Dizi (Array):

- Veriler bellekte ardışık olarak saklanır, dolayısıyla sonraki ve önceki elemana ulaşmak için fazladan bir bellek gereksinimi yoktur.
- Boyut dinamik olarak değiştirilebilir ancak dinamik bellek yönetimi gerektirir (malloc, free, vb).
- 1 [N], 2[N, M], 3 [N, M, K] veya daha fazla [N, M, K, …, Z] boyutlu olarak organize edilebilir.

N elemanlı ve tek boyutlu bir dizi *(A)* aşağıdaki şekilde görselleştirilebilir.

İndis	0	1	2	3	4	5	6	..	N-1
Veri	1	2	3	4	5	6	7	..	N

- Erişim *A[indis]* şeklinde gerçekleşebilir. Erişim tek denemede O (1) mümkün, diğer işlemler?
- Sıralı bir dizide araya eleman eklemek veya silmek istersek kaydırma (shift) gerekli, dizilerle zor gibi?
- Bellekte N tane eleman için yer var ama N tane ardışık yer yok, dizilerle mümkün değil! Peki bu durumda ne yapmalı?

Bağlantılı Liste (Linked List) :

- Verilerin bellekte ardışık olarak bulunması gerekli değil.
- Uzunluğu dinamik olarak değiştirilebilir.
- Elemanlar arası bağlantı için fazladan bellek lazım. (Trade-off between array and linked list)

N elemanlı ve tek boyutlu bir bağlantılı liste aşağıdaki şekilde görselleştirilebilir. (Mavi renk varsayımdır.)

Veri	1	2	3	4	5	6	..	N-2	N-1	N
Bellekteki Adresi	**100**	**108**	**200**	**194**	**94**	**380**	**…**	**…**	**250**	**40**
Sonraki Elemanın Adresi	**108**	**200**	**194**	**94**	**380**	**…**	**..**	**250**	**40**	**NULL**

- Tüm listeyi gezmek için neyi bilmeliyiz?
- Listenin sonuna geldiğimizi nasıl anlarız?
- Şimdiye kadarki bilgilerimizle elemanlar için A dizisi ve bir sonraki elemanları gösteren işaretçi PO[] dizisi ile tek yönlü liste oluşturursak ve bu listeye 7 elemanını ekleyelim. Ekleme yaparken dizilerdeki kaydırma (shift) işleminin maliyetinden kurtulabiliyoruz.

i	A[]	PO[]
0		5
1	2	3
2	5	4
3	4	2
4	8	-1
5	1	1

i	A[]	PO[]
0		5
1	2	3
2	5	6
3	4	2
4	8	-1
5	1	1
6	7	4

- Listeden şimdi de 4'ü silelim. Kaydırma(shift) maliyeti yine azaldı.

i	A[]	PO[]
0		5
1	2	3
2	5	6
3	4	2
4	8	-1
5	1	1
6	7	4

i	A[]	PO[]
0		5
1	2	2
2	5	6
3	4	2
4	8	-1
5	1	1
6	7	4

Kod Örneği:

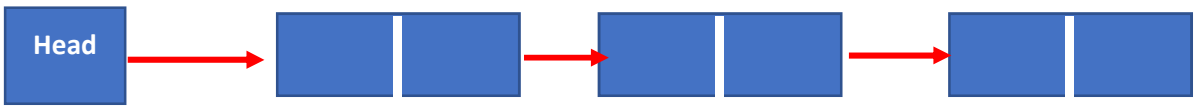
- Bir linkli liste oluşturalım.
- Eleman ekleyen, silen ve listeleyen fonksiyonları yazalım
- LinkedLists.c'yi inceleyim..

➢ Listede ileri-geri gidip gelmek için belleği daha fazla kullansak, o zaman ne kadar maliyetli olur?

Veri	1	2	3	4	5	6	..	N-2	N-1	N
Bellekteki Adresi	100	108	200	194	94	380	...	...	250	40
Sonraki Elemanın Adresi	108	200	194	94	380	...	..	250	40	NULL
Önceki Elemanın Adresi	NULL	100	108	200	194	94	380	...	...	250

➢ Listenin tümünü gezmek dizilere göre maliyetli oldu, peki ne kadar?
➢ İstediğimiz bir elemana doğrudan (A[i]) erişemiyoruz, bu pek iyi olmadı :) Yine trade-off :)
➢ Tek yönlü listede son elemanda, iki yönlü listede ilk ve son elemanda NULL yazan yerlere ne yazarsak liste dairesel hale gelir?
➢ Her ne kadar örnekteki listeler bağlantılı liste gibi görünse de, bunlar gerçekte dizilerle oluşturuldu. Yani N elemanı saklamak için bellekte yine ardışık 2N tane hücre kullandık. Ama bağlantılı listeler ardışık olarak saklanmak zorunda değildi? Çözüm pointer (pascal, C, C++) veya referans (C++, Java, C#) kullanımı.
➢ Tek (ileri) yönlü bir bağlantılı listeyi,
  - Bellekte farklı yerlerde saklanan **Düğüm (NODE)** adı verilen küçük parçalara bölelim,

  - Bu Düğümlerin bellekteki adreslerini ilk elemandan (**HEAD**) başlayıp pointer/referans yoluyla birbirine bağlayalım
  - Son elemanın bir sonraki elemanın adresi sakladığı yeri de NULL yapalım.



➢ İki (ileri/geri) yönlü bir bağlantılı listeyi,
  - Bellekte farklı yerlerde saklanan **Düğüm (NODE)** adı verilen küçük parçalara bölelim,