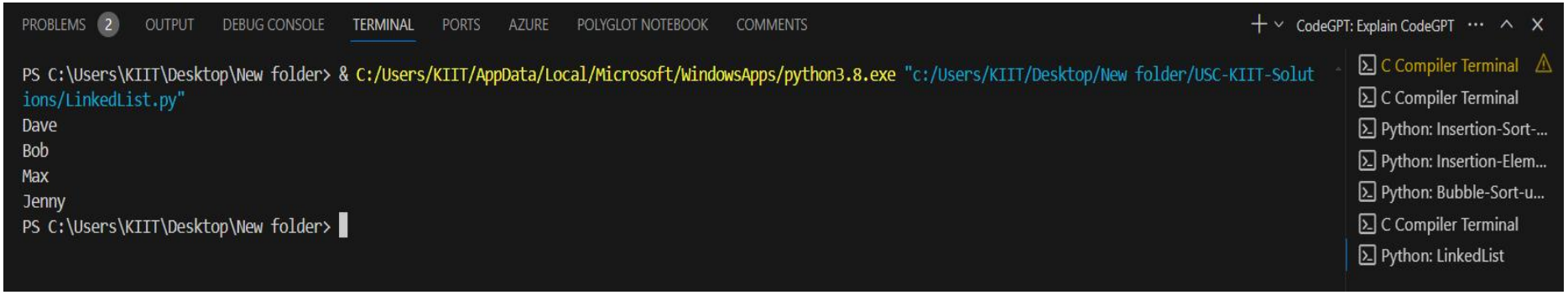
```python
# Q.1 ADVANCED- LINKED LISTS:


class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class LinkedList:
    def traversal(self):
        first = self.head
        while first:
            print(first.data)
            first = first.next
    def insert_new_header(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        self.head = new_node
    def search(self, x):
        temp = self.head
        while temp is not None:
            if temp.data == x:
                return True
            temp = temp.next
        else:
            return False
    def delete_node(self, data):
        temp = self.head
        while temp is not None:
            if temp.data == data:
                break
            prev = temp
            temp = temp.next
        prev.next = temp.next
    def delete_tail(self):
        temp = self.head
        while temp.next.next is not None:
```

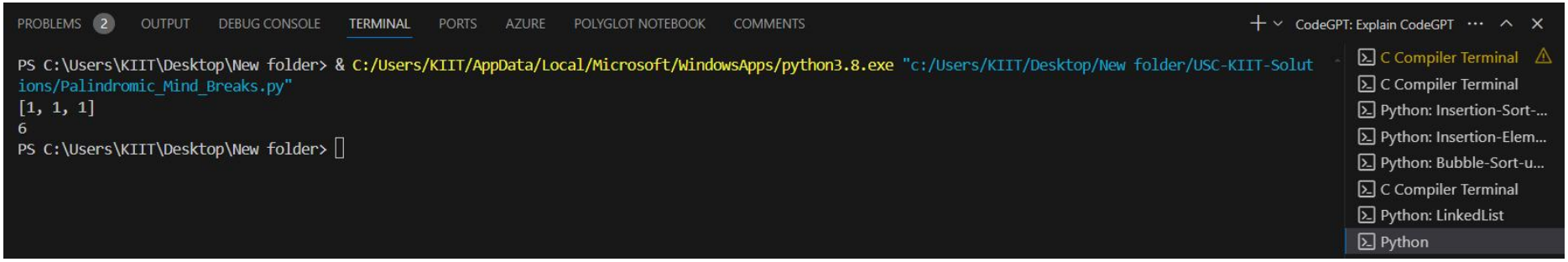
```
            temp = temp.next
        temp.next = None
family = LinkedList()
family.head = Node("Bob")
wife = Node("Amy")
first_kid = Node("Max")
second_kid = Node("Jenny")
family.head.next = wife
wife.next = first_kid
first_kid.next = second_kid
family.insert_new_header("Dave")
#family.delete_tail()
#print(family.search("Bob"))
family.delete_node("Amy")
family.traversal()
```



```python
# MINDBREAKER- PALINDROMIC MATRIX PATHS:

# --------------UNIQUE PATHS
def paths(m, n):
    row = [1] * n
    print(row)
    for i in range(m-1):
        newRow = [1] * n
        for j in range(n-2, -1, -1):
            newRow[j] = newRow[j+1] + row[j]
        row = newRow
    return row[0]

print(paths(3, 3))
# THE PATHS
# aaaa (0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2)
# aaba (0, 0) -> (0, 1) -> (1, 1) -> (2, 1) -> (2, 2)
# aabb (0, 0) -> (0, 1) -> (1, 1) -> (2, 1) -> (2, 2)
# abba (0, 0) -> (1, 0) -> (2, 0) -> (2, 1) -> (2, 2)
# [(0, 1), (0, 2), (1, 1), (1, 2)]
# 0, 0, 3, 2
```



```python
import numpy as np

def split(matrix):
    row, col = matrix.shape
    row2, col2 = row // 2, col // 2
    return matrix[:row2, :col2], matrix[:row2, col2:], matrix[row2:, :col2], matrix[row2:, col2:]
def strassen_recur(x, y):
    # Base case: 1x1 matrix
    if len(x) == 1:
        return x * y
    # Splitting the matrices into quadrants
    a, b, c, d = split(x)
    e, f, g, h = split(y)
    # Computing the 7 products using the Strassen algorithm
    p1 = strassen_recur(a, f - h)
    p2 = strassen_recur(a + b, h)
    p3 = strassen_recur(c + d, e)
    p4 = strassen_recur(d, g - e)
    p5 = strassen_recur(a + d, e + h)
    p6 = strassen_recur(b - d, g + h)
    p7 = strassen_recur(a - c, e + f)
    # Computing the values of the 4 quadrants of the final matrix
    c1 = p5 + p4 - p2 + p6
    c2 = p1 + p2
    c3 = p3 + p4
    c4 = p1 + p5 - p3 - p7
    # Combining the 4 quadrants into a single matrix
    c = np.vstack((np.hstack((c1, c2)), np.hstack((c3, c4))))
    return c
# Example usage:
x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6], [7, 8]])
result = strassen_recur(x, y)
```

```
print(result)
```