Datatyper

Primitiva datatyper:
- booleska
- karaktärer (char)
- heltalstyper
- flyttalstyper

Referenstyper:
- pekare till arrayer
- pekare till objekt (instanser av klasser)

Account

```
public class Account {

  public Account(int initBalance) {
    balance = initBalance;
  }

  private int balance;

  public int balance() {
    return balance;
  }

  public int deposit(int amount) {
    balance = balance + amount;
    return balance;
  }

  public int withdraw(int amount) {
    if( balance < amount ) {
      return 0;
    } else {
      balance = balance - amount;
      return amount;
    }
  }

}
```


Heltalslista

```
class ListNode {
  int n;
  ListNode next;
}

public class IntList {

  private int size;
  private ListNode first;

  public IntList() {
    size = 0;
  }

  public void insert(int n) {
    /* Insert n in list */
  }

  public int size() {
    return size;
  }

  public void remove(int n) {
    /* Remove n from list */
  }

  public boolean find(int n) {
    /* Check if n occurs in list */
  }

} // End Class IntList
```


Insert (oordnad lista)

```
public void insert(int n) {

  ListNode tmp = new ListNode();

  tmp.n = n;
  tmp.next = first;
  first = tmp;
  size++;
}
```

Remove (oordnad lista)

```
public void remove(int n) {

  if( first != null ) {

    if( first.n == n ) {
      first = first.next;
      size--;
    }
    else {

      ListNode tmp = first;

      while( tmp.next != null )
        if( tmp.next.n == n ) {
          tmp.next = tmp.next.next;
          size--;
        } else tmp = tmp.next;
    }
  }
```


Find (oordnad lista)

```
public boolean find(int n) {

  ListNode tmp = first;

  while( tmp != null ) {
    if( tmp.n == n ) return true;
    else tmp = tmp.next;
  }
  return false;
}
```


Insert (ordnad lista)

```
public void insert(int n) {
  if( first == null ) { /* Empty list */
    first = new ListNode();
    first.n = n; first.next = null;
    size = 1;
  }
  else { /* Non-empty list */
    ListNode tmp = first;
    if( n < tmp.n ) { /* Insert first */
      ListNode node = new ListNode();
      node.n = n; node.next = first;
      first = node;
      size++;
    }
    else if( n == tmp.n ) return;
    else {
      while( tmp.next != null ) {
        if( n < tmp.next.n ) {
          ListNode node = new ListNode();
          node.n = n; node.next = tmp.next;
          tmp.next = node;
          size++;
          return;
        }
        else if( n == tmp.next.n ) return;
        else tmp = tmp.next;
      }
      ListNode node = new ListNode();
      node.n = n;
      node.next = null;
      tmp.next = node;
      size++;
    }
  }
}
```


Remove (ordnad lista)

```
public void remove(int n) {

  if( first != null ) {

    if( first.n == n ) {
      first = first.next;
      size--;
    } else {

      ListNode tmp = first;

      while( tmp.next != null) {
        if( n < tmp.next.n ) return;
        else if( tmp.next.n == n ) {
          tmp.next = tmp.next.next;
          size--;
        } else tmp = tmp.next;
      }
    }
  }
}
```

Find (ordnad lista)

```
public boolean find(int n) {

  ListNode tmp = first;

  while( tmp != null ) {
    if( n < tmp.n ) return false;
    else if( n == tmp.n ) return true;
    else tmp = tmp.next;
  }
  return false;
}
```


Rekursion

Fakultetsfunktionen:

```
0! = 1
1! = 1 = 1*0! = 1*1 = 1
2! = 2*1 = 2*1! = 2*1 = 2
3! = 3*2*1 = 3*2! = 3*2 = 6
4! = 4*3*2*1 = 4*3! = 4*6 = 24
:
```

Det vill säga:

```
0! = 1
n! = n * (n - 1)!  (för alla n > 0)
```

Eller i Java:

```
int fact(int n) {

  if( n == 0 ) return 1;
  else return n * fact(n - 1);

}
```


Summor

$$sum(m, n) = \sum_{x=m}^{n} x^2$$

Version 1:

```
int sum(int m, int n) {

  if( m > n ) return 0;
  else return m*m + sum(m + 1, n);

}
```

Version 2:

```
int sum(int m, int n) {

  if( m > n ) return 0;
  else return n*n + sum(m, n - 1);

}
```


Fibonacci

$fib(0) = 0$
$fib(1) = 1$
$fib(n) = fib(n-1) + fib(n-2)$ (då $n > 1$)

```
int fib(int n) {

  if( n == 0 ) return 0;
  else if( n == 1 ) return 1;
  else return (fib(n-1)+fib(n-2));

}
```

Euklides algoritm

Observation:

- *Om D delar A och B så delar D också A – B och B*
- *Om D delar A – B och B så delar D också A och B*

```
long gcd(long a, long b) {

  if( b == 0 )
    return a;
  else
    return gcd(b, a % b);

}
```


”Reversering” av array

```
void reverseArray(int[] A) {

  reverseArray(A, 0, A.length - 1);

}

void reverseArray(int[] A, int m, int n) {

  if (m < n) {
    int tmp = A[m];
    A[m] = A[n];
    A[n] = tmp;
    reverseArray(A, m + 1, n - 1);
  }

}
```


Insert (rekursiv)

```
public void insert(int n) {

  first = insert(n, first);

}

private ListNode insert(int n, ListNode x) {

  if( x == null ) { /* The list is empty */
    ListNode node = new ListNode();
    node.n = n;
    node.next = null;
    size++;
    return node;
  } else { /* The list is non-empty */
    if( n < x.n ) {
      ListNode node = new ListNode();
      node.n = n;
      node.next = x;
      size++;
      return node;
    } else if( n > x.n ) {
      x.next = insert(n, x.next);
      return x;
    }
    else return x;
  }

}
```


Remove (rekursiv)

```
public void remove(int n) {

  first = remove(n, first);

}

private ListNode remove(int n, ListNode x) {

  if( x == null ) /* The list is empty */
    return null;
  else { /* The list is non-empty */
    if( n < x.n ) return x;
    else if( n == x.n ) {
      size--;
      return x.next;
    } else {
      x.next = remove(n, x.next);
      return x;
    }
  }

}
```

Find (rekursiv)

```
public boolean find(int n) {

  return find(n, first);

}

private boolean find(int n, ListNode x) {

  if( x == null ) /* The list is empty */
    return false;
  else { /* The list is non-empty */
    if( n < x.n ) return false;
    else if( n == x.n ) return true;
    else return find(n, x.next);
  }

}
```