EINFÜGEN IN SORTIERTE LISTE

Einfügen

```
// pre: verkettete Liste, erstes Element first
// post: value in Liste aufsteigend sortiert eingefügt
public void Insert (int value)
{
// Wenn Liste leer, dann first = neues Element ohne Nachfolger
// sonst suche Einfügeposition und füge ein
}
```

Einfügen

```
public void Insert (int value)
{
    if (first == null) {
        first = new Node(value, null);
    }
    else {
        // suche Einfügeposition und füge ein.
    }
}
```

Einfügen

```
public void Insert (int value)
{
     if (first == null) {
          first = new Node(value, null);
     }
     else {
          if (value <= first.value) // Fall a
               first = new Node(value, first);
          else { // Fälle b und c
               Value v = first;
               while (v != null && v.value > value) // Suche Nachfolgerknoten
                    v = v.next;
               Value n = new Node(value, v); // Neuer Knoten mit diesem Nachfolger, v= null möglich
               // Einfügen dieses Knotens nach Vorgänger von v
          }
     }
}
```

Einfügen

```
public void Insert (int value)
{
	if (first == null) {
		first = new Node(value, null);
	}
	else {
		if (value <= first.value) // Fall a
			first = new Node(value, first);
		else { // Fälle b und c
			Node prev = first; // != null
			Node v = prev.next;
			while (v != null && v.value > value){ // Suche Nachfolgerknoten
				prev = v;
				v = v.next;
			}
			Value n = new Node(value, v); // Neuer Knoten mit diesem Nachfolger, v= null möglich
			prev.next = v;
		}
	}
}
```

Vereinfachen, Testen, fertig.

```
public void Insert (int value)
{
    if (first == null || value <= first.value)
        first = new Node(value, first);
    else {
        Node prev = first;
        Node v = prev.next;
        while (v != null && v.value > value){ // suche Nachfolger und Vorgänger
            prev = v;
            v = v.next;
        }
        prev.next = new Node(value, v); // Einfügen
    }
}
```

POINT IN POLYGON ALGORITHMUS

PointInPolygon

```
public boolean PointInPolygon(int px, int py) {
  // Keine oder eine Ecke → return false
  // Zähle Kanten mit echtem Schnitt
  // mit horizontaler Geraden durch px, py
  // links von px
  // Wenn Anzahl ungerade, return true
  // Wenn Anzahl gerade, return false
}
```

PointInPolygon

```
public boolean PointInPolygon(int px, int py) {
  if (first == last) // keine oder eine Ecke
    return false;
  // Zähle Kanten mit echtem Schnitt
  // mit horizontaler Geraden durch px, py
  // links von px
  // Wenn Anzahl ungerade, return true
  // Wenn Anzahl gerade, return false
}
```

PointInPolygon

```
public boolean PointInPolygon(int px, int py) {
   if (first == last) // keine oder eine Ecke
      return false;
   Vertex v = first; // Invariante: first.next != first
   int count = 0;
   do {
      // wenn Schnitt von v – v.next
      // mit Horizontaler durch px, py links von px
      // dann inkrementiere count
      v = v.next;
   } while (v != first);
   return (count % 2 == 1);
}
```

PointInPolygon

```
public boolean PointInPolygon(int px, int py) {
   if (first == last) // keine oder eine Ecke
      return false;
   Vertex v = first;  // Invariante: first.next != first
   int count = 0;
   do {
      if (IntersectHorizontalLeft(px, py, v, v.next))
         ++count;
      v = v.next;
   } while (v != first);
   return (count % 2 == 1);
}
```

PointInPolygon

```
public boolean PointInPolygon(int px, int py) {
   if (first == last) // keine oder eine Ecke
      return false;
   Vertex v = first;  // Invariante: first.next != first
   boolean b = false; // "Zähl"variable (false -> true -> false ... )
   do {
      if (IntersectHorizontalLeft(px, py, v, v.next))
         b = !b;
      v= v.next;
   } while (v != first);
   return b;
}
```

Schnitt?

```
boolean IntersectHorizontalLeft(int x, int y, Vertex a, Vertex b) {
   // y-Wert zwischen p.y und q.y, nicht jedoch unten
   if (y <= p.y && y > q.y || y > p.y && y <= q.y) {
      // Schnittpunktberechnung
      double sx = p.x + (q.x - p.x) * (double)(y-p.y) / (q.y-p.y);
      // Schnittpunkt links
      return (sx <= x); // Schnittpunkt links
   }
   return false; // Kein Schnittpunkt
}
```

Konversion und Fliesskommarechnung nötig? → Umformen