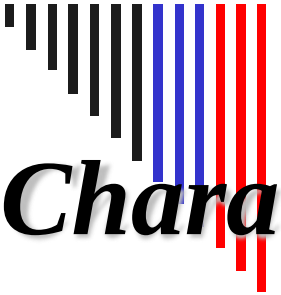


Unit II

Character set, Operators, Expressions

2-1 Character Set in C

- A character set is a set of characters that are valid in C language. The character set in C is grouped into the following categories.
  - Alphabets(A..Z, a..z)
  - Digits(0..9)
  - Special characters
  - White spaces

Special Characters in C Programming

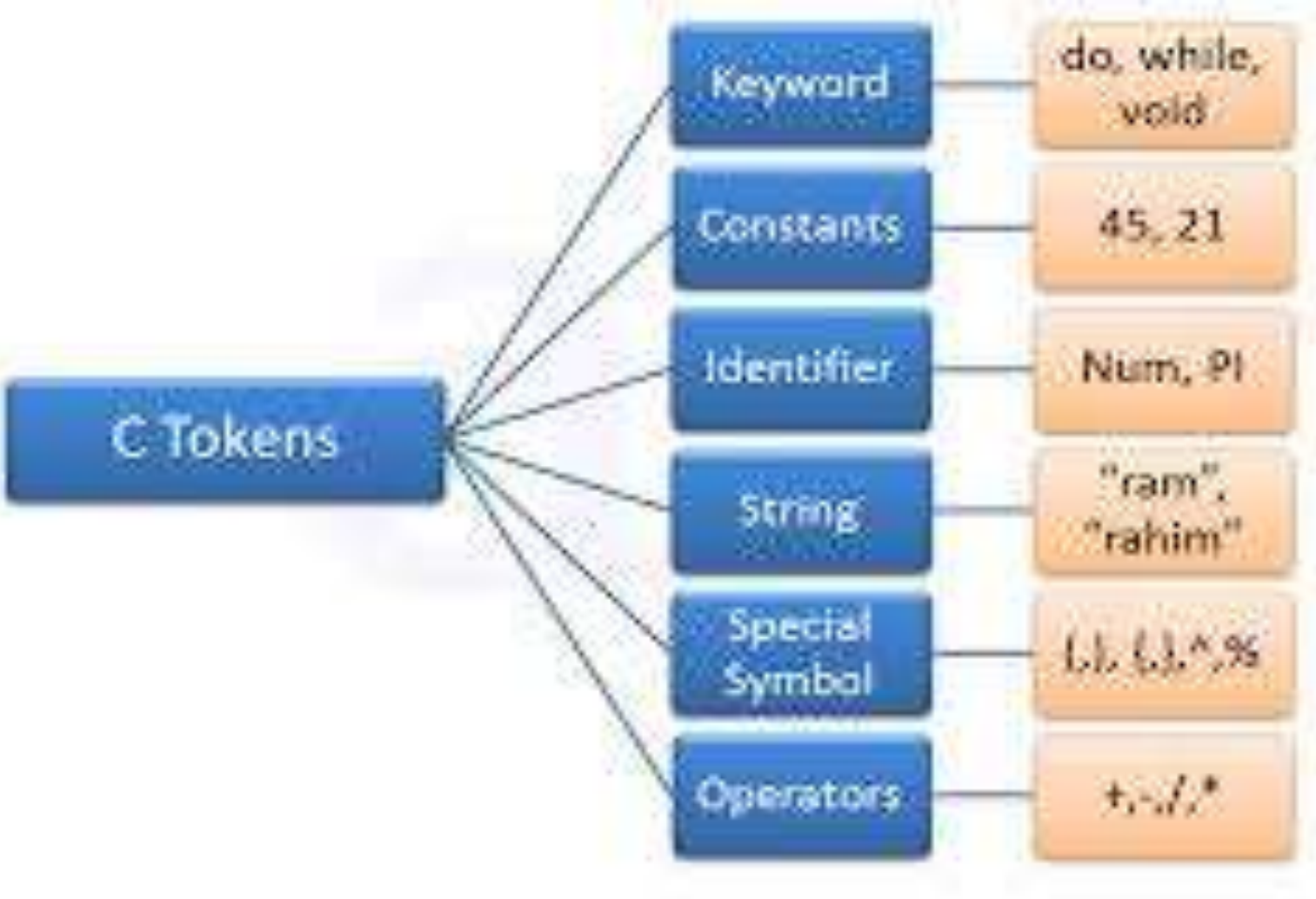
,	<	>	.	_
(	)	;	$	:
%	[	]	#	?
'	&	{	}	"
^	!	*	/	|
-	\	~	+	

White space Characters

Blank space, newline, horizontal tab, carriage return and form feed.

Types of Tokens in C

Every smallest individual units in a C program are known as C tokens. Tokens are of six types. They are:



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;
```

- Identifier names must be different from keywords.
  You cannot use int as an identifier because int is a keyword.

Rules for naming identifiers

- A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.
- The first letter of an identifier should be either a letter or an underscore.
- You cannot use keywords like int, while etc. as identifiers.
- There is no rule on how long an identifier can be.
- However, you may run into problems in some compilers if the identifier is longer than 31 characters.
- Identifier names are case sensitive. For example: FACT and fact are not same in C

You can choose any name as an identifier if you follow the above rule, however, give meaningful names to identifiers that make sense.

Keywords

- Keywords are predefined, reserved words used in programming that have special meanings. They cannot be used as identifiers. For eg:

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

C Keywords

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

Variables

- In programming, a variable is a container (storage area) to hold data.
- To indicate the storage area, each variable should be given a unique name (identifier).
- Variable names are just the symbolic representation of a memory location
- The value of a variable can be changed, hence the name variable.

For example:

```
int playerScore = 95;
```

Here, playerScore is a variable of int type. Here, the variable is assigned an integer value 95.

.

Constants

a constant using the #define preprocessor directive
a constant using the #define preprocessor directi

- If you want to define a variable whose value cannot be changed, you can use the const keyword. This will create a constant. For example,
- const double PI=3.14
- Notice, we have added keyword const.

  Here, PI is a symbolic constant; its value cannot be changed.

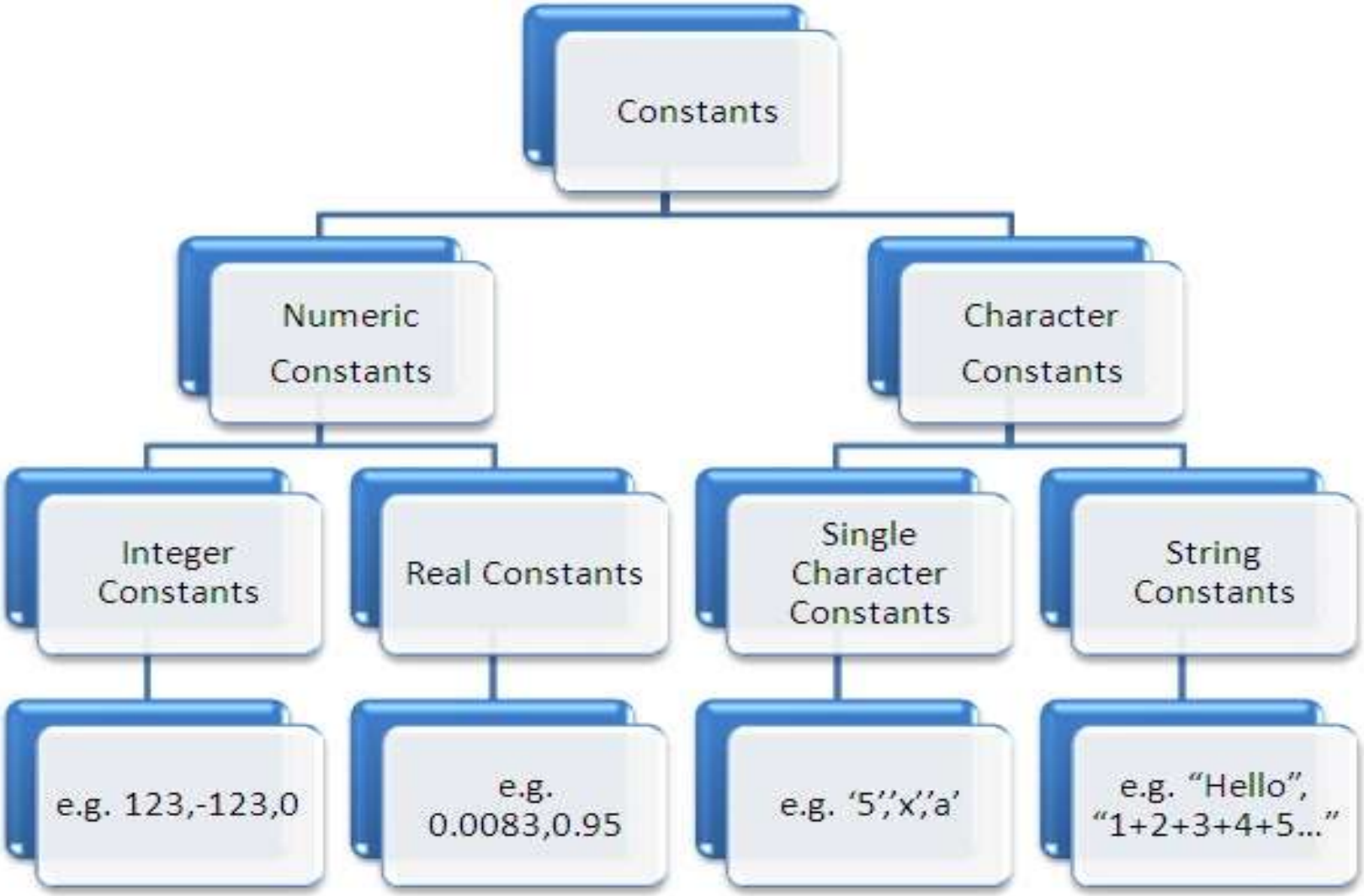
```
const double PI = 3.14;
PI = 2.9; //Error
```

You can also define a constant using the #define preprocessor directive

```
# define PI 3.14
```

Constants

- There are four type of constants in C. They are:
  - Integer constants
  - Character constants
  - Real/Floating point constants
  - String constants


Constants
Numeric Constants
Character Constants
Integer Constants
Real Constants
Single Character Constants
String Constants
e.g. 123,-123,0
e.g. 0.0083,0.95
e.g. '5','x','a'
e.g. "Hello", "1+2+3+4+5..."

Integer Constants

- An integer constant is an integer quantity which contains a sequence of digits.
- It should not have a decimal point.
- Blanks and commas are not allowed within an integer constant.
- An integer constant can be either +ve or -ve.
- The constant must lie within the range of the declared data type (including qualifiers long,short etc.).
- Example of valid integer constants:- 976 8987 5 -25 etc.
- Example of invalid integer constants:- 78.43 7-8 89,76 etc.

Floating Point Constants

- They are also known as real constants.
- A real constant can either be represented in decimal notation or in exponential form.
- It can be either +ve or -ve.
- Commas and blank space are not allowed within a real constant.
- Decimal Notation
  - Here, a whole number is followed by a decimal point and a fractional part.
  - It is possible to omit digits before the decimal point and also after the decimal point but not both.

  + 215. .567 -.087 14.43

Exponential Notation

- A floating-point constant in exponential form is of the general form: mantissa e exponent or mantissa E exponent
- The mantissa is either an integer or a real number in decimal notation, with an optional + or - sign
- The exponent is an integer number with an optional + or - sign
- The allowable range for floating-point constants are -3.4E38 to 3.4E38
- Examples:
  4.56E4 -5.6E-6 -0.4E7 -10.6E-2

Character Constants

- A "character constant" is formed by enclosing a single character from the representable character set within single quotation marks (' ')
- A character can be
  - alphabet like a,b,A,C etc
  - special character like &,^, $, #,@ etc
  - single digit from 0 through 9.
  - an escape sequence character like space ‘ ‘ or a null character ‘\o’ or a new line ‘\n’ etc.
  - Example: ‘A’ ‘a’ ‘ b’ ‘8’ ‘#’ etc.
- Note: ‘8’ is not same as 8.

ASCII codes

- Each character has a corresponding ASCII value. **ASCII value** is the numeric code of a particular character and is stored inside the machine's character set.
- Therfore, every charcter has a numeric value associated with it.
- Example:

printf("%d", 'a'); this will print 97 which is the asci value of letter a.

printf("%c", '97'); will output the letter a.

Since each character constant represents an integer value, it is possible to perform arithmetic operations on them.

String Constants

- A string constant is a collection of characters enclosed in double quotations “”
- It may contain alphabets, digits, special characters and blank space.

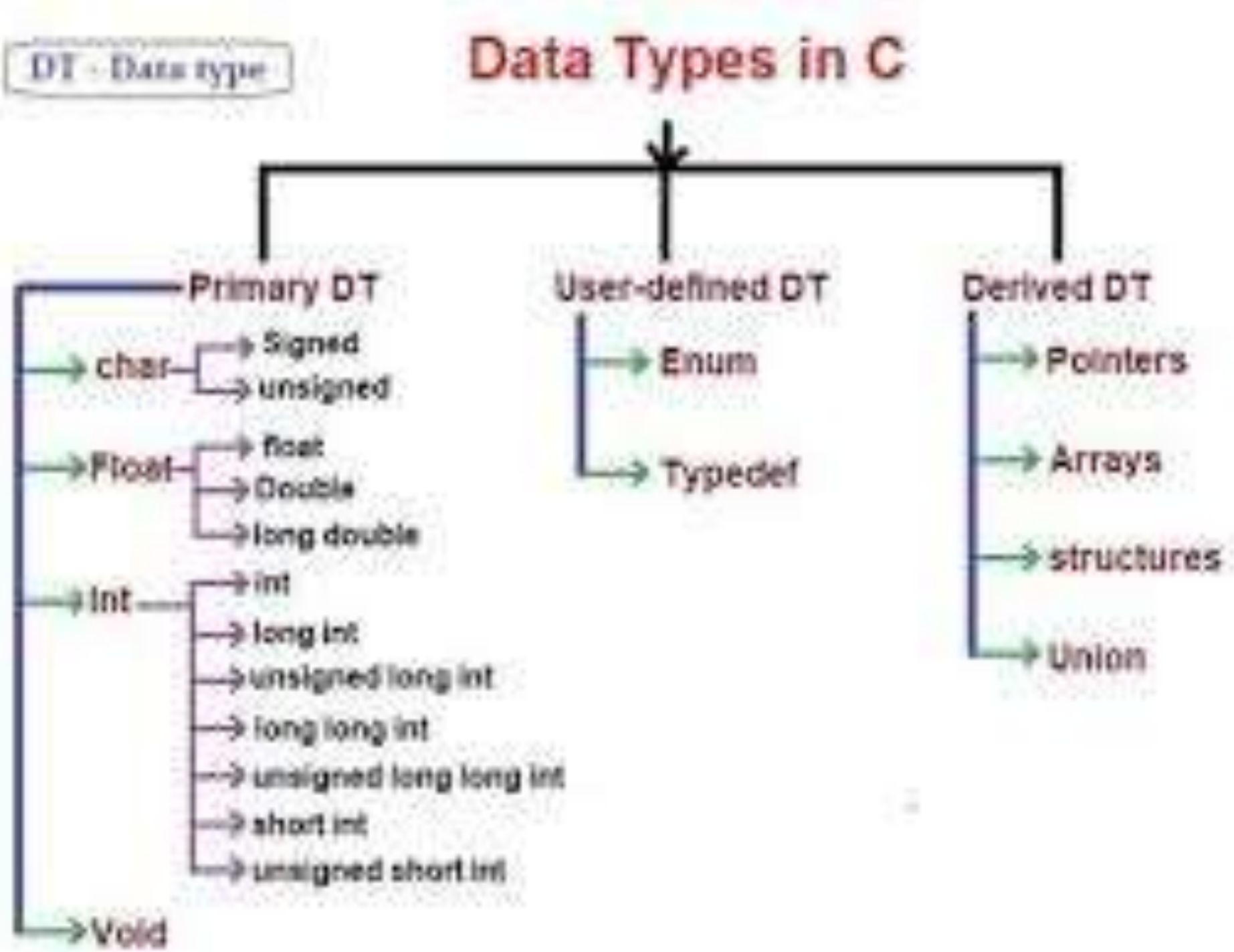
- **Example:** “Circuits Today123”

Backslash character constants

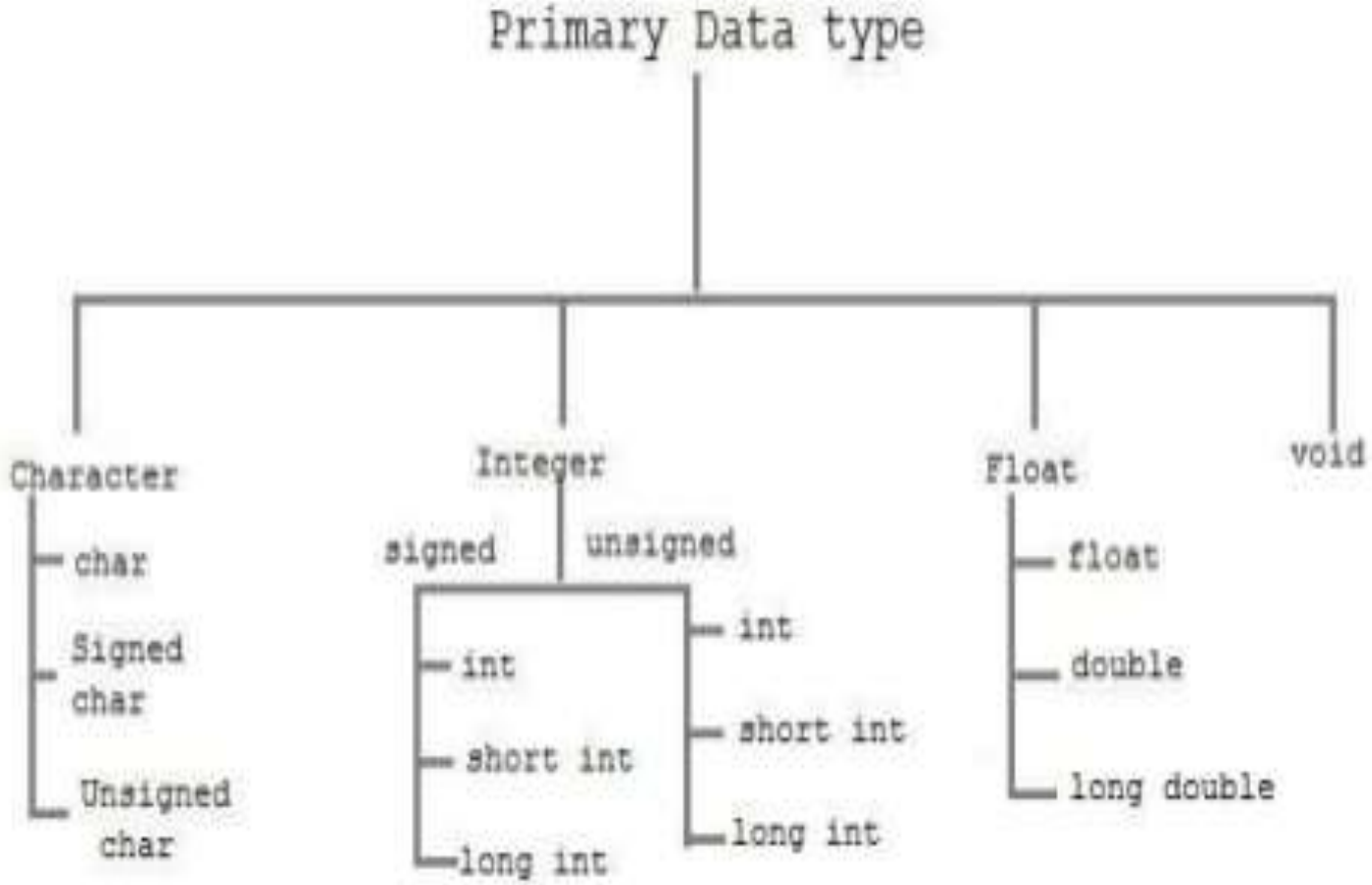
- C supports some special backslash character constants that are used in output functions.
- Each backslash character is a single character but is made up of a combination of two characters.
- These character combinations are also known as escape sequences.
- To represent a newline character, single quotation mark, or certain other characters in a character constant, you must use escape sequences

List of backslash character constants

Escape Sequence	Represents
\a	Bell (alert)
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\'	Single quotation mark
\"	Double quotation mark
****	Backslash
\?	Literal question mark


DT - Data type
Data Types in C
Primary DT
User-defined DT
Derived DT
char
Signed
unsigned
Float
float
Double
long double
Int
int
long int
unsigned long int
long long int
unsigned long long int
short int
unsigned short int
Void
Enum
Typedef
Pointers
Arrays
structures
Union

Primary Data Types in C



Integers

- Integers are whole numbers with a range of values supported by a particular machine.
- Integers occupy one word of storage.
- Word sizes vary in different types of machines(16 bit or 32 bits)
- In a 16 bit machine, the size of the integer values range from -32,768 to 32,767(-2^{15} to $2^{15}-1$)
- A signed integer uses 1 bit for sign and 15 bits for magnitude of the number
- By default, an int declaration assumes a signed number.

Integers

- **000, 001,010,011,100,101,110,111**
- 0= + ve sign 0 to 3
- 1=-ve sign -1 to -4 (values ranging from -2^2 to 2^2-1 i.e. -4,-3,-2,-1,0,1,2,3)

0 0 - 0

0 1 - 1

1 0 – 2

1 1 - 3

1 0 0 - 4

Char Data types

- Used to define characters.
- A single character occupies one byte(8 bits)

Data Type Qualifiers

- Each of these data type has got qualifiers.
- The purpose of a qualifier is to manipulate the range of a particular data type or its size.
- The 4 qualifiers in C are
  - **Long**
  - **Short**
  - **Signed**
  - **unsigned.**

  long and short are called size qualifiers

  signed and unsigned are called sign qualifiers.

Long and short qualifiers

- The qualifiers long and short are used to increase storage size of the data type.
- Integer data type int is normally 2 byte.
- If you declare it as **long int** – then its size will increase from **2 bytes to 4 bytes**.
- Similarly if you declare it as **short int** – its size will reduce from **2 bytes to 1 byte.**

Signed Qualifiers

- Use of unsigned qualifiers on data types allows all the bits to be used for magnitude and no bit will be kept aside for sign.
- This unsigned qualifier is used when we are dealing with only positive numbers.
- If we declare **unsigned int** – then its range is from 0 to 65535. (0 to 2^{16}-1)
- **Signed int** – then its range is from (-32767 to 32768). In the case of signed int, one bit (MSB) is used to store the sign of the integer +/-.

Float Point Types

- Floating point numbers are stored in 32 bits of memory with 6 digits of precision.
- Floating point numbers are defined in C by keyword float.
- When accuracy provided by float is not enough, type double and long double can be used.
- Double – 64 bits, 16 bit precision
- Long double- 80 bits, 25-26 bits precision

Data Type	Memory Size	Range	Format specifiers
char			
signed char	1 byte	−128 to 127	%c
unsigned char	1 byte	0 to 255	%uc
int			
signed int	2 byte	−32,768 to 32,767	%d
unsigned int	2 byte	0 to 65,535	%ud
short int			
signed short int	1 byte	−128 to 127	%hd
unsigned short int	1 byte	0 to 255	%uhd
long int			%ld
signed long int	4 byte	-2147483648 to 2147483647	%ld
unsigned long int	4 byte	O to 42949667295	%uld
float	4 byte	3.4e-38 to 3.4 e+ 38	%f
double	8 byte	1.7e – 308 to 1.7e+308	%lf
long double	10 byte	3.4e – 4932 to 3.4e+4932	%Lf

User Defined Data types

- **User defined** data types are those data types which are defined by the user/programmer himself. There are two types:
  - **Enumerated data type**
  - **Typedef data type**

Enumerated data type

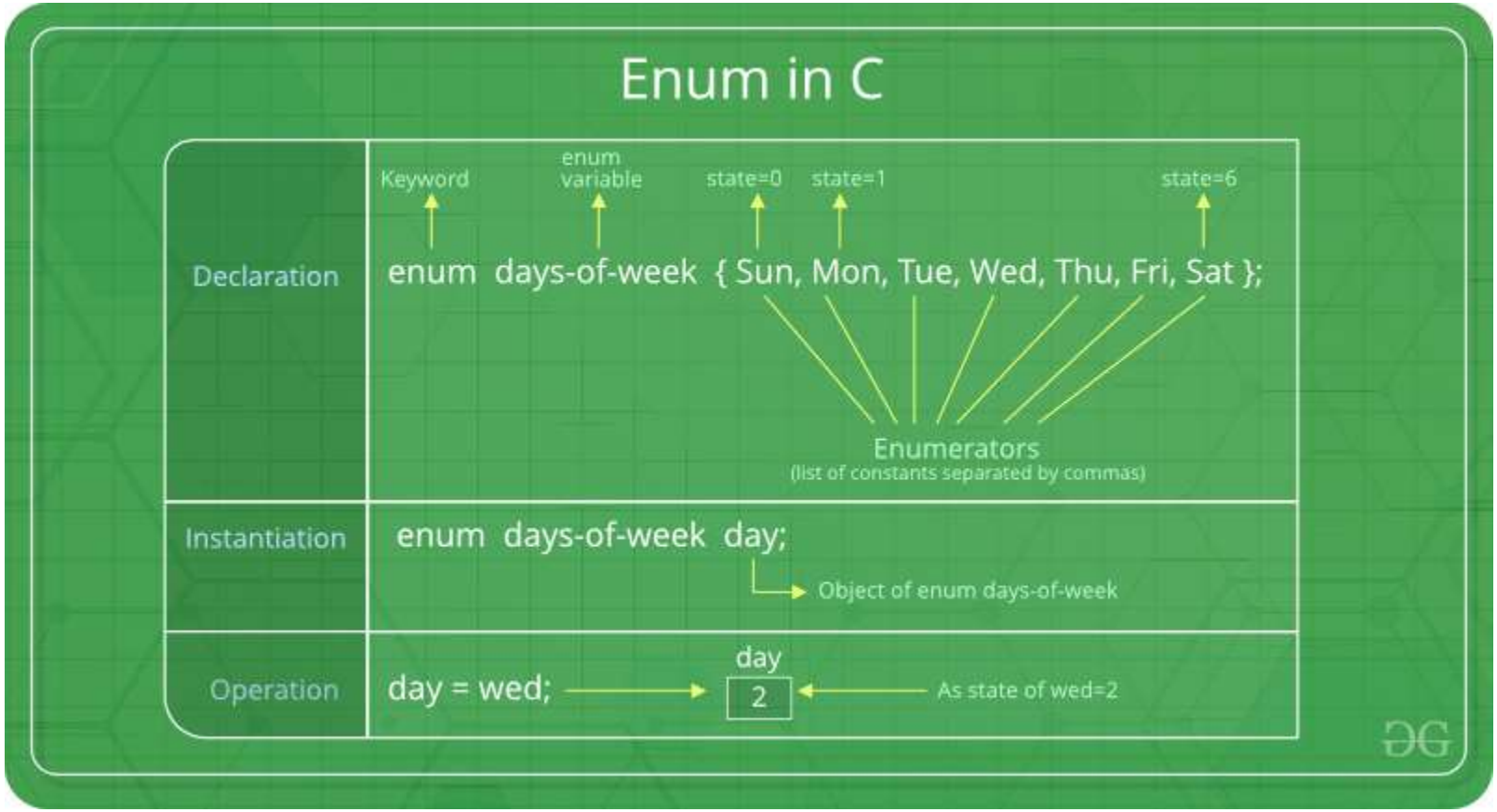
- **Enumeration**: Enumeration (or enum) is a user defined data type in C. It defines a identifier with a list of values(called enumeration constants) enclosed within braces.
- New variables can be declared whose data type is the enumerated identifier and can have only one of the values in the list.
- The compiler automatically assigns integer digits beginning with 0 to all enumeration constants.
- Automatic assignments can be overridden by assigning values explicitly to the enumeration constants.

Example of Enumerated data type

```
#include<stdio.h>
#include<conio.h>
enum day{mon,tues,wed,thurs,fri,sat,sun};
void main()
{
enum day weekday, weekend:
weekday=wed;
weekend=sat;
printf("weekday=%d", weekday);
printf("weekend=%d",weekend);
getch();
}
Weekday=2
Weekend=5
```

```
#include<stdio.h>
#include<conio.h>
enum day{mon,tues,wed=5,thurs,fri,sat,sun};
void main()
{
enum day weekday, weekend:
weekday=wed;
weekend=sat;
printf("weekday=%d", weekday);
printf("weekend=%d",weekend);
getch();
}
Weekday=2
Weekend=5
```

Here, the value of wed has been explicitly changed to 5. As a result the values of thurs, fri, sat change to 6 7 8 etc.


Enum in C
Declaration
Keyword
enum variable
state=0
state=1
state=6
enum days-of-week { Sun, Mon, Tue, Wed, Thu, Fri, Sat };
Enumerators
(list of constants separated by commas)
Instantiation
enum days-of-week day;
Object of enum days-of-week
Operation
day = wed;
day
2
As state of wed=2

Typedef data type

- C allows you to define explicitly new data type names by using the keyword typedef.
- typedef, does not actually create a new data class, rather it defines a name for an existing type.
- Using typedef one can also aid in self-documenting code by allowing descriptive names for the standard data types.
- **Syntax:**

```
typedef type name;
```

- Here, *type* is any C data type and *name* is the new name for this data type.
  This defines another name for the standard *type* of C.

Typedef user defined data type

Example:

typedef int units;

typedef float marks;

Here, units symbolizes int and marks symbolizes float.

They can later be used to declare variables

units batch 1, batch 2;

marks m1, m2;

Derived Data Types

- Derived data types are derived from the primary data types.
- They include:
  - Arrays
  - Pointers
  - Functions
  - Structure
  - Unions

Type Conversion

- In C, it is possible to convert one data type to another. There are two methods:
- Implicit conversion
- Explicit conversion

Explicit Conversion

- Explicit conversion: The user converts the data into a certain data type he needs. Th.

  is is called type casting.

```
int a=5, b=2;
float c;
c= (float)a/b;
```

Here, a is explicitly converted to float.

Implicit Conversion

- In C, lower data type is automatically promoted to upper data type in expressions.

Expression with

Char, short, int -> int

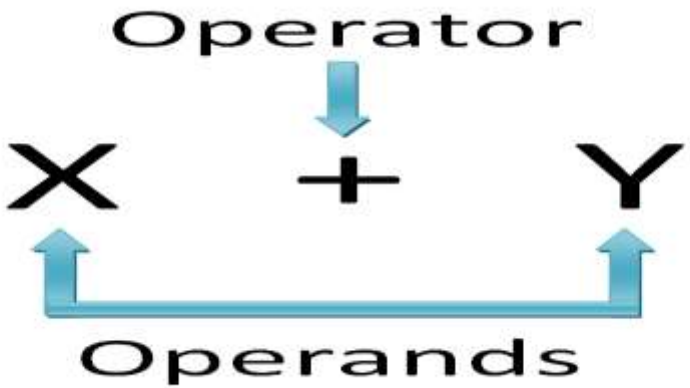
Float, non-float-> float

double, non-double-> double

Long int, unsigned int-> long int

Operators and Expressions

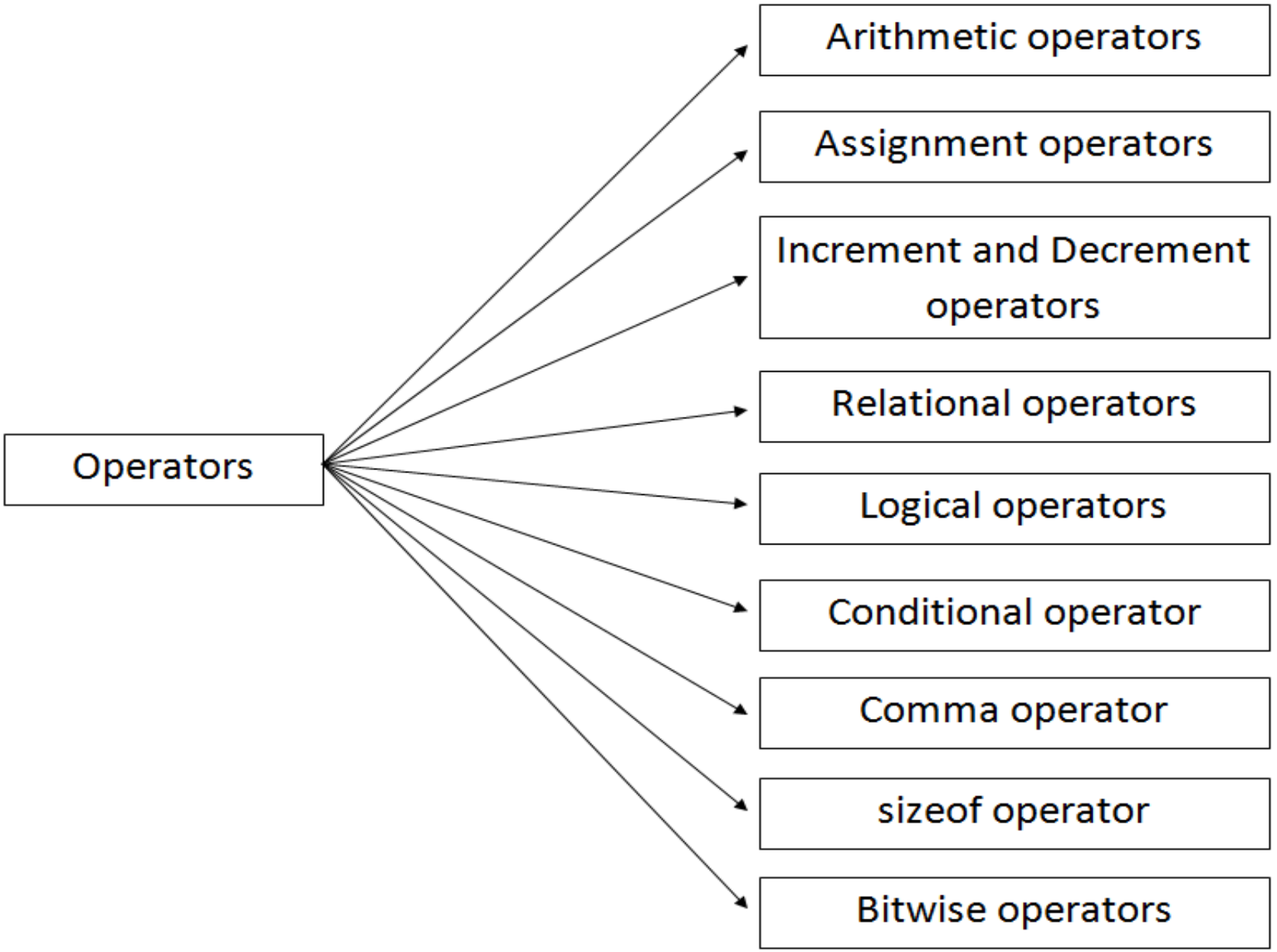
- An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.
- An operand is a data item on which an operator acts. Some operators require two operands, while others act upon only one operand.
- An expression is defined as combination of both operator and operand



Operators

C includes a large number of operators that fall under several different categories, which are:

- Arithmetic operators
- Assignment operators
- Increment and Decrement operators
- Relational operators
- Logical operators
- Conditional operator
- Comma operator
- sizeof operator
- Bitwise operators


Operators
Arithmetic operators
Assignment operators
Increment and Decrement operators
Relational operators
Logical operators
Conditional operator
Comma operator
sizeof operator
Bitwise operators

Arithmetic operators in C

Arithmetic operators are used for numeric calculations. They are of two types:

- Unary arithmetic operators:
- Binary arithmetic operators

Unary Operators

Operator	Meaning
+	Unary plus operator; indicates positive value (numbers are positive without this, however)
-	Unary minus operator; negates an expression

Unary operators require only one operator. There are only two unary operators.

Example:

+x −y

Binary Operators

- Binary operators require two operands. There are 5 binary arithmetic operators

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Modulus operator

- The modulus operator (%) finds the remainder of an integer division. This operator can be applied only to integer operands and cannot be used on float or double operands.
- While performing modulo division, the sign of the result is always the sign of the first operand (the dividend). Therefore,

```
 15 %  7 =  1
-15 %  7 = -1
 15 % -7 =  1
-15 % -7 = -1
```

- Note that unary plus and unary minus operators are different from the addition and subtraction operators.
- The second operand must be nonzero for division and modulus operations.

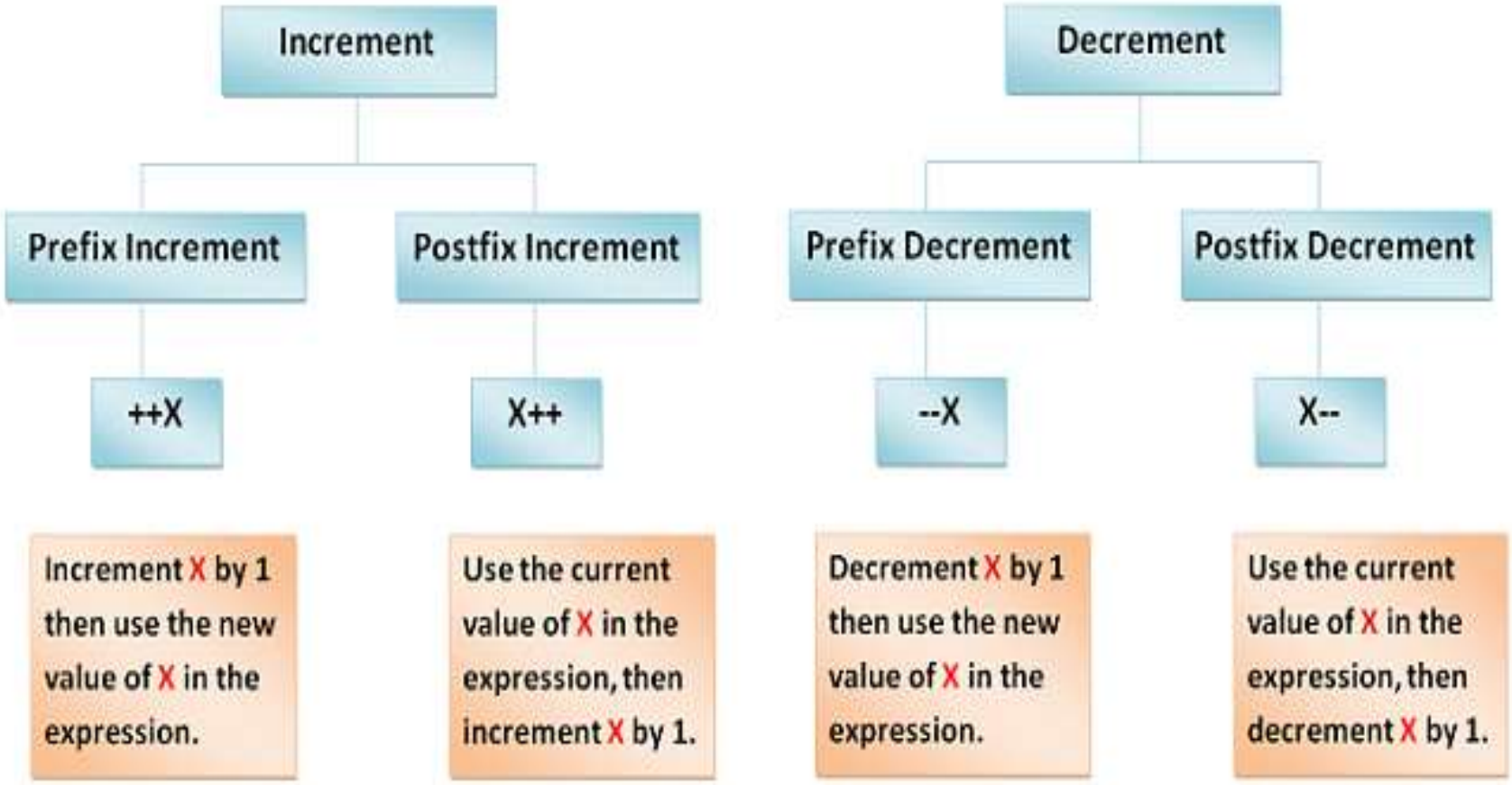
example

- When both operands are integers, the resulting value is always an integer. Let us take two variables a and b. The value of a is 13 and b is 4, the results of the following operations are:

Expression	Result
a + b	17
a - b	9
a * b	52
a / b	3 (decimal part truncates)
a % b	1 (Remainder after integer division)

Increment and Decrement Operators in C

- The increment(++) and decrement(--) operators are unary operators that increment and decrement value of a variable by 1.
- ++x is equivalent to x = x + 1
  - -x is equivalent to x = x - 1


Increment
Prefix Increment
Postfix Increment
++X
X++
Increment X by 1 then use the new value of X in the expression.
Use the current value of X in the expression, then increment X by 1.
Decrement
Prefix Decrement
Postfix Decrement
--X
X--
Decrement X by 1 then use the new value of X in the expression.
Use the current value of X in the expression, then decrement X by 1.

Prefix Increment

Let's take a variable x whose value is 5.

The statement:

y = ++x;

means first increment the value of x by 1 then assign the value of x to y. This single statement is equivalent to these two statements:

```
x = x + 1; x=6
y = x;       y=6
```

Prefix Decrement

Let's take a variable x whose value is 5.

The statement:

y = − −x;

means first decrement the value of x by 1 then assign the value of x to y. This single statement is equivalent to these two statements:

```
x = x − 1;  x=4
y = x;      y=4
```

Postfix Increment

Let's take a variable x whose value is 5.

The statement:

y = x++;

means first assign the value of x to y and then increment the value of x by 1. This single statement is equivalent to these two statements:

```
y = x;                  y=5, x=6
x = x + 1;
```

Postfix Decrement

Let's take a variable x whose value is 5.

The statement:

y = x− −;
means first assign the value of x to y and then decrement the value of x by 1. This single statement is equivalent to these two statements:

y = x; y=5, x=4

x = x − 1;

Relational Operators in C

- Relational operators are used to compare values of two expression.
- Expressions that contain relational operators are called relational expressions
- Relational operators return true or false value, depending on whether the conditional relationship between the two operands holds or not. If the relation is true then the result of relational expression is 1 and if the relation is false then the result of relational expression is 0.

Relational Operators

- The relational operators are:

Operator	Meaning
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Relational Operators

- Let us take two variables a=10 and b=8, and form simple relational expressions with them:

Expression	Relation	Result
a < b	False	0
a <= b	False	0
a > b	True	1
a >= b	True	1
a == b	False	0
a != b	True	1

Relational Operators

- Always remember in C a non-zero value means true and 0 means false.
- The relational operators are generally used in if...else construct and loops, which we will learn in upcoming tutorials.
- Assignment operator (=) and equal to operator (==).
  - The assignment operator (=) is used to assign the value of variable or expression, while equal to (==) is a relational operator used for comparison (to compare value of both left and right side operands).

Logical Operators in C

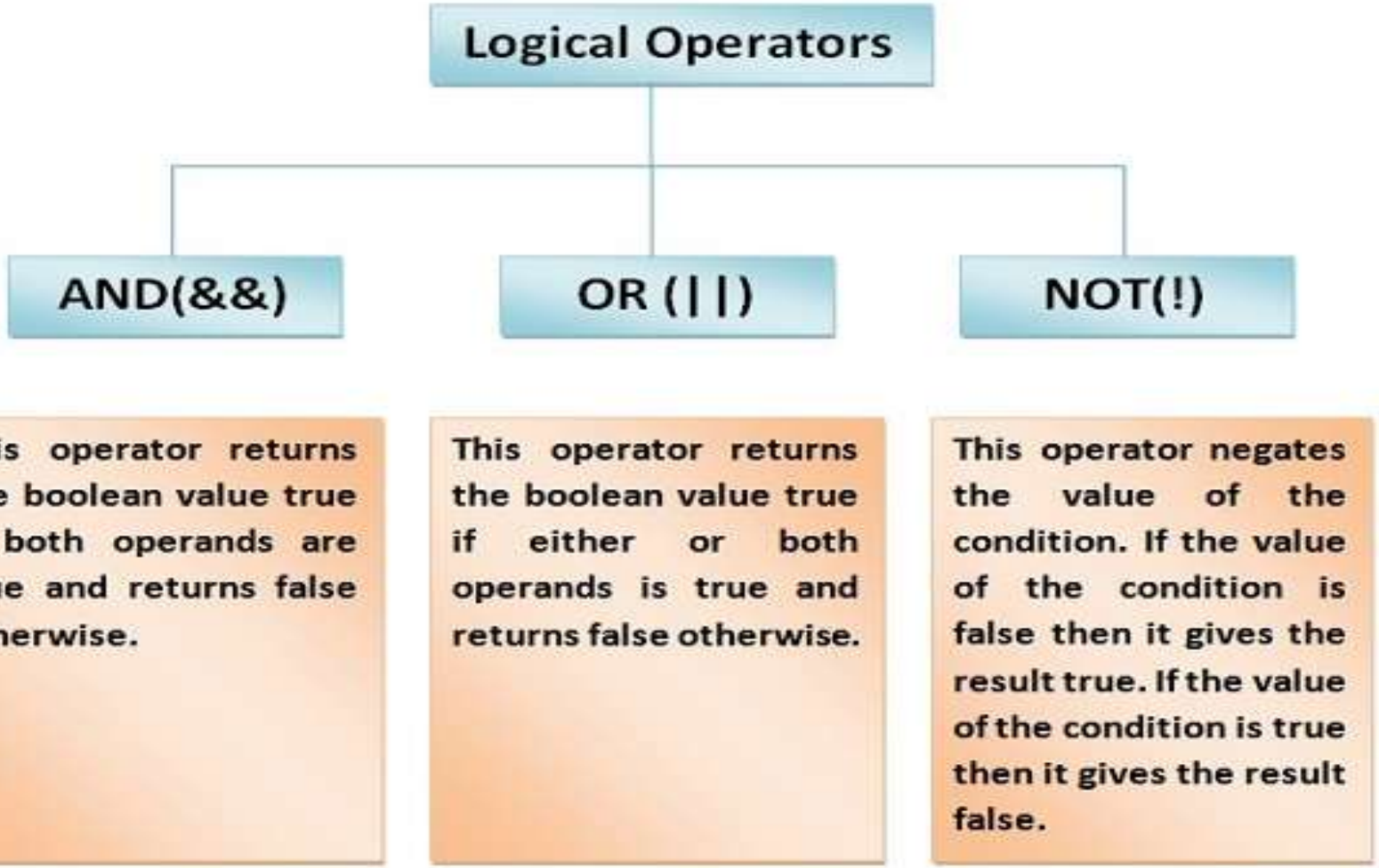
- In a programming language, an expression that combines two or more expressions is termed as a logical expression.
- For forming these logical expressions we use logical operators. If the result of logical operator is true then 1 is returned otherwise 0 is returned. The operands may be constants, variables or expressions.

Logical operators

- There is three type of logical operators:

Operator	Meaning
&&	AND
||	OR
!	NOT

- && and | | are binary operators while ! is a unary operator.
- In C any non-zero value is regarded as true and zero is regarded as false.


Logical Operators
AND(&&)
OR (||)
NOT(!)
This operator returns the boolean value true if both operands are true and returns false otherwise.
This operator returns the boolean value true if either or both operands is true and returns false otherwise.
This operator negates the value of the condition. If the value of the condition is false then it gives the result true. If the value of the condition is true then it gives the result false.

AND(&&) operator

The logical AND operator (&&) returns the boolean value true if both operands are true and returns false otherwise.

Syntax:

operand1 && operand2

The truth table of logical AND operator is:

Operand1	Operand2	Result
True	True	True
True	False	False
False	True	False
False	False	False

AND operator- Example

Let's take an example:

```
int a = 10, b = 9, c = 5, k = 0;
```

Expression	Intermediate Expression	Result
(a>2) && (b==10)	true && false	false
(b>=c) && (b>a)	true && false	false
(c==5) && (c<b)	true && true	true
a && b	true && true	true
a && k	true && false	false

OR(||)operator

- The logical OR operator (||) returns the boolean value true if either or both operands is true and returns false otherwise.

Syntax:

operand1 || operand2

- The truth table of logical OR operator:

Operand1	Operand2	Result
True	True	True
True	False	True
False	True	True
False	False	False

Logical OR- Example

Let's take an example:

```
int a = 10, b = 9, c = 5;
```

Expression	Intermediate Expression	Result
(a>2) | | (b==9)	true | | true	true
(b>=c) | | (b>a)	true | | false	true
(c==1) | | (c<b)	false | | true	true
a | | b	true | | true	true
(c==1) | | 0	false | | false	false

NOT(!) operator

- The logical NOT operator(!) negates the value of the condition.
- If the value of the condition is false then it gives the result true. If the value of the condition is true then it gives the result false.

Syntax:

!operand

The truth table of logical NOT operator is:

Condition	Result
False	True
True	False

Let's take an example:

int a = 10, b = 9;

Expression	Intermediate Expression	Result
! (a>2)	! true	false
! ((b>2) && (b>a))	! false	true
! a	! true	false
! (a>b)	! true	false
! (a||b)	! true	false

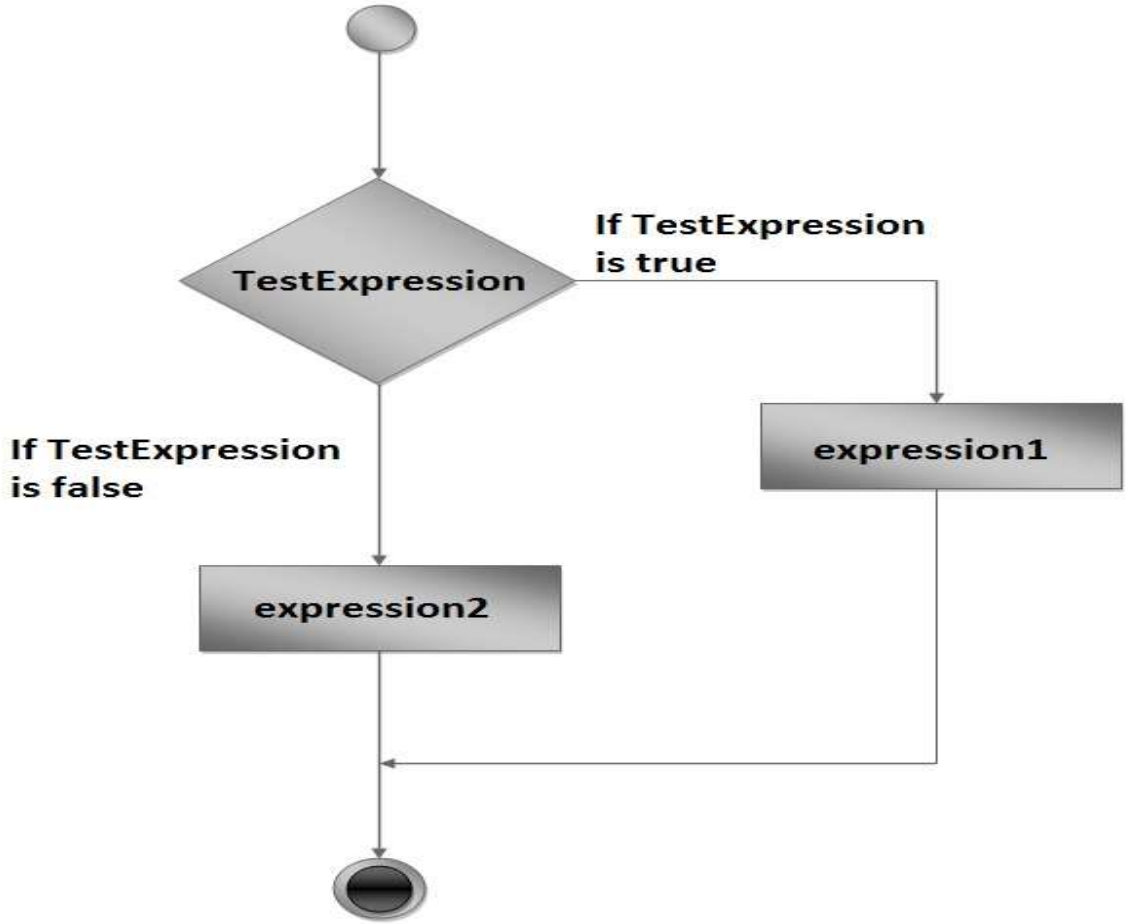
Conditional operator is also known as **ternary operator** (? and :) which requires three expressions as operands.

Syntax:

TestExpression ? expression1 : expression2

- The conditional operator works as follows:
- TestExpression is evaluated first.
  - If the TestExpression evaluates to true(nonzero), then expression1 is evaluated and it becomes the value of overall conditional expression.
  - If TestExpression evaluates to false(zero), then expression2 is evaluated and it becomes the value of overall conditional expression.

Conditional Operator



Examples

- int a = 7, b = 2;
  a>b ? a : b

Since a > b ? a : b is an expression, we can assign its value to a variable.

```
large = a > b ? a : b;
```

The conditional expression above performs the same operation as the following if/else statement:

```
if(a>b)/* Expression a>b tested */
    large=a; /* Executes if a>b is true */
else
    large=b; /* Executes if a>b is false */
```

Examples

```
int a=3, b=2, c=1, d=2;

a > b ? (c = 5) : (d = 4)
```

The conditional expression above performs the same operation as the following if/else statement:

```
if(a>b)/* Expression a>b tested */
    c = 5; /* Executes if a>b is true */
else
    d = 4; /* Executes if a>b is false */
```

Nesting of Conditional Operators

The general form of a nested conditional operators is,

```
TestExpression1  ? (TestExpression2 ? expression3 : expression4)
                 : (TestExpression3 ? expression5 : expression6)
```

Here is how conditional operator works

- TestExpression1 is evaluated first.
- If the TestExpression1 evaluates to true(nonzero), then (TestExpression2 ? expression3 : expression4) is evaluated.
- If TestExpression1 evaluates to false(zero), then (TestExpression3 ? expression5 : expression6) is evaluated.

Nested Conditional Operator-Example

Let's take an example:

```
int a=5, b=7, c=4;

large=((a>b)?(a>c ? a : c):(b>c ? b : c));
```

The conditional expression above performs the same operation as the following if/else statement:

Nested Conditional Operator-Example

```
if (a > b)
{
    if (a > c)
        large = a;
    else
        large = c;
}
else
{
    if (b > c)
        large = b;
    else
        large = c;
}
```

Comma Operator

- In C programming language, comma (,) can be used in two contexts :
- As a separator
  - Comma acts as a separator when used with function calls and definitions, variable declarations, enum declarations, and similar constructs. Example: int x, y, z; int a=5, b=3;
- As an operator
  - The comma operator (represented by the token ,) is a binary operator that takes two operands. It works by evaluating the first operand and then evaluates the second operand and returns the value (and type) as the result of the expression.

Coma Operator

- int a,b,sum; /* Comma as a separator. */
- sum = (a = 4, b = 3, a+b); /* Comma as an operator. */

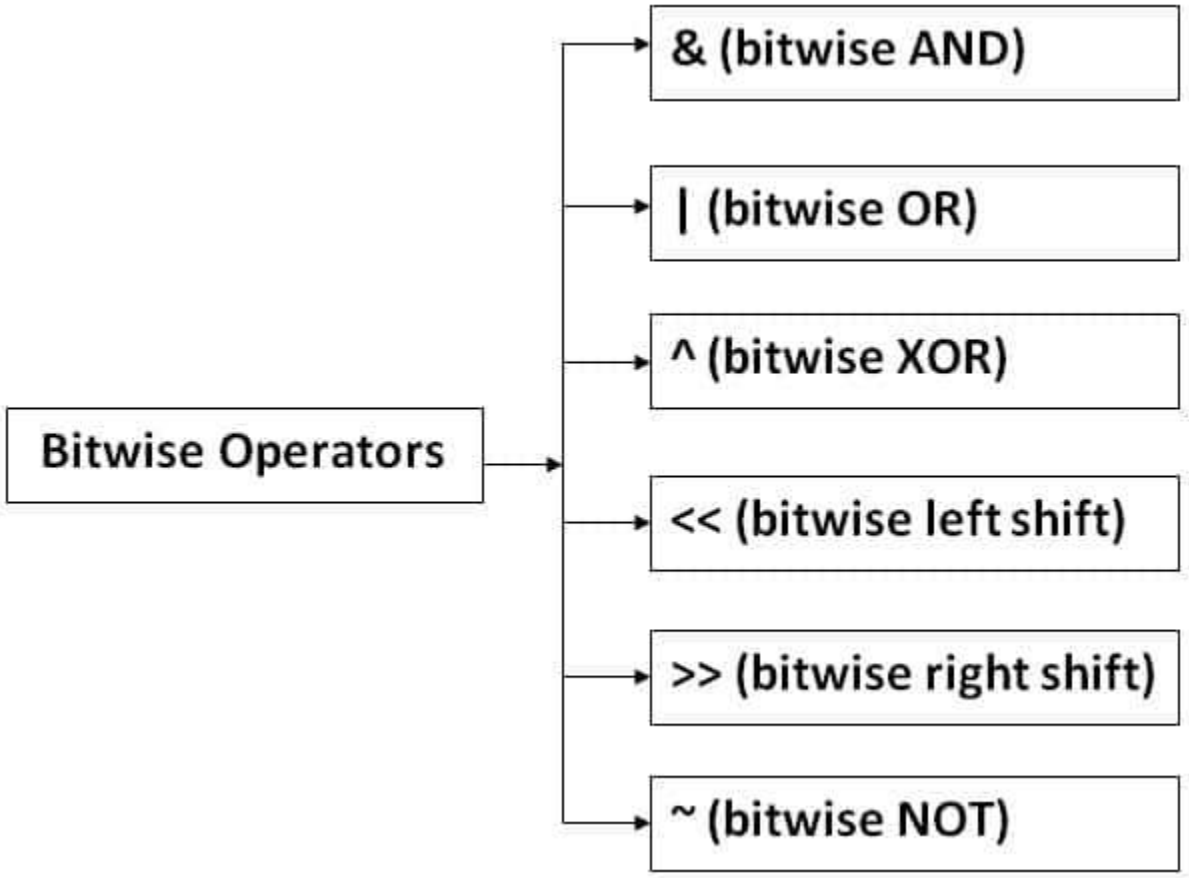
```
#include <stdio.h>

int main(void)
{
    int a, b, sum; /* Comma as a separator. */
    sum = (a=4, b=3, a+b); /* Comma as an operator. */
    printf("Sum = %d\n", sum);

    getch();
}
```

Bitwise Operators and sizeof() Operator in C

- C provides bitwise operators that let you perform logical operations on the individual bits of integer values, and shift the bits right or left.



& (bitwise AND)

- & (bitwise AND) : Bitwise AND is a binary operator and requires two operands.
- When we use the bitwise AND operator, the bit in the first operand is ANDed with the corresponding bit in the second operand. The result of AND is 1 only when the bits in both operands are 1, otherwise it is 0.

| (bitwise OR)

- | (bitwise OR) : Bitwise OR is a binary operator and requires two operands.
- When we use the bitwise OR operator, the bit in the first operand is ORed with the corresponding bit in the second operand. The result of OR is 0 only when the bits in both operands are 0, otherwise it is 1.

^ (bitwise XOR)

- ^ (bitwise XOR) : Bitwise XOR is a binary operator and requires two operands.
- When we use the bitwise XOR operator, the bit in the first operand is XORed with the corresponding bit in the second operand. The result of XOR is 1, if bits of both operands have different value, otherwise it is 0.

<< (bitwise left shift)

- << (bitwise left shift) : Bitwise left shift operator is used for shifting the bits left.
- It requires two operands; the left operand is the operand whose bits are shifted and the right operand indicates the number of bits to be shifted.

>> (bitwise right shift)

- >> (bitwise right shift) : Bitwise right shift operator is used for shifting the bits right.
- It requires two operands; the left operand is the operand whose bits are shifted and the right operand indicates the number of bits to be shifted.

~ (bitwise NOT)

- ~ (bitwise NOT) : Bitwise NOT (One's complement) operator is an unary operator (works on only one operand).
- It changes 1 to 0 and 0 to 1.

Example

```
#include <stdio.h>

int main(void)
{
    unsigned int a = 4, b = 5; /* a = 4(00000100), b = 5(00000101) */

    printf("a = %d, b = %d¥n", a, b);

    printf("a&b = %d¥n", a&b); /* The result is 00000100 */
    printf("a|b = %d¥n", a|b);  /* The result is 00000101 */

    printf("a^b = %d¥n", a^b); /* The result is 00000001 */

    printf("~a = %d¥n", a = ~a);   /* The result is 11111011 */
    printf("b<<1 = %d¥n", b<<1);  /* The result is 00001010 */
    printf("b>>1 = %d¥n", b>>1);  /* The result is 00000010 */
    getch().;
}
```

sizeof() Operator

- sizeof is an unary operator that gives the size of its operand in terms of bytes.
- The operand can be a variable, constant or any datatype(int, float, char etc).
- For example, sizeof(char) gives the size occupied by a char data type.

Example- sizeof() operator

```
#include <stdio.h>

int main(void)
{
    printf("Size of int = %u\n", sizeof(int));
    printf("Size of char = %u\n", sizeof(char));
    printf("Size of float = %u\n", sizeof(float));
    printf("Size of double = %u\n", sizeof(double));

    return 0;
}
```

```
Size of int = 2
Size of char = 1
Size of float = 4
Size of double = 8
```

Assignment Operator in C

- In C language, the assignment operator is responsible for assigning values to the variables.

  They are of two types:
- Simple assignment operator
- Compound assignment operators

Simple assignment Operator

- The simple assignment operator (=) causes the value of the right hand side operand to be stored in the left hand side operand.
- The operand on the left hand side should be a variable, while the operand on the right hand side can be any variable, constant or expression.
- Here are some examples of assignment expressions:

```
x = 7;              /* 7 is assigned to x */
y = 3;              /* 3 is assigned to y */
z = x + y + 10; /* Value of expression x+y+10 is
assigned to z */
x = y;
```

Compound Assignment Operators

- Compound-assignment operators provide a shorter syntax for assigning the result of an **arithmetic** or **bitwise** operator.
- They perform the operation on the two operands before assigning the result to the first operand.

Arithmetic compound assignment operators

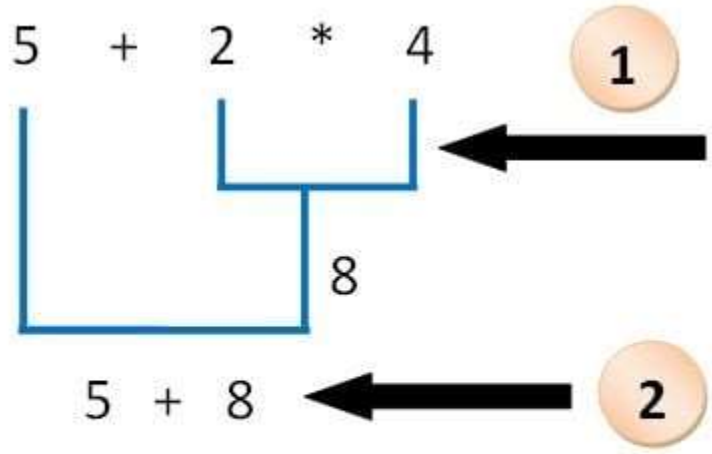
Operator	Symbol	Description
Addition	+=	x += 5 is equivalent to x = x + 5
Subtraction	-=	y -= 5 is equivalent to y = y - 5
Multiplication	*=	z *= 5 is equivalent to z = z *5
Division	/=	a /= 5 is equivalent to a = a / 5
Modulus	%=	b %= 5 is equivalent to b = b % 5

Example

```
#include <stdio.h>
void (void)
{
   int x=10, y=4;
   x += y;
   printf("+= Operator Example, Value of x = %d ¥n", x);
   x -= y;
   printf("-= Operator Example, Value of x = %d ¥n", x);
   x *= y;
   printf("*= Operator Example, Value of x = %d ¥n", x);
   x /= y;
   printf("/= Operator Example, Value of x = %d ¥n", x);
   x %= y;
   printf("%%= Operator Example, Value of x = %d ¥n", x);
 getch();
   }
```

Operator Precedence and Associativity in C

- **Operator precedence** determines which operator is performed first in an expression with more than one operators with different precedence.
- For example, x = 5 + 2 * 4;



Ans : 13

Operator	Description	Precedence level	Associativity
( ) [ ] . -> ++ --	Parentheses: grouping or function call Brackets (array subscript) Dot operator (Member selection via object name) Arrow operator(Member selection via pointer) Postfix increment/decrement	1	Left to Right
+ - ++ -- ! ~ * & (datatype) sizeof	Unary plus Unary minus Prefix increment/decrement Logical NOT One's complement Indirection Address (of operand) Type cast Determine size in bytes on this implementation	2	Right to Left
* / %	Multiplication Division Modulus	3	Left to Right
+ -	Addition Subtraction	4	Left to Right
<< >>	Left shift Right shift	5	Left to Right
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	6	Left to Right
== !=	Equal to Not equal to	7	Left to Right
&	Bitwise AND	8	Left to Right
^	Bitwise XOR	9	Left to Right
|	Bitwise OR	10	Left to Right
&&	Logical AND	11	Left to Right
||	Logical OR	12	Left to Right
? :	Conditional operator	13	Right to Left
= *= /= %= += -= &= ^= != <<= >>=	Assignment operators	14	Right to Left
,	Comma operator	15	Left to Right