

## Chapter- 2

### IDE and Introduction to C

---

#### IDE:

- IDE Stands for “Integrated Development Environment”.
- IDE provides a window where all the c programs are written, compiled and linked to execute the program.
- As all these 3 processes are provided by a single environment i.e. IDE.
- Turbo C, Dev C++, Code Blocks etc. are the IDE’s and all these IDE’s are easily available on internet.
- We can download any of them for writing the c program.

#### Options available in IDE and its use:

##### 1. New Option:

- When we click on new option then a new window appears where we write the program.

##### 2. Save & Save as:

- After writing the program we need to save them with proper extension.

##### 3. Compile Option:

- In compile option compiler converts program into machine code (object code).
- If there is any mistake in the program then also compiler throws the error.

##### 4. Run Option:

- Run option is used to execute the program. If there is no error in the program the after clicking on run option we will get the output.

##### 5. Debug option:

- When a program does not run correctly, IDEs provide debugging tools that allow programmers to examine different variables and inspect their code in a deliberate way.
- Value of a variable can be seen using debug option while the program is under execution.

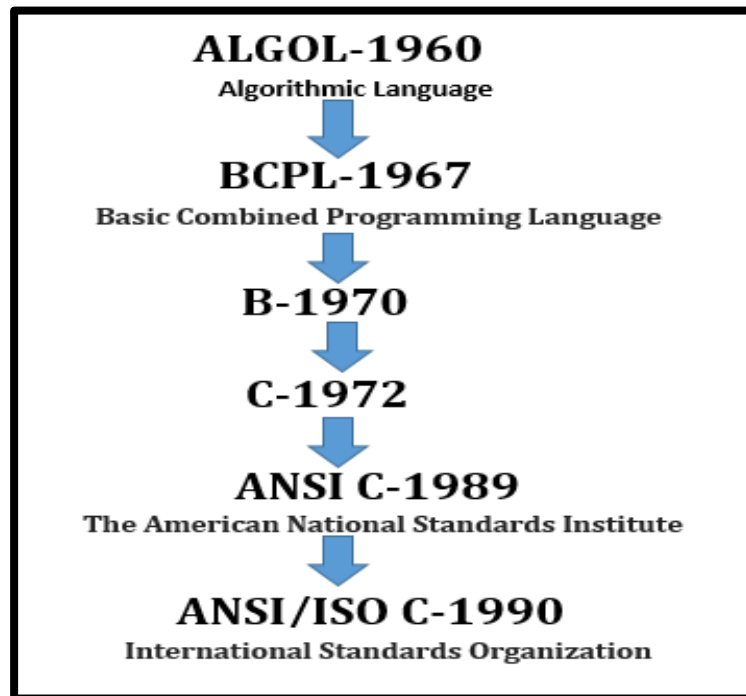
#### Introduction to C language:

- C is a procedural programming language that is extremely popular, simple and flexible. It was initially developed by “**Dennis Ritchie**” in the year **1972** at AT&T Bell Laboratories.
- It was mainly developed as a system programming language to write an operating system “UNIX”.
- C is a base for the programming. If you know 'C,' you can easily grasp the knowledge of the any other programming Language.
- Some examples of the use of C:
  - a) Operating Systems
  - b) Language Compilers
  - c) Assemblers
  - d) Text Editors and Data Bases

#### History of C Language/Standards of C Language:

- The base or father of programming languages is 'ALGOL' (Algorithmic Language). It was first introduced in 1960.
- In 1967, a new computer programming language was announced called as 'BCPL' which stands for Basic Combined Programming Language. **BCPL** was designed and developed by Martin Richards.
- In 1970 a new programming language called 'B' was introduced by Ken Thompson that contained multiple features of 'BCPL'.

- In 1972, a great computer scientist Dennis Ritchie created a new programming language called 'C' at the Bell Laboratories. It was created from 'ALGOL', 'BCPL' and 'B' programming languages. 'C' programming language contains all the features of these languages and many more additional concepts that make it unique from other languages.
- American National Standards Institute (ANSI) defined a commercial standard for 'C' language in 1989. Later, it was approved by the International Standards Organization (ISO) in 1990. 'C' programming language is also called as 'ANSI C'.



### C Library Functions/built-in /pre-defined:

- C Standard library functions or simply C Library functions are inbuilt functions in C programming.
- The prototype i.e. function declaration of these functions are present in their respective header files. To use these functions we need to include the header file in our program.
- Function definitions of these functions are present in library file.
- Extension of header file is .h and library file is .lib
- Various built in function are printf( ), scanf(), getch(), getche(), getchar(), puts(), gets(), sqrt(), pow(), abs(), strcat(), strlwr(),strupr(),strrev(),strlen(),sleep(),system(),malloc(),exit(0) etc.
- And all the above built in functions are defined in some header files and in order to use any function we need to include the respective header file in our program.

Header File	Functions
stdio.h	printf() , scanf(),puts(),gets() etc.
string.h	strcat(),strlwr(),strupr(),strrev(),strlen() etc.
math.h	sqrt(),pow(),abs(),round() etc.
ctype.h	isdigit(),islower(),isupper() etc.

### How to Use Library Functions:

- **#include** is a way of including files in the program and is mostly written at the beginning of any C
- The #include is a preprocessor directive and it is used to paste code of given file into current file.

- The **C Preprocessor** is not a part of the compiler, but is a separate step in the compilation process. In simple terms, a C Preprocessor is just a text substitution tool and it instructs the compiler to do required pre-processing before the actual compilation.
- All preprocessor commands begin with a hash symbol (#).

#### Syntax to use header file:

```
#include <file_name.h>
```

#### Example:

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

#### Structure of c program:

```
#include<header file>
int main() //main function
{
    Statements-1;
    Statements-2;
    return 0;
}
```

**Video Link:** <https://youtu.be/3ElKF6oWWqE>

#### #include<header file>

- As we know that before using any variable we declare it first in our program and then only we use that variable.
- Similarly, before using any predefined function, we will have to declare that function first in our program and as we know declaration i.e. prototype of the functions are defined in the header file only. so when we write the line #include<header file> then all the function which is declared in respective header file are copied into our program and then we use those functions.
- For eg. stdio.h contains declaration of printf(),scanf() etc. so when we write #include<stdio.h> then declaration of printf(),scanf() etc. are copied into the program at the beginning. Then we use these function efficiently.

#### int main()

- main() function is mandatory function to write in any program.
- Execution of any program start from the main function only.
- It is the entry point for a program. Compiler starts the execution from main function only.
- **int** is the return type of main function that means main function will return an integer value at the end of the program.

```
{ }
```

- { } curly brackets defines scope of a block.
- All the program should be written inside the curly brackets "{ }" only.

#### return 0;

- return 0; statement causes the termination of main function.
- 0 value indicates successful execution of the program.
- statement written after return 0; never get executed.

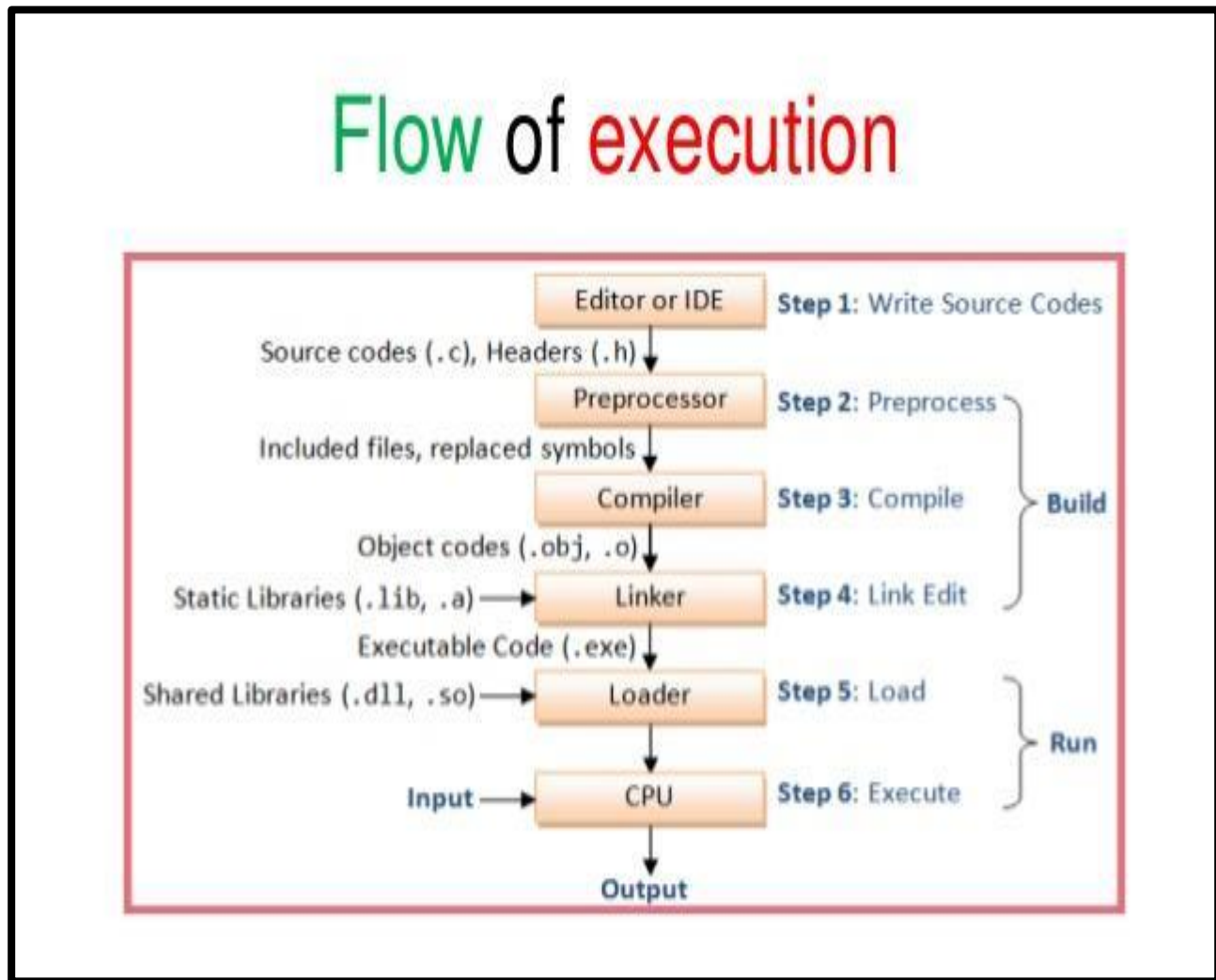
### Program to display a message on output screen using c language:

```
#include<stdio.h>
int main()
{
printf(“ Good Morning”);
return 0;
}
```

#### Output:

Good Morning

### Flow of Execution of a program



1. C program (source code) is sent to preprocessor first. The preprocessor is responsible to convert preprocessor directives into their respective values. The preprocessor generates an expanded source code.
2. Expanded source code is sent to compiler which compiles the code and converts it into object code i.e. machine code.

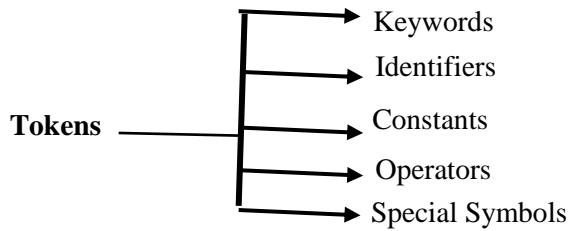
3. The object code is sent to linker which links it to the library such as header files. Then it is converted into executable code. A simple.exe file is generated. For Eg. if we have use printf() function, then definition of printf() is present in library file, so linker will link the calling function with its definition.
4. The executable code is sent to loader which loads it into memory and then it is executed. After execution, output is sent to console.

Steps to learn English language	Steps to learn C Language
<b>Alphabets</b>	<b>Character set</b> Eg: Letters from A to Z, a to z Digits from 0 to 9 Symbols such as +, -, *, <, >, etc.
<b>Words</b>	<b>Tokens</b> Keywords: do, while, for, if, int, float, etc. Variables: sum, area, first, marks, name, etc. Constants: 10, 10.5, -5.9, 'a', "MCU", etc.
<b>Sentences</b>	<b>Statements /Instructions</b> $si = p * t * r / 100;$ Total=A + B + C
<b>Set of Sentence/Paragraph</b>	<b>Program</b> <pre>#include&lt;stdio.h&gt; int main { int si, p = 1000, t = 2, r = 5; si = p*t*r/100; printf("simple interest = %d",si); return 0; }</pre>

### Tokens:

- Tokens in C is the most important element to be used in creating a program in C.
- We can define the token as the smallest individual element in C. For example, we cannot create a sentence without using words; similarly, we cannot create a program in C without using tokens in C.
- Therefore, we can say that tokens in C is the building block or the basic component for creating a program in C language.

The tokens in C language are broadly classified as shown below:



### 1. Keywords:

- The words that have **pre-defined** meaning for C compiler are called keywords. Since they are reserved for specific purpose in C language, they are also called **reserved words**.
- The keywords have some specific purpose in C language and hence cannot be used as user defined names like variable names or function names. The different keywords of C language are shown below.

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

**Video Link:** <https://youtu.be/bNcmfPM2XZ0>

### 2. Identifiers:

- C identifiers represent the name in the C program, for example, variables, functions, arrays, structures, unions, labels, etc.
- An identifier can be composed of letters such as uppercase, lowercase letters, underscore, digits, but the starting letter should be either an alphabet or an underscore.
- Identifier is a synonym for name.

#### Rules for constructing C identifiers

- The first character of an identifier should be either an alphabet or an underscore “\_”, and then it can be followed by any of the character, digit, or underscore.
- It should not begin with any numerical digit.
- In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
- Commas or blank spaces cannot be used within an identifier.
- Keywords cannot be represented as an identifier.
- The length of the identifiers is not fixed.
- Identifiers should be written in such a way that it is meaningful, short, and easy to read.

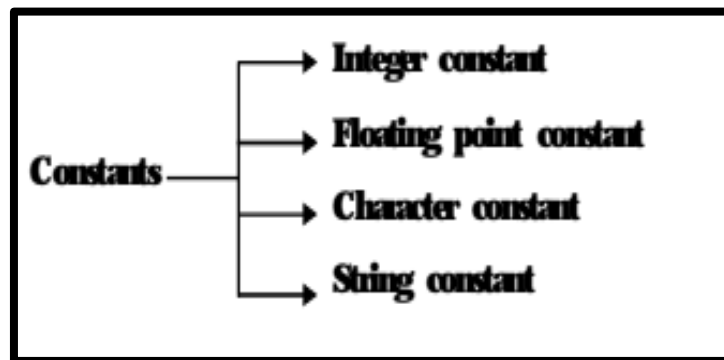
Identifiers	valid/Invalid	Invalidity Reason
abc	valid	
12a	<b>invalid</b>	Cannot start with digit
a12	valid	
_12	valid	

first	valid	
First_1	valid	
if	<b>invalid</b>	It is keyword
_	valid	Underscore is allowed
int	<b>invalid</b>	It is a keyword
Sum-of-digit	<b>invalid</b>	Hyphen(-) is not allowed
First second	<b>invalid</b>	Space is not allowed
int1	valid	
@abc	<b>Invalid</b>	Special symbol not allowed

**Video Link:** <https://youtu.be/oiryKOPmZvk>

### 3. Constants:

A constant is a data item which will not be changed during the execution of a program.



#### 1. Integer constant:

An integer constant is a whole number without any decimal point.

E.g.: 10, 20, -10, 1000 etc.

#### 2. Floating point constant:

A Floating point constant is a number with decimal point.

e.g.: 10.2, 123.34, 1.3 etc.

#### 3. Character constants:

A character constant is a single alphabet, a single digit or a single special symbol enclosed within single quotes.

E.g.: 'a', 'Z', 'B', '+', '@' etc.

#### Escape sequence characters:

- An escape sequence character begins with a '\' and is followed by one character.
- A backslash along with one character gives rise to special print effect by changing (escaping) the meaning of the character. Since an escape sequence character starts with backslash, they are also called **backslash constants**.

**The commonly used escape sequences characters are shown below:**

Character	Escape sequence character	Use
New Line	\n	Cursor moves to the next line.
Horizontal tab	\t	Cursor moves towards right by 8 positions.
Bell	\a	Beep sound.
Slash	\\	To display slash
Double quotes	\"	To display double quote

**Note:** The backslash characters such as '\a', '\n', '\t', etc. are non-printable characters.

### **String constant:**

A sequence of characters (i.e. collection of characters) enclosed within a pair of double quotes is called a string constant.

e.g.: "sahyog", "hello", "morning" etc.

### **Variables:**

- When we want to store any information (data) on our computer/laptop, we store it in the computer's memory space. Instead of remembering the complex address of that memory space where we have stored our data, our operating system provides us with an option to create folders, name them, so that it becomes easier for us to find it and access it.
- Similarly, in C language, when we want to use some data value in our program, we can store it in a memory space.
- A **variable** is a name of the memory location. It is used to store data. Its value can be changed.
- A **variable** in C language must be given a type, which defines what type of data the variable will hold.

### **Rules for defining variables:**

- A variable can have alphabets, digits, and underscore.
- A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- No whitespace is allowed within the variable name.
- A variable name **must not be** any reserved word or keyword, e.g. int, float, etc.

### **Syntax to declare a variable:**

Data\_type Variable\_name;

#### **Example:**

```
int n1;  
float num;  
char ch;
```

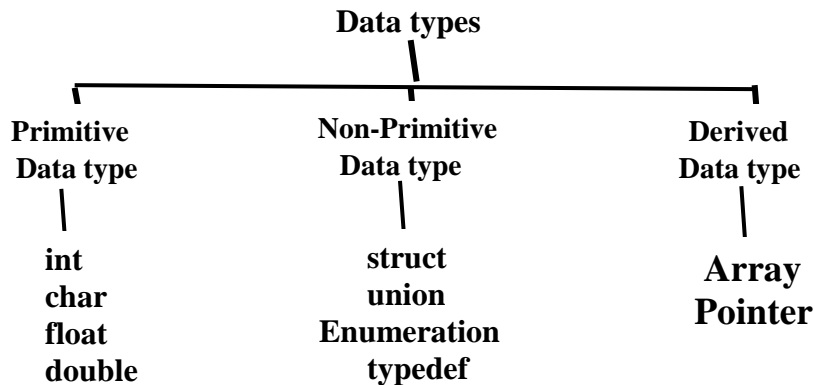
### **Data type:**

- Data types specify the type of data. Data type determines the type of data a variable will hold.
- C language has some predefined set of data types to handle various kinds of data that we can use in our program. These data types have different storage capacities.



## Type of data types:

1. Primitive data type (Built-in)
2. Non-primitive data type(user defined data type)
3. Derived data type



**Video Link:** <https://youtu.be/d5oKuXehp0o>

## Primitive data type:

### 1. int

- It is a keyword which is used to define integer numbers. Normally they are associated with the variables to store signed integer values in memory locations. That is, using this data type both positive and negative numbers can be stored in the memory.
- 10,-20, 501, 1003 etc. are integer values.

### 2. char

- Character data type allows a variable to store only one character.
- “char” keyword is used to refer character data type.
- For example, ‘A’ can be stored using char data type. You can’t store more than one character using char data type.
- We can also store numerical value too in char type but if will store numerical value then “ASCII” value will be printed.

- **Note:**
- Every character has a corresponding ASCII value to it ranging from -128 to 127.
- Numbers as a character has their corresponding ASCII values too. For example, ‘1’ as char has ASCII value 49, ‘A’ has ASCII value 65.

### 3. float

- It is a keyword which is used to define floating point numbers. The floating point numbers are also called real numbers. Normally they are associated with the variables (also called identifiers) to store floating point numbers in memory locations.
- That is, using this data type both positive and negative floating point numbers can be stored in the memory.
- Eg: 1.23, 2234.4532 etc.

#### 4. Double:

- Double is also used to define floating point numbers.
- But double data type accepts 15 digit after decimal whereas float accepts 6 digits.
- Eg: 523.45353

#### Modifiers in C Language:

1. Size Modifier
2. Sign Modifier

##### 1. Size Modifier:

- Size Modifiers are prefixed with basic data types to modify (either increase or decrease) the amount of storage space allocated to a variable.
- Short, long, long long are the size modifiers.
- For example, storage space for int data type is 4 byte. We can increase the range by using long int which is 8 byte. We can decrease the range by using short int which is 2 byte.
- short is only used with int type.
- long can be used with int and double.
- Any size modifier cannot be used with float and char data type.
- long long can not be used with float and double.
- Long long can be used with int type only.

##### 2. Sign Modifier:

- Sign modifiers decides whether the variable will hold positive or negative number.
- signed & unsigned are the sign modifiers.
- In signed type we can store both positive as well as negative value but in unsigned type we can store only positive value.
- By default the data type is signed only.
- Sign modifiers are used with int and char only.
- Sign modifiers cannot be used with float and double.

#### Size & Range of data type:

Data type	Space	Range	Format specifier
int	4 byte	-2147483648 to 2147483647	%d
char	1 byte	-128 to 127	%c
float	4 byte	6 digits	%f
double	8 byte	15 digit	%lf
short int	2 byte	-32768 to 32767	%hi
long int	4 byte	-2147483648 to 2147483647	%ld
long long int	8 byte	-9223372036854775808 to 9223372036854775807	%lld
signed short int	2 byte	-32768 to 32767	%hi

unsigned short int	2 byte	0 to 65535	%hu
Signed int	4 byte	-2147483648 to 2147483647	%d
unsigned int	4 byte	0 to 4294967295	%u
unsigned long long int	8 byte	0 to 18446744073709551615	%llu
signed long long int	8 byte	-9223372036854775808 to 9223372036854775807	%lld
Signed char	1 byte	-128 to 127	%c
unsigned char	1 byte	0 to 255	%hc

**Note:**

- By default int and char data type is of signed type that means we can store positive as well as negative value in it.
- int and signed int are same data type.
- short int and signed short int are same.
- long long int and signed long long int are same.
- char and signed char are same.

**Format specifier:**

The format specifier is used during input and output. It is a way to tell the compiler what type of data is in a variable during taking input using scanf() or printing using printf().

Some examples are %c, %d, %f, etc.

- Format specifier for hexadecimal - %x or %X
- Format specifier octal - %o
- Format specifier decimal/integer - %d

**Note:** Hexadecimal, Octal and Decimal Values comes under integer constant so we can store value of these types into int datatype.

Eg: int a=0XA is valid (Hexadecimal value)

int b=010 is valid (Octal value)

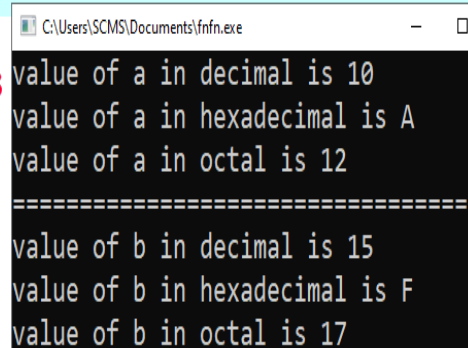
int c=102 is valid (Decimal value)

Hexadecimal	Octal	Decimal
Hexadecimal values comes under integer constant.	Octal values comes under integer constant.	Decimal values comes under integer constant.
An hexadecimal integer constant can be any combination of digits from '0' to '9' along with the letters 'A' to 'F' or 'a' to 'f'. This constant has to be preceded by <b>0X</b> OR <b>0x</b> .	An octal integer constant can be any combination of digits from '0' to '7' With a prefix <b>0</b> (digit zero).	A decimal integer constant can be any combination of digits from '0' to '9'.

EG: 0x18A, 0XB, 0XFFF	EG: 01612, 034567, -0765	EG: 100, -7, 989
%x or %X	%o	%d

## 1. Program for Hexadecimal Value

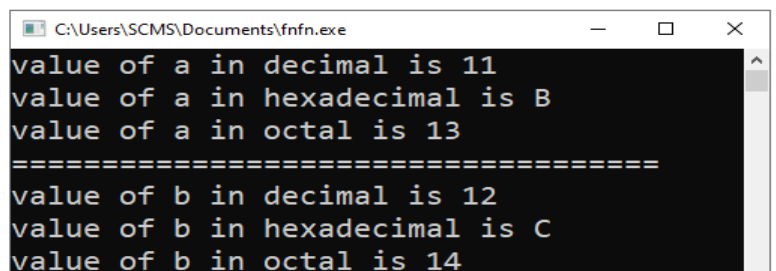
```
#include<stdio.h>
int main()
{
    int a=0XA;
    printf("value of a in decimal is %d",a);
    printf("\nvalue of a in hexadecimal is %X",a);
    printf("\nvalue of a in octal is %o",a);
    printf("\n===== \n");
    int b=15;
    printf("value of b in decimal is %d",b);
    printf("\nvalue of b in hexadecimal is %X",b);
    printf("\nvalue of b in octal is %o",b);
    return 0;
}
```



```
C:\Users\SCMS\Documents\fnfn.exe
value of a in decimal is 10
value of a in hexadecimal is A
value of a in octal is 12
=====
value of b in decimal is 15
value of b in hexadecimal is F
value of b in octal is 17
```

## 2. Program for Octal Value

```
#include<stdio.h>
int main()
{
    int a=013;
    printf("value of a in decimal is %d",a);
    printf("\nvalue of a in hexadecimal is %X",a);
    printf("\nvalue of a in octal is %o",a);
    printf("\n===== \n");
    int b=12;
    printf("value of b in decimal is %d",b);
    printf("\nvalue of b in hexadecimal is %X",b);
    printf("\nvalue of b in octal is %o",b);
    return 0;
}
```

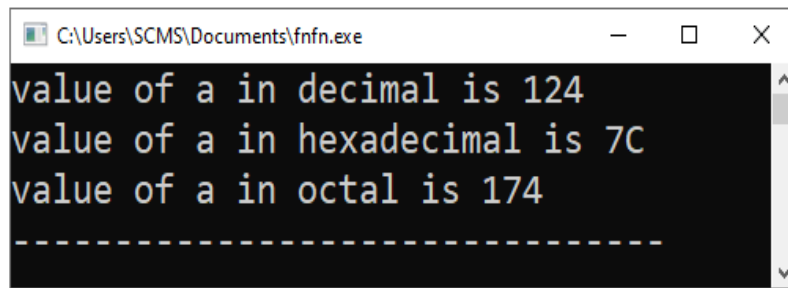


```
C:\Users\SCMS\Documents\fnfn.exe
value of a in decimal is 11
value of a in hexadecimal is B
value of a in octal is 13
=====
value of b in decimal is 12
value of b in hexadecimal is C
value of b in octal is 14
```

### 3. Program for decimal Value

```
#include<stdio.h>
int main()
{
    int a=124;
    printf("value of a in decimal is %d",a);
    printf("\nvalue of a in hexadecimal is %X",a);
    printf("\nvalue of a in octal is %o",a);

    return 0;
}
```



```
C:\Users\SCMS\Documents\fnfn.exe
value of a in decimal is 124
value of a in hexadecimal is 7C
value of a in octal is 174
-----
```

### Program to get range of data type:

- In c language many macros are available with some fixed values in it. For e.g. INT\_MIN is a macro which is having the minimum value for int type and INT\_MAX is having the maximum value for int type and SHRT\_MIN is having the minimum value for short type etc. and all these macros are available in limits.h header file.
- It is not possible to remember the range of all the data type, then theses macros can be used wherever range of any type is required.

### Program

```
#include <stdio.h>
#include <limits.h>
int main()
{
    printf("The minimum value of SIGNED CHAR = %d\n", SCHAR_MIN);
    printf("The maximum value of SIGNED CHAR = %d\n", SCHAR_MAX);
    printf("The maximum value of UNSIGNED CHAR = %d\n", UCHAR_MAX);

    printf("The minimum value of SHORT INT = %d\n", SHRT_MIN);
    printf("The maximum value of SHORT INT = %d\n", SHRT_MAX);

    printf("The minimum value of INT = %d\n", INT_MIN);
```

```

printf("The maximum value of INT = %d\n", INT_MAX);

printf("The minimum value of CHAR = %d\n", CHAR_MIN);
printf("The maximum value of CHAR = %d\n", CHAR_MAX);

printf("The minimum value of LONG = %ld\n", LONG_MIN);
printf("The maximum value of LONG = %ld\n", LONG_MAX);
return 0;
}

```

### Output:

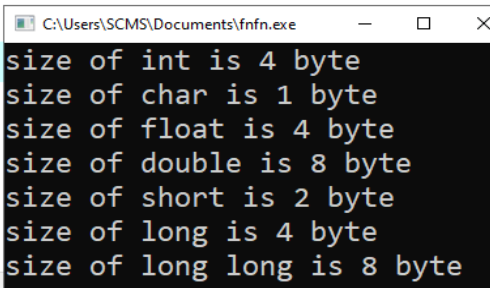
The minimum value of SIGNED CHAR = -128  
 The maximum value of SIGNED CHAR = 127  
 The maximum value of UNSIGNED CHAR = 255  
 The minimum value of SHORT INT = -32768  
 The maximum value of SHORT INT = 32767  
 The minimum value of INT = -2147483648  
 The maximum value of INT = 2147483647  
 The minimum value of CHAR = -128  
 The maximum value of CHAR = 127  
 The minimum value of LONG = -2147483648  
 The maximum value of LONG = 2147483647

### Program to display size of data type.

```

#include<stdio.h>
int main()
{
    printf("size of int is %d byte",sizeof(int));
    printf("\nsize of char is %d byte",sizeof(char));
    printf("\nsize of float is %d byte",sizeof(float));
    printf("\nsize of double is %d byte",sizeof(double));
    printf("\nsize of short is %d byte",sizeof(short));
    printf("\nsize of long is %d byte",sizeof(long));
    printf("\nsize of long long is %d byte",sizeof(long long));
    return 0;
}

```



C:\Users\SCMS\Documents\fnfn.exe  
 size of int is 4 byte  
 size of char is 1 byte  
 size of float is 4 byte  
 size of double is 8 byte  
 size of short is 2 byte  
 size of long is 4 byte  
 size of long long is 8 byte

**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.
  1. Arithmetic operators
  2. Assignment operators
  3. Increment / Decrement operators
  4. Relational operators
  5. Logical operators
  6. Conditional operator
  7. Bitwise operators
  8. Special operators

**1. Arithmetic Operator:**

The operators that are used to perform arithmetic operations such as Addition, subtraction, multiplication etc., are called arithmetic operators.

**The meaning of all the operators along with examples is shown below:**

Arithmetic operator	Description	Example	Result	Priority	Associativity
+	Used for addition	10 + 20	30	2	Left to Right
-	Used for subtraction	50-10	40	2	Left to Right
*	Used for multiplication	10 * 20	200	1	Left to Right
/	Used for division and gives quotient	10/5	2	1	Left to Right
%	Used to get remainder. It is read as modulus operator or mod	4%2	0	1	Left to Right

Note: Modulus operator denoted by % is used only for integer values and not for Floating point numbers. This operator returns the remainder after division.

**Program to perform arithmetic operations.**

```

#include<stdio.h>
int main()
{
    int first=20, second=2, add, subtract, multiply, divide, mod;

    add = first + second;
    subtract = first - second;
    multiply = first * second;
    divide = first / second;
    mod=first % second;

    printf("Addition = %d\n", add);
    printf("Subtraction = %d\n", subtract);
    printf("Multiplication = %d\n", multiply);
    printf("Division = %d\n", divide);
    printf("Modulus is = %d\n", mod);
    return 0;
}

```

## 2. Assignment Operator:

- An operator which is used to copy the data or result of an expression into a memory location (which is identified by a variable name), is called an assignment operator.
- Copying or storing into a memory location is called assigning and hence the name. The assignment operator is denoted by '=' sign.
- Assignment always happens from **right to left**.

### Syntax (General rule) for assignment:

Variable = expression/value;

### For Example:

- **a = b;** /\* Store the value of b into a \*/
- **area = L \* B;** /\* Compute the product and store in variable area \*/
- **pi = 3.1416;** /\* Store the number 3.1416 using the variable pi \*/

### Normal mistakes during typing the program:

- **a == b;** /\* Error : "==" is relational operator. This can be used only to compare. Not for copying \*/
- **a = b \* 10** /\* Error: No semicolon at the end \*/
- **10 + b = c;** /\*Error: Expression not allowed on LHS of assignment operator



### Shorthand assignment operators :

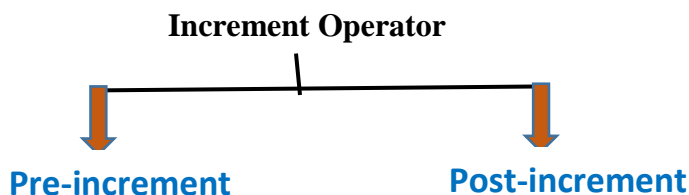
Shorthand assignment operators such as +=, -=, \*=, /= etc., can be used to assign values. The table below shows shorthand operators, shorthand statements and the meaning associated with them :

Shorthand operator	Shorthand statement	Meaning	Explanation	Associativity
+=	A+=2	A=A+2	Perform A + 2 and store the result in a	Right to Left
-=	B-=C	B=B-C	Perform B - C and store the result in B	Right to Left
*=	A*=3	A=A*3	Perform A *3 and store the result in A	Right to Left
/=	A/=B	A=A/B	Perform A /B and store the result in A	Right to Left
%=	A%=B	A=A%B	Perform A %B and store the result in A	Right to Left

### 3. Increment and Decrement operators:

#### Increment operator:

- ‘++’ is an increment operator. This is a unary operator. It increments the value of the operand by one.
- The increment operator is classified into two categories as shown below:



#### a. Pre-increment Operator:

- If the increment operator ++ is placed before (pre) the operand, then the Operator is called pre-increment.
- As the name indicates, pre-increment means increment before (pre) the operand value is used. So, the operand value is incremented by 1 first, and then this incremented value is used.
- Eg: ++a, ++b etc.

#### Program to show the use of pre-increment operator.

```
#include<stdio.h>
int main()
{
int a = 20,b;
b=++a;
printf("a=%d\n",a);
printf("b=%d\n",b);
return 0;
}
```

#### Output:

```
a=21
b=21
```

### b. Post-increment Operator:

- If the increment operator ++ is placed after (post) the operand, then the operator is called post-increment. As the name indicates, post-increment means increment
- after (post) the operand value is used. So, operand value is used first and then the operand value is incremented by 1.
- Eg: a++, b++ etc.

### Program to show the use of post-increment operator.

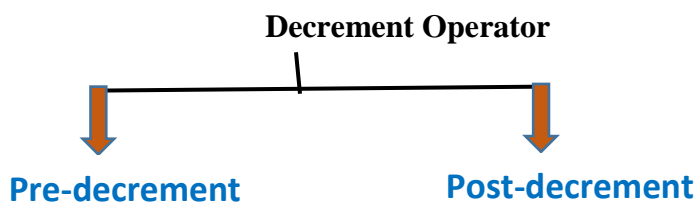
```
#include<stdio.h>
int main()
{
int a = 20;
int b;
b=a++;
printf("a=%d\n",a);
printf("b=%d\n",b);
return 0;
}
```

### Output:

```
a=21
b=20
```

### Decrement operator:

- ‘--’ is a decrement operator. This is an unary operator. It decrements the value of the operand by one.
- The decrement operator is classified into two categories as shown below:



### a. Pre-decrement Operator:

- If the decrement operator - - is placed before (pre) the operand, then the Operator is called pre-decrement.
- As the name indicates, pre-decrement means decrement before (pre) the operand value is used. So, the operand value is decremented by 1 first, and then this decremented value is used.
- Eg: --a, --b etc.

### Program to show the use of pre-decrement operator.

```
#include<stdio.h>
int main()
{
int a = 20;
int b;
b=--a
```

```
printf("a=%d\n",a);
printf("b=%d\n",b);
return 0;
}
```

**Output:**  
**a=19**  
**b=19**

#### b. Post-increment Operator:

- If the decrement operator -- is placed after (post) the operand, then the operator is called post-decrement.
- As the name indicates, post-decrement means decrement after (post) the operand value is used. So, operand value is used first and then the operand value is decremented by 1.
- Eg: a--, b-- etc.

#### Program to show the use of post-increment operator.

```
#include<stdio.h>
int main()
{
int a = 20;
int b;
b=a--;
printf("a=%d\n",a);
printf("b=%d\n",b);
return 0;
}
```

**Output:**  
**a=19**  
**b=20**

#### 4. Relational operators:

- The relational operators, also called comparison operators, are used to compare two operands. They are used to find the relationship between two operands and hence are called relational operators.
- The two operands may be constants, variables or expressions. The relationship between these two operands results in true (1) or false(0).

The relational operators and the meaning associated with them are shown in the following table:

Operator	Description	Example	Priority	Associativity
<	Less than	10 < 20	1	Left to right
<=	Less than or equal	12 <= 18	1	Left to right
>	Greater than	15>10	1	Left to right
>=	Greater than or equal	15>=3	1	Left to right
==	Equal to	10 ==9	2	Left to right
!=	Not equal to	5 !=2	2	Left to right

#### Program to show the use of relational operator.

```

#include<stdio.h>
int main()
{
    int a=10,b=20,r1,r2,r3,r4;

    r1= (a<b);
    printf("a<b = %d",r1);

    r2=(a>b);
    printf("\na>b = %d",r2);

    r3=(a==b);
    printf("\na==b = %d",r3);

    r4=(a!=b);
    printf("\na!=b = %d",r4);

    return 0;
}

```

```

C:\Users\SCMS\Documents\fnfn.exe
a<b = 1
a>b = 0
a==b = 0
a!=b = 1
-----

```

## 5. Logical Operator:

- As we have logic gates such as AND, OR and NOT whose output is 1 or 0, we also have logical operators.
- Logical Operators are used to combine 2 more relational expressions.
- After evaluation, expression consisting of logical operators results in either true (1) or false (0) and hence such expressions are called logical expressions.

Logical operators and the meaning associated with them are shown in the following table:

Operator	Description	Example	Priority	Associativity
&&	Logical And. If all the conditions are true then result will be true	(10<20) && (23>5)=1 (21<43) &&(5==5)=0	2	Left to right
	Logical OR. If any one of the o condition is true then result will be true	(18>=17)    (15<10)=1 (10!=10) &&(5>6)=0	3	Left to right
!	Logical NOT. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false and if a condition is false then Logical NOT operator will make it false true.	!(10>23)=1 !( (21<43)&&(5==5))=0	1	Left to right

- a. Logical AND:** The result of logical ‘AND’ operator denoted by && is true if and only if both the operands are evaluated to true. If one of the operands is evaluated to false.

Operand 1	Operand 2	Result
True(1)	True(1)	True(1)
True(1)	False(0)	False(0)
False(0)	True(1)	False(0)
False(0)	False(0)	False(0)

- b. Logical OR:** The result of logical ‘OR’ operator denoted by || is true if and only if at least one of the operands is evaluated to true. If both the operands are evaluated to false, the result is false.

Operand 1	Operand 2	Result
True(1)	True(1)	True(1)
True(1)	False(0)	True(1)
False(0)	True(1)	True(1)
False(0)	False(0)	False(0)

- c. Logical Not:** The logical ‘NOT’ denoted by ! can be true or false. The result is true if the operand is false and the result is false if the operand is true.

Operand 1	Result
True(1)	False(0)
False(0)	True(1)

#### 1. Program to show the use of logical operator

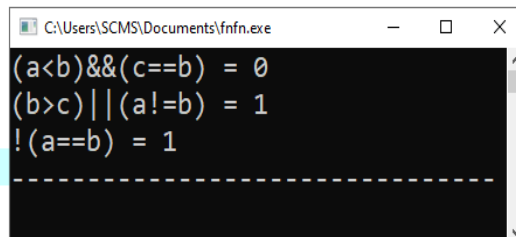
```
#include<stdio.h>
int main()
{
    int a=10,b=20,c=30,d=23,r1,r2,r3;

    r1= (a<b)&&(c==b);
    printf("(a<b)&&(c==b) = %d",r1);

    r2=(b>c)|| (a!=b);
    printf("\n(b>c)|| (a!=b) = %d",r2);

    r3!=(a==b);
    printf("\n!(a==b) = %d",r3);

    return 0;
}
```



```
C:\Users\SCMS\Documents\fnfn.exe
(a<b)&&(c==b) = 0
(b>c)|| (a!=b) = 1
!(a==b) = 1
-----
```

#### 6. Conditional Operator:

- The conditional operator is also known as a ternary operator.
- As conditional operator works on three operands, so it is also known as the ternary operator.
- It is represented by two symbols, i.e. '?' and ':'.
- The behavior of the conditional operator is similar to the 'if-else' statement.

##### Syntax1:

Expression1? expression2: expression3;

## Syntax2:

Variable=Expression1? expression2: expression3;

### Meaning of the above syntax.

- In the above syntax, the expression1 is a Boolean condition that can be either true or false(Yes/No) value.
- If the expression1 results into a true value, then the expression2 will execute.
- If the expression1 returns false value then the expression3 will execute.

### Example1:

Suppose, A=10, B=20

(A<B) ? printf("A is the smallest number") : printf("B is the smallest number");

### Output:

A is the smallest number

### Example2:

Suppose, A=30, B=12

C= (A<B) ? A : B;

printf("Smallest no is %d",C);

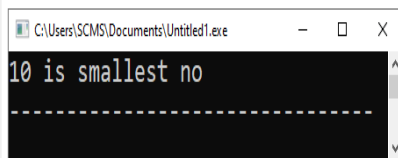
### Output:

Smallest no is 12

## Program to display smallest number between 2 numbers using conditional operator:

```
#include<stdio.h>
int main()
{
    int a=10,b=20;

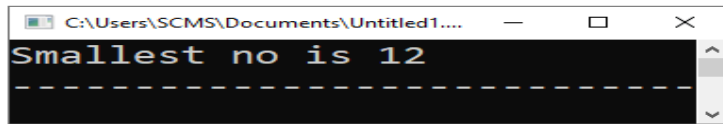
    (a<b) ? printf("%d is smallest no",a):printf("%d is smallest no",b);
    return 0;
}
```



## Program to display smallest number between 2 numbers using conditional operator:

```
#include<stdio.h>
int main()
{
    int a=30,b=12,c;

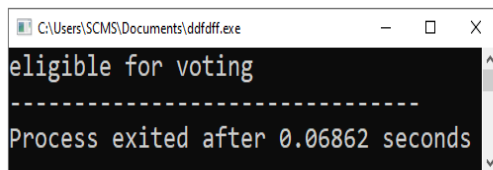
    c= (a<b) ? a : b;
    printf("Smallest no is %d",c);
    return 0;
}
```



**Program to check eligibility for voting using conditional operator:**

```
#include <stdio.h>
int main()
{
    int age=23;

    (age>=18)? printf("eligible for voting") : printf("not eligible for voting");
    return 0;
}
```



## 7. Bit-wise Operator:

- In arithmetic-logic unit (which is within the CPU), mathematical operations like: addition, subtraction, multiplication and division are done in bit-level.
- To perform bit-level operations in C programming, bitwise operators are used.

**The bit-wise operators and the meaning associated with them are shown in the following table:**

Operator	Description	Example	Priority	Associativity
&	Bitwise AND.	12 & 15	3	Left to right
	Bitwise OR	5   2	5	Left to right
^	Bitwise XOR	11 ^ 10	4	Left to right
~	Bitwise NOT	~10	1	Left to right
<<	Bitwise Left Shift	10<<1	2	Left to right
>>	Bitwise Right Shift	12>>1	2	Left to right

### Bitwise AND (&):

- Bitwise AND operator is denoted by the single ampersand sign (&). Two integer operands are written on both sides of the (&) operator.

- The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

**Perform bitwise AND operation on two integers 12 and 25.**

**12 = 00001100 (In Binary)**

**25 = 00011001 (In Binary)**

**Bit Operation of 12 and 25**

**00001100  
& 00011001**

---

**00001000 = 8 (In decimal)**

**Program for bitwise and (&) operator:**

```
#include <stdio.h>
int main()
{
    int a=12,b=25,c;
    c=a&b;
    printf("12 & 25=%d",c);
    return 0;
}
```

**Output:**

**12 & 25=8**

**Bitwise OR (|):**

- The bitwise OR operator is represented by a single vertical sign (|). Two integer operands are written on both sides of the (|) symbol.
- The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1.

**12 = 00001100 (In Binary)**

**25 = 00011001 (In Binary)**

**Bitwise OR Operation of 12 and 25**

**00001100  
| 00011001**

---

**00011101 = 29 (In decimal)**

**Program for bitwise or (|) operator:**

```
#include <stdio.h>
int main()
{
    int a=12,b=25,c;
```



```

c=a | b;
printf("12 | 25=%d",c);
return 0;
}

```

**Output:**

12 | 25=29

### Bitwise Not ( ~):

- The bitwise NOT operator is also known as one's complement operator/ complement Operator.
- It is represented by the symbol tilde (~). It takes only one operand or variable and performs complement operation on an operand.
- When we apply the complement operation on any bits, then 0 becomes 1 and 1 becomes 0.

**35 = 00100011 (In Binary)**

**Bitwise complement Operation of 35**

**~ 00100011**

**11011100 = 220 (In decimal)**

### Twist in bitwise complement operator in C Programming

- The bitwise complement of 35 (~35) is -36 instead of 220, but why?
- For any integer n, bitwise complement of n will be - (n+1). To understand this, you should have the knowledge of 2's complement.

### 2's Complement

- Two's complement is an operation on binary numbers. The 2's complement of a number is equal to the complement of that number plus 1. For example:
- The bitwise complement of 35 is 220 (in decimal). The 2's complement of 220 is -36. Hence, the output is -36 instead of 220.

**Bitwise complement of any number N is -(N+1). Here's how:**

bitwise complement of N = ~N (represented in 2's complement form) 2's complement of ~N = -((~N)+1) = -(N+1)

### Steps to solve bitwise not:

1. Take 1's Complement
2. Take 2's complement

### Question 1:

Find out complement of 35 i.e. ~35.

### Solution:

**35 -> 00100011 (binary of 35)**

**Step 1: 1's complement of 35 -> 11011100**

**Step 2: 2's complement -> - ( 00100011 + 1)**

**= -(00100100)=- 36 (decimal of 00100100)**

**So , ~35= -36**

#### **Program for Bitwise Not (~) Operator:**

```
#include <stdio.h>
int main()
{
    int a=35,b;
    b=~a;
    printf("~35=%d",b);
    return 0;
}
```

**Output:**

**~35=-36**

#### **Bitwise XOR (^):**

- Bitwise exclusive OR operator is denoted by (^) symbol. Two operands are written on both sides of the exclusive OR operator.
- The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite (Different).

**12 = 00001100 (In Binary)**

**25 = 00011001 (In Binary)**

#### **Bitwise XOR Operation of 12 and 25**

**00001100**

**^ 00011001**

---

**00010101 = 21 (In decimal)**

#### **Program for bitwise xor (^) operator:**

```
#include <stdio.h>
int main()
{
    int a=12,b=25,c;
    c=a^b;
    printf("12^25=%d",c);
    return 0;
}
```

**Output:**  
 $12 \wedge 25=21$

X	Y	X & Y	X   Y	X ^ Y	~X
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

### Bitwise Left Shift (<<):

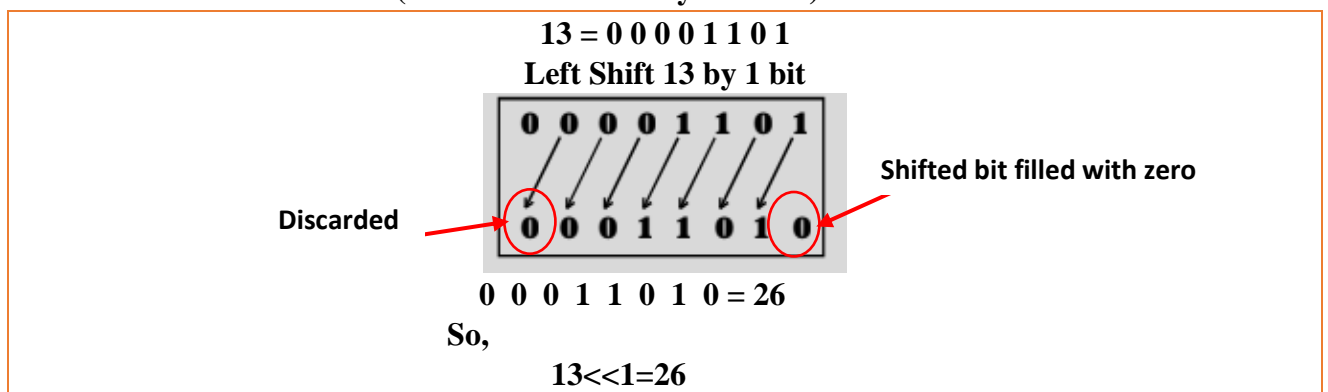
- Left shift operator shifts all bits towards left by a certain number of specified bits. The bit positions that have been shifted by the left shift operator are filled with 0.
- The symbol of the left shift operator is <<.

#### Syntax:

Operand << n;

#### Where,

- Operand is an integer expression on which we apply the left-shift operation.
- n is the number of bits to be shifted.(value of n can be any number)



### Program for Left Shift Operator:

```
#include <stdio.h>
int main()
{
    int a=13,b;
    b=a<<1;
    printf("13<<1=%d",b);
    return 0;
}
```

**Output:**  
 $13 \ll 1 = 26$

### Bitwise Right Shift (>>):

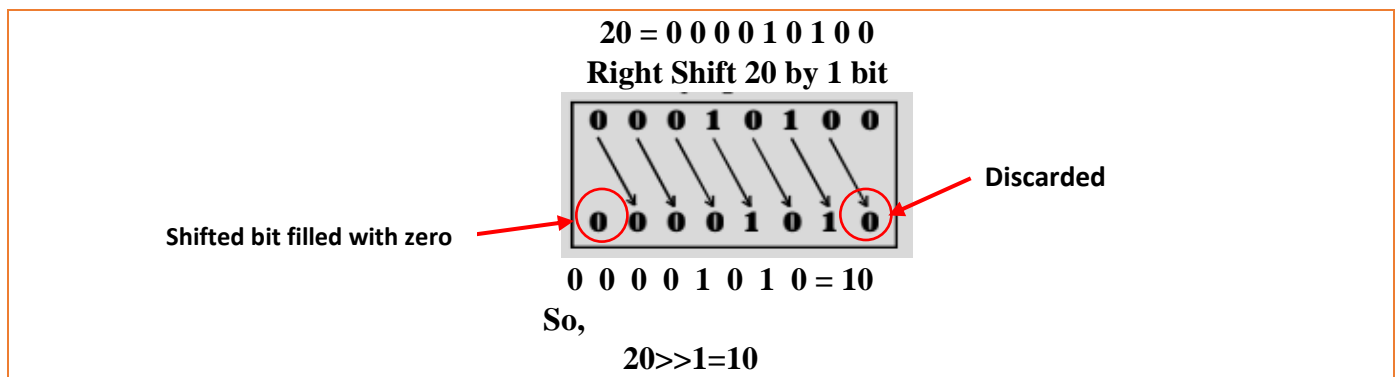
- Right shift operator shifts all bits towards right by a certain number of specified bits. The bit positions that have been shifted by the right shift operator are filled with 0.
- The symbol of the right shift operator is >>.

**Syntax:**

Operand >> n;

**Where,**

- Operand is an integer expression on which we apply the right-shift operation.
- n is the number of bits to be shifted.(value of n can be any number)



**Program for right Shift Operator:**

```
#include <stdio.h>
int main()
{
    int a=20,b;
    b=a>>1;
    printf("20>>1=%d",b);
    return 0;
}
```

**Output:**

20>>1=10

**8. Special Operators:**

Operator	Description	Example
,	Comma Operator	int a,b,c,d
sizeof	sizeof operator	sizeof(int)
&	Ampersand (address of) Operator	&operand

**Comma Operator ( , ):**

- The comma operator represented as a comma (,) accepts two operands and it is used to combine two or more statements into a single statement. Thus, compact statements can be written using comma operator.
- The statements are executed one by one from left to right and hence it is left associative operator.

**Example:**

Suppose we have declare 3 variables a1, a2 and a3 of int type like this,

```
int a1;
```

```
int a2;
```

```
int a3;
```

**So the above statement can be written into a single statement using comma operator like this,**

```
int a1, a2, a3;
```

**Sizeof operator:**

- This operator is used to find the number of bytes occupied by a variable or a datatype in the computer memory.

**The syntax is shown below:**

```
sizeof(variable_name / Data_type);
```

**Example:**

```
char ch;
```

```
sizeof(int); //4 byte
```

```
sizeof(ch); // 1 byte
```

**Address of Operator (&):**

- The "Address Of" Operator denoted by the **ampersand character (&)**, & is a unary operator, which returns the address of a variable.

**Example:**

```
int a;
```

```
&a ; //returns address of a
```

**Program to display address of a variable**

```
#include <stdio.h>
int main()
{
    int a=20;
    printf("Address of a in decimal :%d",&a);
    printf("\nAddress of a in Hexadecimal :%X",&a);
    return 0; }
```

**Output:**

Address of a in decimal :6487580

Address of a in Hexadecimal :62FE1C

**Note:**

1. Operators with 1 operand is known as unary operator.  
&, ++, --, ~ are the unary operators.
2. Operators with 2 operands is known as binary operator.

+, -, \*, / are the binary operators.

3. Operators with 3 operands is known as ternary operator.

? : is ternary operator.

### Hierarchy of operators:

Operator category	Operators in order of precedence (highest to lowest)	Associativity
<b>0</b> <b>  </b>	<b>Innermost brackets/ functions calls</b> <b>any element reference</b>	<b>L → R</b> <b>(Left to right)</b>
<b>Unary operators</b>	<b>++, --, !, sizeof, ~, +, -, &amp;, *</b>	<b>R → L</b> <b>(Right to left)</b>
<b>Member access</b>	<b>* or -&gt;</b>	<b>L → R</b>
<b>Arithmetic operators</b>	<b>*, /, %</b>	<b>L → R</b>
<b>Arithmetic operators</b>	<b>-, +</b>	<b>L → R</b>
<b>Shift operator</b>	<b>&lt;&lt;, &gt;&gt;</b>	<b>L → R</b>
<b>Relational operators</b>	<b>&lt;, &lt;=, &gt;, &gt;=</b>	<b>L → R</b>
<b>Equality operators</b>	<b>==, !=</b>	<b>L → R</b>
<b>Bit-wise AND</b>	<b>&amp;</b>	<b>L → R</b>
<b>Bit-wise X-OR</b>	<b>^</b>	<b>L → R</b>
<b>Bit-wise OR</b>	<b> </b>	<b>L → R</b>
<b>Logical AND</b>	<b>&amp;&amp;</b>	<b>L → R</b>
<b>Logical OR</b>	<b>  </b>	<b>R → R</b>
<b>Conditional operator</b>	<b>?</b>	<b>R → L</b>
<b>Assignment operators</b>	<b>=, +=, -=, /=, *=, %=, &amp;=,  =, &lt;&lt;=, &gt;&gt;=</b>	<b>R → L</b>
<b>Comma operator</b>	<b>,</b>	<b>L → R</b>

### Precedence operators:

- In C language, each operator is associated with a priority value. Based on the priority, the expressions are evaluated. A part of the expression with priority value 1 is evaluated first; part of the expression with priority value 2 is evaluated next and so on.
- These priority values that are associated with various operators are called Precedence of operators.

### Associativity Operators:

- If all the operators in an expression have equal priority, then the direction order chosen (left to right evaluation or right to left evaluation) to evaluate an expression is called associativity of operators.

**Associativity Operators are classified into 2 categories:**

- ☐ Left associative (Left to right associative)
- ☐ Right associative (Right to left associative)

#### 1. Left Associative:

- In an expression, if there are two or more operators having the same priority and are evaluated from left-to-right, then the operators are called Left to Right associative operators.

**For example,**

$$8 + 4 + 3 = 15$$

$$8 - 4 - 3 = 1$$

In the above example + and – both are having equal priority so in this case according to associativity this expression will be evaluated and associativity of + and – and **left to right**. So above expression will execute from left to right.

## 2. Right Associative:

- In an expression, if there are two or more operators having the same priority and are evaluated from right-to-left, then the operators are called Right to Left associative operators.

**For Example,**

**i = j = k = 10;**

Above expression will execute from right to left because 3 times = operator is used and associativity of = is right to left.

\*\*\*\*