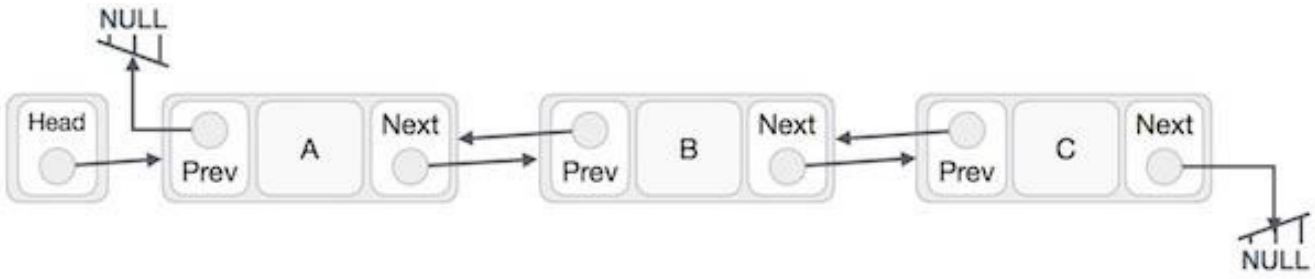
Doubly Linked List Data Structure

What is Doubly Linked List?

Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, forward as well as 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 a link field called next.
- 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.

Basic Operations in Doubly Linked List

Following are the basic operations supported by a list.

- **Insertion** – Adds an element at the beginning of the list.
- **Insert Last** – Adds an element at the end of the list.
- **Insert After** – Adds an element after an item of the list.
- **Deletion** – Deletes an element at the beginning of the list.
- **Delete Last** – Deletes an element from the end of the list.
- **Delete** – Deletes an element from the list using the key.
- **Display forward** – Displays the complete list in a forward manner.
- **Display backward** – Displays the complete list in a backward manner.

Doubly Linked List - Insertion at the Beginning

In this operation, we create a new node with three compartments, one containing the data, the others containing the address of its previous and next nodes in the list. This new node is inserted at the beginning of the list.

Algorithm

```
1. START
2. Create a new node with three variables: prev, data, next.
3. Store the new data in the data variable
4. If the list is empty, make the new node as head.
5. Otherwise, link the address of the existing first node to the
next variable of the new node, and assign null to the prev variable.
```

```
6. Point the head to the new node.
7. END
```

Example

Following are the implementations of this operation in various programming languages –

C | C++ | Java | Python

```c
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
struct node {
   int data;
   int key;
   struct node *next;
   struct node *prev;
};

//this link always point to first Link
struct node *head = NULL;

//this link always point to last Link
struct node *last = NULL;
struct node *current = NULL;

//is list empty
bool isEmpty(){
   return head == NULL;
}

//display the doubly linked list
void printList(){
   struct node *ptr = head;
   while(ptr != NULL) {
```

```
      ptr = ptr->next;
   }
}

//insert link at the first location
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;
}
void main(){
   insertFirst(1,10);
   insertFirst(2,20);
   insertFirst(3,30);
   insertFirst(4,1);
   insertFirst(5,40);
   insertFirst(6,56);
   printf("\nDoubly Linked List: ");
   printList();
}
```

Output

```
Doubly Linked List: (6,56) (5,40) (4,1) (3,30) (2,20) (1,10)
```

Doubly Linked List - Insertion at the End

In this insertion operation, the new input node is added at the end of the doubly linked list; if the list is not empty. The head will be pointed to the new node, if the list is empty.

Algorithm

```
1. START
2. If the list is empty, add the node to the list and point
   the head to it.
3. If the list is not empty, find the last node of the list.
4. Create a link between the last node in the list and the
   new node.
5. The new node will point to NULL as it is the new last node.
6. END
```

Example

Following are the implementations of this operation in various programming languages –

C | C++ | Java | Python

```c
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
struct node {
   int data;
   int key;
   struct node *next;
```

```
};

//this link always point to first Link
struct node *head = NULL;

//this link always point to last Link
struct node *last = NULL;
struct node *current = NULL;

//is list empty
bool isEmpty(){
   return head == NULL;
}

//display the doubly linked list
void printList(){
   struct node *ptr = head;
   while(ptr != NULL) {
      printf("(%d,%d) ",ptr->key,ptr->data);
      ptr = ptr->next;
   }
}

//insert link at the first location
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;
}

//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;

      //mark old last node as prev of new link
      link->prev = last;
   }

   //point last to new last node
   last = link;
}
void main(){
   insertFirst(1,10);
   insertFirst(2,20);
   insertFirst(3,30);
   insertFirst(4,1);
   insertLast(5,40);
   insertLast(6,56);
```

```
   printList();
}
```

Output

```
Doubly Linked List: (4,1) (3,30) (2,20) (1,10) (5,40) (6,56)
```

Doubly Linked List - Deletion at the Beginning

This deletion operation deletes the existing first nodes in the doubly linked list. The head is shifted to the next node and the link is removed.

Algorithm

```
1. START
2. Check the status of the doubly linked list
3. If the list is empty, deletion is not possible
4. If the list is not empty, the head pointer is
   shifted to the next node.
5. END
```

Example

Following are the implementations of this operation in various programming languages –

C | C++ | Java | Python

```
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
struct node {
   int data;
   int key;
```

```
   struct node *prev;
};

//this link always point to first Link
struct node *head = NULL;

//this link always point to last Link
struct node *last = NULL;
struct node *current = NULL;

//is list empty
bool isEmpty(){
   return head == NULL;
}

//display the doubly linked list
void printList(){
   struct node *ptr = head;
   while(ptr != NULL) {
      printf("(%d,%d) ",ptr->key,ptr->data);
      ptr = ptr->next;
   }
}

//insert link at the first location
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
```

```
   }

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

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

//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;
}
void main(){
   insertFirst(1,10);
   insertFirst(2,20);
   insertFirst(3,30);
   insertFirst(4,1);
   insertFirst(5,40);
   insertFirst(6,56);
   printf("Doubly Linked List: \n");
   printList();
   printf("\nList after deleting first record: \n");
   deleteFirst();
```

```
}
```

Output

```
Doubly Linked List: (6,56) (5,40) (4,1) (3,30) (2,20) (1,10)
List after deleting first record: (5,40) (4,1) (3,30) (2,20) (1,10)
```

Doubly Linked List - Complete Implementation

Following are the complete implementations of Doubly Linked List in various programming languages –

C | C++ | Java | Python

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
struct node {
   int data;
   int key;
   struct node *next;
   struct node *prev;
};

//this link always point to first Link
struct node *head = NULL;

//this link always point to last Link
struct node *last = NULL;
struct node *current = NULL;

//is list empty
bool isEmpty(){
   return head == NULL;
}
```

```
//display the list in from first to last
void displayForward(){

   //start from the beginning
   struct node *ptr = head;

   //navigate till the end of the list
   printf("\n[ ");
   while(ptr != NULL) {
      printf("(%d,%d) ",ptr->key,ptr->data);
      ptr = ptr->next;
   }
   printf(" ]");
}

//display the list from last to first
void displayBackward(){

//start from the last
   struct node *ptr = last;

   //navigate till the start of the list
   printf("\n[ ");
   while(ptr != NULL) {

      //print data
      printf("(%d,%d) ",ptr->key,ptr->data);

      //move to next item
      ptr = ptr ->prev;
      printf(" ");
   }
   printf(" ]");
}

//insert link at the first location
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;
}

//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;

      //mark old last node as prev of new link
      link->prev = last;
```

```
   }

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

//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;
}

//delete link at the last location
struct node* deleteLast(){

   //save reference to last link
   struct node *tempLink = last;

   //if only one link
   if(head->next == NULL) {
      head = NULL;
   } else {
      last->prev->next = NULL;
   }
   last = last->prev;

   //return the deleted link
```

```
   return tempLink;
}

//delete a link with given key
struct node* delete(int key){

   //start from the first link
   struct node* current = head;
   struct node* previous = NULL;

   //if list is empty
   if(head == NULL) {
      return NULL;
   }

   //navigate through list
   while(current->key != key) {

      //if it is last node
      if(current->next == NULL) {
         return NULL;
      } else {

         //store reference to current link
         previous = current;

         //move to next link
         current = current->next;
      }
   }

   //found a match, update the link
   if(current == head) {

      //change first to point to next link
      head = head->next;
   } else {

      //bypass the current link
```

```
      current->prev->next = current->next;
   }
   if(current == last) {

      //change last to point to prev link
      last = current->prev;
   } else {
      current->next->prev = current->prev;
   }
   return current;
}
bool insertAfter(int key, int newKey, int data){

   //start from the first link
   struct node *current = head;

   //if list is empty
   if(head == NULL) {
      return false;
   }

   //navigate through list
   while(current->key != key) {

      //if it is last node
      if(current->next == NULL) {
         return false;
      } else {

         //move to next link
         current = current->next;
      }
   }

   //create a link
   struct node *newLink = (struct node*) malloc(sizeof(struct nc
   newLink->key = key;
   newLink->data = data;
   if(current == last) {
```

```
      newLink->next = NULL;
      last = newLink;
   } else {
      newLink->next = current->next;
      current->next->prev = newLink;
   }
   newLink->prev = current;
   current->next = newLink;
   return true;
}
int main(){
   insertFirst(1,10);
   insertFirst(2,20);
   insertFirst(3,30);
   insertFirst(4,1);
   insertFirst(5,40);
   insertFirst(6,56);
   printf("\nList (First to Last): ");
   displayForward();
   printf("\n");
   printf("\nList (Last to first): ");
   displayBackward();
   printf("\nList , after deleting first record: ");
   deleteFirst();
   displayForward();
   printf("\nList , after deleting last record: ");
   deleteLast();
   displayForward();
   printf("\nList , insert after key(4) : ");
   insertAfter(4,7, 13);
   displayForward();
   printf("\nList , after delete key(4) : ");
   delete(4);
   displayForward();
```

Output

```
List (First to Last):
[ (6,56) (5,40) (4,1) (3,30) (2,20) (1,10)  ]

List (Last to first):
[ (1,10)  (2,20)  (3,30)  (4,1)  (5,40)  (6,56)   ]
List , after deleting first record:
[ (5,40) (4,1) (3,30) (2,20) (1,10)  ]
List , after deleting last record:
[ (5,40) (4,1) (3,30) (2,20)  ]
List , insert after key(4) :
[ (5,40) (4,1) (4,13) (3,30) (2,20)  ]
List , after delete key(4) :
[ (5,40) (4,13) (3,30) (2,20)  ]
```