C Programming Language

C Identifiers

- C identifiers represent the name in the C program, for example, variables, functions, arrays, structures, unions, labels, etc.
- An identifier can be composed of letters such as uppercase, lowercase letters, underscore, digits, but the starting letter should be either an alphabet or an underscore.
- If the identifier is not used in the external linkage, then it is called as an internal identifier.
- If the identifier is used in the external linkage, then it is called as an external identifier.

- We can say that an identifier is a collection of alphanumeric characters that begins either with an alphabetical character or an underscore, which are used to represent various programming elements such as variables, functions, arrays, structures, unions, labels, etc.
- There are 52 alphabetical characters (uppercase and lowercase), underscore character, and ten numerical digits (0-9) that represent the identifiers.
- There is a total of 63 alphanumerical characters that represent the identifiers

Rules for constructing C identifiers

- The first character of an identifier should be either an alphabet or an underscore, and then it can be followed by any of the character, digit, or underscore.
- It should not begin with any numerical digit.
- In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
- Commas or blank spaces cannot be specified within an identifier.
- Keywords cannot be represented as an identifier.
- The length of the identifiers should not be more than 31 characters.
- Identifiers should be written in such a way that it is meaningful, short, and easy to read.

- Example of valid identifiers
- total, sum, average, _m _, sum_1, etc.
- Example of invalid identifiers
- 2sum (starts with a numerical digit)
- int (reserved word)
- char (reserved word)
- m+n (special character, i.e., '+')

- Types of identifiers
- Internal identifier
- External identifier
- Internal Identifier
- If the identifier is not used in the external linkage, then it is known as an internal identifier. The internal identifiers can be local variables.
- External Identifier
- If the identifier is used in the external linkage, then it is known as an external identifier. The external identifiers can be function names, global variables.

Differences between Keyword and Identifier

Keyword	Identifier
Keyword is a pre-defined word.	The identifier is a user-defined word
It must be written in a lowercase letter.	It can be written in both lowercase and uppercase letters.
Its meaning is pre-defined in the c compiler.	Its meaning is not defined in the c compiler.
It is a combination of alphabetical characters.	It is a combination of alphanumeric characters.
It does not contain the underscore character.	It can contain the underscore character.

- Let's understand through an example.

```
int main()
{
    int a=10;
    int A=20;
    printf("Value of a is : %d",a);
    printf("\nValue of A is :%d",A);
    return 0;
}
```

- Output
- Value of a is : 10
- Value of A is :20
- The above output shows that the values of both the variables, 'a' and 'A' are different. Therefore, we conclude that the identifiers are case sensitive.

C Operators

- An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise, etc.

- There are following types of operators to perform different types of operations in C language.
- Arithmetic Operators
- Relational Operators
- Shift Operators
- Logical Operators

- Bitwise Operators
- Ternary or Conditional Operators
- Assignment Operator
- Misc Operator

Comments in C

- Comments in C language are used to provide information about lines of code.
- It is widely used for documenting code. There are 2 types of comments in the C language.
- Single Line Comments
- Multi-Line Comments

- Single Line Comments
- Single line comments are represented by double slash \\. Let's see an example of a single line comment in C.

```
#include<stdio.h>
int main(){
    //printing information
    printf("Hello C");
return 0;
}
```

- **Mult Line Comments**
- Multi-Line comments are represented by slash asterisk * ... *\. It can occupy many lines of code, but it can't be nested. Syntax:

/*

code

to be commented

*/

- Let's see an example of a multi-Line comment in C.

```
#include<stdio.h>
int main(){
   /*printing information
      Multi-Line Comment*/
    printf("Hello C");
return 0;
}
```

- **C Format Specifier**
- The Format specifier is a string used in the formatted input and output functions. The format string determines the format of the input and output. The format string always starts with a '%' character.
- The commonly used format specifiers in printf() function are:

Format specifier	Description
%d or %i	It is used to print the signed integer value where signed integer means that the variable can hold both positive and negative values.
%u	It is used to print the unsigned integer value where the unsigned integer means that the variable can hold only positive value.
%o	It is used to print the octal unsigned integer where octal integer value always starts with a 0 value.
%x	It is used to print the hexadecimal unsigned integer where the hexadecimal integer value always starts with a 0x value. In this, alphabetical characters are printed in small letters such as a, b, c, etc.

%X	It is used to print the hexadecimal unsigned integer, but %X prints the alphabetical characters in uppercase such as A, B, C, etc.
%f	It is used for printing the decimal floating-point values. By default, it prints the 6 values after '.'.
%e/%E	It is used for scientific notation. It is also known as Mantissa or Exponent.
%g	It is used to print the decimal floating-point values, and it uses the fixed precision, i.e., the value after the decimal in input would be exactly the same as the value in the output.

%p	It is used to print the address in a hexadecimal form.
%c	It is used to print the unsigned character.
%s	It is used to print the strings.
%ld	It is used to print the long-signed integer value.

- **ASCII value in C**
- **What is ASCII code?**
- The full form of ASCII is the American Standard Code for Information Interchange.
- It is a character encoding scheme used for electronics communication. Each character or a special character is represented by some ASCII code, and each ascii code occupies 7 bits in memory.

- In C programming language, a character variable does not contain a character value itself rather the ascii value of the character variable.
- The ascii value represents the character variable in numbers, and each character variable is assigned with some number range from 0 to 127.
- For example, the ascii value of 'A' is 65.
- In the above example, we assign 'A' to the character variable whose ascii value is 65, so 65 will be stored in the character variable rather than 'A'.

```
#include <stdio.h>
int main()
{
  char ch;    // variable declaration
 printf("Enter a character");
scanf("%c",&ch);  // user input
printf("\n The ascii value of the ch variable is : %d", ch);
return 0;
}
```

- Now, we will create a program which will display the ascii value of all the characters.

```
#include <stdio.h>
int main()
{
 int k;   // variable declaration
 for(k=0;k<=255;k++)  // for loop from 0-255
 {
     printf("\nThe ascii value of %c is %d", k,k);
 }
return 0;
}
```

- **Constants in C**
- A constant is a value or variable that can't be changed in the program, for example: 10, 20, 'a', 3.4, "c programming" etc.
- There are different types of constants in C programming.

List of Constants in C

Constant	Example
Decimal Constant	10, 20, 450 etc.
Real or Floating-point Constant	10.3, 20.2, 450.6 etc.
Octal Constant	021, 033, 046 etc.
Hexadecimal Constant	0x2a, 0x7b, 0xaa etc.
Character Constant	'a', 'b', 'x' etc.
String Constant	"c", "c program", "c in javatpoint" etc.

- 2 ways to define constant in C
- There are two ways to define constant in C programming
- const keyword
- #define preprocessor
- 1) C const keyword
- The const keyword is used to define constant in C programming.

- const float PI=3.14;
- Now, the value of PI variable can't be changed.
- #include<stdio.h>

```
int main(){
const float PI=3.14;
Printf ("The value of PI is: %f", PI);
return 0;
}
```

- If you try to change the the value of PI, it will render compile time error.

```
#include<stdio.h>
int main(){
const float PI=3.14;
PI=4.5;
printf("The value of PI is: %f",PI);
return 0;
}
```

Tokens in C

Tokens in C is the most important element to be used in creating a program in C. We can define the token as the smallest individual element in C.

For `example, we cannot create a sentence without using words; similarly, we cannot create a program in C without using tokens in C.

Therefore, we can say that tokens in C is the building block or the basic component for creating a program in C language.

Classification of tokens in C

Tokens in C language can be divided into the following categories:

Keywords in C

Identifiers in C

Strings in C

Operators in C

Constant in C

Special Characters in C

- **C Boolean**
- In C, Boolean is a data type that contains two types of values, i.e., 0 and 1. Basically, the bool type value represents two types of behavior, either true or false. Here, '0' represents false value, while '1' represents true value.
- In C Boolean, '0' is stored as 0, and another integer is stored as 1.
- We do not require to use any header file to use the Boolean data type in C++, but in C, we have to use the header file, i.e., stdbool.h. If we do not use the header file, then the program will not compile.
- Syntax
- bool variable_name;

- In the above syntax, bool is the data type of the variable, and variable_name is the name of the variable.

```
#include <stdio.h>
#include<stdbool.h>
int main()
{
bool x=false; // variable initialization.
```

```
if(x==true) // conditional statements
{
printf("The value of x is true");
}
else
printf("The value of x is FALSE");
return 0;
}
```

- **Static in C**
- Static is a keyword used in C programming language. It can be used with both variables and functions, i.e., we can declare a static variable and static function as well.
- An ordinary variable is limited to the scope in which it is defined, while the scope of the static variable is throughout the program.
- Static keyword can be used in the following situations:

- **Static global variable**
- When a global variable is declared with a static keyword, then it is known as a static global variable.
- It is declared at the top of the program, and its visibility is throughout the program.
- **Static function**
- When a function is declared with a static keyword known as a static function. Its lifetime is throughout the program.

- **Static local variable**
- When a local variable is declared with a static keyword, then it is known as a static local variable.
- The memory of a static local variable is valid throughout the program, but the scope of visibility of a variable is the same as the automatic local variables.
- However, when the function modifies the static local variable during the first function call, then this modified value will be available for the next function call also.

- **Static member variables**
- When the member variables are declared with a static keyword in a class, then it is known as static member variables.
- They can be accessed by all the instances of a class, not with a specific instance.
- **Static method**
- The member function of a class declared with a static keyword is known as a static method.
- It is accessible by all the instances of a class, not with a specific instance.

- **Properties of a static variable**
- The following are the properties of a static variable:
- The memory of a static variable is allocated within a static variable.
- Its memory is available throughout the program, but the scope will remain the same as the automatic local variables. Its
- value will persist across the various function calls.
- If we do not assign any value to the variable, then the default value will be 0.
- A global static variable cannot be accessed outside the program, while a global variable can be accessed by other source files.

- **Programming Errors in C**
- Errors are the problems or the faults that occur in the program, which makes the behavior 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:
- If we want to declare the variable of type integer,
- int a; // this is the correct form
- 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.

- **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.

- **Linker error**
- Linker errors are mainly generated when the executable file of the program is not created.
- This can be happened either due to the wrong function prototyping or usage of the wrong header file.
- For example, the main.c file contains the sub() function whose declaration and definition is done in some other file such as func.c. During the compilation, the compiler finds the sub() function in func.c file, so it generates two object files, i.e., main.o and func.o. At the execution time, if the definition of sub() function is not found in the func.o file, then the linker error will be thrown. The most common linker error that occurs is that we use Main() instead of main().

- **Logical error**
- The logical error is an error that leads to an undesired output. These errors produce the incorrect output, but they are error-free, known as logical errors.
- These types of mistakes are mainly done by beginners. The occurrence of these errors mainly depends upon the logical thinking of the developer.
- If the programmers sound logically good, then there will be fewer chances of these errors.

- **Semantic error**
- Semantic errors are the errors that occurred when the statements are not understandable by the compiler.
- The following can be the cases for the semantic error:
- Use of a un-initialized variable.
- int i;
- i=i+2;
- Type compatibility
- int b = "javatpoint";

- Errors in expressions
- int a, b, c;
- a+b = c;
- Array index out of bound
- int a[10];
- a[10] = 34;