```
#include<stdio.h>
#define MAX_SIZE 101 // Maximum Size Limit

int N; // Number Of Processess

// A Structure Consists of :-
// id  :-  Name Of The Process
// at  :-  Arrival Time
// bt  :-  Burest Time
// p   :-  Priority
// ft  :-  Finished Time
// st  :-  Starting Time
// wt  :-  Waiting Time
// trt :-  Total TurnAround Time

struct data {
   int num;
   char id[5];
   int at;
   int bt;
   int p;
   int rt;
   int ft;
   int st;
   int wt;
   int trt;
};

// Variables Which Are Usefull For The PriorityQueue And Queue
// pqf :- Priority Queue Front
// pqr :- Priority Queue Rear
// rqf :- Queue Front
// rqr :- Queue Rear

int pqf = -1, pqr = -1;
int rqf = -1, rqr = -1;

// Declaration of Array of Structures

struct data* priorityQueue[MAX_SIZE];
struct data* queue[MAX_SIZE];

// Sorting the Process
// Sort According to the arrival time if any two arrival time are equal then
// Sort According to there Process Id

void sort(struct data p[]) {
  int i, j;
  struct data tmp;

  for (i = 0; i < N; i++) {

    for (j = i; j >= 1; j--) {

      if (p[j].at < p[j - 1].at) {

        tmp = p[j - 1];
        p[j - 1] = p[j];
        p[j] = tmp;
      }
      else if (p[j].at == p[j - 1].at) {
```

```
                if (p[j].num < p[j].num) {

                    tmp = p[j - 1];
                    p[j - 1] = p[j];
                    p[j] = tmp;
                }
            }
        }
    }
}

// pqEmpty is a function which tell Priority Queue is empty or not

int pqEmpty() {

    return (pqf == -1 && pqr == -1); // If pqf and pqr both are equal to -1 then Priority Queue is Empty else not empty
}

// pqTop is a function which returns the top of the Priority Queue

struct data* pqTop() {

    return priorityQueue[pqf];
}

// check is a function which put data into the desire position

void check(struct data *x) {

    int i, j;
    for (i = 0; i <= pqr; i++) {

        if (x->p < priorityQueue[i]->p) {

            for (j = pqr + 1; j > i; j--) {

                priorityQueue[j] = priorityQueue[j - 1];
            }

            priorityQueue[i] = x;
            return;
        }
    }
    priorityQueue[i] = x;
}

// pqPush is a function which push data into the Priority Queue

void pqPush(struct data* x) {

    if (pqf == -1 && pqr == -1) { // If Priority Queue is empty then

        pqf++; // Increment the both the values to 1
        pqr++;
        priorityQueue[pqr] = x;
        return;
    }
    else {

        check(x);
    }

    pqr++;
}

// pqPop is a function which pop out the data from Priority Queue

void pqPop() {

    int i;

    if (pqf == -1 && pqr == -1) {

        return;
    }
    for (i = 0; i < pqr; i++) {

        priorityQueue[i] = priorityQueue[i + 1];
    }

    pqr--;

    if (pqr == -1)
        pqf = -1;
}

// rpEmpty is a function which tells Queue is Empty or Not

int rqEmpty() {

    return (rqf == -1 && rqr == -1);
```

```
}


// rqFront is a function which returns the top of elemeent from Queue


struct data* rqFront() {
  return queue[rqf];
}


// rqPush ia a function which push elements into the Queue


void rqPush(struct data* x) {


  if (rqf == -1 && rqr == -1) {


    rqf++;
    rqr++;
    queue[rqr] = x;
    return;
  }
  else {


    rqr++;
  }


  queue[rqr] = x;
}


// rqPop is a function which pop out the element from Queue


void rqPop() {
    if (rqf == -1 && rqr == -1) {

        return;
    }

    for (int i = 0; i <= rqr; i++) {

        queue[i] = queue[i + 1];
    }

    rqr--;
    if (rqr == -1)
        rqf = -1;
}

// It calculate the average waiting time and average turnaround time

void calculation(struct data p[], int g[], int n) {

    int i, j;
    float avgWt = 0, avgTrt = 0;

    for (i = 0; i < N; i++) {

        for (j = n - 1; j >= 0; j--) {

            if (g[j] == p[i].num) {

                p[i].ft = j + 1;
                break;
            }
        }
    }

    for (i = 0; i < N; i++) {

        for (j = 0; j < n; j++) {

            if (g[j] == p[i].num) {

                p[i].st = j;
                break;
            }
        }
    }

    for (i = 0; i < N; i++) {

        p[i].wt = p[i].ft - p[i].at - p[i].bt;
        p[i].trt = p[i].wt + p[i].bt;
        avgWt += p[i].wt;
        avgTrt += p[i].trt;
    }

    printf("Id \t ArrivalTime \t BurestTime \t WaitingTime \t TurnAroundTime \n");

    for (i = 0; i < N; i++) {

        printf("%d \t    %d \t\t    %d \t\t    %d \t\t    %d \n", p[i].num, p[i].at, p[i].bt, p[i].wt, p[i].trt);
    }

    avgWt /= N;
```

```c
    avgTrt /= N;

    printf("\n\n");
    printf("Average Waiting Time And Average Turn Around Time \n\n");

    printf("%f %f", avgWt, avgTrt);

    printf("\n");

}

// Implementatio of Multi Level Queue

void MLQ(struct data p[]) {

    int tt = 0; // Sum of all burst
    tt += p[0].at + p[0].bt;

    for (int i = 1; i < N; i++) {

        if (tt < p[i].at)
            tt = p[i].at;

        tt += p[i].bt;
    }

    int ghart[tt]; // Ghant Chart
    int cpu_state = 0; // Status of the CPU

    for (int i = 0; i < tt; i++)
        ghart[i] = -1;
```

```
struct data* current;
int pq_process = 0; // Status of the Priority Queue
int rq_process = 0; // Status of the Queue
int q = 2; // Time Quantum for Round Robin

for (int i = 0; i < tt; i++) {

    for (int j = 0; j < N; j++) {

        if (i == p[j].at) {
            pqPush(&p[j]); // Pushing all elements which has arrived
        }
    }

    if (cpu_state == 0) { // Checking Status of CPU

        if (!pqEmpty()) {

            current = pqTop();
            pqPop();
            pq_process = 1;
            cpu_state = 1;
        }
        else if (!rqEmpty()) {

            current = rqFront();
            rqPop();
            rq_process = 1;
            q = 2;
            cpu_state = 1;
    }
}
else if (cpu_state == 1) {

    if (pq_process == 1 && !pqEmpty()) {

        if (pqTop()->p < current->p) {

            rqPush(current);
            current = pqTop();
            pqPop();
        }
    }
    else if (rq_process == 1 && !pqEmpty()) {

        rqPush(current);
        current = pqTop();
        pqPop();
        rq_process = 0;
        pq_process = 1;
    }
}

if (cpu_state == 1) {

    if (pq_process == 1) {

        current->rt--;
        ghart[i] = current->num;

        if (current->rt == 0) {
                cpu_state = 0;
                pq_process = 0;
            }
        }
        else if (rq_process == 1) {

            current->rt--;
            q--;
            ghart[i] = current->num;

            if (current->rt == 0) {

                cpu_state = 0;
                rq_process = 0;
            }
            else if (q == 0) {

                cpu_state = 0;
                rq_process = 1;
                rqPush(current);
            }
        }
    }
}

// Printing Ghart Chart

printf("\n\n");

for (int i = 0; i < tt; i++) {
        printf("%d ", ghart[i]);
    }

    printf("\n\n");

    calculation(p, ghart, tt);

}



int main() {

    printf("Enter Number of process \n");
    scanf("%d", &N);

    struct data p[N];

    printf("Enter Process Id, Arrival Time, Burest Time, Priority \n");
    for (int i = 0; i < N; i++) {
        scanf("%d%d%d%d", &p[i].num, &p[i].at, &p[i].bt, &p[i].p);
        p[i].rt = p[i].bt;
    }

    sort(p);   be

    MLQ(p);

    return 0;
}
```