VERİ YAPILARI VE PROGRAMLAMA

GİRİŞ

Mikrobilgisayar sisteminde veri veya veri grupları, önceden yeri belirlenmiş (Computed address) bellek (RAM) alanı yada alanlarında tutulmaktadır. Bir tek verinin veya bir veri grubunun bellekte önceden yerini belirlemenin, o bellek alanının veya bellek alanlarının isimlendirilmesi demek olduğunu biliyoruz. Bu isimlendirilme; bir tek veri (sayısal veya alfa sayısal) için yapılıyorsa tekil (scaler) özellikte değişken isimlendirilmesi olduğunu ve herhangi bir anda sayısal veya alfa sayısal bir veriyi tanımlanan o isimdeki bellek alanında tutabildiğimizi, yada bir grup veri için her birinin aynı özellikte, aynı büyüklükte veya eşit uzunlukta alanlardan meydana gelme koşulu ile tanımlanan bellek alanlarında tutabildiğimizi ***Algoritma ve Programlamaya Giriş*** dersimizden öğrenmiştik. Yapılan bu açıklamalar doğrultusunda, verinin tekil veya bir grubu oluşturacak şekilde, bir tek bellek alanının veya bellek alanlarının, bir isim ile tanımlanmasına göre veri alanları yeniden gruplandırılırsa;

- Basit veri yapıları
- Basit olmayan veri yapıları

başlıkları altında toplanabilir.

Basit Veri Yapıları ile, bir isim altında tekil özellikteki bir tek verinin tutulmasını anlıyoruz. ***Basit Olmayan Veri Yapıları*** ile de birden fazla aynı özellikte ve aynı büyüklükteki veri grubu yada gruplarını veya farklı özellikte ve farklı büyüklükteki veri gruplarının bir isim altında tutulmasını tanımlıyoruz.

Yukarıda tanımlanan veri yapılarının ikincisi olan Basit Olmayan Veri Yapıları, ilerleyen satırlarda ***Veri Yapıları ve Programlama*** adı altındaki çalışmaların konularını oluşturacaktır. Aynı zamanda Basit Olmayan Veri Yapıları başlığı altında tanımlanan veri gruplarının bellekteki duruşlarına göre ve yapıdaki herhangi bir veriye erişimi bakımından, bu yapıların özellikleri ve davranışları üzerinde durulacaktır.

BASİT OLMAYAN VERİ YAPILARI

GİRİŞ

Bellekte tanımlanan veri grupları; bellekteki durumlarına ve yapıdaki herhangi bir veriye erişim bakımından, tanımlamalarına göre farklı yapısal özellikler ve farklı davranışlar göstermektedirler. Yapılan tanımlamaları, farklı yapısal özellikleri ve farklı davranışları göz önünde bulundurularak yeniden gruplandırılırsa, Basit Olmayan Veri Yapıları;

1- Doğrusal Listeler
2- Doğrusal Olmayan Listeler

başlıkları altında toplanabilir.

Bu gruplandırmalara ve yapısal özellikleri de göz önünde tutularak ***Doğrusal Listeler*** ve ***Doğrusal Olmayan Listeler*** yapısal ve davranışları bakımından farklılıklar göstermeleri nedeni ile yeniden genel tanımları ve bu başlıklar altında Basit olmayan veri yapıları gruplandırmaları yapılırsa;

- **Doğrusal listeler**

  - Sıradan bellek konumlu doğrusal listeler

    - Diziler (Arrays)
    - Yığıtlar (Stacks)
    - Kuyruklar (Queues)

  - Bağlaçlı bellek konumlu doğrusal listeler

    - Tek bağlaçlı doğrusal listeler

      - Doğrusal tek bağlaçlı doğrusal listeler
      - Dairesel tek bağlaçlıdoğrusal listeler

    - Çift bağlaçlı doğrusal listeler
      - Doğrusal çift bağlaçlı doğrusal listeler
      - Dairesel çift bağlaçlı doğrusal listeler

- **Doğrusal olmayan listeler**

  Ağaç yapıları (İkili ağaç yapıları)

Basit Olmayan Veri yapılarını yukarıda verilen konu başlıkları altında davranışları bakımından ilerleyen satırlarda inceleneceklerdir.

Doğrusal listeler

- Sıradan bellek konumlu doğrusal listeler
  - Diziler (Arrays)
  - Yığıtlar (Stacks)
  - Kuyruklar (Queues)

DOĞRUSAL LİSTELER

GİRİŞ

Bellekte birbiri ardı sıra olacak şekilde, bellek konumlarında sıralanmış elemanlardan oluşan veri alanlarına (gruplarına) **Doğrusal Listeler** adı verilir.

Doğrusal Listelerdeki elemanlar; bellekte, ya birbiri ardı sıra gelen bellek konumlarında ya da liste elemanları farklı bellek konumlarında olup, birbiri ardı sıralığı bağlaç denilen bir özel yapı aracılığı ile sağlanarak oluşturulan yapılardır. Yapılan bu tanımlamaya göre Doğrusal listeleri bellekteki konumlarına göre;

1- Sıradan bellek konumlu doğrusal listeler

2- Bağlaçlı bellek konumlu doğrusal listeler

Başlıkları altında toplanabilir.

Sıradan Bellek Konumlu Doğrusal Listeler' deki elemanlar arasındaki bağlantı, elemanların birbiri ardı sıra gelen bellek konumlarında olması ile gerçekleşmektedir.

Diğer taraftan bellekte bir listeyi oluşturan fakat belleğin farklı konumlarında bulunabilen veri grupları da bulunmakta ve doğrusal listeler içinde değerlendirilmektedir. Ancak bu veri grupları arasındaki birbiri ardı sıralık, bağ olarak isimlendirilen özel bir yapı aracılığıyla gerçekleşmektedir. Buna göre; dizinin eleman adresleri birbiri ardı sıra olmayan, farklı adres konumlarında bulunabilen doğrusal listelere ***Bağlaçlı Bellek Konumlu Doğrusal Listeler*** adı verilir.

Doğrusal listelerdeki elemanlar, bellek alanlarında birbiri ardında veya bir bağ aracılığı ile birbiri ardı sıra getirilmektedirler. Bu özelliklerinden dolayı doğrusal listelerin herhangi bir elemanına ulaşıp herhangi bir elemanı ile işlem yürütülebilir. **X** gibi **n** elemanlı bir doğrusal liste

X(1),X(2),X(3),……X(k-1),X(k),X(k+1),…….X(n)

şeklinde yazılabilir ve

doğrusal listeler üzerinde yapılabilecek işlemler genel olarak aşağıdaki gibi sıralanabilir.

- Listenin **k.** Sırasındaki elemana erişip, elemanlar içindeki bilgi yada bilgiler kontrol edilebilir veya değiştirilebilir.
- **k.** Elemandan hemen önceki yere yeni bir eleman eklenebilir.
- **k.** Elemanı listeden çıkarılabilir.
- İki veya daha fazla listeyi bir listede birleştirilebilir.
- Bir liste iki veya daha fazla liste haline getirilebilir.
- Listenin kopyası çıkarılabilir.
- Liste elemanları sıralanabilir.
- Liste içindeki bir eleman aranabilir.

Yukarıdaki açıklamalar doğrultusunda ***Sıradan Bellek Konumlu*** ve ***Bağlaçlı Bellek Konumlu*** doğrusal listeler ilerleyen satırlarda ayrıntılı olarak inceleneceklerdir.

SIRADAN BELLEK KONUMLU LİSTELER

GİRİŞ

Genel anlamda bir dizinin (doğrusal listenin) tanımlanması ile, dizi sınırlandırılmış olmaktadır. Buna göre; tanımlanan her dizi bellekte tanımlanan eleman sayısına bağlı olarak, belli büyüklükte bir bellek alanı tutacaktır. Dizi için tutulan bellek alanında, dizi elemanlarının birbiri ardı sıra veya yan yana sıralanış durumlarına göre, her bir dizi elemanına erişim, elemanların sıralanışına bağlı olarak gerçekleşir.

Örneğin; tanımlanan **n** elemanlı bir **X** doğrusal listesinde **n>0** olmak koşuluyla **k.** sıradaki elemanını **X(k)** elemanı olarak isimlendirilir.

Bellekte bir birleri ardı sıra sıralanışları bakımından Sıradan Bellek Konumlu Listeler adı altında toplanan dizileri, davranışları bakımından üç başlık altında toplanırlar. Bunlar;

- Diziler (Arrays)
- Yığınlar (Stacks)
- Kuyruklar (Queues)

olarak isimlendirilirler.

Sıradan Bellek Konumlu Listeleri sırası ile aşağıdaki satırlarda inceleyelim.

DİZİLER (ARRAYS)

Bellekte yan yana veya birbiri ardı sıra gelen bellek konumlarında sıralanmış elemanlardan oluşan, istenildiğinde herhangi bir elemanına ulaşıp herhangi bir işlemi yürütebilme yada istenildiğinde herhangi bir yerine eleman eklenebilme veya eleman çıkarabilme özelliğinde olan veri alanlarına **dizi** (Array) denir.

Diziler bilgisayar belleğinde eşit uzunluktaki aynı tipten birbiri ardı sıra gelen bilgi sahalarıdır.

Herhangi bir tekil (scaler) değişkenin kullanacağı değer için bellekte yeri belirlenmiş (Computed address) bir bellek adresi gerekiyorsa, dizilerde de aynı bu şekilde tek bir değişken adı altında , eşit uzunluktaki (büyüklükteki) alanlardan oluşan bir bellek alanı gerekmektedir. Tanımlanan bu bellek alanının en basit formu tek boyutlu dizilerdir. Dizilerin diğer formları çok boyutlu diziler olarak isimlendirilir.

Yukarıdaki gibi eşit uzunlukta veya büyüklükte oluşan alanlardan meydana gelen bir bellek alanı demekle, bir dizinin (tek boyutlu veya çok boyutlu dizi olsun) bellek alanını sınırlandırmış oluruz. Bir diziyi sınırlandırmak demek , dizideki tutulabilen veri sayısını belirlemek demektir. Dizinin eleman sayısı belirsiz olamaz. Buna göre; **n** elemanlı bir diziyi açıklıyorsak, **n** tane eşit büyüklükte bellek alanı ard arda gelecek demektir. Bir bellek alanı ile bir tek veri için tanımlanan büyüklük anlatılmaktadır.

Eğer bir bellek alanı 1 byte büyüklükte ise **n * 1** byte'lık bir alan **n** elemanlı bir dizi için bellekte yer açılıyor demektir. Dizinin elemanları aynı büyüklükte ve aynı özellikte olmalıdır. Örneğin; dizinin tüm elemanları sayısal veya karakter yada alfa sayısal özelliktedir. Bir dizide farklı özelliklerde eleman olamaz.

Programlama dillerinde tanımlanan dizinin bellekte belli bir isim altında n adet aynı özellik ve aynı büyüklükte veriler tutuluyorsa, tanımlanan dizi isminin altında ardışık tam sayılarla isimlendirilen belli büyüklüklerdeki bellek yerlerinden konuşulabilir. Bu bellek yerlerinin bir biri ardı sıra tam sayılarla isimlendirilmesine dizinin **indisi** adı verilir. İndis sayısal olarak belirlenen başlangıç ve bitiş değerleri olarak dizi elemanlarının nereden başlayıp, nerede sona ereceğini gösterir. Bu nedenle dizinin ilk elemanı alt sınır indisi, son elemanı üst sınır indis değerini göstermektedir. İndis değerlerine bakarak dizinin kaç elemanlı olduğu bulunabilir. Bir dizinin alt sınır indis değerini **AS** ve üst sınır indis değerini **US** olarak gösterilirse;

```
Dizi_eleman_sayısı=(US - AS)+1
```

ifadesi ile tek boyutlu dizinin eleman sayısı hesaplanabilir.

Örnek:

Algoritmada tanımlanan **DIM A(100)** dizisinin alt sınırı 1 ve üst sınır değeri 100 olarak kabul edilsin.

Kaç elemanlı bir dizi olduğunu bulmak için yukarıda verilen formülde yerine koyarsak;

```
Eleman_sayısı=(100-1)+1
              =100-1+1
              =100     adet olduğu hesaplanır.
```

Örnek:

A dizisinin DIM A(-2 to 8) olarak tanımlandığı kabul edilsin.

```
Dizinin eleman sayısı=(8-(-2))+1
                     =(8+2)+1
                     =8+2+1
                     =11 sayısına ulaşılır.
```

Dolayısıyla tanımlanan A dizisi 11 elemanlıdır denilir.

Diğer taraftan A dizisinin herhangi bir elemanının yerini bulmak, dizinin ilk elemanının bellekteki yerine göre belirlenmektedir. A dizisi bellekte tanımlandığında dizinin bütün elemanlarının toplam büyüklüğü kadar bir bellek alanı ayrılmış olur. Ayrılan bu alanın ilk yeri A dizisinin temel adresi (base address) diye isimlendirilir ve **base(A)** ile gösterilir. Dizinin her bir elemanının büyüklüğünü esize olarak kabul edelim. **A** dizisinin ilk elemanı **A[1]** elemanı olduğuna göre referans olarak alınırsa, **A[1]** elemanı **base(A)** konumundadır. **A[2]** elemanı ise **base(A)+esize** ve **A[3]** elemanı **base(a)+2*esize**'da dır denilir.

Genel olarak A dizisinin herhangi bir elemanının adresi

```
Base(A)+(indis -1)* esize
```

şeklinde belirlenebilir. Böylece dizinin herhangi bir elemanının yeri, dizinin indisine bağlı olarak gerçekleşmiş olur.

Bir dizinin indis değeri, herhangi bir sayısal büyüklükten başlamış olabilir. Buna göre yukarıdaki formülü yeniden düzenleyebiliriz. Bu durumu takip eden örnekle açıklayalım.

Örnek:

A gibi bir dizinin ilk indis değeri 10, maxsimum indis değeri 100 olsun. **Base(A)**, A dizisinin ilk elemanı olan **A[10**] 'nun konumundadır. **A[11]** ise **base(A) + esize** konumundadır. **A[12]** ise **base(a) + 2 * esize** konumundadır.

Genel anlamda; **AS, A** dizisinin tam sayı alt sınırı kabul edilsin. **A** dizisinin **A[i]**'ninci elemanı

```
Base(a)+(i-AS)*esize
```

adresinde konumlanmış demektir.

Tek Boyutlu Dizilerin Pascal da Kullanımı

Pascal da bir dizinin tanımlanması ile bellek alanında, ne kadar büyüklükte bir bellek alanının ayrılacağını veya kullanılacağını önceden derleyiciye bildirmiş oluyoruz. Bu bildirme işlemine dizinin tiplendirme adı verilir. Buna göre Pascal genel bildirimi;

```
Var
Dizi_değişken_ismi : Array [alt_sınır_değeri .. üst_sınır_değeri] of tip_ismi;
```

veya

```
Type
Dizi_tip_ismi = Array [alt_sınır_değeri .. üst_sınır_değeri] of tip_ismi;

Var
 Dizi_değişken_ismi : Dizi_tip_ismi;
```

Şeklindedir.

Örnek:

```
Var
   b: Array[1..100] of integer;
```

veya

```
Type
    Tip_ismi = Array [1..100] of integer;

Var
   b: Tip_ismi;
```

satırları ile bellekte bir biri ardı sıra gelen 100 adet her biri 2 byte (integer) büyüklüğünde **b** dizi değişken ismi ile yer ayrılmış olur. Ayrılan yerlerin ilkinin adresi **b** dizisinin temel adresi (Base Address) adını alır ve **Base(b)** ile gösterilir. Dizinin her bir elemanının büyüklüğü **esize** olarak kabul edelim. **B[1]** elemanı referans olarak alınırsa, **b[1]** elemanı **base(b)** konumundadır. **b(2)** elemanı **base(b) + esize** ve **b(3)** elemanı **base(b) + 2 * esize** dadır.

Genel anlamda **b[i]** elemanının adresini söyleyecek edecek olursak;

```
Base(b)+(i – 1) * esize
```

Adres konumundadır.

Bir dizideki herhangi bir elemanın yerini o elemanın indis ifadesi ile belirleyebiliriz. Böylece dizideki herhangi bir elemanın yerini göstermek mümkün olur.

Aşağıdaki gibi **C** isimli dizi değişken tanımlanmış olsun;

```
Var
     C: Array [10..100] of integer;
```

Base(C), C dizisinin ilk elemanı olan C[10] ‘ nun konumuna gönderilir.
C[11], Base(C) + esize konumundadır.
C[12], Base(C) +2 * esize konumunda
C[i], Base(C) + (i – 10) * esize konumundadır.

Genel anlamda D gibi bir dizinin herhangi bir elemanının adresini ifade edecek olursak;

```
Base( D ) + (ius – ias) * esize
```
konumundadır.

NOT:

ius: indis üst sınır değeri
ias: indis alt sınır değeri

İki Boyutlu Diziler

İki boyutlu dizilere bakıldığında; belli bir eleman sayısına sahip olan bir dizinin, iki farklı indis bildirimi aynı anda yapılıyor demektir. Bu nedenle; yatay ve dikey bölümlerden oluşan bir tabloya göre iki boyutlu diziyi açıklamanın yararı vardır. **A(3,5)** elemanlı olarak tanımlanmış bir A dizisi için aşağıdaki gibi bir tablo düzenlenebilir.

Yukarıda verilen tabloda taralı alanın yerini **S2.** satır ile **K4.** kolonun kesiştikleri alan olarak tanımlarız. Buna göre A dizisinin A(2,4) elemanıdır diye isimlendirebiliriz.

Pascal' da bir dizinin eleman tipi diğer bir dizi olabilir. Örnek olarak Pascal programlama dilinde aşağıdaki gibi bir tanımlama yapılabilir.

```
Type
     matrix= array [1..3] of array[1..5] of integer;
```

Yukarıdaki tanımlamaya göre tanımlanan dizi üç elemandan meydana gelmektedir. Dizinin her bir elemanı yeni bir diziyi oluşturduğunu ve bu dizinin elemanları ise beş elemandan meydana geldiğini söyleyebiliriz. Bu anlatılanı aşağıdaki gibi kısaltarak yazabiliriz.

```
Type
     matrix = array [1..3,1..5] of integer;

     Var
         A : matrix;
```

Yukarıdaki bildirim ile dizinin bir elemanına, **matrix** tipinde (Dolayısı ile integer tipte) **A** ismindeki dizi değişkenin satır ve kolon tamsayı bildirimi ile ulaşılabilir. Örneğin, yukarıda verilen tablodaki koyu renkli alanın yerini **2.** satır **4.**kolon olarak tanımlarız ve **A** dizisinin **A[2][4]** veya **A[2,4]** elemanı şeklinde isimlendirebiliriz. Bu nedenle dizinin tanımlanmasında iki farklı indis ismi oluşuyor demektir.

Aşağıda **b** ismi ile Pascal' a göre iki boyutlu dizi değişkeni tanımlanmıştır. Tanımlanan dizi değişkeninin her bir elemanı 2 byte (integer) büyüklüğündedir.

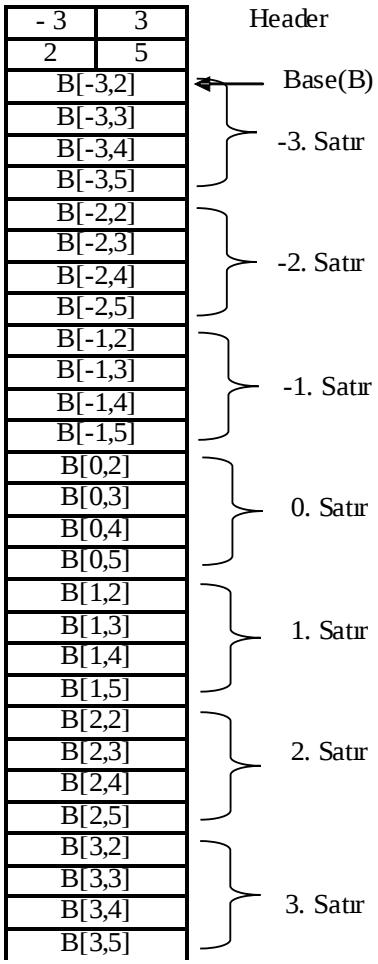
```
var
     b:array[-3..3,2..5] of integer;
```

Aşağıda verilen tabloya göre tanımlanan B dizisinin -3..3 aralığı ile verilen sayısal değerler birinci boyutu (Aşağıda tabloda satırlar olarak gösterilen), 2 .. 5 aralığı ile verilen sayısal değerler ise ikinci boyutu (Aşağıdaki tabloda Kolonlar olarak gösterilen) gösteriyor. Satırlar

veya kolanlardaki elemanlar sayısı, üst sınır değerinden alt sınır değerinin farkından elde edilen sayıya 1 sayısı ilave edilerek elde edilen sayıya eşittir. Bu sayı boyutun alanı diye isimlendirilir. İlk örnekte verilen A dizisindeki ilk boyutun alanı 3 – 1 + 1 = 3'tür ve ikinci boyutun alanı 5 – 1 + 1 = 5 tir.Bu nedenle A dizisi 3 satır ve 5 kolondan oluşmaktadır. B dizisindeki satır sayısı 3 - (- 3) + 1 = 7, ve kolon sayısı 5 – 2 + 1 = 4 'tür.İki boyuttaki dizinin eleman sayısı, satır sayısı ve kolon sayısının sonucuna eşittir. Böylece A dizisi 3 * 5 = 15 elemanı ve B dizisi de 7 * 4 = 28 elemanı kapsamaktadır.

b[-3,2]	b[-3,3]	b[-3,4]	b[-3,5]
b[-2,2]	b[-2,3]	b[-2,4]	b[-2,5]
b[-1,2]	b[-1,3]	b[-1,4]	b[-1,5]
b[0,2]	b[0,3]	b[0,4]	b[0,5]
b[1,2]	b[1,3]	b[1,4]	b[1,5]
b[2,2]	b[2,3]	b[2,4]	b[2,5]
b[3,2]	b[3,3]	b[3,4]	b[3,5]

Her iki boyut için bir index tip tanımlıdır. Bildirilen index tipteki elemanların her birinin büyüklükleri aynıdır ve her bir boyuta ait eleman sayıları toplamı dizinin toplam eleman sayısına eşittir. Dizi için bellekte ayrılan toplam alan, dizi için tanımlanan index tip büyüklüğü ile açıklanır.



Bellekteki iki boyutlu bir dizinin gösteriminin bir metodu ana satır(row - major) gösterimidir. Bu gösterim altında, dizinin ilk satırı, bellekte ayrılan ilk dizi elemanlarının bellek konumlarını tutar. İkinci satır bir sonraki bellekte ayrılan ikinci dizi elemanlarının bellek konumlarını tutar ve böylece devam eder. Böylelikle iki boyutlu dizinin tablodaki yatay ve dikey bildirimlerine karşılık gelen, bir biri ardı sıra gelen bellek konumlarının sayısal olarak isimlendirilmeleri iki boyuta göre ardışık iki tam sayılar ile ifade edilmektedir. Bu nedenle dizinin alt ve üst sınır indis değerleri iki boyut için bellekte bir birini takip edecek şekilde düzenlenirler.

Yukarıdaki şekilde iki boyutlu B isimli dizisinin bellekteki sıralanış yapısına göre verilmiştir. Şekilde ilk boyut indisi alt sınır değeri -3 ve üst sınır değeri 3, ikinci boyut indisi alt sınır değeri 2 ve üst sınır değeri 5 olan bir B dizisinin bellekteki ilk elemanın (B[-3,2] elemanı) yeri base(B) dir. Dizinin ikinci elemanı (B[-3,3]) ise Base(B) + esize konumundadır. B dizisi iki boyutlu olduğuna göre, dizinin herhangi bir elemanının bellek yeri neresidir? Sorusuna karşılık dizinin yatay dikey tablo karşılığına gelen gösterimindeki bir yatayın kaç kolondan ve dizinin tamamının kaç satırdan oluştuğunu bilmek gerekir. Öncelikle, iki boyutlu B gibi bir dizinin genel tanımına bakalım;

```
Var
     B : array[AS1..US1 , AS2..US2] of tip_ismi;
```

Bu tanım ile dizinin, AS1 ve US1 birinci boyutun sınır aralığı , AS2 ve US2 ikinci boyutun sınır aralığı olarak verilmiştir. B dizisinin ilk elemanı base(B)'nın adresi B[AS1,AS2]'dedir. Örneğin şekil de B dizisi için base(B) B[-3,2]' nin adresidir ve şekilde bir yatayı 4 adet kolon değeri ile

tamamladığımız görülmektedir. Bu nedenle bir sonraki yataya geçiş adımlarımızın büyüklüğünü (4 adet kolon değeri) R ile tanımlarsak; dizinin yatayına ait herhangi bir elemanının yeri;

```
Base( B ) + (İ1 – AS1) * R * esize
```

adresindedir.

Böylelikle;herhangi bir B dizi elemanı B[İ1,İ2] olduğu kabulüne göre B dizisinin herhangi bir elemanının adresi ise;

```
Base( B ) + [( İ1 – AS1 ) * R + (İ2 – AS2)] * esize
```

dadır.

Örnek:

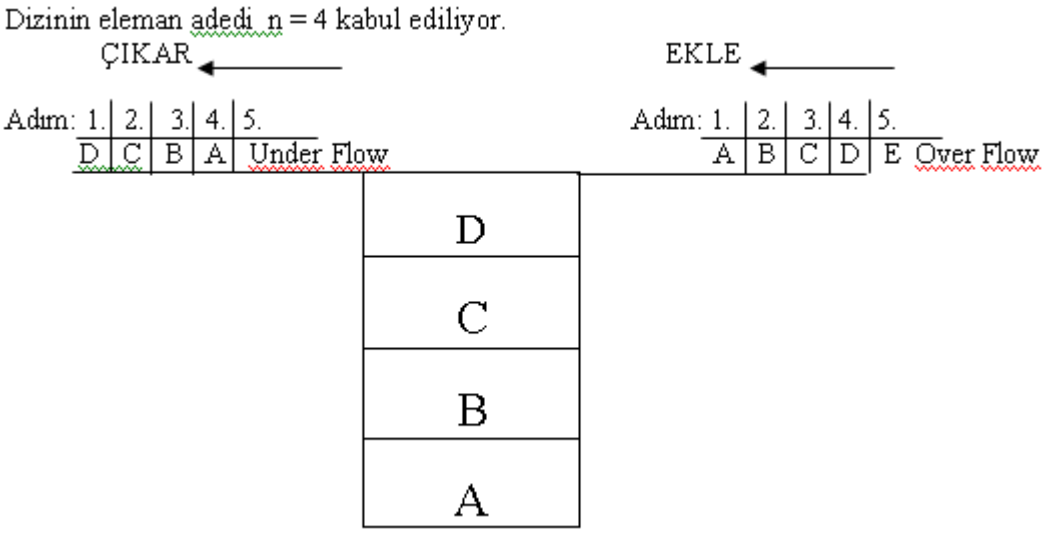
B dizisinin B[2,5] elemanının adresi (B dizisinin her bir elemanı 2 byte olduğu kabul edilirse) hesaplanırsa;

```
Base( B) + [(2 – ( - 3)) * 4 + ( 5 – 2) ] * 2
     = [ (2 + 3) * 4 + ( 3)] * 2
     = [ (5 * 4 + 6]
     = [20 + 6 ]
     = 26 byte’ dır.
```

YIĞIN (STACK**)**

GİRİŞ

Yığın (Stack), Diziler de (Array) olduğu gibi, birbiri ardı sıra gelen bellek konumlarından oluşan veri gruplarının tutulduğu alanlardır. Ancak dizilerden farklı olarak, yığına koyulan veriler, birbiri üzerine gelecek şekilde bellek konumlarında yer alırlar. Bu tür doğrusal listelere örnek olarak, bir biri üzerine yığılmış tabak yığınını gösterebiliriz. Tabak yığınındaki herhangi bir elemanı almak için, o tabağın üzerindeki tüm tabakları yığından kaldırılması gerekmektedir. Buna göre; yığına veri girişi yapıldığında, yığındaki en son elemanın üzerine gelecek şekilde, en son girilen veri yığında yer alır. Yığından eleman çıkarmak istendiğinde, yığının en üstündeki eleman yığından çıkar. Aşağıda verilen şekil incelendiğinde, yığının ilk elemanı A, sonraki elemanları sırasıyla yığının 2. elemanı B, 3. elemanı C ve sonuncu 4. elemanı D elemanı olarak görülmektedir. Yığının 4 elemanlı tanımlanmış olması nedeni ile, beşinci bir elemanı yığına eklemek isterseniz, bu durumda yığında taşma (Over Flow) kontrolu yürütülür. Yığından değer çıkarıldığında, ilk çıkan değer yığının en üstündeki 4. eleman olan D değeridir. Çıkarma işlemine devam ederseniz, sırasıyla yığının 3. elemanı C , 2. elemanı B ve son olarak 1. elemanı olan A değerinin çıkarıldığını görmektesiniz. Bundan sonra çıkarma işlemine devam ederseniz yığında boş (Under Flow) kontrolü yürütülür.



Yukarıda anlatılanların Ekleme ve Çıkarma algoritmaları ve karşılıkları olan Pascal programını adım adım gerçekleştirebilirsiniz.

Yığına Eleman Ekleme (Push) Algoritması

Takip eden satırlarda verilen algoritmada **Top** ismini, yığında herhangi bir indis karşılığında düşünmelisiniz. Bu nedenle **Top** ismi, yığında o andaki en son eleman değerinin yerini gösterecektir. **n** adet eleman tanımlı dizinin, kabul edilebilir maksimum eleman sayısını açıklar. **S(top), top** ismine bağlı olarak **top** elemanlı diziye karşılık gelmektedir.

```
If Top>=n Then
          Print 'Stack Over Flow' & Exit
EndIf
Top ← Top + 1
S(Top) ← Data & Exit
```

Algoritma yazılımında, **If** cümlesindeki **Top** isminin içeriği, **n** isminin içeriğinden büyük veya **n** isminin içeriğine eşit ise, yığında öncelikle **Over Flow** kontrolü yürütüldüğüne dikkat ediniz. **If** cümlesindeki eşitlik veya büyüklük koşulu gerçekleşmemiş ise, bir alt satırda görüldüğü gibi **Top** ismine **1** sayısı eklenir ve yeni **Top** ismi olarak saklanır. Son satırda **Data** ismi ile yığına girilecek değer ne ise, **S(Top)** dizisinin **top'** ıncı yerine girilecektir.

Yukarıda verilen algoritmaya göre; pascal program karşılığını 4 elemanlı **Stack** olduğu kabul ederek, bir procedure alt alanı olarak aşağıdaki satırlardaki gibi yazabilirsiniz.

```
Procedure  Push(Parametre listesi);

  Begin

  If Top>=4 Then
           Begin
             Writeln('Stack Over Flow');
             Exit;
           end
             Else
```

```
                    Begin
                       Top:=Top + 1;
                        S[Top]:= Data;
                    End;
End;
```

Yukarıda verilen ekleme algoritmasının doğruluğunu adım adım test edelim. Aşağıdaki test işlemi ekleme algoritmasına beş adım için uygulanıyor ve **n=4** kabul ediliyor. Beşinci adımda yığının **Over Flow** özelliğinin gerçekleştiğine dikkat ediniz..

TEST(EKLEME) (Beş adım için)

Adım	Top >= n	Top ← Top + 1	S(Top) ← Data
1	0 >= 4 (H)	1 ← 0 + 1	S(1) ← A
2	1 >= 4 (H)	2 ← 1 + 1	S(2) ← B
3	2 >= 4 (H)	3 ← 2 + 1	S(3) ← C
4	3 >= 4 (H)	4 ← 3 + 1	S(4) ← D
5	4 >= 4 (E)	Stack Over Flow & Exit	

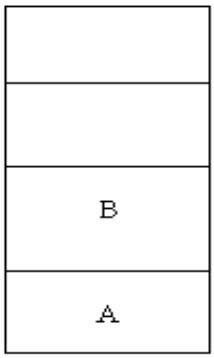
Başlangıç olarak; ilk adımda **Top** isminin içeriği **0** (sıfır) kabul edilerek algoritma işletilmeye başlanıyor. **Top>=n** koşulu gerçekleşmediği için algoritmanın bir alt satırı **Top ← Top + 1** işletime giriyor. Burada **top** içeriği **1** oluyor. Son satırda **S(top) ← Data** ifadesinde **S(1)** yerine **A** değeri atanıyor. Birinci adımdan sonraki yığının durumu aşağıdaki şekilde olduğu gibidir.

EKLE

Adım:	1.	2.	3.	4.	5.
	A	B	C	D	E

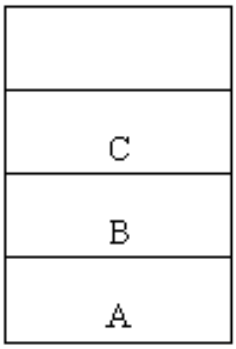
İkinci adımda **Top** değeri **2** oluyor ve yığının ikinci eleman yerine **B** değeri atanıyor. Yığına ikinci değerin eklenmesinden sonraki durumu takip eden şekilde görüyorsunuz.

EKLE
Adım: 1. 2. 3. 4. 5.
A B C D E



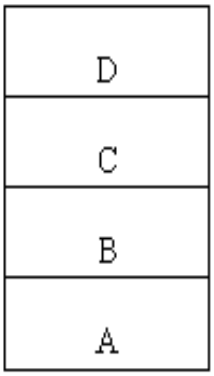
Üçüncü adımda **Top** değeri **3** oluyor ve yığının üçüncü eleman yerine **C** değeri atanıyor. Yığına üçüncü değerin eklenmesinden sonraki durumu aşağıdaki şekilde görüyorsunuz.

EKLE
Adım: 1. 2. 3. 4. 5.
A B C D E



Dördüncü adımda **Top** değeri **4** oluyor ve yığının dördüncü eleman yerine **D** değeri atanıyor. Yığına dördüncü değerin eklenmesinden sonraki durumu aşağıdaki şekilde görüyorsunuz.

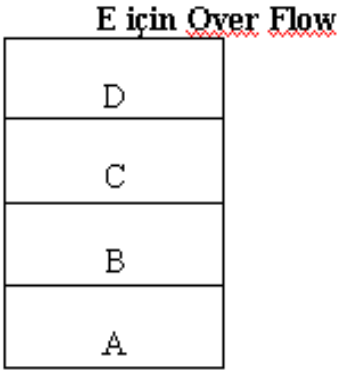
EKLE
Adım: 1. 2. 3. 4. 5.
A B C D E



Beşinci adımda **Top** değerinin **4** olması nedeni ile **Top < = n** koşulunu sağladığından ekleme işlemi için **5. adımda** çalıştırmak istediğimiz algoritmada **Over Flow** kontrolünün yürütüldüğünü görmekteyiz. Beşinci adımdan sonraki durumu aşağıdaki şekilde görüyorsunuz.

EKLE

Adım:	1.	2.	3.	4.	5.
	A	B	C	D	E



Bu durumda, yığına yeni bir elemanın eklenemeyeceğini anlıyoruz.

Yığından Eleman Çıkarma (POP) Algoritması

Aşağıdaki algoritmada yığından eleman çıkarmak için öncelikli işlem **Top** ismine bağlı olarak yığının **Under Flow** özelliğini kontrol etmektir. Bu nedenle **Top<=0** koşulu **If** cümlesi ile gerçekleşip gerçekleşmediğinin kontrolü yürütülüyor. Koşul gerçekleştiğinde **UnderFlow** kontrolü yapılıp yazılımdan çıkılır. Koşul gerçekleşmemiş ise bir alt satır işletime alınır. **POP ← S(Top)** satırındaki **S** dizisinin **Top'** ıncı yerindeki değer çıkarılarak **POP** ismine atanıyor ve sonuncu satırda **Top ← Top – 1** satırı ile **Top** isminin içeriğinden sayısal **1** (bir) değeri çıkarılarak yeni değer olarak **Top** ismine atanır. Böylece, yığından bundan sonra çıkacak değerin yerini **top** isminde tutmuş oluyoruz.

```
If Top<=0 Then
          Print ‘ Stack Under Flow’ & Exit
EndIf
POP ← S(Top)
Top ← Top -1 & Exit
```

Yukarıda verilen algoritmaya göre; pascal program karşılığını, 4 elemanlı **Stack** olduğunu kabul ederek, bir **`Function`** alt alanı olarak aşağıdaki satırlardaki gibi yazabilirsiniz.

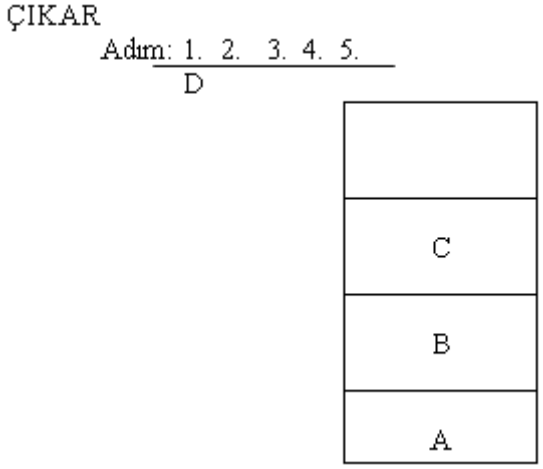
```
Function pop(Parametre listesi):integer;

  Begin
       If top<=0 Then
                  Begin
                   Writeln('Stack Under Flow');
                   Exit;
                  End
                Else
                  Begin
                       Pop := S[top];
                       top := top – 1;
                  End;
  End;
```

Yığından eleman çıkarma işleminde anlatılanların doğruluğunu sağlayabilmek için aşağıdaki test işlemini uygulayalım.

TEST(ÇIKARMA) (Beş adım için)

Adım	Top < = 0	Pop ← S (Top)	Top ← Top – 1
1	4< = 0(H)	D ← S(4)	3 ← 4 – 1
2	3<= 0(H)	C ← S(3)	2 ← 3 – 1
3	2<= 0(H)	B ← S(2)	1 ← 2 – 1
4	1<= 0(H)	A ← S(1)	0 ← 1 – 0
5	0<=0(E) Stack Under Flow & Exit		

Başlangıç olarak; ilk adımda **Top** isminin içeriği **4** kabul edilerek algoritmanın ilk satırındaki **Top<=0** koşulu gerçekleşmediği için algoritmanın bir alt satırında **POP ← S(Top)** işletime giriyor. Burada **POP** ismine yığının **S(4)** eleman değeri olan **D** değeri **POP** ismine atanıyor. Bir sonraki satır olan son satırda **Top ← Top – 1** ifadesinde yığın geriye kalan en son elemanın indis değerini ve dolayısı ile en son elemanın yeri bulunuyor. Bunun anlamı; bundan sonraki çıkarmada çıkacak elemanın ne olduğunu görüyoruz demektir. Yığından ilk değer çıkarıldıktan, yani ilk adımdan sonraki yığın durumunu aşağıdaki şekilde görmektesiniz.



İkinci adımda; birinci adımdan sonra **Top** değeri bir eksilerek **3** sayısına düşüyor. Bu nedenle ikinci adım sonunda yığının **top'** uncu yani **3** yerinden **POP** ismine **C** değeri atanarak yığından çıkarılmış olur.

Çıkarma işleminden sonraki yığının son durumu aşağıdaki şekilde görüyorsunuz.

ÇIKAR

Adım:	1.	2.	3.	4.	5.
	D	C			

Üçüncü adımda: ikinci adımdan sonra **Top** değeri bir eksilerek **2** sayısına düşüyor. Bu nedenle üçüncü adım sonunda yığının **top'** uncu yani **2** yerinden **POP** ismine **B** değeri atanarak yığından çıkarılmış olur.

Çıkarma işleminden sonraki yığının son durumu takip eden şekilde görüyorsunuz.

ÇIKAR

Adım:	1.	2.	3.	4.	5.
	D	C	B		

Dördüncü adımda; üçüncü adımdan sonra **Top** değeri bir eksilerek **1** sayısına düşüyor. Bu nedenle dördüncü adım sonunda yığının **top'** uncu yani **1** yerinden **POP** ismine **A** değeri atanarak yığından çıkarılmış olur. Çıkarma işleminden sonraki yığının son durumunu aşağıdaki şekilde görüyorsunuz.

ÇIKAR

Adım:	1.	2.	3.	4.	5.
	D	C	B	A	

Beşinci adımda; dördüncü adım sonunda **Top** değeri bir eksilerek **0** sayısına düşüyor olması nedeni ile tekrar çıkarma işlemi için algoritma bir adım daha (5. adım için) çalıştırıldığında **if** cümlesindeki **Top < = 0** koşulunu sağladığından çıkarma işlemi için **5. adımda** yığın **Under**

Flow kontrolünü yürütecektir. Beşinci adımdan sonraki durumu aşağıdaki şekilde görüyorsunuz.

ÇIKAR
Adım: 1. 2. 3. 4. 5.
D C B A **Stack Under Flow**

Buraya kadar algoritma ile anlatılanları pascal program karşılığını tasarlayabilmek için öncelikli işlem pascal' da Record (Kayıt) ve Set (Küme) yapıları tanımanız gerekmektedir. Aşağıdaki ilerleyen satılarda bu yapıların tanım ve özelliklerine kısaca değinilecek ve örneklendirmelerle basit olarak açıklanacaktır.

PASCAL' da **Record ve Set yapılar**

Record (Kayıt**) yapılar**

Pascal' da **Type** tanım bloğu kullanarak, standart tiplerin (Integer, Real, String gibi) dışında, kendi değişken tipinizi yaratabilirisiniz. İlk tip yaratmayı, Record yapılar olarak düşünebiliriz. Record yapılar, iki veya daha fazla farklı tipteki değişkenlerin, bir yapı altında toplanmasından meydana gelir. Record yapı da bir öğrencinin ismini ve numarasını birlikte nasıl kullanacağınızı aşağıda verilen Pascal örneğinde görmektesiniz.

```
Program RecordTipler;

   Type
      ogrencitipi = Record
                Numara : Integer;
                isim : String;
                   end;
   begin
   end.
```

Yukarıda verilen örnekten sonra, değişken tanım bloğunda (Var) tanımlanacak bir değişken ismini, **ogrencitipi** tip ismi ile tanımlayarak, uygulama bloğunda kullanabilme durumunu gerçekleştirmiş olursunuz. Aşağıdaki Pascal yazılım örneğinde görüldüğü gibi.

```
Program RecordTipler;

   Type
      ogrencitipi = Record
                numara : Integer;
                isim : String;
                     end;
   var
      ögrenci: ogrencitipi;

   begin
   end.
```

Record yapıda alt alan olarak tanımlanan, öğrenciye ait numara ve isim alanlarına, aşağıda verilen Pascal program örneğinde olduğu gibi, sayısal ve alfa sayısal değerleri program içinden atama yolu ile girebilirsiniz.

```
Program RecordTipler;

  Type
      ogrencitipi = Record
                     numara : Integer;
                     isim : String;
                    end;

   var
         ögrenci: ogrencitipi;

      begin
         ogrenci.Numara := 12345;
         ogrenci.isim := 'Rıdvan ŞEN';
      end.
```

Record yapıda alt alan olarak tanımlanan alanlara, aşağıda verilen Pascal program örneğinde olduğu gibi, sayısal ve alfa sayısal değerleri program dışından, veri satırından değer okutabilirsiniz.

```
Program RecordTipler;

      Type
           ogrencitipi = Record
                          numara : Integer;
                          isim : String;
                         end;

      var
           ögrenci: ogrencitipi;

      begin
```

```
            Write('Örenci NumarasınıVeriniz..:');
            Readln(ogrenci.Numara);
            Write('Örenci ismini Veriniz...:');
            Readln(ogrenci.isim);

        end.
```

Record yapıda alt alan olarak dizi tanımlanabilir ve tanımlanan dizinin alt alanlarına, aşağıda verilen Pascal program örneğinde olduğu gibi, ayrı veri satırından değer okutabilirsiniz.

```
Program RecordTipler;

      Type
        dizitipi = Record
                     dz:Array[1..5] of Integer;
                   end;

      var
         dizi: dizitipi;
            i:byte;
      begin
           for i:=1 to 5 do

      Readln(dizi.dz);

      end.
```

İki farklı veya daha fazla tip ismi ile aynı Record yapı aşağıdaki gibi tanımlanabilir.

```
Program RecordTipler;

      Type
          ogrencitipi,iscitipi = Record
                                   numara : Integer;
                                   isim : String;
                                 end;
      var
         ögrenci: ogrencitipi;isci:iscitipi;

      begin

         ogrenci.Numara := 12345;
         ogrenci.isim := 'Rıdvan ŞEN';

     isci.Numara := 54321;
          isci.isim := 'Hakan ŞUMNULU';

      end.
```

İki farklı veya daha fazla tip ismi ile farklı Record yapı aşağıdaki gibi tanımlanabilir.

```
Program RecordTipler;

     Type
         ogrencitipi = Record
                     numara : Integer;
                     isim : String;
                        end;
          iscitipi = Record
                     numara : Integer;
                     isim : String;
                      end;

     var
            ögrenci: ogrencitipi;isci:iscitipi;

     begin
            ogrenci.Numara := 12345;
            ogrenci.isim := 'Rıdvan ŞEN';
            isci.Numara := 54321;
            isci.isim := 'Ayşe GÜL';

     end.
```

Set (Küme**) yapılar**

Bir diğer yapı, **set** (Küme) yapılarıdır. Küme yapıları, record yapıdan daha kolay uygulanabilir yapı olmakla beraber, çok kullanışlı değillerdir. Takip eden satırlarda verilen Pascal program örneğinde, küme (Set) tipi ile kaplan, kedi ve aslan isimlerinin depolandığı hayvan tip isimlendirmesinin nasıl yapıldığını görmektesiniz.

```
Program SetTipler;
     Type
         Hayvantipi = set of (Kaplan, Kedi, Aslan);

     var
        hayvan :Hayvantipi;

     begin
        Hayvan := dog;
     end.
```

Yukarıda verilen Pascal programında çok yararlı olmayan yol olan set yapıda Readln veya Writeln kullanamayabilirsiniz. Aşağıdaki Pascal program örneği ile 'a' ve 'z' karakter aralığında küme tipi oluşturabilir ve belirlenen sınırlar içerisinde harfleri kullanabilmeyi test edebilirsiniz.

```
program SetTipler;

      Type
         Alfabe = 'a'..'z';

      var
         Harf: Set of Alfabe;
         c: Char;

      begin
         c := ReadKey;
         if c in [harf] then
                  Writeln('Girdiğiniz Harf..:',c);
      end.
```

Aşağıdaki Pascal program örneği, 0 ve 9 sayıları aralığında set yapı oluşturabilir ve belirlenen sınırlar içerisinde sayıları kullanabilmeyi test edebilirsiniz.

```
program SetTipler;
      Type
         sayitipi = 0 . .  9 ;
      var
         sayi: Set of sayitipi;
         c: byte;
       begin
         c := ReadKey;
         if c in [sayi] then
                 Writeln('Girdiğiniz Harf..:',c);
       end.
```

Yukarıda Pascal tanımlı yapıları örnekleri ile uygulayarak öğrendiğinize göre, buraya kadar anlatılan "*Sıradan Bellek Konumlu Doğrusal Listeler*" adı altında Yığıt (Stack) konu başlığı ile anlatılan algoritmadaki ve karşılığı pascal programlamadaki ekleme ve çıkarma işlemlerini **Record** yapı oluşturarak gerçekleştirmek isterseniz, aşağıdaki satırları yazmalısınız. Record yapı içinde , **item** ve **top** ismi ile iki adet alt alan tanımlaması yapılmıştır. Bu alanlardan **item** 4 elemanlı dizi değişken isim, diğeri ise 0 – 4 sayısal aralığı temsil eden değişken isim özelliklerinde olan isimlerdir. Aşağıda bu özelliği görmektesiniz.

```
Const
      Maxstack=4;
Type
      Stack = Record
             item : Array[1..maxstack] of integer;
             top  : 0 .. maxstack;
            End;
```

Yapılan tanımlamada record yapının ismi **stack** ismi olarak bizim tarafımızdan belirlendi. **Stack** ismi şu andan başlayarak bir tip ismi olma nedeni ile değişken tanım bloğunda **stack** ismini kullanarak bir değişken tanımlayabilirsiniz. Tanımlanacak bu değişken ile, record yapının alt alanlarındaki tanımlanmış olan **item** ve **top** isimli değişkenlere ulaşabilirsiniz. Aşağıdaki

satırda olduğu gibi, değişken tanım bloğu olan **var** bloğunda, **stk** isminde, tipi **stack** olan bir değişkeni tanımlayabilirsiniz.

```
Var
   Stk:Stack;
```

Yukarıdaki satırda olduğu gibi, **Stk** isimli değişkeni tanımlamakla, size pascal programınızın ana uygulama alanından record yapının alt alanları olan **item** ve **top** isimli değişken alanlarına ulaşmanızı sağlayacaktır.

Pascal programlamada, alt program olarak düzenlenen (Procedure, function gibi) yapılarda da yukarıda record yapıda tanımlanan **stack** tip ismi ile parametreleri tanımlayabilir ve aynı record yapının alt alanlarına ulaşabilirsiniz. Bu alanlar ile ilgili işlemler gerçekleştirebilirsiniz.

Aşağıda verilen örnekte olduğu gibi, **Push** ismi ile bir **procedure** alt alanın başlık kesiminde, **st** ismindeki parametrenizi **stack** tip ismi ile tanımlayabilirsiniz. Böylece record yapının alt alanlarındaki **item** ve **top** isimli alanlara, tanımlanan **procedure** alt program alanından ulaşmış olursunuz. Aşağıdaki **push** isimli alt program, ekleme algoritması karşılığına gelen pascal programıdır.

```
Procedure push(var stpsh:stack; veri:integer);
Begin
      If stpsh.top=maxstack Then Writeln('Stack Over Flow')
                          Else Begin
                                  stpsh.top:=stpsh.top + 1;
                                  stpsh.item[stpsh.top]:= veri;
                               end;
End;
```

Ana program uygulama alanından, **Push** isimli alt programı işletilmek üzere çağırdığınızda, **var** bloğunda tanımlı **stk** gerçek değişkeni üzerinden, alt programın **st** isimli parametresi ile transfer işlemini gerçekleştirmiş olursunuz. Alt programdaki **veri** isimli parametre karşılığında, ana programda tanımlayacağınız gerçek değişken ismi ile de yığına eklenecek değerin ne olacağına karar vermiş olursunuz.

Pascal' da yığından değer çıkarma işlemini gerçekleştiren, bir alt programı **function** olarak düşünebilirsiniz. Aşağıdaki gibi, ismi **Pop** olan bir alt program örneğini yazabilirsiniz.

```
Function pop(var stpop:stack):integer;
Begin
      If top<=0 Then

                  Writeln('Stack Under Flow')

                  Else
                    Begin
                             Pop := stpop.item[stpop.top];
                             stpop.top := stpop.top – 1;
                     End;
End;
```

Yığının altprogramlarında, **Overflow** veya **UnderFlow** kontrolünü ayrı bir altprogram ile kontrol etmek isteyebilirsiniz. Bunun için aşağıdaki gibi bir function yazabilirsiniz.

```
Function bos(sbos:stack):boolean;

Begin
      İf sbos.top=0 Then
                       Bos:=true
                     Else
                       Bos:=false;
End;
```

Tanımlanan fonksiyonun çalışma testini gerçekleştirebilmeniz için, aşağıda verilen komut cümlesini işletime koyacağınız uygun komut satırından işletebilirsiniz.

```
If bos(stk) Then
              { Stack boş iletisi burada verilecek}
            Else
              { Stack'ın bos olmadığına ait uygulama}
                  {burada gerçekleştirilecek}
```

INFIX, SUFFIX VE PREFIX İFADELER

Matematiksel ifadelerde operator, iki operand arasında kullanılmış ise bu tür ifadelere **infix** ifadeler denir. Örneğin, A + B ifadesi **infix** ifadedir. **Infix** ifadelere karşı A ve B toplamının kullanım ifadesi için iki alternativ vardır. Bunlar **Prefix** ve **Postfix** ifadelerdir.

Operator, iki operand' ın sağında kullanılmış ise bu tür ifadelere **suffix** ifadeler denir. Örneğin, AB+ ifadesi suffix ifadedir.

Operator, iki operand' ın solunda kullanılmış ise bu tür ifadelere **prefix** ifadeler denir. Örneğin, +AB ifadesi suffix ifadedir.

İnfix ifadelerin Suffix veya Prefix ifadelere dönüştürebilirsiniz. Dönüştürmelerle ilgili örnek uygulamaları takip eden satırlarda verilen tablolardan inceleyebilirsiniz.

INFIX ifadenin SUFFİX ifadeye dönüştürülmesi

INFIX	SUFFIX-1	SUFFIX-2	SUFFIX-3
A + B	A B +		
A + B + C	(A B +) + C	A B + C +	
A + (B + C)	A + (B C +)	A B C + +	
A + B * C	A + (B C *)	A B C * +	
A * (B + C)	A * (B C +)	A B C + *	
A * B * C	(A B *) * C	A B * C *	
A + ((B * C) * D)	A + ((B C *) * D	A + (BC * D *)	A B C * D * +

INFIX ifadenin PREFİX ifadeye dönüştürülmesi

INFIX	PREFİX-1	PREFİX-2	PREFİX-3
A + B	+ A B		
A + B + C	(+ A B) + C	+ + A B C	
A + (B + C)	A + (+ B C)	+ A + B C	
A + B * C	A + (* B C)	+ A * B C	
A * (B + C)	A * (+ B C)	* A + B C	
A * B * C	(* A B) * C	* * A B C	
A + ((B * C) * D)	A + ((* B C) * D	A + (* * B C D)	+ A * * B C D

SUFFIX ifadenin INFIX ifadeye dönüştürülmesi

Suffix ifadenin Infix ifadeye dönüştürülmesi için aşağıdaki adımlar uygulanır.

1. İfadenin en solundaki operatör bulunur.
2. Bu operatörün işlemi solundaki iki operanda uygulanır.
3. Sonuç yeni bir operatör olarak saklanır.

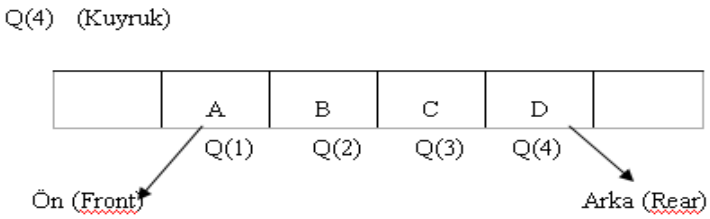
Aşağıdaki örneği inceleyiniz.

SUFFIX	INFIX - 1	INFIX - 2	INFIX - 3
A B C * D * +	A (B * C) D * +	A ((B * C) * D) +	A + ((B * C) * D)

KUYRUKLAR (QUEUES**)**

Kuyrukta bir elemana erişim; bir uçtan (ön) eleman çıkarma işlemlerinin, diğer uçtan (arka) da eleman ekleme işlemlerinin yapıldığı doğrusal listelerdir. Tanımlanan bir dizide elemanlar aynı özellikte, aynı büyüklükte, yan yana veya birbiri ardı sıra gelen bellek alanlarında bulunabilen sıradan bellek konumlu listelerdir.

Bu tür doğrusal listelere örnek; bir otobüs durağındaki sırada bekleyen yolcuları gösterebiliriz. Otobüs durağında sıraya ilk giren yolcu otobüse ilk binecektir (sıranın önü). Sıraya en son giren yolcu ise en son binecektir (sıranın sonu). Yada bir bilgisayar sistemindeki yazıcıdan çıktısı alınmak istenen dosyaların sistem tarafından sıraya konulması gibi. Aşağıdaki şekil tanımlanan 4 elemanlı bir Q dizisinin durumunu göstermektedir.



Bir kuyrukta eleman ekleme ve eleman çıkarma işlemleri, ***İlk Giren ilk Çıkar*** (FIFO - First In First Out) ifadesi ile tanımlanır. Bu nedenle, kuyruktaki bilgilerin oluşturulması veya silinmesi, kuyruğun önünde veya arkasında bulunan elemanın bulunduğu yere bağlı olarak, tanımlanacak indis isimleri ile izleyebilir ve uygulamaya koyabilirsiniz. Kuyrukta eleman ekleme işleminde, o anda sona eklenen elemanın yerini, tanımlanan arka indis ismi ile yerini belirleyebilirsiniz. Kuyruktan eleman çıkarma işleminde, o anda önden çıkarılan elemandan sonraki yeri, tanımlanan ön indis ismi ile eleman yerlerini takip edebilirsiniz. Çünkü eleman çıktıktan sonra, önden bir sonraki eleman, öndeki ilk eleman durumunda oluyor. Yığında olduğu gibi kuyrukta da ekleme veya çıkarma işlemlerini yürütürken, kuyrukta **Over Flow** veya **Under Flow** kontrolünü yapmak zorundasınız.

Kuyrukları iki başlık altında inceleyebilirsiniz. Bunlar;

- Basit Yapılı kuyruk
- Dairesel Yapılı kuyruk

Takip eden satırlarda Basit Yapılı ve Dairesel Yapılı kuyrukların yapılarını, davranışlarını ve özelliklerini algoritma, karşılığı pascal yazılımları ve test işlemleri ile incelenecektir.

Basit Yapılı Kuyruk

Basit yapıda, eleman ekleme veya çıkarma işlemleri, tanımlanan n elemanlı bir kuyruğun **n.** inci elemanın varlığına veya kuyruğun ön indis değerinin sıfır olmadığına göre yapılmaktadır. Arka indis ismi kuyruğun maksimum indis değerine eşit veya maksimum indis değerinden büyük olması durumunda dizide her zaman **overflow** kontrolü yürütülür. Ön indis isminin 0(sıfır)a eşitliğinde ise **Underflow** kontrolü yürütülmektedir. Aşağıdaki satırlarda basit yapılı kuyruğa ait ekleme ve silme algoritmaları görülmektedir. Algoritmalarda kullanılan f=ön indis ismi, r=arka indis ismi, Q=kuyruk, n=maksimum eleman sayısı, Veri= Kuyruğa eklenen bilgiyi ifade etmektedir.

Basit Yapılı Kuyruğa Eleman Ekleme (Insert) Algoritması

Aşağıda verilen algoritmada **r** ismi, kuyruğun arka indisi karşılığında düşünmelisiniz. Bu nedenle **r** kuyruktaki en arkadaki değerin yerini gösterecektir. **n,** kuyrukta kabul edilebilir maksimum eleman sayısını göstermektedir. **Q(r); r** ismine bağlı olarak **r** elemanlı diziye karşılık gelmektedir.

Aşağıdaki yazılımda **If** cümlesindeki **r** isminin içeriği **n** isminin içeriğinden büyük veya **n** isminin içeriğine eşit ise kuyrukta öncelikle **Over Flow** kontrolü yürütüldüğüne dikkat ediniz. **If** cümlesindeki eşitlik veya büyüklük koşulu gerçekleşmemiş ise, bir alt satırda görüldüğü gibi **r** ismine **1** sayısı eklenir ve yeni **r** ismi olarak saklanır. **Q(r) ← Veri** satırda **Veri** ismi ile kuyruğa girilecek değer ne ise **Q(r)** dizisinin **r'** inci yerine eklenecektir. Algoritmanın son satırında **f=0** karşılaştırmasında ilk anda kuyruğun boş olması nedeni ile ilk verinin eklenip

eklenmediği kontrolü yürütülerek, **f** ismi **1** sayısına düzenlenir ki işlemlerin ilerleyen durumlarında çıkarma işlemi yürütülecek ise nereden çıkarılacağının da tespiti yapılmış olur.

```
If  r >=n then print 'Over Flow' & Exit
r←r+1
Q(r) ←Veri
if f=0 then f ←1 & Exit
```

Yukarıda verilen algoritmaya göre; pascal program karşılığını, 4 elemanlı **Basit Yapılı kuyruk** olduğunu kabul ederek, bir **procedure** alt alanı oluşturabilirsiniz. Alt alana değerler tanımlayacağınız parametreler aracılığı ile taşıyabilecek şekilde aşağıdaki pascal program satırlarını yazabilirsiniz.

Procedure Ekle(Parametre listesi);

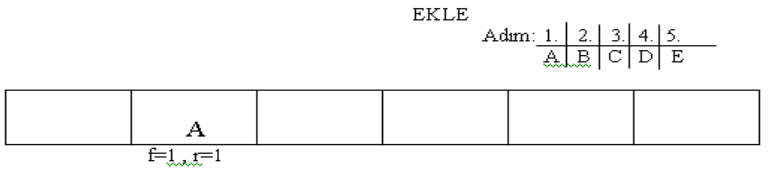
```
Begin
 If  r >=4 then
            Begin
             Writeln('Over Flow');
             Exit;
            End
           Else
            Begin
             r := r + 1;
             Q[r] := Veri;
            End;
  if f=0 then f:=1;
 End;
```

Yukarıda verilen ekleme algoritmasının ve karşılığı pascal programının doğruluğunu aşağıda verilen tabloda adım adım test edelim. Aşağıdaki test işlemi ekleme algoritmasına ve karşılığı pascal yazılımına beş adım için uygulanıyor. Tabloda **n=4** kabul ediliyor. Beşinci adımda kuyruğun **Over Flow** özelliğinin gerçekleştiğine dikkat ediniz..

TEST :EKLEME (n=4 kabul edilsin)

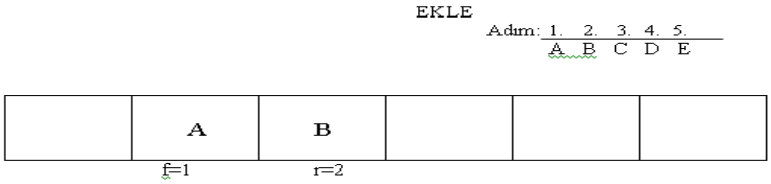
Adım	Veri	r>=n	r←r+1	Q(r) ←Veri	f = 0	f←1
1	A	0 >= 4 (H)	1← 0 + 1	Q(1)← A	0 = 0 (E)	1 ← 1
2	B	1 >= 4 (H)	2←1 + 1	Q(2)← B	1 = 0 (E)	- -
3	C	2 >= 4 (H)	3← 2 + 1	Q(3)← C	1 = 0 (E)	--
4	D	3 >= 4 (H)	4← 3 + 1	Q(4)← D	1 = 0 (E)	--
5	E	4 >0= 4 (E)	OverFlow&Exit			

Başlangıç olarak; Birinci adımda **f** ve **r** ismlerinin içerikleri **0** (sıfır) kabul edilerek algoritma işletilmeye başlanıyor. **r >= n** koşulu gerçekleşmediği için algoritmanın bir alt satırı **r ← r + 1** işletime giriyor. Burada **r** içeriği **1** oluyor. Üçüncü satırdaki **Q(r) ← Veri** ifadesinde **Q(1)** yerine **A** değeri atanıyor. Son satırda **f = 0** satırında **f** isminin içeriği başlangıçta sıfır olması nedeni ile koşul sağlanmış oluyor ve **f** içeriği **1** sayısına düzenleniyor. Böylelikle kuyruğa ilk değer eklendikten sonra ön ve arka indislerin değeri **1** oluyor. Birinci adımdan sonraki kuyruk durumunu aşağıdaki şekilde görmektesiniz.

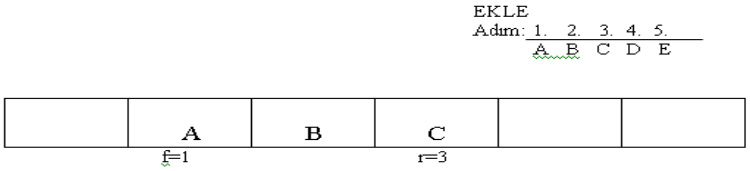


İkinci adımda **r** değeri **2** oluyor, kuyruğun ikinci elemanı **Q(r)← veri** ifadesine uygun olarak Q dizisinin **2. (Q(2))** yerine **B** değeri atanıyor. **f=1** durumunu koruduğuna dikkat ediniz.

Kuyruğa ikinci değerin eklenmesinden sonraki durumu aşağıdaki şekilde görüyorsunuz.

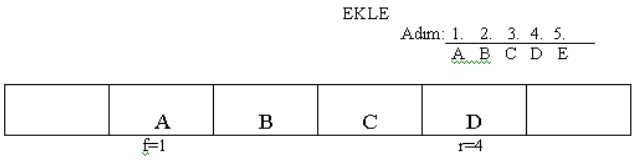


Üçüncü adımda **r** değeri **3** oluyor, kuyruğun üçüncü elemanı **Q(r)← veri** ifadesine uygun olarak Q dizisinin **3. (Q(3))** yerine **C** değeri atanıyor. Üçüncü adımda da **f=1** durumunu koruduğuna dikkat ediniz. Kuyruğa üçüncü değerin eklenmesinden sonraki durumu aşağıdaki şekilde görüyorsunuz.

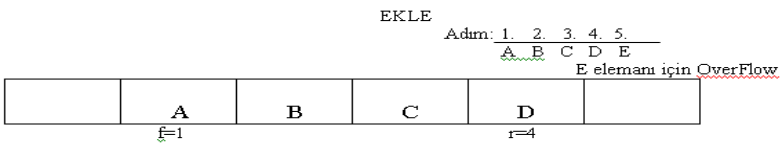


Dördüncü adımda **r** değeri **4** oluyor, kuyruğun dördüncü elemanı **Q(r)← veri** ifadesine uygun olarak Q dizisinin **4. (Q(4))** yerine **C** değeri atanıyor. Dördüncü adımda da **f=1** durumunu koruduğuna dikkat ediniz.

Kuyruğa dördüncü değerin eklenmesinden sonraki durumu aşağıdaki şekilde görüyorsunuz.



Beşinci adımda; **r >= n** koşulu, **r** değerinin **4, n** değerinin de **4** olması nedeni ile kuyruk **Over Flow** kontrolünü yürütüyor ve bloktan çıkılıyor. Beşinci adımdan sonra, **f=1** durumunun korunduğuna dikkat ediniz. Kuyruğun **OverFlow** kontrolünden sonraki durumu aşağıdaki şekilde görüyorsunuz.



Kuyruğa Eleman Ekleme İşleminde Bellek Görünümü

Aşağıda **f** ve **r** indis isimlerine göre adım adım kuyruğun bellek görünümünü tablo halinde görmektesiniz.

Adım	f=0	r=0				
1	f=1	r=1	A			
2	f=1	r=2	A	B		
3	f=1	r=3	A	B	C	
4	f=1	r=4	A	B	C	D

PASCAL' da Record yapı kullanarak Basit Yapılı Kuyruğa eleman ekleme

Pascal da kuyrukları tanımlarken; kuyruk elemanlarını taşıyan bir dizi (Array) kullanılmıştı. Bu nedenle, **dizi** ismindeki dizi 4 elemanlı ve aynı zamanda kuyruğun ilk ve son eleman değerlerinin yerini gösteren **on**, **arka** indis isimleri tanımlanan bir aralıkta (0-4 sayıları aralığı) indis değerlerini tutmaktadır. Bu tanımlamalar ile bir record yapının alt alanlarını aşağıdaki pascal satırlarını yazarak oluşturabilirsiniz.

```
Const
      Maxq=4;
Type
      Kuyruk=Record
            dizi : Array [1 . . Maxq] of integer;
            On, arka : 0 . . Maxq;
            End ;
```

Takip eden satırda, tanımlanan record yapının alt alanlarına ulaşacak değişken ismi tanımlamasının nasıl yapıldığını görmektesiniz.

```
Var
    Kyrk: Kuyruk;
```

Var bloğunda **Kyrk** değişken ismi record yapının tip ismi ile tanımlandığından, **kyrk** değişkeni ile ana program alanından, record yapının alt alanlarına veri işleyebilirsiniz. Aşağıdaki verilen pascal satırlarını uygularsanız record yapının alt alanlarına veri girişini gerçekleştirmiş olursunuz.

```
Kyrk . arka := kyrk . arka + 1;
Kyrk . dizi [Kyrk . arka ]:= veri;
```

Kuyruktan bir elemanın değerini çıkarmak istiyorsanız, sırası ile aşağıda verilen satırlardaki gibi bir yazılım gerçekleştirebilirsiniz.

```
Veri := Kyrk . dizi [Kyrk . on ];
Kyrk .on := Kyrk . on + 1;
```

Aşağıda verilen Ekle isimli procedure alt programını uygularsanız, 4 elemanlı bir kuyruğa record yapının alt alanlarına veri girişini gerçekleştirirsiniz.

```
Procedure Ekle(Parametre listesi);
          Begin
            If  Krk.arka>=4 then
                      Begin
                         Writeln('Over Flow');
```

```
                    Exit;
                End

                Else

                Begin
                 Krk.arka := krk.arka + 1;
                 Krk.dizi[krk.arka] := Veri;
                End;
          if krk.on=0 then krk.on:=1;
        End;
```

Pascal da yukarıda verilenlere göre; istenildiğinde ekleme işlemi, istenildiğinde çıkarma işlemi yapabilen basit yapılı bir kuyruğun programını algoritmaları da göz önünde tutularak yazabilirsiniz.

Kuyruktan Eleman Silme (Delete) Algoritması

Aşağıdaki algoritmada, kuyruktan eleman çıkarmak için öncelikli işlem, **f** ismine bağlı olarak kuyruğun **Under Flow** özelliğini kontrol etmektir. Bu nedenle, **f <= 0** koşulu **If** cümlesi ile gerçekleşip gerçekleşmediğinin kontrolü yürütülüyor. Koşul gerçekleştiğinde, **UnderFlow** kontrolü gerçekleşip yazılımdan çıkılır. Koşul gerçekleşmemiş ise, bir alt satır işletime alınır. **Veri ← Q(f)** satırındaki **Q** dizisinin **f'** ıncı yerindeki değer çıkarılarak **Veri** ismine atanıyor. Bir sonraki satırda **f = r** koşulunun gerçekleşip gerçekleşmediği sorgulanıyor. Koşul gerçekleşmiş ise **f** ile **r** sıfır değerine düzenleniyor ve bloktan çıkılıyor**. f = r** koşulu gerçekleşmemiş ise son satır işletime alınır ve **f** değeri bir arttırılarak yeni **f** değeri olarak tutulur. **f** isminin yeni bir değeri tutması bir sonraki adımda kuyruktan çakacak ilk elemanın ne olduğunu size göstermektedir.

```
If  f=0 then print 'Under Flow' & Exit
Veri ← Q(f)
if f=r then set f ←r←0 & Exit
f←f+1 & Exit
```

Yukarıda verilen algoritmaya göre; pascal program karşılığını, 4 elemanlı **Basit Yapılı kuyruk** olduğunu kabul ederek, silme işlemini yapan **sil** isimli bir **Function** alt alanı oluşturabilirsiniz. **Function** Alt alanı ile veri alış verişini, tanımlayacağınız parametreler aracılığı ile gerçekleştirebilecek şekilde, aşağıdaki pascal program satırlarını yazabilirsiniz.

function Sil(Parametre listesi):tipismi;

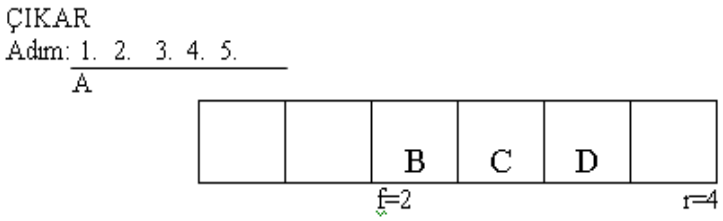
```
Begin
     If  r >=4 then
                Begin
                 Writeln('Under Flow');
                 Exit;
                End;
      sil:=Q(f);
     if f=r then
            Begin
                f := 0;
                r := 0;
                exit;
            end;
     f := f + 1;
End;
```

Kuyruktan eleman çıkarma işleminde anlatılanların doğruluğunu sağlayabilmek için aşağıdaki test işlemini uygulayalım.

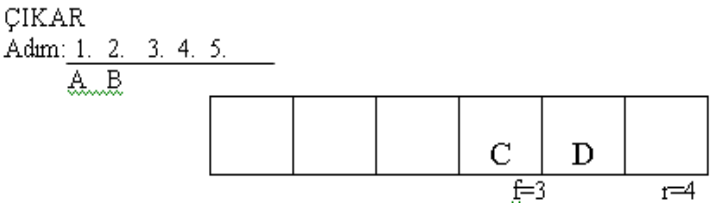
TEST:(SİLME)

Adım	f=0	V←Q(f)	f=r	f←r←0	f←f+1
1	1=0 (H)	A←Q(1)	1=4 (H)	-	2←1+1
2	2=0 (H)	B←Q(2)	2=4 (H)	-	3←2+1
3	3=0 (H)	C←Q(3)	3=4 (H)	-	4←3+1
4	4=0 (H)	D←Q(4)	4=4 (E)	f←0 r←0	-
5	0=0 (E)	**Underflow & exit**			

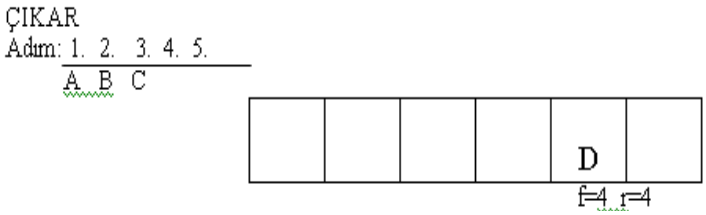
Başlangıç olarak; ilk adımda **f** isminin içeriği **1** dir. Ekleme işleminde **f** isminin içeriğine **1** sayısı atandıktan sonra bir daha değişmediğini gördünüz. Çıkarma algoritmasının ilk satırında **f=0** koşulu gerçekleşmediğinden bir alt satırdaki **Veri ← Q(f)** ifadesi işletime giriyor ve **Veri** ismine kuyruğun Q**(1).** inci eleman değeri (çünkü f=1) olan **D** değeri atanıyor. Bir sonraki satırda **if** cümlesindeki **f=r** koşulunun gerçekleşip gerçekleşmediği kontrolü yapılıyor. Bu durumda **f=1, r=4** olması nedeni ile algoritmanın son satırındaki **f ← f + 1** ifadesinde **f** isminin içeriği bir arttırılarak, **f** isminin tuttuğu yeni değeri **2** sayısı olur. Kuyruktan ilk değer çıkarıldıktan, ilk adımdan sonraki kuyruğun durumunu aşağıdaki şekilde görmektesiniz.



İkinci adım; birinci adımdan sonra **f=2** olması nedeni ile algoritmada ilk satırdaki **if** cümlesindeki **f=0** koşulu gerçekleşmediğinden ikinci satır uygulanır ve **Q(f)** dizisinin yani dizinin ikinci yerinden **veri** ismine **B** değeri atanarak diziden yeni bir eleman çıkarılmış olur. Üçüncü satırdaki **f=r** koşulu sorgulanır. **f=2, r=4** olması nedeni ile koşul gerçekleşmez. Dördüncü satır işletime girer ve **f** isminin içeriği tekrar bir artarak **f** isminin yeni içeriği **3** sayısına eşit olur. Kuyruktan ikinci değer çıkarıldıktan sonraki durumunu aşağıdaki şekilde görmektesiniz.



Üçüncü adım; İkinci adımdan sonra **f=3** olması nedeni ile algoritmada ilk satırdaki **if** cümlesindeki **f=0** koşulu gerçekleşmediğinden ikinci satır uygulanır ve **Q(f)** dizisinin yani dizinin üçüncü yerinden **veri** ismine **C** değeri atanarak diziden yeni bir eleman çıkarılmış olur. Üçüncü satırdaki **f=r** koşulu sorgulanır. **f=3, r=4** olması nedeni ile koşul gerçekleşmez. Dördüncü satır işletime girer ve **f** isminin içeriği tekrar bir artarak **f** isminin yeni içeriği **4** sayısına eşit olur. Kuyruktan üçüncü değer çıkarıldıktan
sonraki durumunu aşağıdaki şekilde görmektesiniz.

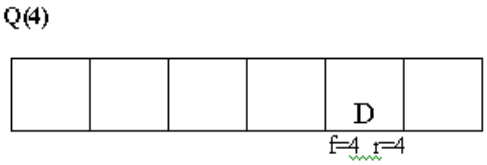


Dördüncü adım; üçüncü adımdan sonra **f=4** olması nedeni ile algoritmada ilk satırdaki **if** cümlesindeki **f=0** koşulu gerçekleşmediğinden ikinci satır uygulanır ve **Q(f)** dizisinin yani dizinin dördüncü yerinden **veri** ismine **D** değeri atanarak diziden yeni bir eleman çıkarılmış olur. Üçüncü satırdaki **f=r** koşulu sorgulanır. **f=4, r=4** olması nedeni ile koşul gerçekleşmiş olacağından. **Then** ifadesinden sonraki **f** ve **r** isimleri sıfır değerine düzenlenir. Bu durumda **f=0, r=0** olur. Kuyruktan dördüncü değer çıkarıldıktan sonraki durumunu aşağıdaki şekilde görmektesiniz.

Beşinci adım; dördüncü adımdan sonra **f=0** olması nedeni ile algoritmada ilk satırdaki **if** cümlesindeki **f=0** koşulu gerçekleşmiş olacağından kuyrukta **Under Flow** kontrolu yürütülecektir.

Yukarıda tanımlandığı gibi Q ismindeki bir dizinin 4 elemanlı olduğu kabul edilerek ekleme ve çıkarma işlemlerine ait testleri gördüğünüz gibi gerçekleştirildi. Tanımlanan kuyruğa basit yapı denilmesi; kuyruk dört elemanlı ise, dördüncü eleman yerinde bir değer var ise bundan sonra kuyruğa eklenecek her bir değer için kuyrukta **Over Flow** kontrolü yürütülecektir.

Bu durumu, Q dizisinin aşağıdaki şekilde gösterildiği gibi 4. eleman yerinde D gibi bir elemanın var olduğunu kabul edelim.



Bu durumda; **f=4** ve **r=4** olduğu ve kuyrukta 4. eleman yerinde de **D** değeri görülmektedir. Ekleme algoritmasına göre **E** gibi bir elemanı eklemeye çalışırsak; ilk satırda **if** cümlesinde **r >=n** koşuluna göre şekilde **r=4** olduğu ve başlangıçta **n=4** kabulü yapıldığı için koşul gerçekleşmiş olur ve **then** ifadesinden sonraki **Over Flow** iletisinin yazdırılacağını görebilirsiniz. Anlatılmaya çalışılan dizinin sonuncu elemanında bir değer var ise diğer elemanlarının durumuna bakılmaksızın kuyrukta **Over Flow** kontrolu yürütülür. Bundan dolayı bu tip kuyruklara **basit yapılı** kuyruk adı verilir.

Kuyruktan Eleman Silme İşleminde Bellek Görünümü

Aşağıdaki tabloda, **f** ve **r** indis isimlerine göre adım adım kuyruğun bellek görünüm durumarını görmektesiniz.

Adım						Çıkan
1	f=1 r=4	A	B	C	D	A
2	F=2 r=4	-	B	C	D	B
3	F=3 r=4	-	-	C	D	C
4	F=4 r=4	-	-	-	D	D
5	f=r f=0 r=0 underflow					

Basit yapılı kuyruklar, bellek kullanım yönünden pek yararlı değildir. Ayrıca, kuyruk kapasitesi çok çabuk dolmakta ve boş alanların yerine yenileri eklenememektedir. Bu tür tıkanıklıkları önlemek için dairesel kuyruklar kullanılmaktadır.

PASCAL' da Record yapı kullanarak Basit Yapılı Kuyruktan eleman silme

Önceki satırlarda kuyruğa eleman ekleme işleminde tanımlanmış record yapının alt alanlarına ulaşıp, eleman silme işlemini gerçekleştirebilirsiniz. Aşağıda verilen Function **Sil** isimli alt programı yazarsanız, record yapının alt alanından eleman silebilirsiniz.

```
function Sil(Parametre listesi):tipismi;
  Begin
        If  krk.arka >=4 then
                    Begin
                     Writeln('Under Flow');
                     Exit;
                    End;
         sil:=Q(krk.on);
        if krk.on=krk.arka then
                                Begin
                                Krk.on := 0;Krk.arka := 0;exit;
                                end;
        krk.on := krk.on + 1;
   End;
```