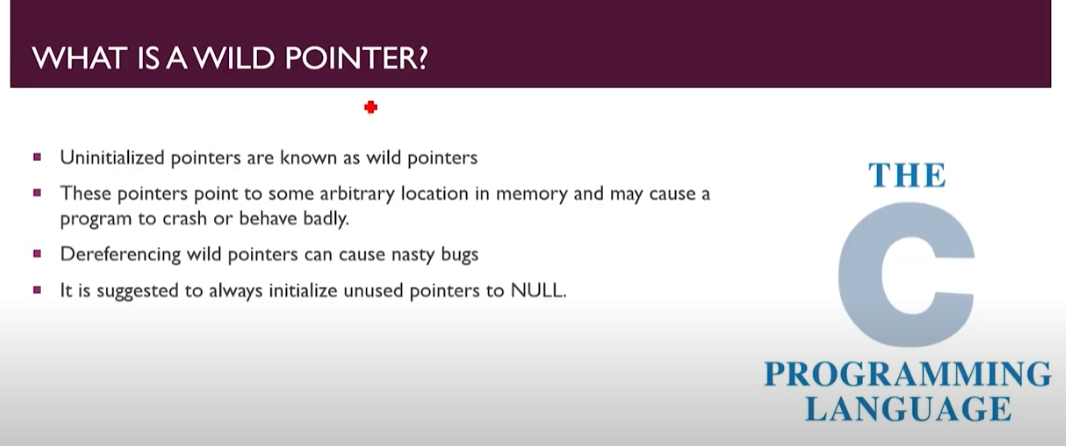
Content 40

Wild Pointers In C

A wild pointer is a simple concept, but a separate tutorial had to be made to make you aware. So, let's start with the definition.

***“Uninitialized pointers are known as void pointers.”***



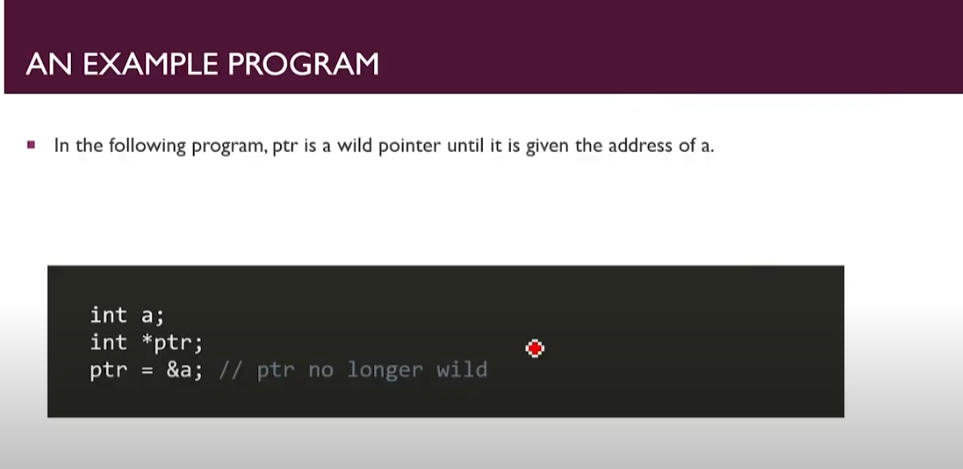
#### For Example:

int \*ptr;

In the above example, we created a pointer but didn't give it any value, so it becomes a wild pointer.

Its **disadvantage** is that it will store any garbage value in it, meaning it will hold some arbitrary memory location. Due to the storage of some random location, it can cause a lot of bugs in the program, and sometimes the programmer will not even be able to identify the cause.

Here It becomes Wild pointer



**Code1:**

1. #include<stdio.h>
2. int main(int argc, char const \*argv[])
3. {   int a=54;
4. int \*ptr; //Here it becomes wild Pointer
5. \*ptr=44;
6. return 0;
7. }

At line 5 the wild pointer is doing a nasty work. Hence it is going to Garbage values address and printing 34 there. Which was very bad.

#### Solution:

To avoid the bugs and errors it can cause in a program; we prefer to convert a void pointer to a **NULL pointer**. By doing so, our pointer will not point to any memory location, as it will point to 0 or NULL location. We can convert a wild pointer to a NULL pointer by merely placing it equal to NULL. Let us see it in C syntax.

#### Syntax:

int \*ptr = NULL;

So, we will adopt this method if we are not using our pointer to point at some memory location.

**“Here you can give a specific address or you can make it Null other wise it will ruin your program.”**

**But above if the program is like;**

1. #include<stdio.h>
2. int main(int argc, char const \*argv[])
3. {   int a=54;
4. int \*ptr;
5. ptr=&a;
6. \*ptr=44;
7. return 0;
8. }

#### At line 5 no we point ptr to a which means ptr is no more as Wild Pointer.