C. all process

D. init process

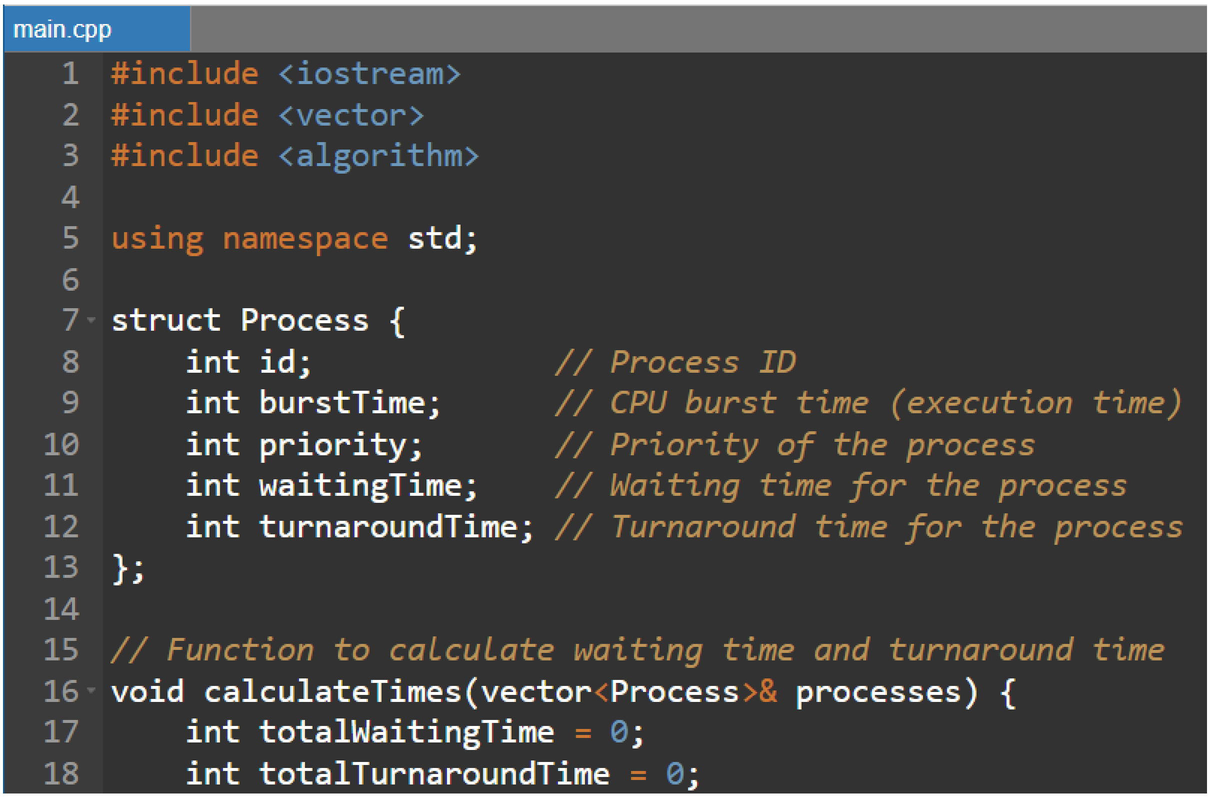
4. Differentiate between pre-emptive and non pre-emptive scheduling?

Sol. The basic difference between preemptive and non-preemptive scheduling is that in preemptive scheduling the CPU is allocated to the processes for the limited time. While in Non-preemptive scheduling, the CPU is allocated to the process till it terminates or switches to waiting state.

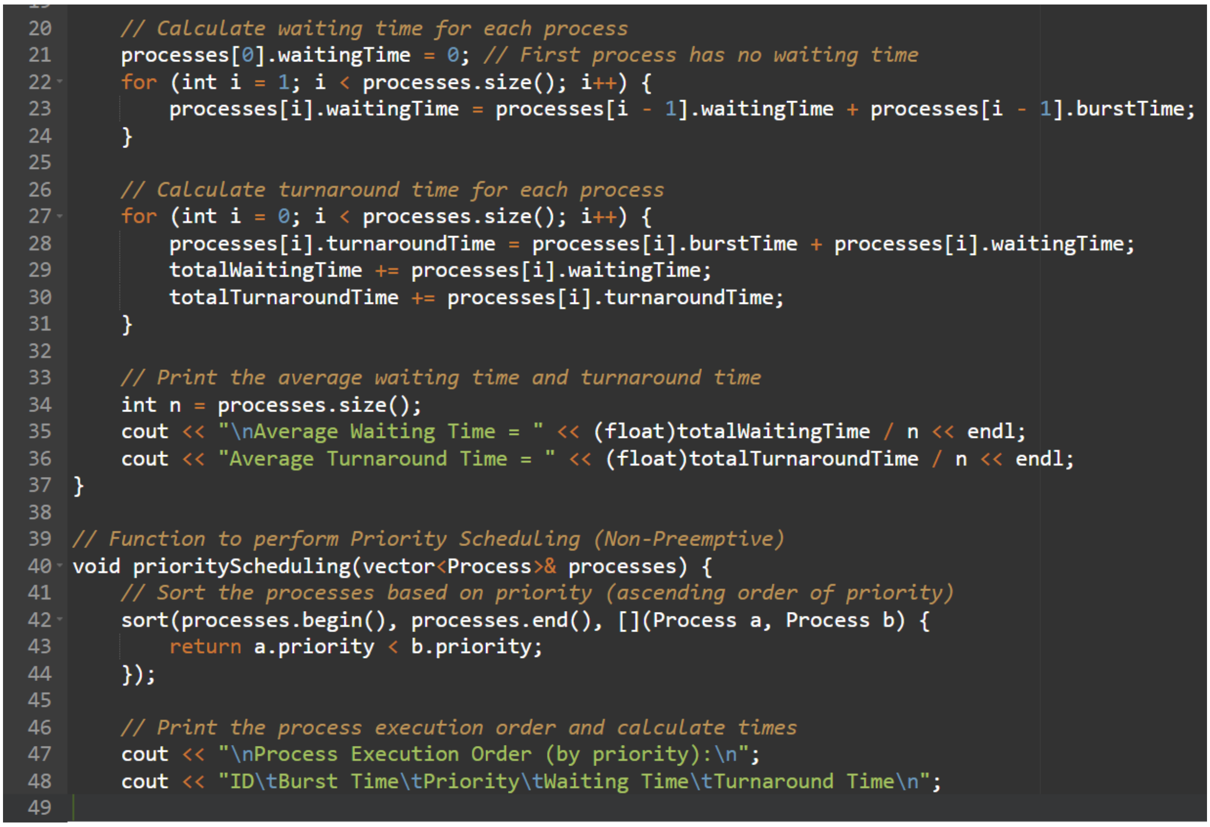
Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

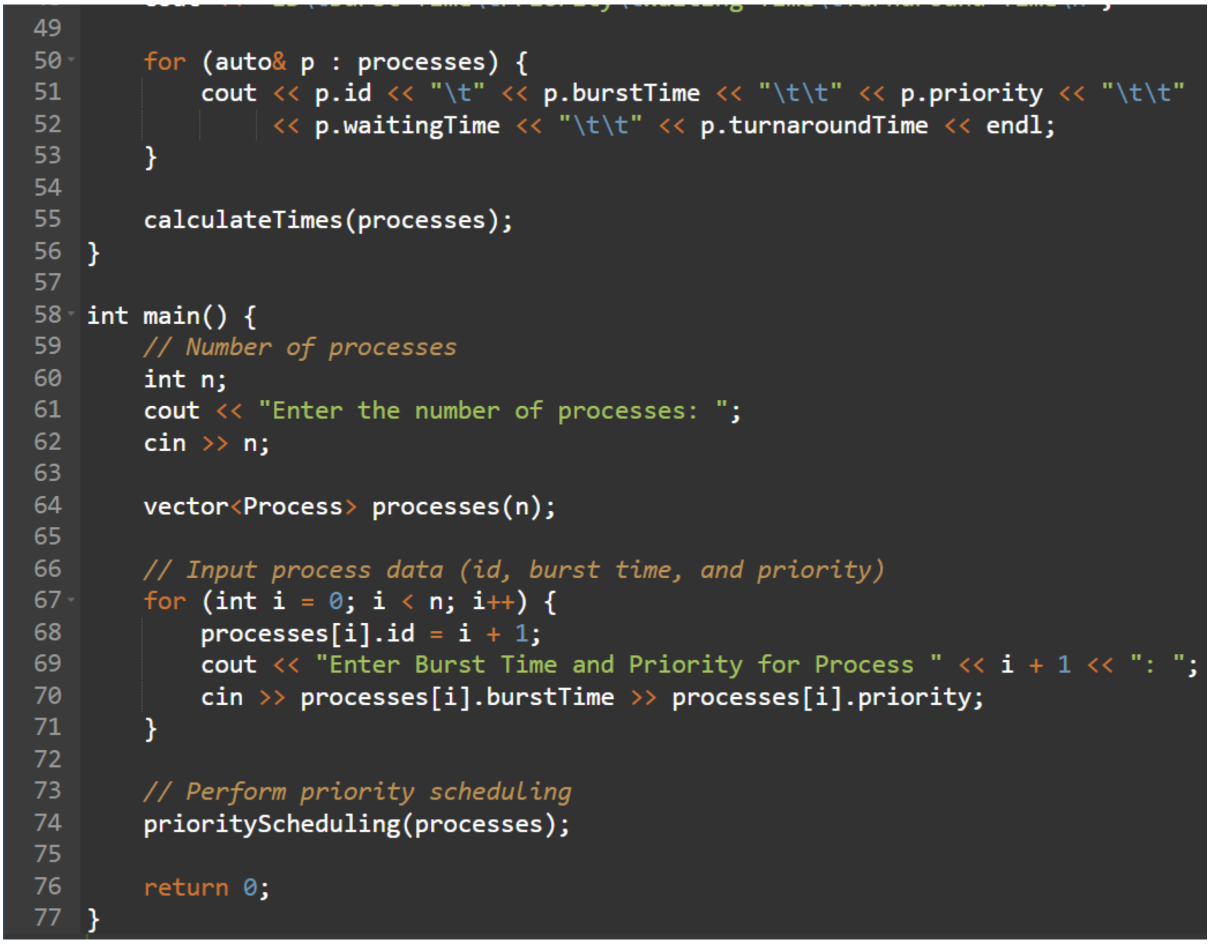
Priority Scheduling Non-preemptive



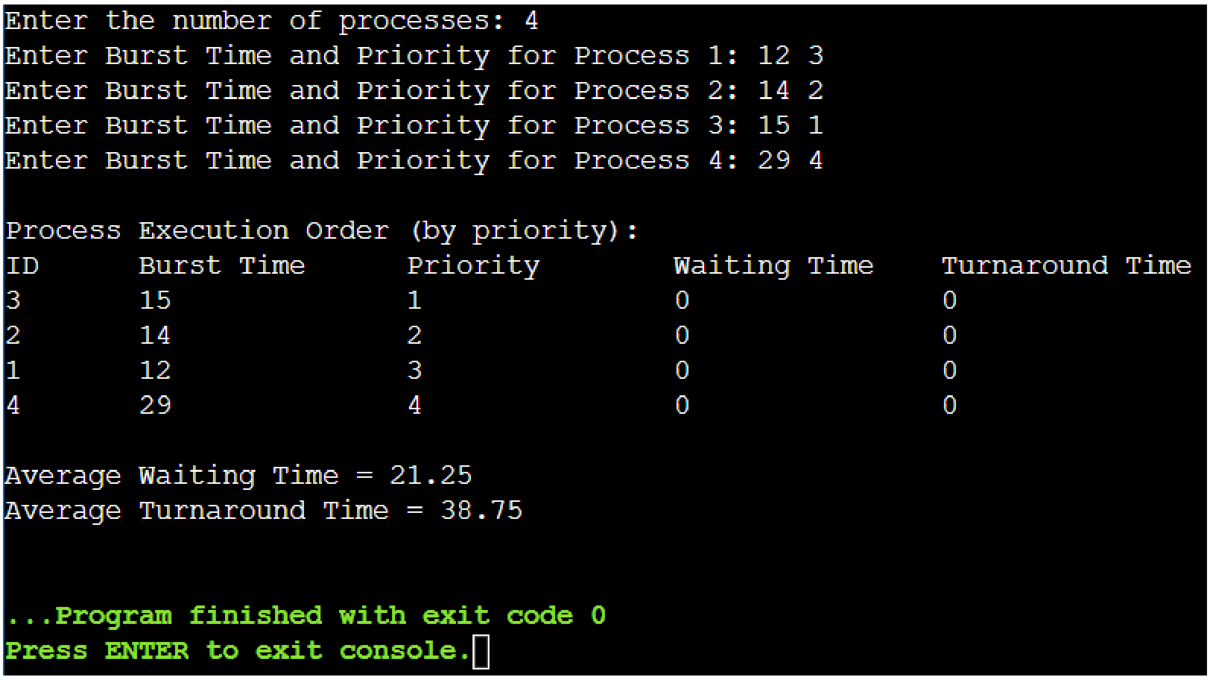
```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Process {
    int id;             // Process ID
    int burstTime;      // CPU burst time (execution time)
    int priority;       // Priority of the process
    int waitingTime;    // Waiting time for the process
    int turnaroundTime; // Turnaround time for the process
};

// Function to calculate waiting time and turnaround time
void calculateTimes(vector<Process>& processes) {
    int totalWaitingTime = 0;
    int totalTurnaroundTime = 0;
```



```
    // Calculate waiting time for each process
    processes[0].waitingTime = 0; // First process has no waiting time
    for (int i = 1; i < processes.size(); i++) {
        processes[i].waitingTime = processes[i - 1].waitingTime + processes[i - 1].burstTime;
    }

    // Calculate turnaround time for each process
    for (int i = 0; i < processes.size(); i++) {
        processes[i].turnaroundTime = processes[i].burstTime + processes[i].waitingTime;
        totalWaitingTime += processes[i].waitingTime;
        totalTurnaroundTime += processes[i].turnaroundTime;
    }

    // Print the average waiting time and turnaround time
    int n = processes.size();
    cout << "\nAverage Waiting Time = " << (float)totalWaitingTime / n << endl;
    cout << "Average Turnaround Time = " << (float)totalTurnaroundTime / n << endl;
}

// Function to perform Priority Scheduling (Non-Preemptive)
void priorityScheduling(vector<Process>& processes) {
    // Sort the processes based on priority (ascending order of priority)
    sort(processes.begin(), processes.end(), [](Process a, Process b) {
        return a.priority < b.priority;
    });

    // Print the process execution order and calculate times
    cout << "\nProcess Execution Order (by priority):\n";
    cout << "ID\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n";

```



```cpp
    cout << "ID\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n";

    for (auto& p : processes) {
        cout << p.id << "\t" << p.burstTime << "\t\t" << p.priority << "\t\t"
             << p.waitingTime << "\t\t" << p.turnaroundTime << endl;
    }

    calculateTimes(processes);
}

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

    vector<Process> processes(n);

    // Input process data (id, burst time, and priority)
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        cout << "Enter Burst Time and Priority for Process " << i + 1 << ": ";
        cin >> processes[i].burstTime >> processes[i].priority;
    }

    // Perform priority scheduling
    priorityScheduling(processes);

    return 0;
}
```



```
Enter the number of processes: 4
Enter Burst Time and Priority for Process 1: 12 3
Enter Burst Time and Priority for Process 2: 14 2
Enter Burst Time and Priority for Process 3: 15 1
Enter Burst Time and Priority for Process 4: 29 4

Process Execution Order (by priority):
ID      Burst Time      Priority        Waiting Time    Turnaround Time
3       15              1               0               0
2       14              2               0               0
1       12              3               0               0
4       29              4               0               0

Average Waiting Time = 21.25
Average Turnaround Time = 38.75


...Program finished with exit code 0
Press ENTER to exit console.
```

Priority Scheduling Preemptive

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// Structure to represent a process
struct Process {
    int pid;           // Process ID
    int arrival_time;   // Arrival Time
    int burst_time;     // Burst Time (CPU Time)
    int priority;       // Priority (lower value means higher priority)
    int waiting_time;   // Waiting Time (calculated)
    int turnaround_time; // Turnaround Time (calculated)
};

// Comparator to sort processes by arrival time
bool compareArrival(Process a, Process b) {
    return a.arrival_time < b.arrival_time;
}

// Comparator to sort processes by priority and burst time for preemption
bool comparePriority(Process a, Process b) {
    if (a.priority == b.priority)
```

```
        return a.arrival_time < b.arrival_time;
    return a.priority < b.priority; // Higher priority comes first
}

// Function to calculate waiting and turnaround time
void calculateTimes(vector<Process>& processes) {
    int n = processes.size();
    int current_time = 0;
    int completed = 0;
    vector<bool> is_completed(n, false);
    int last_process_time = 0;

    while (completed < n) {
        int idx = -1;
        int min_priority = 9999;

        // Find the process with highest priority which has arrived and is not yet completed
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= current_time && !is_completed[i]) {
                if (processes[i].priority < min_priority) {
                    min_priority = processes[i].priority;
                    idx = i;
                }
            }
        }
        if (idx != -1) {
            // Execute the selected process for one unit of time (preemptive)
            processes[idx].burst_time--;
            current_time++;

            // If the process is finished
            if (processes[idx].burst_time == 0) {
                processes[idx].turnaround_time = current_time - processes[idx].arrival_time;
                processes[idx].waiting_time = processes[idx].turnaround_time - (processes[idx].burst_time + 1);
                is_completed[idx] = true;
                completed++;  // This is where the error was
            }
        } else {
            // No process is ready to execute, move time forward
            current_time++;
        }
    }
}
```