#pragma Directive in C/C++

This directive is a special purpose directiv d to turn on or off some features. This type of directives are compiler-specific i.e., they va mpiler to compiler. Some of the #pragma directives are discussed below:

1. **#pragma startup and #pragma exit**: These directives helps us to specify the functions that are needed to run before program startup(before the control passes to main()) and just before program exit (just before the control returns from main()).
   Note: Below program will not w
   Look at the below program:

```
#include<stdio.h>

void func1();
void func2();

#pragma startup func1
#pragma exit func2

void func1()
{
    printf("Inside func1()\n");
}

void func2()
{
    printf("Inside func2()\n");
}

int main()
{
    printf("Inside main()\n");

    return 0;
}
```

Output:

```
Inside func1()
Inside main()
Inside func2()
```

The above code will produce the output as given below when run on GCC compilers:

```
Inside main()
```

This happens because GCC does not support **#pragma startup or exit**. However, you can use the below code for a similar output on GCC compilers.

```
#include<stdio.h>

void func1();
void func2();

void __attribute__((constructor)) func1();
void __attribute__((destructor)) func2();

void func1()
{
    printf("Inside func1()\n");
}

void func2()
{
    printf("Inside func2()\n");
}

int main()
{
    printf("Inside main()\n");

    return 0;
}
```

Output:

```
Inside func1()
Inside main()
Inside func2()
```

2. **#pragma warn Directive**: This directive is used to hide the warning messages which are displayed during compilation. This may be useful for us when we have a large program and we want to solve all the errors before looking on warnings then by using it we can focus on errors by hiding all warnings. we can again let the warnings be visible by making slight changes in syntax.
Syntax:

```
#pragma warn +xxx (To show the warning)
#pragma warn -xxx (To hide the warning)
#pragma warn .xxx (To toggle between hide and show)
```

We can hide the warnings as shown below:

- **#pragma warn -rvl**: This directive hides those warning which are raised when a function which is supposed to return a value does not return a value.
- **#pragma warn -par**: This directive hides those warning which are raised when a function does not uses the parameters passed to it.
- **#pragma warn -rch**: This directive hides those warning which are raised when a code is unreachable. For example: any code written after the return statement in a function is unreachable.

Example:

```
// Example to explain the working of
// #pragma warn directive
// This program is compatible with C/C++ compiler

#include<stdio.h>

#pragma warn -rvl /* return value */
#pragma warn -par /* parameter never used */
#pragma warn -rch /*unreachable code */

int show(int x)
{
    // parameter x is never used in
    // the function

    printf("GEEKSFORGEEKS");

    // function does not have a
    // return statement
}

int main()
{
    show(10);

    return 0;
}
```

Output:

```
GEEKSFORGEEKS
```

The above program compiles successfully without any warnings to give the output "GEEKSFORGEEKS".

3. **#pragma GCC poison**: This directive is supported by the GCC compiler and is used to remove an identifier completely from the program. If we want to block an identifier then we can use the **#pragma GCC poison** directive.
Example:

```
// Program to illustrate the
// #pragma GCC poison directive

#include<stdio.h>

#pragma GCC poison printf

int main()
{
    int a=10;

    if(a==10)
    {
        printf("GEEKSFORGEEKS");
    }
    else
        printf("bye");

    return 0;
}
```

The above program will give the below error:

```
prog.c: In function 'main':
prog.c:14:9: error: attempt to use poisoned "printf"
         printf("GEEKSFORGEEKS");
         ^
prog.c:17:9: error: attempt to use poisoned "printf"
         printf("bye");
         ^
```

4. **#pragma GCC dependency**: The #pragma GCC dependency allows you to check the relative dates of the current file and another file. If the other file is more recent than the current file, a warning is issued. This is useful if the current file is derived from the other file, and should be regenerated.
Syntax:

```
#pragma GCC dependency "parse.y"
#pragma GCC dependency "/usr/include/time.h" rerun fixincludes
```

5. **#pragma GCC system_header**: This pragma takes no arguments. It causes the rest of the code in the current file to be treated as if it came from a system header.
6. **#pragma once**: The #pragma once directive has a very simple concept. The header file containing this directive is included only once even if the programmer includes it multiple times during a compilation. This is not included in any ISO C++ standard. This directive works similar to the

#include guard idiom. Use of #pragma once saves the program from multiple inclusion optimisation.

Syntax:

```
#pragma once
```

Recommended Posts:

Speed up Code executions with help of Pragma in C/C++

C/C++ #include directive with Examples

Inline namespaces and usage of the "using" directive inside namespaces

Implementing upper_bound() and lower_bound() for Ordered Set in C++

How to flatten a Vector of Vectors or 2D Vector in C++

<complex.h> header file in C with Examples

Remove odd frequency characters from the string

Different ways to use Const with Reference to a Pointer in C++

std::to_address in C++ with Examples

Program to create Custom Vector Class in C++

std::is_trivially_copy_constructible in C/C++

Difference between Python and C++

tgamma() method in C/C++ with Examples

boost::type_traits::is_array Template in C++

VineetKumar2
Check out this Author's contributed articles.

If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please Improve this article if you find anything incorrect by clicking on the "Improve Article" button below.

Article Tags : C C++ C-Macro & Preprocessor Macro & Preprocessor

Practice Tags : C CPP

11

1.8

To-do Done

Based on **5** vote(s)

Feedback/ Suggest Improvement Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

A computer science portal for geeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org