_Generics Keyword in C

GeeksforGeeks ⋮⋮ 8/29/2017

The **_Generic** keyword in C is used to implement a generic code that can execute statements based on the type of arguments provided. It can be implemented with C macros to imitate function overloading and also helps to write type-safe code that can operate on different data types without having to manually write separate logic for each type. The _Generic keyword was first introduced in the C11 Standard.

Syntax of _Generic in C

```
_Generic( (expression),

    type_1: statements,

    type_2: statements,

    .

    .

    default: statements

)
```

where,

- **expression:** This test expression's value type is used to determine which statements to execute.
- **default:** Specifies the statement to be executed when no matching type is found.
- **type_1, type_2...:** Types for which we have defined the genetic code.
- **default**: Fallback value if no matching type is found.

The syntax is similar to the switch statement, where we have different statements for different case values of the test expression. The expression is evaluated, and based on the type of the expression, the

corresponding value for that type is returned and If none of the types match, the default value (if provided) is chosen.

Using _Generic in C

We can use the _Generic keyword anywhere in our code where we need some generic code. The below example demonstrates how to use _Generic keyword:

Example:

```
// C program to illustrate macro function.

#include <stdio.h>

int main(void)

{

    // _Generic keyword acts as a switch that chooses

    // operation based on data type of argument.

    printf("%d\n", _Generic(1.0L, float : 1, double : 2,

                            long double : 3, default : 0));


    printf("%d\n", _Generic(1L, float : 1, double : 2,

                            long double : 3, default : 0));


    printf("%d\n", _Generic(1.0L, float : 1, double : 2,

                            long double : 3));

    return 0;

}
```

Output

```
3
0
3
```

_Generic in Macros

A major drawback of Macro in C is that the arguments lack type checking, i.e. a macro can operate on different types of variables(like char, int, double,..) without type checking. We can use _Generic to include the statements for different argument types in such macros.

Example

```
// C program to illustrate macro function.

#include <stdio.h>


// Defining the macro with generic code for

//different argument types

#define geeks(T) _Generic((T), \

        char* : "String", \

        int : "Integer", \

        long : "Long Integer", \

        default : "Others")


int main(void)

{


    // Here A is a string.

    printf("%s\n", geeks("A"));


    // floating point value

    printf("%s\n", geeks(5));


      // float type which is not defined in the macro

      printf("%s", geeks(5.12));


    return 0;
```

```
}
```

Output

```
String
Integer
Others
```

As we can see, different expressions are returned for different datatypes. This behaviour is similar to C++ function overloading where the parameter types specify which version of the function will be executed. So, we can use the _Generic in C to provide some features of function overloading in our program.

Advantages of _Generic

Following are some main advantages of _Generic in C:

- It can be used to stimulate function overloading when used with macros.
- Allows the functionality to execute the code based on the type of parameters.

Disadvantages of _Generic

The major disadvantages of _Generic are as follows:

- The major disadvantage of _Generic is its usage in macros as they are vulnerable to errors.
- It is difficult to understand and debug in case of error due to complex syntax and limited view of expanded code.