Defining Named Constants

In C, **named constants** are used to **represent fixed values in a program that do not change during its execution**. There are two ways to define these constants:

1. Using the **#define** preprocessor directive or
2. Using the **const** keyword.

Named constants are beneficial for code readability and maintenance, as they make it clear what the constant represents and allow you to easily update its value if needed.

Using #define Directive:

```
#include <stdio.h>

// Define a named constant using #define
#define PI 3.14159

int main()
{
    // Using the named constant
    double radius = 5.0;
    double area = PI * radius * radius;

    printf("Area of the circle: %f\n", area);

    return 0;
}
```

In the above example, PI is a named constant representing the value of π (pi). Using **#define** is a simple way to create a constant, but note that it **doesn't reserve memory for the constant**; instead, it **performs direct textual substitution** as it is preprocessor directive.

Using const Keyword:

```
#include <stdio.h>

int main()
{
    // Define a named constant using const keyword
    const double PI = 3.14159;

    // Using the named constant
    double radius = 5.0;
    double area = PI * radius * radius;

    printf("Area of the circle: %f\n", area);
    return 0;
}
```

Using the **const keyword** is another way to define named constants. This method **allows you to specify a data type and reserves memory for the constant**. It is generally considered a more modern and type-safe approach.

Choose between #define and const based on your needs and coding conventions. const is preferred when you want to specify a data type, and it offers better type safety. However, #define can be more flexible in some situations, especially when dealing with simple constants and macros.