**Types of Linked List**

# Types of Linked List

* **Difficulty Level :** [Easy](https://www.geeksforgeeks.org/easy/)
* **Last Updated :** 02 Sep, 2022

<https://www.geeksforgeeks.org/types-of-linked-list/?ref=lbp>

 Read

 Discuss

A [linked list](https://www.geeksforgeeks.org/data-structures/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](https://www.geeksforgeeks.org/pointers-in-c-and-c-set-1-introduction-arithmetic-and-array/). In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.

## Types Of Linked List:

### ****1.**** [****Singly Linked List****](https://www.geeksforgeeks.org/data-structures/linked-list/singly-linked-list/)

It is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type.

The node contains a pointer to the next node means that the node stores the address of the next node in the sequence. A single linked list allows the traversal of data only in one way. Below is the image for the same:

Below is the structure of the singly linked list

|  |
| --- |
| // Node of a doubly linked list  static class Node {      int data;        // Pointer to next node in LL      Node next;  };    // this code is contributed by shivani |

#### ****Creation and Traversal of Singly Linked List:****

|  |
| --- |
| // Java program to illustrate  // creation and traversal of  // Singly Linked List  class GFG {        // Structure of Node      static class Node {          int data;          Node next;      };        // Function to print the content of      // linked list starting from the      // given node      static void printList(Node n)      {          // Iterate till n reaches null          while (n != null) {              // Print the data              System.out.print(n.data + " ");              n = n.next;          }      }        // Driver Code      public static void main(String[] args)      {          Node head = null;          Node second = null;          Node third = null;            // Allocate 3 nodes in          // the heap          head = new Node();          second = new Node();          third = new Node();            // Assign data in first          // node          head.data = 1;            // Link first node with          // second          head.next = second;            // Assign data to second          // node          second.data = 2;          second.next = third;            // Assign data to third          // node          third.data = 3;          third.next = null;            printList(head);      }  }    // This code is contributed by Princi Singh |

**Output**

1 2 3

**Time Complexity:** O(N)  
**Auxiliary Space:** O(N)

### ****2.**** [****Doubly Linked List****](https://www.geeksforgeeks.org/doubly-linked-list/)

A doubly linked list or a two-way linked list is a more complex type of linked list that contains a pointer to the next as well as the previous node in sequence.

Therefore, it contains three parts of data, a pointer to the next node, and a pointer to the previous node. This would enable us to traverse the list in the backward direction as well. Below is the image for the same:

#### ****Structure of Doubly Linked List:****

|  |
| --- |
| // Doubly linked list  // node  static class Node {      int data;        // Pointer to next node in DLL      Node next;        // Pointer to the previous node in DLL      Node prev;  };    // This code is contributed by shivani |

#### ****Creation and Traversal of Doubly Linked List:****

|  |
| --- |
| // Java program to illustrate  // creation and traversal of  // Doubly Linked List    import java.util.\*;  class GFG {        // Doubly linked list      // node      static class Node {          int data;          Node next;          Node prev;      };        static Node head\_ref;        // Function to push a new      // element in the Doubly      // Linked List      static void push(int new\_data)      {          // Allocate node          Node new\_node = new Node();            // Put in the data          new\_node.data = new\_data;            // Make next of new node as          // head and previous as null          new\_node.next = head\_ref;          new\_node.prev = null;            // Change prev of head node to          // the new node          if (head\_ref != null)              head\_ref.prev = new\_node;            // Move the head to point to          // the new node          head\_ref = new\_node;      }        // Function to traverse the      // Doubly LL in the forward      // & backward direction      static void printList(Node node)      {          Node last = null;            System.out.print("\nTraversal in forward"                           + " direction \n");          while (node != null) {              // Print the data              System.out.print(" " + node.data + " ");              last = node;              node = node.next;          }            System.out.print("\nTraversal in reverse"                           + " direction \n");            while (last != null) {              // Print the data              System.out.print(" " + last.data + " ");              last = last.prev;          }      }        // Driver Code      public static void main(String[] args)      {          // Start with the empty list          head\_ref = null;            // Insert 6.          // So linked list becomes          // 6.null          push(6);            // Insert 7 at the beginning.          // So linked list becomes          // 7.6.null          push(7);            // Insert 1 at the beginning.          // So linked list becomes          // 1.7.6.null          push(1);            System.out.print("Created DLL is: ");          printList(head\_ref);      }  }    // This code is contributed by Princi Singh |

**Output**

Created DLL is:

Traversal in forward direction

1 7 6

Traversal in reverse direction

6 7 1

### ****3.**** [****Circular Linked List****](https://www.geeksforgeeks.org/circular-linked-list/)

A circular linked list is that in which the last node contains the pointer to the first node of the list.

While traversing a circular linked list, we can begin at any node and traverse the list in any direction forward and backward until we reach the same node we started. Thus, a circular linked list has no beginning and no end. Below is the image for the same:

Below is the structure of the Circular Linked List:

|  |
| --- |
| // Structure for a node  static class Node {      int data;        // Pointer to next node in CLL      Node next;  };    // This code is contributed by shivanisinghss2110 |

#### ****Creation and Traversal of Circular Linked List:****

|  |
| --- |
| // Java program to illustrate  // creation and traversal of  // Circular LL    import java.util.\*;  class GFG {        // Structure for a      // node      static class Node {          int data;          Node next;      };        // Function to insert a node      // at the beginning of Circular      // LL      static Node push(Node head\_ref, int data)      {          Node ptr1 = new Node();          Node temp = head\_ref;          ptr1.data = data;          ptr1.next = head\_ref;            // If linked list is not          // null then set the next          // of last node          if (head\_ref != null) {              while (temp.next != head\_ref) {                  temp = temp.next;              }              temp.next = ptr1;          }            // For the first node          else              ptr1.next = ptr1;            head\_ref = ptr1;          return head\_ref;      }        // Function to print nodes in      // the Circular Linked List      static void printList(Node head)      {          Node temp = head;          if (head != null) {              do {                  // Print the data                  System.out.print(temp.data + " ");                  temp = temp.next;              } while (temp != head);          }      }        // Driver Code      public static void main(String[] args)      {          // Initialize list as empty          Node head = null;            // Created linked list will          // be 11.2.56.12          head = push(head, 12);          head = push(head, 56);          head = push(head, 2);          head = push(head, 11);            System.out.print("Contents of Circular"                           + " Linked List\n ");            // Function call          printList(head);      }  }    // This code is contributed by gauravrajput1 |

**Output**

Contents of Circular Linked List

11 2 56 12

### 4. [Doubly Circular linked list](https://www.geeksforgeeks.org/doubly-circular-linked-list-set-1-introduction-and-insertion/)

A Doubly Circular linked list or a circular two-way linked list is a more complex type of linked list that contains a pointer to the next as well as the previous node in the sequence. The difference between the doubly linked and circular doubly list is the same as that between a singly linked list and a circular linked list. The circular doubly linked list does not contain null in the previous field of the first node. Below is the image for the same:

Below is the structure of the Doubly Circular Linked List:

|  |
| --- |
| // Structure of a Node  static class Node  {      int data;        // Pointer to next node in DCLL      Node next;        // Pointer to the previous node in DCLL      Node prev;  };    //this code is contributed by shivanisinghss2110 |

#### ****Creation and Traversal of Doubly Circular Linked List:****

|  |
| --- |
| // C++ program to illustrate creation  // & traversal of Doubly Circular LL    #include <bits/stdc++.h>  using namespace std;    // Structure of a Node  struct Node {      int data;      struct Node\* next;      struct Node\* prev;  };    // Function to insert Node at  // the beginning of the List  void insertBegin(struct Node\*\* start, int value)  {      // If the list is empty      if (\*start == NULL) {          struct Node\* new\_node = new Node;          new\_node->data = value;          new\_node->next = new\_node->prev = new\_node;          \*start = new\_node;          return;      }        // Pointer points to last Node      struct Node\* last = (\*start)->prev;        struct Node\* new\_node = new Node;        // Inserting the data      new\_node->data = value;        // Update the previous and      // next of new node      new\_node->next = \*start;      new\_node->prev = last;        // Update next and previous      // pointers of start & last      last->next = (\*start)->prev = new\_node;        // Update start pointer      \*start = new\_node;  }    // Function to traverse the circular  // doubly linked list  void display(struct Node\* start)  {      struct Node\* temp = start;        printf("\nTraversal in"             " forward direction \n");      while (temp->next != start) {          printf("%d ", temp->data);          temp = temp->next;      }      printf("%d ", temp->data);        printf("\nTraversal in "             "reverse direction \n");      Node\* last = start->prev;      temp = last;        while (temp->prev != last) {            // Print the data          printf("%d ", temp->data);          temp = temp->prev;      }      printf("%d ", temp->data);  }    // Driver Code  int main()  {      // Start with the empty list      struct Node\* start = NULL;        // Insert 5      // So linked list becomes 5->NULL      insertBegin(&start, 5);        // Insert 4 at the beginning      // So linked list becomes 4->5      insertBegin(&start, 4);        // Insert 7 at the end      // So linked list becomes 7->4->5      insertBegin(&start, 7);        printf("Created circular doubly"             " linked list is: ");      display(start);        return 0;  } |

**Output**

Created circular doubly linked list is:

Traversal in forward direction

7 4 5

Traversal in reverse direction

5 4 7

### 5. [Header Linked List:](https://www.geeksforgeeks.org/header-linked-list-in-c/)

A header linked list is a special type of linked list that contains a header node at the beginning of the list.

So, in a header linked list **START** will not point to the first node of the list but **START** will contain the address of the header node. Below is the image for Grounded Header Linked List:

Below is the Structure of the Grounded Header Linked List:

|  |
| --- |
| // Structure of the list  struct link {      int info;        // Pointer to the next node      struct link\* next;  }; |

#### Creation and Traversal of Header Linked List:

|  |
| --- |
| // Java program to illustrate creation  // and traversal of Header Linked List    class GFG {      // Structure of the list      static class link {          int info;          link next;      };        // Empty List      static link start = null;        // Function to create header of the      // header linked list      static link create\_header\_list(int data)      {            // Create a new node          link new\_node, node;          new\_node = new link();          new\_node.info = data;          new\_node.next = null;            // If it is the first node          if (start == null) {                // Initialize the start              start = new link();              start.next = new\_node;          }          else {                // Insert the node in the end              node = start;              while (node.next != null) {                  node = node.next;              }              node.next = new\_node;          }          return start;      }        // Function to display the      // header linked list      static link display()      {          link node;          node = start;          node = node.next;            // Traverse until node is          // not null          while (node != null) {                // Print the data              System.out.printf("%d ", node.info);              node = node.next;          }          System.out.printf("\n");            // Return the start pointer          return start;      }        // Driver Code      public static void main(String[] args)      {          // Create the list          create\_header\_list(11);          create\_header\_list(12);          create\_header\_list(13);            // Print the list          System.out.printf("List After inserting"                            + " 3 elements:\n");          display();          create\_header\_list(14);          create\_header\_list(15);            // Print the list          System.out.printf("List After inserting"                            + " 2 more elements:\n");          display();      }  }    // This code is contributed by 29AjayKumar |

**Output**

List After inserting 3 elements:

11 12 13

List After inserting 2 more elements:

11 12 13 14 15

# Types of Linked List

* **Difficulty Level :** [Easy](https://www.geeksforgeeks.org/easy/)
* **Last Updated :** 02 Sep, 2022

 Read

 Discuss

A [linked list](https://www.geeksforgeeks.org/data-structures/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](https://www.geeksforgeeks.org/pointers-in-c-and-c-set-1-introduction-arithmetic-and-array/). In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.

## Types Of Linked List:

### ****1.**** [****Singly Linked List****](https://www.geeksforgeeks.org/data-structures/linked-list/singly-linked-list/)

It is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type.

The node contains a pointer to the next node means that the node stores the address of the next node in the sequence. A single linked list allows the traversal of data only in one way. Below is the image for the same:

Below is the structure of the singly linked list

|  |
| --- |
| // Node of a doubly linked list  static class Node {      int data;        // Pointer to next node in LL      Node next;  };    // this code is contributed by shivani |

#### ****Creation and Traversal of Singly Linked List:****

|  |
| --- |
| // Java program to illustrate  // creation and traversal of  // Singly Linked List  class GFG {        // Structure of Node      static class Node {          int data;          Node next;      };        // Function to print the content of      // linked list starting from the      // given node      static void printList(Node n)      {          // Iterate till n reaches null          while (n != null) {              // Print the data              System.out.print(n.data + " ");              n = n.next;          }      }        // Driver Code      public static void main(String[] args)      {          Node head = null;          Node second = null;          Node third = null;            // Allocate 3 nodes in          // the heap          head = new Node();          second = new Node();          third = new Node();            // Assign data in first          // node          head.data = 1;            // Link first node with          // second          head.next = second;            // Assign data to second          // node          second.data = 2;          second.next = third;            // Assign data to third          // node          third.data = 3;          third.next = null;            printList(head);      }  }    // This code is contributed by Princi Singh |

**Output**

1 2 3

**Time Complexity:** O(N)  
**Auxiliary Space:** O(N)

### ****2.**** [****Doubly Linked List****](https://www.geeksforgeeks.org/doubly-linked-list/)

A doubly linked list or a two-way linked list is a more complex type of linked list that contains a pointer to the next as well as the previous node in sequence.

Therefore, it contains three parts of data, a pointer to the next node, and a pointer to the previous node. This would enable us to traverse the list in the backward direction as well. Below is the image for the same:

#### ****Structure of Doubly Linked List:****

|  |
| --- |
| // Doubly linked list  // node  static class Node {      int data;        // Pointer to next node in DLL      Node next;        // Pointer to the previous node in DLL      Node prev;  };    // This code is contributed by shivani |

#### ****Creation and Traversal of Doubly Linked List:****

|  |
| --- |
| // Java program to illustrate  // creation and traversal of  // Doubly Linked List    import java.util.\*;  class GFG {        // Doubly linked list      // node      static class Node {          int data;          Node next;          Node prev;      };        static Node head\_ref;        // Function to push a new      // element in the Doubly      // Linked List      static void push(int new\_data)      {          // Allocate node          Node new\_node = new Node();            // Put in the data          new\_node.data = new\_data;            // Make next of new node as          // head and previous as null          new\_node.next = head\_ref;          new\_node.prev = null;            // Change prev of head node to          // the new node          if (head\_ref != null)              head\_ref.prev = new\_node;            // Move the head to point to          // the new node          head\_ref = new\_node;      }        // Function to traverse the      // Doubly LL in the forward      // & backward direction      static void printList(Node node)      {          Node last = null;            System.out.print("\nTraversal in forward"                           + " direction \n");          while (node != null) {              // Print the data              System.out.print(" " + node.data + " ");              last = node;              node = node.next;          }            System.out.print("\nTraversal in reverse"                           + " direction \n");            while (last != null) {              // Print the data              System.out.print(" " + last.data + " ");              last = last.prev;          }      }        // Driver Code      public static void main(String[] args)      {          // Start with the empty list          head\_ref = null;            // Insert 6.          // So linked list becomes          // 6.null          push(6);            // Insert 7 at the beginning.          // So linked list becomes          // 7.6.null          push(7);            // Insert 1 at the beginning.          // So linked list becomes          // 1.7.6.null          push(1);            System.out.print("Created DLL is: ");          printList(head\_ref);      }  }    // This code is contributed by Princi Singh |

**Output**

Created DLL is:

Traversal in forward direction

1 7 6

Traversal in reverse direction

6 7 1

### ****3.**** [****Circular Linked List****](https://www.geeksforgeeks.org/circular-linked-list/)

A circular linked list is that in which the last node contains the pointer to the first node of the list.

While traversing a circular linked list, we can begin at any node and traverse the list in any direction forward and backward until we reach the same node we started. Thus, a circular linked list has no beginning and no end. Below is the image for the same:

Below is the structure of the Circular Linked List:

|  |
| --- |
| // Structure for a node  static class Node {      int data;        // Pointer to next node in CLL      Node next;  };    // This code is contributed by shivanisinghss2110 |

#### ****Creation and Traversal of Circular Linked List:****

|  |
| --- |
| // Java program to illustrate  // creation and traversal of  // Circular LL    import java.util.\*;  class GFG {        // Structure for a      // node      static class Node {          int data;          Node next;      };        // Function to insert a node      // at the beginning of Circular      // LL      static Node push(Node head\_ref, int data)      {          Node ptr1 = new Node();          Node temp = head\_ref;          ptr1.data = data;          ptr1.next = head\_ref;            // If linked list is not          // null then set the next          // of last node          if (head\_ref != null) {              while (temp.next != head\_ref) {                  temp = temp.next;              }              temp.next = ptr1;          }            // For the first node          else              ptr1.next = ptr1;            head\_ref = ptr1;          return head\_ref;      }        // Function to print nodes in      // the Circular Linked List      static void printList(Node head)      {          Node temp = head;          if (head != null) {              do {                  // Print the data                  System.out.print(temp.data + " ");                  temp = temp.next;              } while (temp != head);          }      }        // Driver Code      public static void main(String[] args)      {          // Initialize list as empty          Node head = null;            // Created linked list will          // be 11.2.56.12          head = push(head, 12);          head = push(head, 56);          head = push(head, 2);          head = push(head, 11);            System.out.print("Contents of Circular"                           + " Linked List\n ");            // Function call          printList(head);      }  }    // This code is contributed by gauravrajput1 |

**Output**

Contents of Circular Linked List

11 2 56 12

### 4. [Doubly Circular linked list](https://www.geeksforgeeks.org/doubly-circular-linked-list-set-1-introduction-and-insertion/)

A Doubly Circular linked list or a circular two-way linked list is a more complex type of linked list that contains a pointer to the next as well as the previous node in the sequence. The difference between the doubly linked and circular doubly list is the same as that between a singly linked list and a circular linked list. The circular doubly linked list does not contain null in the previous field of the first node. Below is the image for the same:

Below is the structure of the Doubly Circular Linked List:

|  |
| --- |
| // Structure of a Node  static class Node  {      int data;        // Pointer to next node in DCLL      Node next;        // Pointer to the previous node in DCLL      Node prev;  };    //this code is contributed by shivanisinghss2110 |

#### ****Creation and Traversal of Doubly Circular Linked List:****

|  |
| --- |
| // C++ program to illustrate creation  // & traversal of Doubly Circular LL    #include <bits/stdc++.h>  using namespace std;    // Structure of a Node  struct Node {      int data;      struct Node\* next;      struct Node\* prev;  };    // Function to insert Node at  // the beginning of the List  void insertBegin(struct Node\*\* start, int value)  {      // If the list is empty      if (\*start == NULL) {          struct Node\* new\_node = new Node;          new\_node->data = value;          new\_node->next = new\_node->prev = new\_node;          \*start = new\_node;          return;      }        // Pointer points to last Node      struct Node\* last = (\*start)->prev;        struct Node\* new\_node = new Node;        // Inserting the data      new\_node->data = value;        // Update the previous and      // next of new node      new\_node->next = \*start;      new\_node->prev = last;        // Update next and previous      // pointers of start & last      last->next = (\*start)->prev = new\_node;        // Update start pointer      \*start = new\_node;  }    // Function to traverse the circular  // doubly linked list  void display(struct Node\* start)  {      struct Node\* temp = start;        printf("\nTraversal in"             " forward direction \n");      while (temp->next != start) {          printf("%d ", temp->data);          temp = temp->next;      }      printf("%d ", temp->data);        printf("\nTraversal in "             "reverse direction \n");      Node\* last = start->prev;      temp = last;        while (temp->prev != last) {            // Print the data          printf("%d ", temp->data);          temp = temp->prev;      }      printf("%d ", temp->data);  }    // Driver Code  int main()  {      // Start with the empty list      struct Node\* start = NULL;        // Insert 5      // So linked list becomes 5->NULL      insertBegin(&start, 5);        // Insert 4 at the beginning      // So linked list becomes 4->5      insertBegin(&start, 4);        // Insert 7 at the end      // So linked list becomes 7->4->5      insertBegin(&start, 7);        printf("Created circular doubly"             " linked list is: ");      display(start);        return 0;  } |

**Output**

Created circular doubly linked list is:

Traversal in forward direction

7 4 5

Traversal in reverse direction

5 4 7

### 5. [Header Linked List:](https://www.geeksforgeeks.org/header-linked-list-in-c/)

A header linked list is a special type of linked list that contains a header node at the beginning of the list.

So, in a header linked list **START** will not point to the first node of the list but **START** will contain the address of the header node. Below is the image for Grounded Header Linked List:

Below is the Structure of the Grounded Header Linked List:

|  |
| --- |
| // Structure of the list  struct link {      int info;        // Pointer to the next node      struct link\* next;  }; |

#### Creation and Traversal of Header Linked List:

|  |
| --- |
| // Java program to illustrate creation  // and traversal of Header Linked List    class GFG {      // Structure of the list      static class link {          int info;          link next;      };        // Empty List      static link start = null;        // Function to create header of the      // header linked list      static link create\_header\_list(int data)      {            // Create a new node          link new\_node, node;          new\_node = new link();          new\_node.info = data;          new\_node.next = null;            // If it is the first node          if (start == null) {                // Initialize the start              start = new link();              start.next = new\_node;          }          else {                // Insert the node in the end              node = start;              while (node.next != null) {                  node = node.next;              }              node.next = new\_node;          }          return start;      }        // Function to display the      // header linked list      static link display()      {          link node;          node = start;          node = node.next;            // Traverse until node is          // not null          while (node != null) {                // Print the data              System.out.printf("%d ", node.info);              node = node.next;          }          System.out.printf("\n");            // Return the start pointer          return start;      }        // Driver Code      public static void main(String[] args)      {          // Create the list          create\_header\_list(11);          create\_header\_list(12);          create\_header\_list(13);            // Print the list          System.out.printf("List After inserting"                            + " 3 elements:\n");          display();          create\_header\_list(14);          create\_header\_list(15);            // Print the list          System.out.printf("List After inserting"                            + " 2 more elements:\n");          display();      }  }    // This code is contributed by 29AjayKumar |

**Output**

List After inserting 3 elements:

11 12 13

List After inserting 2 more elements:

11 12 13 14 15