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

struct proc_struct {
    char proc_name[20];
    int arr_time, bur_time, comp_time, rem;
}temp_Struct;


void faculty_Queue(int no_of_process) {

    int count, arr_time, bur_time, quan_time;
    struct proc_struct faculty_proc[no_of_process];

    for(count = 0; count < no_of_process; count++) {
        printf("Enter the details of Process[%d]", count+1);
        puts("");
        printf("Process Name : ");
        scanf("%s", faculty_proc[count].proc_name);

        printf("Arrival Time : ");
        scanf("%d", &faculty_proc[count].arr_time);

        printf("Burst Time : ");
        scanf("%d", &faculty_proc[count].bur_time);
        puts("");
    }
    printf("Now, enter the quantum time for FACULTY queue : ");
    scanf("%d", &quan_time);


    // sorting the processes by their ARRIVAL time.
    // if the ARRIVAL time is same then scheduling is based on FCFS.
    for(count = 0; count < no_of_process; count++) {
        for(int x = count +1; x < count; x++){
            if(faculty_proc[count].arr_time > faculty_proc[x].arr_time) {
                temp_Struct = faculty_proc[count];
                faculty_proc[count] = faculty_proc[x];
                faculty_proc[x] = temp_Struct;
            }
        }
    }

    // initialy all the burst time is rem and completion of process is zero.
    for(count = 0; count < no_of_process; count++) {
        faculty_proc[count].rem = faculty_proc[count].bur_time;
        faculty_proc[count].comp_time = 0;
    }

    int total_time, queue, round_robin[20];
    total_time = 0;
    queue = 0;
    round_robin[queue] = 0;


    int flag, x, n, z, waiting_time = 0;
    do {
        for(count = 0; count < no_of_process; count++){
            if(total_time >= faculty_proc[count].arr_time){
                z = 0;
                for(x = 0; x <= queue; x++) {
                    if(round_robin[x] == count) {
                        z++;
                    }
                }
                if(z == 0) {
                    queue++;
                    round_robin[queue] == count;
                }
            }
        }

        if(queue == 0) {
            n = 0;
        }
        if(faculty_proc[n].rem == 0) {
            n++ ;
        }
        if(n > queue) {
            n = (n - 1) % queue;
        }
        if(n <= queue) {
            if(faculty_proc[n].rem > 0) {
                if(faculty_proc[n].rem < quan_time){
                    total_time += faculty_proc[n].rem;
                    faculty_proc[n].rem = 0;
                }else {
                    total_time += quan_time;
                    faculty_proc[n].rem -= quan_time;
                }
                faculty_proc[n].comp_time = total_time;
            }
            n++;
        }
        flag = 0;

        for(count = 0; count < no_of_process; count++) {
            if(faculty_proc[count].rem > 0) {
                flag++;
                    }
                }
            }while(flag != 0);


            puts("\n\t\t\t********************************************");
            puts("\t\t\t*****   ROUND ROBIN ALGORITHM OUTPUT   *****");
            puts("\t\t\t********************************************\n");
            printf("\n|\tProcess Name\t  |\tArrival Time\t  |\tBurst Time\t |\tCompletion Time  \t|\n");

            for(count = 0; count < no_of_process; count++){
                waiting_time = faculty_proc[count].comp_time - faculty_proc[count].bur_time - faculty_proc[count].arr_time;

                printf("\n|\t  %s\t    |\t  %d\t   |\t  %d\t   |\t  %d\t   |\n", faculty_proc[count].proc_name, faculty_proc[count].arr_time, faculty_proc[count].bur_time, faculty_proc[count].comp_time);
            }

        }


        void student_Queue(int no_of_process) {

            int count, arr_time, bur_time, quan_time;
            struct proc_struct student_proc[no_of_process];

            for(count = 0; count < no_of_process; count++) {
                printf("Enter the details of Process[%d]", count+1);
                puts("");
                printf("Process Name : ");
                scanf("%s", student_proc[count].proc_name);

                printf("Arrival Time : ");
                scanf("%d", &student_proc[count].arr_time);

                printf("Burst Time : ");
                scanf("%d", &student_proc[count].bur_time);
            }
            printf("Now, enter the quantum time for STUDENT queue : ");
            scanf("%d", &quan_time);


            // sorting the processes by their ARRIVAL time.
            // if the ARRIVAL time is same then scheduling is based on FCFS.
            for(count = 0; count < no_of_process; count++) {
                for(int x = count +1; x < count; x++){
                    if(student_proc[count].arr_time > student_proc[x].arr_time) {
                        temp_Struct = student_proc[count];
                        student_proc[count] = student_proc[x];
                        student_proc[x] = temp_Struct;
                    }
                }
            }

            // initialy all the burst time is rem and completion of process is zero.
            for(count = 0; count < no_of_process; count++) {
                student_proc[count].rem = student_proc[count].bur_time;
                student_proc[count].comp_time = 0;
            }

            int total_time, queue, round_robin[20];
            total_time = 0;
            queue = 0;
            round_robin[queue] = 0;
        }


        int main(int argc, char const *argv[]) {
            int select_queue, no_of_process;

            puts("Please choose a queue to post your query : ");
            puts("1. FACULTY queue.");
            puts("2. STUDENT queue.");
            printf("> ");
            scanf("%d", &select_queue);

            switch(select_queue) {
                case 1 :
                        printf("Enter number of process for FACULTY queue : ");
                        scanf("%d", &no_of_process);

                        faculty_Queue(no_of_process);

                        break;

                case 2 :
                        printf("Enter number of process for STUDENT queue : ");
                        scanf("%d", &no_of_process);

                        student_Queue(no_of_process);

                        break;

                default :
                        printf("Please selet the correct option by running the program again.");
            }
```

```

            return 0;
        }
```