**OPERATORS ASSOCIATIVITY AND PRECEDENCE**

**1. Use operator associativity, evaluate the folowing expressions and predict the output**

**a. x = 34 + 12/4 – 56**

* According to operator precedence, division has higher precedence than addition and subtraction. So first, the division is evaluated.
  + 12 / 4 = 3
* The expression becomes:
  + x = 34 + 3 - 56
* Now, we evaluate addition and subtraction from left to right:
  + 34 + 3 = 37
  + 37 - 56 = -19

**b. 12 + 3 - 4 / 2 < 3 + 1**

* First, evaluate the division (4 / 2):
  + 4 / 2 = 2
* The expression becomes:
  + 12 + 3 - 2 < 3 + 1
* Now, evaluate addition and subtraction from left to right:
  + 12 + 3 = 15
  + 15 - 2 = 13
* The expression now becomes:
  + 13 < 3 + 1
* Evaluate the right-hand side:
  + 3 + 1 = 4
* Now, compare:
  + 13 < 4 is false.

**c. (2 + (3 + 2) ) \* 10**

* First, evaluate the expression inside the inner parentheses:
  + 3 + 2 = 5
* The expression becomes:
  + (2 + 5) \* 10
* Now, evaluate the addition inside the outer parentheses:
  + 2 + 5 = 7
* The expression becomes:
  + 7 \* 10 = 70

**d. 34 + 12/4 – 45**

* First, evaluate the division (12 / 4):
  + 12 / 4 = 3
* The expression becomes:
  + 34 + 3 - 45
* Now, evaluate addition and subtraction from left to right:
  + 34 + 3 = 37
  + 37 - 45 = -8

**2. Rewrite the following expressions with improved readability**

**a. age < 18 && height < 48 || age > 60 && height > 72**

**b. char name value**

**c. char $name**

**a. age < 18 && height < 48 || age > 60 && height > 72**

This expression checks two conditions:

* If the age is less than 18 and the height is less than 48, or
* If the age is greater than 60 and the height is greater than 72.

To improve readability, we can add parentheses to clarify the logical order of operations, since logical AND (&&) has higher precedence than logical OR (||).

**b. char name value**

**name\_value** is a more readable and valid identifier name than just name and value together, since we typically avoid spaces in variable names. Additionally, we use underscores for separation, making it easier to understand.

EX: char name\_value = 'A';

**c. char $name**

Avoid using the $ symbol in variable names and instead use a more conventional name like name$ if $ is acceptable for your environment, or name if it's not.

**3. Predict the value of a after each statement.**

**int main(void)**

**{**

**int i = 10;**

**char a = 'd';**

**a += 10;**

**a \*= 5;**

**a /= 4;**

**a %= 2;**

**a \*= a + i;**

**return 0;**

**}**

**A:**

a += 10;

* The += operator adds 10 to the value of a.
* a = 100 (ASCII value of 'd') + 10 = 110
* Now, a = 110.

a \*= 5;

* The \*= operator multiplies the value of a by 5.
* a = 110 \* 5 = 550
* Now, a = 550. However, a is a char, and char variables in C are typically 1 byte, which can hold values from -128 to 127 (signed char) or 0 to 255 (unsigned char). Since 550 exceeds the maximum value of 127 for a signed char, it will overflow.
  + In signed char, 550 % 256 = 550 - 256 \* 2 = 550 - 512 = 38. So, a = 38.
  + The corresponding character for 38 in ASCII is '&'.

a /= 4;

* The /= operator divides a by 4.
* a = 38 / 4 = 9 (integer division).
* Now, a = 9. The ASCII character for 9 is \t (tab), but since a is now a small integer, we focus on the value itself.

a %= 2;

* The %= operator calculates the remainder when a is divided by 2.
* a = 9 % 2 = 1 (remainder when 9 is divided by 2).
* Now, a = 1.

a \*= a + i;

* The \*= operator multiplies a by the value of a + i.
* a + i = 1 + 10 = 11
* a = 1 \* 11 = 11
* Now, a = 11.

**4. Consider a = 12, b = 3, predict the output of the following .**

**a. (a>100) && (b<10)**

**b. (a==4) && (b==2)**

**c. (a==11) && (a++)**

**A:**

**a. (a>100) && (b<10)**

First condition: a > 100

* a = 12, so 12 > 100 is false.

Second condition: b < 10

* b = 3, so 3 < 10 is true.

**Output for (a):**

* The result of the expression (a > 100) && (b < 10) is **false**.

**b. (a==4) && (b==2)**

First condition: a == 4

* a = 12, so 12 == 4 is false.

Second condition: b == 2

* b = 3, so 3 == 2 is false.

**Output for (b):**

* The result of the expression (a == 4) && (b == 2) is **false**.

**c. (a==11) && (a++)**

First condition: a == 11

* a = 12, so 12 == 11 is false.

Second condition: a++

* a++ is a postfix increment, which means a will be incremented after the condition is evaluated.

**Output for (c):**

* The result of the expression (a == 11) && (a++) is false, and a becomes 13 after the expression is evaluated.

**5. Consider a = 10, b = 11, predict the output of the following .**

**a. (a>10) || (b<10)**

**b. a || 12.12**

**c. a || b**

**d. !(a > 5)**

**A:**

**a. (a>10) || (b<10)**

First condition: a > 10

* a = 10, so 10 > 10 is false.

Second condition: b < 10

* b = 11, so 11 < 10 is false.

**Output for (a):**

* The result of the expression (a > 10) || (b < 10) is false.

**b. a || 12.12**

The logical OR (||) operator checks if either operand is true. In C, any non-zero value is considered true, and 0 is considered false.

a = 10, which is non-zero (so it is treated as true).

12.12 is a non-zero floating-point number, so it is also treated as true.

**Output for (b):**

* The result of the expression a || 12.12 is **true**.

**c. a || b**

a = 10, which is non-zero (so it is treated as true).

b = 11, which is also non-zero (so it is treated as true).

**Output for (c):**

* The result of the expression a || b is **true**.

**d. !(a > 5)**

First condition: a > 5

* + a = 10, so 10 > 5 is true.

The ! operator negates the result of the condition.

* + !(true) is false.

**Output for (d):**

* The result of the expression !(a > 5) is false.

**6. Consider int age = 10, height = 45, year = 2000; Predict the output of the following.**

**a. (age < 12 && height < 48) || (age > 65 && height > 72)**

**b. (year % 4 == 0 && year % 100 != 0 ) || (year % 400 == 0);**

**A:**

**a. (age < 12 && height < 48) || (age > 65 && height > 72)**

* First condition: age < 12 && height < 48
  + age = 10, so 10 < 12 is true.
  + height = 45, so 45 < 48 is true.
  + Since both conditions are true, the whole expression age < 12 && height < 48 is true.
* Second condition: age > 65 && height > 72
  + age = 10, so 10 > 65 is false.
  + height = 45, so 45 > 72 is false.
  + Since both conditions are false, the whole expression age > 65 && height > 72 is false.

**b. (year % 4 == 0 && year % 100 != 0 ) || (year % 400 == 0);**

First condition: year % 4 == 0 && year % 100 != 0

* year = 2000
* 2000 % 4 == 0: This is true (2000 is divisible by 4).
* 2000 % 100 != 0: This is false (2000 is divisible by 100).

Second condition: year % 400 == 0

* 2000 % 400 == 0: This is false (2000 is not divisible by 400).

**Output for (b):**

* The result of the expression (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0) is false.