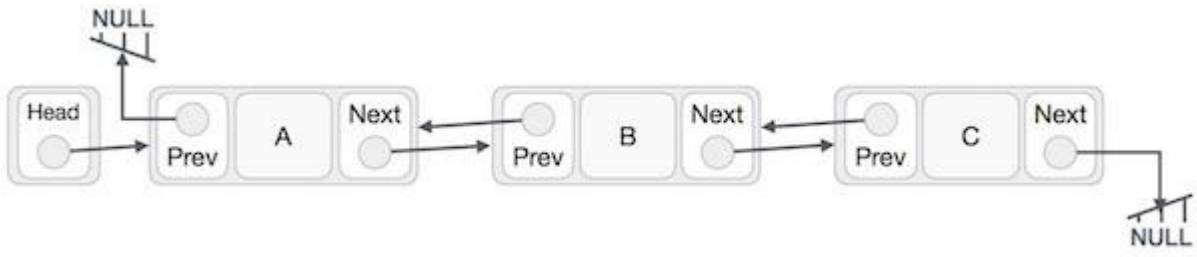
Doubly Linked List Data Structure

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

</> Open Compiler

```python
#Python code for doubly linked list
class Node:
    def __init__(self, data=None, key=None):
        self.data = data
        self.key = key
        self.next = None
        self.prev = None
#this link always point to first Link
head = None
#this link always point to last Link
last = None
current = None
#is list empty
def is_empty():
    return head == None
#display the doubly linked list
def print_list():
    ptr = head
    while ptr != None:
        print(f"({ptr.key},{ptr.data})")
        ptr = ptr.next
#insert link at the first location
def insert_first(key, data):
    global head, last
    #create a link
    link = Node(data, key)
    if is_empty():
        #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_first(1,10)
insert_first(2,20)
insert_first(3,30)
insert_first(4,1)
insert_first(5,40)
```

```
insert_first(6,56)
print("Doubly Linked List: ")
print_list()
```

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

</> Open Compiler

```
class Node:
    def __init__(self, data=None, key=None):
        self.data = data
        self.key = key
        self.next = None
        self.prev = None
head = None
last = None
```

```
current = None
def isEmpty():
    return head == None
def printList():
    ptr = head
    while ptr != None:
        print(f"({ptr.key},{ptr.data})", end=" ")
        ptr = ptr.next
def insertFirst(key, data):
    global head, last
    link = Node(data, key)
    if isEmpty():
        last = link
    else:
        head.prev = link
    link.next = head
    head = link
def insertLast(key, data):
    global head, last
    link = Node(data, key)
    if isEmpty():
        last = link
    else:
        last.next = link
        link.prev = last
    last = link
insertFirst(1,10)
insertFirst(2,20)
insertFirst(3,30)
insertFirst(4,1)
insertLast(5,40)
insertLast(6,56)
print("Doubly Linked List: ", end="")
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

</> Open Compiler

```
#Python code for doubly linked list
class Node:
    def __init__(self, data=None, key=None):
        self.data = data
        self.key = key
        self.next = None
        self.prev = None
#this link always point to first Link
head = None
#this link always point to last Link
last = None
current = None
#is list empty
def isEmpty():
    return head == None
#display the doubly linked list
def printList():
    ptr = head
    while ptr != None:
        print(f"({ptr.key},{ptr.data}) ", end="")
        ptr = ptr.next
```

```
#insert link at the first location
def insertFirst(key, data):
    #create a link
    global head, last
    link = Node(data, key)
    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
    head = link
#delete first item
def deleteFirst():
     #save reference to first link
    global head, last
    tempLink = head
    #if only one link
    if head.next == None:
        last = None
    else:
        head.next.prev = None
    head = head.next
    #return the deleted link
    return tempLink
insertFirst(1,10)
insertFirst(2,20)
insertFirst(3,30)
insertFirst(4,1)
insertFirst(5,40)
insertFirst(6,56)
print("Doubly Linked List:")
printList()
print("\nList after deleting first record:")
deleteFirst()
printList()
```

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


```python
class Node:
    def __init__(self, key, data):
        self.key = key
        self.data = data
        self.next = None
        self.prev = None

class DoublyLinkedList:
    def __init__(self):
        self.head = None
        self.last = None

    def is_empty(self):
        return self.head is None

    def display_forward(self):
        ptr = self.head
        print("[", end=" ")
        while ptr:
            print("({}, {})".format(ptr.key, ptr.data), end=" ")
            ptr = ptr.next
        print("]")

    def display_backward(self):
        ptr = self.last
        print("[", end=" ")
        while ptr:
```

```
            print("({}, {})".format(ptr.key, ptr.data), end=" ")
            ptr = ptr.prev
        print("]")

    def insert_first(self, key, data):
        link = Node(key, data)
        if self.is_empty():
            self.last = link
        else:
            self.head.prev = link
        link.next = self.head
        self.head = link

    def insert_last(self, key, data):
        link = Node(key, data)
        if self.is_empty():
            self.last = link
        else:
            self.last.next = link
            link.prev = self.last
        self.last = link

    def delete_first(self):
        if self.is_empty():
            return None
        temp_link = self.head
        if self.head.next is None:
            self.last = None
        else:
            self.head.next.prev = None
        self.head = self.head.next
        return temp_link

    def delete_last(self):
        if self.is_empty():
            return None
        temp_link = self.last
        if self.head.next is None:
            self.head = None
        else:
            self.last.prev.next = None
        self.last = self.last.prev
```

```
        return temp_link

    def delete(self, key):
        current = self.head
        while current and current.key != key:
            current = current.next
        if current is None:
            return None
        if current == self.head:
            self.head = self.head.next
        else:
            current.prev.next = current.next
        if current == self.last:
            self.last = current.prev
        else:
            current.next.prev = current.prev
        return current

    def insert_after(self, key, new_key, data):
        current = self.head
        while current and current.key != key:
            current = current.next
        if current is None:
            return False
        new_link = Node(new_key, data)
        if current == self.last:
            new_link.next = None
            self.last = new_link
        else:
            new_link.next = current.next
            current.next.prev = new_link
        new_link.prev = current
        current.next = new_link
        return True

# Example usage
dll = DoublyLinkedList()
dll.insert_first(1, 10)
dll.insert_first(2, 20)
dll.insert_first(3, 30)
dll.insert_first(4, 1)
dll.insert_first(5, 40)
```

```
dll.insert_first(6, 56)
print("List (First to Last):")
dll.display_forward()
print()
print("List (Last to First):")
dll.display_backward()
print("List, after deleting first record:")
dll.delete_first()
dll.display_forward()
print("List, after deleting last record:")
dll.delete_last()
dll.display_forward()
print("List, insert after key(4):")
dll.insert_after(4, 7, 13)
dll.display_forward()
print("List, after delete key(4):")
dll.delete(4)
dll.display_forward()
```

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) (7, 13) (3, 30) (2, 20) ]
List, after delete key(4):
[ (5, 40) (7, 13) (3, 30) (2, 20) ]
```

C++ Tutorial

C Programming Tutorial

C# Tutorial

PHP Tutorial

R Tutorial

HTML Tutorial

CSS Tutorial

JavaScript Tutorial

SQL Tutorial

TRENDING TECHNOLOGIES

Cloud Computing Tutorial

Amazon Web Services Tutorial

Microsoft Azure Tutorial

Git Tutorial

Ethical Hacking Tutorial

Docker Tutorial

Kubernetes Tutorial

DSA Tutorial

Spring Boot Tutorial

SDLC Tutorial

Unix Tutorial

CERTIFICATIONS

Business Analytics Certification

Java & Spring Boot Advanced Certification

Data Science Advanced Certification

Cloud Computing And DevOps

Advanced Certification In Business Analytics

Artificial Intelligence And Machine Learning

DevOps Certification

Game Development Certification

Front-End Developer Certification

AWS Certification Training

Python Programming Certification

COMPILERS & EDITORS

Online Java Compiler

Online Python Compiler

Online Go Compiler

Online C Compiler

Online C++ Compiler

Online C# Compiler

Online PHP Compiler

Online MATLAB Compiler

Online Bash Terminal

Online SQL Compiler

Online Html Editor