Pertanyaan bagus dan penting! Mari kita bahas satu per satu:

**✅ 1. Kenapa harus pakai -> (arrow operator)?**

Karena kamu sedang bekerja dengan **pointer ke struct**, bukan struct langsung.

**Contoh:**

node \*p = new node;

p adalah **pointer**, bukan objek node biasa. Jadi kalau mau mengakses isinya (misalnya data), kamu **tidak bisa** pakai titik (.), harus pakai ->:

p->data = 10; // ✅ benar

Kalau kamu pakai . di pointer, **error**, karena pointer bukan objek langsung:

p.data = 10; // ❌ salah

**Ingat:**

| **Tipe** | **Aksesnya** |
| --- | --- |
| Objek biasa | pakai . |
| Pointer ke objek | pakai -> |

**✅ 2. Kenapa harus pakai new node?**

Karena kamu membuat **node baru di heap memory**, bukan di stack.

**Penjelasan:**

head = new node;

Berarti kamu **meminta memori dinamis (di heap)** untuk satu struct node, dan head akan menyimpan **alamatnya**.

Kenapa di heap? Karena:

* Jumlah node bisa bertambah/berkurang **dinamis** saat program berjalan.
* Node tidak akan hilang meskipun keluar dari scope (fungsi main, atau blok lain).
* Stack terbatas ukurannya — heap lebih fleksibel.

Kalau kamu buat begini:

node temp;

temp.data = 10;

temp disimpan di stack, dan akan **hilang saat keluar dari scope**. Ini **nggak cocok untuk linked list**, karena node harus tetap hidup dan saling terhubung lewat next.

**Ilustrasi analogi gampang:**

node \*p = new node;

Bayangkan kamu bangun rumah (new node) dan p adalah alamatnya. Untuk buka pintu rumah itu, kamu butuh p->pintu.

**Kesimpulan:**

| **Hal** | **Penjelasan** |
| --- | --- |
| -> | Dipakai untuk mengakses anggota struct lewat pointer |
| new node | Membuat node baru di heap, supaya bisa hidup lama dan fleksibel dalam linked list |

Kalau kamu ingin, saya juga bisa tunjukkan versi tanpa new, tapi hanya cocok buat latihan kecil — tidak cocok buat linked list sebenarnya. Mau?