C++ Headers, Linkage & Storage Classes - Interview Notes

Header Files Key Principles

- Headers are copied (textually) into every .cpp file that includes them.
- Only declarations should be in header files.
  Avoid definitions (e.g., int a = 10;) in headers unless static or inline.

Linkage Types

Declaration	Linkage	Visibility
int x = 5; (in .cpp)	External (default)	Visible across translation units
static int x = 5;	Internal	Visible only in current .cpp file
extern int x;	No definition	Needs definition elsewhere

Common Mistakes and Their Fixes

1. int x = 10; in a header ODR violation
   Fix: use extern int x; in header and define int x = 10; in one .cpp

2. Global static variable in header internal linkage, each .cpp has its own copy
3. Static class member declared in header but not defined linker error
   Must define in one .cpp file

4. extern just declares, not defines
   extern in header, define once in .cpp

Good Practices

- Declare global vars in header using extern, define once in a .cpp
- Use #pragma once or include guards in headers
- Only define const, constexpr, or inline functions in headers
- Use static in headers only for separate copies
- Avoid logic-heavy code in headers

Pro-Level Edge Cases

- static int x; in header = multiple x (1 per .cpp)

C++ Headers, Linkage & Storage Classes - Interview Notes

- inline int x = 5; (C++17+) = safe in header (one definition)
- Class declarations in headers are fine, static definitions go in .cpp

Interview Tip

- Be clear on:
  Why static limits linkage
  Why extern needs definition elsewhere
  Why global defs in headers multiple defs
  How compiler & linker treat headers (copy-paste)