LAB TEST-02

NAME:-ANUSHA PEDDAPELLI

BATCH:-06(M)

COURSE:- AI Assisted Coding **DATE:-17-09-2025**

Subgroup M
M.1 — [S09M1] Stable sort employees by dept asc, salary desc

Scenario (sports analytics):
Context:
HR exports in sports analytics require deterministic sorting for payroll audits.
Your Task:
Sort employees by dept ascending and salary descending (stable), and re-emit CSV.
Data & Edge Cases:
name,dept,salary rows provided.
AI Assistance Expectation:
Use AI to outline csv.DictReader/Writer usage and key composition.
Constraints & Notes:
Stable sort within department by salary desc.
Sample Input
name,dept,salary
Raj,Eng,120
Maya,HR,90
Abi,Eng,110
Sample Output
Raj,Eng,120
Abi,Eng,110
Maya,HR,90

Acceptance Criteria: Stable and correct ordering

PROMPT:-

Write a program that reads employee details from a CSV file and sorts the employees by department (ascending) and then by salary (descending) while ensuring stable sorting. Finally, print or save the sorted data back into a CSV file.

CODE:-

```python
LAB.M.1.py > ...

# Example usage and output
if __name__ == "__main__":
    # Sample data for demonstration
    employees = [
        {'name': 'Alice', 'dept': 'HR', 'salary': '5000'},
        {'name': 'Bob', 'dept': 'IT', 'salary': '7000'},
        {'name': 'Charlie', 'dept': 'HR', 'salary': '6000'},
        {'name': 'David', 'dept': 'IT', 'salary': '6500'},
        {'name': 'Eve', 'dept': 'Finance', 'salary': '5500'}
    ]
    # Write sample data to input.csv
    with open('input.csv', 'w', newline='') as f:
        writer = csv.DictWriter(f, fieldnames=['name', 'dept', 'salary'])
        writer.writeheader()
        writer.writerows(employees)
    # Sort and output to output.csv
    sort_employees('input.csv', 'output.csv')
    # Print output.csv contents
    with open('output.csv', newline='') as f:
        print(f.read())
```

```python
LAB.M.1.py > ...
import csv
from typing import List

Zencoder
def sort_employees(input_csv: str, output_csv: str) -> None:
    with open(input_csv, newline='') as infile:
        reader = list(csv.DictReader(infile))
        # Stable sort: dept ascending, salary descending
        sorted_rows = sorted(reader, key=lambda x: (x['dept'], -int(x['salary'])))
    with open(output_csv, 'w', newline='') as outfile:
        writer = csv.DictWriter(outfile, fieldnames=reader[0].keys())
        writer.writeheader()
        writer.writerows(sorted_rows)

# Example usage and output
if __name__ == "__main__":
    # Sample data for demonstration
    employees = [
        {'name': 'Alice', 'dept': 'HR', 'salary': '5000'},
        {'name': 'Bob', 'dept': 'IT', 'salary': '7000'},
        {'name': 'Charlie', 'dept': 'HR', 'salary': '6000'},
        {'name': 'David', 'dept': 'IT', 'salary': '6500'},
        {'name': 'Eve', 'dept': 'Finance', 'salary': '5500'}
    ]
    # Write sample data to input.csv
    with open('input.csv', 'w', newline='') as f:
        writer = csv.DictWriter(f, fieldnames=['name', 'dept', 'salary'])
```

OUTPUT:-

```
PS C:\Users\Administrator\OneDrive\ai> & C:/Python313/python.exe c:/Users/Administrator/OneDrive/ai/LAB.M.1.py
name,dept,salary
Eve,Finance,5500
Charlie,HR,6000
Alice,HR,5000
Bob,IT,7000
David,IT,6500

PS C:\Users\Administrator\OneDrive\ai>
```

OBSERVATION:-

1. The sorting requires two keys:
    - dept → ascending

- salary → descending

2. Since salary must be descending, we can negate the salary (or use reverse=True in sorting).
3. The sort must be stable → Python's built-in sorted() is stable.
4. After sorting, the order of employees in the same department with equal salaries must remain unchanged.

M.2 — [S09M2] Process movement commands
Scenario (sports analytics):
Context:
A sports analytics simulation moves an agent on a grid from (0,0) given movement commands.
Your Task:
Parse commands like N2, E1, S3, W4, validate them, and compute final (x,y).
Data & Edge Cases:
Ignore invalid tokens; N increases y, E increases x.
AI Assistance Expectation:
Use AI to scaffold parsing and state updates; add tests for invalid tokens.
Constraints & Notes:
Return final (x,y) tuple.
Sample Input
['N2', 'E1', 'S1', 'E2']
Sample Output
(3,1)
Acceptance Criteria: Validates tokens; handles negatives not required.

PROMPT:-

Parse movement commands like N2, E1, S1, E2, validate them, and compute the final (x, y) position on a grid starting from (0, 0). Ignore invalid tokens. N increases y, E increases x. Return the final (x, y) tuple.

CODE:-

AB_TEST-02.ANUSHA.py > ...

```python
def parse_and_move(commands):
    x, y = 0, 0
    valid_dirs = {'N', 'S', 'E', 'W'}
    for cmd in commands:
        # Validate: must be at least 2 chars, direction valid, steps positive integer
        if len(cmd) < 2 or cmd[0] not in valid_dirs or not cmd[1:].isdigit():
            continue
        direction = cmd[0]
        steps = int(cmd[1:])
        if direction == 'N':
            y += steps
        elif direction == 'S':
            y -= steps
        elif direction == 'E':
            x += steps
        elif direction == 'W':
            x -= steps
    return (x, y)

# Sample test
commands = ['N2', 'E1', 'S1', 'E2']
print(parse_and_move(commands))  # Output: (3, 1)
```

OUTPUT:-

```
[Running] python -u "c:\Users\PEDDAPELLI ANUSHA\OneDrive\Desktop\Btech.2nd yr\AI\LAB_TEST-02.ANUSHA.py"
(3, 1)

[Done] exited with code=0 in 0.419 seconds
```

OBSERVATION:-

The code correctly parses each command, validates direction and steps, and updates the agent's position.
Invalid commands are ignored.
For the sample input ['N2', 'E1', 'S1', 'E2'], the output is (3, 1), which matches the expected result.
The function meets the scenario requirements and handles edge cases as specified.