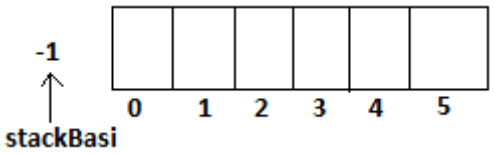
Yığıt

Yığıt en son eklenen elemanın, ilk çıktığı özel bir veri yapısıdır. Birçok alanda sıklıkla kullanılmaktadır. Örnek olarak bir web tarayıcısında bir önce gezmiş olduğunuz web sitesine gitmek için geri butonuna bastığınızda yığıttan pop işlemi yapılır ve en son eklenen eleman getirilir. Yığıt dizi ile gerçekleştirilebildiği gibi liste kullanılarak ta gerçekleştirilebilir.

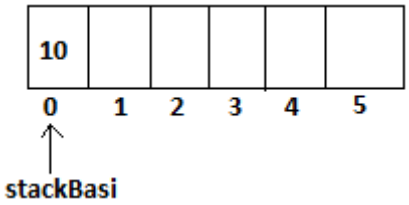
Yığıtın Dizi ile Gerçekleştirimi

Yığıtta bulunan elemanlar bir dizi içerisinde tutulur. Diziye eleman ekleme işlemi (push işlemi), dizide en son elemanı gösteren indeks bir arttırılarak indeksin yeni gösterdiği yere eklenir.

Yığıt ilk durumda boş olacağı için yığıtın başını tutan indeks değeri -1'i gösterecektir.



Yığıta örneğin 10 sayısı eklendiğinde dizi aşağıdaki gibi bir duruma dönüşür.




```
#ifndef STACK_HPP
#define STACK_HPP

#include <iostream>
using namespace std;

#include "ElemanYok.hpp"

template <typename Nesne>
class Stack{
      private:
          Nesne *elemanlar;
          int stackBasi;
          int elemanSayisi;
          int kapasite;
…
```

Burada gerçekleştirilen yığıt şablon olduğu için bütün kodlar başlık dosyasında bulunacaktır. Kapasite dizinin alabileceği max eleman sayısını gösterir. Eleman sayısı kapasiteyi doldurduğunda yerAc metodu çağrılarak varolan kapasitenin 2 katı kadar yeni bir dizi oluşturulur.

```
…
        bool dolumu(){
                return elemanSayisi == kapasite;
        }
        void yerAc(int boyut)
        {
                Nesne *tmp = new Nesne[boyut];
                for(int j=0;j<elemanSayisi;j++) tmp[j]= elemanlar[j];
                if(elemanlar != NULL) delete [] elemanlar;
                elemanlar = tmp;
                kapasite = boyut;
        }
…
```

Yığıttan kurucu metot (constructor) yardımıyla bir nesne türetildiği zaman dizi göstericisi NULL'ı gösterecek şekilde nesneyi oluşturur.

```
…
public:
        Stack(){
                elemanlar=NULL;
                stackBasi=-1;
                elemanSayisi=0;
                kapasite=0;
        }
…
```

İlk durumda ortada bir dizi olmadığı için kapasite sıfırdır. Yığıtta eleman olup olmadığını stackBasi değişkeninin -1 göstermesi ile ya da elemanSayisi değişkenine bakarak karar verilebilir.

```
bool isEmpty() const{
        return stackBasi == -1;
}
```

Yığıta eleman ekleme (push) konu başında belirtildiği gibi indeks değeri ile oynama yapılarak gerçekleştirilir. Burada dikkat edilmesi gereken durum eleman sayısının kapasiteyi aşıp aşmama durumudur.

```
void push(const Nesne &eleman){
                if(dolumu()){
                        yerAc(max(1,2*kapasite));
                }
                stackBasi++;
                elemanlar[stackBasi] = eleman;
                elemanSayisi++;
        }
```

Yığıttan eleman silme (pop) işleminde yapılan sadece stackBasi indeks değerini bir çekmektir. Tabi yığıt boş iken bu metodun çağrılması bir hata olacağı için ElemanYok hatası bu durumlarda fırlatılmaktadır.

```
void pop() throw(ElemanYok){
                if(isEmpty()) throw ElemanYok("Eleman yok");
                stackBasi--;
                elemanSayisi--;
        }
```

Yığıtı kullanan programcı yığıtta sıradaki elemanı görmek isteyebilir. Bu işlem elemanı yığıttan çıkarmamalıdır. Bu işlemin özel adı top metodudur.

```
const Nesne& top()const throw(ElemanYok){
                        if(isEmpty()) throw ElemanYok("Eleman yok");
                        return elemanlar[stackBasi];
                }
```

Yığıt ile iş bitip ayrılan nesne belleğe geri iade edilmesi gerektiğinde yıkıcı metot çağrılır. Bu yıkıcı metot makeEmpty metodunu çağırarak yığıtı ilk durumuna getirecektir.

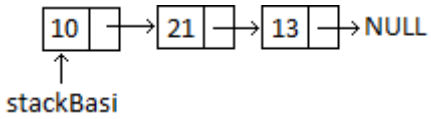
```
void makeEmpty(){ // stack'i ilk haline çivirir
                        stackBasi=-1;
                        elemanSayisi=0;
                        kapasite=0;

                        if(elemanlar != NULL) delete [] elemanlar;
                }
                ~Stack(){
                        makeEmpty();
                }
};
#endif
```

Yığıtın Liste ile Gerçekleştirimi

Bu gerçekleştirimde yığıtın her bir elemanı bir düğüm içerisinde tutulur. Bağıl listenin, diziden daha kötü bir erişilme performansı olmasına rağmen yığıtta bu çok fark etmeyecektir. Bunun nedeni yığıtın yapısında gizlidir. En son eklenen eleman ilk silineceği için push işleminde listenin sürekli başına ekleme yapılır ve pop işleminde ise listenin ilk elemanı silinecektir. Dizi sınırlı bir alana sahip olduğu için genişletme işlemi gerekmekteydi Bağıl liste dinamik bir yapıya sahip olduğu için böyle bir işleme gerek bulunmamaktadır.

Örneğin yığıta ilk olarak 13 sayısı daha sonra 21 sayısı ve son olarak 10 sayısının eklendiğini düşünürsek yığıtın son hali aşağıdaki gibi olacaktır.



Bu yapıyı sağlayabilmek için ilk önce bir düğüm yapısına ihtiyaç vardır.

```
#ifndef STACK_HPP
#define STACK_HPP

#include <iostream>
using namespace std;

#include "ElemanYok.hpp"

template <typename Nesne>
class Dugum{
    public:
        Nesne eleman;
        Dugum *ileri;
        Dugum(const Nesne& elm,Dugum *ilr=NULL){
                eleman=elm;
                ileri=ilr;
        }
};
…
```

Yığıt şablon bir yığıt olduğu için düğüm de şablon bir düğüm olmalıdır. Yığıt oluştuğu ilk durumda hiç eleman olmadığı için stackBasi NULL değerini göstermektedir.

```
template <typename Nesne>
class Stack{
        private:
                Dugum<Nesne> *stackBasi;
        public:
                Stack(){
                        stackBasi=NULL;
                }
…
```

Dolayısıyla yığıtın boş olup olmadığını bu değere bakarak öğrenebiliriz.

```
bool isEmpty() const{
                        return stackBasi == NULL;
                }
```

Yığıta eleman ekleme (push) yapılırken stackBasi gelen bu yeni düğümü gösterirken, yeni düğümün ilerisi daha önceki yığıtı gösterecektir. Bu işlemi düğüm sınıfının yapıcı metodundan faydalanarak tek satırda gerçekleştirebiliriz.

```
void push(const Nesne& eleman){
                    stackBasi = new Dugum<Nesne>(eleman,stackBasi);
          }
```

Yığıttan eleman silme (pop) işlemi listenin başındaki düğümü silme işlemidir.

```
void pop() throw(ElemanYok){
                    if(isEmpty()) throw ElemanYok("Eleman yok");
                    Dugum<Nesne> *silinecek = stackBasi;
                    stackBasi = stackBasi->ileri;
                    delete silinecek;
          }
```

Yine aynı şekilde top işlemi listenin başındaki düğümün içerdiği elemanın ne olduğunu gösterme işlemidir.

```
const Nesne& top() const throw(ElemanYok){
                    if(isEmpty()) throw ElemanYok("Eleman yok");
                    return stackBasi->eleman;
          }
```

Yığıtın temizlenme işlemi yığıtta eleman kalmayana kadar pop metodunu çağırma şeklinde yapılabilir.

```
…
          void makeEmpty(){
                    while(!isEmpty())
                              pop();
          }
          ~Stack(){
                    makeEmpty();
          }
};
#endif
```

Hazırlayan

Arş. Gör. Dr. M. Fatih ADAK