* wild Pointers in C++.

A wild pointer is a pointer in C++ that is uninitialized or has been deleted. This means that the pointer does not point to a valid memory location, and accessing or dereferencing a wild pointer can result in undefined behavior.

int *p; → declared but uninitialized, this is a wild pointer.

↓

this pointer does not pointing to a valid memory location. If we try to access or dereference a wild pointer then this will given error.

* common scenario where wild pointer can occur :-

i) uninitialized pointers: If we declare a pointer variable without initializing it, the pointer will contain a random value that points to some memory location in the computers memory.
If we try to access or dereference this pointer, you may access memory that you should not access,

Date..................

which can cause you program to crash or behave unpredictably.

For example,

```
int *ptr;
cout << *ptr;
```

In the above example, ptr is declared but not initialized. when we try to dereference it using *ptr, we will get an undefined behavior.

ii) **Deleting pointers :** If we delete a pointer & then try to access or derefere-nce it, we will be accessing memory that has already been deallocated. This can cause your program to crash or behave unpredictably.

For example,

```
int *ptr;
delete ptr;
cout << *ptr;
```

iii) Pointer to non-existent variab
-le: If we create a pointer that points to a non-existent variable or object, we will be accessing a memory that does not contain a valid object. This can cause program to crash or behave unpredictably.

For example,

```
int *ptr = &x;
cout << *ptr;
```

To avoid wild pointers in c++, we should always initialize the pointers to a valid memory location or to "nullptr", if they are not pointing to anything.
And, we should always check if a pointer is pointing to a valid memo-ry location before dereferencing it to avoid accessing a wild pointer.