Data Type and Operators

Data Type

Primary

- It is also known as primitive or fundamental data type.
- It is built-in datatype which are predefined in the C-language.
    - Integer (Store the whole numbers without decimal values. eg- 5,-6,etc.)
    - Character (Store only a single character. eg - ‘A’,’a’)
    - Float (Store floating-point numbers. eg - 2.5, -3.1)
    - Double (Stores large floating-point numbers.

Data Type

- Derived (Derived from the primitive or built-in datatypes)
  - Array (Collection of items of same data type)
  - Pointers (Stores the address of another variable)
  - Functions (Block of code)

- User Defined
  - Structure (Collection of variables under single name)
  - Union (Collection of variables under single name)
  - Enumeration (Used to assign names to integral constants)

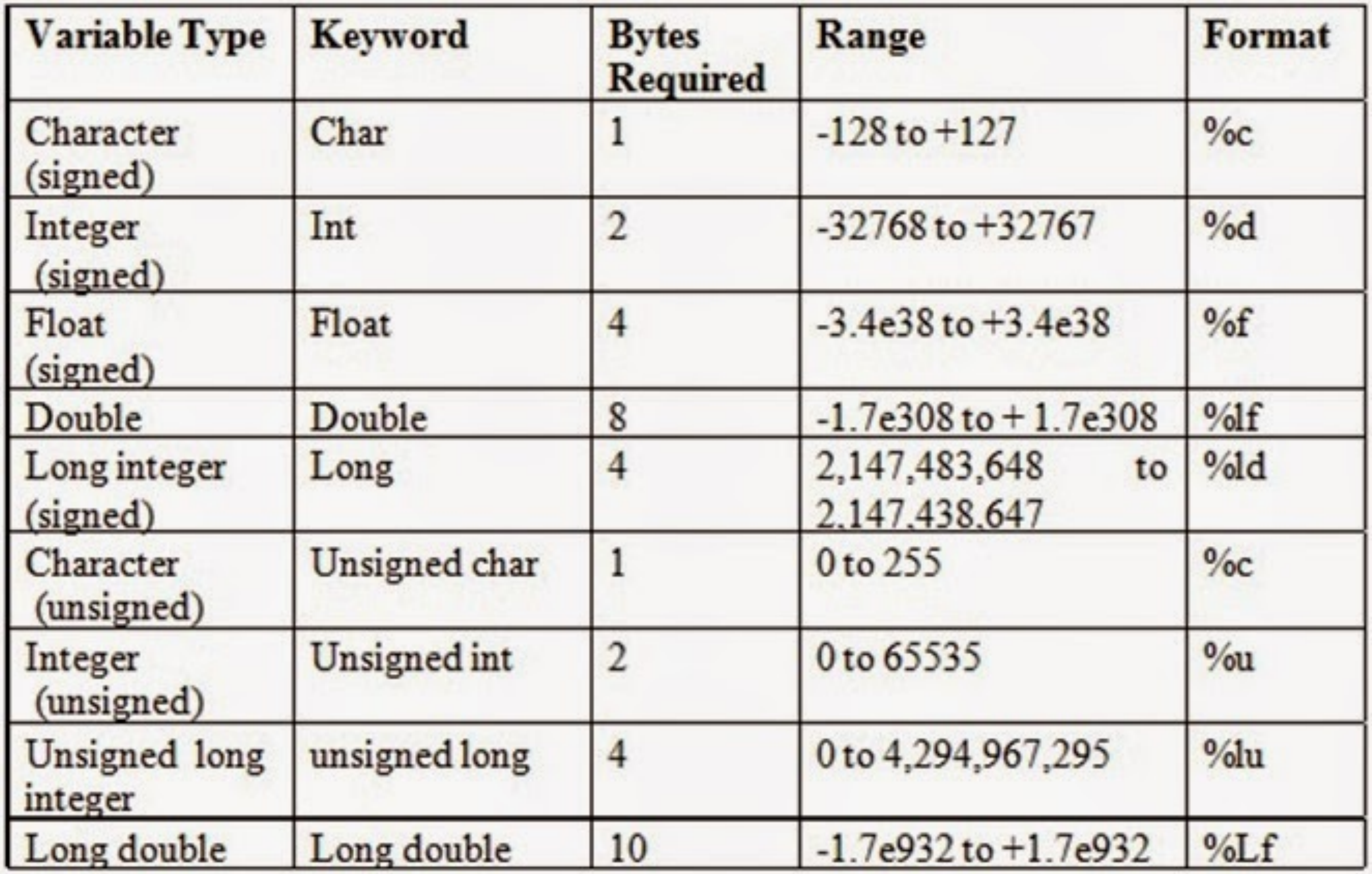
Modifiers

- Modifiers are keywords in c which changes the meaning of basic data type.
- It specifies the amount of memory space to be allocated for a variable.
- Modifiers are prefixed with basic data types to modify the memory allocated for a variable.
- There are two types of modifiers in C:
  - **Size specifiers**— short and long
  - **Sign specifiers**— signed and unsigned

Modifiers

- **short**
  - It limits user to store small integer values.
  - It can be used only on int data type.
  - Syntax: short int variablename = 18;
- **long**
  - It allows user to stores very large number.
  - long int variablename = 827337203685421584;
- **signed**
  - It says that user can store negative and positive values.
  - It is default modifier of int and char data type if no modifier is specified.
  - Syntax: signed int variablename = -544;
- **unsigned**
  - Store only positive values in the given data type (int and char).
  - Syntax: unsigned int variablename = 486;

Format Specifier



Variable Type	Keyword	Bytes Required	Range	Format
Character (signed)	Char	1	-128 to +127	%c
Integer (signed)	Int	2	-32768 to +32767	%d
Float (signed)	Float	4	-3.4e38 to +3.4e38	%f
Double	Double	8	-1.7e308 to + 1.7e308	%lf
Long integer (signed)	Long	4	2,147,483,648 to 2,147,438,647	%ld
Character (unsigned)	Unsigned char	1	0 to 255	%c
Integer (unsigned)	Unsigned int	2	0 to 65535	%u
Unsigned long integer	unsigned long	4	0 to 4,294,967,295	%lu
Long double	Long double	10	-1.7e932 to +1.7e932	%Lf

Qualifiers

- Qualifiers are used to change the properties of the existing variables.
- There are three types of qualifiers.
  - const
  - volatile
  - restrict

Qualifiers

- **const**
  - It is used to declare a variable to be read only
  - it can be declared using the keyword const.
  - Syntax: const data-type constant-name = value;

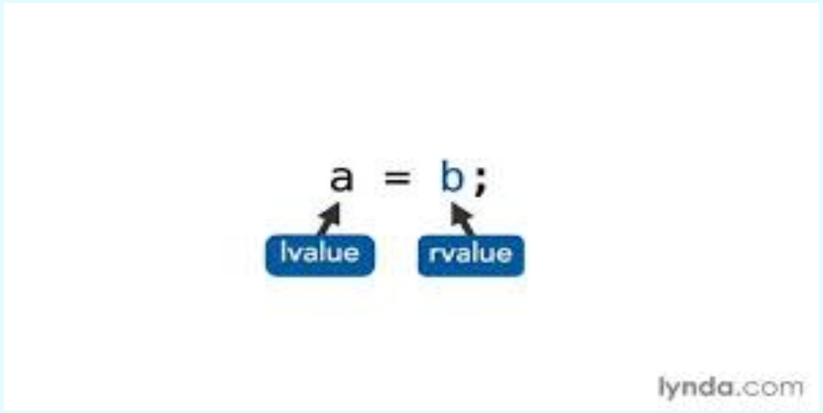
    data-type const constant-name = value;

Key Words

- **Associativity :**The associativity of operators determines the order in which operators of equal precedence are evaluated when they occur in the same expression. Most operators have a left-to-right associativity, but some have right-to-left associativity.
- **Precedence :**The precedence of operators determines the order in which different operators are evaluated when they occur in the same expression. Operators of higher precedence are applied before operators of lower precedence.

Key Words

- **L-value:** An l-value is an expression to which a value can be assigned.
- **R-value :**An r-value can be defined as an expression that can be assigned to an l-value.

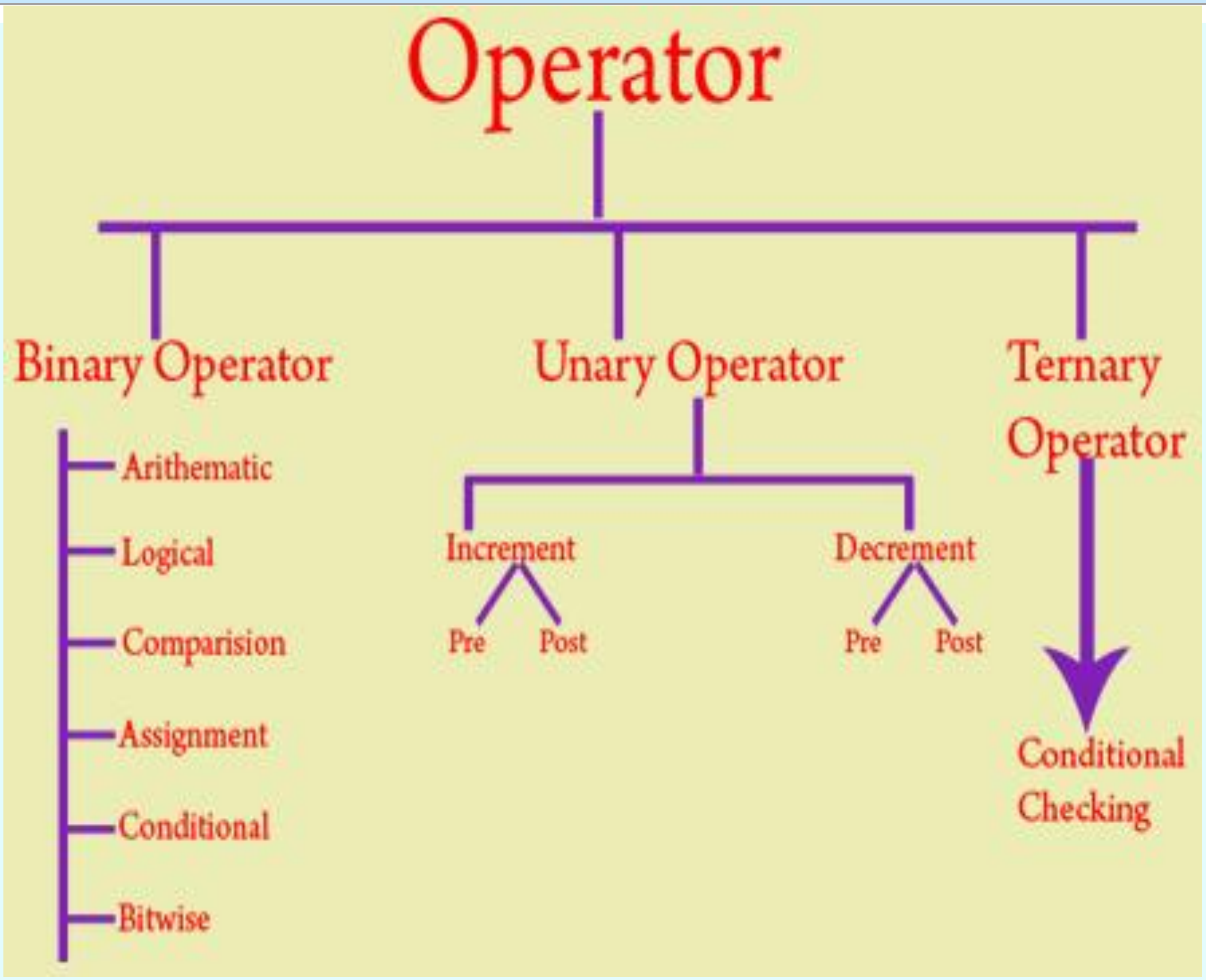


- **Token:** A token is one or more symbols understood by the compiler that help it interpret your code.
- **Whitespace Space, newline, tab character and comment are** collectively known as whitespace.

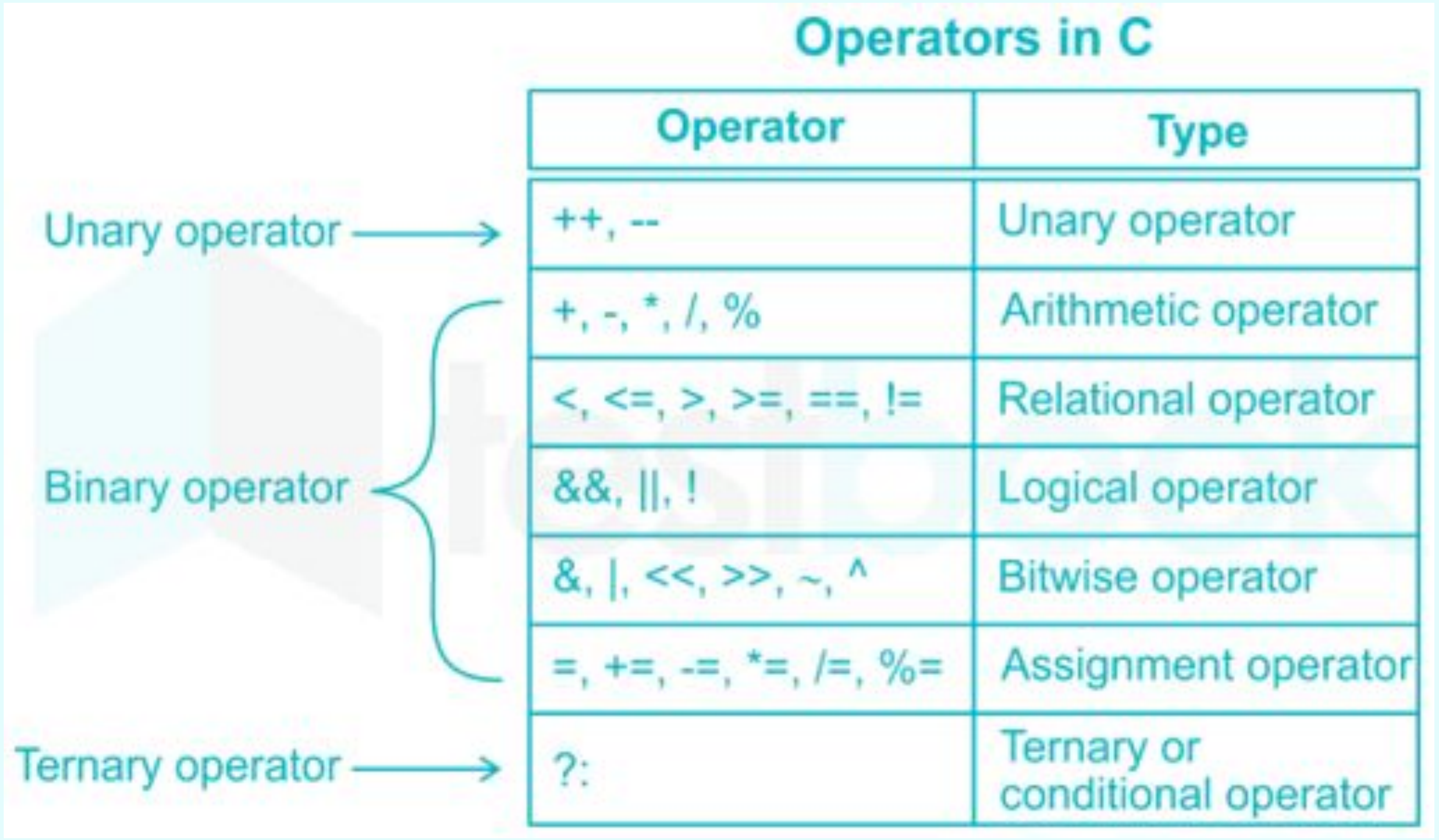
Key Words

- **Operator:** It is a symbol that is used to perform operations.
- **Operand :** An operand may any numerical value, variables or constant on which program makes an operation.
- **Expression:** An expression is a combination of operands and operators.

  ex- Z = X + Y



Unary Operator - Operates on one operand
Binary Operator - Operates on two operands
Ternary operator - Operates on three operands


Unary operator
Binary operator
Ternary operator

Operators in C

Operator	Type
++, --	Unary operator
+, -, *, /, %	Arithmetic operator
<, <=, >, >=, ==, !=	Relational operator
&&, ||, !	Logical operator
&, |, <<, >>, ~, ^	Bitwise operator
=, +=, -=, *=, /=, %=	Assignment operator
?:	Ternary or conditional operator

A-2) Unary Operators

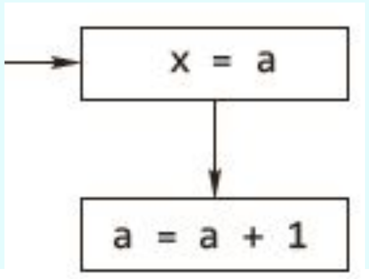
- ***Unary operators:*** The **unary operators** operate on a single operand and following are the examples of **Unary operators**:.
- ***Unary increment and decrement operators*** '++' and '--' operators increment or decrement the value in a variable by 1.
- ***Basic rules for using ++ and – – operators:***

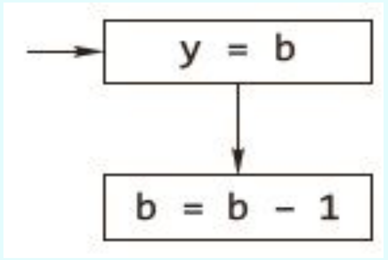
✔ The operand must be a variable but not a constant or an expression.
✔ The operator ++ and -- may precede or succeed the operand.

Postfix

Postfix:

- **(a) x = a++;**
  - ✔ First action: store value of a in memory location for variable x.
  - ✔ Second action: increment value of a by 1 and store result in memory location for variable a.
- **(b) y = b−−;**
  - ✔ First action: put value of b in memory location for variable y.
  - ✔ Second action: decrement value of b by 1 and put result in memory location for variable b.

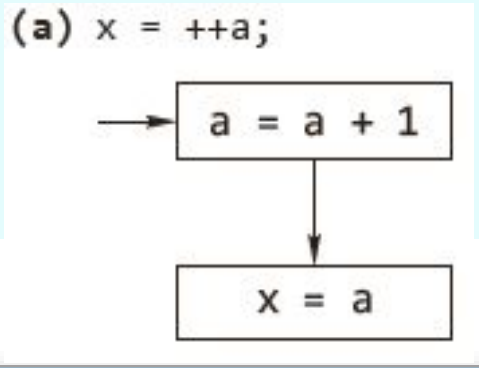


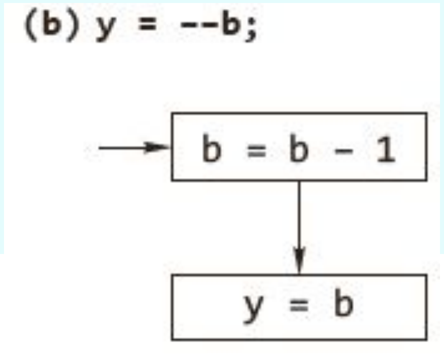


Prefix

Prefix :

- **(a) x = ++a;**
  - *First action: increment value of a* by 1 and store result in memory location for variable a.
  - *Second action: store value of a* in memory location for variable x.
- **(b) y = --b;**
  - *First action: decrement value of b* by 1 and put result in memory location for variable b.
  - *Second action: put value of b in memory location for variable y.*





Example

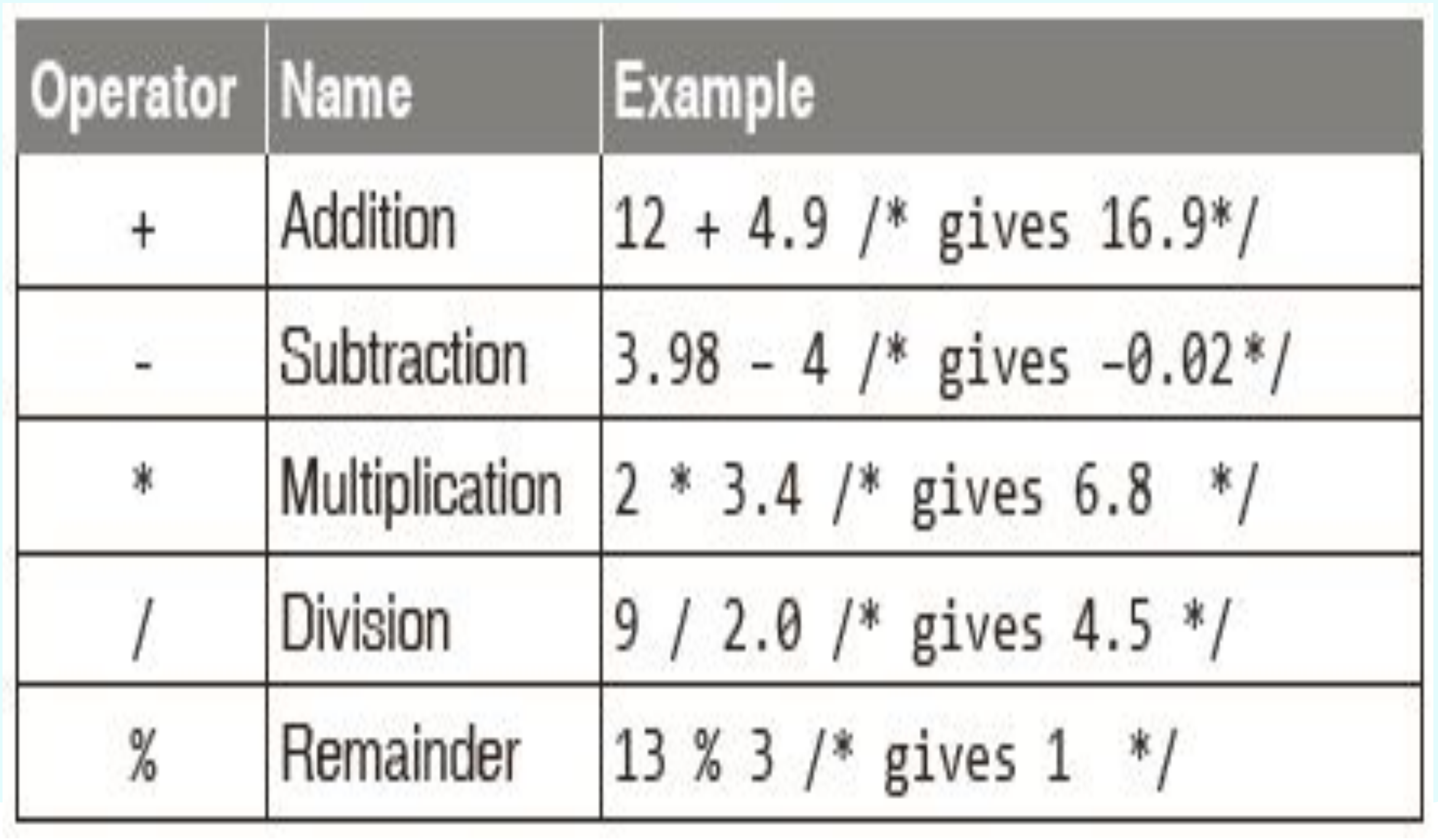
```c
#include <stdio.h>
int main ()
{
    int x, y, a, b;
    a = 10, b = 20;
    x = ++a;
    printf (" Pre Increment Operator");
    printf (" \n The value of x is %d.", x);
    printf (" \n The value of a is %d.", a);
    y = b++;
    printf (" \n\n Post Increment Operator");
    printf (" \n The value of y is %d.", y);
    printf (" \n The value of b is %d.", b);
    return 0;
}
```

Output

```
Pre Increment
Operator
The value of x is 11.
The value of a is 11.

Post Increment
Operator
The value of y is 20.
The value of b is 21.
```

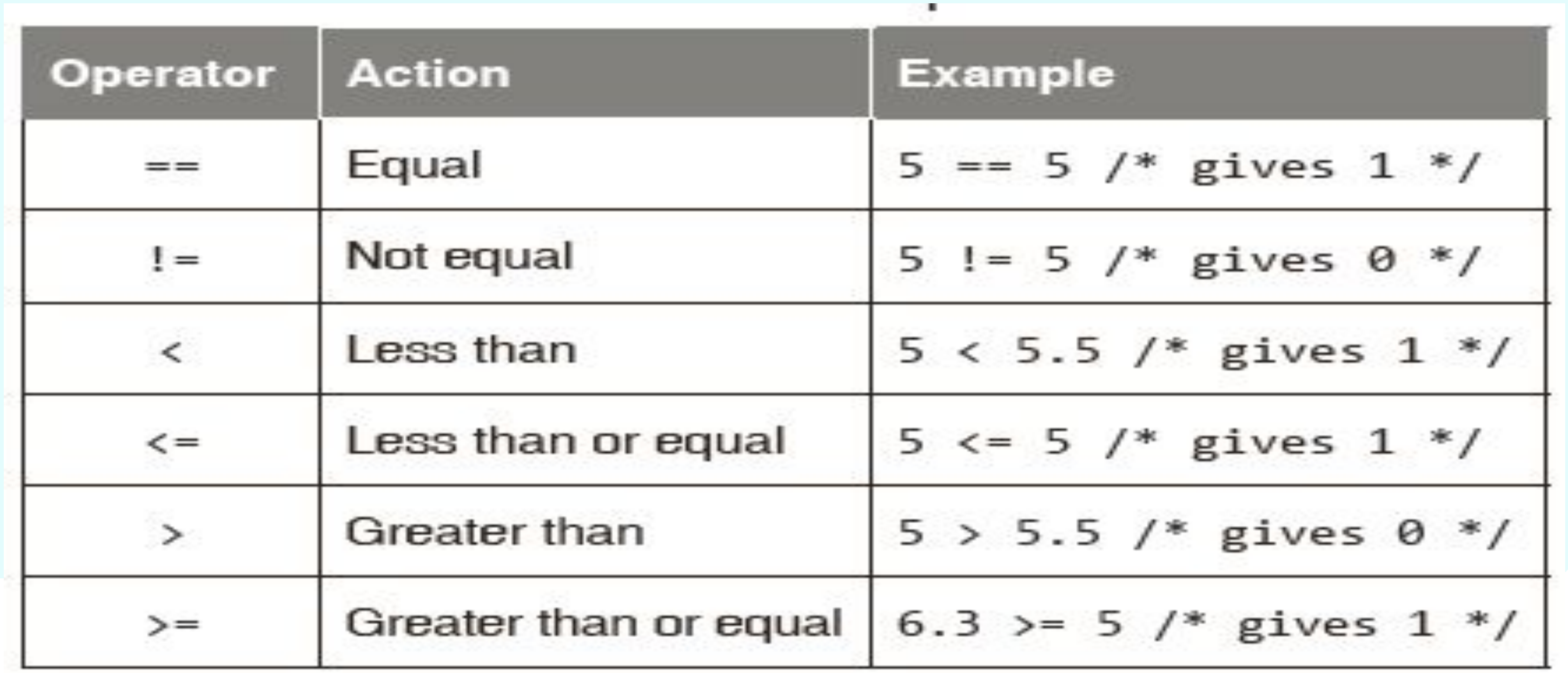
A) Arithmetic Operators



Operator	Name	Example
+	Addition	12 + 4.9 /* gives 16.9*/
-	Subtraction	3.98 - 4 /* gives -0.02*/
*	Multiplication	2 * 3.4 /* gives 6.8 */
/	Division	9 / 2.0 /* gives 4.5 */
%	Remainder	13 % 3 /* gives 1 */

B) Relational Operators

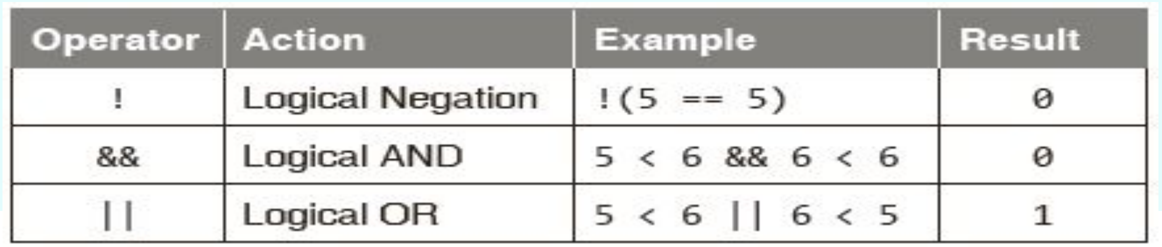
- C provides six relational operators for comparing numeric quantities. Relational operators evaluate to 1, representing the *true outcome, or* 0, representing the *false outcome.*



Operator	Action	Example
==	Equal	`5 == 5 /* gives 1 */`
!=	Not equal	`5 != 5 /* gives 0 */`
<	Less than	`5 < 5.5 /* gives 1 */`
<=	Less than or equal	`5 <= 5 /* gives 1 */`
>	Greater than	`5 > 5.5 /* gives 0 */`
>=	Greater than or equal	`6.3 >= 5 /* gives 1 */`

C) Logical Operators

- C provides three logical operators for forming logical expressions. Like the relational operators, logical operators evaluate to 1 or 0.
  - Logical negation is a unary operator that negates the logical value of its single operand. If its operand is non-zero, it produces 0, and if it is 0, it produces 1.
  - Logical AND produces 0 if one or both its operands evaluate to 0. Otherwise, it produces 1 (both true in this case of 1).
  - Logical OR produces 0 if both its operands evaluate to 0. Otherwise , it produces 1 (if any one is true).



Operator	Action	Example	Result
!	Logical Negation	`!(5 == 5)`	0
&&	Logical AND	`5 < 6 && 6 < 6`	0
||	Logical OR	`5 < 6 || 6 < 5`	1

Bitwise Operators

- C provides six bitwise operators for manipulating the individual bits in an integer quantity.
- Bitwise operators expect their operands to be integer quantities and treat them as bit sequences.
-
    - Bitwise *negation is a unary operator that complements the bits in* its operands.
    - Bitwise AND compares the corresponding bits of its operands and produces a 1 when both bits are 1, and 0 otherwise.
    - Bitwise OR compares the corresponding bits of its operands and produces a 0 when both bits are 0, and 1 otherwise.
    - Bitwise *exclusive or compares the corresponding bits of its operands and* produces a 0 when both bits are 1 or both bits are 0, and 1 otherwise.

Bitwise Operators

OP	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 0000 1111

A = 0011 1100 (60)
B = 0000 1101 (13)

Example

```
#include <stdio.h>

int main()
{
   int a = 12, b = 25;
   printf("Bitwise AND  Output = %d\n", a & b);
   printf("Bitwise OR Output = %d\n", a | b);
   printf("Bitwise EXOR Output = %d\n", a ^ b);
   printf("Complement Output = %d\n", ~a);
   printf("Right Shift Output = %d\n", a>>2);
   printf("Left Shift Output = %d\n", a<<2);
   return 0;
}
```

D) Assignment Operators

Oper ator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
−=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C −= A is equivalent to C = C − A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A

Example

```
#include <stdio.h>
int main()
{
int a = 21;
int c ;
c = a; printf("Line 1 - = Operator Example, Value of c = %d\n", c );
c += a; printf("Line 2 - += Operator Example, Value of c = %d\n", c );
c -= a; printf("Line 3 - -= Operator Example, Value of c = %d\n", c );
c *= a; printf("Line 4 - *= Operator Example, Value of c = %d\n", c );
c /= a; printf("Line 5 - /= Operator Example, Value of c = %d\n", c );
c = 200;
c %= a; printf("Line 6 - %= Operator Example, Value of c = %d\n", c );
return 0;
}
```

```
Line 1 - = Operator Example, Value of c = 21
Line 2 - += Operator Example, Value of c = 42
Line 3 - -= Operator Example, Value of c = 21
Line 4 - *= Operator Example, Value of c = 441
Line 5 - /= Operator Example, Value of c = 21
Line 6 - %= Operator Example, Value of c = 11
```

Conditional Operators

- The conditional operator has three expressions (ternary).
  - It has the general form
  - **expression1 ? expression2 : expression3**
  - First, expression1 is evaluated; it is treated as a logical condition.
  - If the result is non-zero, then expression2 is evaluated and its value is the final result. Otherwise, expression3 is evaluated and its value is the final result.
- For example,int m = 1, n = 2, min;
- min = (m < n ? m : n);

Conditional Operators

```
#include <stdio.h>

int main()
{
int a , b;
a = 10;
printf( "Value of b is %d\n", (a == 1) ? 20: 30 );
printf( "Value of b is %d\n", (a == 10) ? 20: 30);
return 0;
}
```

This will produce following result:

```
Value of b is 30
Value of b is 20
```

Comma Operators

- This operator allows the evaluation of multiple expressions, separated by the comma, from left to right in order and the evaluated value of the rightmost expression is accepted as the final result. The general form of an expression using a comma operator is
- Expression M = (expression1, expression2, …,expression N);
- where the expressions are evaluated strictly from left to right and their values discarded, except for the last one, whose type and value determine the result of the overall expression.

Comma Operators

```
int main()
 {
int num1 = 1, num2 = 2;              /*separator*/
int res;
res = (num1, num2);                  /* operator*/
printf("%d", res);
return 0;
}
```

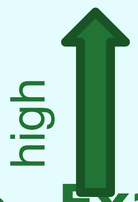
Sizeof Operators

- C provides a useful operator, sizeof, for calculating the size of any data item or type.
- It takes a single operand that may be a type name (e.g., int) or an expression (e.g.,100) and returns the size of the specified entity in bytes.

```
#include <stdio.h>
int main()
{
    printf("%lu\n", sizeof(char));
    printf("%lu\n", sizeof(int));
    printf("%lu\n", sizeof(float));
    printf("%lu", sizeof(double));
    return 0;
}
```

Precedence & Associativity

- **Precedence of operators**
- If more than one operators are involved in an expression, C language has a predefined rule of priority for the operators. This rule of priority of operators is called operator precedence.
- In C, precedence of
  - Arithmetic operators(*, %, /, +, -)
  - Relational operators(==, !=, >, <, >=, <=)
  - Logical operators(&&, || and !).



- **Example of precedence**
- (1 > 2 + 3 && 4)
- This expression is equivalent to: ((1 > (2 + 3)) && 4)
- i.e, (2 + 3) executes first resulting into 5 then, first part of the expression (1 > 5) executes resulting into 0 (false) then, (0 && 4) executes resulting into 0 (false)
- **Output - 0**

Precedence & Associativity

Associativity of operators

If two operators of same precedence (priority) is present in an expression, Associativity of operators indicate the order in which they execute.

Example of associativity

1 == 2 != 3

Here, operators == and != have same precedence.

The associativity of both == and != is left to right, i.e, the expression on the left is executed first and moves towards the right.

Thus, the expression above is equivalent to :

((1 == 2) != 3)

i.e, (1 == 2) executes first resulting into 0 (false)

then, (0 != 3)

executes resulting into 1 (true)

Precedence & Associativity

Precedence

Example : Associativity of Operator

```
#include <stdio.h>
void main()
{
   int a = 20, b = 10,  c = 15, d = 5, e;
   e = (a + b) * c / d;
   printf("Result is : %d", e );
   return 0;
}
```

WAP to perform the addition of two integers & display the result.

Code

```
#include<stdio.h>
void main()
{
int a, b, c;
printf("Enter two numbers to add :\n");
scanf("%d%d",&a,&b);
c = a + b;
printf("\n The addition of %d and %d is %d", a,b,c);
}
```

Assignment

- WAP to display the following message by using multiple printf statement.

  If The End Is Good,
  Then It Is Good,
  Whatever Be The Means.

- WAP to display the above message using single printf statement.
- WAP to print your BIO-DATA (Name, Regd.no”, Branch, JEE Rank, Gender, Phone no., Address etc.) using printf statement.