Einfügen eines Elements in eine …

… doppelt-verkettet Liste

```
PROCEDURE Insert(VAR l: List; n: NodePtr);
    VAR
      succ: NodePtr; (*successor of new node n*)
  BEGIN
    Assert(Sorted(l), 'before Insert: list not sorted');
    IF l.first = NIL THEN BEGIN (*l.last = NIL, too*)
        l.first := n;
        l.last := n;
      END (*THEN*)
    ELSE BEGIN
        succ := l.first;
        WHILE (succ <> NIL) AND (n^.val > succ^.val) D.B.
          succ := succ^.next;
        END; (*WHILE*)
        IF succ = l.first THEN BEGIN (*prepend n*)
            n^.next := l.first;
            l.first^.prev := n;
            l.first := n;
          END (*THEN*)
        ELSE IF succ = NIL THEN BEGIN (*append n*)
            n^.prev := l.last;
            l.last^.next := n;
            l.last := n;
          END (*ELSE*)
        ELSE BEGIN (*insert n in the middle, before succ*)
            n^.prev := succ^.prev;
            n^.next := succ;
            succ^.prev^.next := n;
            succ^.prev := n;
        END; (*ELSE*)
      END; (*ELSE*)
    Assert(Sorted(l), 'after Insert: list not sorted');
  END; (*Insert*)
```

Einfügen eines Elements in eine …

… doppelt-verkettet Liste

```
PROCEDURE Insert(VAR l: List
    VAR
      succ: NodePtr; (*succe
  BEGIN
    Assert(Sorted(l), 'befor
    IF l.first = NIL THEN BE
        l.first := n;
        l.last := n;
      END (*THEN*)
    ELSE BEGIN
        succ := l.first;
        WHILE (succ <> NIL)
          succ := succ^.next
        END; (*WHILE*)
        IF succ = l.first TH
            n^.next := l.fir
            l.first^.prev :=
            l.first := n;
          END (*THEN*)
        ELSE IF succ = NIL T
            n^.prev := l.las
            l.last^.next :=
            l.last := n;
          END (*ELSE*)
        ELSE BEGIN (*insert
            n^.prev := succ^
            n^.next := succ;
            succ^.prev^.next
            succ^.prev := n;
        END; (*ELSE*)
      END; (*ELSE*)
    Assert(Sorted(l), 'after
  END; (*Insert*)
```

… doppelt-verkettet Liste **mit Anker**

```
PROCEDURE Insert(l: ListPtr; n: NodePtr);
  VAR
    succ: NodePtr; (*successor of new node n*)
BEGIN
  Assert(Sorted(l), 'before Insert: list not sorted');







  succ := l^.next;
  WHILE (succ <> l) AND (n^.val > succ^.val) D. B.
    succ := succ^.next;
  END; (*WHILE*)











  n^.prev := succ^.prev;
  n^.next := succ;
  succ^.prev^.next := n;
  succ^.prev := n;


  Assert(Sorted(l), 'after Insert: list not sorted');
END; (*Insert*)
```