```c
#include <stdio.h>
#include <stdlib.h>

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

struct ExecSegment{
   int pid, start, end;
};

// ---------- Input & Sorting ----------
void inputProcesses(struct Process p[], int n){
   for (int i = 0; i < n; i++){
      printf("Enter PID, Arrival Time, Burst Time for process %d: ", i + 1);
      scanf("%d %d %d", &p[i].pid, &p[i].at, &p[i].bt);
      p[i].remaining = p[i].bt;
      p[i].ct = p[i].tat = p[i].wt = 0;
   }
}

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

// ---------- FCFS ----------
int fcfs(struct Process p[], int n, struct ExecSegment gantt[]){
   int time = 0, s = 0;
   for (int i = 0; i < n; i++){
      if (time < p[i].at)
         time = p[i].at; // CPU finishes a process and the next process hasn't arrived yet
      gantt[s].pid = p[i].pid;
      gantt[s].start = time;
      time += p[i].bt; // time advanced by bt
      gantt[s].end = time;
      p[i].ct = time;
      p[i].tat = p[i].ct - p[i].at;
      p[i].wt = p[i].tat - p[i].bt;
      s++;
   }
   return s;
}
```

```
// ---------- SJF (Non-Preemptive) ----------
int sjfNonPreemptive(struct Process p[], int n, struct ExecSegment gantt[]){
   int done[100] = {0}, time = 0, s = 0, completed = 0; // done : track which processes have
finished
   while (completed < n){
      int idx = -1, minBT = 1e9;
      for (int i = 0; i < n; i++)
         /* it searches for the process that meets 3 conditions:
         1. It has not yet been completed (!done[i]).
         2. It has already arrived (p[i].at <= time).
         3. Among all such processes, it finds the one with the minimum burst time (p[i].bt <
minBT).*/
         if (!done[i] && p[i].at <= time && p[i].bt < minBT)
            minBT = p[i].bt, idx = i;
      if (idx == -1)
      {
         time++;
         continue;
      } // no process has arrived yet (idx == -1), the time simply increments.

      gantt[s].pid = p[idx].pid;
      gantt[s].start = time;
      time += p[idx].bt;
      gantt[s].end = time;
      p[idx].ct = time;
      p[idx].tat = time - p[idx].at;
      p[idx].wt = p[idx].tat - p[idx].bt;
      done[idx] = 1;
      completed++;
      s++;
   }
   return s;
}

// ---------- SJF (Preemptive) ----------
int sjfPreemptive(struct Process p[], int n, struct ExecSegment gantt[], int execOrder[], int
*execCount){
   int time = 0, s = 0, completed = 0, lastPid = -1;
   int rem[n]; // remaining burst time for each process
   for (int i = 0; i < n; i++)
      rem[i] = p[i].bt;
   *execCount = 0;

   while (completed < n)
   {
      int idx = -1, minBT = 1e9;
      for (int i = 0; i < n; i++)
         if (p[i].at <= time && rem[i] > 0 && rem[i] < minBT)
            minBT = rem[i], idx = i;
      if (idx == -1)
      {
```

```
            time++;
            continue;
        }

        /*The last variable acts as a memory of previously executing process. It shows Gantt
chart only starts new block when the CPU is assigned to a difft process, making final output
clean.*/
        if (lastPid != p[idx].pid)
        {
            gantt[s].pid = p[idx].pid;
            gantt[s].start = time;
            s++;
            lastPid = p[idx].pid;
        }

        time++;
        rem[idx]--;
        execOrder[(*execCount)++] = p[idx].pid;
        gantt[s - 1].end = time;

        if (rem[idx] == 0)
        {
            p[idx].ct = time;
            p[idx].tat = time - p[idx].at;
            p[idx].wt = p[idx].tat - p[idx].bt;
            completed++;
        }
    }
    return s;
}

// ---------- Round Robin ----------
int roundRobin(struct Process p[], int n, int tq, struct ExecSegment gantt[], int execOrder[],
int *execCount){
    int queue[100], front = 0, rear = 0, visited[100] = {0}; // visited array ensures processes are
added to the queue only once upon arrival.
    int time = p[0].at, completed = 0, s = 0;
    *execCount = 0;
    queue[rear++] = 0;
    visited[0] = 1;

    while (completed < n){
        if (front == rear){ // queue empty
            int nextArrival = -1;
            for (int i = 0; i < n; i++){
                if (!visited[i] && (nextArrival == -1 || p[i].at < nextArrival)) nextArrival = p[i].at;
            }
            if (nextArrival == -1) break;
            time = nextArrival;
            for (int i = 0; i < n; i++)
                if (!visited[i] && p[i].at <= time){
```

```
            queue[rear++] = i;
            visited[i] = 1;
          }
        continue;
      }
      int idx = queue[front++];
      int exec = (p[idx].remaining > tq) ? tq : p[idx].remaining;

      gantt[s].pid = p[idx].pid;
      gantt[s].start = time;
      time += exec;
      gantt[s].end = time;
      s++;
      execOrder[(*execCount)++] = p[idx].pid;
      p[idx].remaining -= exec;

      for (int i = 0; i < n; i++)
        if (!visited[i] && p[i].at <= time){
          queue[rear++] = i;
          visited[i] = 1;
        }
      if (p[idx].remaining > 0)
        queue[rear++] = idx;
      else{
        p[idx].ct = time;
        p[idx].tat = time - p[idx].at;
        p[idx].wt = p[idx].tat - p[idx].bt;
        completed++;
      }
    }
    return s;
}

// ---------- Priority Non-Preemptive ----------
int priorityNonPreemptive(struct Process p[], int n, struct ExecSegment gantt[]){
    int priority[100], done[100] = {0}, time = 0, s = 0, completed = 0;
    for (int i = 0; i < n; i++){
      printf("Priority for P%d: ", p[i].pid);
      scanf("%d", &priority[i]);
    }
    while (completed < n){
      int idx = -1, minPr = 1e9;
      for (int i = 0; i < n; i++)
        if (!done[i] && p[i].at <= time && priority[i] < minPr)
          minPr = priority[i], idx = i;
      if (idx == -1){
        time++;
        continue;
      }
      gantt[s].pid = p[idx].pid;
      gantt[s].start = time;
```

```
        time += p[idx].bt;
        gantt[s].end = time;
        p[idx].ct = time;
        p[idx].tat = time - p[idx].at;
        p[idx].wt = p[idx].tat - p[idx].bt;
        done[idx] = 1;
        completed++;
        s++;
    }
    return s;
}

// ---------- Priority Preemptive ----------
int priorityPreemptive(struct Process p[], int n, struct ExecSegment gantt[], int execOrder[],
int *execCount){
    int rem[n], priority[n];
    *execCount = 0;
    for (int i = 0; i < n; i++){
        rem[i] = p[i].bt;
        printf("Priority for P%d: ", p[i].pid);
        scanf("%d", &priority[i]);
    }
    int time = 0, s = 0, completed = 0, last = -1;
    while (completed < n){
        int idx = -1, minPr = 1e9;
        for (int i = 0; i < n; i++)
            if (rem[i] > 0 && p[i].at <= time && priority[i] < minPr)
                minPr = priority[i], idx = i;
        if (idx == -1){
            time++;
            continue;
        }
        if (last != p[idx].pid){
            gantt[s].pid = p[idx].pid;
            gantt[s].start = time;
            s++;
            last = p[idx].pid;
        }
        time++;
        rem[idx]--;
        execOrder[(*execCount)++] = p[idx].pid;
        gantt[s - 1].end = time;
        if (rem[idx] == 0){
            p[idx].ct = time;
            p[idx].tat = time - p[idx].at;
            p[idx].wt = p[idx].tat - p[idx].bt;
            completed++;
        }
    }
    return s;
}
```

```
// ---------- Printing Functions ----------
void printGantt(struct ExecSegment gantt[], int segCount){
   printf("\nGantt Chart:\n ");
   for (int i = 0; i < segCount; i++) printf("------"); printf("-\n|");
   for (int i = 0; i < segCount; i++) printf(" P%-2d |", gantt[i].pid); printf("\n ");
   for (int i = 0; i < segCount; i++) printf("------"); printf("-\n");
   for (int i = 0; i < segCount; i++) printf("%-6d", gantt[i].start);
   if (segCount > 0) printf("%d\n", gantt[segCount - 1].end);
}

void printReadyQueue(int execOrder[], int execCount){
   printf("\nReady Queue: ");
   for (int i = 0; i < execCount; i++)
      printf("P%d%s", execOrder[i], (i != execCount - 1) ? " | " : "");
   printf("\n");
}

void printTable(struct Process p[], int n){
   float totalWT = 0, totalTAT = 0;
   printf("\nPID\tAT\tBT\tCT\tTAT\tWT\n");
   for (int i = 0; i < n; i++){
      printf("%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].at, p[i].bt, p[i].ct, p[i].tat, p[i].wt);
      totalWT += p[i].wt;
      totalTAT += p[i].tat;
   }
   printf("\nAverage TAT: %.2f\nAverage WT: %.2f\n", totalTAT / n, totalWT / n);
}

// ---------- Main Function ----------
int main(){
   int n, algo, tq = 0;

   // Take input for processes once
   printf("Enter number of processes: ");
   scanf("%d", &n);
   struct Process p[n];
   inputProcesses(p, n);
   sortByArrival(p, n);

   // Infinite loop for algorithm menu
   while (1)
   {
      // Create copies so original data remains unchanged for each algorithm
      struct Process tempP[n];
      for (int i = 0; i < n; i++)
         tempP[i] = p[i];

      struct ExecSegment gantt[1000];
      int execOrder[1000], execCount = 0, segCount = 0;
```

```
// Algorithm selection menu
printf("\nSelect Scheduling Algorithm:\n");
printf("1. FCFS\n2. Round Robin\n3. SJF\n4. Priority\n5. Exit\nChoice: ");
scanf("%d", &algo);

switch (algo){
case 1:
   segCount = fcfs(tempP, n, gantt);
   for (int i = 0; i < segCount; i++)
      execOrder[i] = gantt[i].pid;
   execCount = segCount;
   break;

case 2:
   printf("Enter Time Quantum: ");
   scanf("%d", &tq);
   segCount = roundRobin(tempP, n, tq, gantt, execOrder, &execCount);
   break;

case 3:{
   int type;
   printf("Select SJF Type:\n1. Non-Preemptive\n2. Preemptive\nChoice: ");
   scanf("%d", &type);
   segCount = (type == 1)
              ? sjfNonPreemptive(tempP, n, gantt)
              : sjfPreemptive(tempP, n, gantt, execOrder, &execCount);
   if (type == 1){
      for (int i = 0; i < segCount; i++)
         execOrder[i] = gantt[i].pid;
      execCount = segCount;
   }
   break;
}

case 4:{
   int type;
   printf("Select Priority Type:\n1. Non-Preemptive\n2. Preemptive\nChoice: ");
   scanf("%d", &type);
   segCount = (type == 1)
              ? priorityNonPreemptive(tempP, n, gantt)
              : priorityPreemptive(tempP, n, gantt, execOrder, &execCount);
   if (type == 1){
      for (int i = 0; i < segCount; i++)
         execOrder[i] = gantt[i].pid;
      execCount = segCount;
   }
   break;
}

case 5:
   printf("Exiting program...\n");
```

```
            return 0;

        default:
            printf("Invalid choice. Try again.\n");
            continue;
        }

        // Display results
        printGantt(gantt, segCount);
        printReadyQueue(execOrder, execCount);
        printTable(tempP, n);
    }
}
```

Output:

```
manasvi@manasvi:/mnt/c/Users/bhute/Desktop/oslab/Assignment3$ gcc 3.c -o first
manasvi@manasvi:/mnt/c/Users/bhute/Desktop/oslab/Assignment3$ ./first
Enter number of processes: 5
Enter PID, Arrival Time, Burst Time for process 1: 1 2 3
Enter PID, Arrival Time, Burst Time for process 2: 2 3 4
Enter PID, Arrival Time, Burst Time for process 3: 3 4 5
Enter PID, Arrival Time, Burst Time for process 4: 4 5 6
Enter PID, Arrival Time, Burst Time for process 5: 5 6 7

Select Scheduling Algorithm:
1. FCFS
2. Round Robin
3. SJF
4. Priority
5. Exit
Choice: 1

Gantt Chart:
 -------------------------------
| P1  | P2  | P3  | P4  | P5  |
 -------------------------------
2     5     9     14    20    27

Ready Queue: P1 | P2 | P3 | P4 | P5

PID     AT      BT      CT      TAT     WT
1       2       3       5       3       0
2       3       4       9       6       2
3       4       5       14      10      5
4       5       6       20      15      9
5       6       7       27      21      14

Average TAT: 11.00
Average WT: 6.00
```

```
Select Scheduling Algorithm:
1. FCFS
2. Round Robin
3. SJF
4. Priority
5. Exit
Choice: 2
Enter Time Quantum: 3

Gantt Chart:
 ------------------------------------------------------------
| P1  | P2  | P3  | P4  | P5  | P2  | P3  | P4  | P5  | P5  |
 ------------------------------------------------------------
2     5     8     11    14    17    18    20    23    26    27

Ready Queue: P1 | P2 | P3 | P4 | P5 | P2 | P3 | P4 | P5 | P5

PID     AT      BT      CT      TAT     WT
1       2       3       5       3       0
2       3       4       18      15      11
3       4       5       20      16      11
4       5       6       23      18      12
5       6       7       27      21      14

Average TAT: 14.60
Average WT: 9.60
```

```
Select Scheduling Algorithm:
1. FCFS
2. Round Robin
3. SJF
4. Priority
5. Exit
Choice: 3
Select SJF Type:
1. Non-Preemptive
2. Preemptive
Choice: 1

Gantt Chart:
 ------------------------------
| P1  | P2  | P3  | P4  | P5  |
 ------------------------------
2     5     9     14    20    27

Ready Queue: P1 | P2 | P3 | P4 | P5

PID     AT      BT      CT      TAT     WT
1       2       3       5       3       0
2       3       4       9       6       2
3       4       5       14      10      5
4       5       6       20      15      9
5       6       7       27      21      14

Average TAT: 11.00
Average WT: 6.00
```

```
Select Scheduling Algorithm:
1. FCFS
2. Round Robin
3. SJF
4. Priority
5. Exit
Choice: 3
Select SJF Type:
1. Non-Preemptive
2. Preemptive
Choice: 2

Gantt Chart:
 ------------------------------
| P1  | P2  | P3  | P4  | P5  |
 ------------------------------
2     5     9     14    20    27

Ready Queue: P1 | P1 | P1 | P2 | P2 | P2 | P2 | P3 | P3 | P3
 | P3 | P3 | P4 | P4 | P4 | P4 | P4 | P4 | P5 | P5 | P5 | P5
 | P5 | P5 | P5

PID     AT      BT      CT      TAT     WT
1       2       3       5       3       0
2       3       4       9       6       2
3       4       5       14      10      5
4       5       6       20      15      9
5       6       7       27      21      14

Average TAT: 11.00
Average WT: 6.00
```

```
Select Scheduling Algorithm:
1. FCFS
2. Round Robin
3. SJF
4. Priority
5. Exit
Choice: 4
Select Priority Type:
1. Non-Preemptive
2. Preemptive
Choice: 1
Priority for P1: 4
Priority for P2: 3
Priority for P3: 5
Priority for P4: 2
Priority for P5: 3

Gantt Chart:
 ------------------------------
| P1  | P4  | P2  | P5  | P3  |
 ------------------------------
2     5     11    15    22    27

Ready Queue: P1 | P4 | P2 | P5 | P3

PID     AT      BT      CT      TAT     WT
1       2       3       5       3       0
2       3       4       15      12      8
3       4       5       27      23      18
4       5       6       11      6       0
5       6       7       22      16      9

Average TAT: 12.00
Average WT: 7.00
```

```
3. SJF
4. Priority
5. Exit
Choice: 4
Select Priority Type:
1. Non-Preemptive
2. Preemptive
Choice: 2
Priority for P1: 4
Priority for P2: 3
Priority for P3: 5
Priority for P4: 2
Priority for P5: 3

Gantt Chart:
 ----------------------------------------
| P1  | P2  | P4  | P2  | P5  | P1  | P3  |
 ----------------------------------------
2     3     5     11    13    20    22    27

Ready Queue: P1 | P2 | P2 | P4 | P4 | P4 | P4 | P4 | P4 | P2 | P2 | P5 | P5
 | P5 | P5 | P5 | P5 | P5 | P1 | P1 | P3 | P3 | P3 | P3 | P3

PID     AT      BT      CT      TAT     WT
1       2       3       22      20      17
2       3       4       13      10      6
3       4       5       27      23      18
4       5       6       11      6       0
5       6       7       20      14      7

Average TAT: 14.60
Average WT: 9.60

Select Scheduling Algorithm:
1. FCFS
2. Round Robin
3. SJF
4. Priority
5. Exit
Choice: 5
Exiting program...
```