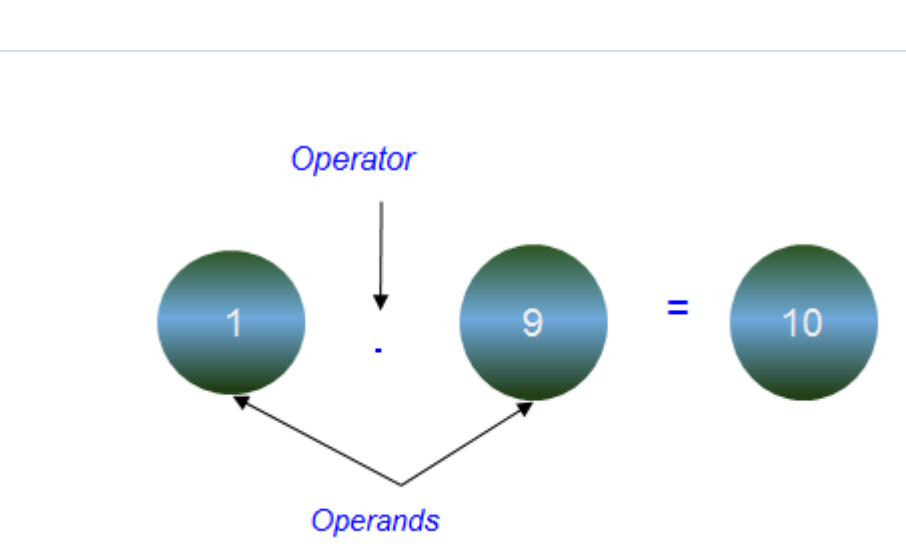
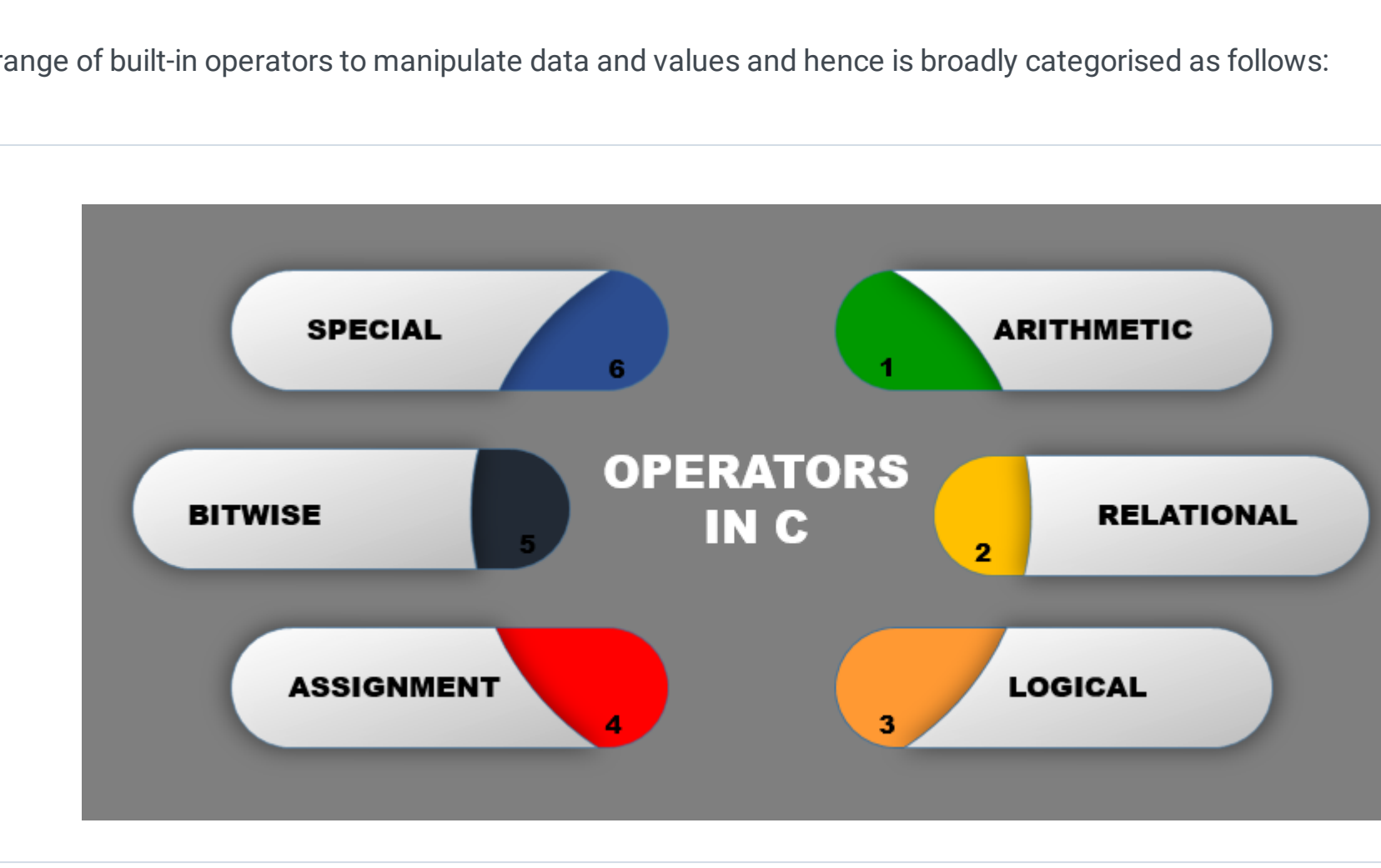
Operators in C

In this tutorial you will grasp the skill to work with different operators used in C to perform logical and arithmetical calculations with the aid of simple and easy examples. Operators are unique symbols that perform some sort of computation. The objects or values on which operators act are known as operands and the combination of operators and operands are termed as expressions in c.



C language support a wide range of built-in operators to manipulate data and values and hence is broadly categorised as follows:



Arithmetic Operators:

Like real life mathematics, arithmetic operators of C do the job of division, multiplication, addition, and subtraction. The involved operators are '/', '*', '+' and '-' respectively. Except these, there are other three operators modulus, increment and decrement operator.Modulus or '%' outputs the remainder of any division of numbers.

Operator	Meaning	Description	Example
+	Addition	Adds two operands or unary plus	10+2=12
-	Subtraction	Subtracts right operand from left operand or unary minus	10-2=8
*	Multiplication	Multiplies two operands	10*2=20
/	Division	Divides left operand by right operand	10/2=5
%	Modulus	Remainder after division	10%2=0
++	Increment	increases value by one unit	++a or a++ = a+1
--	Decrement	decreases value by one unit	--b or b-- = b-1

Example of Arithmetic Operators

```
#include <stdio.h>
int main()
{
    int x = 7,y = 3, z;

    z = x+y;
    printf("Sum: x+y = %d \n",z);
    z = x-y;
    printf("Difference: x-y = %d \n",z);
    z = x*y;
    printf("Product: x*y = %d \n",z);
    z = x/y;
    printf("Quotient: x/y = %d \n",z);
    z = x%y;
    printf("Remainder: x%y = %d \n",z);

    return 0;
}
```

```
Sum: x+y = 10
Difference: x-y = 4
Product: x*y = 21
Quotient: x/y = 2
Remainder: xy = 1
```

Increment and Decrement Operators

Increment operator '++' increases the value of integer by one unit, whereas the decrement operator '--' decreases the same by one unit. These operators can be either prefixed or postfixed with the operand and are used extensively in a different type of loops in C .

Example of Increment and Decrement Operators

```
Example of Increment and Decrement Operators
#include <stdio.h>
int main()
{
    int x = 7,y = 3;

    printf("Increment: ++x = %d \n",++x);
    printf("Decrement: --y = %d \n",--y);
    return 0;
}
```

Output:

```
Increment: ++x = 8
Decrement: --y = 2
```

Relational Operators:

Relational operators do compare the data to give binary outputs i.e. True or False. Here are the six operators demonstrated using the two operands a and b.

Operator	Meaning	Description	Examples
==	Equal to	Returns True if two operands are equal	a==b
!=	Not Equal to	Returns True if two operands are not equal	a!=b
>	Greater than	Returns True if left operands is greater than the right	a>b
<	Less than	Returns True if left operand is less than the right	a<b
>=	Greater than or equal to	Returns True if left operands is greater than or equal to the right	a>=b
<=	Less than or equal to	Returns True if left operand is less than or equal to the right	a<=b

Example of Relational Operators

```
#include <stdio.h>
int main()
{
    int x = 7,y = 3,z;
    z = x>y;
    printf("%d is greater than %d is %d\n",x,y,z);
    z = x<y;
    printf("%d is greater than %d is %d\n",x,y,z);
    z = x<=y;
    printf("%d is greater than %d is %d\n",x,y,z);
    z = x>=y;
    printf("%d is greater than %d is %d\n",x,y,z);
    z = x!=y;
    printf("%d is greater than %d is %d \n",x,y,z);

    return 0;
}
```

Output:

```
7 is greater than 3 is 1
7 is less than 3 is 0
7 is greater than 3 is 1
7 is greater than 3 is 0
7 is greater than 3 is 0
7 is greater than 3 is 1
```

Logical Operators:

These operators perform binary operations to process data at machine level (logic gates like AND, OR, NOR, NAND etc.). If the result is true, it is denoted by returning '1'. The negative result is expressed by '0'. Here is the description of three basic logical operators in C which are extensively used in decision making.

Operator	Meaning	Description	Example
&&	Logical AND/ Conjunction	Returns True if and only if both statements are true	X and Y
||	Logical OR / Disjunction	Returns True if any of the statement is true	X or Y
!	Logical NOT/ Negation	Returns true if operand is a negation	not X

For better understanding of Logical operators you should know about the truth table.

A	B	A && B	A || B	~A
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Truth Table for Logical Operators

The complement of AND is called NAND and OR is called NOR. They are used in conjunction with other operators like A! &B, A! =B etc.

Example of Logical Operators

```
#include <stdio.h>
int main()
{
    int x = 7,y = 3,z;
    z = (x==7)&&(x>y);
    printf("(x==7)&&(x>y) is %d\n",z);
    z = (x==7)||(x>y);
    printf("(x==7)||(x>y) is %d\n",z);
    z = (x!=7)||(y!=3);
    printf("(x!=7)&&(y!=) is %d\n",z);
    return 0;
}
```

Output:

```
(x==7)&&(x>y) is 1
(x==7)||(x>y) is 1
(x!=7)&&(y!=) is 0
```

Common bitwise operators are listed in the below table.Binary representation of 3 is 0000 0011 and that of 4 is 0000 0100.

Operators	Meaning	Description	Example
&	Binary AND	Result is 1 if both operands are true otherwise 0	3&4 =0 0000 0011 & 0000 0100 0000 0000
|	Binary OR	Result is 1if any one operand is true otherwise 0	3|4=7 0000 0011 | 0000 0100 0000 0111
^	Binary XOR	Result is 1 if it's both operands are different and 0 if both operands are same	3^4=7 0000 0011 ^ 0000 0100 0000 0111
~	Binary Ones Complement	Result is the negation of the operand	~3= -(4) ~ 0000 0011 1111 1100
<<	Binary Left Shift	Aligns the bits to the left	3<<2 = 12
>>	Binary Right Shift	Aligns the bits to the right	3>>2 = 0

The first operator '&' is of AND type which copies any bit to result if the bit exists in both operands. '|' functions as OR operator. It replicates a bit,- if it exists in either or both of the operands. '^' denotes XOR operation. It is positive if it exists in any of the operands but not the both. Except these, there is a complement operator which has the effect of a flipping bit. It is denoted by the '~' symbol.

Example of Bitwise Operators

```
#include <stdio.h>
int main()
{
    int x = 3,y = 4,z;
    z = x&y;
    printf("x&y is %d\n",z);
    z = x|y;
    printf("x&y is %d\n",z);
    z = x^y;
    printf("x&y is %d\n",z);
    printf("x&y is %d\n",~x);
    return 0;
}
```

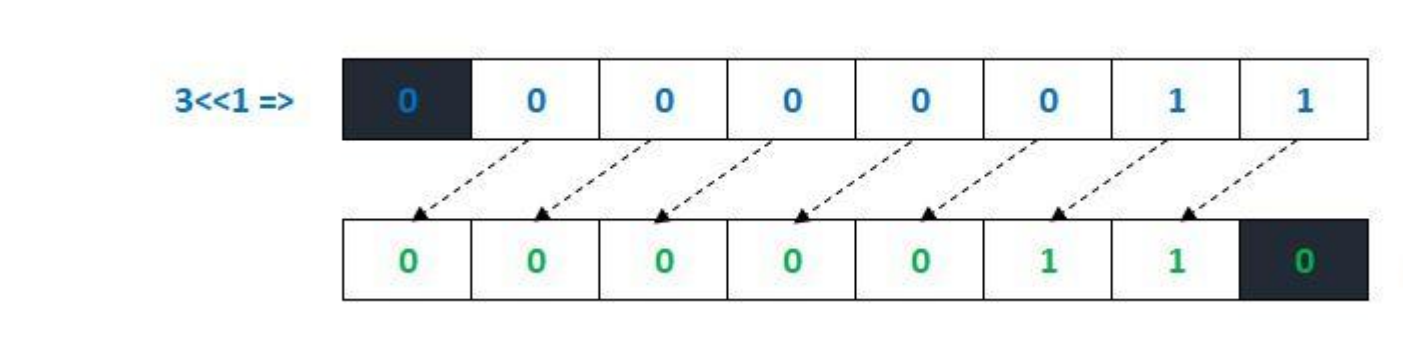
Output:

```
x&y is 0
x|y is 7
x^y is 7
~x is -4
```

Shift Operators in C

Bitwise Shift operators '<<' and '>>' are called binary left shift and the right shift operators respectively. The value of the operands is the left side of moved by the amount specified on the right-hand side of the operator.

Bitwise Shift Left Representation



Bitwise Shift Left Representation

Example of Bitwise Shift Operators

```
#include <stdio.h>
int main()
{
    int x = 3;

    printf("x<<1 is %d\n",x<<1);
    printf("x<<2 is %d\n",x<<2);
    printf("x<<2 is %d\n\n",x<<3);

    printf("x>>1 is %d\n",x>>1);
    printf("x>>2 is %d\n",x>>2);
    printf("x>>3 is %d\n",x>>3);

    return 0;
}
```

Output:

```
x<<1 is 6
x<<2 is 12
x<<2 is 24

x>>1 is 1
x>>2 is 0
x>>2 is 0
```

Where do we use bitwise operators?

Since bitwise operators are used to manipulate bit level datas they are not common in the real world. However they reign the world of low level or otherwise called machine language. As we all know low level operations use the binary format of 0 and 1 to manipulate datas. Listed below are some of the areas in which bitwise operators are used.

- Primarily used in embedded system
- Data Compression - winrar,zip
- Encryption
- Networking
- Hardware manipulation
- Graphics

Assignment Operators

As its name indicates , assignment operators are used to assign values to variables. '=' (equals) is the most basic type of them assigning the value at the right-hand side to the left-hand side. C=A+B will impose the value of (A+B) to C. Below table gives you the other assignment operators used in C to perform arithmetic operations.

Operators	Example	Meaning
=	a = 10	
+=	a+=10	a=a+10
-=	a-=10	a=a-10
*=	a*=10	a=a*10
/=	a/=10	a=a/10
%=	a%=10	a=a

Example of Assignment Operators

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

    a = x;
    printf("a = %d\n", a);
    a += x;
    printf("a = %d\n", a);
    a -= x;
    printf("a = %d\n", a);
    a *= x;
    printf("a = %d\n", a);
    a /= x;
    printf("a = %d\n", a);
    a %= x;
    printf("a = %d\n", a);

    return 0;
}
```

Output:

```
a = 10
a = 20
a = 10
a = 100
a = 10
a = 0
```

Special Operators

sizeof Operator

This unary operator returns the size of the operand in bytes. For example, it will return 4 in case of integers. It is very helpful in space management in large programs.

Example of Sizeof Operator

```
#include <stdio.h>
void main()
{

int a;
int short b;
short c;
int long d;
long e;
 printf(" Size of  a is %d\n",sizeof(a));
 printf(" Size of  b is %d\n",sizeof(b));
 printf(" Size of  c is %d\n",sizeof(c));
 printf(" Size of  d is %d\n",sizeof(d));
 printf(" Size of  e is %d\n",sizeof(e));

}
```

Output:

```
Size of  a is 4
Size of  b is 2
Size of  c is 2
Size of  d is 4
Size of  e is 4
```

Conditional/ Ternary Operator(?:):

These are some uniquely defined operators in c called ternary operators. '?:' is also known as a conditional operator is widely used in decision making and routing the flow of program execution in the desired direction. The syntax is usually (exp)?A:B which means if the expression is true it returns A otherwise returns B. The following program illustrates this:

Example of Conditional Operators

```
#include <stdio.h>
void main()
{
    int x=7,z;
    z=(x==8)? "True" : "False"  ;
     printf("Value of x is 7 ? %s\n",z);

}
```

Output:

```
Value of x is 7 ? False
```

Reference or pointer Operator (&) :

Reference operator will return the address of any variable writing the variable name followed by it. You will see the use of this reference operator in the upcoming tutorial - pointers in C.

Dereference Operator(*):

Widely used operator '*' denoting pointer variables fall among this category which you will learn later in our tutorial of pointers.

Operators Precedence:

Now We have grasped the knowledge of nearly all the operators used in C. However we often fall in situations, where we have to work with different types of operators simultaneously. If we do not follow the right sequence of applying them, the program will end up crashing.

Operator precedence refers to the order of operators in which they evaluate an expression. Here is the sequence maintained by the compiler that we must abide by:

1. Postfix () [] ->
2. Unary
3. Multiplication/ division
4. Addition/subtraction
5. Shift
6. Relational
7. Equality
8. Bitwise AND
9. Bitwise XOR
10. Bitwise OR