```
def knapsack_dynamic_programming(values, weights, capacity):
    n = len(values)
    # Create a table to store intermediate results, initialize with zeros
    dp = [[0 for _ in range(capacity + 1)] for _ in range(n + 1)]

    # Filling in the dynamic programming table
    for i in range(n + 1):
        for w in range(capacity + 1):
            if i == 0 or w == 0:
                # If there are no items or no capacity, the value is 0
                dp[i][w] = 0
            elif weights[i - 1] <= w:
                # If the item can fit in the knapsack, make a choice to maximize value
                dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w])
            else:
                # If the item doesn't fit, take the value from the row above
                dp[i][w] = dp[i - 1][w]

    # Backtracking to find the selected items
    selected_items = []
    i, w = n, capacity
    while i > 0 and w > 0:
        if dp[i][w] != dp[i - 1][w]:
            # If the value changed, the item was selected
            selected_items.append(i - 1)
            w -= weights[i - 1]
        i -= 1

    selected_items.reverse()  # Reverse the list to maintain the correct order

    # Return the maximum value and the list of selected items
    return dp[n][capacity], selected_items

# Get user input for values, weights, and capacity
n = int(input("Enter the number of items: "))
values = []
weights = []
for i in range(n):
    value = int(input(f"Enter value for item {i+1}: "))
    weight = int(input(f"Enter weight for item {i+1}: "))
    values.append(value)
    weights.append(weight)

capacity = int(input("Enter the knapsack capacity: "))

# Calculate the maximum value and selected items using the function
max_value, selected_items = knapsack_dynamic_programming(values, weights, capacity)

# Print the results
print("Maximum value:", max_value)
print("Selected items:", selected_items)
```