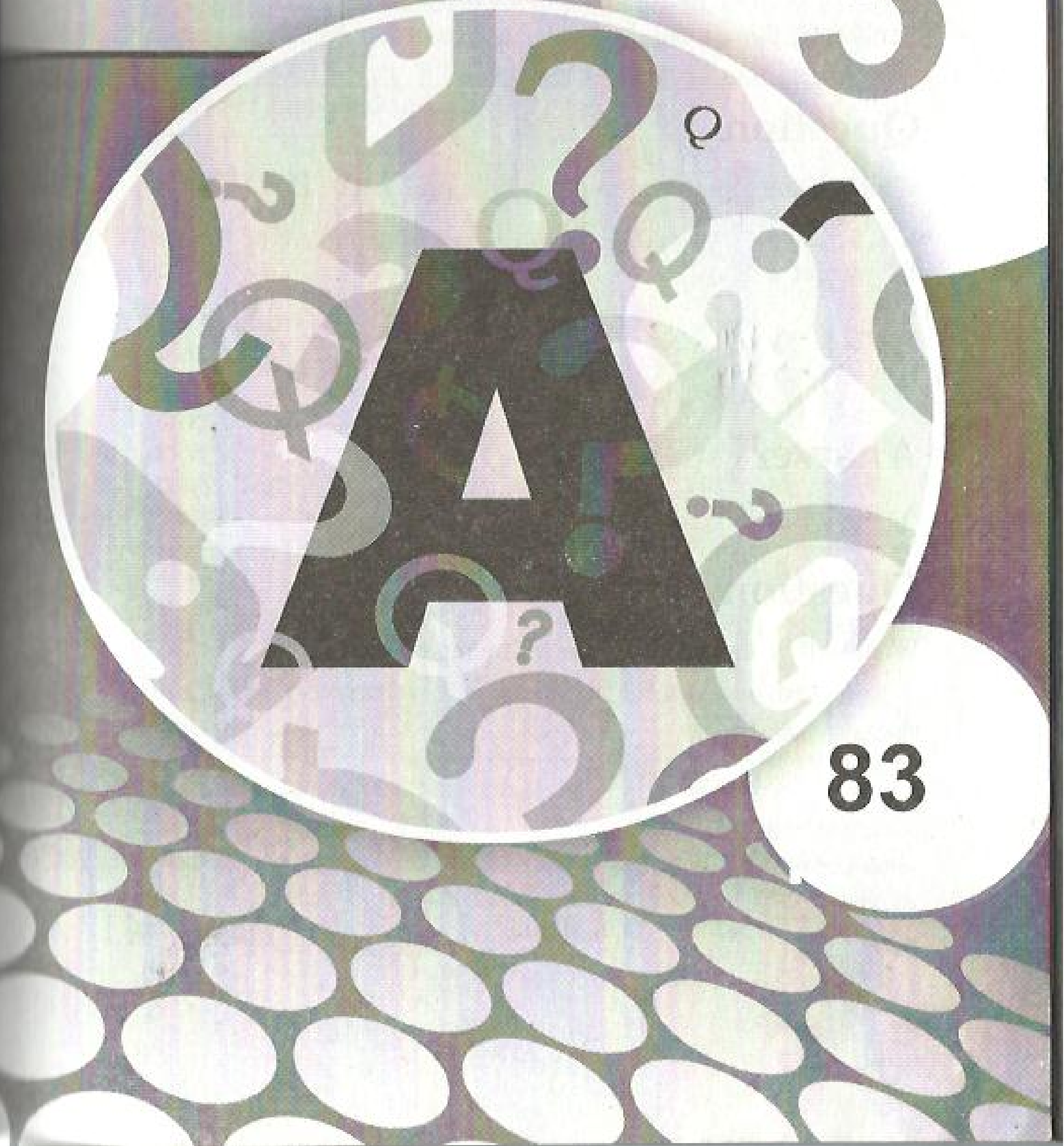
# Expressions

3



e= a+b>e && b+d<a;

As ↓ ↓ ↓
R L

## Question 3.1

In which order do the Relational, Arithmetic, Logical and Assignment operators get evaluated in C?

### Answer

Arithmetic, Relational, Logical, Assignment.

## Question 3.2

Which of the following are unary operators in C?

A. !
B. sizeof
C. ~
D. && → b/n
E. = → b/n

### Answer

A, B, C

## Question 3.3

What will be the output of the following program?

```
#include <stdio.h>
int main( )
{
    static int a[20] ;
    int i = 0 ;
    a[ i ] = i++ ;
    printf ( "%d %d %d\n", a[0], a[1], i ) ;
    return 0 ;
}
```

### Answer

0 0 1

That's what some of the compilers would give. Some other compiler may give a different answer. The reason is, when a single expression causes the same object to be modified, or to be modified and then inspected the behavior is undefined. In this case in the expression *a[ i ] = i++ ;* the variable *i* is being modified and then being used ( inspected ).

## Question 3.4

What will be the output of the following program?

```
#include <stdio.h>
int main( )
{
    int i = 3 ;
    i = i++ ;
    printf ( "%d\n", i ) ;
    return 0 ;
}
```

### Answer

4. But basically the behavior is undefined for the same reason as in 3.3 above.

## Question 3.5

The expression on the right hand side of && and || operators does not get evaluated if the left hand side determines the outcome. [True/False]

## Answer

True. For example if *a* is non-zero then *b* will not be evaluated in the expression *a* || *b*. Likewise, if *a* is 0 then *b* will not be evaluated in the expression *a* && *b*.

## Question 3.6

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    int i = 2 ;
    printf ( "%d %d\n", ++i, ++i ) ;
    return 0 ;              A   B
}
```

ptf(" %d %d", ++i, ++i)

Eva. right to left

display left to right

A. 3 4
B. 4 3
C. 4 4
D. Output may vary from compiler to compiler.

## Answer

D. The order of evaluation of the arguments passed to a function call is unspecified. Hence some compilers would report the output as 4 3 and some as 4 4.

## Question 3.7

Are the following two statements same? [Yes/No]

```
a <= 20 ? b = 30 : c = 30 ;
( a <= 20 ) ? b : c = 30 ;
```

## Answer

No

## Question 3.8

Are the following two statements same? [Yes/No]

```
a <= 20 ? ( b = 10 ) : ( b = 30 ) ;
b = a <= 20 ? 10 : 30 ;
```

## Answer

Yes

## Question 3.9

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    int x = 4, y, z ;
    y = --x ;
    z = x-- ;
    printf ( "%d %d %d\n", x, y, z ) ;
    return 0 ;
}
```

A. 4 3 3
B. 4 3 2
C. 3 3 2
[✓ D.] 2 3 3
E. 2 2 3

## Answer

D

## Question *3.10*

Can you suggest any other way of writing the following expression such that 30 is used only once?

```
a <= 20 ? b = 30 : c = 30 ;
```

*(&a) = a.

## Answer

```
*( ( a <= 20 ) ? &b : &c ) = 30 ;
```

Self

## Question *3.11*

How come that the C standard says that the expression

```
j = i++ * i++ ;
```

is undefined, whereas, the expression

```
j = i++ && i++ ;
```

is perfectly legal?

## Answer

According to the C standard an object's stored value can be modified only once (by evaluation of expression) between two sequence points. A sequence point occurs:

- at the end of full expression (expression which is not a sub-expression in a larger expression)
- at the &&, || and ?: operators
- at a function call (after the evaluation of all arguments, just before the actual call)

Since in the first expression *i* is getting modified twice between two sequence points the expression is undefined. Also, the second expression is legal because a sequence point is occurring at *&&* and *i* is getting modified once before and once after this sequence point.

## Question *3.12*

If *a[i] = i++* is undefined, then by the same reason *i = i + 1* should also be undefined. But it is not so. Why?

## Answer

The standard says that if an object is to get modified within an expression then all accesses to it within the same expression must be for computing the value to be stored in the object. The expression *a[i] = i++* is disallowed because one of the accesses of *i* (the one in *a[i]*) has nothing to do with the value that ends up being stored in *i*. In this case the compiler may not know whether the access should take place before or after the incremented value is stored. Since there's no good way to define it, the standard declares it as undefined. As against this, the expression *i = i + 1* is allowed because *i* is accessed to determine *i*'s final value.

## Question *3.13*

Will the expression **p++ = c* be disallowed by the compiler?

## Answer

No. Because here even though the value of *p* is accessed twice it is used to modify two different objects *p* and **p*.

## Question *3.14*

Which of the following is the correct order of calling functions in the code snippet given below?

```
a = f1 ( 23, 14 ) * f2 ( 12/4 ) + f3( ) ;
```

A. f1, f2, f3
B. f3, f2, f1
C. The order may vary from compiler to compiler
D. None of the above

## Answer

C. Here the multiplication will happen before the addition, but in which order the functions would be called is undefined. In an arithmetic expression the parentheses tell the compiler which operands go with which operators but do not force the compiler to evaluate everything within the parentheses first.

## Question 3.15

Which of the following is correct usage of conditional operators used in C?

A. a > b ? ( c = 30 ) : ( c = 40 ) ;
B. a > b ? c = 30 ; false (:) colon required
C. max = a > b ? a > c ? a : c : b > c ? b : c ;
D. return ( a > b ? a : b ) ;
E. a > b ? return ( a ) : return ( b ) ; two return not needed

## Answer

$max = a > b ? \underbrace{(a > c ? a : c)}_{a\ succeed} : \underbrace{(b > c ? b : c)}_{b\ succeed}$

A, C, D

## Question 3.16

What will be the output of the following program?

```
#include <stdio.h>
```

```
int main( )
{
    char ch ;
    ch = 'A' ;
    printf ( "The letter is " ) ;
    printf ( "%c", ch >= 'A' && ch <= 'Z' ? ch + 'a' - 'A' : ch ) ;
    printf ( "\nNow the letter is " ) ;
    printf ( "%c\n", ch >= 'A' && ch <= 'Z' ? ch : ch + 'a' - 'A' ) ;
    return 0 ;
}
```

## Answer

```
The letter is a
Now the letter is A
```

## Question 3.17

What will be the output of the following program?

```
#include <stdio.h>
int main( )
{
    int i = -3, j = 2, k = 0, m ;
    m = ++i && ++j || ++k ;
    printf ( "%d %d %d %d\n", i, j, k, m ) ;
    return 0 ;
}
```

## Answer

```
-2 3 0 1
```

x2 11.

## Question 3.18 .

What will be the output of the following program?

- 2

```
#include <stdio.h>
int main( )
{
    int i = -3, j = 2, k = 0, m ;
    m = ++i || ++j && ++k ;
    printf ( "%d %d %d %d\n", i, j, k, m ) ;
    return 0 ;
}
```

## Answer

```
-2 2 0 1
```

## Question 3.19

What will be the output of the following program?

```
#include <stdio.h>
int main( )
{
    int i = -3, j = 2, k = 0, m ;
    m = ++i && ++j && ++k ;
    printf ( "%d %d %d %d\n", i, j, k, m ) ;
    return 0 ;
}
```

## Answer

```
-2 3 1 1
```

## Question 3.20

What will be the output of the following program?

```
#include <stdio.h>
int main( )
{
    int i = 2 ;
    int j = i + ( 1, 2, 3, 4, 5 ) ;
    printf ( "%d\n", j ) ;
    return 0 ;
}
```

1, 2, 3, 4, 5

## Answer

7

This is because of the comma operator used in the expression *i + ( 1, 2, 3, 4, 5 )*. The comma operator has left-to-right associativity. The left operand is always evaluated first, and the result of evaluation is discarded before the right operand is evaluated. In this expression *5* is the right-most operand, hence after evaluating expression *( 1, 2, 3, 4, 5 )* the result is *5*, which on adding to *i* results into *7*.

## Question 3.21

State True or False:

A. Associativity of an operator is either Left to Right or Right to Left.

B. Every operator has an Associativity.

C. Associativity has no role to play unless the precedence of operators is same.

D. Two different operators would always have different Associativity.

E. In the expression *a = b = 5* the order of assignment is NOT decided by Associativity of operators. (right to left)

## Answer

A. True
B. True
C. True
D. False
E. False

## Question 3.22

Which of the following correctly shows the hierarchy of arithmetic operations in C?

A. / + * % -
B. * - % / +
C. + - / * %
D. * / % + -

## Answer

D

## Question 3.23

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    int x = 55 ;
    printf ( "%d %d %d\n", x <= 55, x = 40, x >= 10 ) ;
    return 0 ;
}
```

≠70

Right to left
(display)

A. 1 40 1
B. 1 55 1
C. 1 55 0
D. 1 1 1
E. 0 0 0

## Answer

A

## Question 3.24

Which of the following statements are correct about the code snippet given below?

```
int num = 10 ;
k = num > 5 ? k = 30 ; ;
```

colon is required

A. First ; is treated as a null statement
B. Second ; is treated as a statement terminator
C. 30 would be assigned to *k*
D. Compiler would report an error

## Answer

D

## Question 3.25

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    int a =100, b = 200, c ;
    c = ( a == 100 || b > 200 ) ;
    printf ( "c = %d\n", c ) ;
    return 0 ;
}
```

A. c = 100
B. c = 200
C. c = 1
D. c = 0
E. c = 300

## Answer

C

## Question 3.26

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    int x = 12, y = 7, z ;              12!=4
    z = x != 4 || y == 2 ;
    printf ( "z = %d\n", z ) ;
    return 0 ;
}
```

A. z = 0
B. z = 1
C. z = 4
D. z = 2

## Answer

B

## Question 3.27

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    int i = 4, j = -1, k = 0, w, x, y, z ;
(1) w = i || j || k ;
(0) x = i && j && k;          k is zero
(1) y = i || j && k ;
(1) z = i && j || k ;
    printf ( "%d %d %d %d\n", w, x, y, z ) ;
    return 0 ;
}
```

A. 1 1 1 1
B. 1 1 0 1
C. 1 0 0 1
D. 0 1 1 1
[✓ E. 1 0 1 1]

## Answer

E

## Question 3.28

Which of the following statements are correct about the program given below?

```
#include <stdio.h>
```

```
int main( )
{
    float a = 1.5, b = 1.55 ;
    if ( a = b )
        printf ( "a and b are equal\n" ) ;
    else
        printf ( "a and b are not equal\n" ) ;
    return 0 ;
}
```

[✓ A.] The output of the program would be "a and b are equal".
B. The statement *if ( a = b )* would report a compilation error.
C. Floats cannot be compared using *if*.
D. *switch* should be used to compare *floats*.
E. Conditional operators should be used to compare *floats*.

## Answer

A

## Question *3.29*

Which of the following is correct order of evaluation for the expression given below?

```
z = x + y * z / 4 % 2 - 1 ;
```

[✓ A.] * / % + - =
B. = * / % + -
C. / * % - + =
D. * / % - + =
E. - % / * + =

## Answer

A

## Question *3.30*

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    int x, y, z ;
    x = y = z = 1 ;
    z = ++x || ++y && ++z ;
    printf ( "x = %d y = %d z = %d \n", x , y , z ) ;
    return 0 ;
}
```

A. x = 2 y = 1 z = 1
B. x = 2 y = 2 z = 1
C. x = 2 y = 2 z = 2
D. x = 1 y = 2 z = 1

## Answer

A

## Question *3.31*

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    int x, y, z ;
    x = y = z = 1 ;
    printf ( "x = %d y = %d z = %d \n", ++x , y++ , ++z ) ;
    return 0 ;
}
```

A. x = 2 y = 1 z = 2
B. x = 2 y = 2 z = 2
C. x = 2 y = 2 z = 1
D. x = 1 y = 2 z = 1

## Answer

A