Rekursive Datenstrukturen

```
struct llnode1 {
  llnode1 next;
};

struct llnode2 {
  llnode2& next;
};

struct llnode3 {
  llnode3* next;
};
```

Welche der Structs kompilieren?

1. 1,2,3
2. 1,3
3. 2,3
4. 1
5. 2
6. 3

Rekursive Datenstrukturen

```
struct llnode1 {
  llnode1 next;
};

struct llnode2 {
  llnode2& next;
};

struct llnode3 {
  llnode3* next;
};
```

Welche der Structs kompilieren?

1. 1,2,3
2. 1,3
3. 2,3 ●
4. 1
5. 2
6. 3

Rekursive Datenstrukturen

```
struct llnode1 {
  llnode1 next;
};
```

```
struct llnode2 {
  llnode2& next;
};
```

```
struct llnode3 {
  llnode3* next;
};
```

Rekursive Datenstrukturen

```
struct llnode1 {
  llnode1 next;
};
```

```
struct llnode2 {
  llnode2& next;
};
```

```
struct llnode3 {
  llnode3* next;
};
```

Struct **`llnode1`** kompiliert nicht, da er unendlich gross wäre.

Rekursive Datenstrukturen

```
struct llnode1 {
  llnode1 next;
};
```

```
struct llnode2 {
  llnode2& next;
};
```

```
struct llnode3 {
  llnode3* next;
};
```

`llnode2` kompiliert, aber da Referenzen immer initialisiert werden müssen, eignet er sich nur für zyklische Listen:

```
// Zyklisch, zwei Knoten
llnode2 n = llnode2 { llnode2 { n } };
```

Rekursive Datenstrukturen

```
struct llnode1 {
  llnode1 next;
};
```

```
struct llnode2 {
  llnode2& next;
};
```

```
struct llnode3 {
  llnode3* next;
};
```

`llnode2` kompiliert, aber da Referenzen immer initialisiert werden müssen, eignet er sich nur für zyklische Listen:

```
// Zyklisch, zwei Knoten
llnode2 n = llnode2 { llnode2 { n } };
```

Da Referenzen selbst nicht verändert („umgehängt“) werden können, kann die Liste nach der Erstellung nicht mehr vergrössert/-kleinert werden.

Rekursive Datenstrukturen

```
struct llnode1 {
  llnode1 next;
};
```

```
struct llnode2 {
  llnode2& next;
};
```

```
struct llnode3 {
  llnode3* next;
};
```

`llnode3` ist flexibler:

```
// n ist einzelner Knoten
llnode3 n = llnode3{nullptr};
```

Rekursive Datenstrukturen

```
struct llnode1 {
  llnode1 next;
};
```

```
struct llnode2 {
  llnode2& next;
};
```

```
struct llnode3 {
  llnode3* next;
};
```

`llnode3` ist flexibler:

```
// n ist einzelner Knoten
llnode3 n = llnode3{nullptr};

// n ist azyklische Liste mit zwei Knoten
n.next = new llnode3{nullptr};
```

Rekursive Datenstrukturen

```
struct llnode1 {
  llnode1 next;
};
```

```
struct llnode2 {
  llnode2& next;
};
```

```
struct llnode3 {
  llnode3* next;
};
```

`llnode3` ist flexibler:

```
// n ist einzelner Knoten
llnode3 n = llnode3{nullptr};

// n ist azyklische Liste mit zwei Knoten
n.next = new llnode3{nullptr};

// n ist zyklische Liste mit zwei Knoten
n.next->next = &n;
```