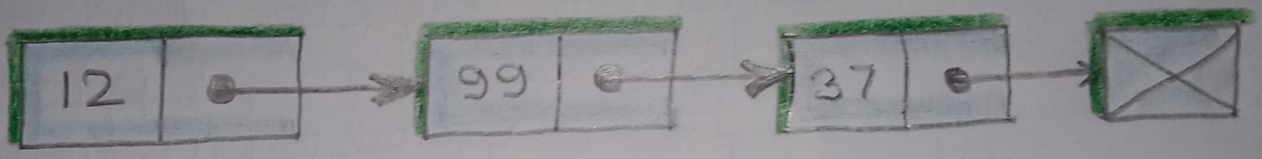
ASSIGNMENT - 2

NAME : AMRITANSHU KESHARI
BRANCH : COMP. SCIENCE & ENGG.
ROLL NUMBER : 19402060006
SUBJECT : D.S.A.
SESSION : 2019-22
SEMESTER : 4th
COLLEGE : GOVT. POLYTECHNIC COLLEGE, ADITYAPUR

Linked list

In computer science, a linked list is a linear collection of data elements whose order is not given by their physical placement in memory. Instead, each element points to the next. It is a datastructure consisting of a collection of nodes which together represent a sequence. In its most basic form, each node contains: data, and a reference (in other words, a link) to the next node in the sequence. The this structure allows for efficient insertion or removal of elements from any position in the sequence during iteration. More complex variants add the additional links, allowing more efficient insertion or removal of nodes at arbitrary positions. A drawback of linked lists is that access time is linear (and difficult to pipeline). Faster access such as random access, is not feasible. Arrays have better cache locality compared to

Linked list



Compared to linked lists.

Here is the image at the left side show a linked list whose nodes contain two fields:

- → an integer value
- → a link to the next node

The last node is linked to a terminator used to signify the end of the list.

Linked lists are among the simplest and most common data structures. They can be used to implement several other common abstract data types, including lists, stacks, queues, associative arrays and S-expressions, though it is not uncommon to implement those data structures directly without using a linked list as the basis.

The principal benefit of a linked list over a conventional array is that the list elements can be easily inserted or removed without reallocation or

reorganization of the entire structure because the data items need not be stored contiguously in memory or on disk, while restructuring an array at run-time is a much more expensive operation. Linked lists allow insertion and removal of nodes at any point in the list, and allow doing so with a constant number of operations by keeping the link being added or removed in memory during list traversal.

On the other hand, since simple linked lists by themselves do not allow random access to the data or any form of efficient indexing, many basic operations are — such as obtaining the last node of the list, finding a node that contains a given datum or locating the place where a new node should be inserted — may require iterating through most or all of the list elements. The advantages and disadvantages of using linked lists are given below. Linked lists are dynamic, so the length of a linked list can increase

increase or decrease as necessary. Each node does not necessarily follow the previous one physically in the memory.

Application of Linked List

Application of linked list data structure. A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown. In the application of linked list in the computer science - Implementation of stacks and queues.

What kind of structure is a linked list ?

A linked list is a collection of structures ordered not by their physical placement in memory but by logical links that are stored as part of the data in the structure itself.

How are links added and removed

removed in a linked list ?

Linked lists allow insertion and removal of nodes at any point in the list, and allow doing so with a constant number of operations by keeping the link previous to the link being added or removed in memory during list traversal.

Uses of Linked List

Here are some uses of linked list are as follows as :

- The list is not required to be contiguously present in the memory. The node can reside any where in the memory and linked together to make a list. This achieves optimized utilization of space.
- The list size is limited to the memory size and does not need to be declared in advance.
- Empty node cannot be present in the linked list.

Linked List Representation

Linked list can be visualized as a chain of nodes, where every node points to the next node.

As per the above illustration, the following are the important points to be considered.

→ Linked List contains a link element called first.
→ Each link carries a data field(s) and a link field called next.
→ Each link is linked with its next link using its next link.
→ Last link carries a link as null to mark the end of the list.

Types of Linked List

Following are the various types of linked list.

- Simple Linked List
  Items navigation is forward only.
- Doubly Linked List

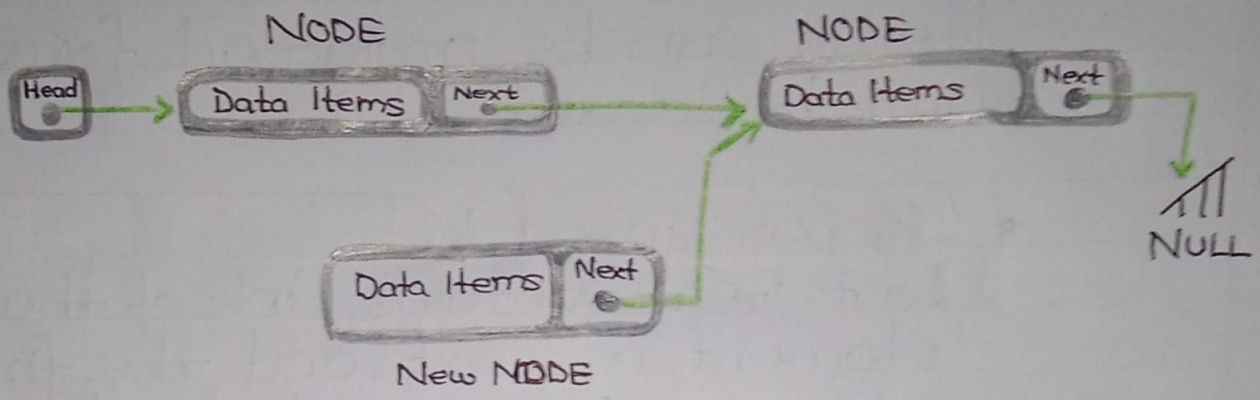
Items can be navigated forward and backward.

- Circular Linked List
  Last item contains link of the first element as next and the first element has a link to the last element as previous.

Basic Operation

Following are the basic operations supported by a list.

- Insertion
  Adds an element at the beginning of the list.
- Deletion
  Deletes an element at the beginning of the list.
- Display
  Display the complete list.
- Search
  Searches an element using the given key.



INSERTION OPERATION

- Delete

Delete an element using the given key.

Insertion Operation

Adding a new node in linked list is a more than one step activity. We shall learn this diagram here. First, create a node using the same structure and find the location where it has to be inserted.

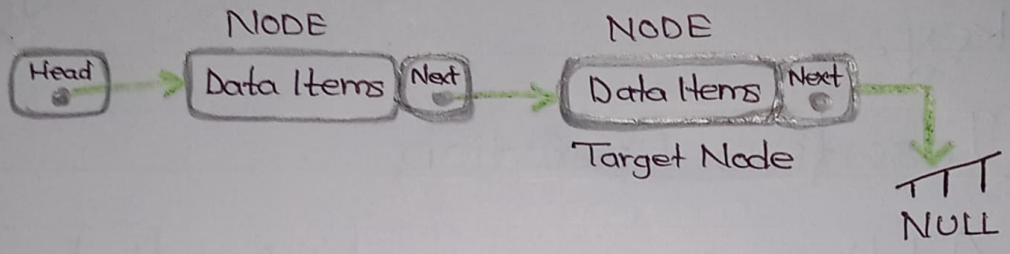
Imagine that we are inserting a node B (NewNode), between A (LeftNode) and C (RightNode). Then point B next to C -

```
NewNode.next -> RightNode;
```

Now, the next node at the left should point to the new node.

```
LeftNode.next --> NewNode;
```

This will put the new node in the



DELETION OPERATION

in the middle of the two. The new list should took look like this – given the figure.

Similar steps should be taken if the node is being inserted at the beginning of the list. While inserting it at the ends the second last mode node of the list should point to the new node and the new node will point to NULL.

Deletion Operation

Deletion is also a more than one step process. We shall learn with pictorial representation. First, locate the target node to be removed, by using searching algorithms.

The left (previous) node of the target node now should point to the next node of the target node –

```
LeftNode.next --> TargetNode.next;
```

This will remove the link that

that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at.

```
TargetNode.next → NULL;
```

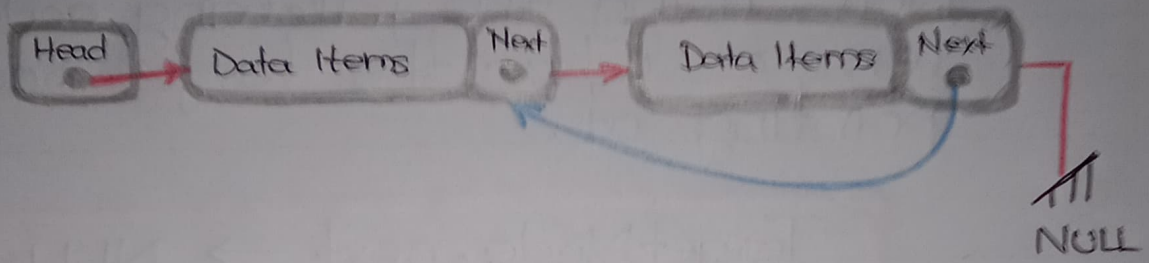
We need to use the deleted node. We can keep that in memory otherwise we can simply deallocate memory and wipe off the target node completely.

Reverse Operation

This operation is a thorough one. We need to make the last node to be pointed by the head node and reverse the whole linked list.

First, we traverse to the end of the list. It should be pointing to NULL. Now, we shall make it point to its previous node.

We have to make sure that the last node is not the last



REVERSE OPERATION

not the last node. So we'll have some temp node, which looks like the head node pointing to the last node. Now, we shall make all left side nodes point to their previous nodes one by one.

Except the node (first node) pointed by the head node, all nodes should point to their predecessor, making them their new successor. The first node will point to NULL.

We'll make the head node point to the new first node by using the temp node.

The linked list is now reversed. To see linked list implementation in C programming language.

Doubly Linked List

Doubly linked list is a variation of linked list in which navigation is possible in both ways, either forward and backward easily as compared to single linked list. Following are the important terms to understand the concept of doubly linked list.

- Link
  Each link of a linked list can store a data called an element.

- Next
  Each link of a linked list contains a link to the next link called Next.

- Prev
  Each link of a linked list contains a link to the previous link called Prev.

- Linked List
  A linked list contains the connection link to the first link called First and to the last link called Last.

Doubly Linked List Representation

As per the above illustration, following are the important points to be considered.

- Doubly Linked List contains a link element called first and last.
- Each link carries a data field(s) and two link fields called next and prev.
- Each link is linked with its next link using its next link.
- Each link is linked with its previous link using its previous link.
- The last link carries a link as null to mark the end of the list.

Some basic operations

Following are the basic operations supported by a list.

Insertion
Adds an element at the beginning of the list.

Deletion
Deletes an element at the beginning of the list.

Insert Last
Adds an element at the end of the list.

Delete Last
Deletes an element at the end of the list.

Insert After
Adds an element after an items of item of the list.

Delete
Delete an element after an item of the list.

Display Forward
Displays the complete list in a forward manner.

Display Backward
Displays the complete list in a backward manner.

Insertion Operation

Following code demonstrates the insertion operation at the beginning of a doubly linked list.

Example :

```
void insertFirst (int key, int data)
{
  // create a link
  struct node* link = (struct node*)
  malloc (sizeof (struct node));

  link -> key = key;
  link -> data = data;

  if (isEmpty ())
  {
    // make it the last link
      last = link;
  }

  else
  {
```

```
// update first prev link
   head -> prev = link;
}

// point it to old first link
   link -> next = head;

// point first to new first link
   head = link;
}
```

Deletion Operation

Following code demonstrates the deletion operation at the beginning of a doubly linked list.

Example

```
// delete first item
struct node * deleteFirst()
{
   // save reference to first link
   struct node * tempLink = head;

   // if only one link
   if (head -> next == NULL)
   {
```

```
    last = NULL;
}

else
   {
      head -> next -> prev = NULL;
   }

   head = head -> next;

   // return the deleted link
   return tempLink;
}
```

Insertion at the End of an Operation

Following code demonstrates the insertion operation at the last position of a doubly linked list.

Example

```
// insert link at the last location
void insertLast (int key, int data)
  {
     // create a link
     struct node *link = (struct node*)
     malloc (sizeof (struct node));
```

```
link -> key = key;
link -> data = data;

if (isEmpty())
  {
    // make it the last link
    last = link;
  }
else
  {
    // make link a new last link
    last -> next = link;
    // make old last node as prev of
    // new link
    link -> prev = last;
  }

// point last to new last node
last = link;
}
```

Circular Linked List

Circular linked list is a variation of linked list in which the first element points to the last element and the last element points to the first element. Both singly linked list

and doubly linked list can be made into a circular linked list.

Singly Linked List as Circular

In singly linked list, the next pointer of the last node points to the first node.

Doubly Linked List as Circular

In doubly linked list, the next pointer of the last node points to the first node and the previous pointer of the first node points to the last node making the circular in both directions.

As per the above illustration, following are the important points to be considered.

- The last link's next points to the first link of the list in both cases of singly as well as doubly linked list.

- The first link's previous points to the last of the list in case of doubly linked list.

Basic Operations

Here are the important operations supported by a circular list are as follows as:

Insert
Insert an element at the start of the list.

Delete
Deletes an element from the start of the list.

Display
Display the list.

Insertion Operation

Following code demonstrates the insertion operation in a circular linked list based on single linked list.

Example

```
insertFirst (data) :
Begin
    create a new node
    node -> data := data
    if the list is empty, then
        head := node
        next of node = head
    else
        temp := head
        while next of temp is not head,
        do
        temp := next of temp
        done
        next of node := head
        next of temp := node
        head := node
    end if
End
```

Deletion Operation

Following code demonstrates the deletion operation in a circular linked list based on single linked list.

Example :

```
deleteFirst ():
Begin
   if head is null , then
      it is Underflow and return
   else if next of head = head , then
      head := null
      deallocate head
   else
      ptr := head
      while next of ptr is not head, do
         ptr := next of ptr
      next of ptr = next of head
      deallocate head
      head := next of ptr
   end if
End
```

Display List Operation

Following code demonstrates the display list operation in a circular linked list.

Example :

```
display ():
```

```
Begin
    if head is null, then
      Nothing to print and return
    else
      ptr := head
      while next of ptr is not head, do
        display data of ptr
        ptr := next of ptr
      display data of ptr
    end if
End
```