

## C Constants and Literals

Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.

Constants can be of any of the basic data types like *an integer constant, a floating constant, a character constant, or a string literal*. There are enumeration constants as well.

Constants are treated just like regular variables except that their values cannot be modified after their definition.

### Integer Literals

An integer literal can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.

An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

Here are some examples of integer literals –

```
212          /* Legal */
215u         /* Legal */
0xFeeL      /* Legal */
078         /* Illegal: 8 is not an octal digit */
032UU       /* Illegal: cannot repeat a suffix */
```

Following are other examples of various types of integer literals –

```
85          /* decimal */
0213        /* octal */
0x4b        /* hexadecimal */
30          /* int */
30u         /* unsigned int */
30l         /* long */
30ul        /* unsigned long */
```

### Floating-point Literals

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.

While representing decimal form, you must include the decimal point, the exponent, or both; and while representing exponential form, you must include the integer part, the fractional part, or both. The signed exponent is introduced by e or E.

Here are some examples of floating-point literals –

```
3.14159     /* Legal */
314159E-5L  /* Legal */
510E        /* Illegal: incomplete exponent */
210f        /* Illegal: no decimal or exponent */
.e55        /* Illegal: missing integer or fraction */
```

### Character Constants

Character literals are enclosed in single quotes, e.g., 'x' can be stored in a simple variable of **char** type.

A character literal can be a plain character (e.g., 'x'), an escape sequence (e.g., '\t'), or a universal character (e.g., '\u02C0').

There are certain characters in C that represent special meaning when preceded by a backslash for example, newline (\n) or tab (\t).

**escape sequence codes –**

Escape sequence	Meaning
\\	\ character
\'	' character
\"	" character
\?	? character
\a	Alert or bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\ooo	Octal number of one to three digits
\xhh . . .	Hexadecimal number of one or more digits

Following is the example to show a few escape sequence characters –

```
#include <stdio.h>
```

```
int main() {
    printf("Hello\tWorld\n\n");

    return 0;
}
```

When the above code is compiled and executed, it produces the following result –

Hello World

## String Literals

String literals or constants are enclosed in double quotes "". A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.

You can break a long line into multiple lines using string literals and separating them using white spaces.

Here are some examples of string literals. All the three forms are identical strings.

```
"hello, dear"
```

```
"hello, \
```

```
dear"
```

```
"hello, " "d" "ear"
```

## Defining Constants

There are two simple ways in C to define constants –

- Using **#define** preprocessor.
- Using **const** keyword.

## The #define Preprocessor

Given below is the form to use #define preprocessor to define a constant –

```
#define identifier value
```

The following example explains it in detail –

```
#include <stdio.h>

#define LENGTH 10
#define WIDTH 5
#define NEWLINE '\n'

int main() {
    int area;

    area = LENGTH * WIDTH;
    printf("value of area : %d", area);
    printf("%c", NEWLINE);

    return 0;
}
```

When the above code is compiled and executed, it produces the following result –

value of area : 50

## The const Keyword

You can use **const** prefix to declare constants with a specific type as follows –

```
const type variable = value;
```

The following example explains it in detail –

```
#include <stdio.h>

int main() {
    const int    LENGTH = 10;
    const int    WIDTH  = 5;
    const char   NEWLINE = '\n';
    int area;

    area = LENGTH * WIDTH;
    printf("value of area : %d", area);
    printf("%c", NEWLINE);

    return 0;
}
```

When the above code is compiled and executed, it produces the following result –

value of area : 50

Note that it is a good programming practice to define constants in CAPITALS.