Special characters in C

Some special characters are used in C, and they have a special meaning which cannot be used for another purpose.

- **Square brackets [ ]:** The opening and closing brackets represent the single and multidimensional subscripts.
- **Simple brackets ( ):** It is used in function declaration and function calling. For example, printf() is a pre-defined function.
- **Curly braces { }:** It is used in the opening and closing of the code. It is used in the opening and closing of the loops.
- **Comma (,):** It is used for separating for more than one statement and for example, separating function parameters in a function call, separating the variable when printing the value of more than one variable using a single printf statement.
- **Hash/pre-processor (#):** It is used for pre-processor directive. It basically denotes that we are using the header file.
- **Asterisk (*):** This symbol is used to represent pointers and also used as an operator for multiplication.
- **Tilde (~):** It is used as a destructor to free memory.
- **Period (.):** It is used to access a member of a structure or a union

Programming Errors in C :

Errors are the problems or the faults that occur in the program, which makes the behaviour of the program abnormal, and experienced developers can also make these faults. Programming errors are also known as the bugs or faults, and the process of removing these bugs is known as **debugging**.

These errors are detected either during the time of compilation or execution. Thus, the errors must be removed from the program for the successful execution of the program.

There are mainly five types of errors exist in C programming:

- **Syntax error**
- **Run-time error**
- **Linker error**
- **Logical error**
- **Semantic error**

Syntax error :

Syntax errors are also known as the compilation errors as they occurred at the compilation time, or we can say that the syntax errors are thrown by the compilers. These errors are mainly occurred due to the mistakes while typing or do not follow the syntax of the specified programming language. These mistakes are generally made by beginners only because they are new to the language. These errors can be easily debugged or corrected.

For example:

1. If we want to declare the variable of type integer,
2. **int** a; //this is the correct form
3. Int a; //this is an incorrect form.

Commonly occurred syntax errors are:

- If we miss the parenthesis (}) while writing the code.
- Displaying the value of a variable without its declaration.
- If we miss the semicolon (;) at the end of the statement.

Let us understand through an example.

```
#include<stdio.h>
int main()
{
a=10;
printf("The value of a is:%d",a);
return 0;
}
```

In the above output, we observe that the code throws the error that 'a' is undeclared. This error is nothing but the syntax error only.

There can be another possibility in which the syntax error can exist, i.e., if we make mistakes in the basic construct. Let's understand this scenario through an example.

```
#include<stdio.h>
int main()
{
int a=2;
if(.)//syntax error
printf("a is greater than 1");
return 0;
}
```

In the above code, we put the (.) instead of condition in 'if', so this generates the syntax error as shown in the below screenshot.

Run-time error :

Sometimes the errors exist during the execution-time even after the successful compilation known as run-time errors. When the program is running, and it is not able to perform the operation is the main cause of the run-time error. The division by zero is the common example of the run-time error. These errors are very difficult to find, as the compiler does not point to these errors.

Let us understand through an example.

```
int main()
{
int a=2;
int b=2/0;
printf("The value of b is:%d",b);
return 0;
}
```

In the above output, we observe that the code shows the run-time error, i.e., division by zero.