NAME: AMIT RAJ THAKUR
PRN: 122B1B183

ASSIGNMENT 2

```
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
#include <iomanip>
#include <climits>

using namespace std;

struct Process {
    string name;
    int arrival;
    int burst;
    int priority;
    int remaining;
    int finish;
    int waiting;
    int turnaround;
};

void inputProcesses(vector<Process>& processes) {
    int n;
    cout << "Enter the number of processes: ";
    cin >> n;
    for (int i = 0; i < n; i++) {
        Process p;
        cout << "Enter process name: ";
        cin >> p.name;
        cout << "Enter arrival time: ";
        cin >> p.arrival;
        cout << "Enter burst time: ";
        cin >> p.burst;
        cout << "Enter priority: ";
        cin >> p.priority;
        p.remaining = p.burst;
        p.finish = -1;
        p.waiting = 0;
        p.turnaround = 0;
        processes.push_back(p);
    }
}

void printResults(const vector<Process>& processes, const vector<string>& gantt, const string& algorithmName) {
    cout << "\nResults for " << algorithmName << ":\n";
    cout << "Process Table:\n";
    cout << "Process | Arrival | Burst | Priority | Finish | Waiting | Turnaround\n";
    for (const auto& p : processes) {
        cout << setw(7) << p.name << " | " << setw(7) << p.arrival << " | " << setw(5) << p.burst << " | " << setw(8) << p.priority << " | " << setw(6) << p.finish << " | " << setw(7) << p.waiting << " | " << setw(10) << p.turnaround << "\n";
    }

    cout << "\nGantt Chart:\n";
    for (const auto& event : gantt) {
        cout << event << " ";
    }
    cout << "\n";

    double avgWaiting = 0, avgTurnaround = 0;
    for (const auto& p : processes) {
        avgWaiting += p.waiting;
        avgTurnaround += p.turnaround;
    }
    avgWaiting /= processes.size();
    avgTurnaround /= processes.size();

    cout << "\nAverage Waiting Time: " << avgWaiting << "\n";
    cout << "Average Turnaround Time: " << avgTurnaround << "\n";
}

void FCFS(vector<Process> processes) {
    vector<string> gantt;
    int currentTime = 0;
    for (auto& p : processes) {
        if (currentTime < p.arrival) {
            currentTime = p.arrival;
        }
        p.waiting = currentTime - p.arrival;
        currentTime += p.burst;
        p.finish = currentTime;
        p.turnaround = p.finish - p.arrival;
        gantt.push_back(p.name);
    }
    printResults(processes, gantt, "FCFS");
}

void RoundRobin(vector<Process> processes, int quantum) {
    vector<string> gantt;
    queue<int> readyQueue;
    int currentTime = 0;
    int completed = 0;
    int n = processes.size();
    vector<int> remainingTime(n);
    for (int i = 0; i < n; i++) {
        remainingTime[i] = processes[i].burst;
    }

    while (completed < n) {
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival <= currentTime && remainingTime[i] > 0) {
                readyQueue.push(i);
            }
        }
        if (readyQueue.empty()) {
            currentTime++;
            continue;
        }
        int idx = readyQueue.front();
        readyQueue.pop();
        int execTime = min(quantum, remainingTime[idx]);
        remainingTime[idx] -= execTime;
        currentTime += execTime;
        gantt.push_back(processes[idx].name);
        if (remainingTime[idx] == 0) {
            completed++;
            processes[idx].finish = currentTime;
            processes[idx].turnaround = processes[idx].finish - processes[idx].arrival;
            processes[idx].waiting = processes[idx].turnaround - processes[idx].burst;
        } else {
            readyQueue.push(idx);
        }
    }
    printResults(processes, gantt, "Round Robin");
}

void SJF(vector<Process> processes, bool preemptive) {
    vector<string> gantt;
    int currentTime = 0;
    int completed = 0;
    int n = processes.size();
    vector<int> remainingTime(n);
    for (int i = 0; i < n; i++) {
        remainingTime[i] = processes[i].burst;
    }

    while (completed < n) {
        int shortest = -1;
        int minRemaining = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival <= currentTime && remainingTime[i] < minRemaining &&
remainingTime[i] > 0) {
                shortest = i;
                minRemaining = remainingTime[i];
            }
        }
        if (shortest == -1) {
            currentTime++;
            continue;
        }
        if (preemptive) {
            remainingTime[shortest]--;
            currentTime++;
            gantt.push_back(processes[shortest].name);
            if (remainingTime[shortest] == 0) {
                completed++;
                processes[shortest].finish = currentTime;
                processes[shortest].turnaround = processes[shortest].finish -
processes[shortest].arrival;
                processes[shortest].waiting = processes[shortest].turnaround -
processes[shortest].burst;
            }
        } else {
            currentTime += remainingTime[shortest];
            remainingTime[shortest] = 0;
            completed++;
            processes[shortest].finish = currentTime;
            processes[shortest].turnaround = processes[shortest].finish - processes[shortest].arrival;
            processes[shortest].waiting = processes[shortest].turnaround -
processes[shortest].burst;
            gantt.push_back(processes[shortest].name);
        }
    }
    printResults(processes, gantt, preemptive ? "SJF (Preemptive)" : "SJF (Non-Preemptive)");
}

void PriorityScheduling(vector<Process> processes, bool preemptive) {
    vector<string> gantt;
    int currentTime = 0;
    int completed = 0;
    int n = processes.size();
    vector<int> remainingTime(n);
    for (int i = 0; i < n; i++) {
        remainingTime[i] = processes[i].burst;
    }

    while (completed < n) {
        int highestPriority = -1;
        int minPriority = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival <= currentTime && remainingTime[i] > 0 && processes[i].priority <
minPriority) {
                highestPriority = i;
                minPriority = processes[i].priority;
            }
        }
        if (highestPriority == -1) {
            currentTime++;
            continue;
        }
        if (preemptive) {
            remainingTime[highestPriority]--;
            currentTime++;
            gantt.push_back(processes[highestPriority].name);
            if (remainingTime[highestPriority] == 0) {
                completed++;
                processes[highestPriority].finish = currentTime;
                processes[highestPriority].turnaround = processes[highestPriority].finish -
processes[highestPriority].arrival;
                processes[highestPriority].waiting = processes[highestPriority].turnaround -
processes[highestPriority].burst;
            }
        } else {
            currentTime += remainingTime[highestPriority];
            remainingTime[highestPriority] = 0;
            completed++;
            processes[highestPriority].finish = currentTime;
            processes[highestPriority].turnaround = processes[highestPriority].finish -
processes[highestPriority].arrival;
            processes[highestPriority].waiting = processes[highestPriority].turnaround -
processes[highestPriority].burst;
            gantt.push_back(processes[highestPriority].name);
        }
    }
    printResults(processes, gantt, preemptive ? "Priority (Preemptive)" : "Priority
(Non-Preemptive)");
}

int main() {
    vector<Process> processes;
    int choice;
    while (true) {
        cout << "\nCPU Scheduling Algorithms:\n";
        cout << "1. FCFS (First-Come, First-Served)\n";
        cout << "2. Round Robin\n";
        cout << "3. SJF (Non-Preemptive)\n";
        cout << "4. SJF (Preemptive)\n";
        cout << "5. Priority (Non-Preemptive)\n";
        cout << "6. Priority (Preemptive)\n";
        cout << "7. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        if (choice == 7) {
            break;
        }

        processes.clear();
        inputProcesses(processes);

        switch (choice) {
            case 1:
                FCFS(processes);
                break;
            case 2:
                int quantum;
                cout << "Enter time quantum: ";
                cin >> quantum;
                RoundRobin(processes, quantum);
                break;
            case 3:
                SJF(processes, false);
                break;
            case 4:
                SJF(processes, true);
                break;
            case 5:
                PriorityScheduling(processes, false);
                break;
            case 6:
                PriorityScheduling(processes, true);
                break;
            default:
                cout << "Invalid choice!\n";
        }
    }
    return 0;
}
```

OUTPUT:

```
CPU Scheduling Algorithms:
1. FCFS (First-Come, First-Served)
2. Round Robin
3. SJF (Non-Preemptive)
4. SJF (Preemptive)
5. Priority (Non-Preemptive)
6. Priority (Preemptive)
7. Exit
Enter your choice: 1
Enter the number of processes: 3
Enter process name: P1
Enter arrival time: 0
Enter burst time: 5
Enter priority: 2
Enter process name: P2
Enter arrival time: 1
Enter burst time: 3
Enter priority: 1
Enter process name: P3
Enter arrival time: 2
Enter burst time: 8
Enter priority: 3

Results for FCFS:
Process Table:
Process | Arrival | Burst | Priority | Finish | Waiting | Turnaround
     P1 |       0 |     5 |        2 |      5 |       0 |          5
     P2 |       1 |     3 |        1 |      8 |       4 |          7
     P3 |       2 |     8 |        3 |     16 |       6 |         14

Gantt Chart:
P1 P1 P1 P1 P1 P2 P2 P2 P3 P3 P3 P3 P3 P3 P3 P3

Average Waiting Time: 3.33333
Average Turnaround Time: 8.66667

CPU Scheduling Algorithms:
1. FCFS (First-Come, First-Served)
2. Round Robin
3. SJF (Non-Preemptive)
4. SJF (Preemptive)
5. Priority (Non-Preemptive)
6. Priority (Preemptive)
7. Exit
Enter your choice: 2
Enter time quantum: 2

Results for Round Robin:
Process Table:
Process | Arrival | Burst | Priority | Finish | Waiting | Turnaround
     P1 |       0 |     5 |        2 |     13 |       8 |         13
     P2 |       1 |     3 |        1 |      7 |       3 |          6
     P3 |       2 |     8 |        3 |     18 |       8 |         16

Gantt Chart:
P1 P1 P2 P2 P3 P3 P1 P1 P3 P3 P1 P3 P3 P3 P3 P3

Average Waiting Time: 6.33333
Average Turnaround Time: 11.6667

CPU Scheduling Algorithms:
1. FCFS (First-Come, First-Served)
2. Round Robin
3. SJF (Non-Preemptive)
4. SJF (Preemptive)
```

```
5. Priority (Non-Preemptive)
6. Priority (Preemptive)
7. Exit
Enter your choice: 3

Results for SJF (Non-Preemptive):
Process Table:
Process | Arrival | Burst | Priority | Finish | Waiting | Turnaround
     P1 |       0 |     5 |        2 |      5 |       0 |          5
     P2 |       1 |     3 |        1 |      8 |       4 |          7
     P3 |       2 |     8 |        3 |     16 |       6 |         14

Gantt Chart:
P1 P1 P1 P1 P1 P2 P2 P2 P3 P3 P3 P3 P3 P3 P3 P3

Average Waiting Time: 3.33333
Average Turnaround Time: 8.66667

CPU Scheduling Algorithms:
1. FCFS (First-Come, First-Served)
2. Round Robin
3. SJF (Non-Preemptive)
4. SJF (Preemptive)
5. Priority (Non-Preemptive)
6. Priority (Preemptive)
7. Exit
Enter your choice: 4

Results for SJF (Preemptive):
Process Table:
Process | Arrival | Burst | Priority | Finish | Waiting | Turnaround
     P1 |       0 |     5 |        2 |     16 |      11 |         16
     P2 |       1 |     3 |        1 |      4 |       0 |          3
     P3 |       2 |     8 |        3 |     12 |       2 |         10

Gantt Chart:
P1 P2 P2 P2 P1 P3 P3 P3 P3 P3 P3 P3 P3 P1 P1 P1

Average Waiting Time: 4.33333
Average Turnaround Time: 9.66667

CPU Scheduling Algorithms:
1. FCFS (First-Come, First-Served)
2. Round Robin
```

```
3. SJF (Non-Preemptive)
4. SJF (Preemptive)
5. Priority (Non-Preemptive)
6. Priority (Preemptive)
7. Exit
Enter your choice: 5

Results for Priority (Non-Preemptive):
Process Table:
Process | Arrival | Burst | Priority | Finish | Waiting | Turnaround
     P1 |       0 |     5 |        2 |      5 |       0 |          5
     P2 |       1 |     3 |        1 |      8 |       4 |          7
     P3 |       2 |     8 |        3 |     16 |       6 |         14

Gantt Chart:
P1 P1 P1 P1 P1 P2 P2 P2 P3 P3 P3 P3 P3 P3 P3 P3

Average Waiting Time: 3.33333
Average Turnaround Time: 8.66667

CPU Scheduling Algorithms:
1. FCFS (First-Come, First-Served)
2. Round Robin
3. SJF (Non-Preemptive)
4. SJF (Preemptive)
5. Priority (Non-Preemptive)
6. Priority (Preemptive)
7. Exit
Enter your choice: 6

Results for Priority (Preemptive):
Process Table:
Process | Arrival | Burst | Priority | Finish | Waiting | Turnaround
     P1 |       0 |     5 |        2 |     16 |      11 |         16
     P2 |       1 |     3 |        1 |      4 |       0 |          3
     P3 |       2 |     8 |        3 |     12 |       2 |         10

Gantt Chart:
P1 P2 P2 P2 P1 P3 P3 P3 P3 P3 P3 P3 P3 P1 P1 P1

Average Waiting Time: 4.33333
Average Turnaround Time: 9.66667

CPU Scheduling Algorithms:
```

```
1. FCFS (First-Come, First-Served)
2. Round Robin
3. SJF (Non-Preemptive)
4. SJF (Preemptive)
5. Priority (Non-Preemptive)
6. Priority (Preemptive)
7. Exit
Enter your choice: 7
```