**LINKED LIST**

**DEFINITION**

A linked list is a linear data structure. Its elements can be traversed using pointers. Linked list is made up of nodes.

**TYPES OF LINKED LIST**

There are four types of linked list:

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

But here we will be talking only on the singly and doubly linked list.

1. **SINGLY LINKED LIST**

Singly linked list is a linear data structure made up of nodes. These nodes have two parts which include;

1. Data
2. Reference of next node

**OPERATIONS OF SINGLY LINKED LIST**

**INSERTION**

* At the beginning
* At the last
* At a specific position

Code:

//A c++ code to insert a node

//in singly linked list

#include <bits/stdc++.h>

using namespace std;

//A class to create nodes

class Node

{

public:

int data;

Node \*next;

};

// A function to insert a node at the

//beginning of singly linked list

void push(Node\*\* head, int newdata)

{

Node\* newnode = new Node();//creating newnode

newnode->data = newdata; //put in data

newnode->next = (\*head); //link newnode to head

(\*head) = newnode; //changing head

}

// A function to insert a node after

//a specific node in a singly linked list

void insertAfter(Node\* prevnode, int newdata)

{

//check if previous node is null

if (prevnode == NULL)

{

cout<< “the given previous node cannot be NULL”;

return;

}

Node\* newnode = new Node();//creating newnode

newnode->data = newdata; //put in data

//link newnode to prevnode’s next node

newnode->next = prevnode->next;

prevnode->next = newnode; //link prevnode to newnode

}

// A function to insert a node at the

//end of singly linked list

void append(Node\*\* head, int newdata)

{

Node\* newnode = new Node();//creating newnode

Node \*last = \*head; // creating a ‘last’ node

newnode->data = newdata; //put in data

newnode->next = NULL; //link newnode with null

//Check if head is null

if (\*head == NULL)

{

\*head = newnode;

return;

}

//traversing ‘last’ node to end of the linked list

while (last->next != NULL)

last = last->next;

//link ‘last’ node with newnode

last->next = newnode;

return;

}

// A function to print the given linked list

// starting from the given node

void printList(Node \*node)

{

while (node != NULL)

{

cout<<" "<<node->data;

node = node->next;

}

}

/\* Driver code\*/

int main()

{

/\* Start with the empty list \*/

Node\* head = NULL;

// Insert 6 at the end,

append(&head, 6);

//6->NULL

// Insert 7 as head

push(&head, 7);

//7->6->NULL

// Insert 1 as head.

push(&head, 1);

//1->7->6->NULL

// Insert 4 at the end

append(&head, 4);

//1->7->6->4->NULL

// Insert 8, after 7

insertAfter(head->next, 8);

//1->7->8->6->4->NULL

cout<<"Created Linked list is: ";

printList(head);

return 0;

}

**DELETION**

* At the beginning
* At the last
* At a specific position

Code:

//A c++ code to insert a node

//in singly linked list

#include <bits/stdc++.h>

using namespace std;

//A class to create node

class Node{

public:

int data;

Node\* next;

};

//insert a node at the beginning

void push(Node\*\* head, int newdata)

{

//create newnode

Node\* newnode = new Node();

newnode->data = newdata;//put in data

newnode->next = (\*head);//link newnode with head

(\*head) = newnode;//changing head

}

//A function to delete a node

void deleteNode(Node\*\* head, int key)

{

Node\* temp = \*head;//creating temp node

Node\* prev = NULL;//creating prev node

//checking if node to be deleted is head the node

if (temp != NULL && temp->data == key)

{

\*head = temp->next;//changing head

delete temp; //delete node

return;

}

else

{

//traversing to find key to delete

while (temp != NULL && temp->data != key)

{

prev = temp;

temp = temp->next;

}

if (temp == NULL)

return;

prev->next = temp->next;

delete temp;//delete node

}

}

// This function prints contents of

// linked list starting from the

// given node

void printList(Node\* node)

{

while (node != NULL)

{

cout << node->data << " ";

  node = node->next;

}

}

// Driver code

int main()

{

// Start with the empty list

Node\* head = NULL;

// Add elements in linked list

push(&head, 7);

push(&head, 1);

push(&head, 3);

push(&head, 2);

puts("Created Linked List: ");

printList(head);

deleteNode(&head, 1);

puts("\nLinked List after Deletion of 1: ");

printList(head);

return 0;

}

**What are the benefits of a singly linked list?**

* You can perform operations like insertion and deletion with ease
* It is a dynamic data structure, i.e., it does not have a fixed size
* It doesn’t require the movement of needs for insertion and deletion
* It doesn’t need elements to be stored in consecutive memory spaces
* It doesn’t waste space as it uses space according to the requirement

**What are the limitations of a singly linked list?**

* It requires more storage space because it also stores the next pointer with data
* If you have to reach any node, then you have to go through every node before it
* You can’t traverse it from anywhere but the head node
* It requires a different amount of time to access any elements
* Sorting is complex in this linked list

**Conclusion**

* Singly linked list is a dynamic data structure
* Singly linked list are faster at insertion and deletion
* Singly linked list are unidirectional linked list i.e **you can only do it one** **direction**
* Singly linked list is good at implementing youstacks and queues

1. **DOUBLY LINKED LIST**

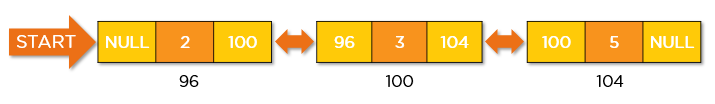
Doubly linked list is a linear data structure made up of nodes

We can traverse it in both directions. It nodes have three parts.

The doubly linked List comprises of:

1. DATA
2. Reference to the next node
3. Reference to the previous node

How Do You Implement a Doubly Linked List?



You create nodes of doubly-linked lists using classes or structures. These nodes are then linked with each other using the next and the previous pointer.

Code:

//A c++ program to implement linked list

#include <bits/stdc++.h>

using namespace std;

/\* A class to create node \*/

class Node

{

public:

int data;

Node \*next;

Node \*prev;

};

//A function to insert at the

//beginning of the list

void push(Node\*\* head, int newdata)

{

//create new node

Node\* newnode = new Node();

/\* put in the data \*/

newnode->data = newdata;

/\* As we are adding at the beginning,

prev is always NULL \*/

newnode->prev = NULL;

/\* link new node's next to head \*/

newnode->next = (\*head);

/\* change prev of head node to newnode \*/

if((\*head) != NULL)

(\*head)->prev = newnode ;

/\* changing head node \*/

(\*head) = newnode;

}

/\* A c++ program to print the list \*/

void printlist(Node \*head)

{

while(head != NULL)

{

cout << head->data << " ";

head = head->next;

}

}

int main()

{

/\* We will start with an empty list \*/

Node\* head = NULL;

/\*lets create a linked list: 2->3->5->7 \*/

push(&head, 7);

push(&head, 5);

push(&head, 3);

push(&head, 2);

cout << "Created Doubly Linked list:" << endl;

printlist(head);

return 0;

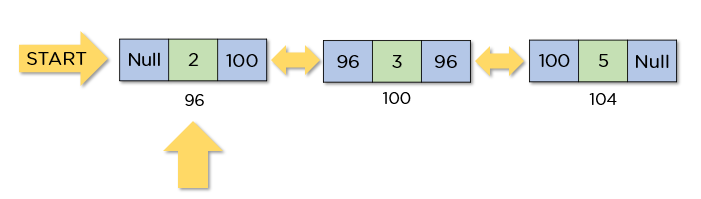
}

**OPERATIONS OF A DOUBLY LINKED LIST**

**TRAVERSAL**

* Normal traversal
* Reverse traversal

**How do you traverse a doubly linked list?**

****

In a doubly linked list traversal operation, we visit every node at least once to display all the data elements or perform operations.

You can traverse this linked list in two different directions, they are:

* Normal traversal, i.e., from head node to tail node
* Reverse traversal, i.e., from tail node to head node

### **Code:**

/\* A C++ code to traverse a linked list \*/

#include <bits/stdc++.h>

using namespace std;

/\* A class to create a node \*/

class Node

{

public:

int data;

Node \*next;

Node \*prev;

};

//A function to insert at the

//beginning of the list

void push(Node\*\* head, int newdata)

{

/\* creating newnode \*/

Node\* newnode = new Node();

/\* put in the data \*/

newnode->data = newdata;

/\* since we are insert at the beginning of the list,

prev is always NULL \*/

newnode->prev = NULL;

/\* link the next of newnode to the head \*/

newnode->next = (\*head);

/\* change prev of head node to newnode \*/

if((\*head) != NULL)

(\*head)->prev = newnode ;

/\* changing head \*/

(\*head) = newnode;

}

/\* A c++ program to traverse the linked list \*/

void traverse(Node \*node)

{

while(node != NULL)

{

cout << node->data << " ";

node = node->next;

}

}

/\* Function to reverse traverse a Linked List \*/

void revtraverse(Node \*\*head)

{

Node\* tail = \*head;

    // Traversing till tail of the linked list

    while (tail->next != NULL) {

        tail = tail->next;

    }

    // Traversing linked list from tail

    // and printing the node->data

    while (tail != \*head) {

   cout << tail->data << " ";

        tail = tail->prev;

    }

    cout << tail->data << endl;;

}

int main()

{

/\* Start with the empty list \*/

Node\* head = NULL;

/\* Let us create a linked list: 2->3->5->7 \*/

push(&head, 7);

push(&head, 5);

push(&head, 3);

push(&head, 2);

cout << "Original Linked list" << endl;

traverse(head);

/\* Reverse linked list \*/

cout << "\nReversed Linked list" << endl;

revtraverse(&head);

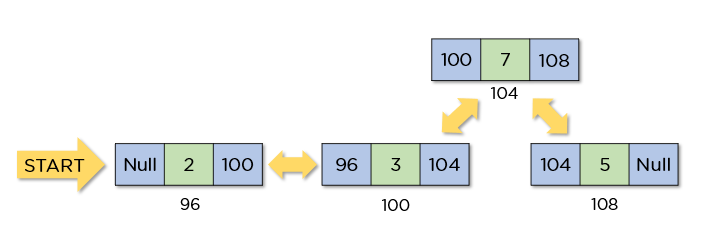
return 0;

}

**INSERTION**

* At the beginning
* At the last
* At a specific position

**How do you insert a node in a doubly linked list?**

****

To insert a node, you need to change the previous node's next (if any) to the new node and the next node's previous (if any) to the new node. You can insert a new node in three different locations.

* At the beginning of the list
* At the end of the list
* After a given node

### **Code:**

/\* A c++ program to perform all insertion operations\*/

#include <bits/stdc++.h>

using namespace std;

// A class to create nodes

class Node

{

public:

int data;

Node\* next;

Node\* prev;

};

/\* A function to insert a node at the beginning of the list\*/

void push(Node\*\* head, int newdata)

{

/\* create newnode \*/

Node\* newnode = new Node();

/\* put in the data \*/

newnode->data = newdata;

/\* link the new node's next to head

and previous as NULL \*/

newnode->next = (\*head);

newnode->prev = NULL;

/\* link the head node's prev to new node \*/

if ((\*head) != NULL)

(\*head)->prev = newnode;

/\* changing head \*/

(\*head) = newnode;

}

/\* A function to insert a node after a given node \*/

void insertAfter(Node\* prevnode, int newdata)

{

/\*1. check if the given prevnode is NULL \*/

if (prevnode == NULL)

{

cout<<"given previous node can't be null";

return;

}

/\* 2. allocate new node \*/

Node\* newnode = new Node();

/\* 3. put in the data \*/

newnode->data = newdata;

/\* 4. Make new node's next as prevnode's next \*/

newnode->next = prevnode->next;

/\* 5. Make the prevnode's next as newnode \*/

prevnode->next = newnode;

/\* 6. Make prevnode as newnode's prev \*/

newnode->prev = prevnode;

/\* 7. Change previous of newnode's next node \*/

if (newnode->next != NULL)

newnode->next->prev = newnode;

}

/\* A function to insert at the end of the list \*/

void append(Node\*\* head, int newdata)

{

/\* create newnode \*/

Node\* newnode = new Node();

Node\* last = \*head;

/\* put in the data \*/

newnode->data = newdata;

/\*This newnode is going to be the last node, so

we will make next of it as NULL\*/

newnode->next = NULL;

/\* check if the Linked List is empty, then make the new

node as head \*/

if (\*head == NULL)

{

newnode->prev = NULL;

\*head = newnode;

return;

}

/\* Else traverse till the last node \*/

while (last->next != NULL)

last = last->next;

/\* Change the next of last node \*/

last->next = newnode;

/\* Make last node as new node's prev \*/

newnode->prev = last;

return;

}

// A function to print the list

void printList(Node\* node)

{

while (node != NULL)

{

cout<<" "<<node->data<<" ";

node = node->next;

}

}

int main()

{

/\* Start with the empty list \*/

Node\* head = NULL;

// Insert 6 at the last

append(&head, 6); //6->NULL

// Insert 7 at the beginning

push(&head, 7); //7->6->NULL

// Insert 1 at the beginning

push(&head, 1); //1->7->6->NULL

// Insert 4 at the end

append(&head, 4); //1->7->6->4->NULL

// Insert 8, after 7

insertAfter(head->next, 8); //1->7->8->6->4->NULL

cout << "Created DLL is: ";

printList(head);

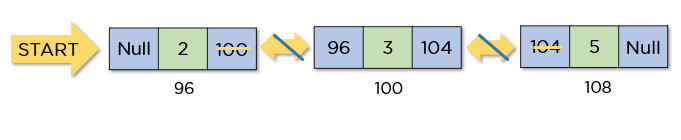
return 0;

}

**DELETION**

* At the beginning
* At the last
* At a specific position

**How do you remove a node from a doubly linked list?**

****

You need to change the previous node's next to the deleted node's next and the next node's previous to the deleted node's previous to remove a node. You can delete a node from three different positions.

* From the beginning of the list
* From the end of the list
* After a given node

Code:

// A C++code to perform all deletion operations on Linked List\*/

#include <bits/stdc++.h>

using namespace std;

/\* A class to create nodes \*/

class Node

{

public:

int data;

Node\* next;

Node\* prev;

};

/\*A Function to delete a node in a Linked List.\*/

void deleteNode(Node\*\* head, Node\* del)

{

/\* base case \*/

if (\*head == NULL || del == NULL)

return;

/\* If head node is the node to be deleted \*/

if (\*head == del)

\*head = del->next;

/\* Change next only if node to be

deleted is NOT the last node \*/

if (del->next != NULL)

del->next->prev = del->prev;

/\* Change prev only if node to be

deleted is NOT the first node \*/

if (del->prev != NULL)

del->prev->next = del->next;

/\* Finally, free the memory occupied by del\*/

free(del);

return;

}

/\* A function to insert a node at the beginning of the list\*/

void push(Node\*\* head, int newdata)

{

/\* create newnode \*/

Node\* newnode = new Node();

/\* put in the data \*/

newnode->data = newdata;

/\* link the new node's next to head

and previous as NULL \*/

newnode->next = (\*head);

newnode->prev = NULL;

/\* link the head node's prev to new node \*/

if ((\*head) != NULL)

(\*head)->prev = newnode;

/\* changing head \*/

(\*head) = newnode;

}

/\* Function to print nodes in a given linked list

This function is the same as printList() of singly linked list \*/

void printList(Node\* node)

{

while (node != NULL)

{

cout << node->data << " ";

node = node->next;

}

}

int main()

{

/\* Start with the empty list \*/

Node\* head = NULL;

/\* Let us create the linked list 2<->3<->5<->7 \*/

push(&head, 7);

push(&head, 5);

push(&head, 3);

push(&head, 2);

cout << "Original Linked list ";

printList(head);

/\* delete nodes from the linked list \*/

deleteNode(&head, head); /\*delete first node\*/

deleteNode(&head, head->next); /\*delete middle node\*/

deleteNode(&head, head->next); /\*delete last node\*/

/\* Modified linked list will be NULL<-5->NULL \*/

cout << "\nModified Linked list ";

printList(head);

return 0;

}

**What are the benefits of doubly linked list?**

* It is easy to reverse this linked list.
* It is easier to delete a node from this linked list as compared to a singly linked list.
* During its execution, it can easily assign or reassign memory.
* Reverse traversal is faster in this linked list.
* You can implement complex data structures like stacks and binary trees.

**What are the limitations of a doubly linked list?**

* It requires more space for each node because these nodes have an extra pointer.
* Its insertion and deletion operations are slower than singly-linked lists as it requires more steps.
* Because of random storage in memory, elements need to be accessed sequentially.

**CONCLUSION**

* Doubly linked list is a dynamic data structure
* Doubly linked list are faster at insertion and deletion
* Doubly linked list is a bidirectional linked list i.e. , **They can be done in both** **directions**