MARUDHAR KESARI JAIN COLLEGE for WOMEN

PG DEPARTMENT OF COMPUTER APPLICATIONS

CLASS : I-BCA

SUBJECT NAME: STRUCTURED PROGRAMMING LANGUAGE IN C

SUBJECT CODE: 23UFC14

SYLLABUS

UNIT-1

Over view of C: Importance of C, sample C program, C program structure, executing C program. Constants, Variables, and Data Types: Character set, C Tokens, keywords and identifiers, constants, variables, data types, declaration of variables, Assigning values to variables--- Assignment statement, declaring variable and constant, as volatile. Operators and Expression.

Overview of C:

What is the C Language?

The C programming language is a general-purpose high-level language developed by Dennis Ritchie at Bell Labs in the 1970s. It was intended to be a low-level language that provided direct access to hardware and system resources, making it highly efficient and flexible. Due to its simplicity and elegance, C has become the building block for many other programming languages.

History of C Language

The C programming language's history can be traced to Dennis Ritchie, who developed it at Bell Labs in the early 1970s. It evolved from an earlier language called B, which was developed by Ken Thompson. B itself was based on the BCPL language.

C gained popularity due to its portability and efficiency. In 1978, Brian Kernighan and Dennis Ritchie published "The C Programming Language," which became the go-to reference for C programmers. Since then, C has been widely used and has influenced the development of other programming languages.

Application of C Programming Language

C programming language finds its application in various domains, including:

- Operating systems: C is extensively used in developing operating systems like Windows, Linux, and macOS.
- Embedded systems: C is the language of choice for developing firmware and software for various embedded systems, such as microcontrollers and IoT devices.
- System Utilities: Many system utilities, such as text editors, compilers, and interpreters, are written in C.
- Game development: C is used to develop games due to its ability to handle complex calculations and graphics.
- Database systems: The core components of databases and their management systems are often built using C.
- Networking: C is used for developing network protocols and applications.

Introduction

C programming is a powerful and versatile programming language that plays a significant role in software development. With its simplicity, efficiency, and wide range of applications, C has stood the test of time and continues to be a popular choice among programmers worldwide.

The importance of C programming cannot be overstated in the world of computer science and programming. As one of the oldest and most widely used programming languages, C has become the foundation for many modern programming languages.

In short, C programming holds immense importance for anyone interested in the world of coding and software development.

Importance of C programming language over other languages

C programming language offers several advantages over other programming languages, including:

- Efficiency: C allows for direct memory manipulation and low-level access to system resources. This results in highly efficient code execution.
- Portability: C code can be easily ported to different platforms without major modifications, thanks to its wide availability of compilers and libraries.
- Speed: C is known for its fast execution speed, making it suitable for developing performance-critical applications.
- Control: C gives programmers fine-grained control over memory management and system resources.
- Compatibility: C code can be easily integrated with code written in other languages like C++, Java, and Python.

Advantages of C Programming Language

Apart from the benefits mentioned above, there are several advantages of using C programming language, including:

- Modularity: C supports modular programming, allowing developers to break down complex programs into smaller, manageable modules for easier development and maintenance.
- Wide community support: C has a large and active development community, making it easy to find resources, libraries, and support online.
- Skill transferability: Once you learn C, it becomes easier to learn other programming languages since many languages, including C++, are based on C.

Disadvantages of C Programming Language

While C programming language offers many advantages, it also has some disadvantages, including:

- Complexity: C can be difficult to learn for beginners due to its low-level nature and the need to understand concepts like pointers and memory management.
- Lack of safety features: C does not have built-in features like garbage collection, which can lead to memory leaks and other runtime errors if not managed properly.
- Vulnerabilities: The low-level nature of C makes it prone to security vulnerabilities like buffer overflows if not correctly handled by the programmer.

Sample C Program:

To understand this example, you should have the knowledge of the following C programming topics:

```
#include <stdio.h>

void main()

{   // printf() displays the string inside quotation

   printf("Hello, World!");

}
```

Parts of C program-

# include <stdio.h> – This command is a preprocessor directive in C that includes all standard input-output files before compiling any C program so as to make use of all those functions in our C program.

void main() – This is the line from where the execution of the program starts. The main() function starts the execution of any C program.

{ (Opening bracket) – This indicates the beginning of any function in the program (Here it indicates the beginning of the main function).

/* some comments */ – Whatever is inside /*——-*/ are not compiled and executed; they are only written for user understanding or for making the program interactive by inserting a comment line. These are known as multiline comments. Single line comments are represented with the help of 2 forward slashes "//——".

printf("Hello World") –The printf() command is included in the C stdio.h library, which helps to display the message on the output screen.

getch() – This command helps to hold the screen.

return 0 –This command terminates the C program and returns a null value, that is, 0.

} (Closing brackets)- This indicates the end of the function. (Here it indicates the end of the main function)

How "Hello, World!" program works?

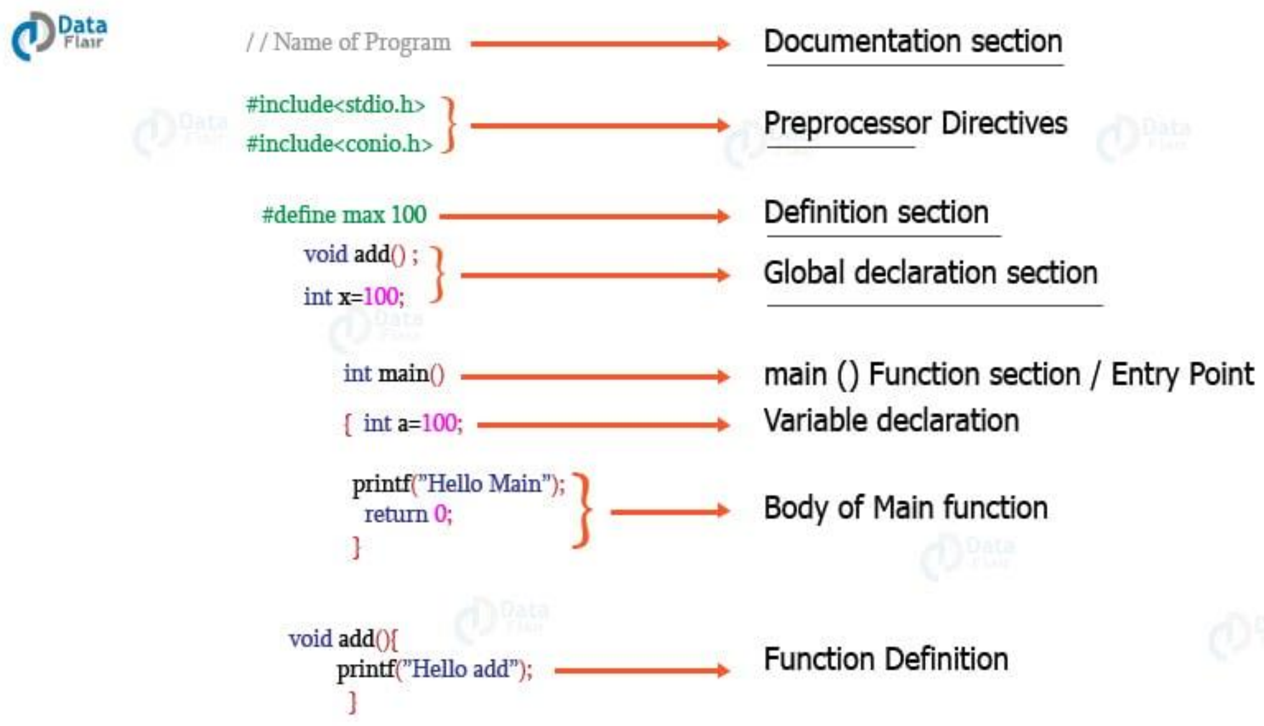
- The #include is a preprocessor command that tells the compiler to include the contents of stdio.h (standard input and output) file in the program.
- The stdio.h file contains functions such as scanf() and printf() to take input and display output respectively.
- If you use the printf() function without writing #include <stdio.h>, the program will not compile.
- The execution of a C program starts from the main() function.

- printf() is a library function to send formatted output to the screen. In this program, printf() displays Hello, World! text on the screen.
- The return 0; statement is the **"Exit status"** of the program. In simple terms, the program ends with this statement.

Basic Structure of C Program

The components of the basic structure of a C program consists of 7 parts

1. Document section
2. Preprocessor/link Section
3. Definition section
4. Global declaration section
5. Function declaration section
6. Main function
7. User-defined function section



```
//Name of Program
#include<stdio.h>
#include<conio.h>
#define max 100
void add() ;
int x=100;
int main()
{ int a=100;
printf("Hello Main");
return 0;
}

void add(){
printf("Hello add");
}
```

Example of C Program Structure

The “Hello World!” example is the most popular and basic program that will help you get started with programming. This program helps you display the output “Hello World” on the output screen.

With the help of this example, we can easily understand the basic structure of a C program.

```
#include <stdio.h>

int main()

{

// Our first basic program in C

printf("Hello World!\n\n");

return 0;

}
```

Structure of C program with example

```
#include <stdio.h> /* Link section */

int subtract = 0; /* Global declaration, definition section */

int all (int, int); /* Function declaration section */

int main () /* Main function */

{

printf("Welcome to DataFlair tutorials!\n\n");

printf ("This is a C program \n");

subtract= all (25,10);

printf ("Subtraction of the two numbers : %d \n", subtract);
return 0;

}

int all (int x, int y) /* User defined function */

{

return x-y; /* definition section */

}
```

1. Documentation Section

It is the section in which you can give comments to make the program more interactive. The compiler won’t compile this and hence this portion would not be displyed on the output screen.

2. Preprocessor directives Section

This section involves the use of header files that are to included necessarily program.

3. Definition section

This section involves the variable definition and declaration in C.

4. Global declaration Section

This section is used to define the global variables to be used in the programs, that means you can use these variables throughout the program.

5. Function prototype declaration section

This section gives the information about a function that includes, the data type or the return type, the parameters passed or the arguments.

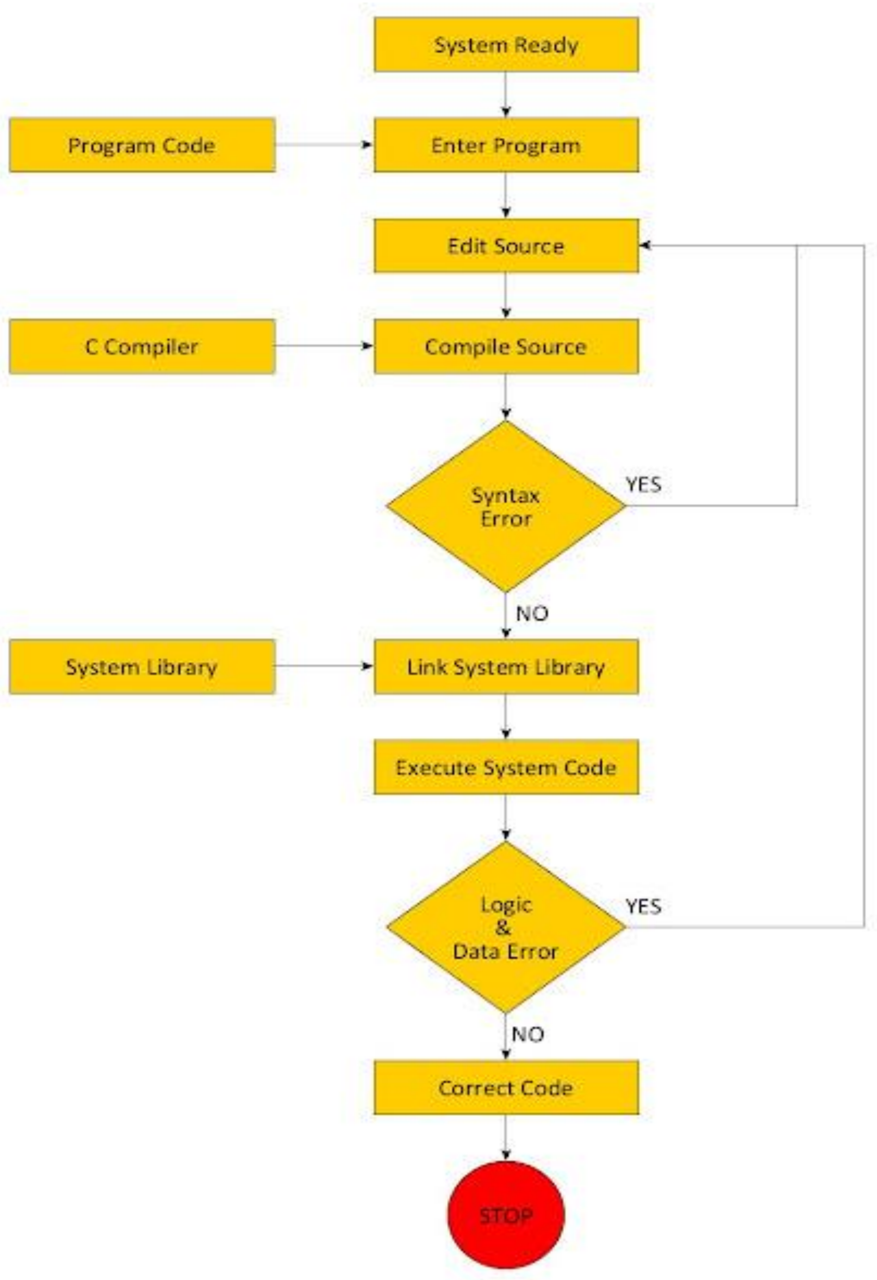
6. Main function

It is the major section from where the execution of the program begins. The main section involves the declaration and executable section.

7. User-defined function section

When you want to define your function that fulfills a particular requirement, you can define them in this section.

Excuting C Program:



1. The C program (source code) firstly, is sent to the preprocessor. To convert the preprocessor directives into their respective values, the preprocessor is responsible. An expanded source code is generated by the preprocessor.

2. 2. Then the expanded source code is sent to the compiler that compiles the code and converts into the assembly code.
3. 3. The assembly code is then sent to the assembler that assembles the code and converts into the object code. Then, a simple.obj file is generated.
4. 4. The object code is then sent to the linker that links it to the library like header files. In the next step, it is converted into an executable code. Then, a simple.exe file is generated.
5. 5. The executable code is then sent to the loader that loads the code into the memory followed by the execution of the code. The output is then sent to the console, after execution.

Character Set In C

Characters are used in forming either words or numbers or even expressions in C programming. Characters in C are classified into 4 groups:

Letters

In C programming, we can use both uppercase and lowercase letters of English language

- Uppercase Letters: A to Z
- Lowercase Letters: a to z

Digits

We can use decimal digits from 0 to 9.

Special Characters

C Programming allows programmer to use following special characters:

` ~ @ ! $ # ^ * % & ( ) [ ] { } < > + = _ – | / \ ; : ' " , . ?

White Spaces

In C Programming, white spaces contains:

- Blank Spaces
- Tab
- Carriage Return
- New Line

Summary of Special Characters in C

- Here is a table that represents all the types of character sets that we can use in the C language:

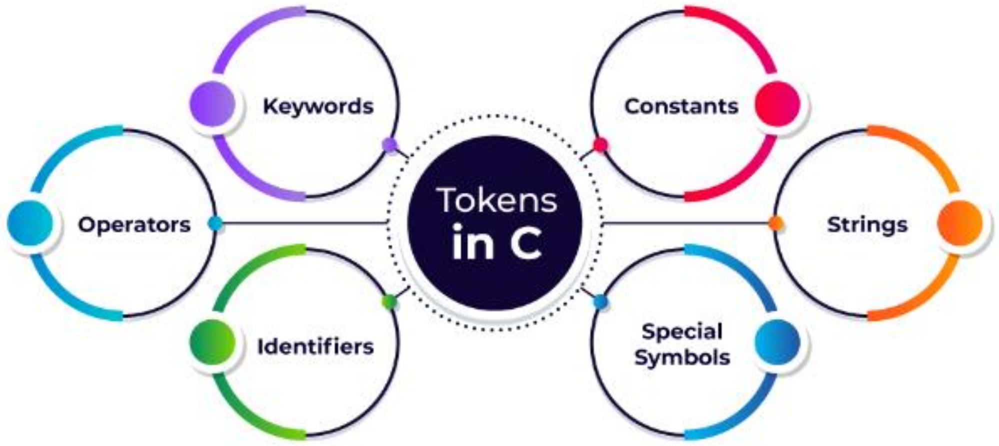
Type of Character	**Description**	**Characters**
Lowercase Alphabets	a to z	a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z
Uppercase Alphabets	A to Z	A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z
Digits	0 to 9	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special Characters	–	` ~ @ ! $ # ^ * % & ( ) [ ] { } < > + = _ – | / \ ; : ‘ “ , . ?
White Spaces	–	Blank Spaces, Carriage Return, Tab, New Line

Tokens in C

A token in C can be defined as the smallest individual element of the C programming language that is meaningful to the compiler. It is the basic component of a C program.

Types of Tokens in C

The tokens of C language can be classified into six types based on the functions they are used to perform. The types of C tokens are as follows:



1. **Keywords**
2. **Identifiers**
3. **Constants**
4. **Strings**
5. **Special Symbols**
6. **Operators**

1. C Token – Keywords

The keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program. Since keywords are referred names for a compiler, they can’t be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed. You cannot redefine keywords. However, you can specify the text to be substituted for keywords before compilation by using C preprocessor directives. **C** language supports **32** keywords which are 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**

2. C Token – Identifiers

Identifiers are used as the general terminology for the naming of variables, functions, and arrays. These are user-defined names consisting of an arbitrarily long sequence of letters and digits with either a letter or the underscore(_) as a first character. Identifier names must differ in spelling and case from any keywords. You cannot use keywords as identifiers; they are reserved for special use. Once declared, you can use the identifier in later program statements to refer to the associated value. A special identifier called a statement label can be used in goto statements.

Rules for Naming Identifiers

Certain rules should be followed while naming c identifiers which are as follows:

- 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 the first 31 characters are significant.

3. C Token – Constants

The constants refer to the variables with fixed values. They are like normal variables but with the difference that their values can not be modified in the program once they are defined.

Constants may belong to any of the data types.

Examples of Constants in C

const int c_var = 20;

const int* const ptr = &c_var;

4. C Token – 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++.

Examples of String

char string[20] = {'g', 'e', 'e', 'k', 's', 'f', 'o', 'r', 'g', 'e', 'e', 'k', 's', '\0'};

char string[20] = "geeksforgeeks";

char string [] = "geeksforgeeks";

5. C Token – Special Symbols

The following special symbols are used in C having some special meaning and thus, cannot be used for some other purpose. Some of these are listed below:

- **Brackets[]:** Opening and closing brackets are used as array element references. 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 mark the start and end of a block of code containing more than one executable statement.

- **Comma (,):** It is used to separate more than one statement like for separating parameters in function calls.
- **Colon(:):** It is an operator that essentially invokes something called an initialization list.
- **Semicolon(;):** It is known as a statement terminator. It indicates the end of one logical entity. That's why each individual statement must be ended with a semicolon.
- **Asterisk (*):** It is used to create a pointer variable and for the multiplication of variables.
- **Assignment operator(=):** It is used to assign values and for logical operation validation.
- **Pre-processor (#):** The preprocessor is a macro processor that is used automatically by the compiler to transform your program before actual compilation.
- **Period (.):** Used to access members of a structure or union.
- **Tilde(~):** Used as a destructor to free some space from memory.

6. C Token – Operators

Operators are symbols that trigger an action when applied to C variables and other objects. The data items on which operators act 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 a 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 can further are classified into:
  1. Arithmetic operators
  2. Relational Operators
  3. Logical Operators
  4. Assignment Operators
  5. Bitwise Operator
- **Ternary Operator**: The operator that requires three operands to act upon is called the ternary operator. Conditional Operator(?) is also called the ternary operator.

What are Constants in C?

As the name suggests, Constants are the variables whose values cannot be changed throughout the execution of the program once they are initialized at the beginning of the program. Constants are also known as literals. A number, a character, a string of characters, an array, a structure, a union, a pointer, and an enum can be set as a constant.

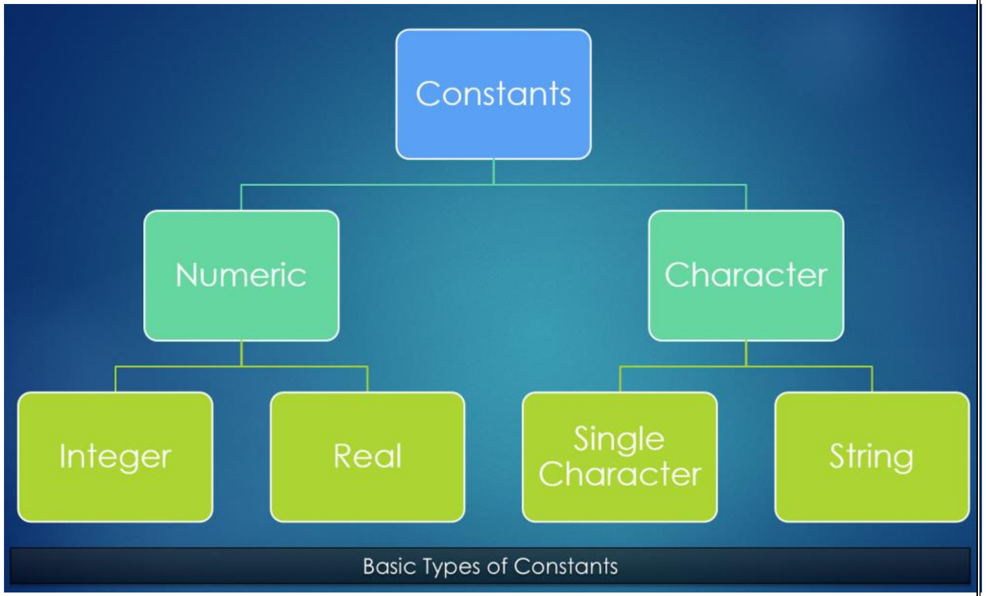
How to Use Constants in C?

In the C programming language, A variable can be used as a constant by the following methods:

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

Before we start creating the constants, Let us have an insight into the different kinds of Constants in C.

Types of Constants in C



Constants can be broadly divided into two types:

Numeric Constants

Numeric constants contain signed or unsigned numerals, or a zero or a decimal. In a nutshell, all types of numbers come under Numeric constants.

Numeric constants are once again divided into three types:

- Decimal Integer
- Octal Integer
- Hexadecimal Integer

Integer Constants

Integer constants are the numbers with decimals (base 10), hexadecimal (base 16), binary (base 2), or octal (base 8). We will understand this better once we look into each of the Integer constants.

Let us know more about each integer constant in detail.

Decimal Integer

Decimal integers are the constants with base 10. Non-zero decimal digits (1 to 9) are decimal integers followed by zero or more decimal digits (0 to 9).

Example: 255,100,69,999, etc.

Octal Integer

Octal integers are the constants with base 8. The digit zero (0) is followed by zero or more octal digits (0 to 7).

Example:0, 0125, 034673, 03245, etc.

Hexadecimal Integer

Hexadecimal integers are the constants with base 16. The sequence starts with 0x followed by one or more hexadecimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, A, b, B, c, C, d, D, e, E, f, F).

Example:0x3b24, 0XF96, 0x21, 0x3AA, 0X29b, 0X4bD, etc.

Real Constants

A constant with the combination of positive, negative, or zero followed by a decimal point and the fractional part is known as a real constant.

Example:-89, 0.123, 45, etc.

Character Constants

One or more characters are enclosed within a single quote (' ') or ("") depending on the type of characters.

Single Character Constants
Character constants having a single character enclosed within the ' ' (Single quote) are known as character constants.
Example: 's', 'c', 'a', 'l', 'e', 'r', etc.

String Constants
Character constants with various special symbols, digits, characters, and escape sequences enclosed within the " "(double quotes) are known as string constants.

Let us look at the examples and try to understand them better.
Example: "Scaler", "by", "InterviewBit", "123", "number1" etc.

As we can see in the above examples, all the words or the strings are enclosed within the double-quotes. If we look at the last two examples, we can see numbers, but the numbers are treated as strings by the computer because they are enclosed inside the double quotes.

Backslash Character Constants

Backslash characters or escape sequences are the types of character constants. A definite backslash character performs a specific task. Let's imagine that we are writing a code, and in the code, we added a couple of sentences, and we want them to be printed in separate lines. For this purpose, we can write the printf statement thrice, but it is not an efficient way to write the code. Now we have to tell the compiler to print the sentences in 3 different lines and how can we tell it to the compiler? We can use the backslash character \n to tell the compiler to print the sentences in 3 different lines. Here is a list of backslash characters and their meaning:

Constants	**Meaning**
\a	beep sound
\b	backspace
\f	form feed

Constants	Meaning
\n	new line
\r	carriage return
\t	horizontal tab
\v	vertical tab
\'	single quote
\"	double quote
\\	backslash
\0	null

Variables:

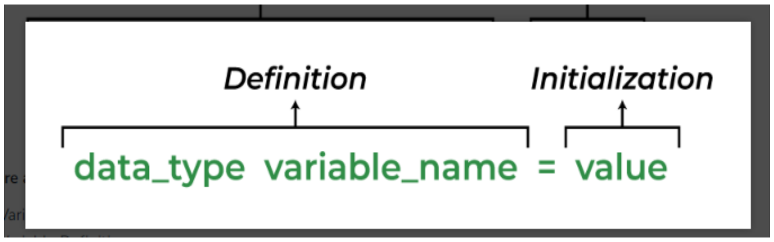
A **variable in C** is a memory location with some name that helps store some form of data and retrieves it when required. We can store different types of data in the variable and reuse the same variable for storing some other data any number of times.

C Variable Syntax

The syntax to declare a variable in C specifies the name and the type of the variable.

```
data_type variable_name = value;    // defining single variable
          or
data_type variable_name1, variable_name2;    // defining multiple variable
```

- **data_type:** Type of data that a variable can store.
- **variable_name:** Name of the variable given by the user.
- **value:** value assigned to the variable by the user.



Example

```
int var;    // integer variable

char a;     // character variable

float fff;  // float variables
```

here are 3 aspects of defining a variable:

1. Variable Declaration
2. Variable Definition
3. Variable Initialization

1. C Variable Declaration

Variable declaration in C tells the compiler about the existence of the variable with the given name and data type. No memory is allocated to a variable in the declaration.

2. C Variable Definition

In the definition of a C variable, the compiler allocates some memory and some value to it. A defined variable will contain some random garbage value till it is not initialized.

Example

```
int var;

char var2;
```

Note: *Most of the modern C compilers declare and define the variable in single step. Although we can declare a variable in C by using extern keyword, it is not required in most of the cases. To know more about variable declaration and definition, click here.*

3. C Variable Initialization

Initialization of a variable is the process where the user assigns some meaningful value to the variable.

Example

```
int var; // variable definition
var = 10; // initialization
    or
int var = 10; // variable declaration and definition
```

Rules for Naming Variables in C

You can assign any name to the variable as long as it follows the following rules:

1. A variable name must only contain alphabets, digits, and underscore.
2. A variable name must start with an alphabet or an underscore only. It cannot start with a digit.
3. No whitespace is allowed within the variable name.
4. A variable name must not be any reserved word or keyword.

Assignment Statement

*An **Assignment statement** is a statement that is used to set a value to the variable name in a program.*

5. Assignment statement allows a variable to hold different types of values during its program lifespan. Another way of understanding an assignment statement is, it stores a value in the memory location which is denoted by a variable name.

Syntax

The symbol used in an assignment statement is called as an **operator**. The symbol is **'='**.

Note: The Assignment Operator should never be used for Equality purpose which is double equal sign '=='.

The ***Basic Syntax of Assignment Statement*** in a programming language is :

variable = expression ;

where,

variable = variable name

expression = it could be either a direct value or a math expression/formula or a function call

Few programming languages such as Java, C, C++ require data type to be specified for the variable, so that it is easy to allocate memory space and store those values during program execution.

data_type variable_name = value ;

Example –

```
int a = 50 ;

float b ;

a = 25 ;

b = 34.25f ;
```

Declare variable as constant in C

The const keyword

Variables can be declared as constants by using the “const” keyword before the datatype of the variable. The constant variables can be initialized once only. The default value of constant variables are zero.

A program that demonstrates the declaration of constant variables in C using const keyword is given as follows.

Example

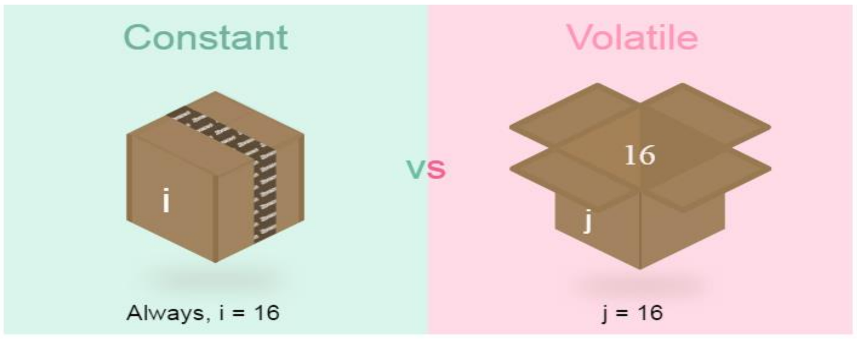
```
#include <stdio.h>

int main() {
   const int a;

   const int b = 12;

   printf("The default value of variable a : %d", a);

   printf("

The value of variable b : %d", b);

   return 0;

}
```

Constants and Volatile

Constants

Things which are all unchangable are said to be **constant** whereas things which are all changable are said to be **volatile**. The following diagram clearly represents the relationship between constant and volatile.

In the following diagram sealed box contains a variable **i** which is initialized by 16 at the time of declaration which indicates that one cannot change the value of variable i after initialization whereas opened box contains a varibale **j** which is intialized by 30 at the time of declaration, open box indicates the one can change the value of variable **j** at anytime.



Constant vs Volatile

The following table represents the constant against volatile

Constant	**Volatile**
Constant variables are **unchangable**.	Volatile variables are **changable**.
Constant variable can be created by using the keyword **const**.	Volatile varibale can be created by using the keyword **Volatile**.
For example **const** int i = 16 is the way of declaring and initializing constant variable.	For example **volatile** int j = 30 is the way of declaring and initializing volatile variable.
Constant variable can only be initialized at the time of declaration.	Volatile varibale can be initialized at anytime.
By default, all variables are not constant.	By default, all variables are volatile.
The const variable is otherwise known as Read Only Variable.	The volatile variable is otherwise known as Read/Write Variable.
Another way to achieve constant variable by the use of #define preprocessor directive e.g.) **#define a 5**.	Another way to achieve volatile variable by the use of nothing e.g.) **int a = 5** is volatile.

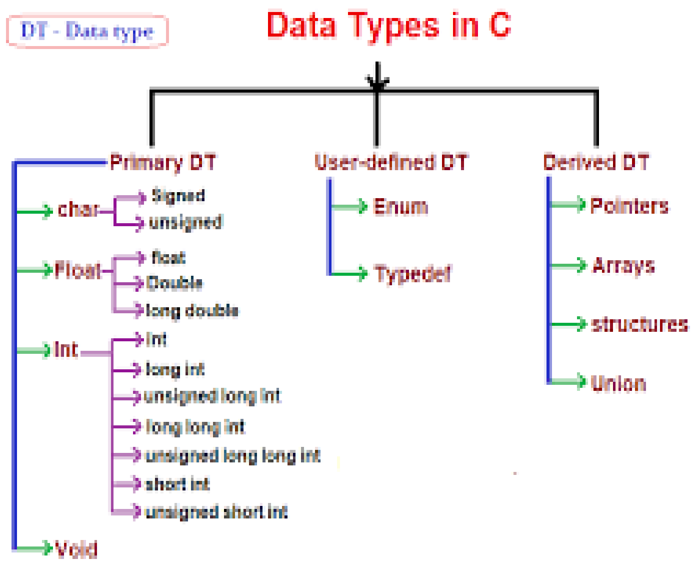
Data Types in C

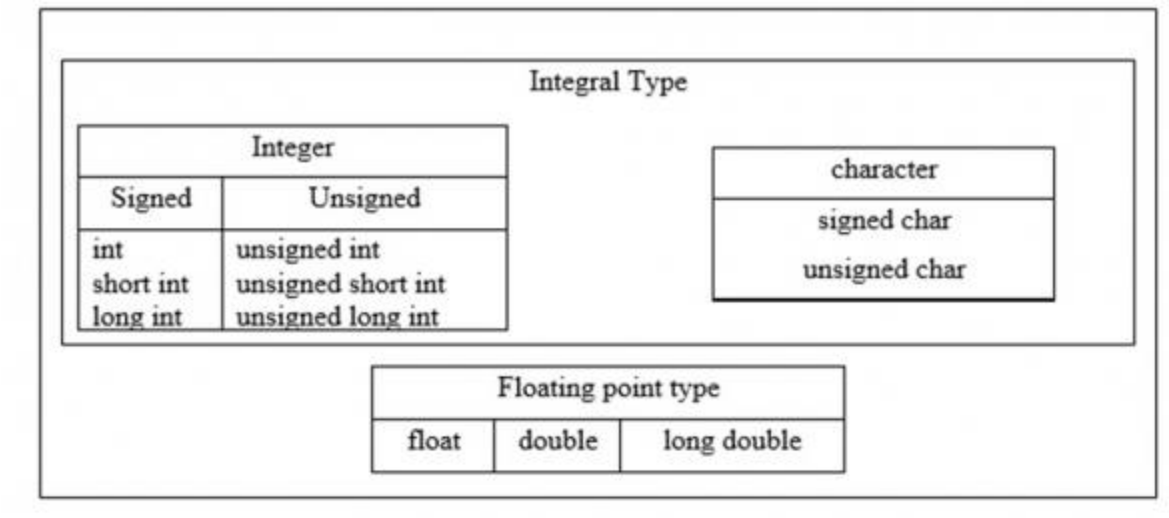
A data type specifies the type of data that a variable can store such as integer, floating, character, etc.

Primary or Fundamental Data types:

‘C’ compilers support four fundamental data types. They are as follows −

- Integer
- Character
- Floating − point
- Double precision floating point





Integral Type

Integer	
Signed	Unsigned
int short int long int	unsigned int unsigned short int unsigned long int

character
signed char
unsigned char

Floating point type		
float	double	long double

Integral data type

Integral data types are used to store whole numbers and characters.

It is further classified into two types −

- Integer data type.
- Character data type.

1. Integer data type

   This data type is used to store whole numbers.

   The integer storage are short int, int and long int in both signed and unsigned forms.

Integer Data Types			
Type	**size(in bytes)**	**Range**	**Control String**
short in (or) signed short int	1	-128 to 127	%h
unsigned short int	1	0 to 255	%uh
int (or) signed int	4	-32768 to 32767	%d or %i
unsigned int	4	0 to 65535	%u
Long int (or) signed long int	4	-2147483648 to 2147483647	%ld
Unsigned long int	4	0 to 4294967295	%lu

2. Character data type

   - ✓ Character data type is used to store characters only.
   - ✓ These characters are internally stored as integers.
   - ✓ Each character has an equivalent ASCII value.
   - ✓ For example, ‘A’ has ASCII value 65.

Character Data Types			
Type	**Size(in bytes)**	**Range**	**Control String**
Char(or) signed Char	1	-128 to 127	%C
unsigned Char	1	0 to 255	%c

3. Floating - point Data type

Floating point data types are used to store real numbers.

float’ is used for 6 digits of accuracy.

‘double' is used for 12 digits of accuracy.

More than 12 digits, ‘long double’ is used.

Floating Point Data Types			
Type	**size(in bytes)**	**Range**	**Control String**
Float	4	3.4E - 38 to 3.4 E + 38	%f
Double	8	1.7 E - 308 to 1.7 E + 308	%lf
long double	16	3.4 E - 4932 to 1.1 E + 4932	%Lf

4. Void Data Type:The **void data type** is an empty data type that refers to an object that does not have a value of any type.

C – Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Increment and Decrement Operators

- Conditional operators
- Bitwise Operators
- Special Operators

1. **Arithmetic Operators:** It is used to performing mathematical operations such as addition, subtraction, multiplication, division, modulus, etc., on the given operands.

Syntax:
C = A + B;

Example:

```
#include <stdio.h>
#include <conio.h>
Void main ()
{
int num1, num2, res;
float f1, f2, res1;
printf (" Enter two integer numbers: ");
scanf ("%d %d", &num1, &num2);
res = num1 + num2;
printf (" Enter two float numbers: \n ");
scanf ("%f %f", &f1, &f2);
res1 = f1 + f2; // use + operator
printf (" The sum of two integer numbers: %d \n", res);
printf (" The sum of two float numbers: %f \n ", res1);
printf (" The sum of two double numbers: %lf", res2);
}
```

2. Relational Operators: Relational Operators are the operators used to create a relationship and compare the values of two operands.

Syntax:
Opr1 == Opr2

Example:

```
#include <stdio.h>

#include <math.h>

void main ()

{

    int a = 5;

    int b = 10;

    printf (" a == b : %d", (a == b));

    if (a == b)

        printf ("\n %d is equal to %d", a, b);

    else

        printf (" \n %d is not equal to %d", a, b);

 }
```

3. Logical Operators:

logical operators are used to evaluate the truth value of an expression or a condition.

There are three logical operators in C: "&&" (logical AND), "||" (logical OR), and "!" (logical NOT).

```
#include <stdio.h>

void main()

{

int x = 22, y = 33;

if (x > 0 && y > 0)

{

   printf("Both values are greater than 0\n");

}

else
{
   printf("Both values are less than 0\n");
}
}
```

4. Assignment Operators: The assignment operator is used to assign the value, variable and function to another variable.

the various types of the assignment operators such as =, +=, -=, /=, *= and %=.

Example of the Assignment Operators:

```
A = 5; // use Assignment symbol to assign 5 to the operand A

B = A; // Assign operand A to the B

B = &A; // Assign the address of operand A to the variable B

A = 10 * 2 + 5; // assign equation to the variable A
```

5. Increment and Decrement Opeartors:

Increment Operators are the unary operators used to increment or add 1 to the operand value. The Increment operand is denoted by the double plus symbol (++). It has two types, Pre Increment and Post Increment Operators.

- ❖ Pre-increment Operator

The pre-increment operator is used to increase the original value of the operand by 1 before assigning it to the expression.

Syntax X = ++A;

- ❖ Post increment Operator

The post-increment operator is used to increment the original value of the operand by 1 after assigning it to the expression.

Syntax X = A++;

Example:

```
#include <stdio.h>

#include <conio.h>

void main ()

{
 // declare integer variables
int x, a, b, c;
printf (" Input the value of X: ");
scanf (" %d", &x);
printf (" Input the value of Y: ");
scanf (" %d", &y);
    // use post-increment operator to update the value by 1
a = x++;
b = y++;
c=++x;
printf (" \n The original value of a: %d", a);
printf (" \n The original value of b: %d", b);
printf (" \n The original value of c: %d", c);
}
```

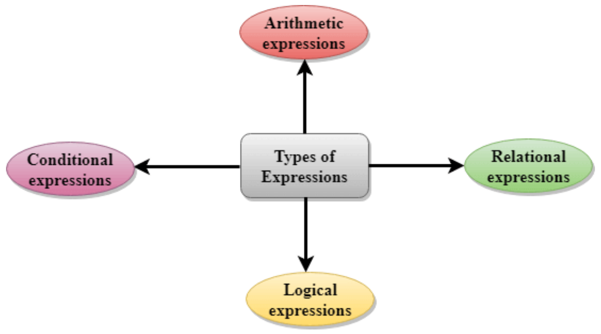
Expressions:

An expression is a formula in which operands are linked to each other by the use of operators to compute a value. An operand can be a function reference, a variable, an array element or a constant.

a-b; In the above expression, minus character (-) is an operator, and a, and b are the two operands.

Types:

- Arithmetic expressions
- Relational expressions
- Logical expressions
- Conditional expressions



Arithmetic Expressions

- An arithmetic expression is an expression that consists of operands and arithmetic operators. An arithmetic expression computes a value of type int, float or double.

<u>Evaluation of Arithmetic Expressions</u>

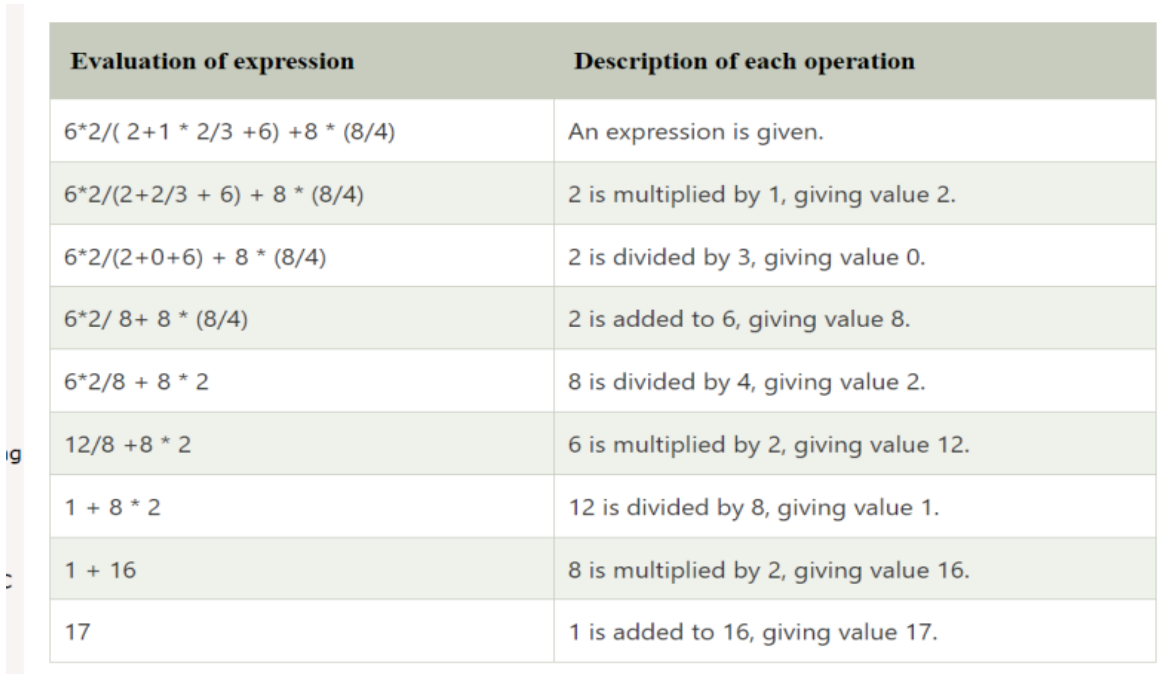
- The expressions are evaluated by performing one operation at a time. The precedence and associativity of operators decide the order of the evaluation of individual operations.

<u>When individual operations are performed, the following cases can be happened:</u>

- When both the operands are of type integer, then arithmetic will be performed, and the result of the operation would be an integer value. For example, 3/2 will yield 1 not 1.5 as the fractional part is ignored.
- When both the operands are of type float, then arithmetic will be performed, and the result of the operation would be a real value. For example, 2.0/2.0 will yield 1.0, not 1.
- If one operand is of type integer and another operand is of type real, then the mixed arithmetic will be performed. In this case, the first operand is converted into a real operand, and then arithmetic is performed to produce the real value. For example, 6/2.0 will yield 3.0 as the first value of 6 is converted into 6.0 and then arithmetic is performed to produce 3.0.

Let's understand through an example.

6*2/ (2+1 * 2/3 + 6) + 8 * (8/4)



Evaluation of expression	Description of each operation
6*2/(2+1 * 2/3 +6) +8 * (8/4)	An expression is given.
6*2/(2+2/3 + 6) + 8 * (8/4)	2 is multiplied by 1, giving value 2.
6*2/(2+0+6) + 8 * (8/4)	2 is divided by 3, giving value 0.
6*2/ 8+ 8 * (8/4)	2 is added to 6, giving value 8.
6*2/8 + 8 * 2	8 is divided by 4, giving value 2.
12/8 +8 * 2	6 is multiplied by 2, giving value 12.
1 + 8 * 2	12 is divided by 8, giving value 1.
1 + 16	8 is multiplied by 2, giving value 16.
17	1 is added to 16, giving value 17.

Relational Expressions

- A relational expression is an expression used to compare two operands.
- It is a condition which is used to decide whether the action should be taken or not.
- In relational expressions, a numeric value cannot be compared with the string value.
- The result of the relational expression can be either zero or non-zero value. Here, the zero value is equivalent to a false and non-zero value is equivalent to true.

Example:

```
#include <stdio.h>
 Void main()
{
   int x=4;
   if(x%2==0)
   {
      printf("The number x is even");
   }
   else
   printf("The number x is not even");
}
```

Logical Expressions

- A logical expression is an expression that computes either a zero or non-zero value.

It is a complex test condition to take a decision.

Example:

```
#include <stdio.h>
int main()
{
   int x = 4;
   int y = 10;
   if ( (x <10) && (y>5))
   {
      printf("Condition is true");
```

```
        }
        else
        printf("Condition is false");
        return 0;
    }
```

Conditional Expressions

- A conditional expression is an expression that returns 1 if the condition is true otherwise 0.
- A conditional operator is also known as a ternary operator.

Syntax of Conditional operator

Suppose exp1, exp2 and exp3 are three expressions.