Operator and Expression - 2

Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Relational operators are used in decision making and loops.

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

Example: Relational Operators

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

    printf("%d == %d = %d \n", a, b, a == b); // true
    printf("%d == %d = %d \n", a, c, a == c); // false

    printf("%d > %d = %d \n", a, b, a > b); //false
    printf("%d > %d = %d \n", a, c, a > c); //false


    printf("%d < %d = %d \n", a, b, a < b); //false
    printf("%d < %d = %d \n", a, c, a < c); //true


    printf("%d != %d = %d \n", a, b, a != b); //false
    printf("%d != %d = %d \n", a, c, a != c); //true


    printf("%d >= %d = %d \n", a, b, a >= b); //true
    printf("%d >= %d = %d \n", a, c, a >= c); //false


    printf("%d <= %d = %d \n", a, b, a <= b); //true
    printf("%d <= %d = %d \n", a, c, a <= c); //true

    return 0;

}
```

Output

```
5 == 5 = 1
5 == 10 = 0
5 > 5 = 0
5 > 10 = 0
5 < 5 = 0
5 < 10 = 1
5 != 5 = 0
5 != 10 = 1
5 >= 5 = 1
5 >= 10 = 0
5 <= 5 = 1
5 <= 10 = 1
```

Exercise 1:

```
int i,j,k;
i = 1;
j = 2;
k = 3;
```

What will be the value of following expression?

```
    i < j
    (i+j) >= k
    (j+k) > (i+5)
    k != 3
    j == 2
```

Exercise 2:

```
int i = 7;
float f = 5.5;
char c = 'w';
```

What will be the value of following expression?

```
    f > 5
    (i+f) <= 10
    c == 119
    c != 'p'
    c >= 10*(i+f)
```

Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning of Operator	Example
&&	Logial 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.

Example : 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) equals to %d \n", result);

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

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

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

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

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

    return 0;
}
```

Output

```
(a == b) && (c > b) equals to 1
(a == b) && (c < b) equals to 0
(a == b) || (c < b) equals to 1
(a != b) || (c < b) equals to 0
!(a != b) equals to 1
!(a == b) equals to 0
```

Explanation of logical operator program

- (a == b) && (c > 5) evaluates to 1 because both operands (a == b) and (c > b) is 1 (true).
- (a == b) && (c < b) evaluates to 0 because operand (c < b) is 0 (false).
- (a == b) || (c < b) evaluates to 1 because (a = b) is 1 (true).
- (a != b) || (c < b) evaluates to 0 because both operand (a != b) and (c < b) are 0 (false).
- !(a != b) evaluates to 1 because operand (a != b) is 0 (false). Hence, !(a != b) is 1 (true).
- !(a == b) evaluates to 0 because (a == b) is 1 (true). Hence, !(a == b) is 0 (false).

Exercise 1:

```
int i = 7;
float f = 5.5;
char c = 'w';

What will be the value of following expression?
    (i >= 6) && (c == 'w')
    (i >= 6) || (c == 119)
    (f < 11) && (i>100)
    (c != 'p') || ((i+f) <= 10)
```

Exercise 2:

```
int i = 7;
float f = 5.5;

What will be the value of following expression?
    f  >  5
    !(f > 5)
    i <= 3
    ! (i <= 3)
    i > (f+1)
    !(i > (f+1))
```

Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

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

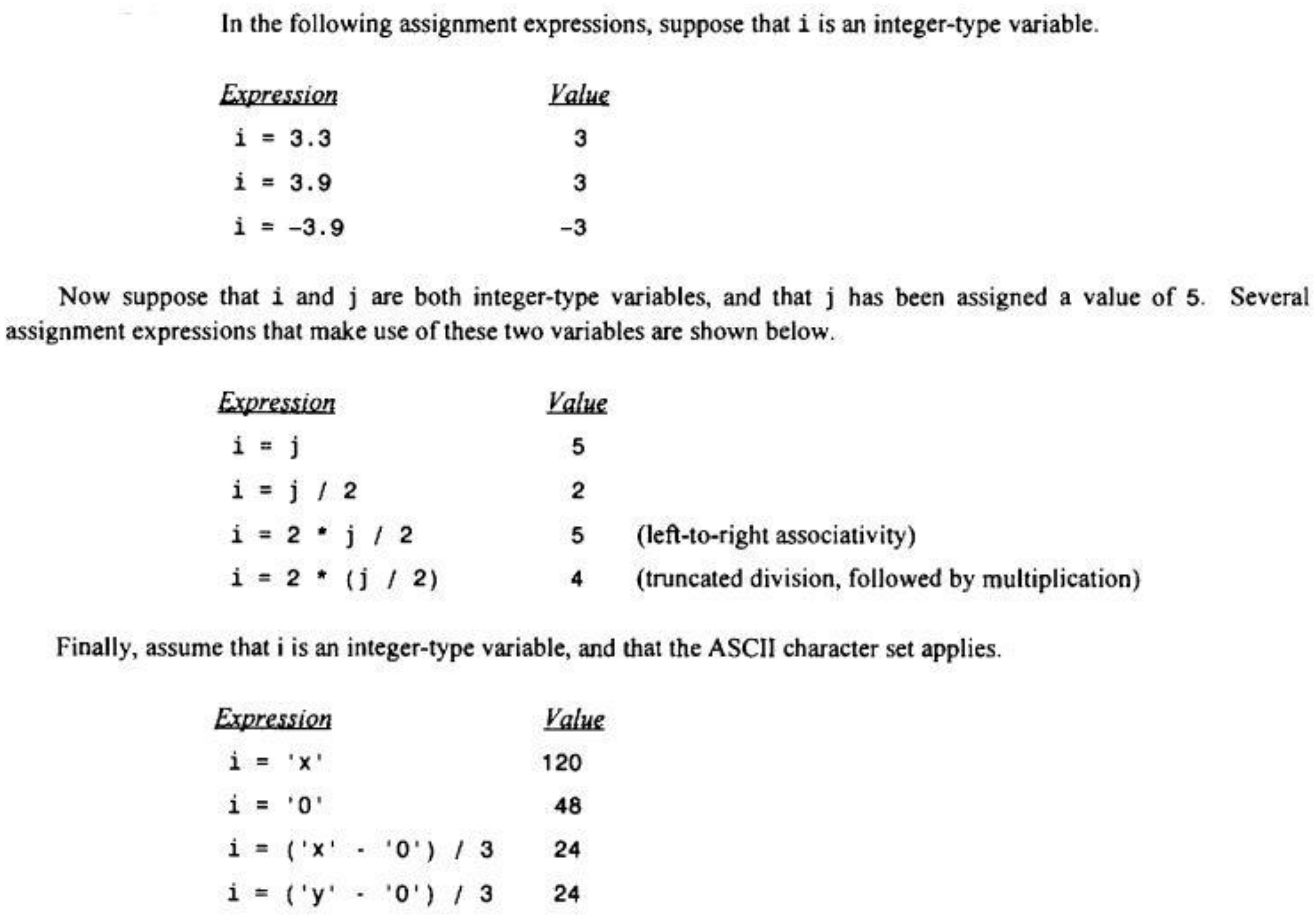
Example: Assignment Operators

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

    c = a;
    printf("c = %d \n", c);

    c += a; // c = c+a
    printf("c = %d \n", c);

    c -= a; // c = c-a
    printf("c = %d \n", c);

    c *= a; // c = c*a
    printf("c = %d \n", c);

    c /= a; // c = c/a
    printf("c = %d \n", c);

    c %= a; // c = c%a
    printf("c = %d \n", c);

    return 0;
}
```

Output

```
c = 5
c = 10
c = 5
c = 25
c = 5
c = 0
```

Some More Example:



In the following assignment expressions, suppose that i is an integer-type variable.

Expression	*Value*
`i = 3.3`	3
`i = 3.9`	3
`i = -3.9`	-3

Now suppose that i and j are both integer-type variables, and that j has been assigned a value of 5. Several assignment expressions that make use of these two variables are shown below.

Expression	*Value*	
`i = j`	5	
`i = j / 2`	2	
`i = 2 * j / 2`	5	(left-to-right associativity)
`i = 2 * (j / 2)`	4	(truncated division, followed by multiplication)

Finally, assume that i is an integer-type variable, and that the ASCII character set applies.

Expression	*Value*
`i = 'x'`	120
`i = '0'`	48
`i = ('x' - '0') / 3`	24
`i = ('y' - '0') / 3`	24

Suppose that i and j are integer variables whose values are 5 and 7, and f and g are floating-point variables whose values are 5.5 and –3.25. Several assignment expressions that make use of these variables are shown below. Each expression utilizes the *original* values of i, j, f and g.

Expression	*Equivalent Expression*	*Final value*
`i += 5`	`i = i + 5`	10
`f -= g`	`f = f - g`	8.75
`j *= (i - 3)`	`j = j * (i - 3)`	14
`f /= 3`	`f = f / 3`	1.833333
`i %= (j - 2)`	`i = i % (j - 2)`	0