

Number System and Operator Precedence



What is meant by Number System?

The ways of representing and working with numbers in a computer is referred to as **Number System**.

We have different types of number systems. The most commonly used ones are:

- Decimal Number System(base 10)
- Binary Number System(base 2)
- Hexadecimal Number System(base 16)
- Octal Number System(base 8)



Why do we need so many different Number Systems?

Let's look at an example from MS Excel.

Consider the relation : $x_n = (16 x_{n-1}) - 3$

Now suppose we start with $x_0 = 0.2$, thus, on physically computing the values we should get the following:

$$x_1 = (16 * 0.2) - 3 = 0.2$$

$$x_2 = (16 * 0.2) - 3 = 0.2$$

$x_3 = (16 * 0.2) - 3 = 0.2$... and so on. Hence logically, every value should be 0.2... But does the computer also give the same results??

Let's find out !



MS Excel gives wrong answers??!!

0.2	0.2
0.2	0.2
0.2	0.2
0.2	0.2
0.2	0.200001
0.200012	0.200195
0.203125	0.25
1	13

Well, there is actually a reason behind MS Excel giving such an output.

As we will study later, the CPU performs all arithmetic operations in binary system, and not all fractions have a terminating representation in Binary system, therefore, such errors do arise!

$(0.2)_{10}$ can be represented in binary system as $(0.001100110...)_{2}$

Hence, since such issues can arise with many other numbers, it's logical for us to have other types of Number Systems as well so that we can take care of such issues to a great extent.



Decimal Number System

Decimal System basically uses the digits from **0-9** to represent all numbers. That's why it is called a **base 10** system. In our everyday life, we usually work with the Decimal System only. Each digit is expressed as power of 10.

To represent a number in a number system of base 'n', we use the following notation: **(number)_n**

Hence within the decimal system we can represent the numbers as follows:
 $(99)_{10}$, $(2238)_{10}$, $(888)_{10}$



Binary Number System

Binary Number System uses **0s** and **1s** to represent numbers. That's why it's called a **base 2** system. The computer understands and interprets the data that we feed it, in binary format. Each digit is expressed as power of 2.

To represent binary numbers, we use the following format: **(number)₂**

Here are certain examples of how decimal numbers are represented in binary format:

- $(20)_{10} = (10100)_2$
- $(33)_{10} = (100001)_2$
- $(43)_{10} = (101011)_2$



Octal Number System

As the name suggests, the Octal Number System uses the digits from **0-7** to represent each number. Hence it's called a **base 8** system. Each digit in octal number system is expressed in powers of 8.

A number in octal system is represented by: **(number)₈**

Here are certain examples of how decimal numbers are represented in octal system:

$$- (470)_{10} = (726)_8$$

$$- (13)_{10} = (15)_8$$

$$- (232)_{10} = (350)_8$$



Hexadecimal Number System

Hexadecimal uses 16 different types of alpha-numeric values to represent numbers. It uses digits from **0-9** and alphabets from **A-F**. That is, the numbers after 9, are represented by letters. So **A=10**, **B=11** ... **F=15**. Hence it's a **base 16** system, where every digit is represented as a power of 16.

A number is represented in Hexadecimal format as: $(\text{number})_{16}$

Here are some examples of decimal numbers represented in Hexadecimal format:

- $(344)_{10} = (158)_{16}$
- $(2595)_{10} = (A23)_{16}$
- $(2748)_{10} = (ABC)_{16}$



Practice Exercise 1

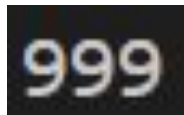
Now that we know about the 3 number systems, let's see how we can represent them in C.

Representing decimal numbers:

We represent decimal numbers using the '%d' operator.

Here's an example:

```
#include <stdio.h>
int main()
{
    int value=999;
    printf("%d\n",value);
    return 0;
}
```



999



Representing Octal Numbers:

We use the ‘%o’ operator to represent a number in Octal format. Here’s an example

```
#include <stdio.h>
int main()
{   int value=999;
    printf("%o\n",value);
    return 0;
}
```

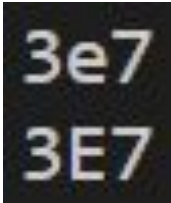
1747



Representing Hexadecimal Numbers

We can use the '%x' to represent the numbers in Hexadecimal format, with the alphabets in **lowercase**, and the '%X' operator to represent the alphabets in **uppercase**. Here's an example:

```
#include <stdio.h>
int main()
{   int value=999;
    printf("%x\n",value);
    printf("%X",value);
    return 0;
}
```



3e7
3E7



How to convert between different bases?

Decimal-to- Binary

To convert numbers from decimal to binary, the given decimal number is divided repeatedly by 2 and the remainders are noted down till we get 0 as the final quotient..

Decimal number : 17

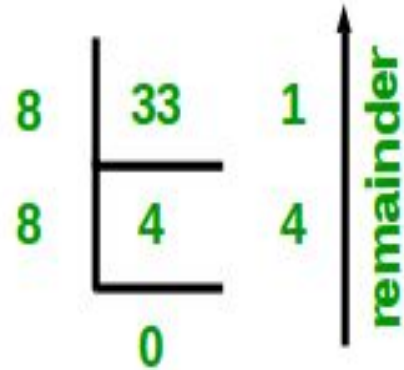
2	17	1
2	8	0
2	4	0
2	2	0
	1	

Binary number : 10001

Decimal-to-Octal

In case of decimal to octal, we divide the number by 8 and write the remainders in the reverse order to get the equivalent octal number.

Decimal Number: 33



Octal Number: 41

Decimal-to-Hexadecimal

In order to convert a decimal number to hexadecimal, we divide the decimal number by 16 because 16 is a base value of hexadecimal numbers. We keep on dividing until the quotient becomes zero. The remainders are noted down and written in the reverse order in the combined form.

Number (Division)	Quotient	Remainder
5386 / 16	336	10 = A
336 / 16	21	0
21 / 16	1	5
1 / 16	0	1

Decimal Value → Hexadecimal Value

$(5386)_{10} \rightarrow (150A)_{16}$



Why does repeated division by 2 convert Decimal to Binary?

When you repeatedly divide a decimal number by 2 and note the remainders, you are essentially breaking down the number into powers of 2.

Starting from the least significant bit (rightmost digit in binary representation), each remainder represents whether the corresponding power of 2 is present in the original decimal number. The remainders, read in reverse order, give you the binary representation of the decimal number.

Similar logic works for octal and hexadecimal too.

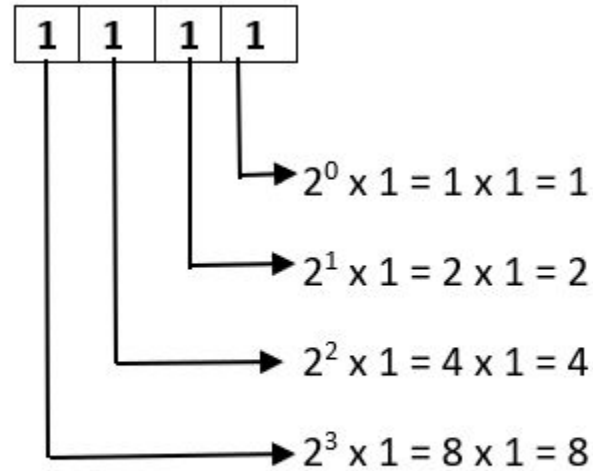
Binary-to-Decimal

Step 1: Write the binary number and count the powers of 2 from right to left (starting from 0).

Step 2: Write each binary digit(right to left) with corresponding powers of 2 from right to left, such that MSB or the first binary digit will be multiplied by the greatest power of 2.

Step 3: Add all the products in the step 2

Step 4: The answer is our decimal number.



Resultant decimal number= $1+2+4+8 = 15$



Binary-to-Octal

Method 1: Converting binary to decimal then to decimal to octal.

Method 2: Converting binary to octal by grouping.

- Group all the 0 and 1 in the binary numbers in a set of three starting from the right side (Least Significant Bit: LSB).
- Add 0's to the left (Most Significant Bit: MSB) if it doesn't form a group of three. Make sure each group must have three numbers.
- Now write the octal equivalent number for each group and the result is the number in the octal number system.



Example

$(1010111100)_2$

Making a group of three = 1 010 111 100

Adding two 0 to complete the set = 001 010 111 100

Now write the octal equivalent number = 1 2 7 4

So, $(1010111100)_2 = (1274)_8$.



Binary-to Hexadecimal

Method 1: Converting binary to decimal then to decimal to hexadecimal.

Method 2: Converting binary to hexadecimal by grouping.

- Group all the 0 and 1 in the binary numbers in a set of four starting from the right side (Least Significant Bit: LSB).
- Add 0's to the left (Most Significant Bit: MSB) if it doesn't form a group of four. Make sure each group must have four numbers.
- Now write the hexadecimal equivalent number for each group and the result is the number in the hexadecimal number system.



Example

$(111111101)_2$

$(111111101)_2 = \underline{1} \underline{1111} \underline{1101}$

$(111111101)_2 = \underline{0001} \underline{1111} \underline{1101}$ (Adding 3 zeros to left)

= 1 FD (Hexadecimal equivalent of each set)

= $(111111101)_2 = (1FD)_{16}$



Octal/Hexadecimal to Binary

Can be done by ungrouping(reverse of grouping)

$$(E9A)_{16} = (1110\ 1001\ 1010)_2$$

$$(770)_8 = (111\ 111\ 000)_2$$



Octal/Hexadecimal to Decimal

- Similar to conversion of Binary to Decimal.
- Only difference is replacing powers of 2 by powers of 8 and 16 respectively.
- Direct conversion from Octal to Hexadecimal or vice-versa is not possible



Practice Exercise 2

Convert 347 from octal to binary.



Answer

011 100 111 = 011100111 (By writing equivalent binary representation for each digit in octal)



Practice Exercise 2

Convert 1A3 from hexadecimal to binary:



Answers

0001 1010 0011 = 110100011 (By writing equivalent binary representation for each digit in hex and removing extra zeros from left)



Practice Exercise 2

Convert the 2F8 from hexadecimal to octal.



Answer

1370(By converting first from hexadecimal to binary by ungrouping and then to octal by grouping)



Operators

Operator is a character that represents a specific mathematical or logical action or process. Operators are used to perform operations on variables and values.

For eg- '+' is an arithmetic operator used to add two numbers.



Types of Operators

- Unary Operator
- Arithmetic Operator
- Relational Operator
- Assignment Operator
- Logical Operator(will be covered later)
- Bitwise Operator(will be covered later)
- Ternary/Conditional Operator(will be covered later)



Unary Operator

A unary operator is an operator that operates on a single operand.

Common examples of Unary operators are:

- Increment operator('++') -Increases the value of variable by one.
- Decrement Operator('--')-Decreases the value of a variable by one.
- Logical NOT Operator('!')- Inverts the logical state of the operand (True to false and vice-versa)

Prefix and postfix increment\decrement

In prefix increment operator increases the value of the variable before its value is used in the expression

```
int a = 5;
int b = ++a; // Prefix increment: 'a' is first incremented,
// then its value is assigned to 'b'
// Now, 'a' and 'b' are both 6
```

In postfix increment operator uses the current value of the variable in the expression and then increments the variable afterward.

```
int x = 10;
int y = x++; // Postfix increment: 'x' is first used,
// then it is incremented, and the result is assigned to 'y'
// Now, 'x' becomes 11, and 'y' is 10
```



Arithmetic Operator

Arithmetic Operators are used to perform mathematical operations on numeric values.

The basic arithmetic operators are Addition(+), Subtraction(-), Multiplication(*), Division(/) and Modulus Operator(%) (Returns the remainder).



Relational Operator

Relational operators are used to compare values and return a boolean result (True or False). These operators are mostly used in conditional statements.

These are some of the relational operators:

- Equal to('==')-Checks if the values of two operands are equal.
- Not equal to('!=')-Checks if the values of two operands are not equal.
- Greater than('>'),Less than('<')
- Greater than or equal to('>='),Less than or equal to('<=')



Assignment Operator

Assignment operators are used to assign values to the variables.

Some commonly used assignment operators are Assignment('='), Addition assignment('+='), Subtraction assignment ('-='), Multiplication assignment('*='), Division assignment('/=')



Operator Precedence

Operator Precedence is the set of rules that dictate the order in which operators are evaluated in an expression

For eg- `int result=4+2/2;`

Here the ('/') operator has higher precedence than ('+') .So the value of result will be 5 and not 3!!

Note-Parentheses '()' can be used to override precedence.

Operator	Description	Associativity
() [] . -> ++ --	Parentheses or function call Brackets or array subscript Dot or Member selection operator Arrow operator Postfix increment/decrement	left to right
++ -- + - ! ~ (type) * & sizeof	Prefix increment/decrement Unary plus and minus not operator and bitwise complement type cast Indirection or dereference operator Address of operator Determine size in bytes	right to left
* / %	Multiplication, division and modulus	left to right
+ -	Addition and subtraction	left to right
<< >>	Bitwise left shift and right shift	left to right
< <= > >=	relational less than/less than equal to relational greater than/greater than or equal to	left to right
== !=	Relational equal to and not equal to	left to right
&	Bitwise AND	left to right
^	Bitwise exclusive OR	left to right
	Bitwise inclusive OR	left to right
&&	Logical AND	left to right
	Logical OR	left to right
? :	Ternary operator	right to left
= += -= *= /= %= &= ^= = <<= >>=	Assignment operator Addition/subtraction assignment Multiplication/division assignment Modulus and bitwise assignment Bitwise exclusive/inclusive OR assignment	right to left
,	Comma operator	left to right



Associativity in Operator Precedence

Associativity is a property of some operators that determines the order in which operations are performed when multiple operators of the same precedence appear in an expression.

Eg-Arithmetic operators are left associative

So the expression `int result=5-3+2;` is evaluated as $(5-3)+2$

I.e value of result is 4.

Exercise-Guess the output.



```
int x=15;  
int y=x=12;  
printf("%d,%d",x,y);
```




Answer

12, 12

```
int a=10,b=8,c=6;  
int res1=c+(a>b);  
int res2=c+a>b;  
printf("%d,%d",res1,res2);
```



Answer

7,1

```
int a=12,b=8,c=16;  
double res1=a+b/c;  
double res2=a+(double)b/c;  
double res3=a+(double)(b/c);  
printf("%lf,%lf,%lf",res1,res2,res3)
```



Answer

12.000000,12.500000,12.000000