B. Siva Shirish -192324016

8. Construct a C program to simulate Round Robin scheduling algorithm withC.

Aim:

To design a C program that simulates the Round Robin scheduling algorithm, where processes are executed in a cyclic order with a fixed time quantum.

Algorithm:

1. Start the program.
2. Input the number of processes, their burst times, and the time quantum.
3. Initialize variables for tracking the remaining burst times, completion status, and time progression.
4. In a cyclic manner:
   - For each process, if it has remaining burst time:
     - Execute the process for the time quantum or until completion, whichever is smaller.
     - Update the remaining burst time and track the time elapsed.
   - If a process is completed, record its completion time, turnaround time, and waiting time.
5. Repeat until all processes are completed.
6. Calculate average waiting time and turnaround time.
7. Display the results.
8. End the program.

Procedure:

1. Include necessary headers: <stdio.h>.
2. Use arrays to store process attributes such as burst times, remaining burst times, waiting times, and turnaround times.
3. Implement a loop to simulate the Round Robin scheduling, iterating through processes in a cyclic manner.

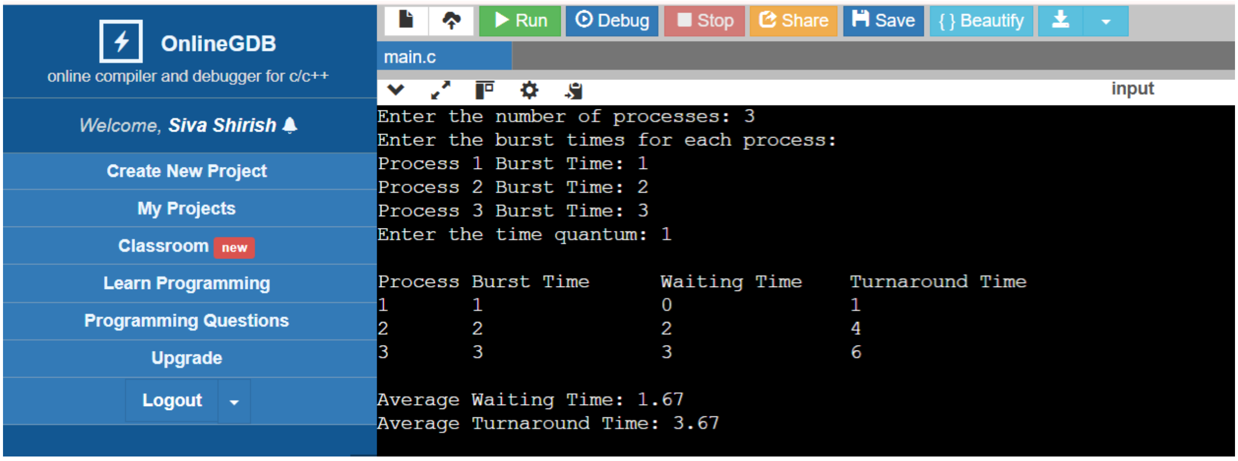
4. Track completion and compute metrics.

Code:

```
#include <stdio.h>

int main() {
   int n, i, time = 0, completed = 0, time_quantum;
   float avg_wait = 0, avg_turnaround = 0;


   printf("Enter the number of processes: ");
   scanf("%d", &n);


   int burst_time[n], remaining_time[n], waiting_time[n], turnaround_time[n],
arrival_time[n], is_completed[n];


   printf("Enter the burst times for each process:\n");
   for (i = 0; i < n; i++) {
      printf("Process %d Burst Time: ", i + 1);
      scanf("%d", &burst_time[i]);
      remaining_time[i] = burst_time[i];
      waiting_time[i] = 0;
      turnaround_time[i] = 0;
      is_completed[i] = 0;
   }


   printf("Enter the time quantum: ");
   scanf("%d", &time_quantum);


   while (completed < n) {
        for (i = 0; i < n; i++) {
            if (remaining_time[i] > 0) {
                if (remaining_time[i] <= time_quantum) {
                    time += remaining_time[i];
                    waiting_time[i] = time - burst_time[i];
                    turnaround_time[i] = time;
                    remaining_time[i] = 0;
                    completed++;
                } else {
                    time += time_quantum;
                    remaining_time[i] -= time_quantum;
                }
            }
        }
    }

    for (i = 0; i < n; i++) {
        avg_wait += waiting_time[i];
        avg_turnaround += turnaround_time[i];
    }

    avg_wait /= n;
    avg_turnaround /= n;

    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", i + 1, burst_time[i], waiting_time[i],
turnaround_time[i]);
```

```
    }

    printf("\nAverage Waiting Time: %.2f\n", avg_wait);
    printf("Average Turnaround Time: %.2f\n", avg_turnaround);

    return 0;
}
```

OUTPUT:



```
Enter the number of processes: 3
Enter the burst times for each process:
Process 1 Burst Time: 1
Process 2 Burst Time: 2
Process 3 Burst Time: 3
Enter the time quantum: 1

Process Burst Time      Waiting Time    Turnaround Time
1       1               0               1
2       2               2               4
3       3               3               6

Average Waiting Time: 1.67
Average Turnaround Time: 3.67
```