Program Structure and Development

Mingzheng Michael Huang

- **Initial Setup: File Reading (read function):** Opens input file, reads simulation parameters (number of particles, arena size, etc.), and initializes global particles and collisions arrays with simulation data.
- **Main Simulation Loop (main function):**Collision Sorting (sortColArr function): Implements insertion sort to prioritize upcoming events, ensuring the next collision processed is always the earliest. **Event Processing (jumpCol function):** Determines the type of the earliest collision (particle-wall or particle-particle) and delegates to the respective update function.
- **Collision Handling: Particle-Wall Collisions (updatePW function):** Updates particle state after a wall collision, inverts velocity if it collides with the boundary. **Particle-Particle Collisions (updatePP function):** Swaps velocities between particles post-collision to simulate elastic interaction.
- **Dynamic Collision Updating:** Array Updates (updateColArr function): After a collision, recalculates potential collisions involving the affected particles and updates the collision array accordingly.
- **Advancement and Finalization:** Position Updates (updateParArr function): Adjusts positions of all particles to the time of the last processed collision. **Simulation Completion:** Finalizes particle positions at user-specified end time, prints final states, and releases resources.
- **Flow Control Constructs: Loops:** Used to process events in time order and update the state of all particles and potential collisions continuously. **Conditional Statements:** Guide the logic flow, distinguishing between collision types and ensuring correct updates only where necessary.

Automated Validation with Bash Script - Overview

Script Functionality:

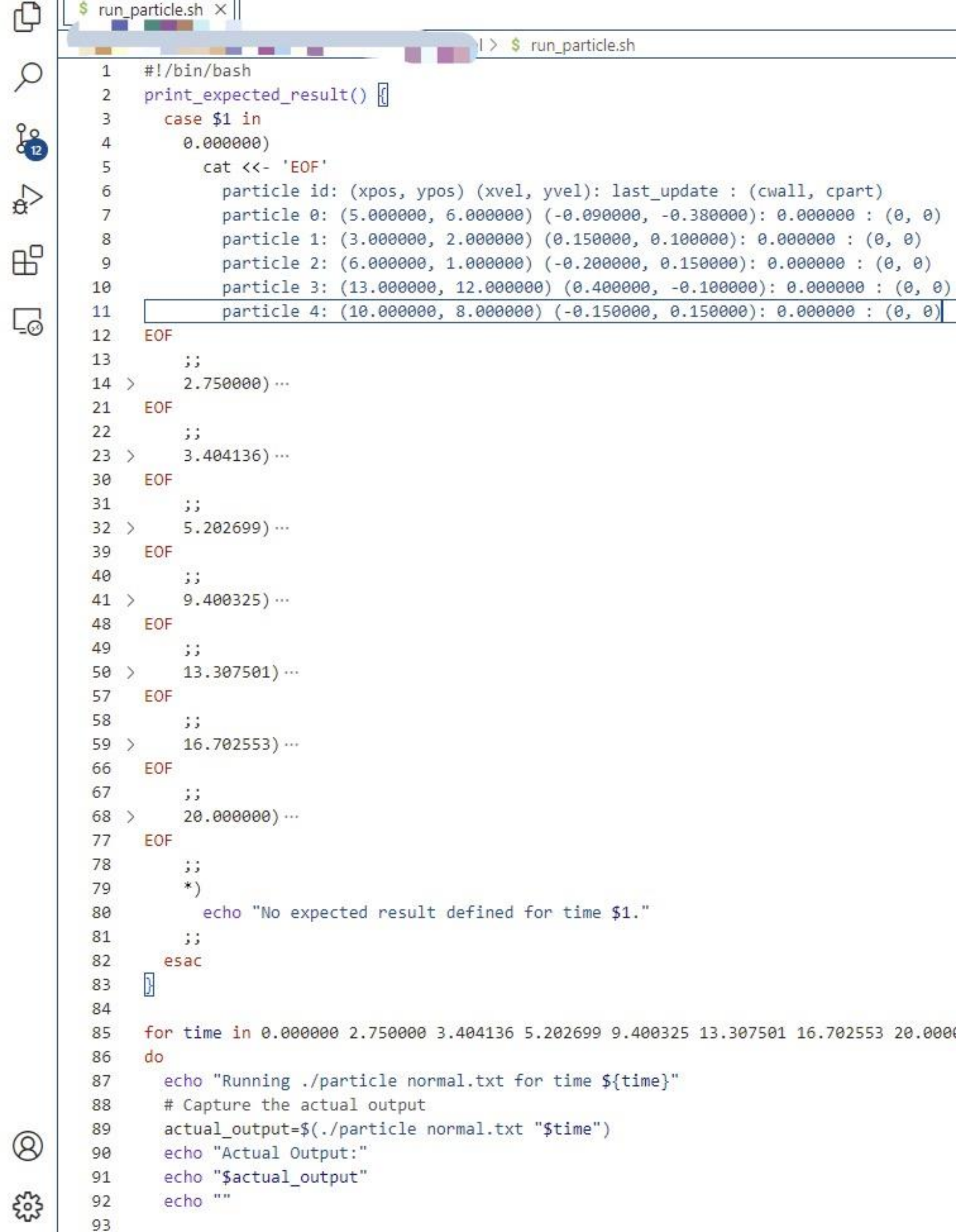
- A Bash script is designed to run the particle simulation program at predefined times.
- For each time value, the script compares the actual output of the program with the expected results.

Efficiency in Testing:

- Automating the process allows for multiple test cases to be run sequentially without manual intervention.
- This ensures comprehensive coverage over a range of different scenarios.

Precision of Evaluation:

- The script captures the actual output of the simulation and displays it alongside pre-defined expected outcomes for verification.
- Any discrepancies are easily noted, allowing for quick debugging and validation.



```bash
#!/bin/bash
print_expected_result() {
  case $1 in
    0.000000)
      cat <<- 'EOF'
        particle id: (xpos, ypos) (xvel, yvel): last_update : (cwall, cpart)
        particle 0: (5.000000, 6.000000) (-0.090000, -0.380000): 0.000000 : (0, 0)
        particle 1: (3.000000, 2.000000) (0.150000, 0.100000): 0.000000 : (0, 0)
        particle 2: (6.000000, 1.000000) (-0.200000, 0.150000): 0.000000 : (0, 0)
        particle 3: (13.000000, 12.000000) (0.400000, -0.100000): 0.000000 : (0, 0)
        particle 4: (10.000000, 8.000000) (-0.150000, 0.150000): 0.000000 : (0, 0)
EOF
    ;;
    2.750000)···
EOF
    ;;
    3.404136)···
EOF
    ;;
    5.202699)···
EOF
    ;;
    9.400325)···
EOF
    ;;
    13.307501)···
EOF
    ;;
    16.702553)···
EOF
    ;;
    20.000000)···
EOF
    ;;
    *)
      echo "No expected result defined for time $1."
    ;;
  esac
}

for time in 0.000000 2.750000 3.404136 5.202699 9.400325 13.307501 16.702553 20.0000
do
  echo "Running ./particle normal.txt for time ${time}"
  # Capture the actual output
  actual_output=$(./particle normal.txt "$time")
  echo "Actual Output:"
  echo "$actual_output"
  echo ""

```

Automated Validation with Bash Script - Execution and Results

Execution Process:

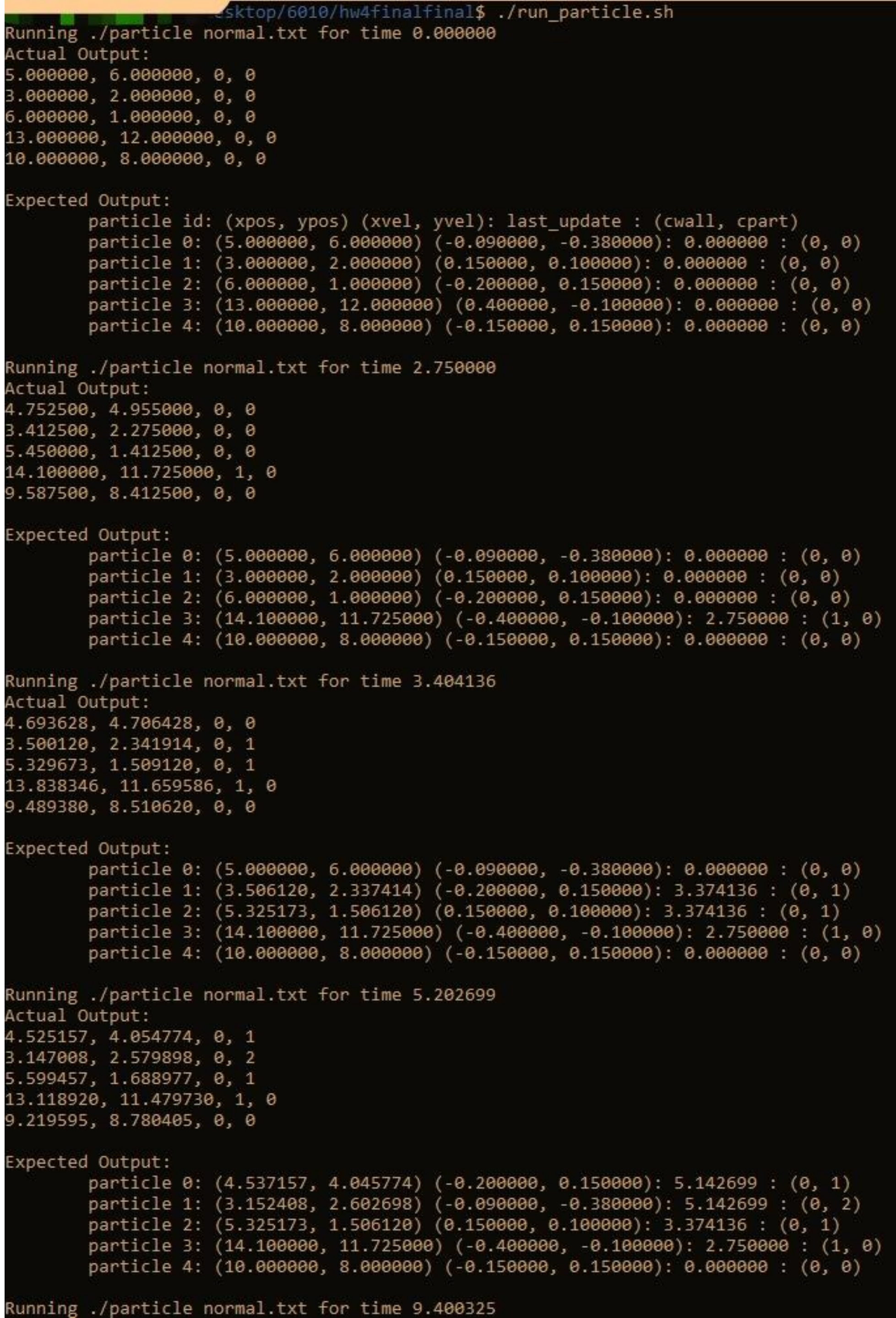
- The script loops through an array of time values, running the particle simulation for each.
- After each run, it captures the simulation's output and invokes a function to print the expected results.

Case-by-Case Validation:

- Each case within the print_expected_result function corresponds to a time value where the output is known.
- The expected output for each time value would be filled in the script, corresponding to theoretical or previously confirmed results.

Outcome and Insights:

- The actual output is echoed on the console, followed by the expected output, facilitating direct comparison.
- This comparison serves as a regression test to ensure that any changes in the code do not alter the correct behavior of the program.
- By confirming the actual output matches the expected results at various times, we validate the program's consistent performance across multiple scenarios.



```
sktop/6010/hw4finalfinal$ ./run_particle.sh
Running ./particle normal.txt for time 0.000000
Actual Output:
5.000000, 6.000000, 0, 0
3.000000, 2.000000, 0, 0
6.000000, 1.000000, 0, 0
13.000000, 12.000000, 0, 0
10.000000, 8.000000, 0, 0

Expected Output:
        particle id: (xpos, ypos) (xvel, yvel): last_update : (cwall, cpart)
        particle 0: (5.000000, 6.000000) (-0.090000, -0.380000): 0.000000 : (0, 0)
        particle 1: (3.000000, 2.000000) (0.150000, 0.100000): 0.000000 : (0, 0)
        particle 2: (6.000000, 1.000000) (-0.200000, 0.150000): 0.000000 : (0, 0)
        particle 3: (13.000000, 12.000000) (0.400000, -0.100000): 0.000000 : (0, 0)
        particle 4: (10.000000, 8.000000) (-0.150000, 0.150000): 0.000000 : (0, 0)

Running ./particle normal.txt for time 2.750000
Actual Output:
4.752500, 4.955000, 0, 0
3.412500, 2.275000, 0, 0
5.450000, 1.412500, 0, 0
14.100000, 11.725000, 1, 0
9.587500, 8.412500, 0, 0

Expected Output:
        particle 0: (5.000000, 6.000000) (-0.090000, -0.380000): 0.000000 : (0, 0)
        particle 1: (3.000000, 2.000000) (0.150000, 0.100000): 0.000000 : (0, 0)
        particle 2: (6.000000, 1.000000) (-0.200000, 0.150000): 0.000000 : (0, 0)
        particle 3: (14.100000, 11.725000) (-0.400000, -0.100000): 2.750000 : (1, 0)
        particle 4: (10.000000, 8.000000) (-0.150000, 0.150000): 0.000000 : (0, 0)

Running ./particle normal.txt for time 3.404136
Actual Output:
4.693628, 4.706428, 0, 0
3.500120, 2.341914, 0, 1
5.329673, 1.509120, 0, 1
13.838346, 11.659586, 1, 0
9.489380, 8.510620, 0, 0

Expected Output:
        particle 0: (5.000000, 6.000000) (-0.090000, -0.380000): 0.000000 : (0, 0)
        particle 1: (3.506120, 2.337414) (-0.200000, 0.150000): 3.374136 : (0, 1)
        particle 2: (5.325173, 1.506120) (0.150000, 0.100000): 3.374136 : (0, 1)
        particle 3: (14.100000, 11.725000) (-0.400000, -0.100000): 2.750000 : (1, 0)
        particle 4: (10.000000, 8.000000) (-0.150000, 0.150000): 0.000000 : (0, 0)

Running ./particle normal.txt for time 5.202699
Actual Output:
4.525157, 4.054774, 0, 1
3.147008, 2.579898, 0, 2
5.599457, 1.688977, 0, 1
13.118920, 11.479730, 1, 0
9.219595, 8.780405, 0, 0

Expected Output:
        particle 0: (4.537157, 4.045774) (-0.200000, 0.150000): 5.142699 : (0, 1)
        particle 1: (3.152408, 2.602698) (-0.090000, -0.380000): 5.142699 : (0, 2)
        particle 2: (5.325173, 1.506120) (0.150000, 0.100000): 3.374136 : (0, 1)
        particle 3: (14.100000, 11.725000) (-0.400000, -0.100000): 2.750000 : (1, 0)
        particle 4: (10.000000, 8.000000) (-0.150000, 0.150000): 0.000000 : (0, 0)

Running ./particle normal.txt for time 9.400325
```