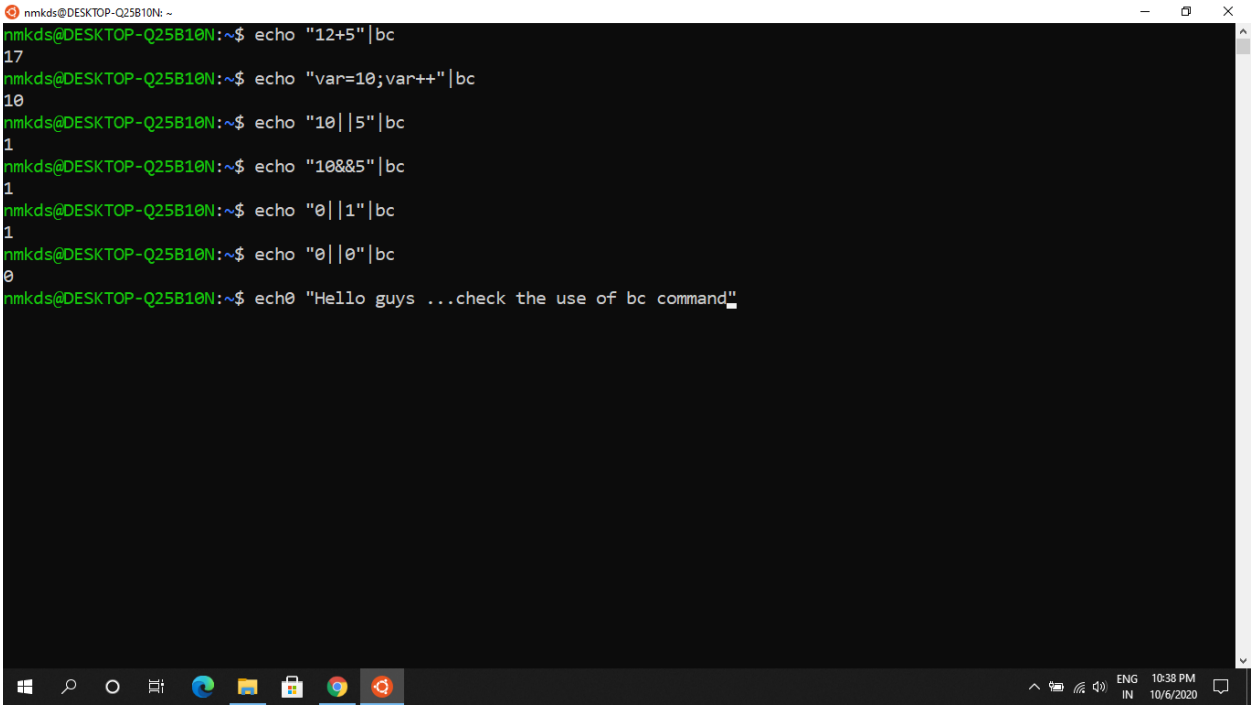
Bc command



bc command is used for command line calculator. It is similar to basic calculator by using which we can do basic mathematical calculations.
Arithmetic operations are the most basic in any kind of programming language. Linux or Unix operating system provides the **bc command** and **expr command** for doing arithmetic calculations. You can use these commands in bash or shell script also for evaluating arithmetic expressions.

Syntax:

bc [-hlwsqv] [long-options] [file ...]

Options:

-h, {- -help } : Print the usage and exit
-i, {- -interactive } : Force interactive mode
-l, {- -mathlib } : Define the standard math library
-w, {- -warn } : Give warnings for extensions to POSIX bc
-s, {- -standard } : Process exactly the POSIX bc language
-q, {- -quiet } : Do not print the normal GNU bc welcome
-v, {- -version } : Print the version number and copyright and quit

The bc command supports the following features:

- Arithmetic operators
- Increment or Decrement operators
- Assignment operators
- Comparison or Relational operators

- Logical or Boolean operators
- Math functions
- Conditional statements
- Iterative statements

1. Arithmetic Operators

```
Input : $ echo "12+5" | bc

Output : 17

Input : $ echo "10^2" | bc

Output : 100
```

How to store the result of complete operation in variable?

Example:

```
Input:

$ x=`echo "12+5" | bc`

$ echo $x

Output:17
```

Explanation: Stores the result of first line of input in variable x and then display variable x as **$x**.

2. Assignment Operators

The list of assignments operators supported are:

- **var = value** : Assign the vale to the variable
- **var += value** : similar to var = var + value
- **var -= value** : similar to var = var – value
- **var *= value** : similar to var = var * value
- **var /= value** : similar to var = var / value
- **var ^= value** : similar to var = var ^ value
- **var %= value** : similar to var = var % value

Examples:

```
Input: $ echo "var=10;var" | bc

Output: 10
```

Explanation: Assign 10 to the variable and print the value on the terminal.

```
Input: $ echo "var=10;var^=2;var" | bc

Output: 100
```

Explanation: Squares the value of the variable and print the value on the terminal.

How to store the result of complete operation in variable?

Example:

```
Input:

$ x=`echo "var=500;var%=7;var" | bc`
```

```
$ echo $x

Output:3
```

Explanation: Stores the result of 500 modulo 7 i.e. remainder of 500/7 in variable x and then display variable x as **$x**.

3. Increment Operators

There are 2 kinds of increment operators:

- **++var** : Pre increment operator, variable is increased first and then result of variable is stored.
- **var++** : Post increment operator, result of the variable is used first and then variable is incremented.

Examples:

```
Input: $ echo "var=10;++var" | bc

Output: 11
```

Explanation: Variable is increased first and then result of variable is stored.

```
Input: $ echo "var=10;var++" | bc

Output: 10
```

Explanation: Result of the variable is used first and then variable is incremented.

4. Decrement Operators

There are 2 kinds of decrement operators:

- **– – var** : Pre decrement operator, variable is decreased first and then result of variable is stored.
- **var – –** : Post decrement operator, result of the variable is used first and then variable is decremented.

Examples:

```
Input: $ echo "var=10;--var" | bc

Output: 9
```

Explanation: Variable is decreased first and then result of variable is stored.

```
Input: $ echo "var=10;var--" | bc

Output: 10
```

Explanation: Result of the variable is used first and then variable is decremented.

5. Comparison or Relational Operators

Relational operators are used to compare 2 numbers. If the comparison is true, then result is **1**. Otherwise(false), returns **0**. These operators are generally used in conditional statements like **if**. The list of relational operators supported in bc command are shown below:

- **expr1<expr2** : Result is 1 if expr1 is strictly less than expr2.
- **expr1<=expr2** : Result is 1 if expr1 is less than or equal to expr2.

- **expr1>expr2** : Result is 1 if expr1 is strictly greater than expr2.
- **expr1>=expr2** : Result is 1 if expr1 is greater than or equal to expr2.
- **expr1==expr2** : Result is 1 if expr1 is equal to expr2.
- **expr1!=expr2** : Result is 1 if expr1 is not equal to expr2.

Examples:

Input: $ echo "10>5" | bc

Output: 1

Input: $ echo "1==2" | bc

Output: 0

6. Logical or Boolean Operators

Logical operators are mostly used in conditional statements. The result of the logical operators is either **1**(TRUE) or **0**(FALSE).

- **expr1 && expr2** : Result is 1 if both expressions are non-zero.
- **expr1 || expr2** : Result is 1 if either expression is non-zero.
- **! expr** : Result is 1 if expr is 0.

Examples:

Input: $ echo "10 && 5" | bc

Output: 1

Input: $ echo "0 || 0" | bc

Output: 0

Input: $ echo "! 0" | bc

Output: 1

7. Mathematical Functions

The built-in math functions supported are :

- **s (x)**: The sine of x, x is in radians.
- **c (x)** : The cosine of x, x is in radians.
- **a (x)** : The arctangent of x, arctangent returns radians.
- **l (x)** : The natural logarithm of x.
- **e (x)** : The exponential function of raising e to the value x.
- **j (n,x)** : The bessel function of integer order n of x.
- **sqrt(x)** : Square root of the number x. If the expression is negative, a run time error is generated.

In addition to the math functions, the following functions are also supported :

- **length(x)** : returns the number of digits in x.

- **read()** : Reads the number from the standard input.
- **scale(expression)** : The value of the scale function is the number of digits after the decimal point in the expression.
- **ibase** and **obase** define the conversion base for input and output numbers. The default for both input and output is base 10.
- **last** (an extension) is a variable that has the value of the last printed number.

Examples:

Input:

$ pi=`echo "h=10;4*a(1)" | bc -l`

$ echo $pi

Output: 3.14159265358979323844

Explanation: Assign the value of "pi" to the shell variable pi. Here, a refers to the arctangent function, which is part of the math library loaded with the -l option.

Input: $ echo "scale($pi)" | bc -l

Output: 20

Explanation: Gives the number of digits after decimal point in value of "pi" calculated in previous example.

Input: $ echo "s($pi/3)" | bc -l

Output: .86602540378443864675

Explanation: Gives sine values at "pi/3" angle. Angle must be in radians. Here, s refers to the sine function

Input: $ echo "c($pi/3)" | bc -l

Output: .50000000000000000001

Explanation: Gives cosine values at "pi/3" angle. Angle must be in radians. Here, c refers to the cosine function.

Input: $ echo "e(3)" | bc -l

Output:20.08553692318766774092

Explanation: Gives exponential^value as output.

Input: $ echo "l(e(1))" | bc -l

Output: .99999999999999999999

Explanation: Gives natural logarithm of the value i.e. w.r.t. base 'e'.

Input: $ echo "obase=2;15" | bc -l

Output: 1111

Explanation: Convert Decimal to Binary.

Input: $ echo "obase=8;9" | bc -l

Output: 11

Explanation: Convert Decimal to Octal.

Input: $ echo "ibase=2;1111" | bc -l

Output: 15

Explanation: Convert Binary to Decimal.

Input: $ echo "ibase=2;obase=8;10" | bc -l

Output: 2

Explanation: Convert Binary to Octal.

8. Conditional Statements

Conditional Statements are used to take decisions and execute statements based on these decisions. bc command supports the if condition.

Syntax:

if(condition**)** {statements} **else** {statemnts}

Example:

Input: $ echo 'n=8;m=10;if(n>m) print "n is greater" else print "m is greater" ' | bc -l

Output: m is greater

9. Iterative statements

bc command supports the for loop and while loop for doing iterations.

Syntax:

for(assignment; condition; updation**)**
{
    statements.....
    .....
    .....
}

OR

while(condition**)**

```
{
    statements.....
    .......
    ........
}
```

Examples:

```
Input: $ echo "for(i=1; i<=10; i++) {i;}" | bc

Output:

1

2

3

4

5

6

7

8

9

10

Input: $ echo "i=1;while(i<=10) {i; i+=1}" | bc

Output:

1

2

3

4

5

6

7

8

9

10
```