För utveckling av verksamhet, produkter och livskvalitet.

ADS : STACK

- Det finns ett par vanligt förekommande ADT:er för samlingar av element som egentligen **är specialfall av listor**. En av dem är modellen *Stack*:
- Def: En stack är en följd av element där borttagnig av ett element alltid avser det element som senast satts in.
- Kallas även LIFO-lista, Last In First Out

För utveckling av verksamhet, produkter och livskvalitet.

ADT: STACK

- En stack beskrivs oftast lodrätt orienterad.
- Operationer sker på toppen av stacken.
- Här läggs nytt element in och vid borttagning avses alltid översta elementet (top).

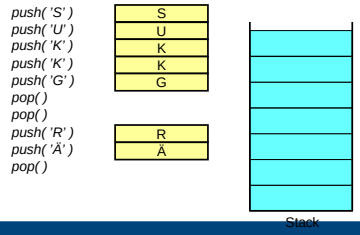
För utveckling av verksamhet, produkter och livskvalitet.

Specifikation av ADT för stack, enligt läroboken (6.6.2)

```
public interface Stack {
    /** Lägg x "överst" på stacken */
    void push(Object x);
  /** Tag bort översta elementet från stacken */
   void pop();
  /** Tag reda på översta elementet på stacken */
  Object top();
  /** Tag reda på och tag bort översta elementet på stacken */
 Object topAndPop();
/** Undersök om stacken är tom */
  boolean isEmpty();
/** Töm stacken */
void makeEmpty();
 }
```

För utveckling av verksamhet, produkter och livskvalitet.

Principen med en stack



Användningsområden

- Datorteknik
  - Metod anrop, Kompilatorer
- Beräkningsmaskin
  - Omvänd polsk (postfix) notation (1950-talet)
  - Kräver ”ett annat sorts tänkande...”
  - Kraftfull!

Implementeringar

- Statisk implementation
  - Arraybaserad
- Dynamiska implementation
  - Länkade listor

Med array...

- ToS – Top of Stack
  - Indexerar senast inlagda data
- Statisk maxstorlek
  - Dubblering möjlig men kostsam
- Snabb

Exempel

```
public class Stack <AnyType>
{
  private ...

  public Stack()
  {
  }

  public void push(AnyType data)
  {
  }

  public void pop()
  {
  }
}
```


Tillstånd / Konstruktorn

```
private AnyType [] array;
private int ToS;

public Stack()
{
  array = (AnyType)new Object[50];
  ToS = -1;
}
```


Metoden -- *'void push(AnyType data)'*

Tidskomplexiteten ??

```
public void push(AnyType data)
{
  if((ToS + 1) < array.length)
  {
    ToS++;
    array[ToS] = data;
  }
}
```


Metoden -- *AnyType pop()'*

Tidskomplexiteten ??

```
public AnyType pop()
{
  AnyType value = null;

  if(ToS >= 0)
  {
    value = array[ToS];
    ToS--;
  }
 return value;
}
```

Med en länkad lista...

- Arbetar bara mot länken närmast huvudet
  - Lägger in först i listan
  - Tar ut först ur listan
- Långsam
  - Minnesallokering

Exempel

```
public class Stack
{
private ...

public Stack()
{
}
public void push(Object x)
{
}

public Object pop()
{
}

}
```


Tillstånd / Konstruktorn

```
private ListNode topofstack;

public Stack()
{
  topofstack = null;
}
```


Metoden -- *'void push(Object data)'*

```
public void push(Object x)
{
 topofstack=new ListNode(x,topofstack);
}
```

Metoden -- *'Object pop()'*

```
public Object pop()
{
ListNode temp=topofstack;
topofstack=topofstack.next
return temp.element;
}
```


Implementationsjämförelse

- Statisk implementation
  - (Kan ha) mycket outnyttjat minne
  - Snabb
- Dynamisk implementation
  - Minimalt outnyttjat minne
  - Långsam

Tillämpning-kompilator och miniräknare – kap

- Beräkning
  -aritmetiska uttryck i infix notation : *4*3+2^3*
  - aritmetiska uttryck i postfix notation: *43*23^4*
- Kompilatorer, kontrolerar om koden är rätt parentiserad:
  rätt: { ( ) } fel: { (})

ADT: Kö - Kallas även FIFO-lista, First In First Out

- En annan modell, som också är specialfall av lista:
- Def: En kö är en följd av element där
- insättning alltid sker i slutet av följden
- borttagning alltid avser följdens första element

Användningsområden

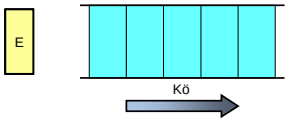
- Printer köer
- Simuleringar , flyg, passersystem ...
- Asynkrona gränssnitt
  - Kommunikation mellan olika processer på samma operativ system
- Data buffring
- Datakommunikation / Internet

Specifikation av ADT -Queue

```
public interface Queue {
/** Sätt in x sist i kön */
void enqueue(Object x);
/** Tag reda på första elementet i kön */
Object getFront();
/** Tag reda på och tag bort första elementet i kön */
Object dequeue();
/** Undersök om kön är tom */
boolean isEmpty();
/** Töm kön */
void makeEmpty();
}
```


Principen med en kö

enqueue('A')
enqueue('B')
enqueue('C')
enqueue('D')
dequeue( )
dequeue( )
enqueue('E')
dequeue( )



Implementeringar

- Statisk implementation
  - Arraybaserad
- Dynamiska implementation
  - Länkade listor

Med en array...

- "Wraparound"
  - Slipper flytta data
  - Kräver start och slut index
  - Snabbt
- Statisk maxstorlek
  - Dubblering möjlig men kostsam

Exempel

```
public class Queue
{
  private ...

  public Queue()
  {
  }

  public void enqueue(Object data)
  {
  }

  public Object dequeue()
  {
  }
}
```


Tillstånd / Konstruktorn

```
private Object [] array;
private int back, front;
private int size;

public Queue()
{
  array = new Object[50];
  back = -1;
  front = 0;
  size =0;
}
```


Metoden -- *'void enqueue(Object data)'*

Tidskomplexiteten ??

```
public void enqueue(Object data)
{
  if(size < array.length)
  {
    back = (back + 1)
    array[back] = data;
    size++;
  }
}
```

Metoden -- *'void enqueue(Object data)'*

med Wraparound

Tidskomplexiteten ??

```
public void enqueue(Object data)
{
  if(size == array.length)
    doubleQueue();
    back = increment(back);
    array[back] = data;
    size++;
 }
```


Metoden -- *'int increment(int x)'*

med Wraparound

```
private int increment(int x)
{
  if(++x == array.length)
    x=0;
  return x;

}
```


Exempel

```
public class Queue
{
  private ...

  public Queue()
  {
  }

  public void enqueue(Object data)
  {
  }

  public Object dequeue()
  {
  }
}
```


Metoden -- *'Object dequeue()'*

```
public Object dequeue()
{
  Object value = null;

  if(size > 0)
  {
    value = array[front];
    front = front + 1;
    size--;
  }

  return value;
}
```

Metoden -- *'Object dequeue()'*

Med wraparound

```
public Object dequeue()
{
  Object value = null;

  if(size > 0)
  {
    value = array[front];
    front = increment(front);
    size--;
  }

  return value;
}
```


Med en länkad lista...

- Arbetar bara mot ändarna på listan
  - Lägger in först i listan
  - Tar ut sist ur listan
- Långsam
  - Minnesallokering
- Behöver ingen ”wrapping”
  - Enklare implementation

Tillstånd/ Konstruktor

```
private ListNode front;
private ListNode back;
public Queue()
{
  front=back=null;
}
```


Metoden -- *'void enqueue(Object data)'*

```
public void enqueue(Object data)
{
  if (isEmpty())
     back=front=new ListNode(data);
  else
     back=back.next=new ListNode(data);

}
```

Metoden -- *'Object dequeue()'*

```
public Object dequeue()
{
  if(isEmpty())
    throw new UnderflowException("Tom kö");
  Object value=front.element;
  front=front.next;
  return value;
}
```


Implementationsjämförelse

- Statisk implementation
  - (Kan ha) mycket outnyttjat minne
  - Snabb
- Dynamisk implementation
  - Minimalt outnyttjat minne
  - Långsam
- Konstant tid