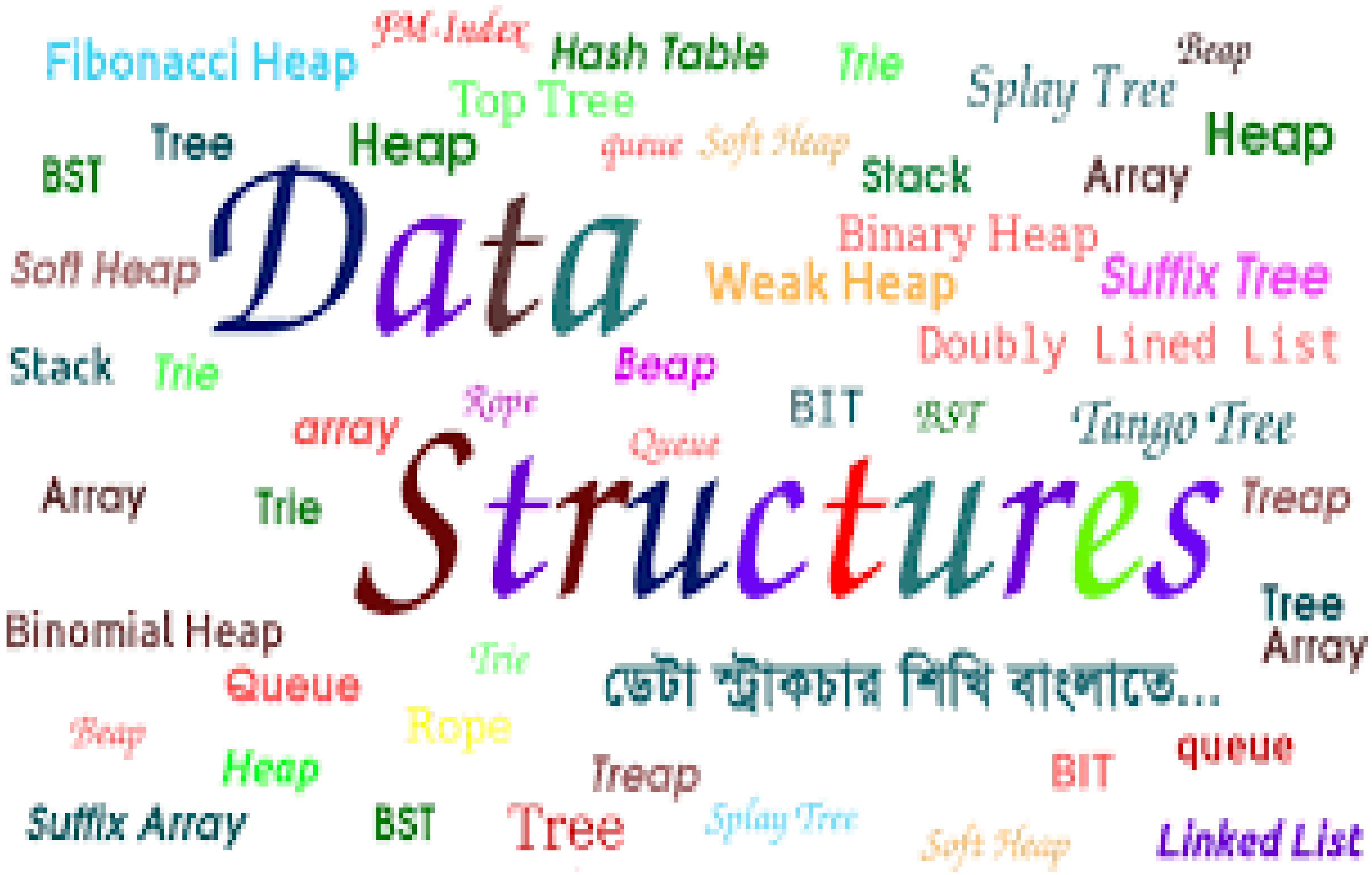
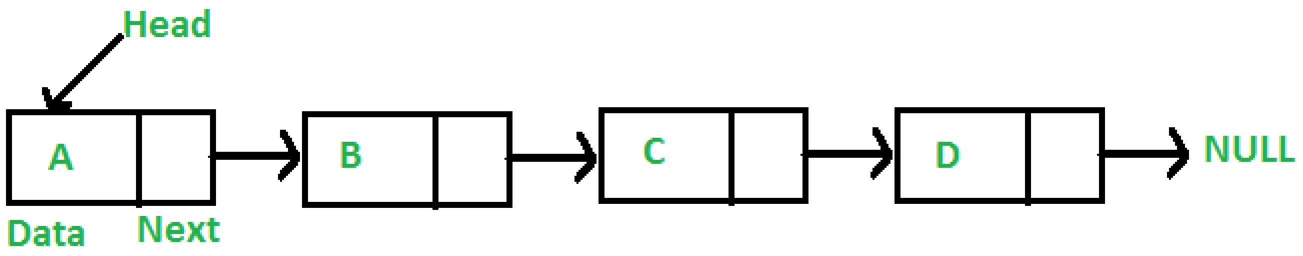

FM-Index
Fibonacci Heap
Hash Table
Trie
Beap
Splay Tree
Top Tree
Tree
Heap
queue
Soft Heap
Heap
BST
Stack
Array
Data
Binary Heap
Soft Heap
Weak Heap
Suffix Tree
Doubly Lined List
Stack
Trie
Beap
Rope
BIT
BST
Tango Tree
array
Queue
Array
Trie
Structures
Treap
Binomial Heap
Trie
Tree
Array
ডেটা স্ট্রাকচার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

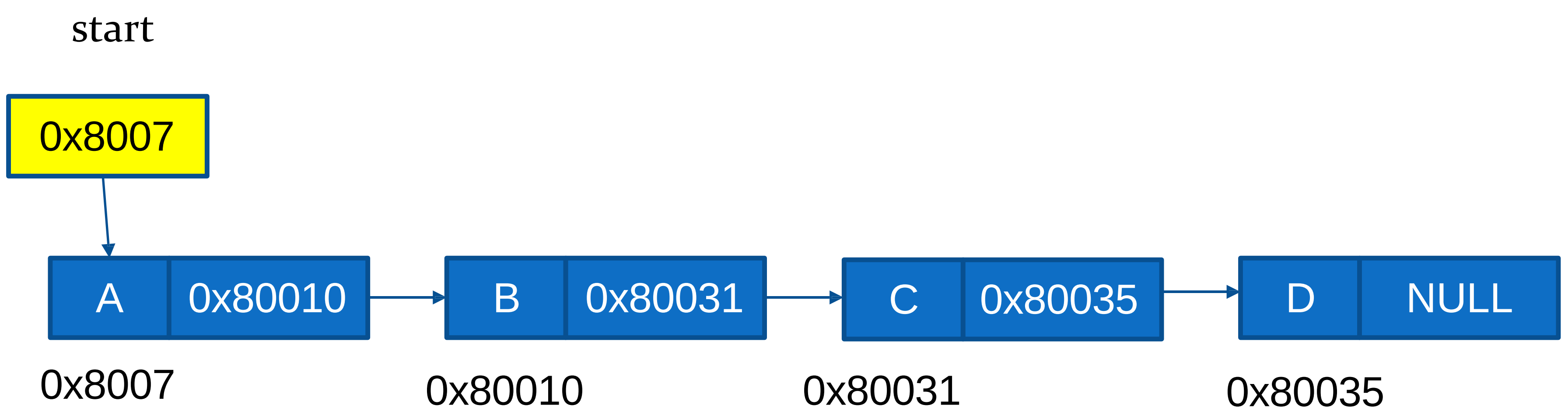
Linked list

- Linked list is linear collection of specially designed data elements called **nodes**, linked to one another by means of pointer.

- Each node is divided into twoparts:
  - the first part contains the information of element
  - The second part(linked or next field) contains the address of the next node



Linked list



- This linked list has four nodes
- The address of the first node is stored in the pointer start
- Each node has two components: data, to store the data, and next, to store the address of the next node

linked list : Advantages and disadvantages

- Some advantages of linked list:

1. Linked list are dynamic data structure. That is, they can grow or shrink during the execution of a program.
2. Efficient memory utilization: In linked list (or dynamic) representation, memory is not pre-allocated. Memory is allocated whenever it is required. And it is deallocated (or removed) when it is not needed.
3. Insertion and deletion are easier and efficient. Linked list provides flexibility in inserting a data item at a specified position and deletion of a data item from the given position.
4. Many complex applications can be easily carried out with linked list.

linked list : Advantages and disadvantages (cont.)

Some disadvantages of linked list:

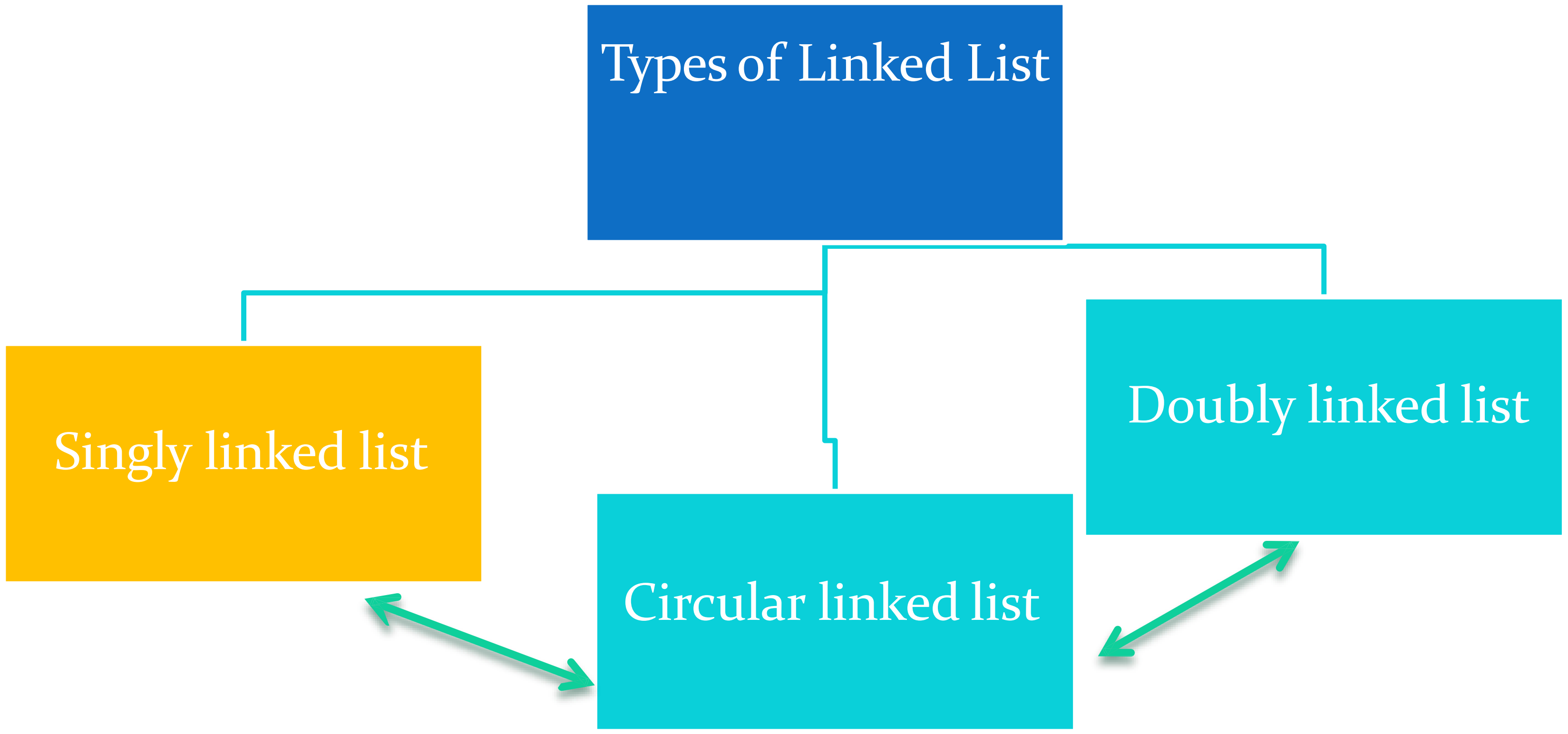
1. More memory: to store an integer number, a node with integer data and address field is allocated. That is more memory space is needed.

1. Access to an arbitrary data item is little bit cumbersome and also timeconsuming

Operations on linked list

Creation	• used to create a linked list
insertion	• used to insert a new node at any specified location in the linked list • Insertion at beginning or end or any specified position
deletion	used to delete a node at any specified location in the linked list Deletion at beginning or end or any specified position
Traversing	• the process of going through all the nodes from one end to another end of a linked list
Concatenation	• the process of appending the second list to the end of the first list

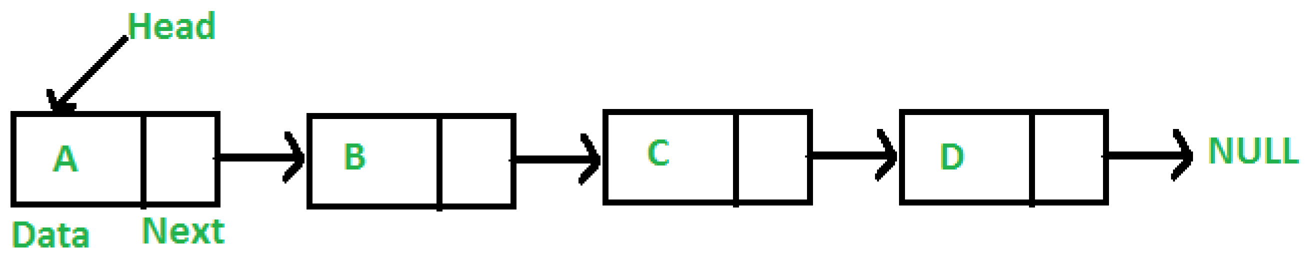
TYPES OF LINKED LIST



Singly linked list

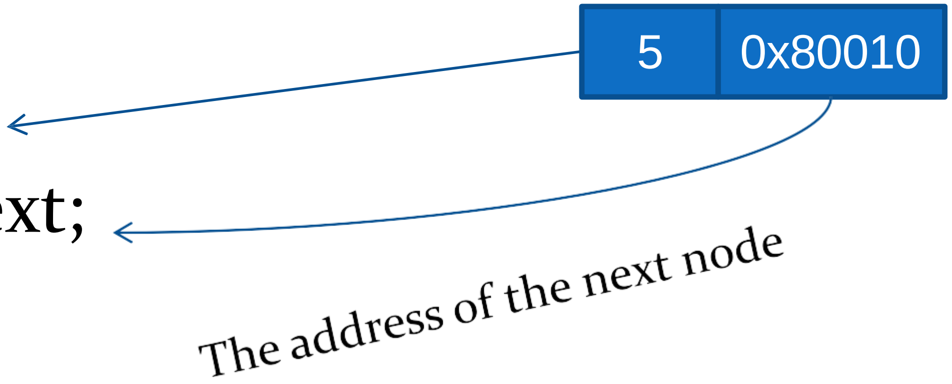
SINGLY LINKED LIST

- All the nodes in a singly linked list are arranged sequentially by linking with apointer
- we can easily access the next node but there is no way of accessing the previous node



SINGLY LINKED LIST: implementation of a node

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



SINGLY LINKED LIST

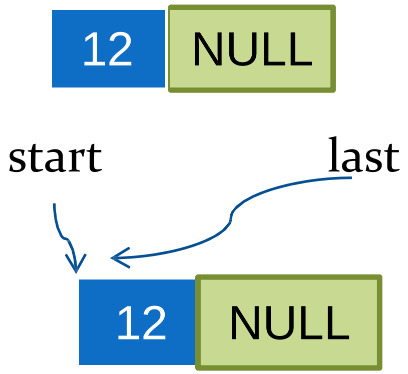
- create a class '**linked_list**' which will contain all the functions and data members required for a linked list.
- This class will use the structure 'node' for the creation of the linked list.
- We have made two nodes – start and last.
- store the first node in 'start' and the last node in 'last'.
- The constructor of the linked list is making both 'start ' and ' last' NULL because we have not yet added any element to our linked list and thus both areNULL.

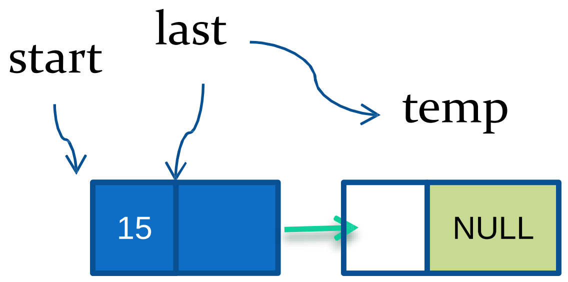
Creation of Linked List

```
class linked_list
{
        private:
                node *start; node * last;
        public:
                linked_list()
                 { start=NULL; last= NULL;}

                void add_node(int value);
                void insert_first(int value) ;
                void display();
 };
```

Linked List:add nodes forward

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

           last = temp;

          }
    }
```





Linked List:add nodes forward

- **(node *temp=new node)** – We are allocating the space required for a node by the newoperator. Now, 'temp' points to a node (or space allocated for the node).

- **(temp->data = value**)– We are giving a value to the 'data' of 'temp' as passed to the function.

- **(temp->next=NULL)** – We have given the value to'data' in the previous line and a value of the pointer 'next' (NULL) in this line and thus making our node 'temp' complete.

- **if (start == NULL)** means that there is no linked listyet and our current node(temp) will be the'start'.

Linked List: add nodes backward

- make a new node

  ```
  node *temp=new node
  ```

- points its next to the start of the existing list

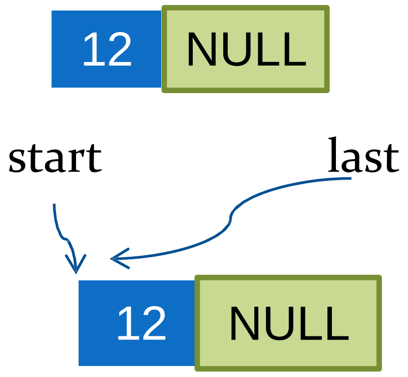
  ```
  temp->next=start
  ```

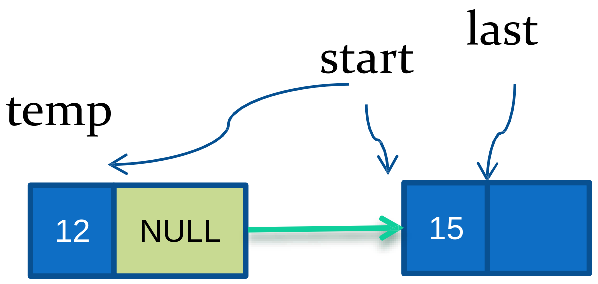
- change the start to the newly added node.

  ```
  start =temp
  ```

Linked List:insert_first

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





insert a node in between a linked list

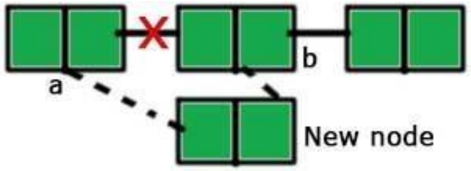
- The steps for inserting a node between two nodes a and b:

1. Make a new node

   node *tmp=new node

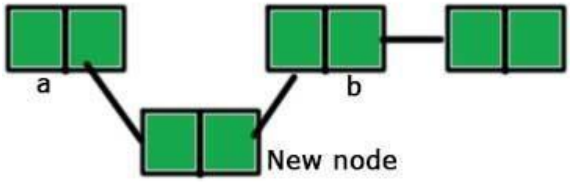
2. Point the 'next' of the new node to the node 'b' (the node after which we have to insert the new node). Till now, two nodes are pointing the same node 'b', the node 'a' and the new node.

   temp->next=b



3. Point the 'next' of 'a' to the new node.

   a->next=temp



Linked List:display

```
void linked_list:: display()
{
        if(start==NULL) {cout<<"list is empty ";return;}
        node *temp=start;

        cout<<" the elements of the list are    "<<endl;

        while(temp!=NULL)
        {
                cout<<temp->data<<"\t";
                temp=temp->next;
        }

}
```

Main function

```
int main()
{
    int value; linked_list list;
    cin>>value;
    list.add_node(value);
    list.add_node(10);
    list.insert_first(4);
    list.insert_first(5);
    list.insert_first(6);
    list.display();
    return 0;
}
```

THANK YOU

Any Question?