Programming in C
(Keyword , Identifier , Operator)

Prepared By
Alok Haldar
Assistant professor
Department of Computer Science & BCA
Kharagpur College

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.

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

Precedence of Operators in C :

The precedence of operator species that which operator will be evaluated first and next. The associativity specifies the operator direction to be evaluated; it may be left to right or right to left.

Let's understand the precedence by the example given below:

```
int value=10+20*10;
```

The value variable will contain **210** because * (multiplicative operator) is evaluated before + (additive operator).

The precedence and associativity of C operators is given below:

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right

Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

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.

1. Single Line Comments
2. 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;
}
```

Output:

```
Hello C
```

Even you can place the comment after the statement. For example:

```
1. printf("Hello C");//printing information
```

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:

```
1. /*
2. code
3. to be commented
4. */
```

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

```
1. #include<stdio.h>
2. int main(){
3. /*printing information
4. Multi-Line Comment*/
5. printf("Hello C");
6. return 0;
7. }
```

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.

```
int main()
{
int b=6;
int c=8;
printf("Value of b is:%d",b);
printf("\nValue of c is:%d",c);
return 0;
}
```

In the above code, we are printing the integer value of b and c by using the %d specifier.

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'.

Let us understand through an example.

We will create a program which will display the ascii value of the character variable.

```
#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;
}
```

In the above code, the first user will give the character input, and the input will get stored in the 'ch' variable. If we print the value of the 'ch' variable by using %c format specifier, then it will display 'A' because we have given the character input as 'A', and if we use the %d format specifier then its ascii value will be displayed, i.e., 65.

Output

The above output shows that the user gave the input as 'A', and after giving input, the ascii value of 'A' will get printed, i.e., 65.

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(int k=0;k<=255;k++)//for loop from 0-255
{
printf("\nThe ascii value of %c is %d", k,k);
}
return 0;
}
```

The above program will display the ascii value of all the characters. As we know that ascii value of all the characters starts from 0 and ends at 255, so we iterate the for loop from 0 to 255.

Now we will create the program which will sum the ascii value of a string.

```
#include<stdio.h>
int main()
{
 int sum=0;//variable initialization
char name[20];//variable initialization
int i=0;//variable initialization
printf("Enter a name: ");
scanf("%s",name);
while(name[i]!='\0')//while loop
{
printf("\nThe ascii value of the character %c is %d",
name[i],name[i]);
sum=sum+name[i];
i++;
}
printf("\nSum of the ascii value of a stringis:%d",sum);
return0;
}
```

In the above code, we are taking user input as a string. After taking user input, we execute the **while** loop which adds the ascii value of all the characters of a string and stores it in a '**sum**' variable.

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.

1. const keyword
2. #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;
}
```

Output:

```
The value of PI is: 3.140000
```

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;
}
```

Output:

```
Compile Time Error: Cannot modify a const object
```

2) C #define preprocessor

The #define preprocessor is also used to define constant. We will learn about #define preprocessor directive later.

Visit here for:#define preprocessor directive.

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 aprogram in C language.

Classification of tokens in C

Tokens in C language can be divided into the five parts.

- Identifiers in C
- Strings in C
- Operators in C
- Constant in C
- Special Characters in C

Let us understand each token one by one.

Keywords in C

Keywords in C can be defined as the **pre-defined** or the **reserved words** having its own importance, and each keyword has its own functionality. Since keywords are the pre-defined words used by the compiler, so they cannot be used as the variable names. If the keywords are used as the variable names, it means that we are assigning a different meaning to the keyword, which is not allowed. C language supports 32 keywords 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**

Identifiers in C

Identifiers in C are used for naming variables, functions, arrays, structures, etc. Identifiers in C are the user-defined words. It can be composed of uppercase letters, lowercase letters, underscore, or digits, but the starting letter should be either an underscore or an alphabet. Identifiers cannot be used as keywords. Rules for constructing identifiers in C are given below:

- 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.

Strings in C

Strings in C are always represented as an array of characters having null character '\0' at the end of the string. This null character denotes the end of the string. Strings in C are enclosed within double quotes, while characters are enclosed within single characters. The size of a string is a number of characters that the string contains.

Now, we describe the strings in different ways:

char a[10] = "javatpoint"; // The compiler allocates the 10 bytes to the 'a' array.

char a[] = "javatpoint"; // The compiler allocates the memory at the run time.

char a[10] = {'j','a','v','a','t','p','o','i','n','t','\0'}; // String is represented in the form of characters.

Operators in C

Operators in C is a special symbol used to perform the functions. The data items on which the operators are applied are known as operands. Operators are applied between the operands. Depending on the number of operands, operators are classified as follows:

Unary Operator

A unary operator is an operator applied to the single operand. For example: increment operator (++), decrement operator (--), sizeof, (type)*.

Binary Operator

The binary operator is an operator applied between two operands. The following is the list of the binary operators:

- Arithmetic Operators
- Relational Operators
- Shift Operators
- Logical Operators
- Bitwise Operators
- Conditional Operators
- Assignment Operator
- Misc Operator

Constants in C

A constant is a value assigned to the variable which will remain the same throughout the program, i.e., the constant value cannot be changed.

There are two ways of declaring constant:

- Using const keyword
- Using #define pre-processor

Types of constants in C

Constant	Example
Integer constant	**10, 11, 34, etc.**
Floating-point constant	**45.6, 67.8, 11.2, etc.**
Octal constant	**011, 088, 022, etc.**
Hexadecimal constant	**0x1a, 0x4b, 0x6b, etc.**
Character constant	**'a', 'b', 'c', etc.**
String constant	**"java", "c++", ".net", etc.**

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.