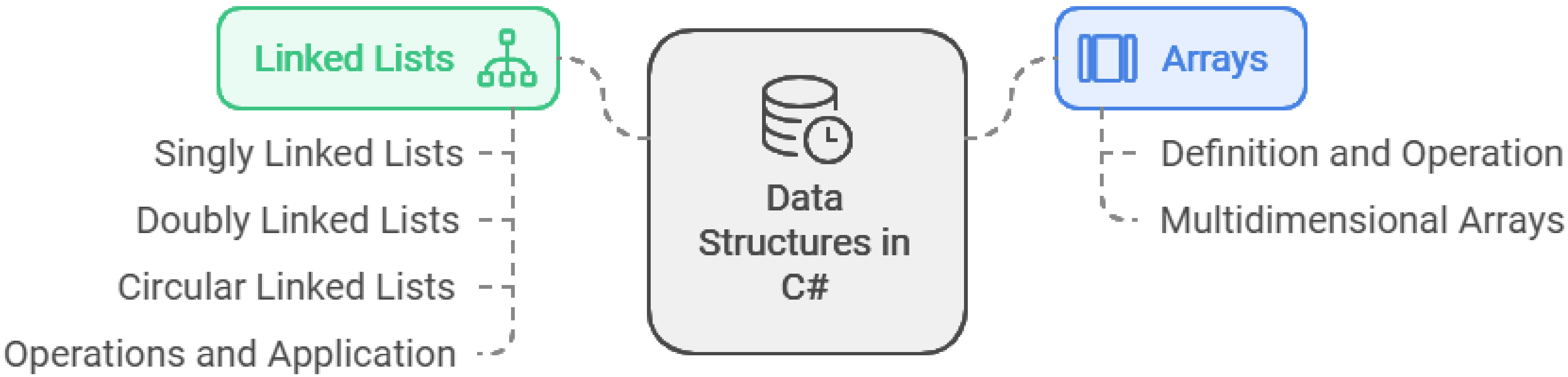
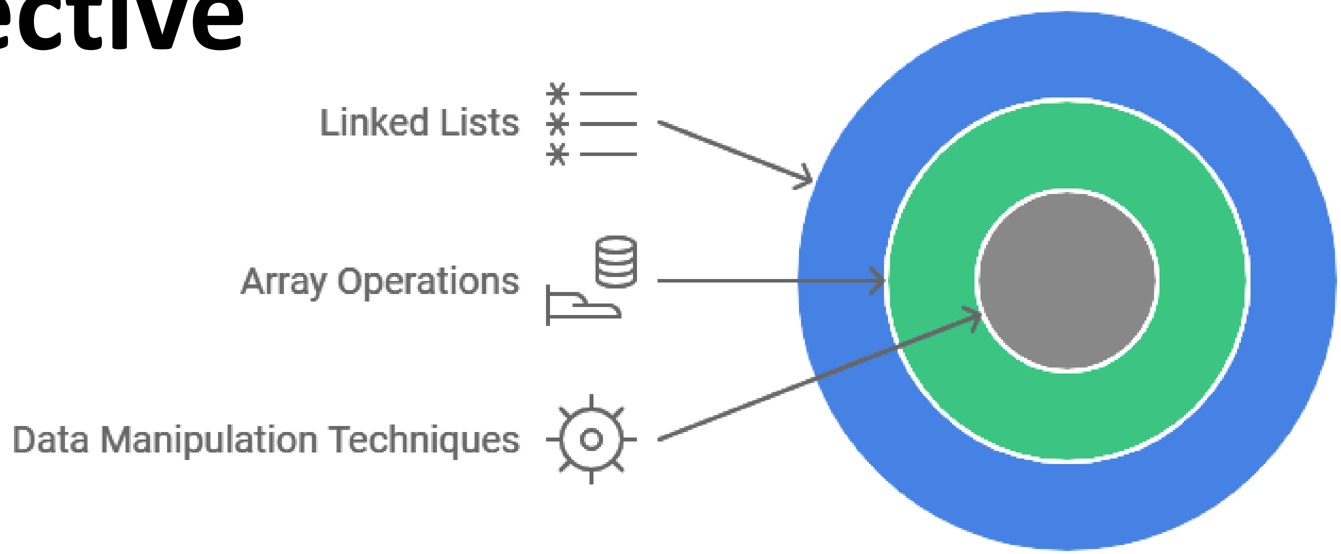
Module 2

Array and Linked List

Content



Objective



Part 1: Array Structure

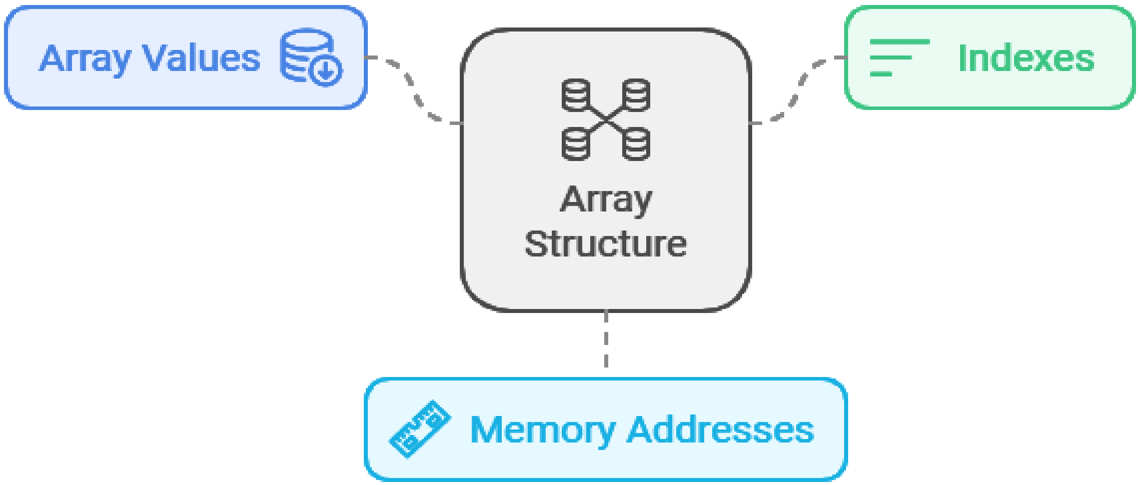
I. Definition and Operation

ACLEDA UNIVERSITY

1. Definition

int x [5] = {25, 35, 40, 55, 60};

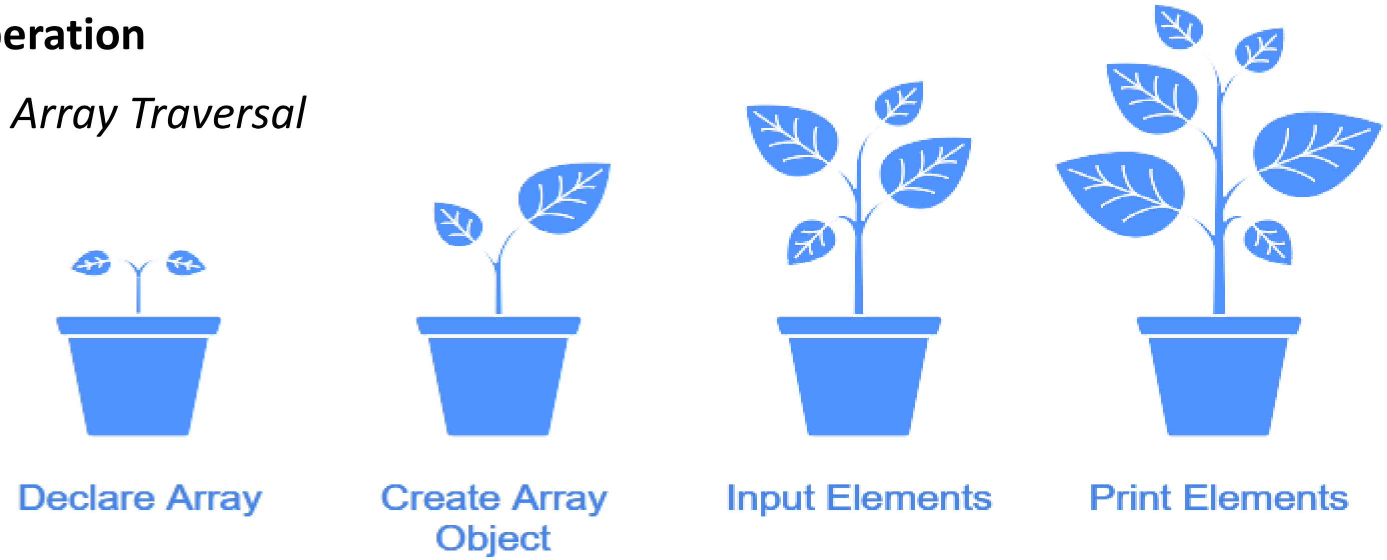
Array Value	25	35	40	55	60
Array Indexes	0	1	2	3	4
Memory Address	1000	1004	1008	1012	1016



I. Definition and Operation

2. Operation

a) *Array Traversal*




```
static public void Main()
{       int n = 10;
        int[] arr = new int[n];
        // initial array of size 10
        for (int i = 0; i < n; i++)
            arr[i] = i + 1;
        // print the original array
        for (int i = 0; i < n; i++)
            Console.Write(arr[i] + " ");
        Console.WriteLine();
}
```

Output:

1 2 3 4 5 6 7 8 9 10

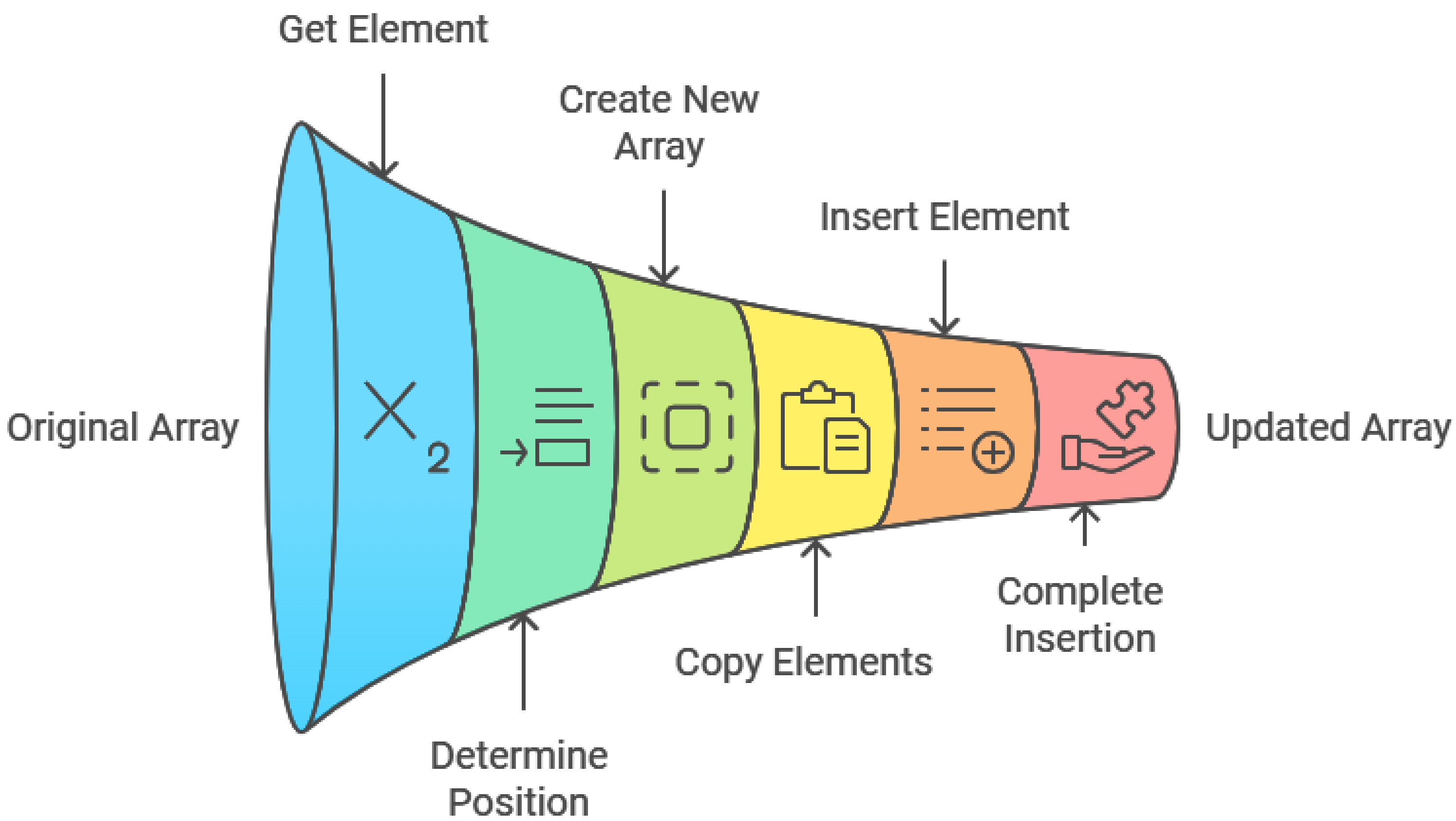
I. Definition and Operation

```
static void Main(string[] args)
{
        // declares an Array of integers.
        int[] intArray;
        // allocating memory for 5
        intArray = new int[5];
        // initialize the first elements
        intArray[0] = 1;
        intArray[1] = 2;
        intArray[2] = 3;
        intArray[3] = 4;
        intArray[4] = 5;
       // accessing the elements using for loop
        Console.Write("For loop :");
        for (int i = 0; i < intArray.Length; i++)
                Console.Write(" " + intArray[i]);
        Console.Read();
}
```

Output: 1 2 3 4 5

2. Operation

b) Array Insertion



I. Definition and Operation

អេស៊ីលីដា ACLEDA UNIVERSITY

2. Operation

b) Array Insertion

```
static void Main(string[] args)
{
        int n = 10, x = 50, pos = 4;
        int[] arr = new int[n+1];
        // Enter array element 1 to 10
        for (int i = 0; i < n; i++)
            arr[i] = i + 1;
        Console.Write("Before Insertion: ");
        for (int i = 0; i < n; i++)
            Console.Write(arr[i] + " ");
        //Insert 50 into position 4
        for (int i = n-1; i >= pos; i--)
            arr[i+1] = arr[i];
        arr[pos] = x;
        n += 1;
        Console.Write("\nAfter Insertion: ");
        for (int i = 0; i < n; i++)
            Console.Write(arr[i] + " ");
        Console.Read();
}
```

Output:

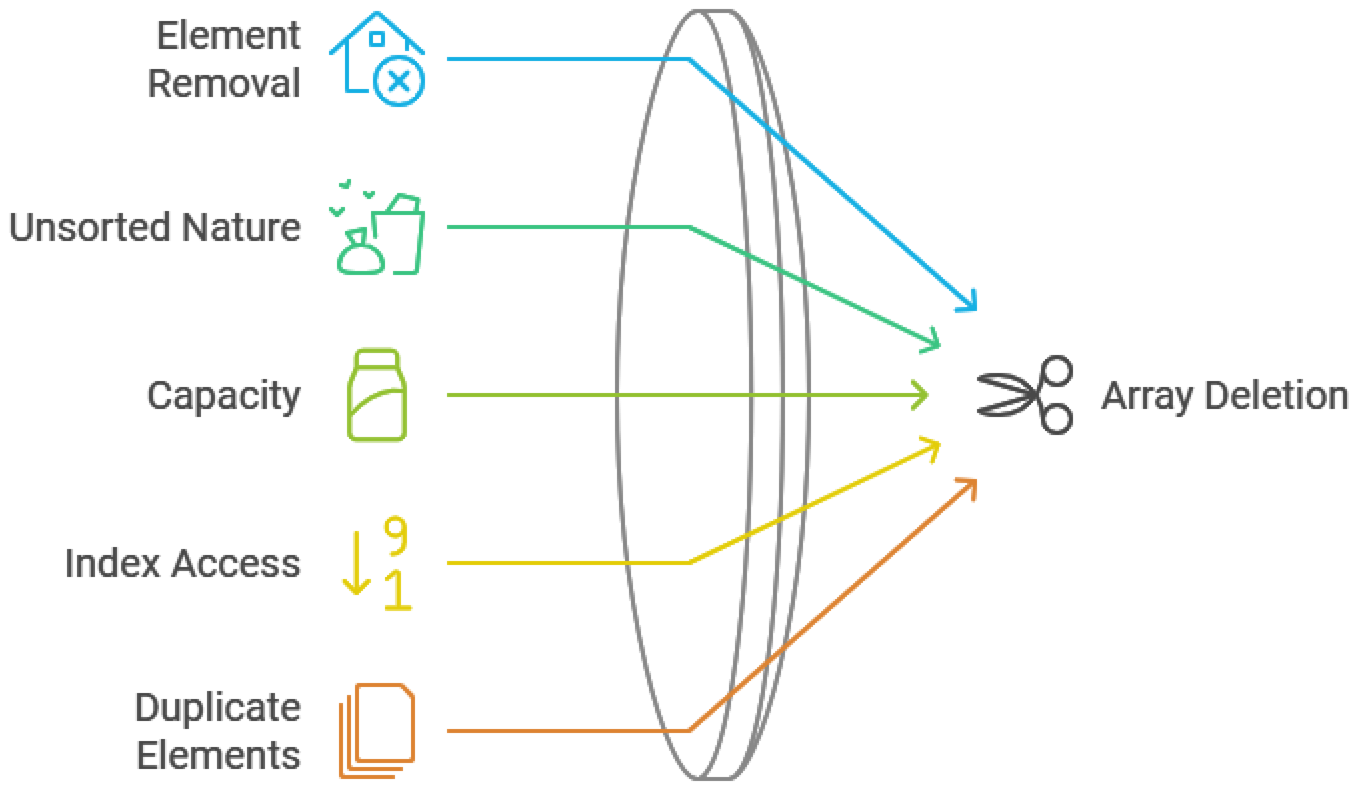
1 2 3 4 5 6 7 8 9 10

1 2 3 4 50 5 6 7 8 9 10

I. Definition and Operation

2. Operation

c) Array Deletion



I. Definition and Operation

អេស៊ីលីដា ACLEDA UNIVERSITY

2. Operation

c) Array Deletion

```
static void Main(string[] args)
{       int pos=0, key = 30;
        int[] array = { 10, 20, 30, 40, 50 };
        int n = array.Length;
        Console.Write("Array before deletion ");
        for (int i = 0; i < n; i++)
            Console.Write(array[i] + " ");
        Console.WriteLine();
        for (int i = 0; i < n; i++)// Find position of element
            if (array[i] == key)
            {       pos = i;  break;   }
            else{   pos = -1;          }
        Console.Write($"Position to delete {pos} ");
        for (int i = pos; i < n - 1; i++) // Deleting element
                array[i] = array[i + 1];
        Console.Write("\nArray after deletion ");
        for (int i = 0; i < n-1; i++)
                Console.Write(array[i] + " ");
        Console.Read();
}
```

Output:

Array before deletion

10 20 30 40 50

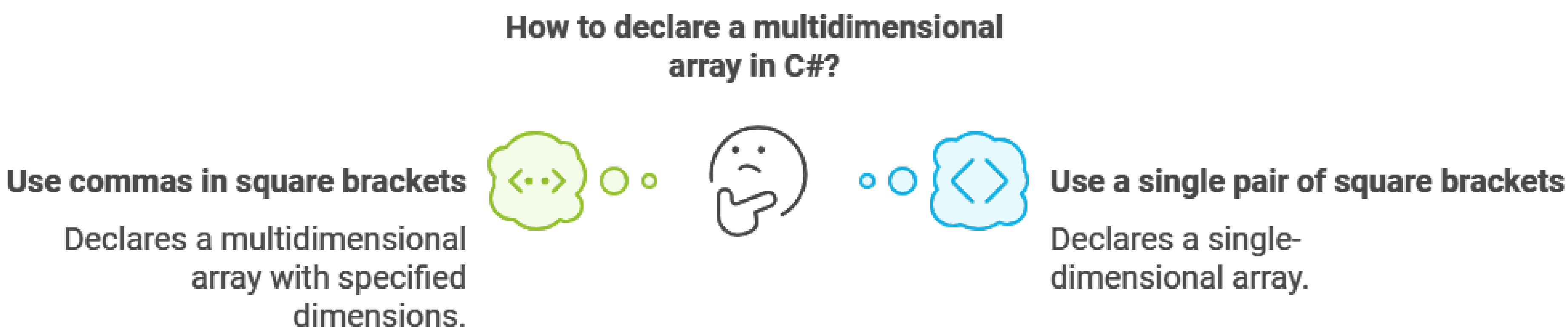
Position to delete 2

Array after deletion

10 20 40 50

II. Multidimensional Arrays

1. 2D Array



// Declare two dimensional array

- int[row, col] arr2d; // two-dimensional array
- int[row, col, dep] arr3d; // three-dimensional array

//initialize array 2D elements

- int[,] arr2d = new int[3,2]{ {1, 2}, {3, 4}, {5, 6} };
- int[, ,] arr3d = new int[2, 2, 3]{ { 1, 2, 3}, {4, 5, 6} },
  { { 7, 8, 9}, {10, 11, 12} }
  };

II. Multidimensional Arrays

```
int[,] numbers = { {1, 4, 2}, {3, 6, 8} };
for (int i = 0; i < numbers.GetLength(0); i++)
{
    for (int j = 0; j < numbers.GetLength(1); j++)
    {
        Console.WriteLine(numbers[i, j]);
    }
}
```

Output: 1 4 2

3 6 8

II. Multidimensional Arrays

2. 3D Array

```
static void Main()
{       int[,] numbers = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };

        // Accessing an element
        Console.WriteLine(numbers[0, 2]); // Outputs 3

        // Modifying an element
        numbers[1, 1] = 10;

        // Looping through the array
        for (int i = 0; i < numbers.GetLength(0); i++)
        {
            for (int j = 0; j < numbers.GetLength(1); j++)
            {
                Console.Write(numbers[i, j] + " ");
            }
         }
            Console.WriteLine();
}
```

Output:
```
3
1  2  3
4  10  6
7  8  9
```

II. Multidimensional Arrays

អេស៊ីលីដា ACLEDA UNIVERSITY

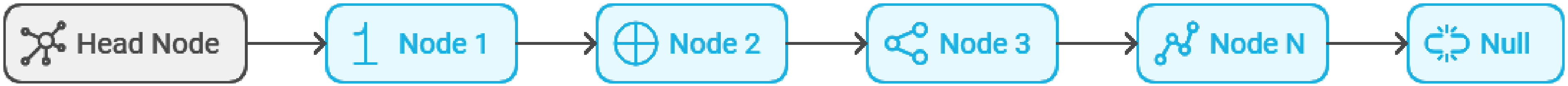
```
static void Main(string[] args)
{
        //Row Size: 3, Column Size: 4
        int[,] NumbersArray = {     {11,12,13,14},
                                    {21,22,23,24},
                                    {31,32,33,34}};

        //Printing Array Elements using for each loop
        Console.WriteLine("Printing Array Elements For Each");
        foreach (int i in NumbersArray)
        {     Console.Write(i + " ");
        }

        //Printing Array Elements using nested for each
        Console.WriteLine("Printing Array Elements For Loop");
        for (int i = 0; i < NumbersArray.GetLength(0); i++)
        {
           for (int j = 0; j < NumbersArray.GetLength(1); j++)
           {     Console.Write(NumbersArray[i, j] + " ");
           }
        }
        Console.ReadKey();
}
```

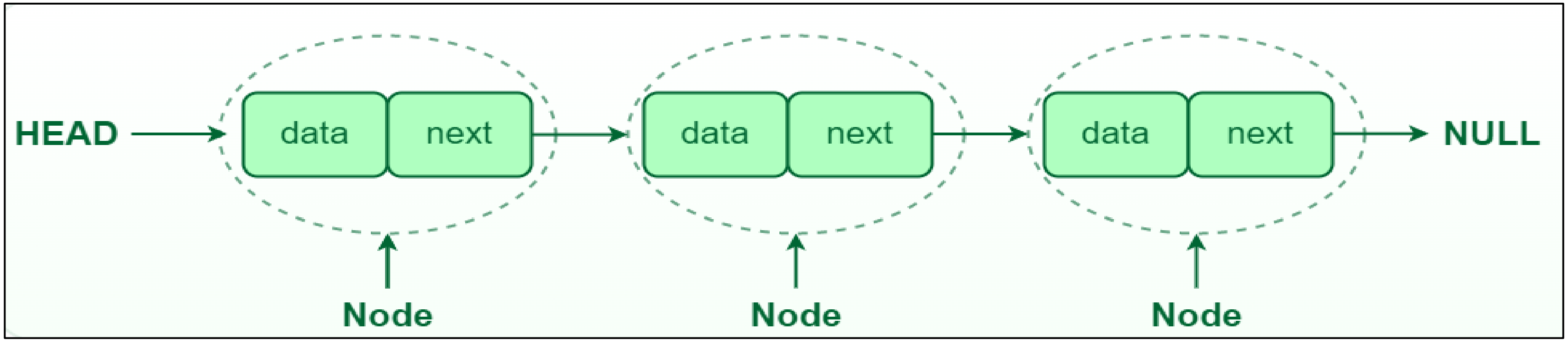
Output:
Printing Array Elements, ForEach
11 12 13 14 21 22 23 24 31 32 33 34
Printing Array Elements, For Loop
11 12 13 14 21 22 23 24 31 32 33 34

Part 2: Linked List

I. Definition and Operation

Definition



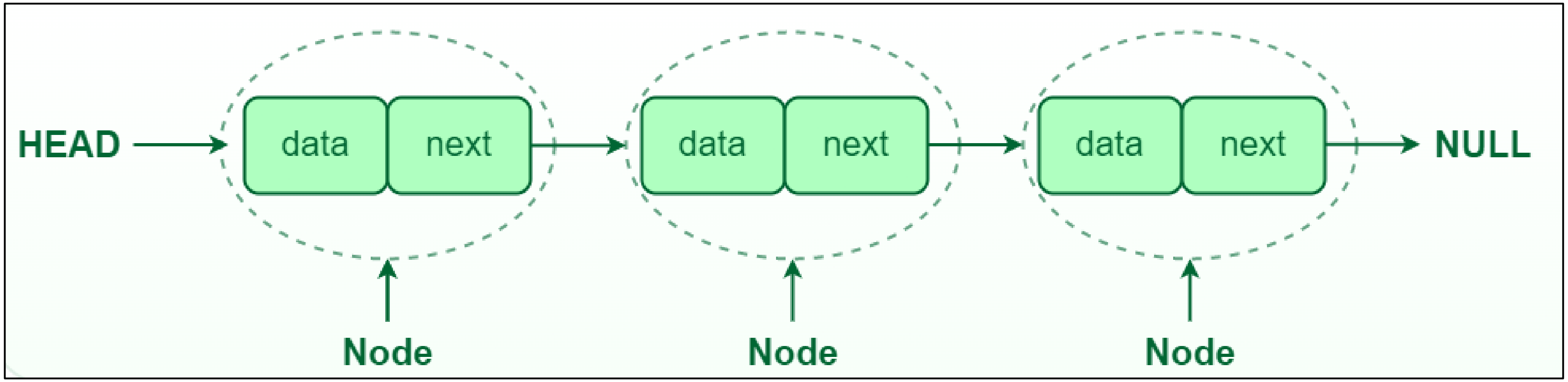


1. **Data:** It holds the actual value or data associated with the node.
2. **Next Pointer:** It stores the memory address of the next node in the sequence.

II. Singly Linked List

Definition

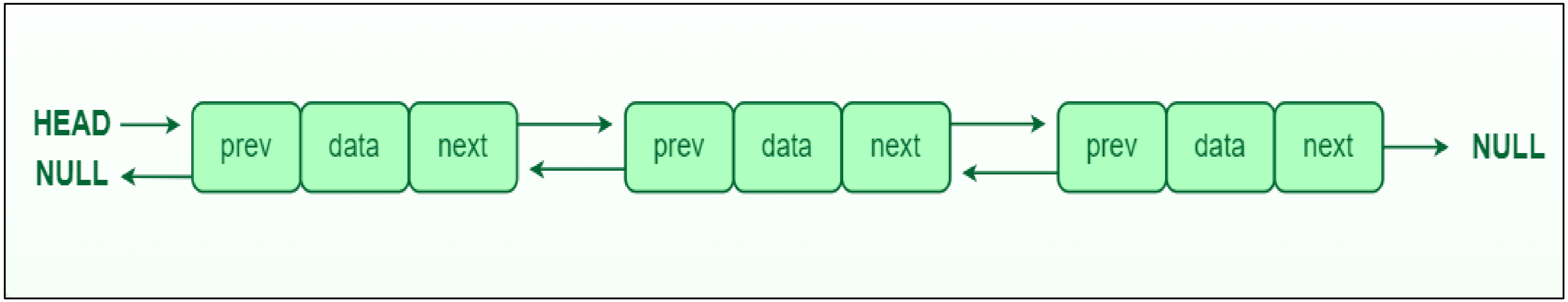
Singly linked list, each node contains a reference to the next node in the sequence. Traversing a singly linked list is done in a forward direction.



III. Doubly linked list

Definition

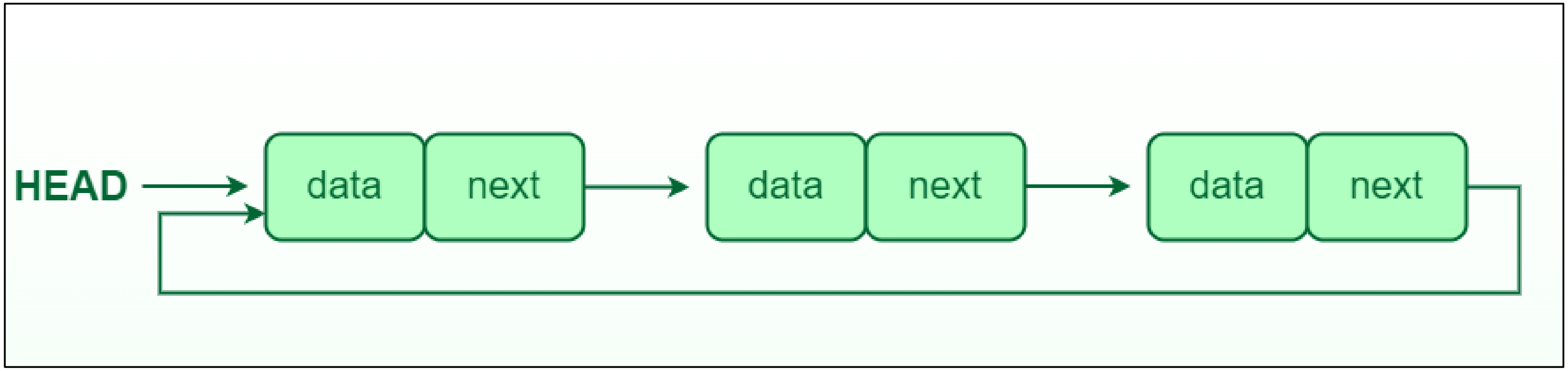
Doubly linked list, each node contains references to both the next and previous nodes. This allows for traversal in both forward and backward directions.



IV. Circular linked list

Definition

Circular linked list, the last node points back to the head node, creating a circular structure. It can be either singly or doubly linked.



V. Linked List Operations

1. Insertion Operation

Insertion: Adding a new node to a linked list involves adjusting the pointers of the existing nodes to maintain the proper sequence.

```
// C# Program to insert the node at the beginning
using System;
public class Node
{
    public int data;
    public Node next;
    public Node(int newData)
    {   data = newData;
        next = null;
    }
}
public class GFG
{
    public static Node InsertAtFront(Node head, int newData)
    {
        Node newNode = new Node(newData);
        newNode.next = head;
        return newNode;
     }
}
```

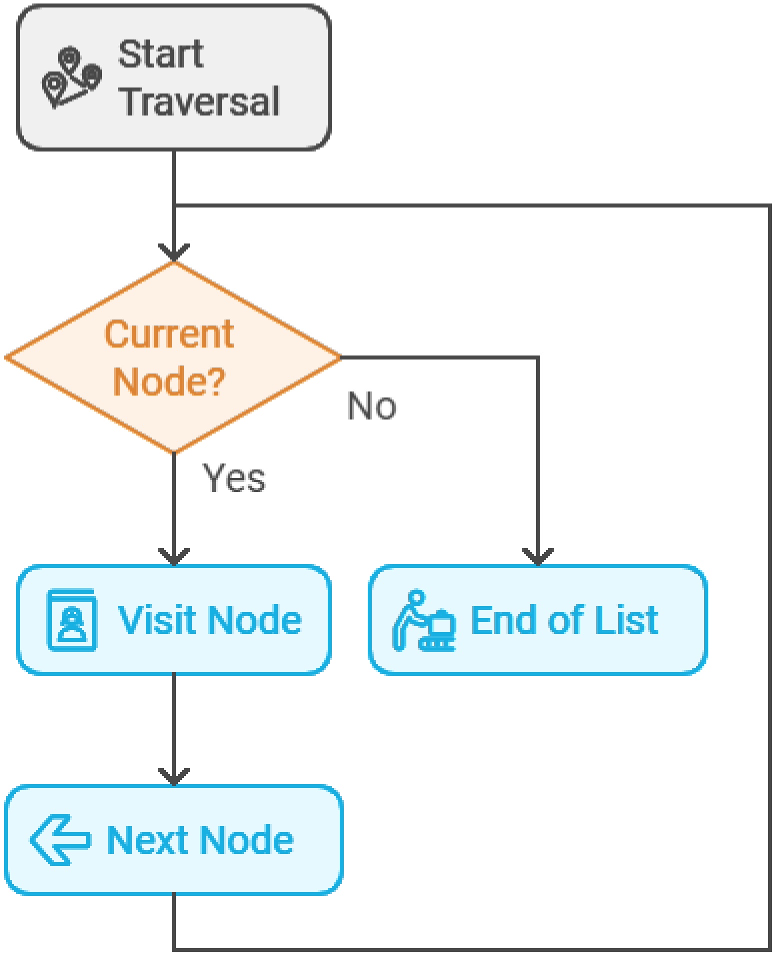
V. Linked List Operations

1. **Insertion**

```
public static void PrintList(Node head)
{
        Node curr = head;
        while (curr != null)
        {   Console.Write(" " + curr.data);
             curr = curr.next;
        }
        Console.WriteLine();
}
// Create Linked list 2->3->4->5
public static void Main()
{
        Node head = new Node(2);
        head.next = new Node(3);
        head.next.next = new Node(4);
        head.next.next.next = new Node(5);
        Console.WriteLine("Original Linked List:");
        PrintList(head);
        Console.WriteLine("After inserting at the front:");
        int data = 1;
        head = InsertAtFront(head, data);
        PrintList(head);
    }
}
```

V. Linked List Operations

2. Traversal Operation



V. Linked List Operations

2. Traversal Operation

```
// Creation and traversal of Linked List
using System;
class GFG {  // Structure of Node
        public class Node {
                public int data;
                public Node next;
        };
//Function to print the content of list
        static void printList(Node n)
        {       // Iterate till n reaches null
                while (n != null) {
                {       Console.Write(n.data + " ");
                        n = n.next;
                }
}
```

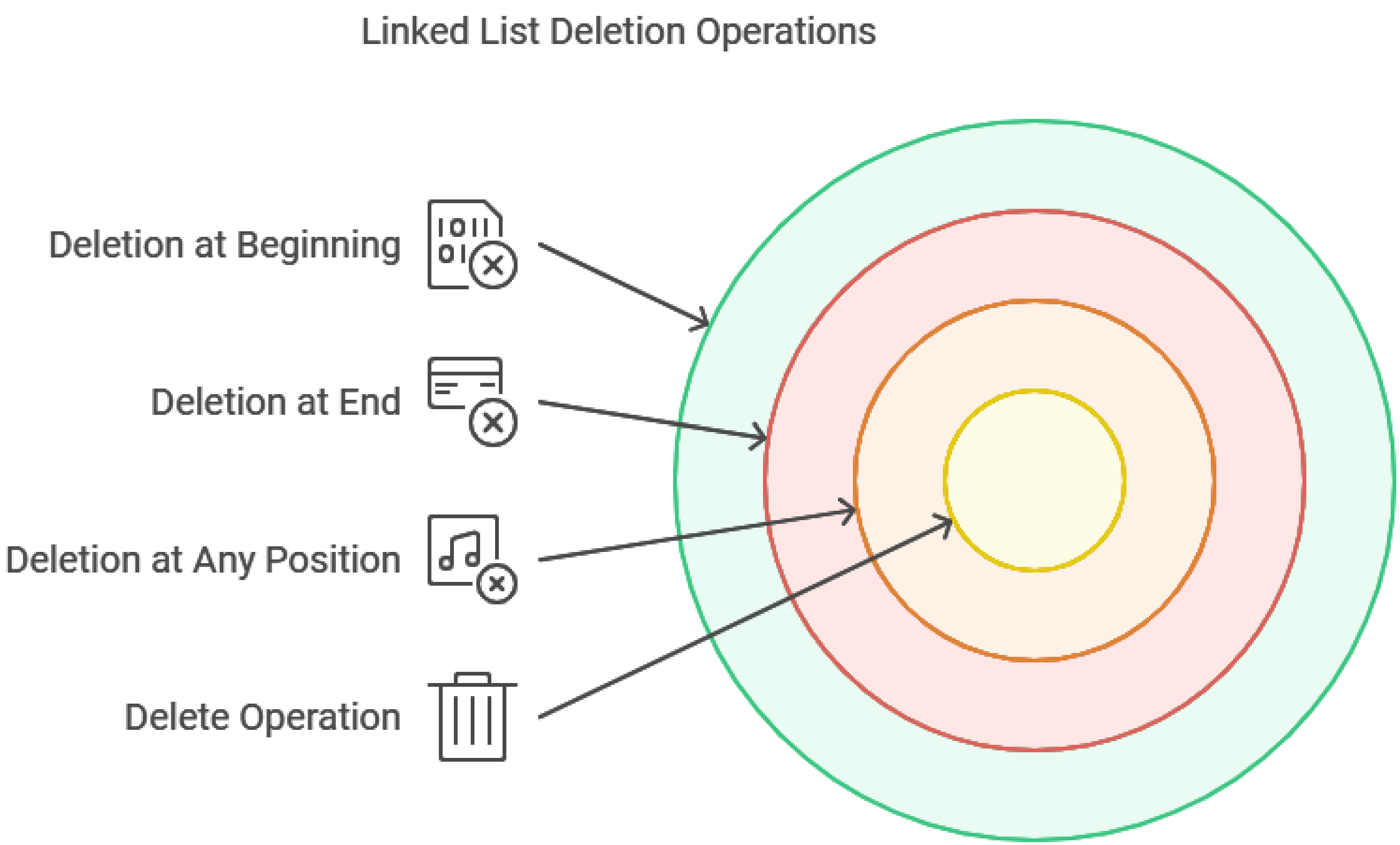
V. Linked List Operations

2. Linked List Traversal

```
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();
        head.data = 1;  // Assign data in first node
        head.next = second; // Link first node with second
        second.data = 2; // Assign data to second node
        second.next = third;
        third.data = 3; // Assign data to third node
        third.next = null;
        printList(head);
}
```

V. Linked List Operations

3. Deletion Operation



V. Linked List Operations

3. Deletion Operation

```
using System;
class Node {
        public int data;
        public int npx;
        public Node(int data) {
                this.data = data;
                this.npx = 0;       }
}
class XorLinkedList {
        private Node head;
        private Node[] nodes;
        public XorLinkedList() {
                head = null;
                nodes = new Node[100];
// assuming 100 as max number of nodes
}  }
```

V. Linked List Operations

3. Delete Operation

```
public void insert(int data) {
        Node node = new Node(data);
        nodes[data - 1] = node; // assuming data starts from 1 and is unique
        if (head != null) {
                node.npx = getPointer(head);
                head.npx = getPointer(node) ^ head.npx;           }
        head = node;     }
public void removeHead() {
        if (head == null) {Console.WriteLine("List Is Empty");
                return;           }
        int nextNodeId = head.npx;
        if (nextNodeId != 0) {
                Node nextNode = dereferencePointer(nextNodeId);
                nextNode.npx ^= getPointer(head);
                nodes[head.data - 1] = null; // removing head node from nodes
                head = nextNode;           }     }
```

V. Linked List Operations

3. Delete Operation

```
public void printList() {
        Node current = head;
        int prevAddr = 0;
        while (current != null) {
                Console.WriteLine(current.data);
                int nextAddr = prevAddr ^ current.npx;
                prevAddr = getPointer(current);
                current = dereferencePointer(nextAddr);      }
}
public void insert(int data) {
        Node node = new Node(data);
        nodes[data - 1] = node; // assuming data starts from 1 and is unique
        if (head != null) {        node.npx = getPointer(head);
                head.npx = getPointer(node) ^ head.npx;              }
        head = node;
}
```

V. Linked List Operations

3. Delete Operation

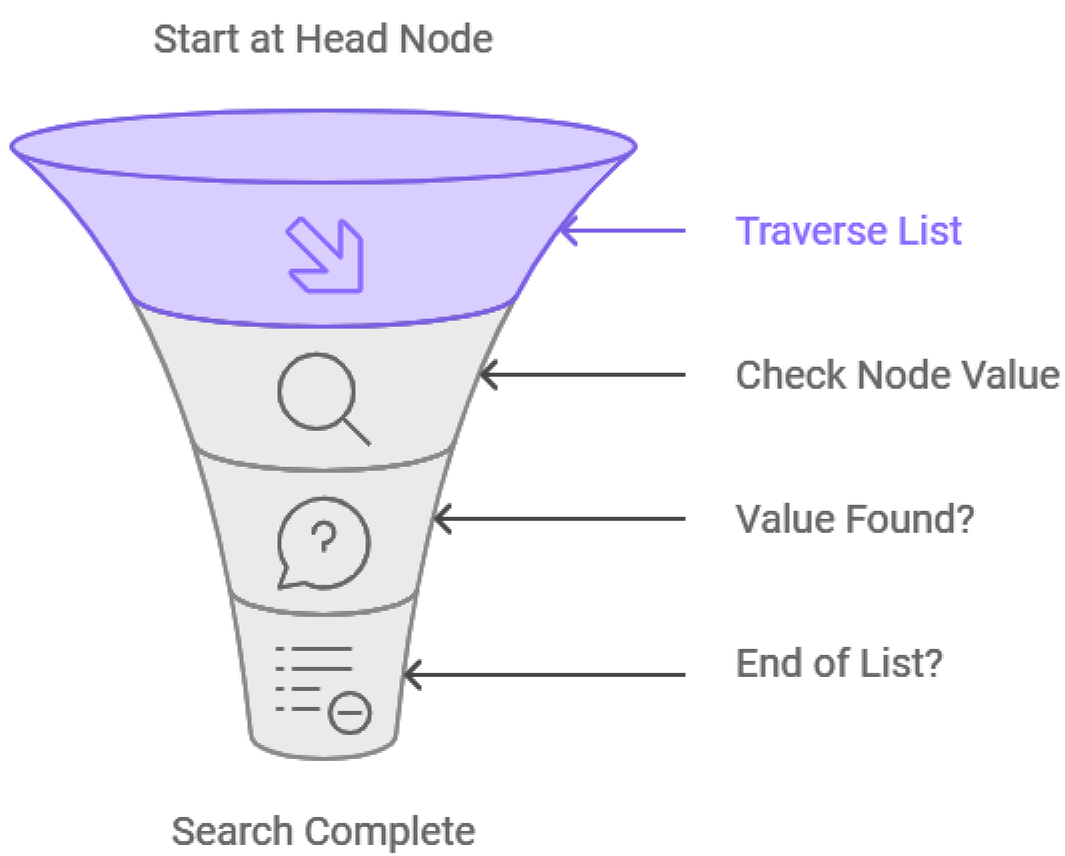
```
public void removeHead() {
        if (head == null) {        Console.WriteLine("List Is Empty");
                return;                 }
        int nextNodeId = head.npx;
        if (nextNodeId != 0) {
                Node nextNode = dereferencePointer(nextNodeId);
                nextNode.npx ^= getPointer(head);
                nodes[head.data - 1] = null; // removing head node from nodes
                head = nextNode;                }
}
private int getPointer(Node node) {
        // Using RuntimeHelpers.GetHashCode to get a unique
        return System.Runtime.CompilerServices.RuntimeHelpers.GetHashCode(node);
}
```

V. Linked List Operations

```
        private Node dereferencePointer(int address)
        {
                for (int i = 0; i < nodes.Length; i++) {
                        if (nodes[i] != null && getPointer(nodes[i]) == address)
                        {       return nodes[i];                }
                        }
                return null;
                }
        }
public class Program {
        public static void Main() {
                XorLinkedList xll = new XorLinkedList();
                xll.insert(10);         xll.insert(20);
                xll.insert(30);         xll.insert(40);
                xll.removeHead();       xll.printList();
        }
}
```

V. Linked List Operations

4. Search Operation:



V. Linked List Operations

4. Search Operation:

```
// Search an element in linked list
using System;  // A Linked List Node
class Node {
    public int Data;
    public Node Next;
// Constructor to initialize new node data
    public Node(int new_data) {
        Data = new_data;
        Next = null;
    }
}
```

V. Linked List Operations

ACLEDA UNIVERSITY

4. **Search Operation:**

```
class GFG { // Driver code
// Checks whether key is present in linked list
   static bool SearchKey(Node head, int key) {
// Initialize curr with the head of linked list
      Node curr = head;
// Iterate over all the nodes
      while (curr != null) {
// If the current node's value is equal to key,
       // return true
       if (curr.Data == key)
           return true;
       // Move to the next node
       curr = curr.Next;
      }
// If there is no node with value as key,
return false
      return false;
   }
```

V. Linked List Operations

4. Search Operation:

```
    static void Main() {
        // Create a hard-coded linked list:
        // 14 -> 21 -> 13 -> 30 -> 10
        Node head = new Node(14);
        head.Next = new Node(21);
        head.Next.Next = new Node(13);
        head.Next.Next.Next = new Node(30);
        head.Next.Next.Next.Next = new Node(10);
        // Key to search in the linked list
        int key = 14;
        if (SearchKey(head, key))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }
}
```

Quizzes

1. What is array data structure?
2. Why do we need arrays?
3. What are the applications of arrays?
4. What are the different types of array data structure?
5. How do we declare a single dimensional array?
6. What is the output of the following code:

```
int[] arr = new int[5] { 1, 2, 3, 4, 5 };
for (int i = 0; i < arr.Length; i++)
{     Console.Write(arr[i] + " ");      }
```

7. How do we declare a two-dimensional array?
8. What is the output of the following code:

```
int[,] a = { { 1, 2 }, { 3, 4 } };
for (int i = 0; i < 2; i++)
{   for (int j = 0; j < 2; j++)
    {     Console.Write(a[i, j] + " ");      }
}
```

9. What is a linked list data structure?
10. What are the applications for the Linked list?
11. What are the different types of Linked List data structures?
12. What is the output of the following code:

```
LinkedList list = new LinkedList();
list.Add(1);
list.Add(2);
list.Add(3);
list.Add(4);
list.PrintList();
```


THANK
YOU!