CENG 201 Veri Yapıları 3: Bağlı Listeler

Öğr.Gör. Şevket Umut ÇAKIR

Pamukkale Üniversitesi

Hafta 3

Anahat

Tek Bağlı Liste

- Ardışık olarak düğümleri(Node) barındıran bir veri yapısıdır
- Her düğüm bir değer(value) vede bir başka düğüme bağlantı(next) içerir
- Listedeki son düğümün bağlantısı boştur(null)
- Bağlı listenin bir başlangıç düğümü vardır

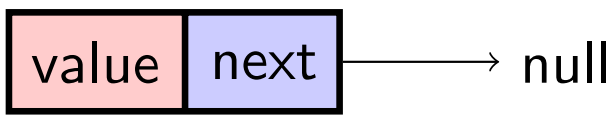


Figure: Tek bağlı liste düğümü(Node)

Düğüm Sınıfı

```
1  public class Node<T>
2  {
3      public T value;//Saklanan değer
4      public Node<T> next;//Bir sonraki eleman
5      public Node (){    }//Varsayılan kurucu metod
6      public Node (T value, Node<T> next)
7      {
8          this.value = value;
9          this.next = next;
10     }
11 }
```

Düğüm Sınıfı Kullanımı

```
1 public static void Main (string[] args)
2 {
3     Node<int> n2 = new Node<int> ();
4     n2.value = 5;
5     Node<int> n1 = new Node<int> (17, n2);
6     Node<int> n3 = new Node<int> ();
7     n3.value = 8;
8     n2.next = n3;
9 }
```

Örnek Kodla Oluşan Yapı

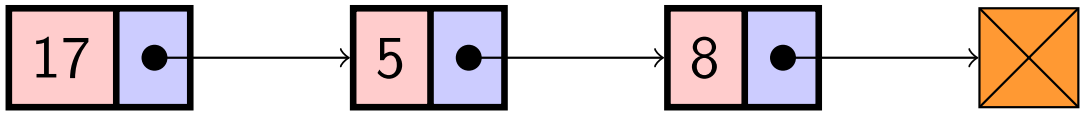
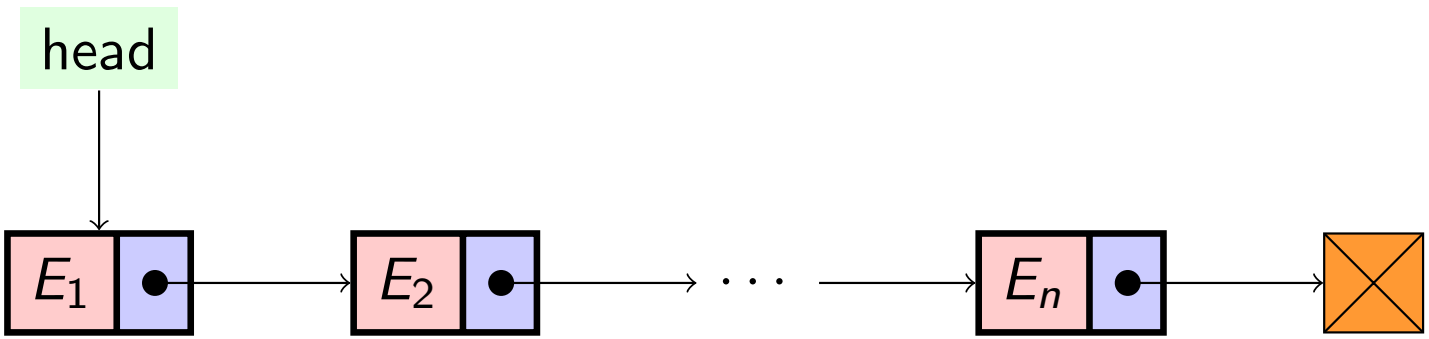


Figure: Temsili gösterim

Ad	Değer	Tür
▸ args	{string[0]}	string[]
▾ n1	{SLinkedList.Node<int>}	SLinkedList.Node<int>
▾ next	{SLinkedList.Node<int>}	SLinkedList.Node<int>
▾ next	{SLinkedList.Node<int>}	SLinkedList.Node<int>
next	(null)	object
value	8	int
value	5	int
value	17	int

Bağlı Liste Gösterimi/Linked List



Bağlı Liste Sınıfı

```
1  public class LinkedList<T>
2  {
3      private Node<T> head;//başlangıç düğümü
4      public LinkedList () {}
5      public void print() {}//Tüm listeyi yazdırır
6      public Node<T> getHead() {}//Ilk düğümü verir
7      public void addHead(T value) {}//Listenin başına ekler
8      public void addTail(T value) {}//Listenin sonuna ekler
9      public void insertAfter(Node<T> node, T value) {}//Verilen
   ↪    düğümden sonraya ekler
10     public void insertAfter(int index, T value) {}//Verilen konumdan
   ↪    sonraya ekler
11     public T removeAfter(Node<T> node) {}//Verilen düğümden sonrakini
   ↪    siler
12     public T removeAt(int index) {}//Verilen konumdaki düğümü siler
13     public T removeHead() {}//Listenin başını siler
14     public T removeTail() {}//Listenin sonunu siler
15 }
```

Soru

Aşağıdaki işlemler ekran çıktıları nasıl olur:

```
1  LinkedList<int> ll = new
    ↪  LinkedList<int> ();
2  ll.addHead (3);
3  ll.addHead (5);
4  ll.addHead (7);
5  ll.print ();
6  ll.addTail (9);
7  ll.print ();
8  ll.insertAfter (ll.getHead (), 15);
9  ll.print ();
10 ll.insertAfter (0, 76);
11 ll.insertAfter (5, 41);
12 ll.addHead (12);
13 ll.print ();
14 ll.removeAfter (ll.getHead ());
15 ll.print ();
16 ll.removeAt (3);
17 ll.print ();
18 ll.removeHead ();
19 ll.print ();
20 ll.removeTail ();
21 ll.print ();
```

Cevap

7 5 3
7 5 3 9
7 15 5 3 9
12 7 76 15 5 3 9 41
12 76 15 5 3 9 41
12 76 15 3 9 41
76 15 3 9 41
76 15 3 9

print metodu

```
public void print() {//Tüm listeyi yazdırır
    Node<T> current=head;
    while (current != null) {
        Console.Write ("{0} ", current.value);
        current = current.next;
    }
    Console.WriteLine ();
}
```

addHead

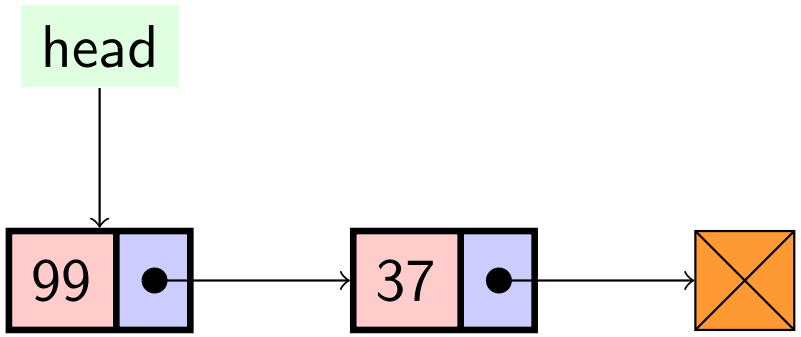


Figure: addHead(12) öncesi

addHead

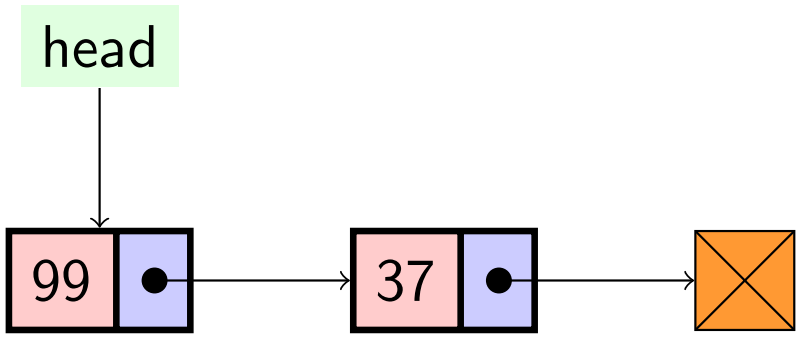


Figure: addHead(12) öncesi

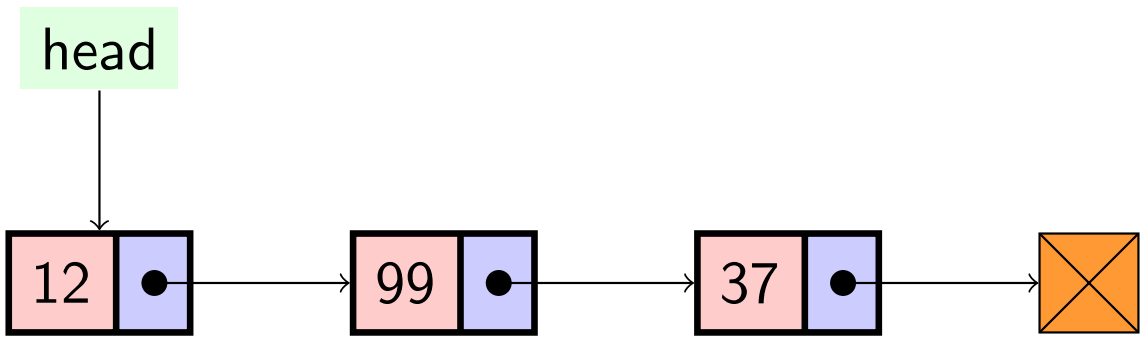


Figure: addHead(12) sonrası

addHead metodu

```
public void addHead(T value) {//Listenin başına ekler
     Node<T> newNode=new Node<T>(value,head);
     head = newNode;
}
```

addTail

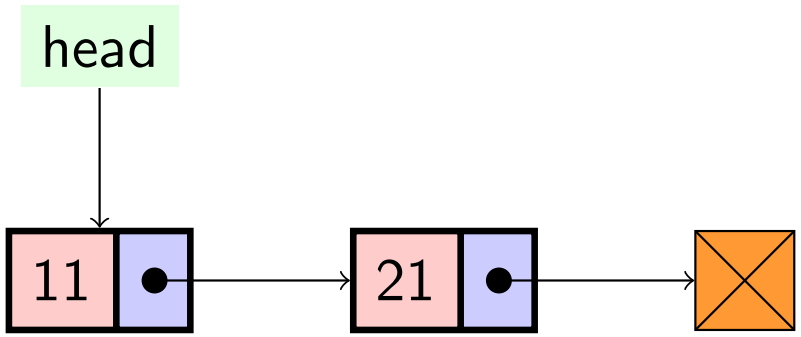


Figure: addTail(24) öncesi

addTail

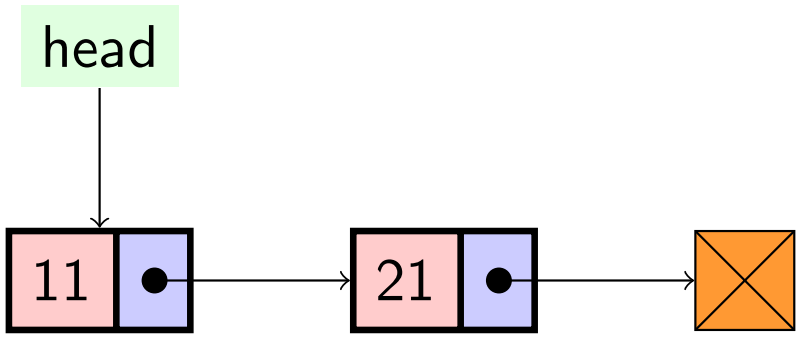


Figure: addTail(24) öncesi

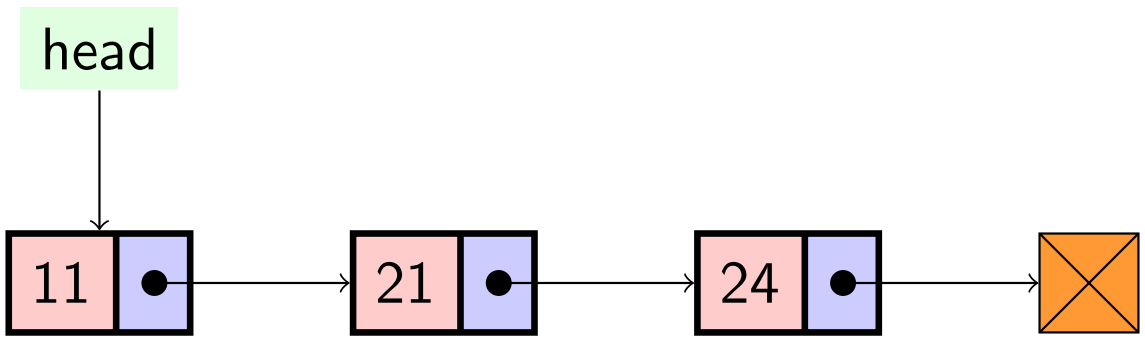


Figure: addTail(24) sonrası

addTail metodu

```
public void addTail(T value) {//Listenin sonuna ekler
    Node<T> current = head;
    while (current.next != null)
        current = current.next;
    current.next = new Node<T> (value, null);
}
```

addAfter metodu

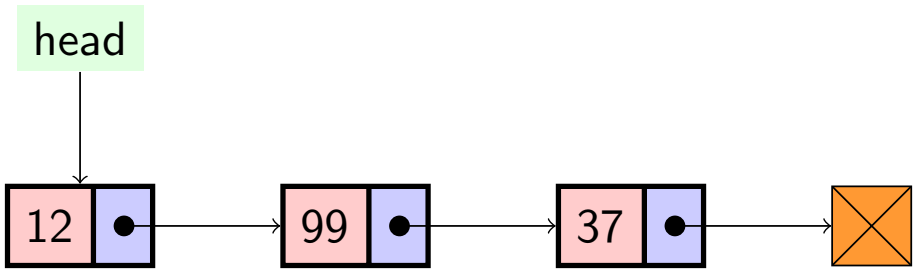


Figure: insertAfter(0,26) öncesi

addAfter metodu

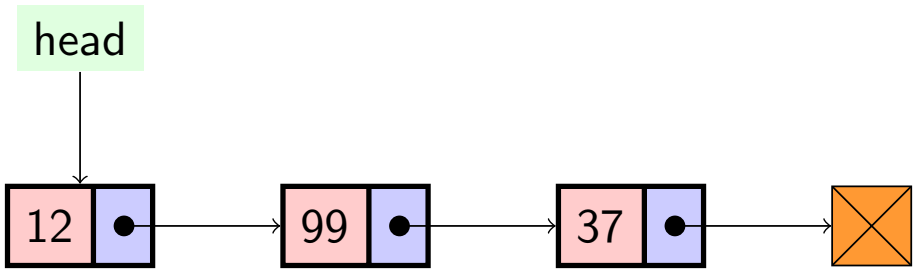


Figure: insertAfter(0,26) öncesi

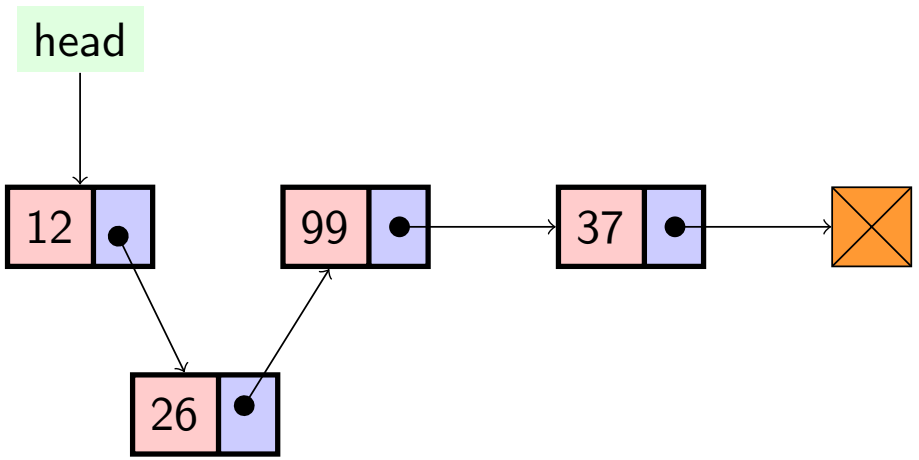


Figure: insertAfter(0,26) sonrası

insertAfter metodları

```
public void insertAfter(Node<T> node, T value) {//Verilen düğümden
 ↪  sonraya ekler
    Node<T> newNode=new Node<T>(value, node.next);
    node.next = newNode;
}
```

insertAfter metodları

```
public void insertAfter(Node<T> node, T value) {//Verilen düğümden
 ↪  sonraya ekler
    Node<T> newNode=new Node<T>(value, node.next);
    node.next = newNode;
}

public void insertAfter(int index, T value) {//Verilen konuma ekler
    Node<T> current = head;
    int currentLocation=0;
    while (current != null && currentLocation < index) {
        current = current.next;
        currentLocation++;
    }
    if (currentLocation != index)
        throw new IndexOutOfRangeException
 ↪  ("Listede yeterli eleman yok!");
    insertAfter (current, value);
}
```

removeHead metodu

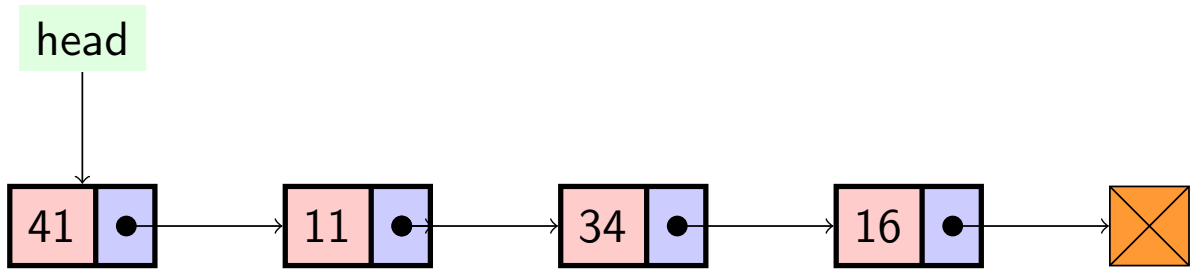


Figure: removeHead() öncesi

removeHead metodu

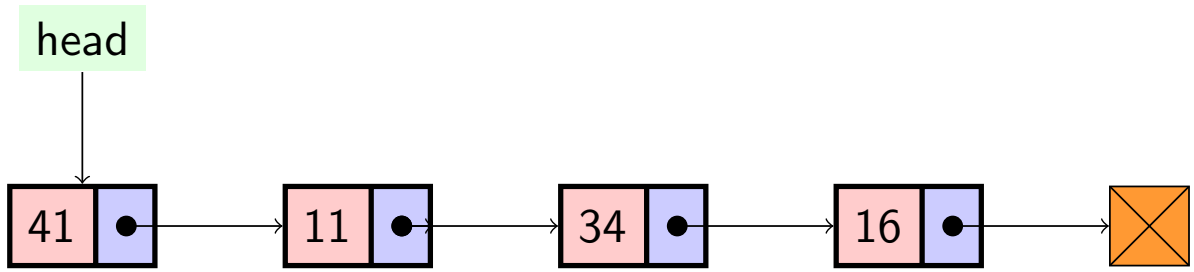


Figure: removeHead() öncesi

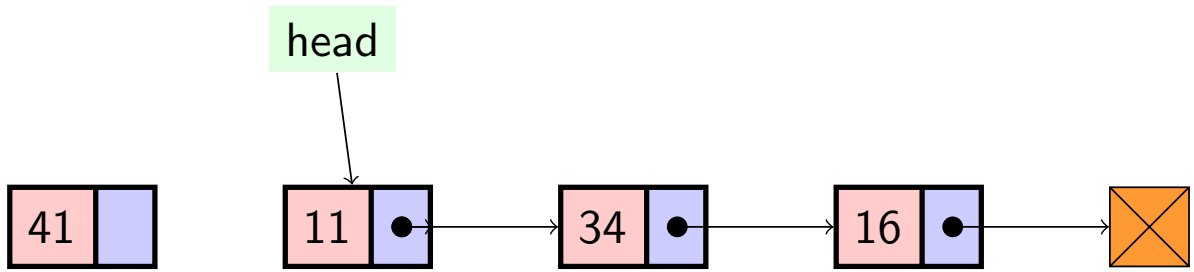


Figure: removeHead() sonrası

removeHead metodu

```
public T removeHead() {//Listenin başını siler
    Node<T> toDelete=head;
    head=head.next;
    toDelete.next = null;
    return toDelete.value;
}
```

removeTail metodu

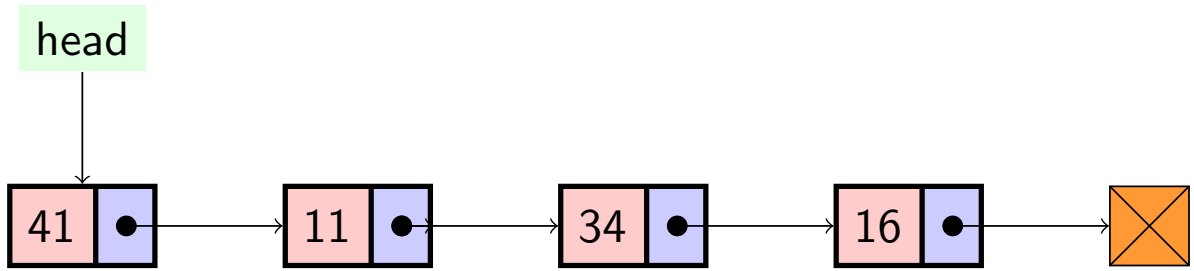


Figure: removeTail() öncesi

removeTail metodu

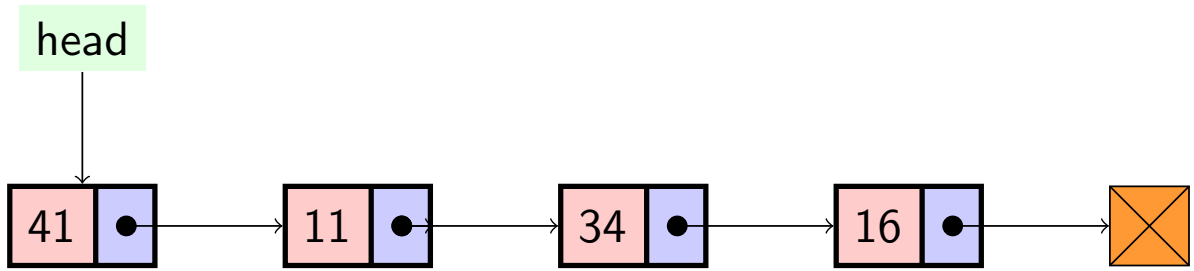


Figure: removeTail() öncesi

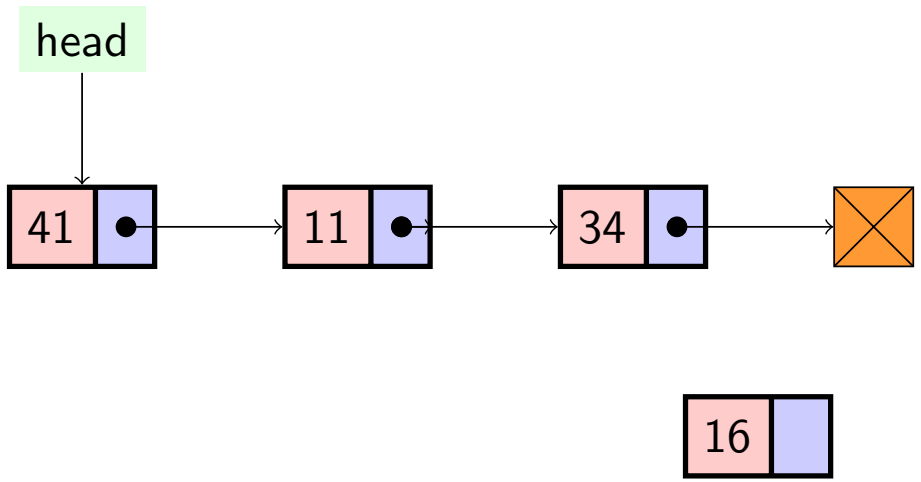


Figure: removeTail() sonrası

removeTail metodu

```
public T removeTail() {//Listenin sonunu siler
    Node<T> current=head;
    Node<T> previous = null;
    T r;
    while (current.next != null){
        previous = current;
        current = current.next;
    }
    if (previous != null) {
        r = previous.next.value;
        previous.next = null;
    } else {
        r = head.value;
        head = null;
    }
    return r;
}
```

Çift Bağlı Liste

- Her düğümde hem sonraki(next) hem de önceki(previous) düğüme bağlantı vardır
- Listenin başını(head) ve sonunu gösteren bağlantılar mevcuttur
- Liste üzerinde ileri ve geri yönlü gezinme yapılabilir

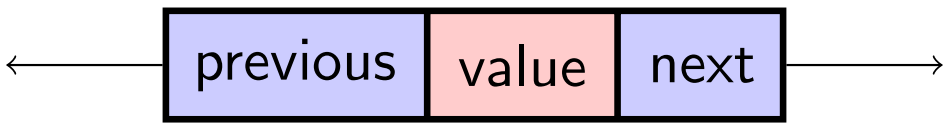
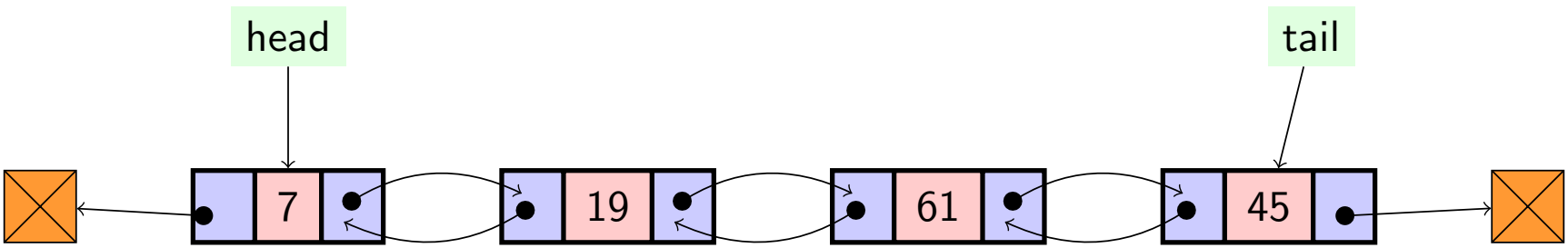


Figure: Çift Bağlı Liste Düğümü(Node)

Çift Bağlı Liste için Node Sınıfı

```
1  public class Node<T>
2  {
3      public T value;//Değeri saklayacak eleman
4      public Node<T> previous;// Bir önceki düğüm
5      public Node<T> next;//Bir sonraki düğüm
6      public Node (){    }
7      public Node (T value, Node<T> previous, Node<T> next)
8      {
9          this.value = value;
10         this.previous = previous;
11         this.next = next;
12     }
13     public Node (T value)
14     {
15         this.value = value;
16     }
17 }
```

Çift Bağlı Liste Temsili Gösterim



DoublyLinkedList Sınıfı

```
1  public class DoublyLinkedList<T>
2  {
3      Node<T> head;//Baştaki eleman
4      Node<T> tail;//Sondaki eleman
5      public DoublyLinkedList (){}
6      public void print(){}//Elemanları yazdırır
7      public void reversePrint(){}//Elemanları tersten yazdırır
8      public void addHead(T value){}//Başa ekler
9      public void addTail(T value){}//Sona ekler
10     public void insertAt(int index, T value) {}//Belirtilen konuma
       ↪  ekler
11     public T removeHead(){}//Baştaki elemanı siler
12     public T removeTail(){}//Sondaki elemanı siler
13     public T removeAt(){int index}//Belirtilen konumdaki elemanı
       ↪  siler
14     public int Count{get {} }//Eleman sayısını verir
15 }
```

print metodları

```
public void print(){
    Node<T> current=head;
    while(current!=null) {
        Console.WriteLine (current.value);
        current = current.next;
    }
}//Elemanları yazdırır
public void reversePrint(){
    Node<T> current=tail;
    while(current!=null) {
        Console.WriteLine (current.value);
        current = current.previous;
    }
}//Elemanları tersten yazdırır
```

addHead metodu

```
public void addHead(T value){
    Node<T> newNode=new Node<T>(value,null,head);
    if (head == null) {
        head = tail = newNode;
        return;
    }
    head.previous = newNode;
    head = newNode;
}//Başa ekler
```

addTail metodu

```
public void addTail(T value){
    Node<T> newNode=new Node<T>(value,head,null);
    if (tail == null) {
        head = tail = newNode;
        return;
    }
    tail.next = newNode;
    tail = newNode;
}//Sona ekler
```

removeHead metodu

```
public T removeHead(){
    if (head == null)
        throw new Exception ("Bağlı liste boş");
    Node<T> toDelete = head;
    head = head.next;
    if (head != null)
        toDelete.next.previous = head;
    else tail = null;
    return toDelete.value;
}//Baştaki elemanı siler
```

removeTail metodu

```
public T removeTail(){
    if (tail == null)
        throw new Exception ("Bağlı liste boş");
    Node<T> toDelete = tail;
    tail = tail.previous;
    if (tail != null)
        toDelete.previous.next = tail;
    else head = null;
    return toDelete.value;
}//Sondaki elemanı siler
```

Count özelliği

```
public int Count{get {
        int count=0;
        Node<T> current = head;
        while (current != null) {
            current = current.next;
            count++;
        }
        return count;
    } }//Eleman sayısını verir
```

Doubly Linked List İşlem Karmaşıklığı

Table: Mevcut uygulamada işlem karmaşıklıkları

İşlem	**Karmaşıklık**
addHead, addTail	$O(1)$
removeHead,removeTail	$O(1)$
insertAt	$O(n)$
removeAt	$O(n)$
Count	$O(n)$
print, printReverse	$O(n)$

LinkedList< T > Sınıfı

System.Collections.Generic içinde

- Diziler kullanılarak gerçekleştirilen bir liste yapısıdır
- Bazı metodları ve özellikleri:
  - `Count, First, Last`
  - `AddFirst(T), AddLast(T), RemoveFirst(T), RemoveLast(T)`
  - `AddBefore(LinkedListNode<T>,T)`
  - `Remove(T), Remove(LinkedListNode<T>)`
  - `Find(T), FindLast(T), Contains(T)`

Listeler ile Yığıt ve Kuyruk

Aşağıdaki işlemler liste kullanılarak nasıl gerçekleştirilebilir?

- Yığıt
  - *push*
  - *pop*
- Kuyruk
  - *enqueue*
  - *dequeue*


prev->next = toDelete->next;
delete toDelete;
// if only forgetting were
// this easy for me.
SNIFF
assert "It's going to be okay.";

[1]

[1] xkcd