# Unit 2 : Basic Element of C

Er. Nipun Thapa

# 2.1. C Character Set

- As every language contains a set of characters used to construct words, statements, etc.
- C language also has a set of characters which include **alphabets, digits**, and **special symbols**.
- C language supports a total of 256 characters.

# 2.1. C Character Set

- Every C program contains statements.
- These statements are constructed using words and these words are constructed using characters from C character set.
- C language character set contains the following set of characters...
  1. Alphabets
  2. Digits
  3. Special Symbols

# 2.1. C Character Set

**Alphabets**

- C language supports all the alphabets from the English language. Lower and upper case letters together support 52 alphabets.
  - lower case letters - **a to z**
  - UPPER CASE LETTERS - **A to Z**

**Digits**

- C language supports 10 digits which are used to construct numerical values in C language.
  - Digits - **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**

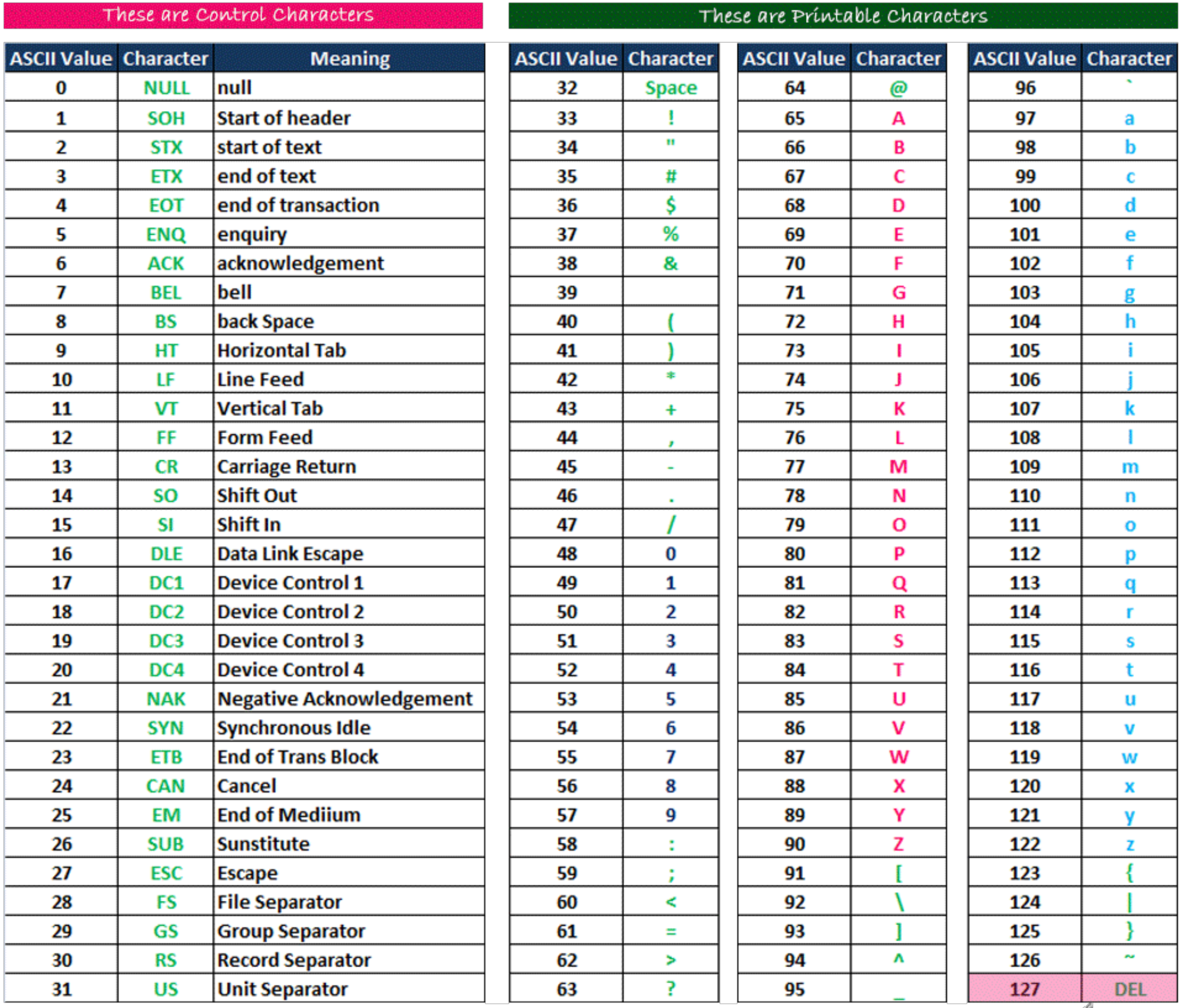
# 2.1. C Character Set

## Special Symbols

- C language supports a rich set of special symbols that include symbols to perform mathematical operations, to check conditions, white spaces, backspaces, and other special symbols.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| + | - | * | / | = | % | & | # |
| ! | ? | ^ | “ | ‘ | ~ | \ | \| |
| < | > | ( | ) | [ | ] | { | } |
| : | ; | . | , | _ | @ | $ | (blank space) |

# Commonly used characters in C with thier ASCII values



These are Control Characters

These are Printable Characters

| ASCII Value | Character | Meaning | ASCII Value | Character | ASCII Value | Character | ASCII Value | Character |
|---|---|---|---|---|---|---|---|---|
| 0 | NULL | null | 32 | Space | 64 | @ | 96 | ` |
| 1 | SOH | Start of header | 33 | ! | 65 | A | 97 | a |
| 2 | STX | start of text | 34 | " | 66 | B | 98 | b |
| 3 | ETX | end of text | 35 | # | 67 | C | 99 | c |
| 4 | EOT | end of transaction | 36 | $ | 68 | D | 100 | d |
| 5 | ENQ | enquiry | 37 | % | 69 | E | 101 | e |
| 6 | ACK | acknowledgement | 38 | & | 70 | F | 102 | f |
| 7 | BEL | bell | 39 |  | 71 | G | 103 | g |
| 8 | BS | back Space | 40 | ( | 72 | H | 104 | h |
| 9 | HT | Horizontal Tab | 41 | ) | 73 | I | 105 | i |
| 10 | LF | Line Feed | 42 | * | 74 | J | 106 | j |
| 11 | VT | Vertical Tab | 43 | + | 75 | K | 107 | k |
| 12 | FF | Form Feed | 44 | , | 76 | L | 108 | l |
| 13 | CR | Carriage Return | 45 | - | 77 | M | 109 | m |
| 14 | SO | Shift Out | 46 | . | 78 | N | 110 | n |
| 15 | SI | Shift In | 47 | / | 79 | O | 111 | o |
| 16 | DLE | Data Link Escape | 48 | 0 | 80 | P | 112 | p |
| 17 | DC1 | Device Control 1 | 49 | 1 | 81 | Q | 113 | q |
| 18 | DC2 | Device Control 2 | 50 | 2 | 82 | R | 114 | r |
| 19 | DC3 | Device Control 3 | 51 | 3 | 83 | S | 115 | s |
| 20 | DC4 | Device Control 4 | 52 | 4 | 84 | T | 116 | t |
| 21 | NAK | Negative Acknowledgement | 53 | 5 | 85 | U | 117 | u |
| 22 | SYN | Synchronous Idle | 54 | 6 | 86 | V | 118 | v |
| 23 | ETB | End of Trans Block | 55 | 7 | 87 | W | 119 | w |
| 24 | CAN | Cancel | 56 | 8 | 88 | X | 120 | x |
| 25 | EM | End of Mediium | 57 | 9 | 89 | Y | 121 | y |
| 26 | SUB | Sunstitute | 58 | : | 90 | Z | 122 | z |
| 27 | ESC | Escape | 59 | ; | 91 | [ | 123 | { |
| 28 | FS | File Separator | 60 | < | 92 | \ | 124 | \| |
| 29 | GS | Group Separator | 61 | = | 93 | ] | 125 | } |
| 30 | RS | Record Separator | 62 | > | 94 | ^ | 126 | ~ |
| 31 | US | Unit Separator | 63 | ? | 95 | _ | 127 | DEL |

DEL is also Control Characters

# 2.2. C Data Types

- The data-type in a programming language is the collection of data with values having fixed meaning as well as characteristics.
- Some of them are an integer, floating point, character, etc.
- Usually, programming languages specify the range values for given data-type.

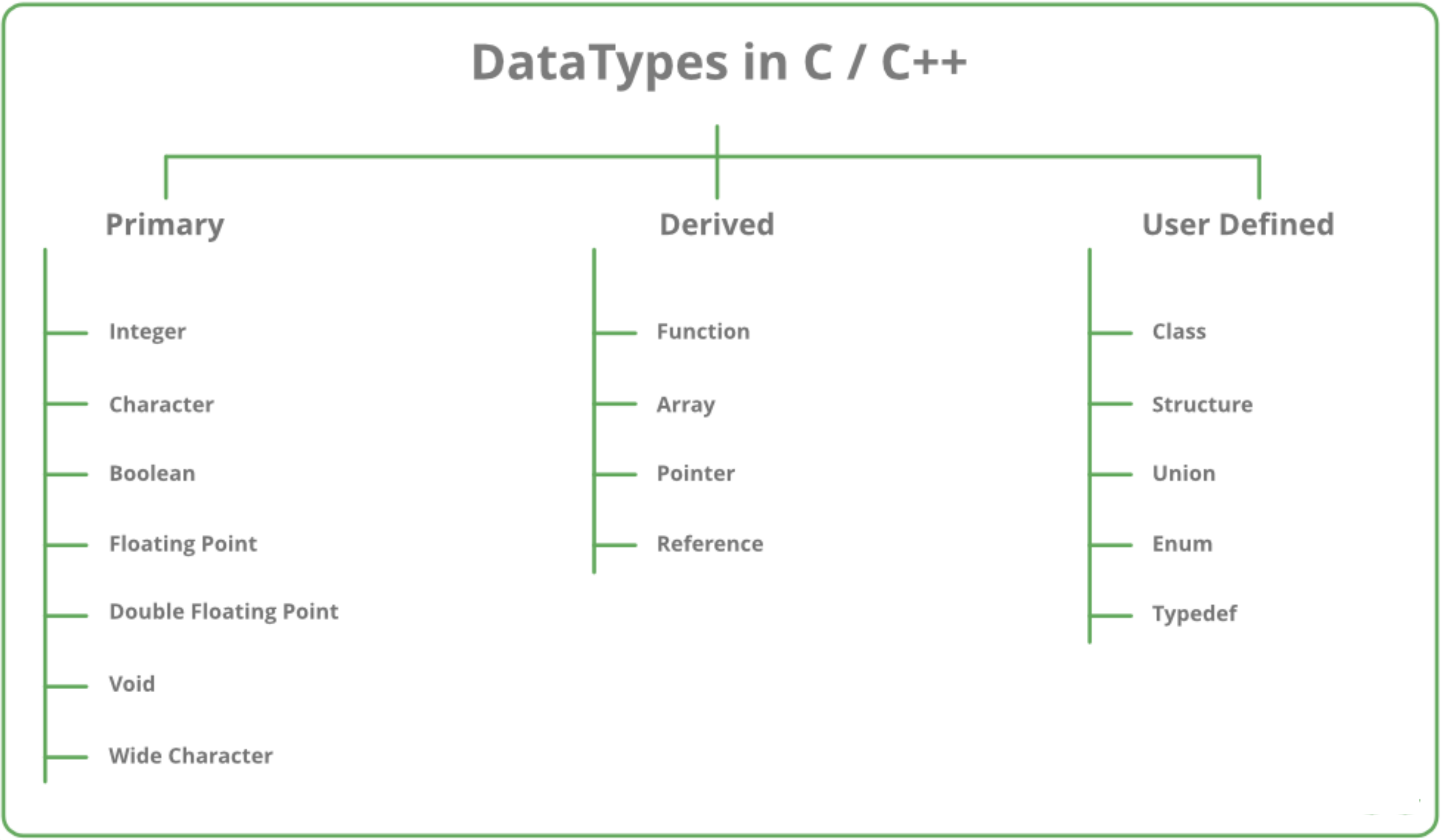
**C Data Types are used to:**

- Identify the type of a variable when it declared.
- Identify the type of the return value of a function.
- Identify the type of a parameter expected by a function.

**ANSI C provides three types of data types :**

- Primary data types : *void*, *int*, *char*, *double* and *float*.
- Derived data types : *Array*, *References*, and *Pointers*
- User defined data types : *Structure*, *Union*, and *Enumeration*

# 2.2. C Data Types



DataTypes in C / C++

Primary

- Integer
- Character
- Boolean
- Floating Point
- Double Floating Point
- Void
- Wide Character

Derived

- Function
- Array
- Pointer
- Reference

User Defined

- Class
- Structure
- Union
- Enum
- Typedef

# 2.2. C Data Types

**1.Primary Data Type**

(a) **Integer Data Types**

Integers are whole numbers that can have both zero, positive and negative values but no decimal values. i.e. non-fractional numbers. For example, 0, -5, 10

In C, integers are divided into three classes

i. Integer ( **int )**

ii. Short integer (**short int**)

iii. Long integer (**long int)**

**in both <u>signed</u> and <u>unsigned</u> form**

# 2.2. C Data Types

**1.Primary Data Type**

**Signed Integer**

-By  default, all integers are signed implicity

-it represent both positive and negative integers.

-The data type qualifier is **signed int** or **int**

-eg : int a; int b

**Unsigned Integer**

-It represent only positive integer

-It is **unsigned int** or **unsigned**.

# 2.2. C Data Types

**1.Primary Data Type**

**Signed Short Integer**

-By default all short integer are signed.

-it represent both positive and negative integers.

-The data type qualifier is **signed short int** or **short int** or **short.**

**Unsigned Short Integer**

-It represent only positive integer

-It is **unsigned Short int** or **unsigned Short**.

# 2.2. C Data Types

**1.Primary Data Type**

**Signed Long Integer**

-By default all short integer are signed.

-it represent both positive and negative integers.

-The data type qualifier is **signed short int** or **short int** or **short.**

**Unsigned Long Integer**

-It represent only positive integer

-It is **unsigned Long int** or **unsigned Long**.

# 2.2. C Data Types

**(b) Floating Point:**

- Float and double are used to hold real numbers.
- In C, floating-point numbers can also be represented in exponential. For example, float n = 22.442e2;

What's the difference between float and double?

- The size of float (single precision float data type) is 4 bytes. And the size of double (double precision float data type) is 8 bytes.

# 2.2. C Data Types

## (C) Character type

Character types are used to store characters value. A single alphabet can be treated as character data type and is defined between single quotation marks. Eg:‘r’,‘H’,‘5’, ‘*’, etc.

# 2.2. C Data Types

| Data Type | Size in bytes (bits) | Value in Range |
|---|---|---|
| Char, signed char | 1 byte (8-bits) | -127 to 127 |
| Unsigned char | 1 byte (8-bits) | 0 to 255 |
| Int, signed int | 2 byte (16-bits) | -32768 to 32768 |
| Unsigned int | 2 byte (16-bits) | 0 to 65535 |
| Short int, signed short int | 2 byte (16-bits) | -32768 to 32768 |
| Unsigned short int | 2 byte (16-bits) | 0 to 65535 |
| Long int, signed long int | 4 byte (32-bits) | -2147483648 to 2147483648 |
| Unsigned long int | 4 byte (32-bits) | 0 to 4294967296 |
| Float | 4 byte (32-bits) | 3.4E-38 to 3.4E+38 |
| Double | 8 byte (64-bits) | 1.7E-308 to 1.7E+308 |
| Long double | 10 byte (80-bits) | 3.4E-4932 to 1.1E+4932 |

# 2.2. C Data Types (Example)

**Program to Print an Integer**

```
#include <stdio.h>
int main()
{
    int number;
    printf("Enter an integer: "); // reads and stores input
    scanf("%d", &number); // displays output
    printf("You entered: %d", number);
    return 0;
 }
```

**Output**

```
Enter an integer: 25
You entered: 25
```

# 2.2. C Data TypesDeclaration

| Format specifier | Description | Supported data types |
|---|---|---|
| %c | Character | char<br>unsigned char |
| %d | Signed Integer | short<br>unsigned short<br>int<br>long |
| %e or %E | Scientific notation of float values | float<br>double |
| %f | Floating point | float |
| %g or %G | Similar as %e or %E | float<br>double |
| %hi | Signed Integer(Short) | short |
| %hu | Unsigned Integer(Short) | unsigned short |

# 2.2. C Data Types Declaration

| Format specifier | Description | Supported data types |
|---|---|---|
| %i | Signed Integer | short<br>unsigned short<br>int<br>long |
| %l or %ld or %li | Signed Integer | long |
| %lf | Floating point | double |
| %Lf | Floating point | long double |
| %lu | Unsigned integer | unsigned int<br>unsigned long |
| %lli, %lld | Signed Integer | long long |
| %llu | Unsigned Integer | unsigned long long |

# 2.2. C Data Types Declaration

| Format specifier | Description | Supported data types |
|---|---|---|
| %o | Octal representation of Integer. | short<br>unsigned short<br>int unsigned int long |
| %p | Address of pointer to void void * | void * |
| %s | String | char * |
| %u | Unsigned Integer | unsigned int<br>unsigned long |
| %x or %X | Hexadecimal representation of Unsigned Integer | short<br>unsigned short<br>int<br>unsigned int<br>long |
| %n | Prints nothing | |
| %% | Prints % character | |

# 2.2. C Data Types Declaration

- Integer :
  - **int n**; represented by : %d or %i
- Signed and unsigned :
  - **unsigned int x**; represented by : %u
  - **int y**; represented by : %d
  - **long a**; represented by : %li
- Float :
  - **float n**; represented by %f
  - **double n**; represented by %lf
  - **long double n**; represented by %Lf
- Character :
  - **char c** = ‘A’ ; represented by %c
  - **char s[10]**; represented by %s

# 2.3. Derived data types

- The data-types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types.
- These can be of four types namely:
  - Function
  - Array
  - Pointers
  - References

# 2.4. User-Defined Data Types

- The data types that are defined by the user are called the derived datatype or user-defined derived data type.
- These types include:
  - Class
  - Structure
  - Union
  - Enumeration
  - Typedef defined DataType

# 2.5. Constant and Variable

## Constant

- As the name suggests the name constants is given to such variables or values in C programming language which cannot be modified once they are defined.
- They are fixed values in a program.
- There can be any types of constants like integer, float, octal, hexadecimal, character constants etc.
- Every constant has some range.
- The integers that are too big to fit into an int will be taken as a long.
- Eg: #define val 10

# 2.5. Constant and Variable

**Variable**

- A **variable** in simple terms is a storage place which has some memory allocated to it.
- Basically, a variable used to store some form of data.
- Different types of variables require different amounts of memory and have some specific set of operations which can be applied to them.

# 2.6. The Size of Operator

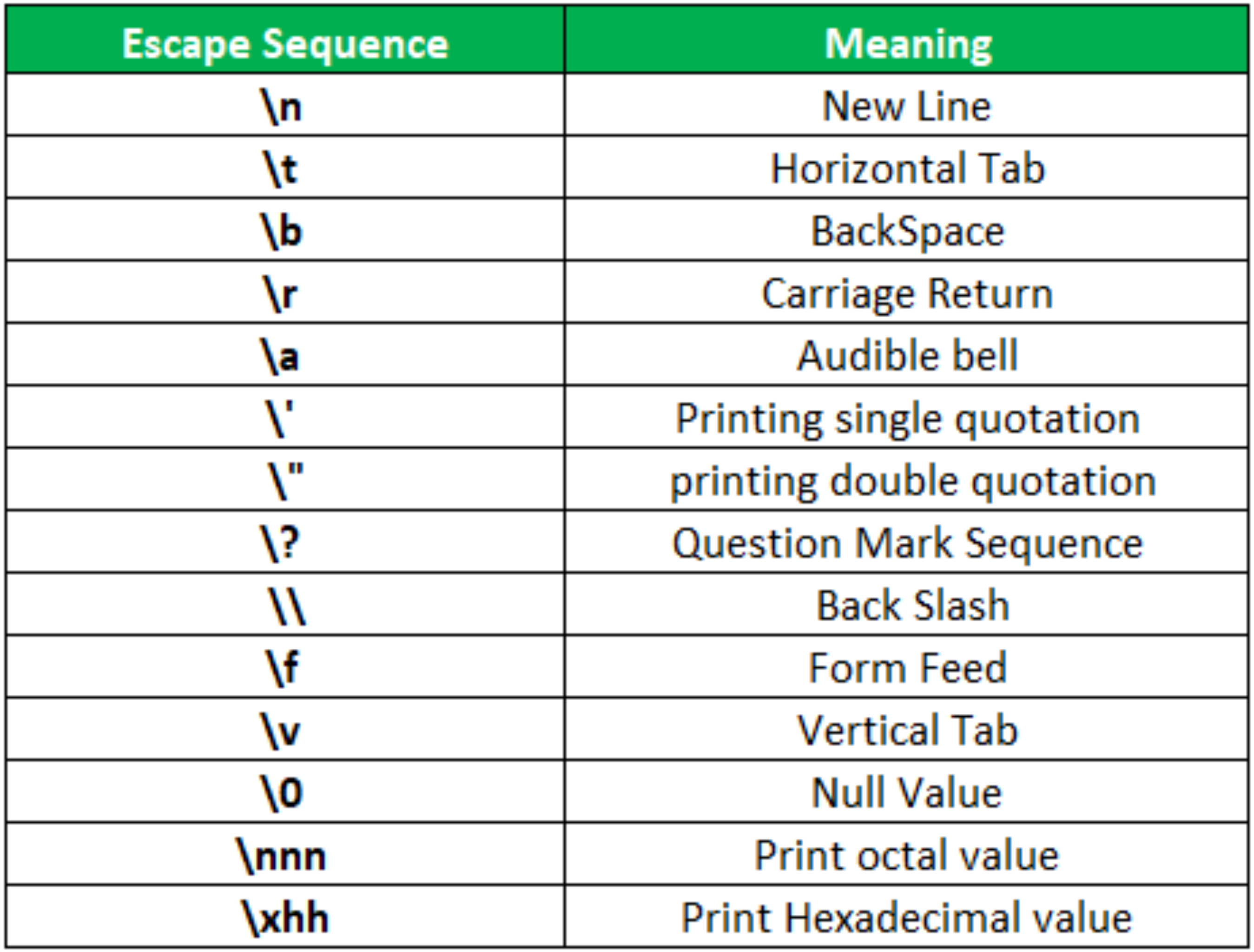
- ***Sizeof*** is a much used operator in the C .
- It is a compile time unary operator which can be used to compute the size of its operand.
- **Sizeof** can be applied to any data-type, including primitive types such as integer and floating-point types, pointer types, or compound datatypes such as Structure, union etc.
- ***sizeof***_()_ operator is used in different way according to the operand type.
- E.g. : #include <stdio.h>

```
     int main()
    {
       printf("%lu\n", sizeof(char));
       printf("%lu\n", sizeof(int));
       printf("%lu\n", sizeof(float));
       printf("%lu", sizeof(double));
       return 0;
   }
```

# 2.7. Escape Sequences

- In C programming language, there are 256 numbers of characters in character set.
- The entire character set is divided into 2 parts i.e. the ASCII characters set and the extended ASCII characters set.
- But apart from that, some other characters are also there which are not the part of any characters set, known as ESCAPE characters.

# 2.7. Escape Sequences



| Escape Sequence | Meaning |
|---|---|
| \n | New Line |
| \t | Horizontal Tab |
| \b | BackSpace |
| \r | Carriage Return |
| \a | Audible bell |
| \' | Printing single quotation |
| \" | printing double quotation |
| \? | Question Mark Sequence |
| \\ | Back Slash |
| \f | Form Feed |
| \v | Vertical Tab |
| \0 | Null Value |
| \nnn | Print octal value |
| \xhh | Print Hexadecimal value |

# 2.7. Escape Sequences (example)

```
// C program to illustrate
// \a escape sequence
#include <stdio.h>
int main(void)
{
    printf("My mobile number "
           "is 7\a8\a7\a3\a9\a2\a3\a4\a0\a8\a");
    return (0);
}
```

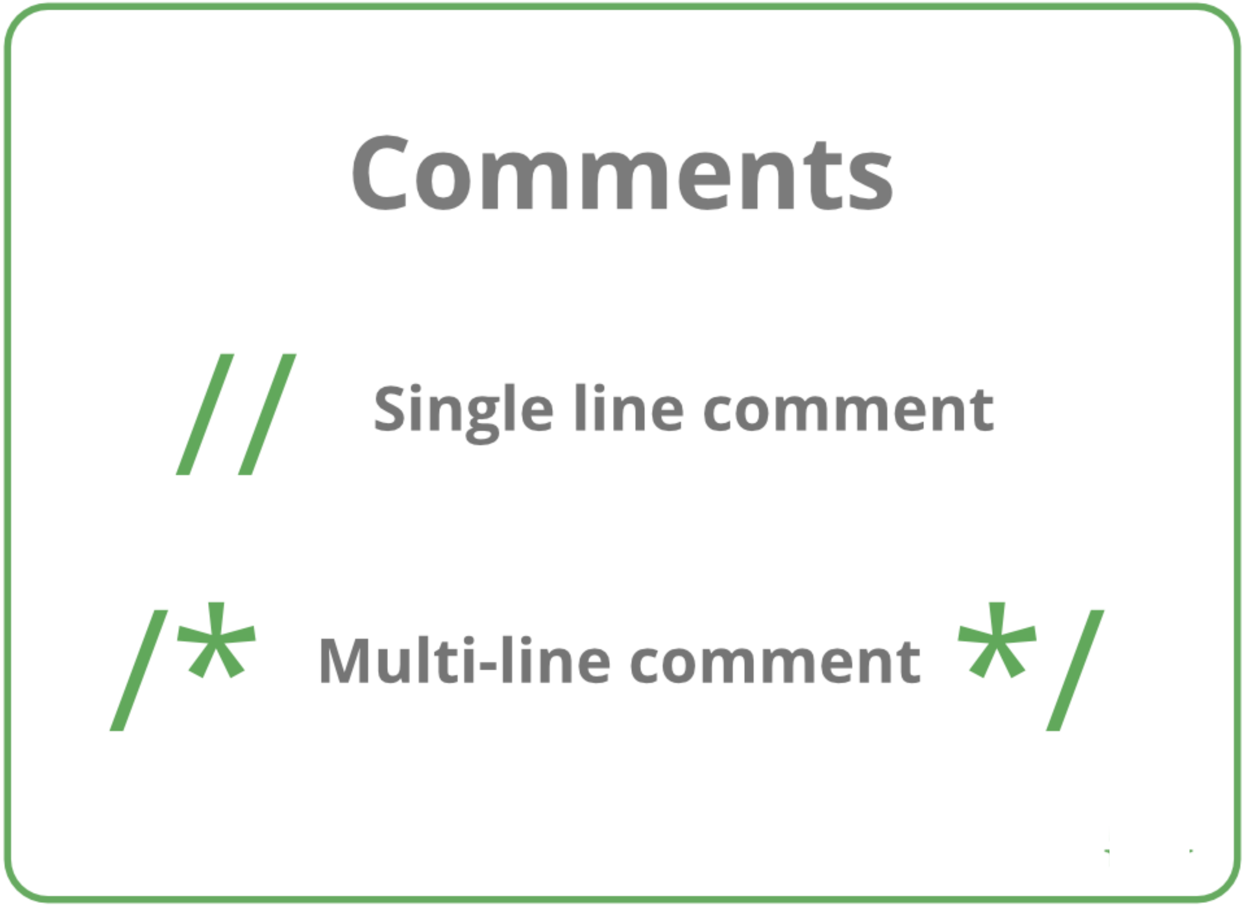
# 2.8. Comments

- A well-documented program is a good practice as a programmer. It makes a program more readable and error finding become easier. One important part of good documentation is Comments.
- In computer programming, a comment is a programmer-readable explanation or annotation in the source code of a computer program
- Comments are statements that are not executed by the compiler and interpreter.

**In C there are two types of comments :**

  - Single line comment
  - Multi-line comment

# 2.8. Comments



Comments

// Single line comment

/* Multi-line comment */

# 2.8. 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;
}
```

# 2.8. Comments

**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;
}
```

# 2.9. C token and Its Type

- A token is the smallest element of a program that is meaningful to the compiler.
- C tokens are the basic buildings blocks in C language which are constructed together to write a C program.
- Each and every smallest individual units in a C program are known as C tokens.

**C tokens are of six types. They are,**

| | |
|---|---|
| 1. Keywords | (eg: int, while), |
| 2. Identifiers | (eg: main, total), |
| 3. Constants | (eg: 10, 20), |
| 4. Strings | (eg: “total”, “hello”), |
| 5. Special symbols | (eg: (), {}), |
| 6. Operators | (eg: +, /,-,*) |

# 2.9. C token and Its Type

**1. KEYWORDS IN C LANGUAGE:**

- Keywords are pre-defined words in a C compiler.
- Each keyword is meant to perform a specific function in a C program.
- Since keywords are referred names for compiler, they can't be used as variable name.
- C language supports 32 keywords which are given below. Click on each keywords below for detail description and example programs.
- **Auto double int struct break else long**<br>**switch case enum register typedef char**<br>**extern return union const float short unsigned**<br>**continue for signed void default goto sizeof**<br>**volatile do if static while**

# 2.9. C token and Its Type

**2. IDENTIFIERS IN C LANGUAGE:**

- Each program elements in a C program are given a name called identifiers. Names given to identify Variables, functions and arrays are examples for identifiers. eg. x is a name given to integer variable in above program.

**RULES FOR CONSTRUCTING IDENTIFIER NAME IN C:**

- They must begin with a letter or underscore(_).
- They must consist of only letters, digits, or underscore. No other special character is allowed.
- It should not be a keyword.
- It must not contain white space.
- It should be up to 31 characters long as only first 31 characters are significant.

Some examples of c identifiers:

| NAME | REMARK |
|---|---|
| _A9 | Valid |
| Temp.var | Invalid as it contains special character other than the underscore |
| void | Invalid as it is a keyword |

# 2.9. C token and Its Type

**3. Constants:**

- Constants are also like normal variables. But, only difference is, their values can not be modified by the program once they are defined.
- Constants refer to fixed values. They are also called as literals. Constants may belong to any of the data type.

**Types of Constants:**

- Integer constants – Example: 0, 1, 1218, 12482
- Real or Floating point constants – Example: 0.0, 1203.03, 30486.184
- Octal & Hexadecimal constants – Example: octal: $(013\ )_8$ = $(11)_{10,}$ Hexadecimal: $(013)_{16}$ = $(19)_{10}$
- Character constants -Example: ‘a’, ‘A’, ‘z’
- String constants -Example: “Nipun Thapa”

# 2.9. C token and Its Type

**4. Strings:**

- Strings are nothing but an array of characters ended with a null character (‘\0’).
- This null character indicates the end of the string.
- Strings are always enclosed in double quotes.
- Whereas, a character is enclosed in single quotes in C and C++.**Declarations for String:**
  - char string[20] = {‘g’, ’e’, ‘e’, ‘k’, ‘s’, ‘f’, ‘o’, ‘r’, ‘g’, ’e’, ‘e’, ‘k’, ‘s’, ‘\0’};
  - char string[20] = “nipun thapa”;
  - char string [] = “nipun thapa”;

# 2.9. C token and Its Type

**5. Special Symbols:**

- The following special symbols are used in C having some special meaning and thus, cannot be used for some other purpose.[] () {}, ; * = #
  - **Brackets[]:** Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.
  - **Parentheses():** These special symbols are used to indicate function calls and function parameters.
  - **Braces{}:** These opening and ending curly braces marks the start and end of a block of code containing more than one executable statement.
  - **comma (, ):** It is used to separate more than one statements like for separating parameters in function calls.
  - **semi colon :** It is an operator that essentially invokes something called an initialization list.
  - **asterisk (*):** It is used to create pointer variable.
  - **assignment operator:** It is used to assign values.
  - **pre processor(#):** The preprocessor is a macro processor that is used automatically by the compiler to transform your program before actual compilation.

# 2.9. C token and Its Type

**6. Operators:**

- Operators are symbols that triggers an action when applied to C variables and other objects. The data items on which operators act upon are called operands.
- Depending on the number of operands that an operator can act upon, operators can be classified as follows:
  - **Unary Operators:** Those operators that require only single operand to act upon are known as unary operators.For Example increment and decrement operators
  - **Binary Operators:** Those operators that require two operands to act upon are called binary operators. **Binary operators are classified into :**
    1. Arithmetic operators
    2. Relational Operators
    3. Logical Operators
    4. Assignment Operators
    5. Conditional Operators
    6. Bitwise Operators
- **Ternary Operators:** These operators requires three operands to act upon. For Example Conditional operator(?:).

# 2.10.Pre-Processor Directives

- In a C program, all lines that start with **#** are processed by preprocessor which is a special program invoked by the compiler.
- In a very basic term, preprocessor takes a C program and produces another C program without any **#**.

Following are some interesting facts about preprocessors in C.

1. When we use ***include*** directive, the contents of included header file (after preprocessing) are copied to the current file. Angular brackets **<** and **>** instruct the preprocessor to look in the standard folder where all header files are held. Double quotes **"** and **"** instruct the preprocessor to look into the current folder (current directory).
2. When we use ***define*** for a constant, the preprocessor produces a C program where the defined constant is searched and matching tokens are replaced with the given expression.

E.g. #include<stdio.h> #define PI 3.14

# Finished

# Unit 2