Need for Data Structure:

- It gives different level of organization data.
- It tells how data can be stored and accessed in its elementary level.
- Provide operation on group of data, such as adding an item, looking up highest priority item.
- Provide a means to manage huge amount of data efficiently.
- Provide fast searching and sorting of data.

Advantages of Data Structures

• **Efficient Memory use:** With efficient use of data structure memory usage can be optimized, for e.g we can use linked list vs arrays when we are not sure about the size of data. When there is no more use of memory, it can be released.

• **Reusability:** Data structures can be reused, i.e. once we have implemented a particular data structure, we can use it at any other place. Implementation of data structures can be compiled into libraries which can be used by different clients.

int a[5] = {2, 3, 5, 7}

4 x 4

int a[100] = { ---- }

400 bytes

int a[5] = {2, 3, 5, 7, 10, 12}

① → Memory management

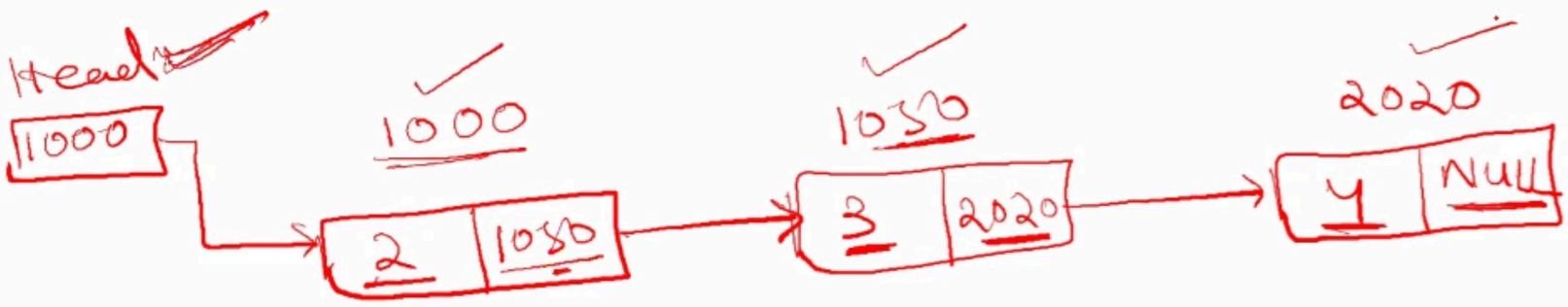
Memory fixed

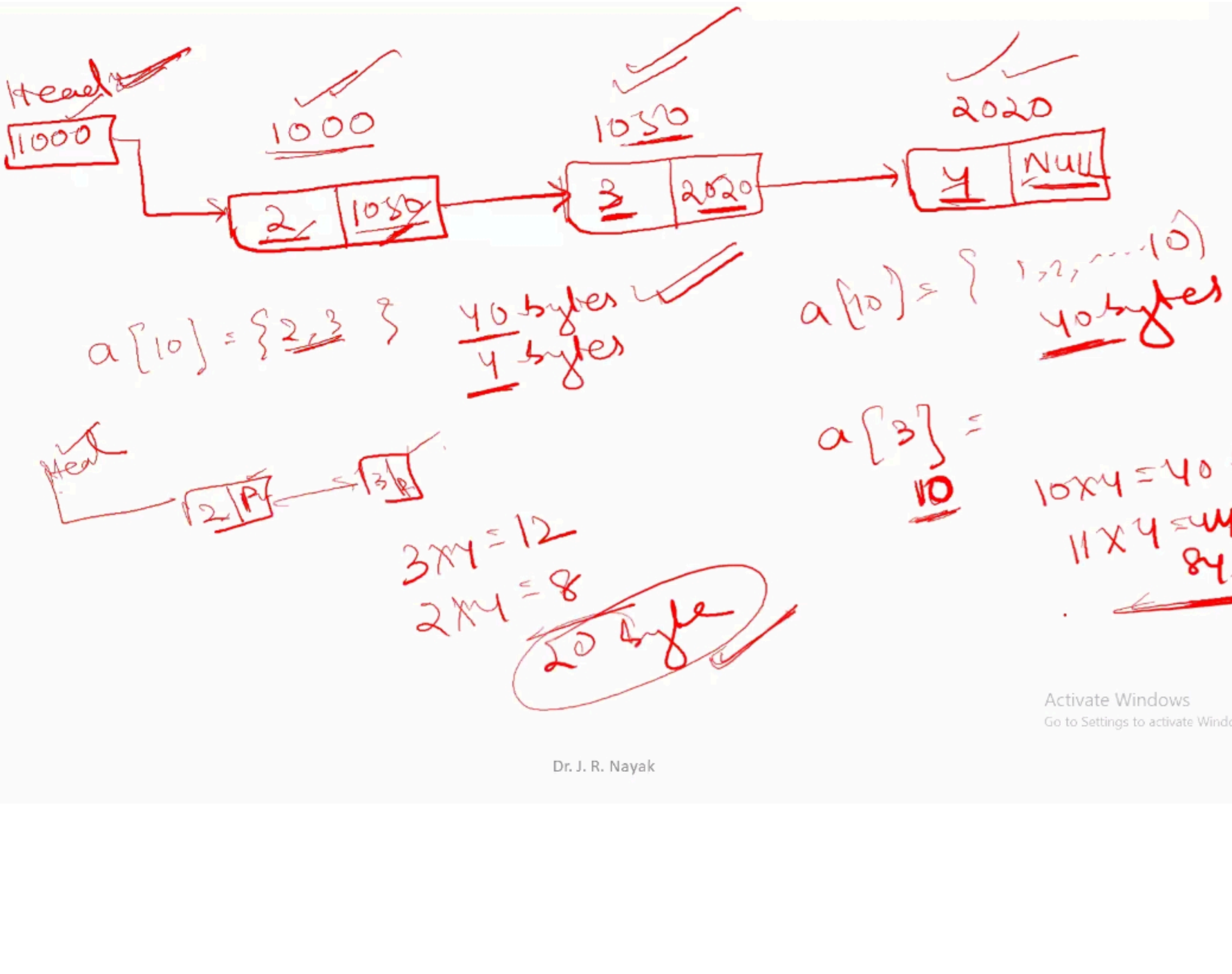
②

Linked List

- A Linked List is a linear data structure consisting of connected nodes where each node has corresponding data and a pointer to the address of the next node.
- The first node of a linked list is called the Head, and it acts as an access point.
- On the other hand, the last node is called the Tail, and it marks the end of a linked list by pointing to a *NULL* value!

- A node consists of the data value and a pointer to the address of the next node within the linked list.
- A linked list is a dynamic linear data structure whose memory size can be allocated or de-allocated at run time based on the operation insertion or deletion, this helps in using system memory efficiently.





Advantages and Disadvantage of Linked list

PROS

- They are dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.

CONS

- The memory is wasted as pointers require extra memory for storage.
- No element can be accessed randomly, it has to access each node sequentially i.e. proper traversal must be done.
- Reverse Traversing is difficult in the linked list(though we can achieve this with the help of Doubly Linked List).

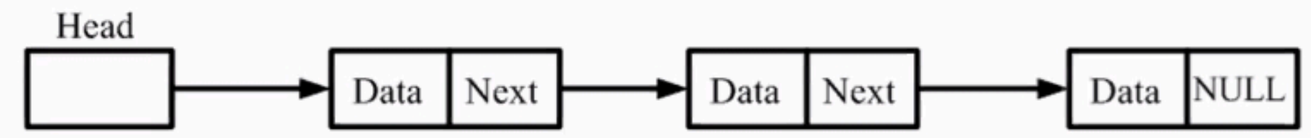
Linked lists real-world applications:

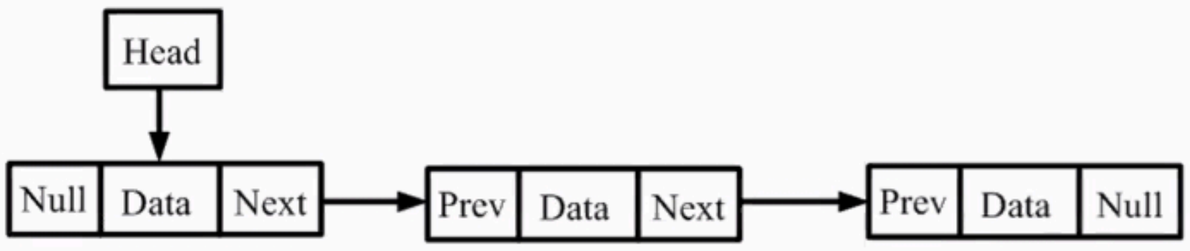
- Music and Video Playlists
- Web Browser History
- Undo and Redo Functionality in Text Editors
- Dynamic Memory Allocation (Operating Systems)
- Social Media Feeds
- Graph Implementations in Google Maps
- Job Scheduling in CPU
- Hash Tables with Collision Handling
- Version Control Systems (Git)
- AI and Machine Learning (Neural Networks)

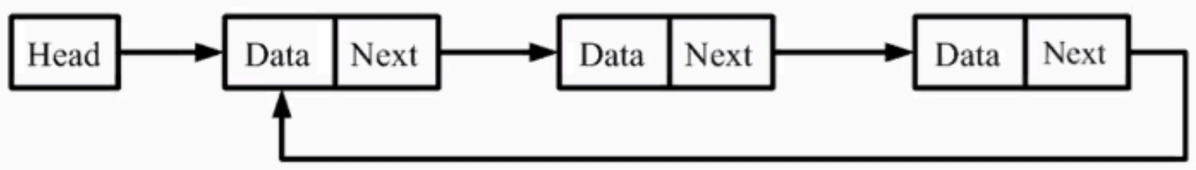
Types of linked lists:

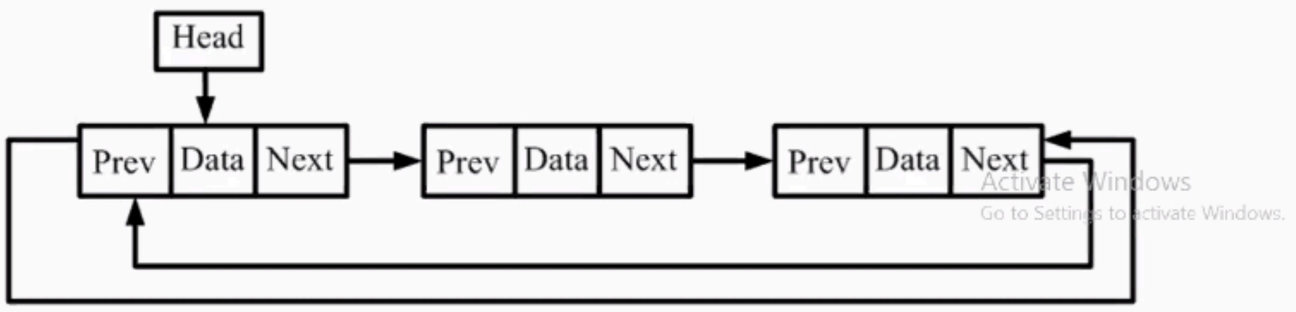
There are four key types of linked lists:

1. Singly linked lists
2. Doubly linked lists
3. Circular linked lists
4. Circular doubly linked list









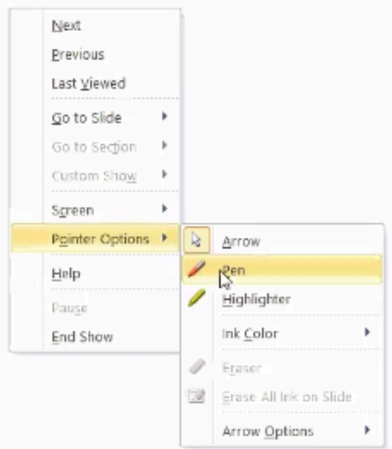
Representation of linked lists

Single linked list

- We can visualize a linked list as a chain of nodes; where every node contains a data field and a reference link to the next node in the list.
- The first node in the sequence is termed as 'head'. And we can identify the last node as the one which points to 'NULL'.
- Each node consists of -
  - A data field
  - Reference to the next node

Algorithm for creating single linked list

1. Create a structure/class Node which has two attributes: data and next. Next is a pointer to the next node.
2. Create two pointers: head <- NULL and temp <- NULL.
3. Add a new node to the list:
   i. Create a new_node. Set new_node <- next as NULL.
   ii. If head <- NULL
   iii. head <- new_node and temp <- new_node
   iv. Else
   v. temp.next <- new_node and temp <- new_node
4. Repeat step 3 for the number of nodes to be entered.



Creating single linked list using structure

```
#include <iostream>
using namespace std;
struct node{
   int data;
   node* next;
};
int main(){
   node* head = nullptr;
   node* temp = nullptr;
   int choice=1;
   while(choice==1){
      node* new_node = new node();
      int value;
      cout<<"Enter the value"<<endl;
      cin>>value;
      new_node->data = value;
      new_node->next = nullptr;
      if(head==nullptr){
         head=new_node;
         temp=new_node;
      }
      else{
         temp->next=new_node;
         temp=new_node;
      }
      cout<<"Enter choice=1 to add a new node ";
      cin>>choice;
   }
   node* temp1=head;
   while (temp1 != nullptr) {
      cout << temp1->data << endl;
      cout << temp1<<endl;
      temp1 = temp1->next;
      }
   return 0;
}
```

Input:

Enter the value

10

Enter choice=1 to add a new node 1

Enter the value

20

Enter choice=1 to add a new node 1

Enter the value

30

Enter choice=1 to add a new node 0

Output:

10

0x776e00

20

0x776e10

30

0x776e20

Creating single linked list using class

```
#include <iostream>
using namespace std;
class node{
   public:
   int data;
   node* next;
};
int main(){
   node* head = nullptr;
   node* temp = nullptr;
   int choice=1;
   while(choice==1){
      node* new_node = new node();
      int value;
      cout<<"Enter the value"<<endl;
      cin>>value;
      new_node->data = value;
      new_node->next = nullptr;
      if(head==nullptr){
         head=new_node;
         temp=new_node;
      }
      else{
         temp->next=new_node;
         temp=new_node;
      }
      cout<<"Enter choice=1 to add a new node ";
      cin>>choice;
   }
   node* temp1=head;
   while (temp1 != nullptr) {
      cout << temp1->data << endl;
      cout << temp1<<endl;
      temp1 = temp1->next;
      }
   return 0;
}
```