# Unit-2

# Elements of C

## Execution Characters/Escape Sequence

Characters are printed on the screen through the keyboard but some characters such as newline, tab and backspace cannot be printed like other normal characters. C supports the combination of backslash (\) and some characters from the C character set to print these characters.

These character combinations are known as escape sequence and represented by two characters. The first character is "\" and second character is from the C character set. Some escape sequences are given below:

| Escape Sequence | Meaning | ASCII Value | Purpose |
|---|---|---|---|
| \b | backspace | 008 | Moves the cursor the previous position of the current line. |
| \a | bell (alert) | 007 | Produces a beep sound for alert |
| \r | carriage return | 013 | Moves the cursor to beginning of the current line |
| \n | newline | 010 | Moves the cursor to the beginning of the next line |
| \f | form feed | 012 | Moves the cursor to the initial position to the next logical page |
| \0 | null | 000 | Null |
| \v | vertical tab | 011 | Moves the cursor to the next vertical tab position |
| \t | horizontal tab | 009 | Moves the cursor to the next horizontal tab position |
| \\ | backslash | 092 | Presents a character with backslash (\) |

## Delimiters

Delimiters are used for syntactic meaning in C. There are given below:

| | | |
|---|---|---|
| : | Colon | used for label |
| ; | semi-colon | end of statement |

| | | |
|---|---|---|
| ( ) | Parenthesis | used in expressions |
| { } | Curly Brackets | used for block of statement |
| [ ] | Square Brackets | used for array |
| # | Hash | preprocessor directives |
| , | Comma | variable delimiter |

## Reserved Keywords

There are certain words that are reserved for doing specific tasks. These words are known as keywords and they have standard, predefined meaning in C. They are always written in lowercase. There are only 32 keywords available in C which are given below:

| | | | |
|---|---|---|---|
| auto | break | case | char |
| const | continue | default | do |
| double | else | enum | extern |
| float | for | goto | if |
| int | long | register | return |
| short | signed | sizeof | static |
| struct | switch | typedef | union |
| unsigned | void | volatile | while |

## Identifiers

All the words that we will use in our C program will be either keywords or identifiers. Keywords are predefined and cannot be changed by the user while identifiers are user defined words and are used to give names to entities like variables, arrays, functions, structures, etc. Rules for naming identifiers are given below:

1. The name should consist of only alphabet (both upper and lower case), digits and underscore sign (_).
2. The first character should be an alphabet or underscore.
3. The name should not be a keyword.
4. we should not use space between the identifier.
5. Since C is case sensitive, the uppercase and lowercase letter are considered different for eg. Code, code and CODE are three different identifiers.

6. An identifier name may be arbitrarily long. Some implementations of C recognize only the first 8 character, though most implementations recognize 31 characters. ANSI standard compilers recognize 31 characters.

The identifiers are generally given meaningful names. Some examples of valid identifiers name-

*salary a total_marks data2 _college NAME*

Some example of invalid identifiers name are-

8ab First character should be an alphabet or underscore
char char is a keyword
sopan # # is a special character
avg no blank space is not permitted.

Each and every alphabet and punctuation symbols are called tokens.

## Variables:

Variable is a name that can be used to store values. Variables can take different values but one at a time. These values can be changed during the execution of the program. A data type is associated with each variable. The data type of the variable decides what values it can take. The rules for naming variables are same as that of naming identifiers.

### Declaration of Variables:

It is necessary to declare a variable before it is used in the program. Declaration of a variable specifies its name and data type. The type and range of values that a variable can store depends upon its data type. The syntax of declaration of a variable is-

datatype variable name;

Here data type may be int, float, char, double etc. Some examples of declaration of variables are:

int x;
float salary;
char grade;

Here x is a variable of type int, salary is a variable of type float, and grade is a variable of type char. We can also declare more than one variable in a single declaration. For example:

int x, y, z, total;

Here x, y, z and total are all variables of type int.

### Initialization of Variables:

When a variable is declared it contains undefined value commonly known as garbage value. If we want we can assign some initial value to the variable during the declaration itself, this is called initialization of variables. For example:

int a = 5;

```
float x = 8.9, y = 10.5;
char ch = 'y';
double num = 0.15197e-7;
int l, m, n total = 0;
```

In the last declaration only variable total has been initialized.

## Datatypes

C supports different types of data. Storage representation of these data types is different in memory. There are four fundamental data types in C, which are int, char, float and double.

'char' is used to store and single character. 'int' is used to store integer value . 'float' is used for storing single precision floating point number and 'double' is used for storing double precision floating point number. We can use type qualifiers with these basic types to get some more types:

There are two types of type qualifiers:

i. Size qualifiers Short, long

ii. Sign qualifiers Signed, Unsigned

When the qualifiers unsigned is used the number is always positive and when signed is used number may be positive or negative. If the sign qualifier is not mentioned, then by default signed qualifier is assumed. The range of values for signed data types is less than that of unsigned types. This is because in signed type, the leftmost bit is used to represent the sign, while in unsigned type this bit is also used to represent the value.

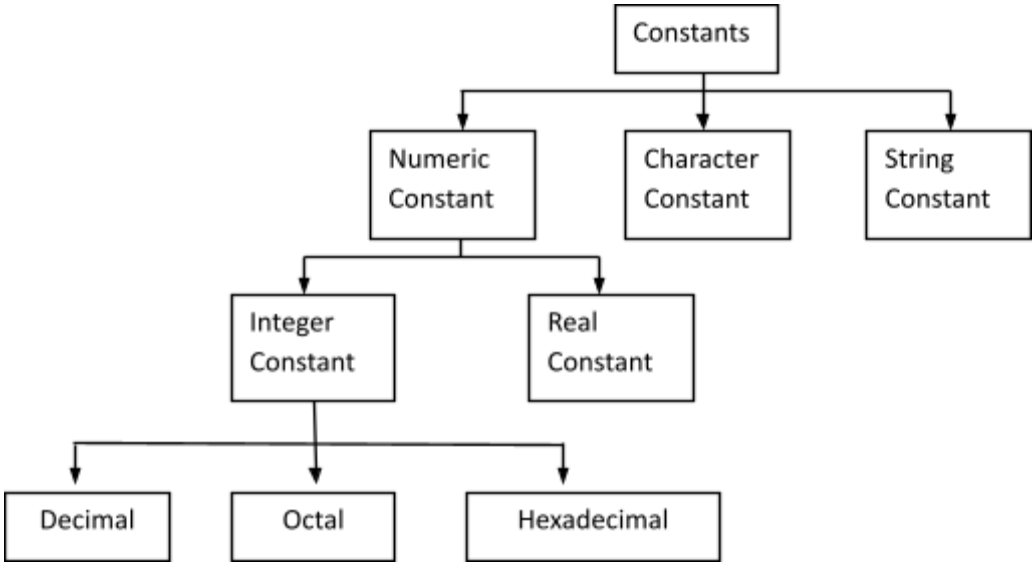
The size and range of different data types on a 16-bit machine is given in the following table. The size and range may vary on machines with different word size.

| Basic data types | Data types with type qualifiers | Size (bytes) | Range |
|---|---|---|---|
| char | char or signed char | 1 | -128 to 127 |
| | unsigned char | 1 | 0 to 255 |
| int | int or signed int | 2 | -32768 to 32767 |
| | unsigned int | 2 | 0 to 65535 |
| | short int or signed short int | 1 | -128 to 127 |
| | unsigned short int | 1 | 0 to 255 |
| | long int or signed long int | 4 | -2147483648 to 2147483647 |
| | unsigned long int | 4 | 0 to 4294967295 |

| | | | |
|---|---|---|---|
| float | float | 4 | 3.4E-38 to 3.4E+38 |
| double | double | 8 | 1.7E -308 to 1.7E+308 |
| | long double | 10 | 3.4E-4932 to 1.1E+4392 |

## Constants

Constant is a value that cannot be changed during execution of the program. There are three types of constant.




### Numeric Constants

Numeric constant consist of numeric digits, they may or may not have decimal point. (.). These are the rules for defining numeric constants-

i. Numeric constant should have at least on digit.
ii. No comma or space is allowed within the numeric constant.
iii. Numeric constant can either be positive or negative but default sign is always positive.

There are two types of numeric constant-

### Integer Constant:

Integer constant are whole number which have no decimal point (.). There are three types of integer constants based on different number systems. The permissible characters that can be used in these constant are-

| | | |
|---|---|---|
| Decimal Constants | 0,1,2,3,4,5,6,7,8,9 | (Base = 10) |
| Octal Constants | 0,1,2,3,4,5,6,7 | (Base = 8) |

| | | |
|---|---|---|
| Hexadecimal constants | 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E, F | (Base = 16) |

Some valid integer decimal constant are-

0
123
3705
23759

Some invalid decimal integer constant are-

| Invalid | Remarks |
|---|---|
| 2.5 | illegal character (.) |
| 3#5 | illegal character (#) |
| 98 5 | No blank space allowed |
| 0925 | first digit can't be zero |
| 8,354 | comma is not allowed |

In octal integer constant, first digit must be 0. For example-

0, 05, 077, 0324

In hexadecimal integer constant, first two characters should be 0x or 0X. Some examples are as-

0X, 0X23, 0X515, 0XFFF, 0XAC

By default the type of integer constant is int. But if the value of integer constant exceeds the range of values represented by int type, the type is taken to be unsigned int or long int. We can also explicitly mention the type of the constant by suffixing it with l or L (for long), u or U (for unsigned), ul or UL (for unsigned long). For example:

| | |
|---|---|
| 6453 | integer constant of type int |
| 45238722UL or 45238722ul | integer constant of type unsigned long |
| 6655u or 6655U | integer constant of type unsigned |

### Real (floating point) constants:

Floating point constant are numeric constants that contain decimal point. Some valid floating point constant are:

0.5
5.3
4000.0
0.0075
76847.
98.0973

For expressing very large or very small real constants, exponential (scientific) form is used. Here the number is written in the mantissa and exponent form, which are separated by 'e' or 'E'. The mantissa can be an integer or a real number, while the exponent can be only an integer (positive or negative). For example the number 1800000 can be written as 1.8e6, here 1.8 is mantissa and 6 is the exponent. Some more examples are as:

| **Number** | | **Exponential form** |
|---|---|---|
| 250000000 | $2.5 \times 10^{8}$ | 2.5e8 |
| 0.0000076 | $7.6 \times 10^{-6}$ | 7.6e-6 |
| -670000 | $-6.7 \times 10^{5}$ | -6.7e5 |

By default the type of a floating point constant is double. We can explicitly mention the type of constant by suffixing it with a f or F (for float type), l or L (for long double). For example-

2.3e5 Token → floating point constant of type double,

2.4e-9 l or 2.4e-9L → floating point constant of type long double,

3.52f or 3.52F → floating point constant of type float.

### Character Constant:

The character constant is a single character that is enclosed within single quotes. Some valid character constants are:

'9' 'D' '$' ' ' '#'

Some invalid character constants are:

| Invalid | Remarks |
|---|---|
| 'four' | There should be only one character within quotes. |
| "d" | Double quotes are not allowed |
| ' ' | No character between single quotes. |
| y | single quotes missing |

Every character constant has a unique integer value associated with it. This integer is the numeric value of the character in the machine's character code. If the machine is using ASCII (American Standard Code for Information Interchange), then the character 'G' represents integer value 71 and the character '5' represents value 53. Some ASCII values are:

A - Z ASCII value (65 - 90)

a - z ASCII value (97-122)

0 - 9 ASCII value (48-57)

; ASCII value (59)

ASCII values of all characters are given in Appendix A.

### String Constants:

A string constants has zero, one or more than one character. A string constant is enclosed within double quotes (" "). At the end of string, \0 is automatically placed by the compiler.

Some examples of string constants are:

"Kumar"
"593"
"8"
" "
"A"

Note that "A" and 'A' are different, the first one is a string constant which consist of character A and \0 while the second one is a character constant which represents integer value 65.

### Symbolic Constants:

If we want to use a constant several times then we can provide it a name. For example if we have to use the constant 3.14159265 at many places in our program, then we can give it a name PI and use this name instead of writing the constant value everywhere. These types of constants are called symbolic constants or named constants.

A symbolic constant is a name that substitutes for a sequence of characters. The characters may represent a numeric constant, a character constant or a string constant.

These constant are generally defined at the beginning of the program as:

#define name value

Here 'name' is the symbolic name for the constant, and is generally written in uppercase letters. 'value' can be numeric, character or string constant.

Some examples of symbolic constants are as-

# define MAX 100
#define PI 3.14159625
#define CH 'y'
#define NAME "Suresh"

In the program, these names will be replaced by the corresponding values. These symbolic constants improve the readability and modifiability of the program.

## Statements:

In C program, instructions are written in the form of statements. A statements is an executable part of the program and causes the computer to carry out some action. Statements can be categorized as:

i. Expression statements
ii. Compound statements
iii. Selection statements (if, if…..else, switch)
iv. Iterative statements (for, while, do….while)

v. Jump statements (goto, continue, break, return)

vi. Label statements (case, default, label statement used in goto)

### Simple Statement:

Simple statement consist of an expression followed by a semi colon. For example:

```
x = 5;
x = y - z;
func( a, b);
```

A statement that has only a semicolon is also known as null statement. For example:

```
;  /*null statement*/
```

### Compound Statement:

A compound statement consists of several statements enclosed within a pair of curly braces { }. Compound statement is also known as block of statements. Note that there is no semicolon after the closing brace. for example:

```
{
  int l = 4, b = 2, h = 3;
  int area, volume;
  area = 2 * (l*b + b *h + l * h);
  volume = l * b * h;
}
```

If any variable is to be declared inside the block then it can be declared only at the beginning of the block. The variables that are declared inside a block can be used only inside that block; all other categories of statements are discussed in further chapters.

## Comments:

Any text or symbol /* to */ is ignored by the compiler is called comment. Comments are not tokens and are not part of executable program. Comments are used for increasing readability of the program. They explain the purpose of the program and are helpful in understanding the program. Comments are written inside /* and */. There can be single line or multiple line comments. We can write comments anywhere in the program except inside a string constant or a character constant.

**Some examples of comments are:**

```
// basic salary */   (Single line comment)
/* this is a C program to calculate                    (multiple line comment)
   Simple interest */
```

Comments can't be nested i.e. we can't write a comment inside another comment.