OPERATORS

Operator & Operand

- The type of operations that can be performed on the data objects are specified by operators.

- The data items on which an operator acts are called its operands.

Unary, Binary, Ternary Operands

- An operator can be unary, binary or ternary depending upon whether it operates on one, two or three operands.

Simple & Compound Expressions

- An operator along with its operands constitute a simple expression.

- A compound expression can be formed by using simpler expressions as operands of the different types of operators.
- The evaluation order of the operators in an expression will be determined by the operator precedence rules followed in the C language.

Arithmetic Operator

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

```c
// Working of arithmetic operators
#include <stdio.h>
int main()
{
    int a = 9,b = 4, c;

    c = a+b;
    printf("a+b = %d \n",c);
    c = a-b;
    printf("a-b = %d \n",c);
    c = a*b;
    printf("a*b = %d \n",c);
    c = a/b;
    printf("a/b = %d \n",c);
    c = a%b;
    printf("Remainder when a divided by b = %d \n",c);

    return 0;
}
```

Suppose `a = 5.0`, `b = 2.0`, `c = 5` and `d = 2`. Then in C programming,

```
// Either one of the operands is a floating-point number
a/b = 2.5
a/d = 2.5
c/b = 2.5

// Both operands are integers
c/d = 2
```

Exercise:

Suppose x, y and z are integer variables that have been assigned the values x = 18, y = 3 and z = -3. Determine the value of each of the following arithmetic expressions.

```
(a) x + y * z     (b) 2 * x + 4 - (y - z)   (c) x / y + z
(d) x % (y * x)   (e) x / z                 (f) x % z + 1
(g) x + y / z     (h) x * (y / z)           (i) (x * z) % y - 1
(j) x * (z % y)
```

Increment and Decrement Operators

- C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.
- These are unary operators

```
// Working of increment and
decrement operators
#include <stdio.h>
int main()
{
    int a = 10, b = 100;
    float c = 10.5, d = 100.5;

    printf("++a = %d \n", ++a);
    printf("--b = %d \n", --b);
    printf("++c = %f \n", ++c);
    printf("--d = %f \n", --d);

    return 0;
}
```

Increment/Decrement as Postfix/Prefix

- If you use the ++ operator as a prefix like: ++var, the value of var is incremented by 1; then it returns the value.
- If you use the ++ operator as a postfix like: var++, the original value of var is returned first; then var is incremented by 1.

```
#include <stdio.h>
int main() {
   int var1 = 5, var2 = 5;

   // 5 is displayed
   // Then, var1 is increased to 6.
   printf("%d\n", var1++);
   printf("%d\n", var1);

   // var2 is increased to 6
   // Then, it is displayed.
   printf("%d\n", ++var2);
   printf("%d\n", var2);

   return 0;
}
```

Assignment Operators

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

```c
// Working of assignment operators
#include <stdio.h>
int main()
{
    int a = 5, c;

    c = a;       // c is 5
    printf("c = %d\n", c);
    c += a;      // c is 10
    printf("c = %d\n", c);
    c -= a;      // c is 5
    printf("c = %d\n", c);
    c *= a;      // c is 25
    printf("c = %d\n", c);
    c /= a;      // c is 5
    printf("c = %d\n", c);
    c %= a;      // c = 0
    printf("c = %d\n", c);

    return 0;
}
```

Relational Operator

Operator	Meaning of Operator	Example
==	Equal to	`5 == 3` is evaluated to 0
>	Greater than	`5 > 3` is evaluated to 1
<	Less than	`5 < 3` is evaluated to 0
!=	Not equal to	`5 != 3` is evaluated to 1
>=	Greater than or equal to	`5 >= 3` is evaluated to 1
<=	Less than or equal to	`5 <= 3` is evaluated to 0

```c
// Working of relational operators
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10;

    printf("%d == %d is %d \n", a, b, a == b);
    printf("%d == %d is %d \n", a, c, a == c);
    printf("%d > %d is %d \n", a, b, a > b);
    printf("%d > %d is %d \n", a, c, a > c);
    printf("%d < %d is %d \n", a, b, a < b);
    printf("%d < %d is %d \n", a, c, a < c);
    printf("%d != %d is %d \n", a, b, a != b);
    printf("%d != %d is %d \n", a, c, a != c);
    printf("%d >= %d is %d \n", a, b, a >= b);
    printf("%d >= %d is %d \n", a, c, a >= c);
    printf("%d <= %d is %d \n", a, b, a <= b);
    printf("%d <= %d is %d \n", a, c, a <= c);

    return 0;
}
```

Logical Operators

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression `((c==5) && (d>5))` equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression `((c==5) || (d>5))` equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression `!(c==5)` equals to 0.

```c
// Working of logical operators

#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) is %d \n", result);

    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) is %d \n", result);

    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) is %d \n", result);

    result = (a != b) || (c < b);
    printf("(a != b) || (c < b) is %d \n", result);

    result = !(a != b);
    printf("!(a != b) is %d \n", result);

    result = !(a == b);
    printf("!(a == b) is %d \n", result);

    return 0;
}
```

Exercises

Suppose a C program contains the following declarations and initial assignments.

```
int i = 18, j = 3;
float x = 0.001, y = -0.05;
char c = 'c', d = 'd';
```

What will be the value of each of the following expressions? Use the values initially assigned to the variables for each expression.

```
(a) 2 * (( i / 5) + (6 * (j + 4)) % (i - j - 2))
(b) (i -3 * j) % (c -2 * d) / (x -y)
(c) -(i + j)
(d) ++i
(e) j++
(f) --j
(g) --x
(h) i <= j + 1
(i) c + 1>d
(j) x <= y
(k) j != 9
(l) c = = 79
(m) 4 * (i + j) > 'e'
(n) (2 / x + y) = = 0
(o) 100 * x + (y = = 0)
(p) 100 * x + y = = 0
(q) !(i <= j)
(r) !(c = = 99)
(s) !(x > 0)
(t) (i > 0) && (j <= 6)
(u) (i >0) | | (j < 6) * (y) * (x > y) && (i > 0) | | (j < 4)
(v) ((x >= y) | | (i < 0)) && (j < 4)
```

Bitwise Operator

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise complement
<<	Shift left
>>	Shift right

Example #1: Bitwise AND

```c
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a&b);
    return 0;
}
```

Example #2: Bitwise OR

```c
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a|b);
    return 0;
}
```

Example #3: Bitwise XOR

```c
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a^b);
    return 0;
}
```

Example #4: Bitwise complement

```c
#include <stdio.h>
int main()
{
    printf("Output = %d\n",~35);
    printf("Output = %d\n",~-12);
    return 0;
}
```

Example #5: Shift Operators

```c
#include <stdio.h>
int main()
{
    int num=212, i;
    for (i=0; i<=2; ++i)
        printf("Right shift by %d: %d\n", i, num>>i);

     printf("\n");

     for (i=0; i<=2; ++i)
        printf("Left shift by %d: %d\n", i, num<<i);

     return 0;
}
```

Comma Operator

Comma operators are used to link related expressions together. For example:

```
int a, c = 5, d;
```

The sizeof operator

The `sizeof` is a unary operator that returns the size of data (constants, variables, array, structure, etc).

Example 6: sizeof Operator

```
#include <stdio.h>
int main()
{
    int a;
    float b;
    double c;
    char d;
    printf("Size of int=%lu bytes\n",sizeof(a));
    printf("Size of float=%lu bytes\n",sizeof(b));
    printf("Size of double=%lu bytes\n",sizeof(c));
    printf("Size of char=%lu byte\n",sizeof(d));

    return 0;
}
```

Ternary or Conditional Operator

Simple conditional operations can be performed with a conditional operator `? :`. A conditional expression uses the conditional operator and is written in the following manner.

```
(expression 1) ? (expression 2) : (expression 3);
```

The evaluation of such an expression begins with the evaluation of expression 1. If the evaluation of expression 1 returns true (i.e. returns a non-zero value) then expression 2 is evaluated, otherwise expression 3 is evaluated.

Example:

```
max = (a > b) ? a : b; (compares a and b and returns the higher value
to max)

result = (a == 0) ? 0: b/a; (prevents division by zero)
```

Exercise:

For a C program having the following declarations and initial assignments:

```
int i = 9, j = 5, k;
float x = 0.05, y = -0.001, z;
char a, b = 'b', c = 'c', d = 'd';
```

```
(i)    k = (i + j) / y        (ii)   i += (j - 3)
(iii)  z = (x + y)/ y         (iv)   k = z = y
(v)    k = (j ==6) ? i : j    (vi)   i = -j
(vii)  k = (j>3) ? i : j      (viii) k *= (x * y)
(ix)   y -= -x                (x)    z = (x > 0.1) ? x : d
(xi)   x *= 2                 (xii)  z = k / (j +1)
(xiii) i /= j                 (xiv)  a = (c < d) ? c : d
(xv)   a = b = d              (xvi)  i -= (j >= 3) ? j : i
```

Precedence

Operator Category	Operators	Associativity
unary operator	- ++ -- ! sizeof (type)	R –> L
arithmetic operator multiply, divide and remainder	* / %	L –> R
arithmetic operator add and subtract	+ -	L –> R
relational operators	< <= > >=	L –> R
equality operators	== !=	L –> R
logical and	&&	L –> R
logical or	||	L –> R
conditional operator	? :	R –> L
assignment operator	= += -= *= /= %=	R –> L

End

Question??