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

int main()
{//start main

printf("******* CPU SCHEDULER PROGRAM ********\n");
printf("******* @author: Kerri McMahon ********\n");


int running =0;
int a;

while(running !=9)

{//start while
printf("Pick an algorithm. press 1 for FCFS, 2 for SJF, 3 for RR -- Press 9 to
quit\n");
scanf("%d", &a);

void fcfs();
void sjf();
void rr();


if (a==9)
{
exit(0);
}

if(a==1)
{
fcfs();

}
 if (a == 2)
{
sjf();
}
if (a == 3)
{
rr();
}
} //end while


return 0;


}//end main


void fcfs()
{//start fcfs


char yorn;
char Y = 'Y';
char y = 'y';
char N = 'N';
char n = 'n';
```

```
printf("********** FIRST COME FIRST SERVE **********\n");
//Processes limit is 10
//Initializing, etc
float burst_time[10], waiting_time[10], turnaround_time[10];
        float average_waiting_time = 0.0, average_turnaround_time = 0.0;
        int count, j, total_process;
        printf("Enter The Number of Processes To Execute:\t");
        scanf("%d", &total_process);

        printf("Would you like randomized burst times? Y or N\n");
            scanf("%s", &yorn);

        if(yorn ==N || yorn ==n)
        {
             printf("\nEnter The Burst Time of Processes:\n\n");
            for(count = 0; count < total_process; count++)
            {
                printf("Process [%d]:", count + 1);
                scanf("%f", &burst_time[count]);
            }
        }

        if (yorn ==Y || yorn ==y)
        {
        srand(time(NULL));
            for(count =0;count < total_process;count++)
                {

                burst_time[count]=(rand() % 21);
                //We dont want a process with a burst time of 0 - doesnt make sense
                if(burst_time[count]==0)
                {
                burst_time[count]=1;
                }

                }
        }
        //Waiting time of first process in queue will always be zero
        waiting_time[0] = 0;

        for(count = 1; count < total_process; count++)
        {
                //This is so that we are able to calculate the average of the array
without NULL values
                //if all processes are not filled.
                waiting_time[count] = 0;
                //Nested for loop. This inner loop "follows" the outer loop to
                //enable summing of the previous processes' burst time.
                for(j = 0; j < count; j++)
                {
                        waiting_time[count] = waiting_time[count] + burst_time[j];
                }
        }
        printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time\n");
        for(count = 0; count < total_process; count++)
        {
                //Going Across and summing process information - Simply the
turnaround time calculation
                turnaround_time[count] = burst_time[count] + waiting_time[count];
                average_waiting_time = average_waiting_time + waiting_time[count]; //
                average_turnaround_time = average_turnaround_time +
turnaround_time[count];
```

```
                printf("\nProcess [%d]\t\t%.2f\t\t%.2f\t\t%.2f", count + 1,
burst_time[count], waiting_time[count], turnaround_time[count]);
        }
        printf("\n");

        //Printing the averages
        average_waiting_time = average_waiting_time / count;
        average_turnaround_time = average_turnaround_time / count;
        printf("\nAverage Waiting Time = %f", average_waiting_time);
        printf("\nAverage Turnaround Time = %f", average_turnaround_time);
        printf("\n");

}//end fcfs



void sjf()
{//start sjf

char yorn;
char Y = 'Y';
char y = 'y';
char N = 'N';
char n = 'n';
printf("********** SHORTEST JOB FIRST **********\n");
int temp, i, j, limit, sum = 0, position;
      float average_wait_time, average_turnaround_time; //Cant be ints
      int burst_time[10], process[10], waiting_time[10], turnaround_time[10];
      printf("\nEnter Total Number of Processes:\t");
      scanf("%d", &limit);


         printf("Would you like randomized burst times? Y or N\n");
            scanf("%s", &yorn);


        if(yorn ==N || yorn == n)
        {
        for(i = 0; i < limit; i++)
            {
            printf("Enter Burst Time For Process[%d]:\t", i + 1);
            scanf("%d", &burst_time[i]);
            process[i] = i + 1;
            }
        }
        if (yorn ==Y || yorn ==y)
        {
        srand(time(NULL));
            for(i =0;i < limit;i++)
                {

                burst_time[i]=(rand() % 21);
                     //We dont want a process with a burst time of 0 - doesnt make
sense
                if(burst_time[i]==0)
                {
                burst_time[i]=1;
                }
                process[i] = i +1;

                }
        }
      for(i = 0; i < limit; i++)
      {
            position = i;
            //This sorts the processes - GANTT
            for(j = i + 1; j < limit; j++)
            {
                  if(burst_time[j] < burst_time[position])
                  {
                        position = j;
                  }
            }
//temp array for sorting
            temp = burst_time[i];
            burst_time[i] = burst_time[position];
            burst_time[position] = temp;
         temp = process[i];
            process[i] = process[position];
            process[position] = temp;
      }
       //Waiting time will always be zero for first process
      waiting_time[0] = 0;
    //Starting at 1 because of this
      for(i = 1; i < limit; i++)
      {
            //So that there are no NULL values in array
            waiting_time[i] = 0;
            //Nested for loop lets us "follow", just as in FCFS
            for(j = 0; j < i; j++)
            {
                  waiting_time[i] = waiting_time[i] + burst_time[j];
            }
            sum = sum + waiting_time[i];
      }
      average_wait_time = (float)sum / limit;
      sum = 0;

    //Printing everything, blah blah
      printf("\nProcess ID\t\tBurst Time\t Waiting Time\t Turnaround Time\n");
      for(i = 0; i < limit; i++)
      {
            turnaround_time[i] = burst_time[i] + waiting_time[i];
            sum = sum + turnaround_time[i];
            printf("\nProcess[%d]\t\t%d\t\t %d\t\t %d\n", process[i], burst_time[i],
waiting_time[i], turnaround_time[i]);
      }
      average_turnaround_time = (float)sum / limit;
      printf("\nAverage Waiting Time:\t%f\n", average_wait_time);
      printf("\nAverage Turnaround Time:\t%f\n", average_turnaround_time);

}//end sjf



void rr()
{
printf("********** ROUND ROBIN **********\n");


char yorn1;//second prompt for random burst times
char yorn;//first prompt for arrival times
char Y = 'Y';
char y = 'y';
char N = 'N';
char n = 'n';
int i, limit, total = 0, x, counter = 0, time_quantum;
      int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10],
temp[10];
      float average_wait_time, average_turnaround_time;
      printf("\nEnter Total Number of Processes:\t");
      scanf("%d", &limit);
//consideration of arrival times.
  printf("\n Do we want to consider arrival times? Y for manual arrival times, N
initializes all to 0\n");
  scanf("%s", &yorn);
  x = limit;



      for(i = 0; i < limit; i++)
      {
            printf("\nEnter Details of Process[%d]\n", i + 1);
            if (yorn== Y || yorn ==y)
                {
                 printf("Arrival Time:\t");
                 scanf("%d", &arrival_time[i]);
                }
            if (yorn == N || yorn ==n)
                {
                arrival_time[i] =0;
                }
            printf("Burst Time:\t");
            scanf("%d", &burst_time[i]);

            temp[i] = burst_time[i];
           // printf(" %d ", burst_time[i]);
      }

      printf("\nEnter Time Quantum:\t");
      scanf("%d", &time_quantum);
      printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");
      for(total = 0, i = 0; x != 0;)
      {

            if(temp[i] <= time_quantum && temp[i] > 0)
            {
               // printf(" line 287 ");
                  total = total + temp[i];
                  temp[i] = 0;
                  counter = 1;
            }
            else if(temp[i] > 0)
            {

                  temp[i] = temp[i] - time_quantum;
                //printf("%d", temp[i]);
                  total = total + time_quantum;
             //printf("total equals: ");
               /// printf(" %d ", total);

            }
            if(temp[i] == 0 && counter == 1)
            {
               // printf("%d", total);
                  x--;
                  printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst_time[i],
total - arrival_time[i], total - arrival_time[i] - burst_time[i]);
                  wait_time = wait_time + total - arrival_time[i] - burst_time[i];
                  turnaround_time = turnaround_time + total - arrival_time[i];
                  counter = 0;
            }
            if(i == limit - 1)
            {
               // printf(" i==limit-1 ");
                  i = 0;
            }
            else if(arrival_time[i + 1] <= total)
            {
                //printf(" i++ ");
                  i++;
            }
            else
            {
                   //printf(" i=0 ");
                  i = 0;
                    // break;
            }
      }
      average_wait_time = wait_time * 1.0 / limit;
      average_turnaround_time = turnaround_time * 1.0 / limit;
      printf("\n\nAverage Waiting Time:\t%f", average_wait_time);
      printf("\nAvg Turnaround Time:\t%f\n", average_turnaround_time);



}//end rr
```