Const in C

The **const** keyword in C is used to declare a variable as read-only, meaning its value cannot be changed during program execution. When a variable is declared with the **const** keyword, its value cannot be modified after initialization.

Example:

C

```c
#include <stdio.h>
int main()
{
    const int x = 10;
    x = x + 1;
    printf("%d", x);
    return 0;
}
```

Output

```
./Solution.c: In function 'main':
./Solution.c:5:7: error: assignment of read-only variable 'x'
     x = x + 1;
       ^
```

Explanation: In this program, trying to modify **x** results in a compilation error because **x** is declared as **const**.

Although const variables are intended to be read-only, their values can technically be modified using **pointers**. However, this is not recommended, as it defeats the purpose of declaring a variable as const.

Practical Example of Constant

Using **const** to declare constants is particularly useful for values that are reused throughout the program, such as the mathematical constant Pi.

C

```c
#include <stdio.h>
const double PI = 3.14;
int main()
{
```

```
    int r;
    printf("Enter value of r:\n");
    scanf("%d", &r);
    printf("Area is: %f", PI * r * r);
    return 0;
}
```

Output

```
Enter value of r:
2
Area is: 12.560000
```

Here, the const declaration ensures that PI remains unchanged during program execution.

Macros for Creating Constants

In C, we can also create constants like Pi using macros with the #define directive rather than using const keyword.

Example:

C

```
#include <stdio.h>
#define PI 3.14
int main()
{
    int r;
    printf("Enter value of r:\n");
    scanf("%d", &r);
    printf("Area is: %f", PI * r * r);
    return 0;
}
```

Output

```
Enter value of r:
2
Area is: 12.560000
```

While macros are a quick way to define constants, they are not recommended because:

- Macros simply perform a text substitution without considering the data type.
- Once defined, macros are available throughout the file, potentially causing conflicts.