Informatik

Abstrakte Datentypen - Liste

Abstrakter Datentyp (ADT) = Datenstruktur + (abstrakte) Operationen

abstrakt = nicht implementiert, nur Vorgaben.

Abstrakter Datentyp (ADT) = Datenstruktur + (abstrakte) Operationen

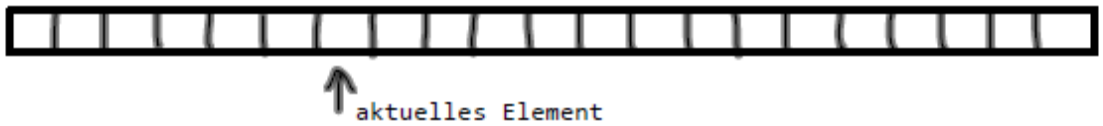
abstrakt = nicht implementiert, nur Vorgaben.

Beschreibung von Datenstrukturen unabhängig von ihrer späteren Implementierung in einer Programmiersprache.

ADTs bilden eine Spezifikation der Schnittstelle nach außen, indem sie Operationen und ihre Funktionalität festlegen

ADT Liste

Definition: eine Liste ist eine (ggf. leere) Folge von Elementen zusammen mit einem so genannten (ggf. undefinierten) aktuellen Element.

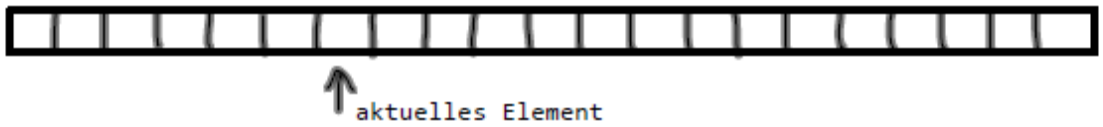


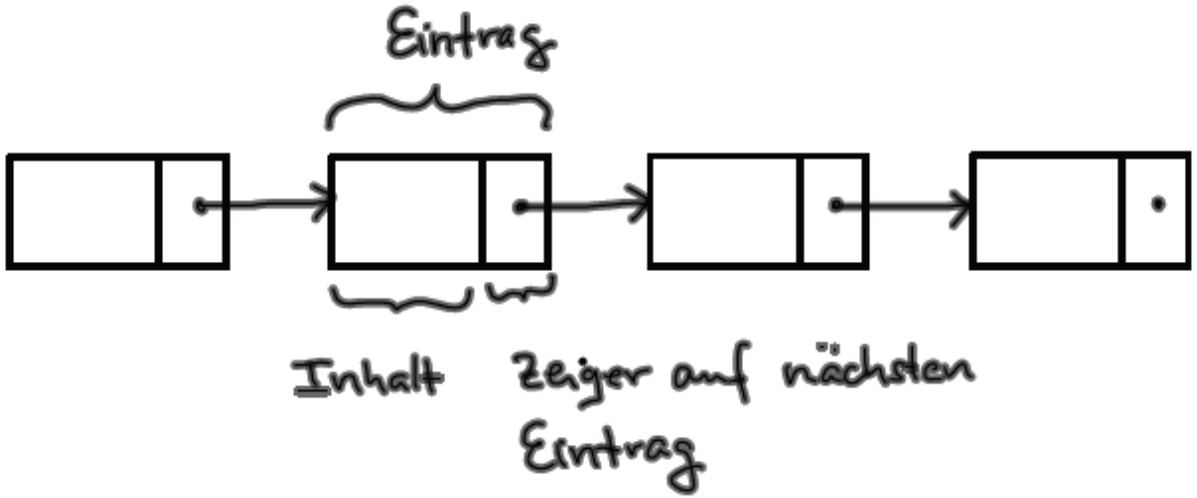
Schnittstelle der ADT Liste:

empty	:	liefert true, falls Liste leer
endpos	:	liefert true, wenn Liste abgearbeitet
reset	:	das erste Listenelement wird zum aktuellen Element
advance	:	der Nachfolger des akt. wird akt. Element
elem	:	liefert das aktuelle Element
insert	:	fügt vor das aktuelle Element ein Element ein, das neue wird zum aktuellen Element
delete	:	löscht das aktuelle Element, der Nachfolger wird zum aktuellen Element.

```
class Liste:
    ''' Eine Liste ist eine (ggf. leere) Folge von Elementen zusammen mit einem
    (ggf. undefinierten) aktuellen Element '''

    def empty(self):
        ''' liefert true, falls Liste leer '''
        pass

    def endpos(self):
        ''' liefert true, wenn die Liste abgearbeitet ist '''
        pass

    def reset(self):
        ''' das erste Listenelement wird aktuelles Element '''
        pass

    def advance(self):
        ''' der Nachfolger des aktuellen Elements wird aktuelles Element '''
        pass

    def elem(self):
        ''' liefert das aktuelle Element '''
        pass

    def insert(self, x):
        ''' Fügt x vor dem aktuellen Element ein, x wird zum neuen aktuellen Element. '''
        pass

    def delete(self):
        ''' löscht das aktuelle Element. Der Nachfolger wird neues aktuelles Element. '''
        pass
```

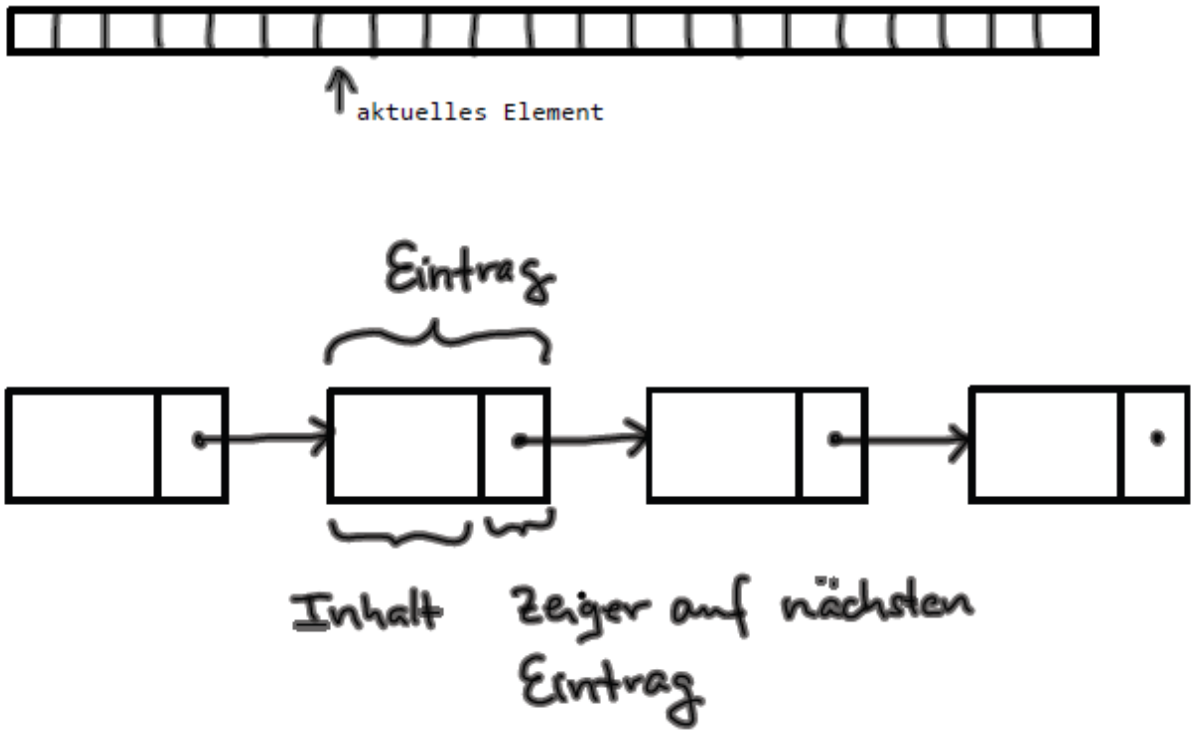
Implementation durch verkettete Einträge





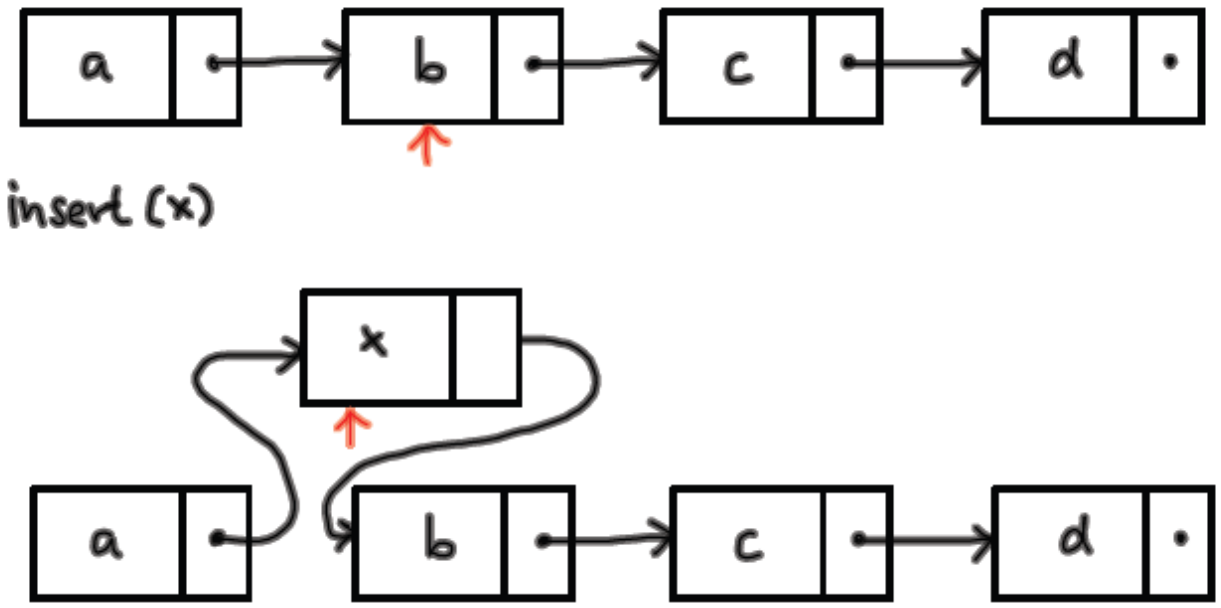

```
class Eintrag:
```

Implementation durch verkettete Einträge

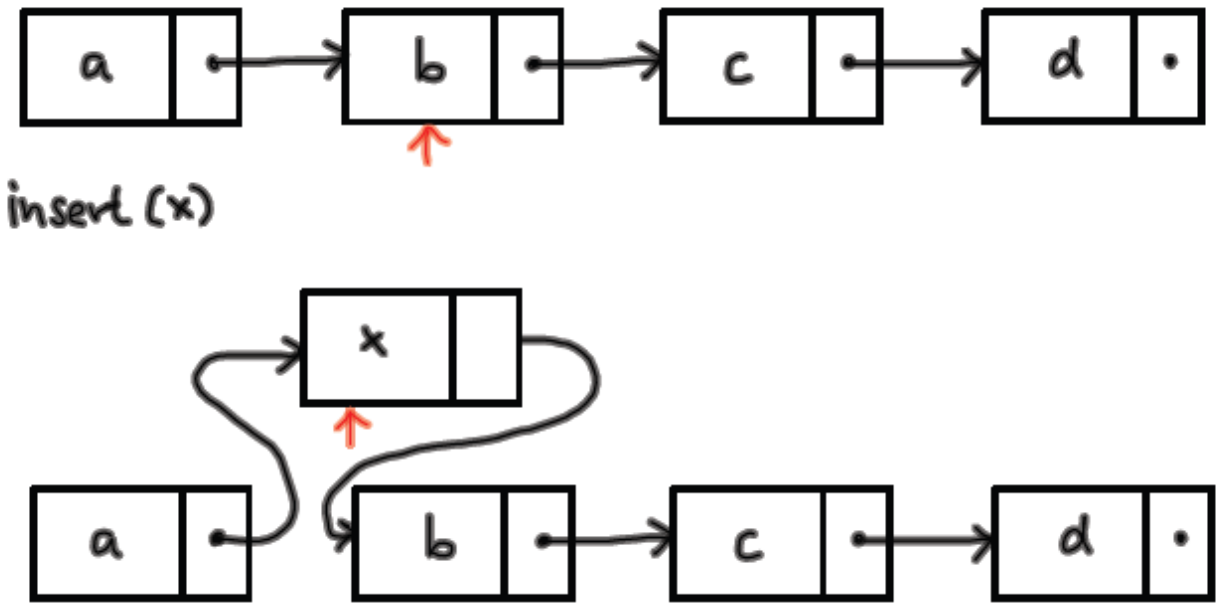



```
class Eintrag:
    def __init__(self):
        self.inhalt = None
        self.next = None
```

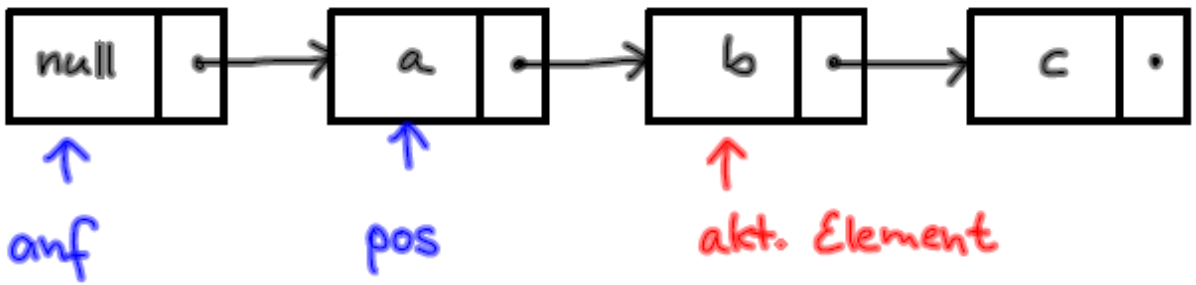
Problem bei der Implementation von insert:



Problem bei der Implementation von insert:



Wie kann man den Zeiger, der auf das aktuelle Element zeigt, auf x umbiegen?



pos zeigt auf den Listen-Eintrag vor dem aktuellen Element.

anf zeigt auf einen dummy Eintrag vor dem ersten Element.

Mit assert-Anweisungen können wir unsere Implementation überprüfen

```
li = Liste()
assert li.empty()
assert li.endpos()
li.insert('A')
assert not li.empty()
assert not li.endpos()
assert li.elem() == 'A'
li.advance()
assert li.endpos()
li.insert('B')
assert not li.endpos()
li.advance()
li.insert('C')
li.reset()
assert li.elem() == 'A'
li.advance()
li.delete()
assert li.elem() == 'C'
li.delete()
assert li.endpos()
li.reset()
li.delete()
assert li.empty()
```

Bei einer nicht-leeren Liste können wir, wenn das aktuelle Element das letzte Element der Liste ist, mit advance() noch einen Schritt vorangehen. Danach ist endpos() True, pos zeigt auf das letzte Element und das aktuelle Element ist schon "jenseits der Liste".

Nach der Implementation

```
class Liste:
    def __init__(self):
```

Nach der Implementation

```
class Liste:
    def __init__(self):
        self.anf = Eintrag()
        self.pos = self.anf

    def empty(self):
```

Nach der Implementation

```
class Liste:
    def __init__(self):
        self.anf = Eintrag()
        self.pos = self.anf

    def empty(self):
        return self.anf.next is None

    def endpos(self):
```

Nach der Implementation

```
class Liste:
    def __init__(self):
        self.anf = Eintrag()
        self.pos = self.anf

    def empty(self):
        return self.anf.next is None

    def endpos(self):
        return self.pos.next is None

    def reset(self):
```

Nach der Implementation

```
class Liste:
    def __init__(self):
        self.anf = Eintrag()
        self.pos = self.anf

    def empty(self):
        return self.anf.next is None

    def endpos(self):
        return self.pos.next is None

    def reset(self):
        self.pos = self.anf

    def advance(self):
```

Nach der Implementation

```
class Liste:
    def __init__(self):
        self.anf = Eintrag()
        self.pos = self.anf

    def empty(self):
        return self.anf.next is None

    def endpos(self):
        return self.pos.next is None

    def reset(self):
        self.pos = self.anf

    def advance(self):
        if self.endpos(): raise RuntimeError("Fehler: Liste am Ende")
        self.pos = self.pos.next

    def elem(self):
```

Nach der Implementation

```
class Liste:
    def __init__(self):
        self.anf = Eintrag()
        self.pos = self.anf

    def empty(self):
        return self.anf.next is None

    def endpos(self):
        return self.pos.next is None

    def reset(self):
        self.pos = self.anf

    def advance(self):
        if self.endpos(): raise RuntimeError("Fehler: Liste am Ende")
        self.pos = self.pos.next

    def elem(self):
        if self.endpos(): raise RuntimeError("Fehler: Liste am Ende")
        return self.pos.next.inhalt

    def insert(self, x):
```

Nach der Implementation

```
class Liste:
    def __init__(self):
        self.anf = Eintrag()
        self.pos = self.anf

    def empty(self):
        return self.anf.next is None

    def endpos(self):
        return self.pos.next is None

    def reset(self):
        self.pos = self.anf

    def advance(self):
        if self.endpos(): raise RuntimeError("Fehler: Liste am Ende")
        self.pos = self.pos.next

    def elem(self):
        if self.endpos(): raise RuntimeError("Fehler: Liste am Ende")
        return self.pos.next.inhalt

    def insert(self, x):
        hilf = Eintrag()
        hilf.inhalt = x
        hilf.next = self.pos.next
        self.pos.next = hilf

    def delete(self):
```

Nach der Implementation

```
class Liste:
    def __init__(self):
        self.anf = Eintrag()
        self.pos = self.anf

    def empty(self):
        return self.anf.next is None

    def endpos(self):
        return self.pos.next is None

    def reset(self):
        self.pos = self.anf

    def advance(self):
        if self.endpos(): raise RuntimeError("Fehler: Liste am Ende")
        self.pos = self.pos.next

    def elem(self):
        if self.endpos(): raise RuntimeError("Fehler: Liste am Ende")
        return self.pos.next.inhalt

    def insert(self, x):
        hilf = Eintrag()
        hilf.inhalt = x
        hilf.next = self.pos.next
        self.pos.next = hilf

    def delete(self):
        if self.endpos(): raise RuntimeError("Fehler: Liste am Ende")
        self.pos.next = self.pos.next.next
```