## Operators & Expressions

An operator is a symbol or letter used to indicate a specific operation on variables in a program. Operators act upon the data items called as operands.

An expression is a combination of operands ie. constants, variables and numbers connected by operators and parenthesis.

A+B  unary/binary

(A+B)/T-(C/D)

A and B are operands and + is an operator.

## Types

- **Arithmetic operators** used to perform various arithmetic calculations like addition, subtraction, multiplication and division.

| Symbol | Operation | Example | Precedence |
|---|---|---|---|
| + | Addition | X+y | Lowest |
| - | Subtraction | x-y | Lowest |

| | | | |
|---|---|---|---|
| * | Multiplication | X*y | Highest |
| / | Division | x/y | Highest |
| % | Modulus/remainder | X%y | highest |

## Order of precedence

| Operators | Order | Associativity |
|---|---|---|
| */ % | Highest | Left to right |
| + - | Lowest | Left to right |

BODMAS

- **Relational Operator** used to compare two values of the operand and the result is always logical ie true[1] or false[0]. It is used for decision making statements. Expressions with relational operators are called relational expressions.

| Symbol | Operation | Example | Precedence |
|---|---|---|---|
| > | Greater than | X>y | Highest |
| >= | Greater than and equal to | x>=y | Highest |
| < | Less than | X<y | Highest |

| | | | |
|---|---|---|---|
| <= | less than and equal to | X<=y | Highest |
| == | Equal to | X==y | Lowest |
| != | Not equal to | X!=y | Lowest |

A=3, b=3  a>b  f  a<b  t  a==b  t  a!=b  t

## Order of precedence

| Operators | Order | Associativity |
|---|---|---|
| >. >=  < <= | Highest | Left to right |
| ==  != | lowest | Left to right |

Suppose I,j,k  are 1,2,3 respectively

| Expression | Interpretation | Value |
|---|---|---|
| i>j | False | 0 |
| I<=(j+k) | True | 1 |
| K==3 | | |
| J!=2 | | |
| k>=(i+j) | | |
| J<k | | |

- Logical operators used to connect relational expressions or logical expressions. The result is always logical true[ ] /false[0]

| Operator | Operation | Example | Precedence |
|---|---|---|---|
| ! | Logical NOT or Negation | !x | Highest |
| && | Logical AND or Conjunction | X&&y | intermediate |
| \|\| | Logical OR or disjunction | X\|\|y | Lowest |

X=4 y=5 a=4 b=7 (x+y=9) && (a+b=11)

F f f f

F t f t

T f f t

T t t t

## Rules

1. Logical NOT is an unary operator, it negate the value of the operand
2. The output of AND operation is true when both operands are true else false.
3. The output of OR operation is true if both or any one operand is true else false.

- **Assignment operator** The most common assignment operator is =. This operator assigns the value in

right side to the left side. For example:

```
var=5  //5 is assigned to var
a=c;   //value of c is assigned to
a
5=c;   // Error! 5 is a constant.
```

Syntax identifier=expression

| Operator | Example | Same as |
|---|---|---|
| = | a=b | a=b |
| += | a+=b | a=a+b |
| -= | a-=b | a=a-b |
| *= | a*=b | a=a*b |
| /= | a/=b | a=a/b |
| %= | a%=b | a=a%b |

**Multiple assignment:**

Identifier1=identifier2=…….

The assignments are carried out from right to left

Eg var1=var2=var3=0

A=0 b=0 c=0 a=b=c=0

- **Unary operator** are that act upon a single operand to produce a new value. It precedes the single operand by unary operator.

| Operator | Operation | Ex |
|---|---|---|
| ++ | Pre increment | ++i |
| ++ | Post increment | i++ |
| -- | Pre decrement | --i |
| -- | Post decrement | i-- |
| & | Address of | &a (returns the address of a in memory) |
| *- | Value at | *a (returns |

| | | |
|---|---|---|
| | address | the value stored at location a) |
| Sizeof() | Size of datatype | Int I ; sizeof(i); returns 2 |

int I, j  sizeof(i)  2   char a[]=”computer”
sizeof(a)  =8

**Unary minus(- operator):** minus sign preceding a numeric constant variable or expression
Eg. -756

**Increment operator(++):** causes its operand o be increased by 1.

**Pre-increment operator:** operator precedes the variable, ie ++I, the value is incremented and then the statement gets executed.

```
Main()
{
Int i=10
Printf(“\n i=%d”,i);
```

```
Printf("\n i=%d",++i);
Printf("\n i=%d",i);
}
Output: i=10
    I=11
    I=11
```

**Post increment :** operator follows the variable ie i++. The statement gets executed and then value is incremented.

```
Main()
{
Int i=10
Printf("\n i=%d",i);
Printf("\n i=%d",i++);
Printf("\n i=%d",i);
    }
Output: i=10
    I=10
    I=11
```

**Decrement operator(--):** causes its operand o be decreased by 1.

**Pre-decrement operator:** operator precedes the variable, ie --I, the value is

decremented and then the statement gets executed.

```
Main()
{
Int i=10
Printf("\n i=%d",i);
Printf("\n i=%d",--i);
Printf("\n i=%d",i);
    }
Output: i=10
    I=9
    I=9
```

**Post decrement :** operator follows the variable ie i--. The statement gets executed and then value is incremented.

```
Main()
{
Int i=10
Printf("\n i=%d",i);
Printf("\n i=%d",i--);
Printf("\n i=%d",i);
    }
Output: i=10
```

*I=10*

*I=9*

**Address of Operator:[&]** when prefixed with a variable return the address of the memory location where variable is stored. It is also used with scanf() function and to initialize the pointers.

The address of x variable can be determined by &x, where & is unary operator which returns the address the of the memory location.eg y=&x, the variable y is called pointer to x, because it points to the location where x is stored in memory, y represents x's address not the value.

| Address of x | value of x |
|---|---|
| Y | x |

**Value at address operator(*):** also called indirection operator. The data item represented by x can be also be accessed by *y where * is unary operator that operates only on a pointer variable, ie x and *y represents same

data item ie  contents of the same memory location

Eg:  int x=20, *y, z

Y=&x

Z=*y

x , *y and z represent the same value ie 20

int x=10, *y

y=&x

printf("%d %d%d%d", x, &x,y,*y, *(&x));

x=  10  &x=1000  y=  1000  *y=10  *(&x) 10

**Not or negation operator(!):** will return its value either true or false. It negates the operand, it is an unary operator and precedes its operands.  syntax. !(exp)

Eg. !a, !b

| A | !a |
|---|---|
| True | False |
| False | True |

X=2  y=3  !(x==y)  t

**Sizeof operator sixeof():** returns the size of its operand in bytes. It precedes its operator.
Syntax: sizeof(exp)
Ex: int a
Float b;
Sizeof(a) = 2
**Sizeof(b)=4**

**Conditional or ternary operator (?:):** it is used to check a condition. It is equal to if then and else statement. Syntax: exp1?exp2:exp3.
Since it operates on three values it is also called ternary operator.
Eg: int x=2, y=4,z
Z = (x>y) ? x : y ;

**7. Bitwise operator:** operators which at bit level and allows the user to manipulate individual bits, these operators are used in low level ie machine level programming.

| Operator | Description | Example |
|---|---|---|
| & | Binary AND Operator | (A & B) will |

| | | |
|---|---|---|
| | copies a bit to the result if it exists in both operands. | give 12 which is 0000 1100 |
| \| | Binary OR Operator copies a bit if it exists in either operand. | (A \| B) will give 61 which is 0011 1101 |
| ^ | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) will give 49 which is 0011 0001 |
| ~ | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number. |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right | A << 2 will give 240 which is 1111 0000 |

| | | |
|---|---|---|
| | operand. | |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 will give 15 which is 0000 1111 |

```
#include <stdio.h>

main()
{

   unsigned int a = 60;  /* 60 =
0011 1100 */
   unsigned int b = 13;  /* 13 =
0000 1101 */
   int c = 0;

   c = a & b;        /* 12 = 0000
1100 */
   printf("Line 1 - Value of c is
%d\n", c );
```

```
   c = a | b;        /* 61 = 0011
1101 */
   printf("Line 2 - Value of c is
%d\n", c );

   c = a ^ b;        /* 49 = 0011
0001 */
   printf("Line 3 - Value of c is
%d\n", c );

   c = ~a;           /*-61 = 1100
0011 */
   printf("Line 4 - Value of c is
%d\n", c );

   c = a << 2;     /* 240 = 1111
0000 */
   printf("Line 5 - Value of c is
%d\n", c );

   c = a >> 2;     /* 15 = 0000
1111 */
   printf("Line 6 - Value of c is
%d\n", c );
}
```

When you compile and execute the above program it produces the following result:

```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15
```