```python
#Problem1
import os

def is_palindromic(n):
  s = str(n)
  return s == s[::-1]
def process_palindromes():
    if not os.path.exists('input.txt'):
        print("Error: can't find 'input.txt'")
        return
    with open('input.txt', 'r') as file:
      lines = file.readlines()

    strings = [line.strip() for line in lines if line.strip()]
    total_palindrome = 0
    results = []
    for s in strings:
      if is_palindromic(s):
          results.append("true")
          total_palindrome += 1
      else:
          results.append("false")
    for result in results:
        print(result)
    print(total_palindrome)

if __name__ == "__main__":
    process_palindromes()
```

```
D:\Boston University\1st\CS526 Data Structures&Algorithm>python Hw3_problem1.py < "input.txt"
false
true
true
2
```

```python
#Problem2
def sub_strings(s):
    unique_subs = set()
    def generate_substrings(start, end):
        if start >= len(s):
            return
        if end > len(s):
            generate_substrings(start+1, start+2)
            return
        substring = s[start:end]
        if substring:
            unique_subs.add(substring)
        generate_substrings(start, end + 1)
    generate_substrings(0, 1)
    return unique_subs
input_str = "abcab"
unique_substrings = sub_strings(input_str)
print(", ".join(sorted(unique_substrings)),"->",len(unique_substrings))
```

```
a, ab, abc, abca, abcab, b, bc, bca, bcab, c, ca, cab -> 12
```

```python
#Problem3
#(1)as an array
def insert_value(item, value):
    if not item:
        item.append(value)
    else:
        top = item.pop()
        insert_value(item, value)
        item.append(top)
def reverse_item(item):
    if item:
        value = item.pop()
        reverse_item(item)
        insert_value(item, value)

arb_items = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
reverse_item(arb_items)
print("Reverse by array:",arb_items)
```

```
Reverse by array: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```python
#(2)as a Linked-list
class ListNode:
    def __init__(self, val=0):
        self.val = val
        self.next = None
class StackLinkedList:
    def __init__(self):
        self.top = None
    def push(self, val):
        new_node = ListNode(val)
        new_node.next = self.top
        self.top = new_node
    def display(self):
        result = []
        current = self.top
        while current:
            result.append(current.val)
            current = current.next
        return result
stack = StackLinkedList()
for val in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
        stack.push(val)
print("Reverse by Linked-list:",stack.display())
```

```
Reverse by Linked-list: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```python
#(3)as a doubly linked-list
class DoublyListNode:
    def __init__(self, val=0):
        self.val = val
        self.next = None
        self.prev = None
class StackDoublyLinkedList:
    def __init__(self):
        self.top = None
    def push(self, val):
        new_node = DoublyListNode(val)
        if not self.top:
            self.top = new_node
        else:
            new_node.next = self.top
            self.top.prev = new_node
            self.top = new_node
    def display(self):
        result = []
        current = self.top
        while current:
            result.append(current.val)
            current = current.next

        return result

stack1 = StackDoublyLinkedList()
for val in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
        stack1.push(val)
print("Reverse by doubly linked-list:", stack1.display())
```

```
Reverse by doubly linked-list: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```python
#Problem4
from google.colab import files

# ==== Upload the file ====
print("Upload your file: ")
uploaded = files.upload()
filename = list(uploaded.keys())[0]
print(f"File name: {filename}")

# ==== Read the file ====
def read_input(filename):
        pairs = []
        with open(filename, 'r') as f:
                lines = f.readlines()
                for line in lines:
                        line = line.strip()
                        if not line or line[0].isdigit():
                                continue
                        data = line.split()
                        bx, by = float(data[1]), float(data[2])
                        gx, gy = float(data[4]), float(data[5])
                        pairs.append(((bx, by), (gx, gy)))
        return pairs

# ==== Determine whether they are across ====
def lines_intersect_or_coincident(A, B, C, D, eps=1e-9):
        d1x = B[0] - A[0]
        d1y = B[1] - A[1]
        d2x = D[0] - C[0]
        d2y = D[1] - C[1]
        # Cross product
        cross = d1x*d2y - d1y*d2x
        if abs(cross) > eps:
                # Not Parallel,intersect
                return True
        else:
                # Parallel, check collinear
                cross2 = (C[0]-A[0]) * d1y - (C[1]-A[1]) * d1x
                if abs(cross2) < eps:
                        # collinear,intersect
                        return True
                else:
                        # Parallel&not collinear, not intersect
                        return False

def check_any_line_intersection(pairs):
    n = len(pairs)
    for i in range(n):
        for j in range(i+1, n):
            A, B = pairs[i]
            C, D = pairs[j]
            if lines_intersect_or_coincident(A, B, C, D):
        # If any two lines intersect or overlap,fails
                return True
    return False

pairs = read_input(filename)
has_intersection = check_any_line_intersection(pairs)

if has_intersection:
        print("All Ghosts: were not eliminated")
else:
        print("All Ghosts: were eliminated")
```