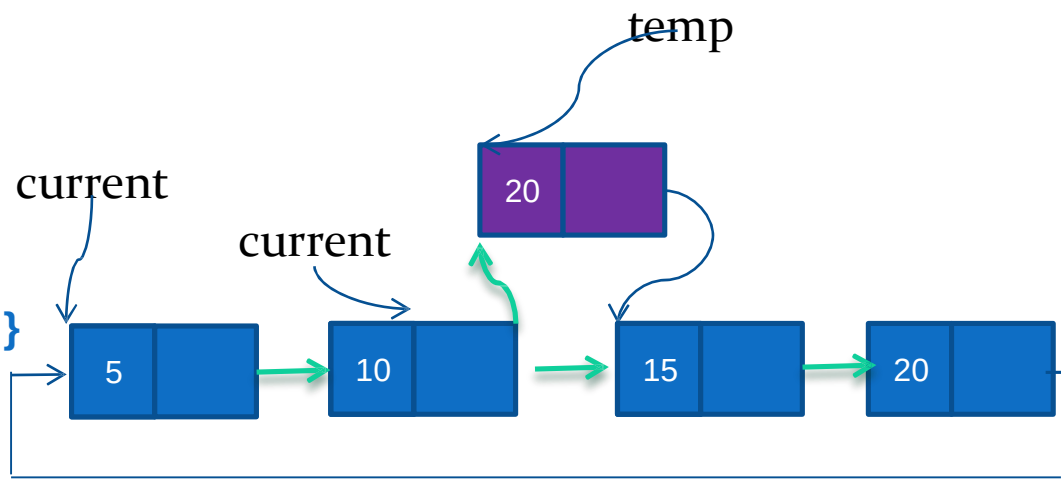
```
void linked_list:: insert_last(int value)
{ //test if list is empty
if (last == NULL)
return createlist(value);
// add required element
node *temp= new node;
temp->data=value;



temp->next=last->next;
last->next=temp;
last=temp;

}
```





Circular singly linked list: insertpos

```
void linked_list::insert_position(int pos, int value)
{
    node *temp,* current;
   current = last->next; // beginning of list

    for(int i=1; i < pos-1; i++)
      { current = current->next;
       if (current == last->next)
         { cout<<"There are less than"
        <<pos<<" elements\n"; return; }
       }//creating the new node
     temp = new node;
    temp->data = value;
    temp->next = current->next;
    current->next = temp;
     if(current==last)     //Element inserted at the end
          last=temp;
```



Circular Linked List:display

```
void linked_list:: display()
{
        if(last==NULL) {cout<<"list is empty";return;}
        node *temp=new node;
        temp=last->next;
        cout<<" the elements of the list are    "<<endl;
        while(temp!=last)
                { cout<<temp->data<<"\t";
                temp=temp->next;
                }
        cout<<last->data<<endl;
}
```


THANK
YOU


Any
Question?