C Comments

The **comments in C** are human-readable explanations or notes in the source code of a C program. A comment makes the program easier to read and understand. These are the statements that are not executed by the compiler or an interpreter.
It is considered to be a good practice to document our code using comments.

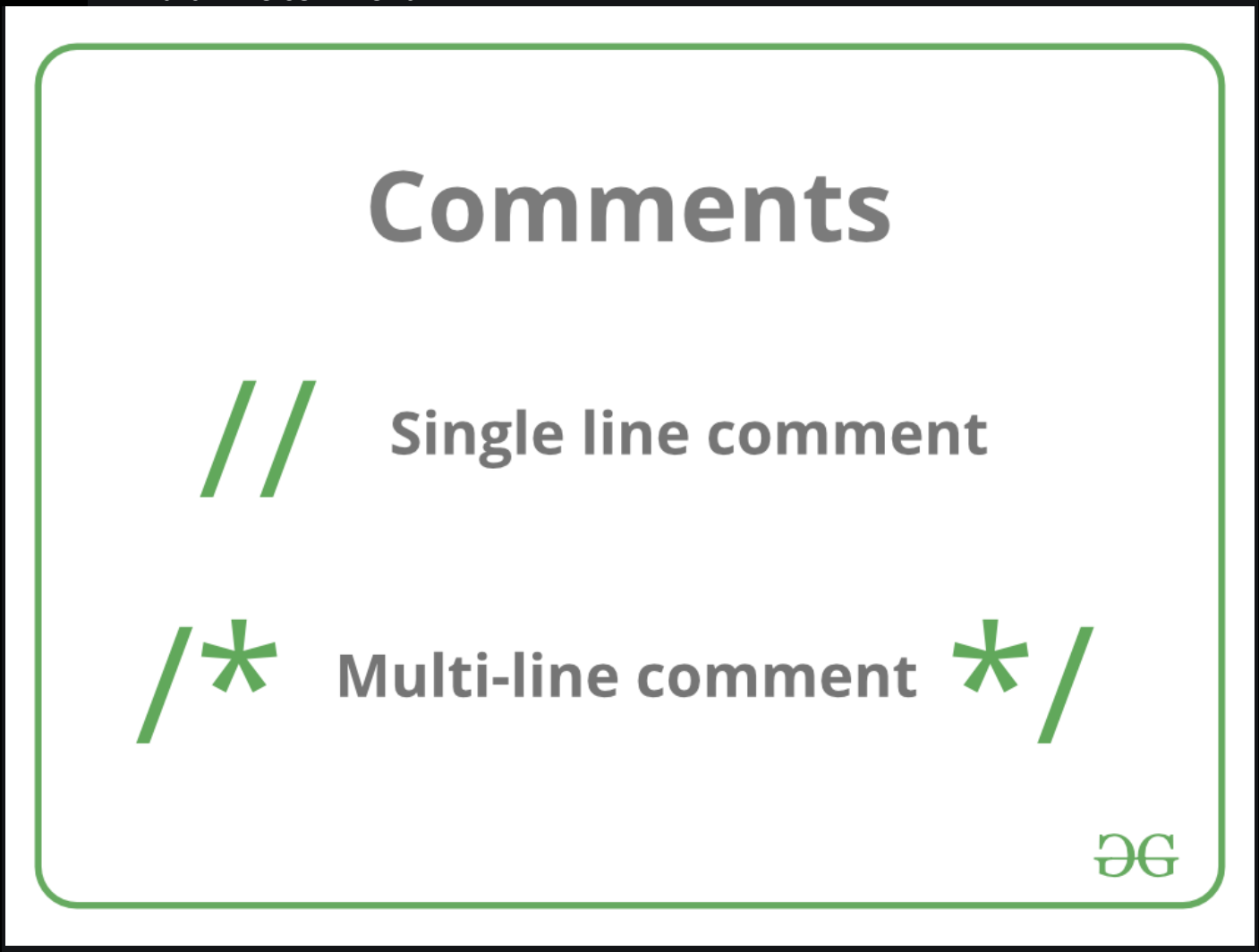
When and Why to use Comments in C programming?

1. A person reading a large code will be bemused if comments are not provided about details of the program.
2. C Comments are a way to make a code more readable by providing more descriptions.
3. C Comments can include a description of an algorithm to make code understandable.
4. C Comments can be used to prevent the execution of some parts of the code.

Types of comments in C

In C there are two types of comments in C language:

- **Single-line comment**
- **Multi-line comment**



Types of Comments in C

1. Single-line Comment in C

A single-line comment in C starts with (//) double forward slash. It extends till the end of the line and we don't need to specify its end.

Syntax of Single Line C Comment

```
// This is a single line comment
```

Example 1: C Program to illustrate single-line comment

- C

```c
// C program to illustrate
// use of single-line comment
#include <stdio.h>

int main(void)
{
    // This is a single-line comment
    printf("Welcome to GeeksforGeeks");
    return 0;
}
```

Output:

```
Welcome to GeeksforGeeks
```

Comment at End of Code Line

We can also create a comment that displays at the end of a line of code using a single-line comment. But generally, it's better to practice putting the comment before the line of code.

Example:

- C

```c
// C program to demonstrate commenting after line of code
#include <stdio.h>

int main() {
    // single line comment here

      printf("Welcome to GeeksforGeeks"); // comment here
    return 0;
}
```

Output

```
Welcome to GeeksforGeeks
```

2. Multi-line Comment in C

The Multi-line comment in C starts with a forward slash and asterisk (/*) and ends with an asterisk and forward slash (*/). Any text between /* and */ is treated as a comment and is ignored by the compiler.
It can apply comments to multiple lines in the program.

Syntax of Multi-Line C Comment

```
/*Comment starts
    continues
    continues
    .
    .
    .
Comment ends*/
```

Example 2: C Program to illustrate the multi-line comment

- C

```c
/* C program to illustrate
use of
multi-line comment */
#include <stdio.h>
int main(void)
{
    /*
    This is a
    multi-line comment
    */

      /*
    This comment contains some code which
    will not be executed.
    printf("Code enclosed in Comment");
    */
```

```
    printf("Welcome to GeeksforGeeks");

    return 0;

}
```

Output:

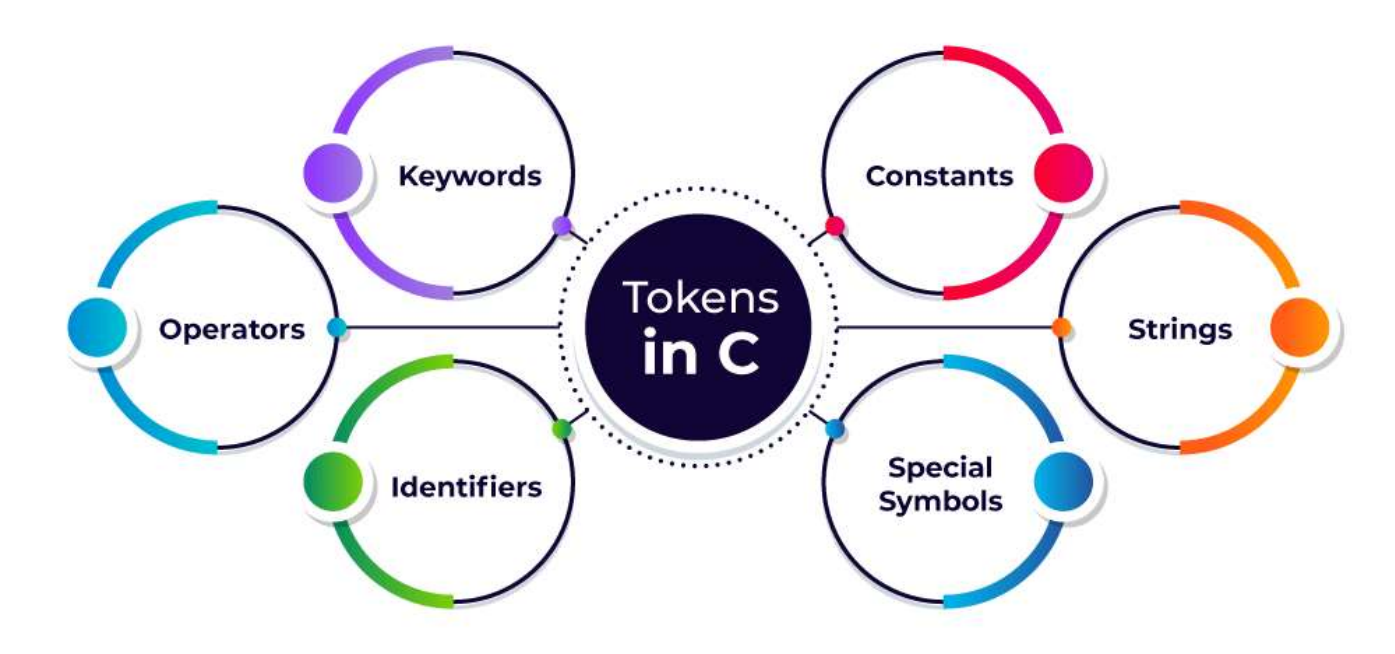
```
Welcome to GeeksforGeeks
```

Tokens in C

A token in C can be defined as the smallest individual element of the C programming language that is meaningful to the compiler. It is the basic component of a C program.

Types of Tokens in C

The tokens of C language can be classified into six types based on the functions they are used to perform. The types of C tokens are as follows:



1. **Keywords**
2. **Identifiers**
3. **Constants**
4. **Strings**
5. **Special Symbols**
6. **Operators**

1. C Token – Keywords

The keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program. Since keywords are referred names for a compiler, they can't be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed. You cannot redefine keywords. However, you can specify the text to be substituted for keywords before compilation by using C preprocessor directives. **C** language supports **32** keywords which are given below:

auto	**double**	**int**	**struct**
break	**else**	**long**	**switch**
case	**enum**	**register**	**typedef**
char	**extern**	**return**	**union**
const	**float**	**short**	**unsigned**
continue	**for**	**signed**	**void**

```
default     goto        sizeof     volatile
do          if          static     while
```

2. C Token – Identifiers

Identifiers are used as the general terminology for the naming of variables, functions, and arrays. These are user-defined names consisting of an arbitrarily long sequence of letters and digits with either a letter or the underscore(_) as a first character. Identifier names must differ in spelling and case from any keywords. You cannot use keywords as identifiers; they are reserved for special use. Once declared, you can use the identifier in later program statements to refer to the associated value. A special identifier called a statement label can be used in goto statements.

Rules for Naming Identifiers

Certain rules should be followed while naming c identifiers which are as follows:

- They must begin with a letter or underscore(_).
- They must consist of only letters, digits, or underscore. No other special character is allowed.
- It should not be a keyword.
- It must not contain white space.
- It should be up to 31 characters long as only the first 31 characters are significant.

Note: *Identifiers are case-sensitive so names like variable and Variable will be treated as different.*

For example,

- **main:** method name.
- **a:** variable name.

3. C Token – Constants

The constants refer to the variables with fixed values. They are like normal variables but with the difference that their values can not be modified in the program once they are defined.

Constants may belong to any of the data types.

Examples of Constants in C

```
const int c_var = 20;
const int* const ptr = &c_var;
```

4. C Token – Strings

Strings are nothing but an array of characters ended with a null character (‘\0’). This null character indicates the end of the string. Strings are always enclosed in double quotes. Whereas, a character is enclosed in single quotes in C and C++.

Examples of String

```
char string[20] = {‘g’, ’e’, ‘e’, ‘k’, ‘s’, ‘f’, ‘o’, ‘r’, ‘g’, ’e’, ‘e’, ‘k’, ‘s’, ‘\0’};
char string[20] = “geeksforgeeks”;
char string [] = “geeksforgeeks”;
```

5. C Token – Special Symbols

The following special symbols are used in C having some special meaning and thus, cannot be used for some other purpose. Some of these are listed below:

- **Brackets[]:** Opening and closing brackets are used as array element references. These indicate single and multidimensional subscripts.
- **Parentheses():** These special symbols are used to indicate function calls and function parameters.
- **Braces{}:** These opening and ending curly braces mark the start and end of a block of code containing more than one executable statement.
- **Comma (,):** It is used to separate more than one statement like for separating parameters in function calls.
- **Colon(:):** It is an operator that essentially invokes something called an initialization list.
- **Semicolon(;):** It is known as a statement terminator. It indicates the end of one logical entity. That’s why each individual statement must be ended with a semicolon.
- **Asterisk (*):** It is used to create a pointer variable and for the multiplication of variables.
- **Assignment operator(=):** It is used to assign values and for logical operation validation.
- **Pre-processor (#):** The preprocessor is a macro processor that is used automatically by the compiler to transform your program before actual compilation.
- **Period (.):** Used to access members of a structure or union.
- **Tilde(~):** Used as a destructor to free some space from memory.

6. C Token – Operators

Operators are symbols that trigger an action when applied to C variables and other objects. The data items on which operators act are called operands.
Depending on the number of operands that an operator can act upon, operators can be classified as follows:

- **Unary Operators:** Those operators that require only a single operand to act upon are known as unary operators.For Example increment and decrement operators
- **Binary Operators:** Those operators that require two operands to act upon are called binary operators. Binary operators can further are classified into:

1. Arithmetic operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Bitwise Operator

- **Ternary Operator**: The operator that requires three operands to act upon is called the ternary operator. Conditional Operator(?) is also called the ternary operator.