Doğrusal Veri Yapıları 3 - Yığın (Stack)

Tolgahan Çepel · Follow
4 min read · Jan 22, 2019

Share

Bilginin geliş sırasına göre, en son gelen elemana ilk erişilen liste yapısına **yığın (stack)** denir. Verilere yalnız bir uçtan erişim sağlanır.

Bu erişimde *Last-In-First-Out (LIFO)* prensibi vardır. Yani son giren eleman, ilk çıkar. Örneğin üst üste dizilen kitapları, yalnızca en üsttekine erişecek şekilde düşünebiliriz.

Stack tasarımı dizi üzerinde veya bağlı liste ile yapılabilir. Bağlı liste kullanarak boyutu sabit olmayan bir stack oluşturabiliriz. Dizi kullanmak için ise sabit bir boyut belirlemeliyiz.

UYGULAMA

Konuyu daha iyi anlatabilmek için, dizi üzerinde gerçekleştirilmiş bir stack uygulaması anlatacağım. İşlemler:

1. Eleman ekle *(push)*
2. Eleman çıkar *(pop)*

Dizinin sabit bir boyutu olmalı. Bu boyutu 6 olarak belirledim. Genel anlamda stack aşağıdaki gibi görünecek.

1. Eleman Ekle (Push)

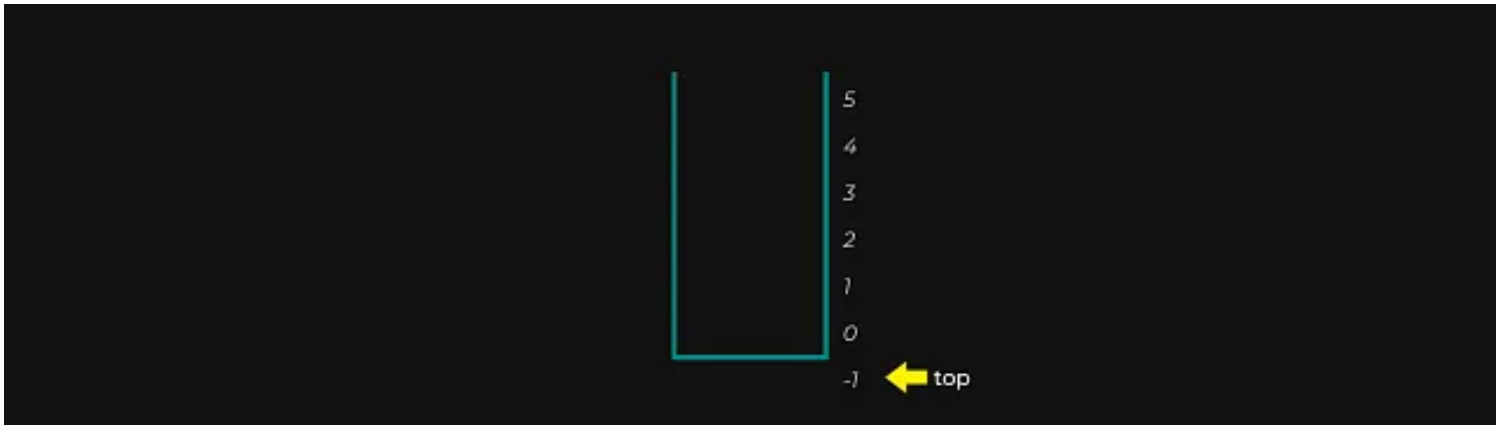
Eleman ekleme işlemi için ***push*** (ittirmek, sıkıştırmak) terimi kulanılır. Push işlemininin kodlarını görelim.

```
10  void Stack::push(int key)
11  {
12      if(stk.top == STACK_SIZE - 1)
13          std::cout << "Stack dolu.";
14      else
15      {
16          stk.top++;
17          stk.data[stk.top] = key;
18      }
19  }
```

Eğer stack doluysa eleman ekleyemeyiz. Bu yüzden öncelikle dizinin dolu olup olmadığını kontrol ediyoruz.

```
if(stk.top == STACK_SIZE - 1)
     std::cout << "Stack dolu.";
```

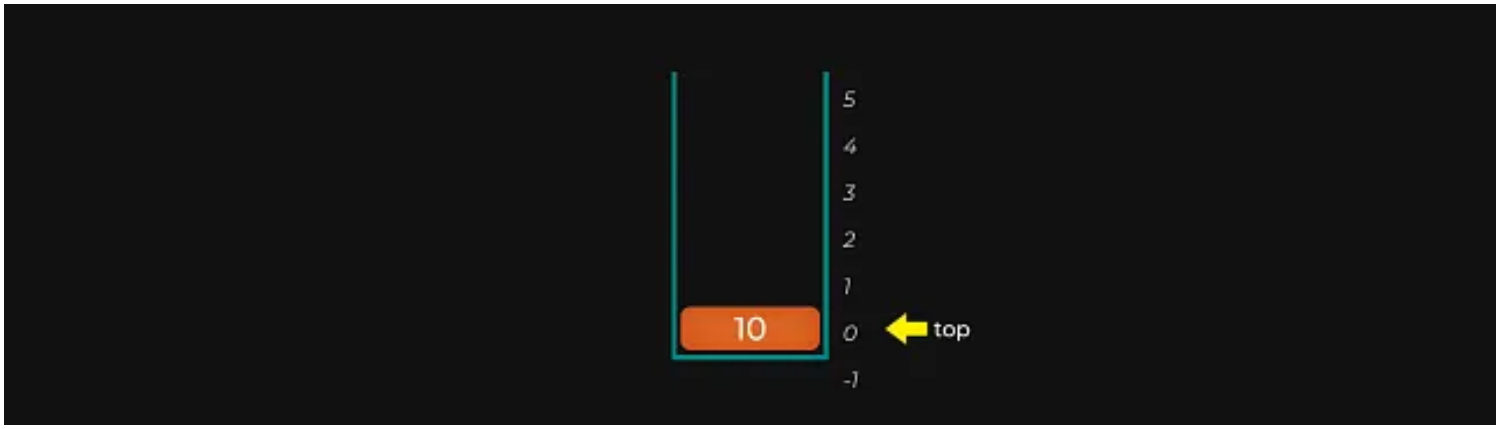
> *Not: En fazla 6 eleman eklenebilir. Fakat indis 0'dan başladığı için* `STACK_SIZE-1` *kullandık.*



Dizinin boş olduğu durum.

Dizi dolu değil ise yeni eleman ekleyebiliriz. Bunun için en başta -1 olan top değerini arttırarak, indise sayıyı atıyoruz.

```
stk.top++;
stk.data[stk.top] = key;
```



Bir eleman eklendiği durum.

Sırasıyla birkaç eleman eklersek stack aşağıdaki gibi görünecektir.

```
push(10);
push(20);
push(30);
push(40);
push(50);
push(60);
```

Tamamen dolu olduğu durum.

> *Daha fazla eleman eklemek istersek, `if` ifadesi ile hata uyarısı alınır.*

2. Eleman Çıkar (Pop)

Eleman çıkarma işlemi için ***pop*** (çıkmak) terimi kullanılır. Pop işlemininin kodlarını görelim.

```
21  int Stack::pop()
22  {
23      if(stk.top == -1)
24          std::cout << "Stack bos.";
25      else
26      {
27          int x = stk.data[stk.top];
28          stk.top--;
29          return x;
30          // ya da return(stk.data[stk.top--]);
31      }
32  }
```

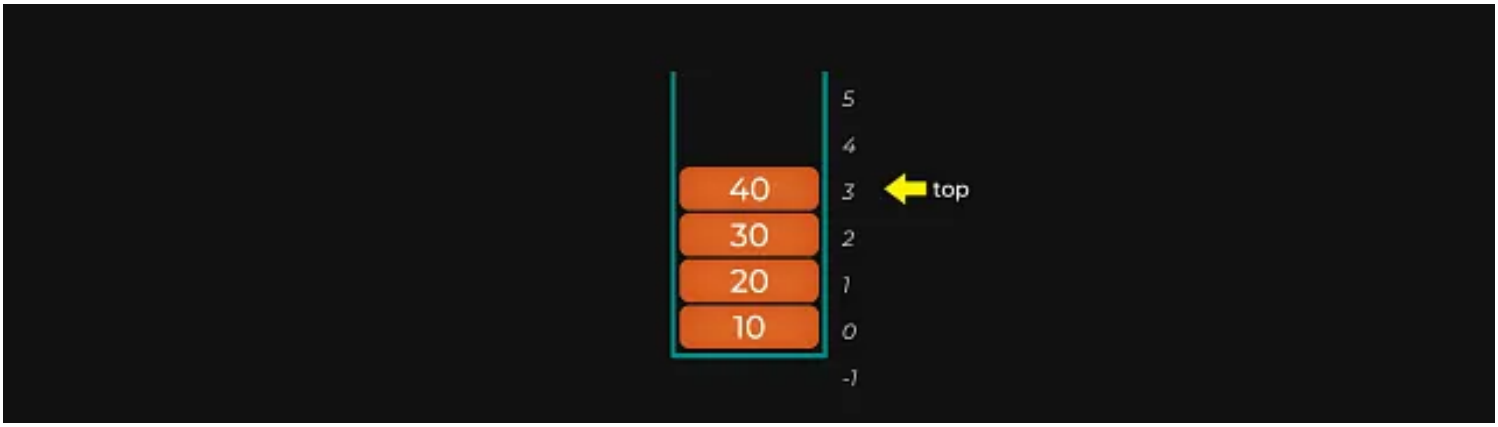
Eğer stack zaten boşsa eleman çıkartamayız. Bu yüzden öncelikle dizinin boş olup olmadığını kontrol ediyoruz.

```
if(stk.top == -1)
     std::cout << "Stack bos.";
```

Dizi boş değil ise eleman çıkarabiliriz. Bu işlemi ***yalnızca top değerini bir azaltarak*** yapacağız. Dizi indisi silinemeyeceği için, sileceğimiz indisi görmezden geleceğiz.

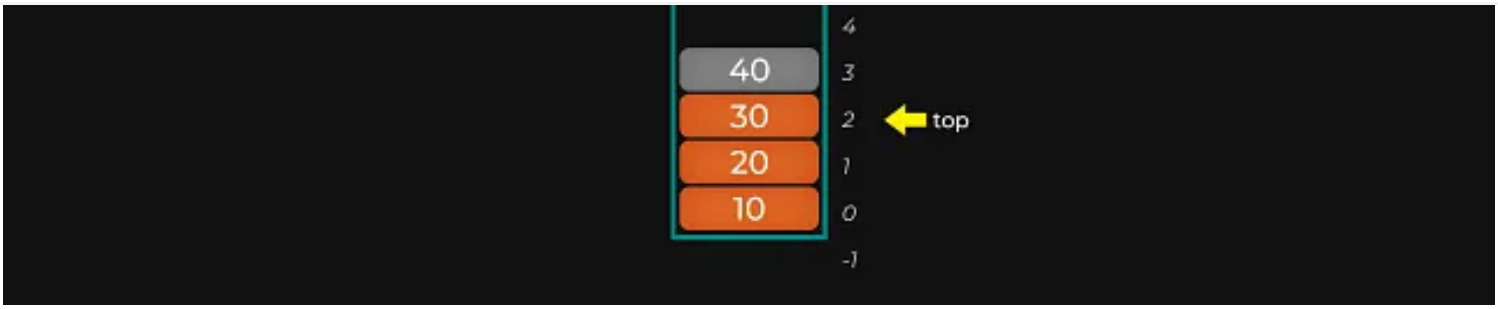
```
int x = stk.data[stk.top];
stk.top--;
return x;
```

`return` kullanılarak silinecek değer fonksiyona geri döndürülür. Böylelikle hangi elemanın çıkarıldığı kontrol edilebilir.



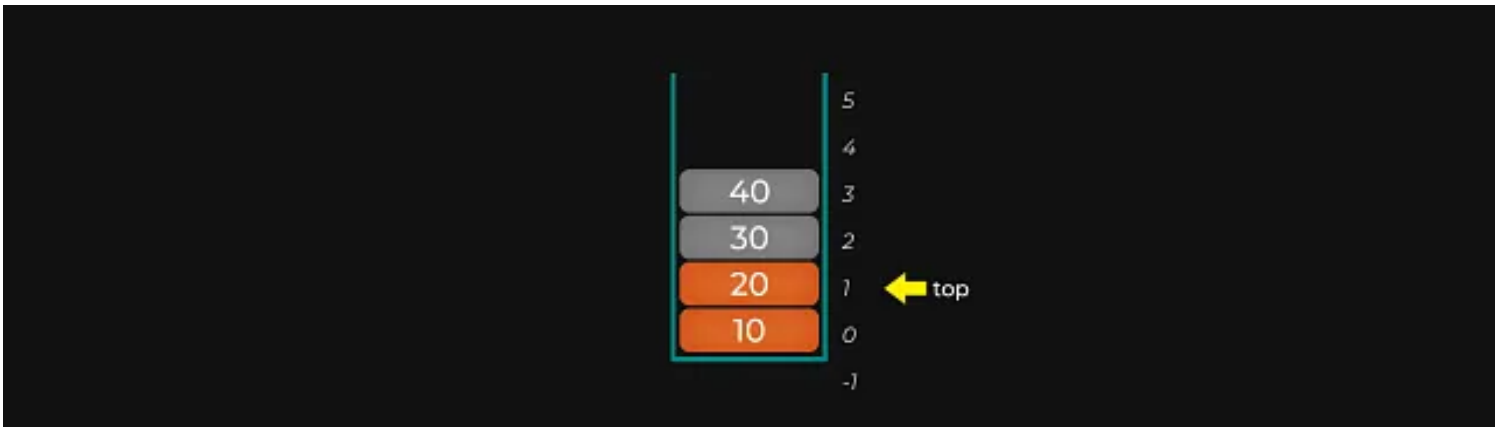
4 eleman olduğu durum.

Dizide 4 eleman varken, bir defa `pop()` işlemi uyguladığımızda aşağıdaki gibi olacaktır.



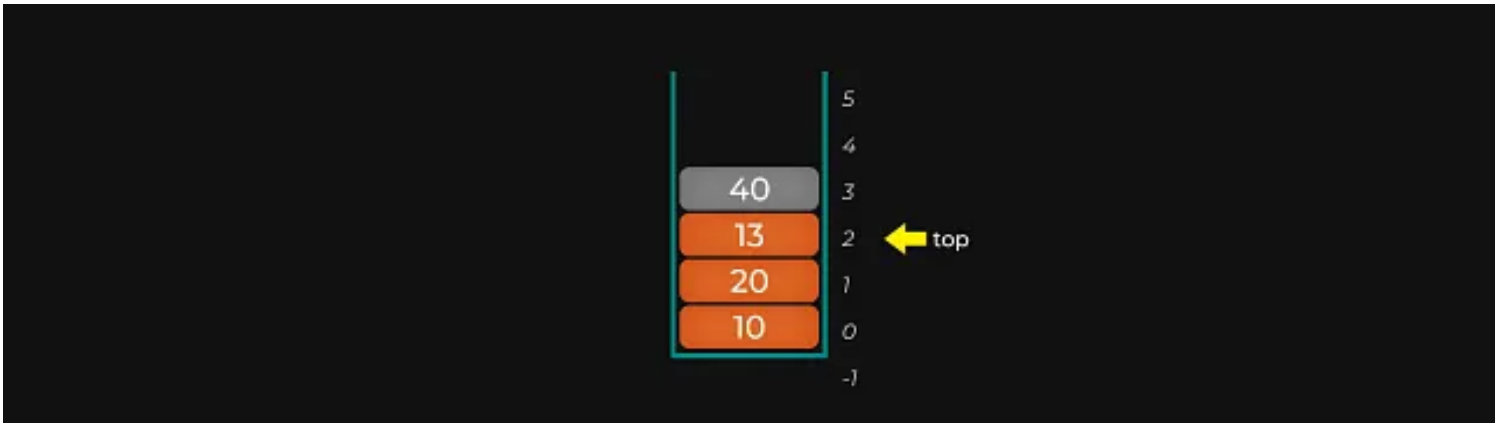
Görüldüğü üzere 40 elemanı hâlâ yerini koruyor. Fakat biz top ifadesinden sonrakileri görmezden geliyoruz.

Bir kez daha `pop()` işlemi uygulayalım.

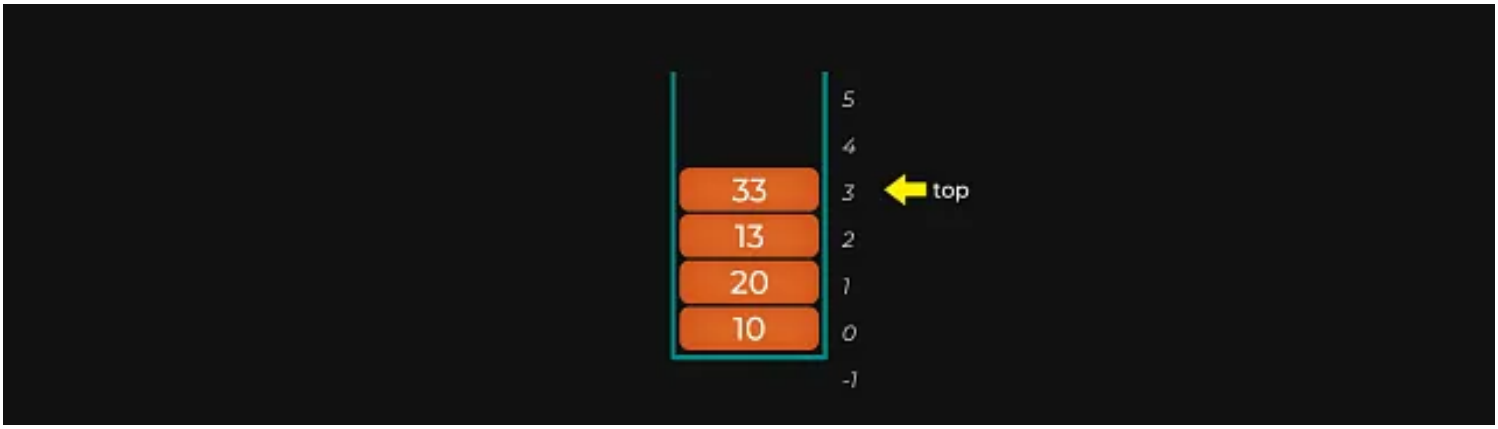


Peki bu durumdayken tekrar eleman eklersek ne olur? — 2. indisin üzerine yazılır.

```
push(13)
```




```
push(33)
```



4 defa `pop()` işlemi uygularsak, artık top indisi -1 olacaktır. Yani diziyi boş kabul edeceğiz.

Boş olduğu durum.

> *Daha fazla eleman çıkarmak istersek, `if` ifadesi ile hata uyarısı alınır.*

Avantajları:

- En son eklenen elemana ve sırasıyla sonrakilere erişilmek isteniyorsa, kullanılması büyük avantaj sağlar.

Kullanım Alanları:

- Undo işlemi stack ile yapılır.
- Web browser'larda geri işlemi (back) için stack kullanılır.
- Matematiksel işlemlerdeki operatörler (+,*,/,-) ve operandlar için stack kullanılır.

Sonuç

Bu yazımda stack (yığın) veri yapısına ve dizi üzerine uygulamasına değindim. Veri yapısının mantığı anlaşılırsa kolaylıkla bağlı liste ile uygulanabilir.

Anlamakta zorlandığınız veya eksik, yanlış gördüğünüz detaylar için ***bana ulaşabilirsiniz***.

GitHub üzerinden paylaştığım örnekleri ***buradan*** inceleyebilirsiniz.

Önceki: ***Doğrusal Veri Yapıları 2 — Bağlı Liste (Linked List)***

Sonraki: ***Doğrusal Veri Yapıları 4 — Kuyruk (Queue)***

Data Structures | Computer Science | Software Development | Data Science | Computer Engineering