

Computer And Programming Fundamentals With C

What is C?

C is a general-purpose programming language created by Dennis Ritchie at the Bell Laboratories in 1972.

It is a very popular language.

C is strongly associated with UNIX, as it was developed to write the UNIX operating system.

• **Why Learn C?**

It is one of the most popular programming language in the world.

If you know C, you will have no problem learning other popular programming languages such as Java, Python, C++, C#, etc, as the syntax is similar.

C is very fast, compared to other programming languages, like [Java](#) and [Python](#)

C is very versatile; it can be used in both applications and technologies

Characteristics of c language:

- **Simple and Efficient**

The basic syntax style of implementing C language is very simple and easy to learn. This enables a programmer to redesign or create a new application.

- **Fast**

C is a compiler-based program. This makes the compilation and execution of codes faster.

- **Extensibility**

You can easily (and quickly) extend a C program. This means that if a code is already written, you can add new features to it with a few alterations. Basically, it allows adding new features, functionalities, and operations to an existing C program.

- **Function-Rich Libraries**

C comes with an extensive set of libraries with several built-in functions that make the life of a programmer easy. Even a beginner can easily code using these built-in functions. You can also create your user-defined functions and add them to C libraries.

The following code is one of the simplest C programs that will help us the basic syntax structure of a C program.

```
#include <stdio.h>
int main()
{
    printf("hello world");
    return 0;
}
```

Program development

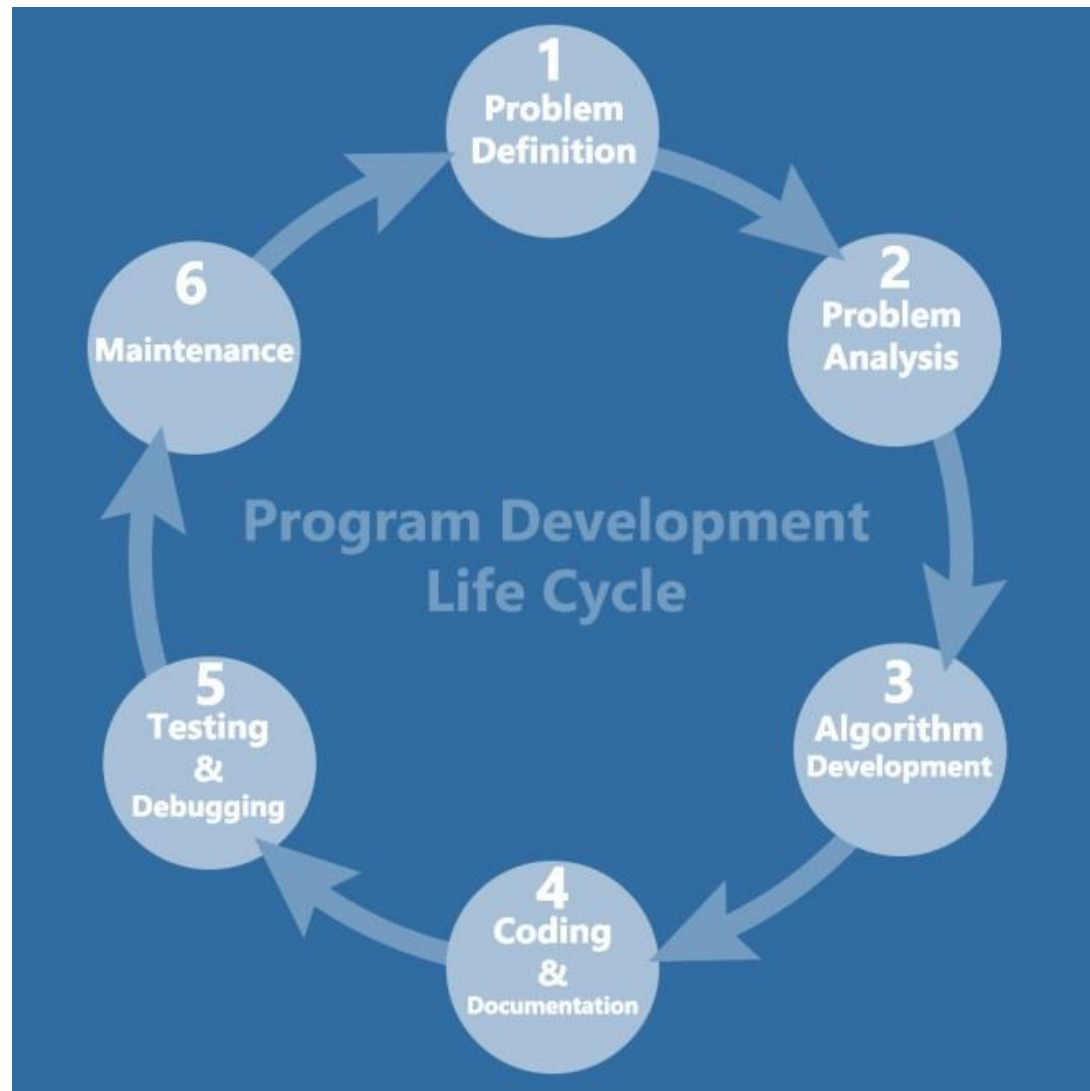
When we want to develop a program by using any programming language, we have to follow a sequence of steps. These steps are called phases in program development.

Program Development C

Program development is the process of creating application programs:

Program development life cycle contains 6 phases, which are as follows –

- Problem Definition.
- Problem Analysis.
- Algorithm Development.
- Coding & Documentation.
- Testing & Debugging
- Maintenance



- **Problem Definition** –In this phase, we need to understand what is the problem statement, what is our requirement and what is the output of the problem solution. All these are included in the first phase of program development life cycle.
- **Problem Analysis** Here, we determine the requirements like variables, functions, etc. to solve the problem. It means that we gather the required resources to solve the problem, which are defined in the problem definition phase.
- **Algorithm Development** Here, we develop a step-by-step procedure that is used to solve the problem. It is very important phase for the program development. We write the solution in step-by-step statements.

- **Coding & Documentation**

- Here, we use a programming language to write or implement the actual programming instructions for the steps defined in the previous phase. We construct the actual program in this phase. We write the program to solve the given problem by using the programming languages C.

- **Testing & Debugging**

- In this phase, we check whether the written code in the previous step is solving the specified problem or not. This means, we try to test the program whether it is solving the problem for various input data values or not. We also test if it is providing the desired output or not.

- **Maintenance**

- In this phase, we make the enhancements. Therefore, the solution is used by the end-user. If the user gets any problem or wants any enhancement, then we need to repeat all these phases from the starting, so that the encountered problem is solved or enhancement is added.

Structure of the C program

Structure of C Program

	1	<i>#include <stdio.h></i>	Header
	2	<i>int main(void)</i>	Main
BODY	3	<i>{</i>	
	4	<i>printf("Hello World");</i>	Statement
	5	<i>return 0;</i>	Return
	6	<i>}</i>	

Components of a C Program:

- **1. Header Files Inclusion – Line 1 [#include <stdio.h>]**

A header file is a file with extension .h which contains C function declarations and macro definitions to be shared between several source files. All lines that start with # are processed by a preprocessor which is a program invoked by the compiler. The .h files are called header files in C.

Some of the C Header files:

- `stdio.h` – Defines core input and output functions
- `math.h` – Defines common mathematical functions.

- **2. Main Method Declaration – Line 2 [int main()]**

- The next part of a C program is to declare the `main()` function. It is the entry point of a C program and the execution typically begins with the first line of the `main()`. The empty brackets indicate that the main doesn't take any parameter.

3. Body of Main Method – Line 3 to Line 6 [enclosed in {}]

The body of a function in the C program refers to statements that are a part of that function. It can be anything like manipulations, searching, sorting, printing, etc. A pair of curly brackets define the body of a function. All functions must start and end with curly brackets.

4. Statement – Line 4 [printf(“Hello World”);]

Statements are the instructions given to the compiler. In C, a statement is always terminated by a **semicolon (;)**. In this particular case, we use printf() function to instruct the compiler to display “Hello World” text on the screen.

5. Return Statement – Line 5 [return 0;]

The last part of any C function is the return statement. The return statement refers to the return values from a function. This return statement and return value depend upon the return type of the function.

- `#include<stdio.h>`
- `#include<conio.h>`
- `int main()`
 - `{`
 - `int num1,num2,sum;`
 - `printf("enter two integers:");`
 - `scanf("%d %d", &num1, &num2);`
 - `sum = num1 + num2;`
 - `printf("%d + %d = %d", num1, num2, sum);`
 - `return 0;`
 - `}`

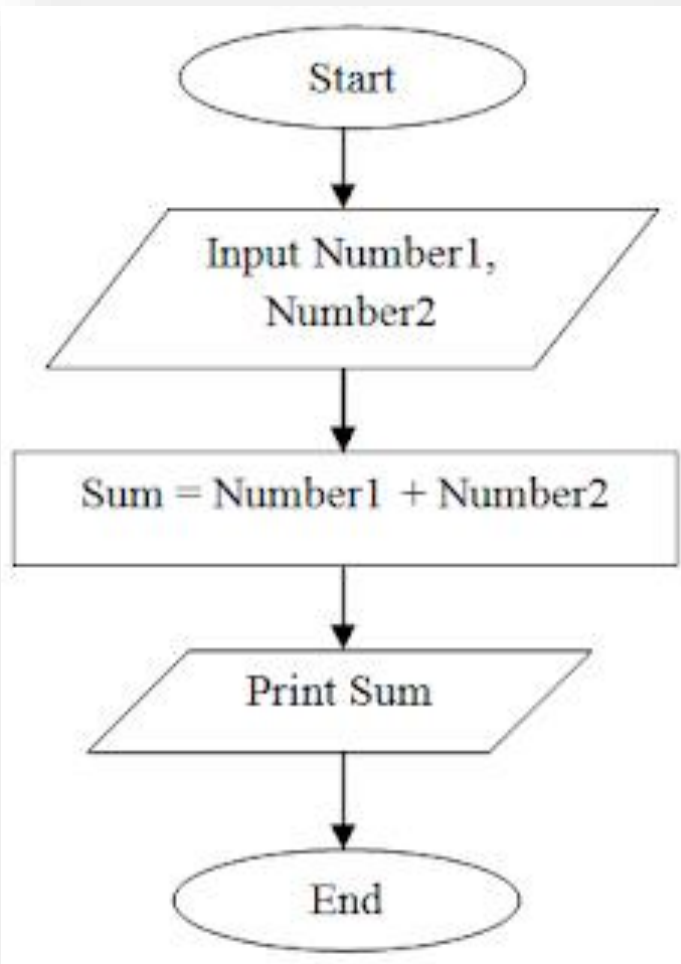
Programming Using Algorithm And Flow Chart

- Algorithms and flowcharts are different mechanisms used for designing different programs, particularly in computer programming.
- An algorithm is a step-by-step summary of the procedure.
- while on the other hand, a flowchart illustrates the steps of a program graphically.

Algorithm

- Example:
- Algorithm to add two numbers in C:
- 1) Start.
- 2) Accept Number one.
- 3) Accept Number two.
- 4) Add both the numbers.
- 5) Print the result.
- 6) End

Flow chart



Flowchart Guidelines

- To create a flowchart, you must follow the following current standard guideline:
- Step 1: Start the program.
- Step 2: Begin Process 1 of the program.
- Step 3: Check some conditions and take a Decision (“yes” or “no”).
- Step 4: If the decision is “yes”, proceed to Process 3. If the decision is “no”, proceed to Process 2 and return to Step 2.
- Step 5: End of the program.

Character set

A character set is a set of alphabets, letters and some special characters that are valid in C language.

Alphabets

Uppercase: A B C X Y Z

Lowercase: a b cx y z

C accepts both lowercase and uppercase alphabets as variables and functions.

Digits

0 1 2 3 4 5 6 7 8 9

Special Characters in C Programming

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

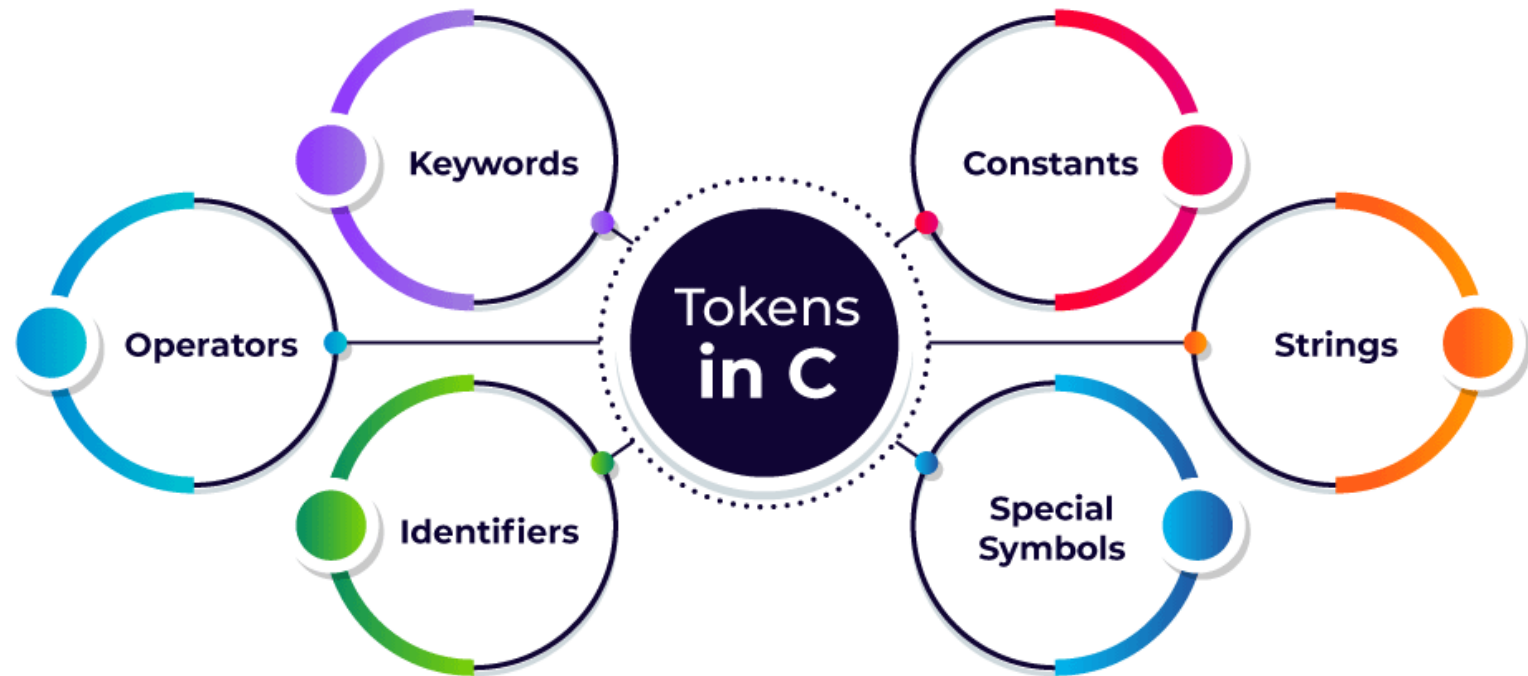
Token:-

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:

Keywords
Identifiers
Constants
Strings Special Symbols
Operators



Use of the Tokens in C

- For instance, without words, you cannot create any sentence- similarly, you cannot create any program without using tokens in C language. Thus, we can also say that tokens are the building blocks or the very basic components used in creating any program in the C language.

C Keywords

Keywords are predefined, reserved words used in programming that have special meanings to the compiler. Keywords are part of the syntax and they cannot be used as an identifier. For example:

```
int money;
```

Here, int is a keyword that indicates money is a variable of type int (integer).

As C is a case sensitive language, all keywords must be written in lowercase.

Keywords

- In 'C', there are 32 keywords. All keywords must be written in lower case.

These 32 keywords are classified into 3 types namely

1. Type related Keywords (16)
2. Storage related Keywords (4)
3. Control flow related Keywords (12)

1. Type related Keywords (16)

int	Short	Void	enum
float	Long	Struct	const
char	Signed	union	volatile
double	Unsign ed	typede f	sizeof

```
char keyword
#include <stdio.h>
int main()
{
    char c = 'a';
    printf("%c", c);
    return 0;
}
```

Output
a

2. Storage related Keywords (4)

Auto	Static	Register	Extern
------	--------	----------	--------

auto keyword

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

```
int main()
```

```
{
```

```
    auto int a = 10;
```

```
    printf("%d", a);
```

```
}
```

Output

10

Control flow related keywords:

if	default	Goto	for
else	case	continu e	while
switch	break	return	do

```
#include <stdio.h>
int main()
{
    int a = 10;
    if(a < 11)
    {
        printf("A is less than 11");
    }
    else
    {
        printf("A is equal to or "
               "greater than 11");
    }
    return 0;
}
```

C Identifiers

Identifier refers to name given to entities such as variables, functions, structures etc.

Identifiers must be unique. They are created to give a unique name to an entity to identify it during the execution of the program. For example:

int money;

double accountBalance;

Here, `money` **and** `accountBalance` are identifiers.

Also remember, identifier names must be different from keywords. You cannot use `int` as an identifier because `int` is a keyword.

Identifiers:

Rules to name a particular identifier:

- 1. It must start with either alphabet or underscore.
- 2. Remaining letters may be alphabet, digit, underscore.
- 3. Identifier would not allow any special symbol except underscore.
- 4. An identifier can be of any length while most compilers of 'C' language recognize only the first 8 characters.
- 5. Do not use keywords as an identifier.

Difference between Keyword and Identifier:

KEYWORD	IDENTIFIER
Keywords are predefined word that gets reserved for working program that have special meaning and cannot get used anywhere else.	Identifiers are the values used to define different programming items such as variables, integers, structures, unions and others and mostly have an alphabetic character.
Specify the type/kind of entity.	Identify the name of a particular entity.
It always starts with a lowercase letter.	First character can be a uppercase, lowercase letter or underscore.
A keyword should be in lower case.	An identifier can be in upper case or lower case.

KEYWORD

IDENTIFIER

A keyword contains only alphabetical characters.

An identifier can consist of alphabetical characters, digits and underscores.

They help to identify a specific property that exists within a computer language.

They help to locate the name of the entity that gets defined along with a keyword.

No special symbol, punctuation is used.

No punctuation or special symbol except 'underscore' is used.

Examples of keywords are: int, char, if, while, do, class etc.

Examples of identifiers are: Test, count1, high_speed, etc.

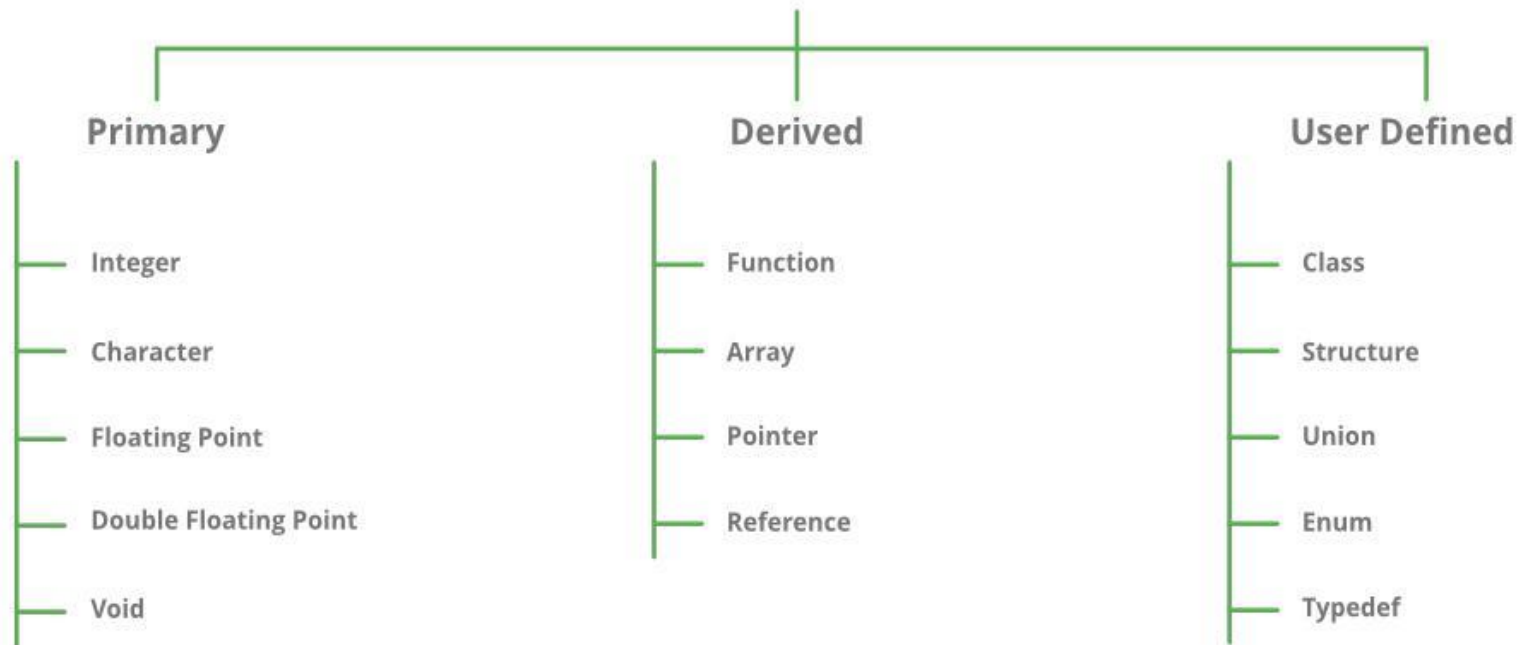
Data Types

Each variable in C has an associated data type. It specifies the type of data that the variable can store like integer, character, floating, double, etc. Each data type requires different amounts of memory and has some specific operations which can be performed over it. The data type is a collection of data with values having fixed values, meaning as well as its characteristics.

The data types in C can be classified as follows:

Types	Description
Primary Data Types	Primary data types are the most basic data types that are used for representing simple values such as integers, float, characters, etc.
User Defined Data Types	The user-defined data types are defined by the user himself.
Derived Types	The data types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types.

DataTypes in C



Different data types also have different ranges up to which they can store numbers. These ranges may vary from compiler to compiler.

Primary data types

Primary data types are also known as the fundamental data types because they are pre-defined or they already exist in the C language. All the other types of data types (derived and user-defined data types) are derived from these data types.

Primary data types in C are of 4 types:

- int
- char
- Float
- Double
- void

Integer (int):

1. We use the keyword `int` for the integer data type.
2. The `int` data type is used to store non-fractional numbers which include positive, negative, and zero values.
3. The range of `int` is **-2,147,483,648 to 2,147,483,647** and it occupies 2 or 4 bytes of memory, depending on the system you're using. For example,

```
int a = 5550; int b = -90, int c = 0; int d = -0.5; //invalid
```

4. We can perform *addition, subtraction, division, multiplication* operations on `int` data type.

Character (char):

1. We use the keyword `char` for the character data type.
2. It is used to store single-bit characters and occupies **1 byte** of memory.
3. You can store alphabets from **A-Z**(and **a-z**) and **0-9** digits using `char` datatype. For example,

```
char b = 'A'; char c = '0'; char d = 0; // ERROR
```

4. For `char` datatype, it is necessary to enclose the data within **single quotes**.
5. You can perform **addition** and **subtraction** operations on `char` datatype values.

Floating-point (float):

1. We use the keyword float for a floating-point data type.
2. The keyword float is used to store **decimal numbers**.
3. It occupies 4 bytes of memory and ranges from **1e-37 to 1e+37**.
4. For example, float a = 0.05; float b = -0.005. float c = 1; // it will become c = 1.000000 because of type-casting
5. We can perform addition, subtraction, division, and multiplication operations on float data type.

Double (double):

1. We use the keyword double for the double data type.
2. The double datatype is used to store **decimal numbers**.
3. It occupies **8 bytes** of memory and ranges from **1e-37 to 1e+37**.
4. Here is how we use it in code,

```
double a = 10.09; double b = -67.9;
```
5. The double datatype has more precision than float so double gives more accurate results as compared to float.
6. We can perform addition, subtraction, division, and multiplication operations on double data type.

Void (void):

1. This means no value.
2. This data type is mostly used when we define functions.
3. The void datatype is used when a function does not return any result.
4. It occupies **0 bytes** of memory.
5. We use the void keyword for void data type.

Below is a list of ranges along with the memory requirement and format specifiers on the **32-bit GCC compiler**.

Data Type	Size (bytes)	Range	
short int	2	-32,768 to 32,767	
unsigned short int	2	0 to 65,535	
unsigned int	4	0 to 4,294,967,295	
int	4	-2,147,483,648 to 2,147,483,647	
long int	4	-2,147,483,648 to 2,147,483,647	
unsigned long int	4	0 to 4,294,967,295	
long long int	8	-(2 ⁶³) to (2 ⁶³)-1	
unsigned long long int	8	0 to 18,446,744,073,709,551,615	
signed char	1	-128 to 127	
unsigned char	1	0 to 255	
float	4	1.2E-38 to 3.4E+38	
double	8	1.7E-308 to 1.7E+308	
long double	16	3.4E-4932 to 1.1E+4932	

Derived data type:

The data-types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types.

Array
structure
union
pointers

User-Defined DataTypes:

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
- Typedef defined DataType

Constants and variables

What is a constant in C?

The constants in C are the read-only variables whose values cannot be modified once they are declared in the C program. The type of constant can be an integer constant, a floating constant, a string constant, or a character constant. In C language, the **const** keyword is used to define the constants.

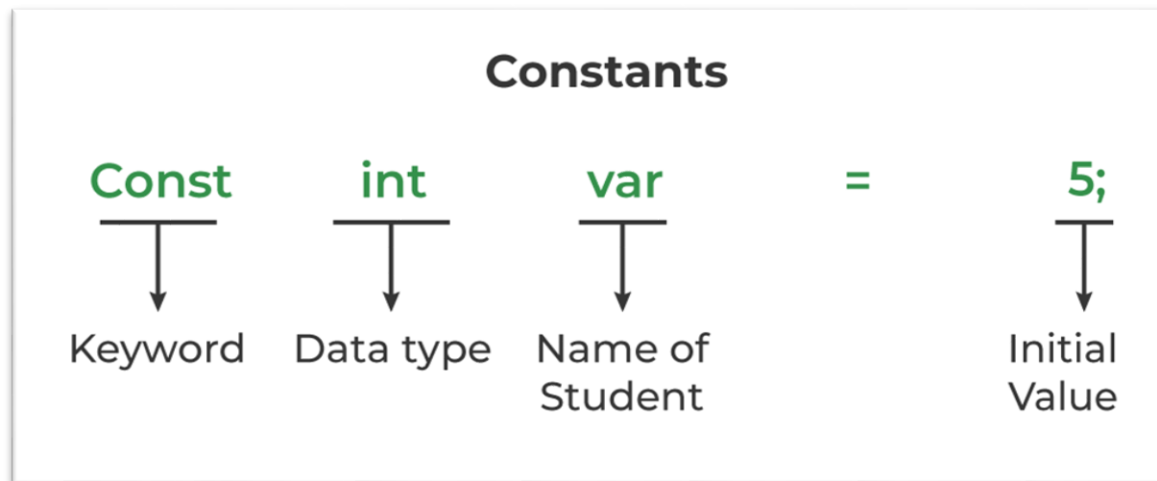
.

How to define a constant in C?

We define a constant in C language using the **const** keyword. Also known as a const type qualifier, the const keyword is placed at the start of the variable declaration to declare that variable as a constant.

Syntax to Define Constant

`const data_type var_name = value;`



- Integer Constants:** Whole number values, such as 42, -10, 0.
- Floating-Point Constants:** Decimal values with a fractional part, like 3.14, -0.5.
- Character Constants:** Single characters enclosed in single quotes, like 'A', '7', '@'.
- String Constants:** Sequences of characters enclosed in double quotes, like "Hello, World!".

C variables

What is a variable in C?

A **variable in C** is a memory location with some name that helps store some form of data and modified 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. The size of the variable depends upon the data type it stores.

C Variable Syntax

data_type variable_name = value;

There are 3 aspects of defining a variable:

- Variable Declaration
- Variable Definition
- 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;
```

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  
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:

A variable name must only contain alphabets, digits, and underscore.

A variable name must start with an alphabet or an underscore only. It cannot start with a digit.

No whitespace is allowed within the variable name.

A variable name must not be any reserved word or keyword.

Valid names	Invalid names
<code>_srujan, srujan_poojari, srujan812, srujan_812</code>	<p><code>srujan poojari</code> <i>It contains a whitespace in between srujan and poojari.</i></p> <p><code>13srujan</code> <i>It starts with a number so we cannot declare it as a variable.</i></p> <p><code>goto, for, switch</code> <i>We can't declare them as variables because they are keywords of C language</i></p>

C Variable Types

The C variables can be classified into the following types:

Local Variables

Global Variables

Static Variables

Automatic Variables

Register Variables

1. Local Variables in C

A **Local variable in C** is a variable that is declared inside a function or a block of code. Its scope is limited to the block or function in which it is declared.

```
#include <stdio.h>
int main()
{
    int x = 10; // local variable
    printf("%d", x);
}
```

2. Global Variables in C

A **Global variable in C** is a variable that is declared outside the function or a block of code. Its scope is the whole program i.e. we can access the [global variable](#) anywhere in the C program after it is declared.

Example of Global Variable in C.

```
// C program to demonstrate use of global variable
```

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

```
int x;
```

```
void function1() { printf("Function 1: %d\n", x); }
```

```
void function2() { printf("Function 2: %d\n", x); }
```

```
int main()
```

```
{
```

```
X=20;
```

```
function1();
```

```
X=50;
```

```
function2();
```

```
return 0;
```

```
}
```

```
#include<stdio.h>
#include<conio.h>
int x;
int function1()
{
printf("Function 1: %d\n", x);
}
int function2()
{
printf("Function 2: %d\n", x);
}
int main()
{

    x=20;
function1();
x=50;
function2();
return 0;
}
```

3.Static Variables in C

A [static variable in C](#) is a variable that is defined using the **static** keyword. It can be defined only once in a C program and its scope depends upon the region where it is declared (can be **global or local**).

The **default value** of static variables is **zero**.

Syntax of Static Variable in C

```
static data_type variable_name = initial_value;
```


• Automatic Variable in C

All the **local** variables are **automatic** variables **by default**. They are also known as auto variables.

Their scope is **local** and their lifetime is till the end of the **block**. If we need, we can use the **auto** keyword to define the auto variables.

The default value of the auto variables is a garbage value.

Syntax of Auto Variable in C

```
auto data_type variable_name;
```

Or

```
data_type variable_name;    (in local scope)
```

Register Variables in C

Register variables in C are those variables that are stored in the **CPU register** instead of the conventional storage place like RAM. Their scope is **local** and exists till the **end** of the **block** or a function.

These variables are declared using the [register](#) keyword.

The default value of register variables is a **garbage value**.

Syntax of Register Variables in C

```
register data_type variable_name = initial_value;
```

C program to demonstrate the definition of register

// variable

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

```
int main()
```

```
{
```

```
    //    register variable
```

```
    register int var = 22;
```

```
    printf("Value of Register Variable: %d\n", var);
```

```
    return 0;
```

```
}
```

Output

Value of Register Variable: 22

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, a variable, an array element or a constant.

Let's see an example:

a-b;

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

There are four types of expressions exist in C:

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

Each type of expression takes certain types of operands and uses a specific set of operators. Evaluation of a particular expression produces a specific value.

For example:

1. $x = 9/2 + a - b;$

The entire above line is a statement, not an expression. The portion after the equal is an expression.

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.

When an expression contains only integral operands, then it is known as pure integer expression when it contains only float operands, it is known as pure float expression, and when it contains both integral and float operands, it is known as mixed mode expression.

Evaluation of Arithmetic Expressions

The expressions are evaluated by performing one operation at a time.

When individual operations are performed, the following cases can be happened:

- 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 float 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 float, then the mixed arithmetic will be performed. In this case, the first operand is converted into a float operand, and then arithmetic is performed to produce the float 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.

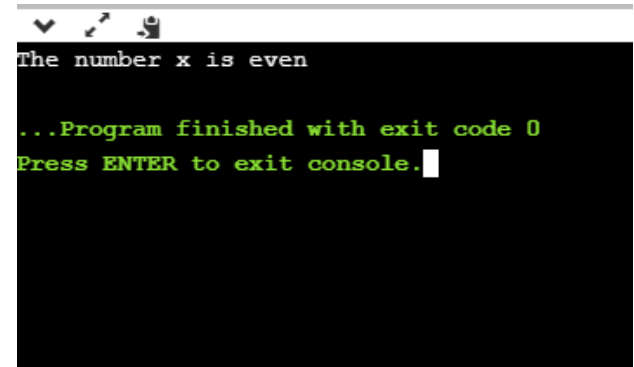
$$6*2 / (2+1 * 2/3 + 6) + 8 * (8/4)$$

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.


```
1.#include <stdio.h>
2. int main()
3.{
4.
5.  int x=4;
6.  if(x%2==0)
7.  {
8.      printf("The number x is even");
9.  }
10. else
11.  printf("The number x is not even");
12.  return 0;
13.}
```

Output

A screenshot of a terminal window with a black background and green text. The window has a title bar with standard Linux window controls (minimize, maximize, close). The output text is: "The number x is even", followed by "...Program finished with exit code 0" and "Press ENTER to exit console." with a white cursor at the end.

```
The number x is even
...Program finished with exit code 0
Press ENTER to exit console.
```

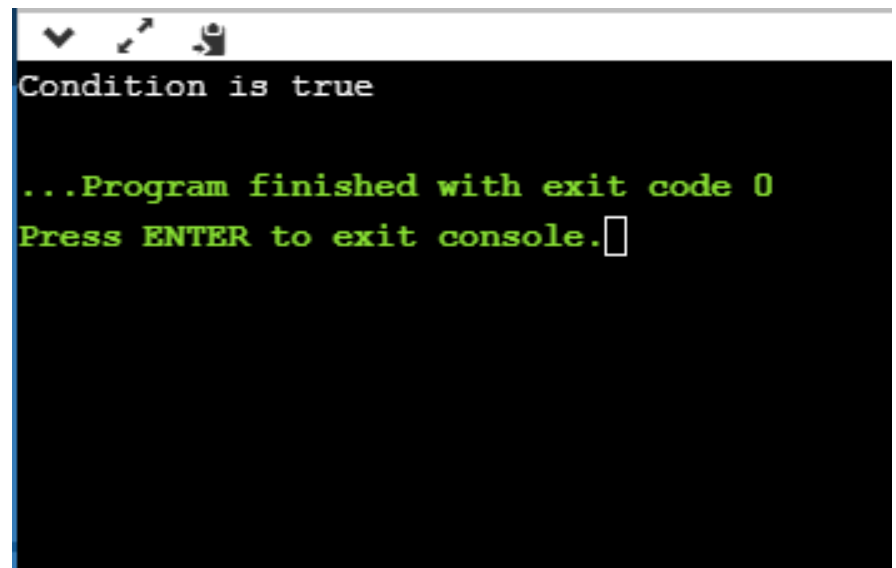
Logical Expressions

- 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.

Logical Expressions	Description
<code>(x > 4) && (x < 6)</code>	It is a test condition to check whether the x is greater than 4 and x is less than 6. The result of the condition is true only when both the conditions are true.
<code>x > 10 y < 11</code>	It is a test condition used to check whether x is greater than 10 or y is less than 11. The result of the test condition is true if either of the conditions holds true value.
<code>! (x > 10) && (y == 2)</code>	It is a test condition used to check whether x is not greater than 10 and y is equal to 2. The result of the condition is true if both the conditions are true.

Let's see a simple program of "&&" operator.

```
1. #include <stdio.h>
2. int main()
3. {
4.     int x = 4;
5.     int y = 10;
6.     if ( (x < 10) && (y > 5))
7.     {
8.         printf("Condition is true");
9.     }
10.    else
11.    printf("Condition is false");
12.    return 0;
13.}
```



```
Condition is true

...Program finished with exit code 0
Press ENTER to exit console.
```

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.

The Syntax of Conditional operator

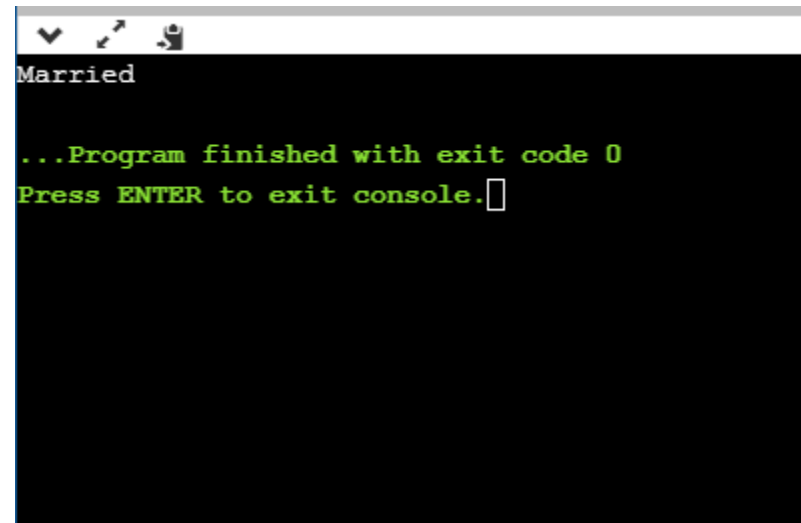
Suppose exp1, exp2 and exp3 are three expressions.

`exp1 ? exp2 : exp3`

The above expression is a conditional expression which is evaluated on the basis of the value of the exp1 expression. If the condition of the expression exp1 holds true, then the final conditional expression is represented by exp2 otherwise represented by exp3.

Let's understand through a simple example.

```
1. #include<stdio.h>
2. #include<conio.h>
3. int main()
4. {
5.     int age = 25;
6.     char status;
7.     status = (age>22) ? 'M': 'U';
8.     if(status == 'M')
9.         printf("Married");
10.    else
11.        printf("Unmarried");
12.    return 0;
13.}
```



The screenshot shows a console window with a dark background. The title bar is light gray and contains three icons: a checkmark, a magnifying glass, and a trash can. The console output is as follows:

```
Married

...Program finished with exit code 0
Press ENTER to exit console.
```

C Programming Operators

An operator is a symbol that operates on a value or a variable. For example: + is an operator to perform addition.

C has a wide range of operators to perform various operations.

C Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

Operator	Meaning of Operator
+	addition
-	subtraction
*	Multiplication
/	Division
%	remainder after division (modulo division)

Working of arithmetic operators

```
#include <stdio.h>
int main()
{
    int a = 9, b = 4, c;
    c = a+b;
    printf("a+b = %d \n", c);
    c = a-b;
    printf("a-b = %d \n", c);
    c = a*b;
    printf("a*b = %d \n", c);
    c = a/b;
    printf("a/b = %d \n", c);
    c = a%b;
    printf("Remainder when a divided by b = %d \n", c);
    return 0;
}
```

Output -

a+b = 13

a-b = 5

a*b = 36

a/b = 2

Remainder when a divided by b=1

Unary operators are the operators that perform operations on a single operand to produce a new value.

Types of unary operators

Types of unary operators are mentioned below:

Unary minus ($-$)

Increment ($++$)

Decrement ($--$)

NOT ($!$)

Addressof operator ($\&$)

Sizeof()

1. Unary Minus

The **minus operator (–)** changes the sign of its argument. A positive number becomes negative, and a negative number becomes positive.

```
int a = 10; int b = -a; // b = -10
```

Unary minus is different from the subtraction operator, as subtraction requires two operands.

C Program to illustrate the use of 'unary minus operator'

```
#include <stdio.h>
int main()
{
    int positive Integer = 100;
    int negative Integer = -positive Integer;
    printf("Positive Integer = %d, positive Integer);
    printf("Negative Integer = %d", negative Integer);
    return 0;
}
```

Output

Positive Integer = -100

Negative Integer = 100

2. Increment

The [increment operator \(++ \)](#) is used to increment the value of the variable by 1. The increment can be done in two ways:

2.1 *prefix increment*

In this method, The value of the operand will be altered *before* it is used.

Example:

```
int a = 1;  
int b = ++a; // b = 2
```

2.2 *postfix increment*

In this method, the operator follows the operand (e.g., a++). The value operand will be altered *after* it is used.

Example:

```
int a = 1;  
int b = a++; // b = 1  
int c = a;   // c = 2
```

```
// C program to illustrate increment
#include <stdio.h>
```

```
int main()
{
    int a = 5;
    int b = 5;
    printf("Pre-Incrementing a = %d\n", ++a);
    printf("Post-Incrementing b = %d", b++);
    return 0;
}
```

Output

```
Pre-Incrementing a = 6
Post-Incrementing b = 5
```

3. Decrement

The [decrement operator \(— \)](#) is used to decrement the value of the variable by

The decrement can be done in two ways:

prefix decrement

In this method, the operator precedes the operand (e.g., `--a`). The value of the operand will be altered *before* it is used.

Example:

```
int a = 1;  
int b = --a;    // b = 0
```

postfix decrement

In this method, the operator follows the operand (e.g., `a--`). The value of the operand will be altered *after* it is used.

Example:

```
int a = 1;  
int b = a--;    // b = 1  
int c = a;      // c = 0
```

```
// C program to illustrate decrement
#include <stdio.h>
```

```
int main()
{
    int a = 5;
    int b = 5;
    printf("Pre-Decrementing a = %d\n", --a);
    printf("Post-Decrementing b = %d", b--);
    return 0;
}
```

Output

```
Pre-Decrementing a =4
Post-Decrementing b = 5
```

4. NOT (!)

The logical NOT operator (!) is used to reverse the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.

Example:

If x is true, then !x is false If x is
false, then !x is true

```
// C program to illustrate NOT operator
```

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

```
int main()
```

```
{
```

```
    int a = 10;
```

```
    int b = 5;
```

```
    if (!(a > b))
```

```
        printf("b is greater than a\n");
```

```
    else
```

```
        printf("a is greater than b");
```

```
    return 0;
```

```
}
```

Output

a is greater than b

C Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

C Bitwise Operators

During computation, mathematical operations like: addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations. Visit [bitwise operator in C](#) to learn more.

Operators	Meaning of operators
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Example 3: Assignment Operators

// Working of assignment operators

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

```
int main()
```

```
{
```

```
    int a = 5, c;
```

```
    c = a;    // c is 5
```

```
    printf("c = %d\n", c);
```

```
    c += a;   // c is 10
```

```
    printf("c = %d\n", c);
```

```
    c -= a;   // c is 5
```

```
    printf("c = %d\n", c);
```

```
    c *= a;   // c is 25
```

```
    printf("c = %d\n", c);
```

```
    c /= a;   // c is 5
```

```
    printf("c = %d\n", c);
```

```
    c %= a;   // c = 0
```

```
    printf("c = %d\n", c);
```

```
    return 0;
```

```
}
```

Output

c = 5

c = 10

c = 5

c = 25

c = 5

c = 0