**nullptr in C++**

Consider the following C++ program that shows problem with NULL (need of nullptr)

C++

// C++ program to demonstrate problem with NULL

#include <bits/stdc++.h>

using namespace std;

// function with integer argument

void fun(int N) { cout << "fun(int)"; return;}

// Overloaded function with char pointer argument

void fun(char\* s) { cout << "fun(char \*)"; return;}

int main()

{

// Ideally, it should have called fun(char \*),

// but it causes compiler error.

fun(NULL);

}

Output:

16:13: error: call of overloaded 'fun(NULL)' is ambiguous

fun(NULL);

**What is the problem with above program?**

NULL is typically defined as (void \*)0 and conversion of NULL to integral types is allowed. So the function call fun(NULL) becomes ambiguous.

C++

// This program compiles (may produce warning)

#include<stdio.h>

int main()

{

int x = NULL;

}

**How does nullptr solve the problem?**

In the above program, if we replace NULL with nullptr, we get the output as “fun(char \*)”.  
nullptr is a keyword that can be used at all places where NULL is expected. Like NULL, nullptr is implicitly convertible and comparable to any pointer type. **Unlike NULL, it is not implicitly convertible or comparable to integral types**.

C++

// This program does NOT compile

#include<stdio.h>

int main()

{

int x = nullptr;

}

Output:

Compiler Error

As a side note, **nullptr is convertible to bool.**

C++

// This program compiles

#include<iostream>

using namespace std;

int main()

{

int \*ptr = nullptr;

// Below line compiles

if (ptr) { cout << "true"; }

else { cout << "false"; }

}

**Output**

false