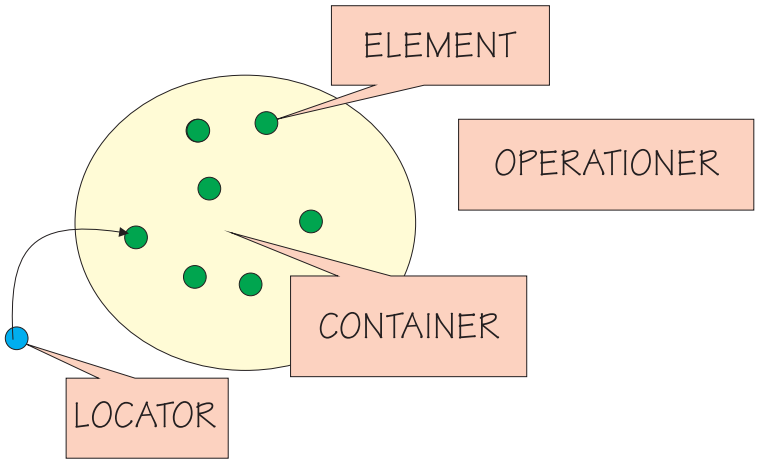
Datatsrukturer



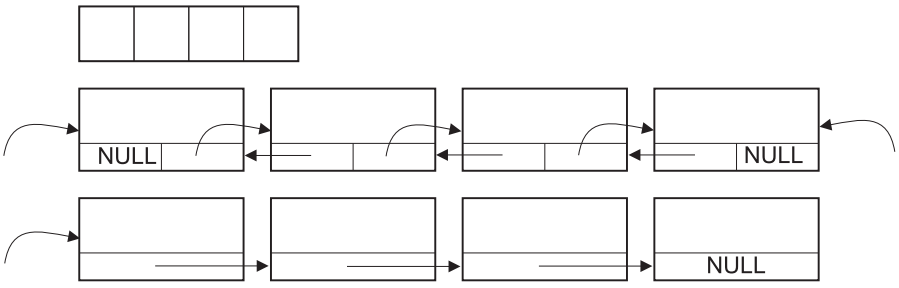
Figur 1:

Till de fundamentala datastrukturerna hör

- SEKVENS (Sequence). En sekvens är en container som lagrar element i en bestämd ordning. De viktigaste operationerna är att lägga till och ta bort element på en given plats. STACK och KÖ är kanske de två mest omtalade datastrukturerna, som båda hör till denna kategori. I programspråket C implementeras dessa ofta som *länkade listor*, men arrayer duger ofta bra.
- PRIORITETSKÖ (Priority Queue). Elementen i denna container tillhör ett *totalt ordnat universum*. Den viktigaste operationerna är att kunna ta bort och returnera det största (alternativt det minsta) elementet i containern. Strukturen har en naturlig användning är vid sortertering och en effektiv implementation är genom en HEAP. Vi kommer senare i kursen, under kapitlet sortering, till HEAPSORT. Det finns en mängd varianter på HEAP som vi inte kommer att beröra i denna kurs: min-max heap, pagodas, deaps, binomial heaps och Fibonacci heaps.
- LEXIKON (Dictionary) Elementen i denna container tillhör också ett *totalt ordnat universum*. De viktigaste operationerna är *söka, lägga till* och `ta bort` element. Till varje element finns en unik nyckel. Här finns många sofistikerade metoder vid implementationen. Vi kommer att beröra HASHTABELLER. Flera metoder innehåller *balanserade träd*: AVL-TREE, RED-BLACK-TREE, 2-3-TREE, 2-3-4-TREE, WEIGHT-BALANCED-TREE, BIASED SERCH TREE, SPLAY TREE. Ofta handlar tillämpningarna om externt minne som till exempel databaser.

SEKVENS (S) är en container som lagrar element där ordningen är viktig

- Operationer:
  - SIZE(N) returnerar antalet element N i containerna S
  - HEAD(c) tilldelar c en locator till första elementet i S
  - TAIL(c) tilldelar c en locator till sista elementet i S
  - LOCATERANK(r,c) tilldelar c en locator till det r:e elementet i S. Om $r < 0$ eller $r > N$, där N är antalet element i S, tilldelas c en NULL locator.
  - PREV(c',c") tilldelar c" en locator i S som föregår elementet med locator c'. Om c' är locator till det första elementet i S tilldelas c" en NULL locator.
  - NEXT(c',c") tilldelar c" en locator i S som följer efter elementet med locator c'. Om c' är locator till det sista elementet i S tilldelas c" en NULL locator.
  - INSERAFTER(e,c',c") sätter in elementet e i S efter elementet med locator c' och returnerar c", som är locator till e.
  - INSERTBEFORE(e,c',c") sätter in elementet e i S före elementet med locator c' och returnerar c", som är locator till e.
  - INSERTHEAD(e,c) sätter in elementet e först i S och returnerar c, som är locator till e.
  - INSERTHEAD(e,c) sätter in elementet e sist i S och returnerar c, som är locator till e.
  - INSERRANK(e,r,c) sätter in e på den r:te positionen och returnerar c, som är en locator till e. Om $r < 0$ eller $r > N$, där N är antalet element i S, tilldelas c en NULL locator.
  - REMOVE(c,e) tar bort och returnerar elementet e med locator c från S
  - MODIFY(c,e) byter ut elementet med locator c i S mot e



Figur 2: *Array, Dubbellänkad lista, Länkad lista*

- Kostnad när sekvensen implementerad som:

Operation	ARRAY	LÄNKAD LISTA	DUBBELLÄNKAD LISTA
SIZE	O(1)	O(1)	O(1)
HEAD	O(1)	O(1)	O(1)
TAIL	O(1)	O(1)	O(1)
LOCATERANK	O(1)	O(N)	O(N)
PREV	O(1)	O(N)	O(1)
NEXT	O(1)	O(1)	O(1)
INSERTAFTER	O(N)	O(1)	O(1)
INSERTBEFORE	O(N)	O(N)	O(1)
INSERTHEAD	O(N)	O(1)	O(1)
INSERTTAIL	O(1)	O(1)	O(1)
INSERTRANK	O(1)	O(N)	O(N)
REMOVE	O(N)	O(N)	O(1)
MODIFY	O(1)	O(1)	O(1)

N är antalet element i S

PRIORITETSKÖ (Priority Queue) Prioritetskö är en container med element i ett ordnat universum.

- Operationer:
  - SIZE(N) returnerar antalet element N i kön Q
  - MAX(c) returnera en locator c till det största elementet i Q
  - INSERT(e,c) placerar elementet e i kön Q och returnerar locator till e
  - REMOVE(c,e) tar bort och returnerer elementet e med locator c från Q
  - REMOVEMAX(e) tar bort och returnerar det största elementet e från Q
  - MODIFY(c,e) ersätter elementet med locator c med elementet e
- Kostnad när sekvensen implementerad som:

Operation	OSORTERAD SEKVENS	SORTERAD SEKVENS	HEAP
SIZE	O(1)	O(1)	O(1)
MAX	O(N)	O(1)	O(1)
INSERT	O(1)	O(N)	O(log N)
REMOVE	O(1)	O(1)	O(log N)
REMOVEMAX	O(N)	O(1)	O(log N)
MODIFY	O(1)	O(N)	O(log N)

N är antalet element i Q. En HEAP är binärt sökträd. Mer om det senare i kursen

LEXIKON (Dictionary)

- Operationer:
  - SIZE(N) returnerar antalet element N i D

- FIND(x,c) om D innehåller ett element med nyckeln x kommer c att vara en locator till detta element
- LOCATEPREV(x,c) tilldelar c en locator till det element i D med den största nyckeln som är mindre än x
- LOCATENEXT(x,c) tilldelar c en locator till det element i D med den minsta nyckeln som är sörre än eller lika med x
- LOCATERANK(r,c) tilldelar c en locator till det r:te elementet i D
- PREV(c',c") tilldelar c" en locator till det element i D med den största nyckeln som är mindre än nykeln till det element som har locator c'
- NEXT(c',c") tilldelar c" en locator till det element i D med den minsta nyckeln som är större än nykeln till det element som har locator c'
- MIN(c) tilldelar c en locator till elementet i D med den minsta nyckeln
- MAX(c) tilldelar c en locator till elementet i D med den största nyckeln
- INSERT(e,c) sätter in ett element e i D och returnerar dess locator c. Om det redan finns ett element e i D med samma nyckel returneras NULL locator.
- REMOVE(c,e) Avlägsnar och returnerar elementet e med locator c från D
- MODIFY(c,e) ersätter elementet med locator c i D med elementet e

- Kostnad när sekvensen implementerad som:

Operation	OSORTERAD SEKVENS	SORTERAD SEKVENS	ARRAY
SIZE	O(1)	O(1)	O(1)
FIND	O(N)	O(N)	O(log N)
LOCATEPREV	O(N)	O(N)	O(log N)
LOCATENEXT	O(N)	O(N)	O(log N)
LOCATERANK	O(N)	O(N)	O(1)
NEXT	O(N)	O(1)	O(1)
PREV	O(N)	O(1)	O(1)
MIN	O(N)	O(1)	O(1)
MAX	O(N)	O(1)	O(1)
INSERT	O(1)	O(N)	O(N)
REMOVE	O(1)	O(1)	O(N)
MODIFY	O(1)	O(N)	O(N)

N är antalet element i D

Operation	(A,B)-TREE	AVL-TREE	BUCKET ARRAY	HASH
SIZE	O(1)	O(1)	O(1)	O(1)
FIND	O(log N)	O(log N)	O(1)	O(N/M)
LOCATEPREV	O(log N)	O(log N)	O(M)	O(N+M)
LOCATENEXT	O(log N)	O(log N)	O(M)	O(N+M)
LOCATERANK	O(log N)	O(log N)	O(M)	O(N+M)
NEXT	O(log N)	O(log N)	O(M)	O(N+M)
PREV	O(log N)	O(log N)	O(M)	O(N+M)
MIN	O(1)	O(1)	O(M)	O(N+M)
MAX	O(1)	O(1)	O(M)	O(N+M)
INSERT	O(log N)	O(log N)	O(1)	O(1)
REMOVE	O(log N)	O(log N)	O(1)	O(1)
MODIFY	O(log N)	O(log N)	O(1)	O(1)

N är antalet element i D. Nycklarna är heltal i $[1, M]$ (A,B)-TREE och AVL-TREE ingår inte i kursen. BUCKET ARRAY är enkelt att implementera. Återkommer i samband med sortering. Även `Hash Tabell` kommer längre fram i kursen.

Exempel Stack

```
class stack:
  def __init__(self):
    self.s=[]
  def pop(self):
    return self.s.pop(0)
  def empty(self):
    return len(self.s)==0
  def push(self,a):
    return self.s.insert(0,a)
  def skrivut(self):
    print self.s

def main():
  a=stack()
  for i in range(10):
    a.push(i)
  while not a.empty():
    print a.pop(),

main()
```

Ett exempel på en stack med hjälp av Python. Med detta exempel vill vi visa idén och tänker inte på effektiviteten hos implementationen. I själva verket använder vi en annan klass, `list`, i Python för att skapa operationerna.

Exempel Prioritetskö

```
import random
class pqueue:
  def __init__(self):
    self.s=[]
  def insert(self,a):
    self.s.append(a)
  def removemax(self):
    a=max(self.s)
    self.s.remove(a)
    return a
  def size(self):
    return len(self.s)
  def skrivut(self):
    print self.s

def main():
  lista=pqueue()
  for i in range(20):
    lista.insert(random.randrange(100))
  lista.skrivut()
  while lista.size()>0:
    print lista.removemax(),

main()
```

Här använder vi en prioritetskö för att sortera 20 slumptal. Återigen en ineffektiv implementering!