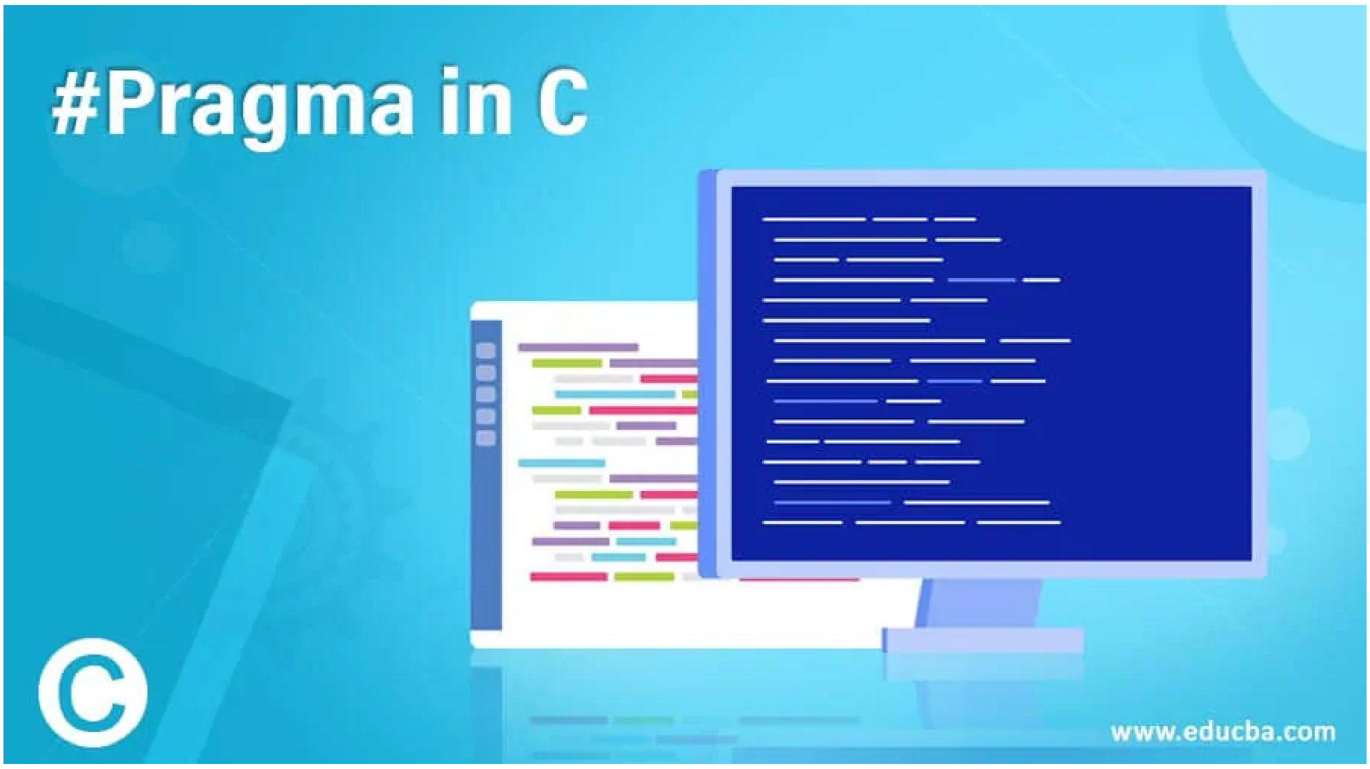
EDUCBA
(https://www.educba.com/software-development/)

← (https://www.educba.com/fscanf-in-c/)

→ (https://www.educba.com/sharp-ifndef-in-c/)



Introduction to #Pragma in C

The #pragma in C is a directive that is provided by the C standard in order to provide ex
required details to the C compiler. These extra details can be anything that was somehow not passed within the program or the code logic. These directives, known as pragma are prefixed

EDUCBA
(https://www.educba.com/software-development/)

Syntax

There are basically two types of syntaxes in which the pragma directive can be implemented in a program. For both syntaxes, the keywords and parameters are the same but the only difference is how we initiate the code statement.

Syntax #1

```
#pragma token-string
```

Here we have a standard way of implementing a pragma directive within a program, which begins with a hashtag followed by the pragma keyword and then a simple token string.

Syntax 2 is similar to the first one with the only difference that we use a different operator before the keyword pragma:

```
__pragma( token-string )
```

With our second format of pragma syntax, we have two underscores prior to the pragm QUIZ keyword. Few of the most commonly used token strings are a startup, exit, warn, etc. based on the requirement, these tokens are implemented. We will learn about these tokens with the help

EDUCBA (https://www.educba.com/software-development/)

How #Pragma work in C?

We will now move on to understanding how the #pragma directive in C actually works. We have already learned that the pragma directive is used to notify the compiler of any special event that is to happen. That is exactly how the pragma directive works within C, these pragma

directives are executed before compilation and followed by preprocessing. This way, the pragma directive is able to notify the compiler to what to ignore and what not to ignore. It is also recommended to use the pragma directive once over in a program.

EDUCBA
(https://www.educba.com/software-development/)

Examples:

Our first example is a simple output statement. We will simply implement the pragma directive with two of its tokens which are startup and exit. The code for the program is as follows:

Code:

```
#include<stdio.h>
#include<conio.h>
void func() ;
#pragma startup func
#pragma exit func
void func(){
printf("\n Simple statement for a pragma directive.");
getch();
}
void main(){
printf("\n This is a sample to test pragma directive.");
getch();
}
```

QUIZ

Code Explanation: Started with a simple system include files that are required for the p
Then we have a single function, followed by the pragma definitions for both startup and exit.
Then we have another function that holds the print statement. Followed by the getch function,

EDUCBA
(https://www.educba.com/software-development/)

[redacted]

Code:

```
#include<stdio.h>
#pragma warn -rvl /* return value */
#pragma warn -par /* parameter never used */
#pragma warn -rch /*unreachable code */
int show(int x) {
printf("\n This is a sample to demonstrate working of pragma.  ");
}
int main() {
show(10);
return 0;
}
```

Code Explanation: Started with single include file then a few pragma derivatives. The pragma derivatives we are implementing here are -rvl, -par, and -rch. These are all part of warn token and what they means is -rvl is for the return value, then the -par is for the parameter, which is never used. And the -rch is for unreachable code. Then we have our code for printing a simple output statement. Then we have our main code, within which we pass our show function and a simple return. Upon successful execution, the code will run smoothly and return a statement that we included.

QUIZ

As expected, the output is the simple statement that we intended to be printed. Moving, we will

EDUCBA
(https://www.educba.com/software-development/)

```
#include<stdio.h>
#pragma GCC poison printf
int main() {
int a=10;
if(a==10)  {
printf("This is statement, a print line which will not be
printed.");
}
else
printf("We wanted to stumble upon an error.");
return 0;
}
```

Code Explanation: Started just as all other examples, then we have our pragma derivative, followed by the keywords GCC poison and a function named that is supposed to be poisoned. Then we have our main, an integer, and an if statement, within which we have our print statements and a return. But we have poisoned our printf statements meaning an error is supposed to occur when the code reaches the printf line.

QUIZ

The poison, when used with pragma GCC, is used to identify and remove some code or part of the code and also to make sure that it does not bother any other part, then we use the GCC poison. we simply define the pragma followed by the GCC poison keyword and the final

EDUCBA
(https://www.educba.com/software-development/)

As expected, the code has given out errors exactly with printf statements. There are many other gcc tokens that can be implemented other than poison, like GCC dependency, warning, header, etc. These string tokens carry a purpose and can be implemented as per the situation.

Conclusion

To conclude, the #pragma in C is a special directive, assigned by the C standards and is responsible for turning on or off a few features. Basically, the pragma directive informs the C compiler that a special request is being made. These directives can be different for different compilers, meaning they are compiler-specific.

Recommended Articles

This is a guide to #Pragma in C. Here we discuss how #Pragma work in C and along with the examples with explanation. You may also have a look at the following articles to learn more –

1. Structure Padding in C (https://www.educba.com/structure-padding-in-c/)
2. Preprocessor Directives in C (https://www.educba.com/preprocessor-directives-in-c/)
3. Convert int to String C# (https://www.educba.com/convert-int-to-string-c-sharp/)
4. C# SortedSet (https://www.educba.com/c-sharp-sortedset/)

EDUCBA

(https://www.educba.com/software-development/)

Learn More

(https://www.educba.com/software-development/courses/software-development-course/?btnz=edu-blg-inline-banner3)

About Us

Blog (https://www.educba.com/blog/?source=footer)

Who is EDUCBA? (https://www.educba.com/about-us/?source=footer)

Sign Up (https://www.educba.com/software-development/signup/?source=footer)

Corporate Training (https://www.educba.com/corporate/?source=footer)

Certificate from Top Institutions (https://www.educba.com/educbalive/?source=footer)

Contact Us (https://www.educba.com/contact-us/?source=footer)

Verifiable Certificate (https://www.educba.com/software-development/verifiable-certificate/?source=footer)

Reviews (https://www.educba.com/software-development/reviews/?source=footer)

QUIZ

EDUCBA (https://www.educba.com/software-development/)

Apps

iPhone & iPad (https://itunes.apple.com/in/app/educba-learning-app/id1341654580?mt=8)

Android (https://play.google.com/store/apps/details?id=com.educba.www)

Resources

Free Courses (https://www.educba.com/software-development/free-courses/?source=footer)

Java Tutorials (https://www.educba.com/software-development/software-development-tutorials/java-tutorial/?source=footer)

Python Tutorials (https://www.educba.com/software-development/software-development-tutorials/python-tutorial/?source=footer)

All Tutorials (https://www.educba.com/software-development/software-development-tutorials/?source=footer)

Certification Courses

All Courses (https://www.educba.com/software-development/courses/?source=footer)

Software Development Course - All in One Bundle (https://www.educba.com/software-development/courses/software-development-course/?source=footer)

Become a Python Developer (https://www.educba.com/software-development/courses/python-certification-course/?source=footer)

Java Course (https://www.educba.com/software-development/courses/java-course/?source=footer)

Become a Selenium Automation Tester (https://www.educba.com/softwa development/courses/selenium-training-certification/?source=footer)

QUIZ

Become an IoT Developer (https://www.educba.com/software-development/courses/iot-course/?source=footer)