1. What are the rules of expression operation?
   soln:

   The rules of expression evaluation in C are governed by the operator precedence and associativity, as well as some other key principles.

   Here are the main rules of expression evaluation:

   1. Operator Precedence: defines the order of evaluation based on operator priority.

   2. Operator Associativity: dictates the order when operators have the same precedence.

   3. Parentheses: override precedence and force explicit evaluation order.

   4. Short-circuit evaluation: applies to `&&` and `||` and can prevent the evaluation of subsequent operands.

   5. Side effects: must be carefully tracked to avoid unexpected behavior.

   6. Comma Operator: The comma operator `,` allows multiple expressions to be evaluated in sequence, and the value of the last expression is used.

   7. Sequence points: are essential to ensure the correct completion of operations (e.g., control flow statement, function calls etc).

2. Pointers-On-C.pdf 5.8 question 5
   soln:

```
int n;
scanf("%d", &n);
bool leap_year = ((n % 400 == 0) ||((n % 4 == 0) && (n % 100 != 0)) );
```

3. Pointers-On-C.pdf 5.8 question 6
   slon:

Operator	Side effect
++, - -	In both prefix and postfix forms, these operators modify the L value on which they operate
=	And all of the other assignment operators: they all modify the L value given as the left operand

4. What are the outputs of the following program? Try in different compilers

slon:

```c
#include<stdio.h>

int main(void){
    int a, b;
    a = 3;
    b = (a++) + (a++) + (a++);
    printf("%d, %d\n", a, b);

    a = 3;
    b = (++a) + (++a) + (++a);
    printf("%d, %d\n", a, b);

    return 0;
}
```

output of the above code for different compiler:

Compiler name	output
x86-64 gcc 14.2	6, 12 6, 16
x86-64 clang 19.1.0	6, 12 6, 15
x86 msvc v19.40 VS17.10	6, 9 6, 18

5. Pointers-On-C.pdf 5.8 question 7

soln:

```c
int a = 20;

    if( 1 <= a <= 10 )
            printf( "In range\n" );
    else
            printf( "Out of range\n" );
```

output of above code is “In range”.

6. Please explain the output of the following program
slon:

a.

```
int a[3][2] = {(1, 2), (3, 4), (5, 6)};

    printf("%d\n", a[0][1]);
```

The initialization of the array int a[3][2] = {(1, 2), (3, 4), (5, 6)}; is not correct in C.

In C, parentheses ( ) are used for expressions, and in this case, the commas inside the parentheses represent a comma operator. The comma operator evaluates all its operands from left to right and returns the value of the rightmost operand.

So, the expression (1, 2) will evaluate to 2, (3, 4) will evaluate to 4, and (5, 6) will evaluate to 6.

So, a[0][1] will be 4. The output will be 4.

b.

```
    int x = 3, y = 0, z = 0;

    if(x = y + z)

        printf("111\n");

    else
```

```
        printf("222\n");
```

output of the above code will be 222. y + z = 0 + 0 = 0, that means x = 0, so this statement evaluate as 0 or false. So else is executed.

c.

```
    int a = 0, b = 0, c = 0, d = 0;
    d = (c = ( a = 11, b = 22)) == 11;

    printf("%d %d", c, d);
```

output : 22 0

The comma operator evaluates all its operands from left to right and returns the value of the rightmost operand. so (a = 11, b = 22) return 22, then assign 22 in c and (c = 22) return 22. After that checking 22 == 11 that is false so d will 0 and c will be 22.

d.

```
    int x = 3, y = 2, z = 3;

    z = x < y ? !x : !((x = 2) || (x = 3)) ? (x = 1) :
(y = 2);

    printf("%d %d %d\n", x, y, z);
```

output: 2 2 2

first check “x < y”, for above code x < y is false so 2nd portion will be executed.In second another ternary operator so first condition “!((x = 2) || (x = 3))” will be checked. “(x = 2)” execute first and it returns 2 so || operator short circuited, (x = 3) will not be checked. Since condition will be false(!true) so 2nd portion of ternary operator will be executed, that means “y = 2” execute.

finally 2 assign in z.