Assignment No. 3

Implement Program for CPU Scheduling Algorithms.

```
// FCFS

#include <iostream>
#include <iomanip>
using namespace std;

struct Process {
   int pid;
   int at;
   int bt;
   int ct;
   int tat;
   int wt;
};

// Sort by Arrival Time
void sortByArrival(Process proc[], int n) {
   for (int i = 0; i < n - 1; i++) {
      for (int j = i + 1; j < n; j++) {
         if (proc[i].at > proc[j].at) {
            Process temp = proc[i];
            proc[i] = proc[j];
            proc[j] = temp;
         }
      }
   }
}
```

```
// Calculate Completion, TAT, WT
void findTimes(Process proc[], int n) {
    proc[0].ct = proc[0].at + proc[0].bt;
    proc[0].tat = proc[0].ct - proc[0].at;
    proc[0].wt = proc[0].tat - proc[0].bt;

    for (int i = 1; i < n; i++) {
        if (proc[i].at > proc[i-1].ct) {
            proc[i].ct = proc[i].at + proc[i].bt; // CPU idle
        } else {
            proc[i].ct = proc[i-1].ct + proc[i].bt;
        }
        proc[i].tat = proc[i].ct - proc[i].at;
        proc[i].wt = proc[i].tat - proc[i].bt;
    }
}

// Square-box style Gantt Chart
void printGanttChart(Process proc[], int n) {
    cout << "\nGantt Chart :\n\n";

    // Top border
    for (int i = 0; i < n; i++) {
        cout << "+-----";
    }
    cout << "+\n";

    // Process IDs inside squares
    for (int i = 0; i < n; i++) {
        cout << "| P" << setw(2) << proc[i].pid << " ";
    }
```

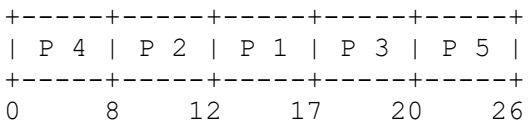
```
    cout << "|\n";

    // Bottom border
    for (int i = 0; i < n; i++) {
        cout << "+-----";
    }
    cout << "+\n";

    // Timeline
    cout << proc[0].at;
    for (int i = 0; i < n; i++) {
        cout << setw(6) << proc[i].ct;
    }
    cout << endl;
}

void findAvgTime(Process proc[], int n) {
    int total_wt = 0, total_tat = 0;

    cout << "\nProcesses  AT\t BT\t CT\t TAT \t WT\n";
    for (int i = 0; i < n; i++) {
        cout << "P" << proc[i].pid << "\t   "
             << proc[i].at << "\t  "
             << proc[i].bt << "\t  "
             << proc[i].ct << "\t  "
             << proc[i].tat << "\t  "
             << proc[i].wt << endl;

        total_wt += proc[i].wt;
        total_tat += proc[i].tat;
    }
```

```
    cout << "\nAverage Waiting Time = " << (float)total_wt / n;
    cout << "\nAverage Turnaround Time = " << (float)total_tat / n << endl;

    printGanttChart(proc, n);
}
int main() {
    int n;
    cout << "Enter the number of processes: ";
    cin >> n;

    Process proc[n];
    for (int i = 0; i < n; i++) {
        cout << "Enter arrival time and burst time for process " << i + 1 << ": ";
        cin >> proc[i].at >> proc[i].bt;
        proc[i].pid = i + 1;
    }

    sortByArrival(proc, n);
    findTimes(proc, n);
    findAvgTime(proc, n);

    return 0;
}
// output :
bvcoew@bvcoew-OptiPlex-3000:~$ g++ fcfs.cpp -o fcfs
bvcoew@bvcoew-OptiPlex-3000:~$ ./fcfs

Enter the number of processes: 5
Enter arrival time and burst time for process 1: 7 5
Enter arrival time and burst time for process 2: 3 4
Enter arrival time and burst time for process 3: 10 3
Enter arrival time and burst time for process 4: 0 8
Enter arrival time and burst time for process 5: 12 6
```

Processes	AT	BT	CT	TAT	WT
P4	0	8	8	8	0
P2	3	4	12	9	5
P1	7	5	17	10	5
P3	10	3	20	10	7
P5	12	6	26	14	8

Average Waiting Time = 5
Average Turnaround Time = 10.2

Gantt Chart :



```
// SJF Non Preemptive

#include <iostream>
#include <iomanip>
using namespace std;

struct Process {
    int pid; // Process ID
    int at;  // Arrival Time
    int bt;  // Burst Time
    int ct;  // Completion Time
    int tat; // Turnaround Time
    int wt;  // Waiting Time
};

// To store execution order
struct Gantt {
    int pid;
    int start;
    int end;
};

void sortByArrival(Process proc[], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (proc[i].at > proc[j].at) {
                Process temp = proc[i];
                proc[i] = proc[j];
                proc[j] = temp;
            }
```

```
            else if (proc[i].at == proc[j].at && proc[i].bt > proc[j].bt) {
                Process temp = proc[i];
                proc[i] = proc[j];
                proc[j] = temp;
            }
        }
    }
}

int findTimes(Process proc[], int n, Gantt gantt[]) {
    int time = 0, completed = 0, gIndex = 0;
    bool done[100] = {false};

    while (completed < n) {
        int idx = -1, minBT = 1e9;

        // Find process with min burst among arrived processes
        for (int i = 0; i < n; i++) {
            if (!done[i] && proc[i].at <= time) {
                if (proc[i].bt < minBT) {
                    minBT = proc[i].bt;
                    idx = i;
                }
            }
        }

        if (idx == -1) {
            time++; // CPU idle
        } else {
            gantt[gIndex].pid = proc[idx].pid;
            gantt[gIndex].start = time;
```

```
            time += proc[idx].bt;
            proc[idx].ct = time;
            proc[idx].tat = proc[idx].ct - proc[idx].at;
            proc[idx].wt = proc[idx].tat - proc[idx].bt;
            done[idx] = true;
            gantt[gIndex].end = time;
            gIndex++;
            completed++;
        }
    }
    return n; // number of executed processes (same as n)
}

void display(Process proc[], int n) {
    int total_wt = 0, total_tat = 0;

    cout << "\nProcesses\tAT\tBT\tCT\tTAT\tWT\n";
    for (int i = 0; i < n; i++) {
        cout << "P" << proc[i].pid << "\t\t"
             << proc[i].at << "\t" << proc[i].bt << "\t"
             << proc[i].ct << "\t" << proc[i].tat << "\t" << proc[i].wt << endl;

        total_wt += proc[i].wt;
        total_tat += proc[i].tat;
    }

    cout << "\nAverage Waiting Time = " << (float)total_wt / n;
    cout << "\nAverage Turnaround Time = " << (float)total_tat / n << endl;
}

void printGanttChart(Gantt gantt[], int n) {
```

```
    cout << "\nGantt Chart :\n\n";

    // Top border
    for (int i = 0; i < n; i++) cout << "+-----";
    cout << "+\n";

    // Process IDs
    for (int i = 0; i < n; i++) {
        cout << "| P" << setw(2) << gantt[i].pid << " ";
    }
    cout << "|\n";

    // Bottom border
    for (int i = 0; i < n; i++) cout << "+-----";
    cout << "+\n";

    // Timeline
    cout << gantt[0].start;
    for (int i = 0; i < n; i++) {
        cout << setw(6) << gantt[i].end;
    }
    cout << endl;
}

int main() {
    int n;
    cout << "Enter number of processes: ";
    cin >> n;

    Process proc[n];
    for (int i = 0; i < n; i++) {
```

```
        cout << "Enter Arrival Time and Burst Time for Process " << i + 1 << ": ";
        cin >> proc[i].at >> proc[i].bt;
        proc[i].pid = i + 1;
    }

    sortByArrival(proc, n);

    Gantt gantt[n];
    int gCount = findTimes(proc, n, gantt);

    display(proc, n);
    printGanttChart(gantt, gCount);

    return 0;
}
```

// output :

```
Enter number of processes: 5
Enter Arrival Time and Burst Time for Process 1: 7 5
Enter Arrival Time and Burst Time for Process 2: 3 4
Enter Arrival Time and Burst Time for Process 3: 10 3
Enter Arrival Time and Burst Time for Process 4: 0 8
Enter Arrival Time and Burst Time for Process 5: 12 6

Processes       AT      BT      CT      TAT     WT
P4              0       8       8       8       0
P2              3       4       12      9       5
P1              7       5       20      13      8
P3              10      3       15      5       2
P5              12      6       26      14      8

Average Waiting Time = 4.6
Average Turnaround Time = 9.8

Gantt Chart :

+-----+-----+-----+-----+-----+
| P 4 | P 2 | P 3 | P 1 | P 5 |
+-----+-----+-----+-----+-----+
0     8    12    15    20    26
```

```
// SJF Preemptive

#include <iostream>
#include <iomanip>
#include <climits>
using namespace std;

struct Process {
    int pid;  // Process ID
    int at;   // Arrival Time
    int bt;   // Burst Time
    int ct;   // Completion Time
    int tat;  // Turnaround Time
    int wt;   // Waiting Time
    int rt;   // Remaining Time
};

// Slot for Gantt chart
struct Slot {
    int pid;   // process id (-1 = idle)
    int start; // start time
    int end;   // end time
};

// Function to calculate times and prepare Gantt chart
int findTimes(Process proc[], int n, Slot slots[]) {
    int time = 0, completed = 0;
    int prev = -2; // previous process ID
    int k = 0;     // slot index
```

```
while (completed < n) {
    int idx = -1, minRT = INT_MAX;

    // Select process with min remaining time among arrived
    for (int i = 0; i < n; i++) {
        if (proc[i].at <= time && proc[i].rt > 0) {
            if (proc[i].rt < minRT) {
                minRT = proc[i].rt;
                idx = i;
            } else if (proc[i].rt == minRT && proc[i].at < proc[idx].at) {
                idx = i;
            }
        }
    }

    if (idx == -1) { // CPU idle
        if (prev != -1) { // new idle slot
            slots[k].end = time;
            k++;
        }
        slots[k].pid = -1; // idle
        slots[k].start = time;
        prev = -1;
        time++;
    } else {
        if (prev != proc[idx].pid) { // context switch
            if (prev != -2) {
                slots[k].end = time; // close previous slot
                k++;
            }
            slots[k].pid = proc[idx].pid;
```

```
            slots[k].start = time;
            prev = proc[idx].pid;
        }
        proc[idx].rt--;
        time++;

        if (proc[idx].rt == 0) {
            proc[idx].ct = time;
            proc[idx].tat = proc[idx].ct - proc[idx].at;
            proc[idx].wt = proc[idx].tat - proc[idx].bt;
            completed++;
        }
      }
   }

   slots[k].end = time; // close last slot
   k++;
   return k; // return number of slots
}

// Display process table
void display(Process proc[], int n) {
   int total_wt = 0, total_tat = 0;

   cout << "\nProcesses\tAT\tBT\tCT\tTAT\tWT\n";
   for (int i = 0; i < n; i++) {
      cout << "P" << proc[i].pid << "\t\t"
           << proc[i].at << "\t" << proc[i].bt << "\t"
           << proc[i].ct << "\t" << proc[i].tat << "\t" << proc[i].wt << endl;

      total_wt += proc[i].wt;
```

```
        total_tat += proc[i].tat;
    }

    cout << "\nAverage Waiting Time = " << (float)total_wt / n;
    cout << "\nAverage Turnaround Time = " << (float)total_tat / n << endl;
}

// Print Gantt Chart (square style)
void printGanttChart(Slot slots[], int k) {
    cout << "\nGantt Chart (SJF Preemptive):\n\n";

    for (int i = 0; i < k; i++) cout << "+-----";
    cout << "+\n";

    for (int i = 0; i < k; i++) {
        if (slots[i].pid == -1)
            cout << "| IDLE";
        else
            cout << "| P" << setw(2) << slots[i].pid << " ";
    }
    cout << "|\n";

    for (int i = 0; i < k; i++) cout << "+-----";
    cout << "+\n";

    cout << slots[0].start;
    for (int i = 0; i < k; i++) cout << setw(6) << slots[i].end;
    cout << endl;
}

int main() {
```

```
    int n;
    cout << "Enter number of processes: ";
    cin >> n;

    Process proc[n];
    for (int i = 0; i < n; i++) {
        cout << "Enter Arrival Time and Burst Time for Process " << i + 1 << ": ";
        cin >> proc[i].at >> proc[i].bt;
        proc[i].pid = i + 1;
        proc[i].rt = proc[i].bt; // Initialize remaining time
    }

    Slot slots[200];
    int k = findTimes(proc, n, slots);
    display(proc, n);
    printGanttChart(slots, k);
    return 0;
}
```

// output

```
Enter number of processes: 4
Enter Arrival Time and Burst Time for Process 1: 0 3
Enter Arrival Time and Burst Time for Process 2: 1 6
Enter Arrival Time and Burst Time for Process 3: 4 4
Enter Arrival Time and Burst Time for Process 4: 6 2

Processes       AT      BT      CT      TAT     WT
P1              0       3       3       3       0
P2              1       6       15      14      8
P3              4       4       8       4       0
P4              6       2       10      4       2

Average Waiting Time = 2.5
Average Turnaround Time = 6.25

Gantt Chart (SJF Preemptive):

+-----+-----+-----+-----+-----+
| P 1 | P 2 | P 3 | P 4 | P 2 |
+-----+-----+-----+-----+-----+
```

```
0     3     4     8    10    15

// Priority Non Preemptive

#include <iostream>
#include <iomanip>
#include <algorithm>
using namespace std;

struct Process {
    int pid;   // Process ID
    int at;    // Arrival Time
    int bt;    // Burst Time
    int pr;    // Priority (lower number = higher priority)
    int ct;    // Completion Time
    int tat;   // Turnaround Time
    int wt;    // Waiting Time
    bool done; // Finished or not
};

// Slot for Gantt chart
struct Slot {
    int pid;   // process id (-1 = idle)
    int start; // start time
    int end;   // end time
};

// Find times and build Gantt chart
int priorityScheduling(Process proc[], int n, Slot slots[]) {
    int time = 0, completed = 0, k = 0;

    while (completed < n) {
        int idx = -1;
        int bestPr = 999999;

        // Find highest priority process among arrived & not completed
        for (int i = 0; i < n; i++) {
            if (!proc[i].done && proc[i].at <= time) {
                if (proc[i].pr < bestPr) {
                    bestPr = proc[i].pr;
                    idx = i;
                }
                else if (proc[i].pr == bestPr && proc[i].at <
proc[idx].at) {
                    idx = i;
                }
            }
        }

        if (idx == -1) { // CPU idle
            slots[k].pid = -1;
            slots[k].start = time;
            time++;
            slots[k].end = time;
            k++;
        } else {
            slots[k].pid = proc[idx].pid;
            slots[k].start = time;
```

```
            time += proc[idx].bt;
            proc[idx].ct = time;
            proc[idx].tat = proc[idx].ct - proc[idx].at;
            proc[idx].wt = proc[idx].tat - proc[idx].bt;
            proc[idx].done = true;
            completed++;

            slots[k].end = time;
            k++;
        }
    }

    return k; // number of slots
}

// Display process table
void display(Process proc[], int n) {
    float total_wt = 0, total_tat = 0;

    cout << "\nProcesses\tAT\tBT\tPR\tCT\tTAT\tWT\n";
    for (int i = 0; i < n; i++) {
        cout << "P" << proc[i].pid << "\t\t"
             << proc[i].at << "\t" << proc[i].bt << "\t" << proc[i].pr
<< "\t"
             << proc[i].ct << "\t" << proc[i].tat << "\t" << proc[i].wt
<< endl;

        total_wt += proc[i].wt;
        total_tat += proc[i].tat;
    }

    cout << "\nAverage Waiting Time = " << total_wt / n;
    cout << "\nAverage Turnaround Time = " << total_tat / n << endl;
}

// Print Gantt Chart (square style)
void printGanttChart(Slot slots[], int k) {
    cout << "\nGantt Chart (Priority Non-Preemptive):\n\n";

    for (int i = 0; i < k; i++) cout << "+-----";
    cout << "+\n";

    for (int i = 0; i < k; i++) {
        if (slots[i].pid == -1)
            cout << "| IDLE";
        else
            cout << "| P" << setw(2) << slots[i].pid << " ";
    }
    cout << "|\n";

    for (int i = 0; i < k; i++) cout << "+-----";
    cout << "+\n";

    cout << slots[0].start;
    for (int i = 0; i < k; i++) cout << setw(6) << slots[i].end;
    cout << endl;
}
```

```
int main() {
    int n;
    cout << "Enter number of processes: ";
    cin >> n;

    Process proc[n];
    for (int i = 0; i < n; i++) {
        cout << "Enter AT, BT, Priority for Process " << i + 1 << ": ";
        cin >> proc[i].at >> proc[i].bt >> proc[i].pr;
        proc[i].pid = i + 1;
        proc[i].done = false;
    }

    Slot slots[200];
    int k = priorityScheduling(proc, n, slots);

    display(proc, n);
    printGanttChart(slots, k);

    return 0;
}
```

// output :

```
Enter number of processes: 4
Enter AT, BT, Priority for Process 1: 0 3 2
Enter AT, BT, Priority for Process 2: 1 6 1
Enter AT, BT, Priority for Process 3: 4 4 3
Enter AT, BT, Priority for Process 4: 6 2 4

Processes       AT      BT      PR      CT      TAT     WT
P1              0       3       2       3       3       0
P2              1       6       1       9       8       2
P3              4       4       3       13      9       5
P4              6       2       4       15      9       7

Average Waiting Time = 3.5
Average Turnaround Time = 7.25

Gantt Chart (Priority Non-Preemptive):

+-----+-----+-----+-----+
| P 1 | P 2 | P 3 | P 4 |
+-----+-----+-----+-----+
0     3     9    13    15
```

```
// Priority Preemptive

#include <iostream>
#include <iomanip>
using namespace std;

struct Process {
    int pid;   // Process ID
    int at;    // Arrival Time
    int bt;    // Burst Time
    int pr;    // Priority (lower = higher priority)
    int ct;    // Completion Time
    int tat;   // Turnaround Time
    int wt;    // Waiting Time
    int rt;    // Remaining Time
    bool done; // Finished or not
};

struct Slot {
    int pid;   // process id (-1 = idle)
    int start;
    int end;
};

// Priority Preemptive Scheduling
int priorityPreemptive(Process proc[], int n, Slot slots[]) {
    int time = 0, completed = 0, k = 0;
    for (int i = 0; i < n; i++) proc[i].rt = proc[i].bt;

    while (completed < n) {
```

```
int idx = -1;
int bestPr = 999999;

// Find highest priority among arrived
for (int i = 0; i < n; i++) {
   if (proc[i].rt > 0 && proc[i].at <= time) {
      if (proc[i].pr < bestPr) {
         bestPr = proc[i].pr;
         idx = i;
      }
      else if (proc[i].pr == bestPr && proc[i].at < proc[idx].at) {
         idx = i;
      }
   }
}

if (idx == -1) { // CPU idle
   slots[k].pid = -1;
   slots[k].start = time;
   time++;
   slots[k].end = time;
   k++;
} else {
   slots[k].pid = proc[idx].pid;
   slots[k].start = time;

   // Run for 1 unit
   time++;
   proc[idx].rt--;

   slots[k].end = time;
```

```
            // If completed
            if (proc[idx].rt == 0) {
                proc[idx].ct = time;
                proc[idx].tat = proc[idx].ct - proc[idx].at;
                proc[idx].wt = proc[idx].tat - proc[idx].bt;
                proc[idx].done = true;
                completed++;
            }

            // Merge consecutive same process slots
            if (k > 0 && slots[k].pid == slots[k-1].pid) {
                slots[k-1].end = slots[k].end;
                k--;
            }
            k++;
        }
    }

    return k; // number of slots
}

// Display process table
void display(Process proc[], int n) {
    float total_wt = 0, total_tat = 0;

    cout << "\nProcesses\tAT\tBT\tPR\tCT\tTAT\tWT\n";
    for (int i = 0; i < n; i++) {
        cout << "P" << proc[i].pid << "\t\t"
             << proc[i].at << "\t" << proc[i].bt << "\t" << proc[i].pr << "\t"
             << proc[i].ct << "\t" << proc[i].tat << "\t" << proc[i].wt << endl;
```

```
        total_wt += proc[i].wt;
        total_tat += proc[i].tat;
    }

    cout << "\nAverage Waiting Time = " << total_wt / n;
    cout << "\nAverage Turnaround Time = " << total_tat / n << endl;
}

// Print Gantt Chart
void printGanttChart(Slot slots[], int k) {
    cout << "\nGantt Chart (Priority Preemptive):\n\n";

    for (int i = 0; i < k; i++) cout << "+-----";
    cout << "+\n";

    for (int i = 0; i < k; i++) {
        if (slots[i].pid == -1) cout << "| IDLE";
        else cout << "| P" << setw(2) << slots[i].pid << " ";
    }
    cout << "|\n";

    for (int i = 0; i < k; i++) cout << "+-----";
    cout << "+\n";

    cout << slots[0].start;
    for (int i = 0; i < k; i++) cout << setw(6) << slots[i].end;
    cout << endl;
}

int main() {
    int n;
```

```
    cout << "Enter number of processes: ";
    cin >> n;

    Process proc[n];
    for (int i = 0; i < n; i++) {
        cout << "Enter AT, BT, Priority for Process " << i + 1 << ": ";
        cin >> proc[i].at >> proc[i].bt >> proc[i].pr;
        proc[i].pid = i + 1;
        proc[i].done = false;
    }

    Slot slots[500];
    int k = priorityPreemptive(proc, n, slots);

    display(proc, n);
    printGanttChart(slots, k);
    return 0;
}
```

// output :

```
Enter number of processes: 4
Enter AT, BT, Priority for Process 1: 0 3 2
Enter AT, BT, Priority for Process 2: 1 6 1
Enter AT, BT, Priority for Process 3: 4 4 3
Enter AT, BT, Priority for Process 4: 6 2 4

Processes       AT      BT      PR      CT      TAT     WT
P1              0       3       2       9       9       6
P2              1       6       1       7       6       0
P3              4       4       3       13      9       5
P4              6       2       4       15      9       7

Average Waiting Time = 4.5
Average Turnaround Time = 8.25

Gantt Chart (Priority Preemptive):

+-----+-----+-----+-----+-----+
| P 1 | P 2 | P 1 | P 3 | P 4 |
+-----+-----+-----+-----+-----+
0     1     7     9     13    15
```

```
// Round Robin :

#include <iostream>
#include <queue>
#include <iomanip>
using namespace std;

struct Process {
    int pid;   // Process ID
    int at;    // Arrival Time
    int bt;    // Burst Time
    int ct;    // Completion Time
    int tat;   // Turnaround Time
    int wt;    // Waiting Time
    int rt;    // Remaining Time
};

struct Slot {
    int pid;
    int start;
    int end;
};

// Round Robin Scheduling
int roundRobin(Process proc[], int n, int tq, Slot slots[]) {
    queue<int> q;
    int time = 0, completed = 0, k = 0;
    bool inQueue[n] = {false};

    for (int i = 0; i < n; i++) proc[i].rt = proc[i].bt;

    // Start at the first process arrival
    while (completed < n) {
        // Push new arrivals
        for (int i = 0; i < n; i++) {
            if (!inQueue[i] && proc[i].at <= time && proc[i].rt > 0) {
                q.push(i);
                inQueue[i] = true;
            }
        }

        if (q.empty()) { // CPU idle
            slots[k].pid = -1;
            slots[k].start = time;
            time++;
            slots[k].end = time;
            k++;
            continue;
        }
        int idx = q.front();
        q.pop();

        slots[k].pid = proc[idx].pid;
        slots[k].start = time;

        int execTime = min(tq, proc[idx].rt);
        proc[idx].rt -= execTime;
        time += execTime;
```

```
        slots[k].end = time;

        // Push new arrivals during execution
        for (int i = 0; i < n; i++) {
            if (!inQueue[i] && proc[i].at <= time && proc[i].rt > 0) {
                q.push(i);
                inQueue[i] = true;
            }
        }

        if (proc[idx].rt > 0) {
            q.push(idx); // put back in queue
        } else {
            proc[idx].ct = time;
            proc[idx].tat = proc[idx].ct - proc[idx].at;
            proc[idx].wt = proc[idx].tat - proc[idx].bt;
            completed++;
        }

        // Merge same consecutive slots
        if (k > 0 && slots[k].pid == slots[k-1].pid) {
            slots[k-1].end = slots[k].end;
            k--;
        }
        k++;
    }

    return k; // number of slots
}

// Display Process Table
void display(Process proc[], int n) {
    float total_wt = 0, total_tat = 0;
    cout << "\nProcesses\tAT\tBT\tCT\tTAT\tWT\n";
    for (int i = 0; i < n; i++) {
        cout << "P" << proc[i].pid << "\t\t"
             << proc[i].at << "\t" << proc[i].bt << "\t"
             << proc[i].ct << "\t" << proc[i].tat << "\t" << proc[i].wt
<< endl;
        total_wt += proc[i].wt;
        total_tat += proc[i].tat;
    }
    cout << "\nAverage Waiting Time = " << total_wt / n;
    cout << "\nAverage Turnaround Time = " << total_tat / n << endl;
}

// Print Gantt Chart
void printGanttChart(Slot slots[], int k) {
    cout << "\nGantt Chart (Round Robin):\n\n";

    for (int i = 0; i < k; i++) cout << "+-----";
    cout << "+\n";

    for (int i = 0; i < k; i++) {
        if (slots[i].pid == -1) cout << "| IDLE";
        else cout << "| P" << setw(2) << slots[i].pid << " ";
    }
```

```
    cout << "|\n";

    for (int i = 0; i < k; i++) cout << "+-----";
    cout << "+\n";

    cout << slots[0].start;
    for (int i = 0; i < k; i++) cout << setw(6) << slots[i].end;
    cout << endl;
}

int main() {
    int n, tq;
    cout << "Enter number of processes: ";
    cin >> n;

    Process proc[n];
    for (int i = 0; i < n; i++) {
        cout << "Enter AT, BT for Process " << i+1 << ": ";
        cin >> proc[i].at >> proc[i].bt;
        proc[i].pid = i+1;
    }

    cout << "Enter Time Quantum: ";
    cin >> tq;

    Slot slots[500];
    int k = roundRobin(proc, n, tq, slots);

    display(proc, n);
    printGanttChart(slots, k);

    return 0;
}
```

// output

```
Enter number of processes: 5
Enter AT, BT for Process 1: 7 5
Enter AT, BT for Process 2: 3 4
Enter AT, BT for Process 3: 10 3
Enter AT, BT for Process 4: 0 8
Enter AT, BT for Process 5: 12 6
Enter Time Quantum: 3

Processes       AT      BT      CT      TAT     WT
P1              7       5       23      16      11
P2              3       4       10      7       3
P3              10      3       18      8       5
P4              0       8       15      15      7
P5              12      6       26      14      8

Average Waiting Time = 6.8
Average Turnaround Time = 12

Gantt Chart (Round Robin):
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| P 4 | P 2 | P 4 | P 2 | P 1 | P 4 | P 3 | P 5 | P 1 | P 5 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
0     3     6     9     10    13    15    18    21    23    26
```