5/10/23 Date 75

(5) -> Dangling Pointers ->

↳ It is a pointer that points to a memory location that has been freed or deallocated. When you dereference a dangling pointer, the program may crash or exhibit undefined behavior.

pointer → assign to memory
↓
delete it
↳ is now
Not assign to anything → Set it to nullptr

ex -> int * p = new int;
*p = 42;
delete p;
cout << *p → Error

How Do They Arise?

(1) -> Freeing memory -> if you delete or free memory while a pointer is still assign to that point.

(2) -> Returning a pointer to local variable ->

↳ If you return a pointer to a local variable from function → then local var goes out of scope → It become Dangling pointers.

Spiral

Date 76

5/10/23

(3) -> using a pointer after it has been deleted or freed

Avoid: i) Use Smart pointer
↳ unique_ptr, shared_ptr & weak_ptr -> they automatically manage memory & prevents dangling pointers

-> they use RAII technique to ensure memory is freed when it is no longer needed.

ii) Use Reference instead of pointers
↳ They do not require explict memory management

iii) Avoid returning pointer to a local variable
↳ use reference var or smart pointer.
but if you are using raw pointer make sure the memory it points to is not freed before the pointer is used.

iv) Nullify after deleting or freeing memory: