PROJECT REPORT

ON

Implementation of the FCFS scheduling algorithm

DATA STRUCTURES

Introduction

First Come First Serve (FCFS) is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival. It is the easiest and simplest CPU scheduling algorithm. In this type of algorithm, a process which requests the CPU first get the CPU allocation first. This is managed with a FIFO queue. The full form of FCFS is First Come First Serve.

As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue and, when the CPU becomes free, it should be assigned to the process at the beginning of the queue.

Characteristics of FCFS method

- It supports non-pre-emptive and pre-emptive scheduling algorithm.
- Jobs are always executed on a first-come, first-serve basis.
- It is easy to implement and use.
- This method is poor in performance, and the general wait time is quite high.

Example of FCFS scheduling

A real-life example of the FCFS method is buying a movie ticket on the ticket counter. In this scheduling algorithm, a person is served according to the queue manner. The person who arrives first in the queue first buys the ticket and then the next one. This will continue until the last person in the queue purchases the ticket. Using this algorithm, the CPU process works in a similar manner.

Project Description

Purpose:

This mini project gives us the knowledge about first come first serve (FCFS) or first in first out (FIFO) system. The purpose of this project is to give us the idea about the basic queue system that we have seen at various places in our day-to-day life.

Motivation:

FCFS scheduling supports non-pre-emptive and pre-emptive scheduling algorithm. It is non-pre-emptive CPU scheduling algorithm, so after the process has been allocated to the CPU, it will never release the CPU until it finishes executing.

Problem statement:

To Implement the FCFS scheduling algorithm using efficient data structure

Project Perspective:

Given n processes with their burst times, the task is to find average waiting time and average turnaround time using FCFS scheduling algorithm. First in, first out (FIFO) simply queues processes in the order that they arrive in the ready queue. In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed.

System Requirements:

- System should have minimum of 125GB SSD in order to speed up the execution of source code.
- I have used replit (online compiler) and Visual Studio code (offline compiler) for compilation and execution of the source code.

Methods and Implementation

- **Module / function Discussion:**

1. `typedef struct node{}node;`

In this module, I have used structure data type to create a linked list node in which I will take data of different data types required by the program. I have also used typedef keyword in order to save time and to overcome some typing errors.

2. `node * createLinkedList(int n);`

This module is used to dynamically allocate the memory to the node for the doubly linked list. This function also asks the user input over arrival and burst time respectively and also creates links between different nodes and returns head.

3. `void display(node * head);`

In this function, traversal of linked list is done and also prepares the table of the input data and hence prints it in output section.

4. `int waiting_time(node * temp,int i);`

This function contains if else statement which is dependent on the for loop coded in *`void all();`* function. Its main aim is to calculate the waiting time of different processes.

5. `void all(node * head , int n);`

Its main aim is to make the chart of different calculated times asked in the project. It also prints the value of average waiting and turnaround time.

6. `node * enqueue(node * head ,int n, int k);`

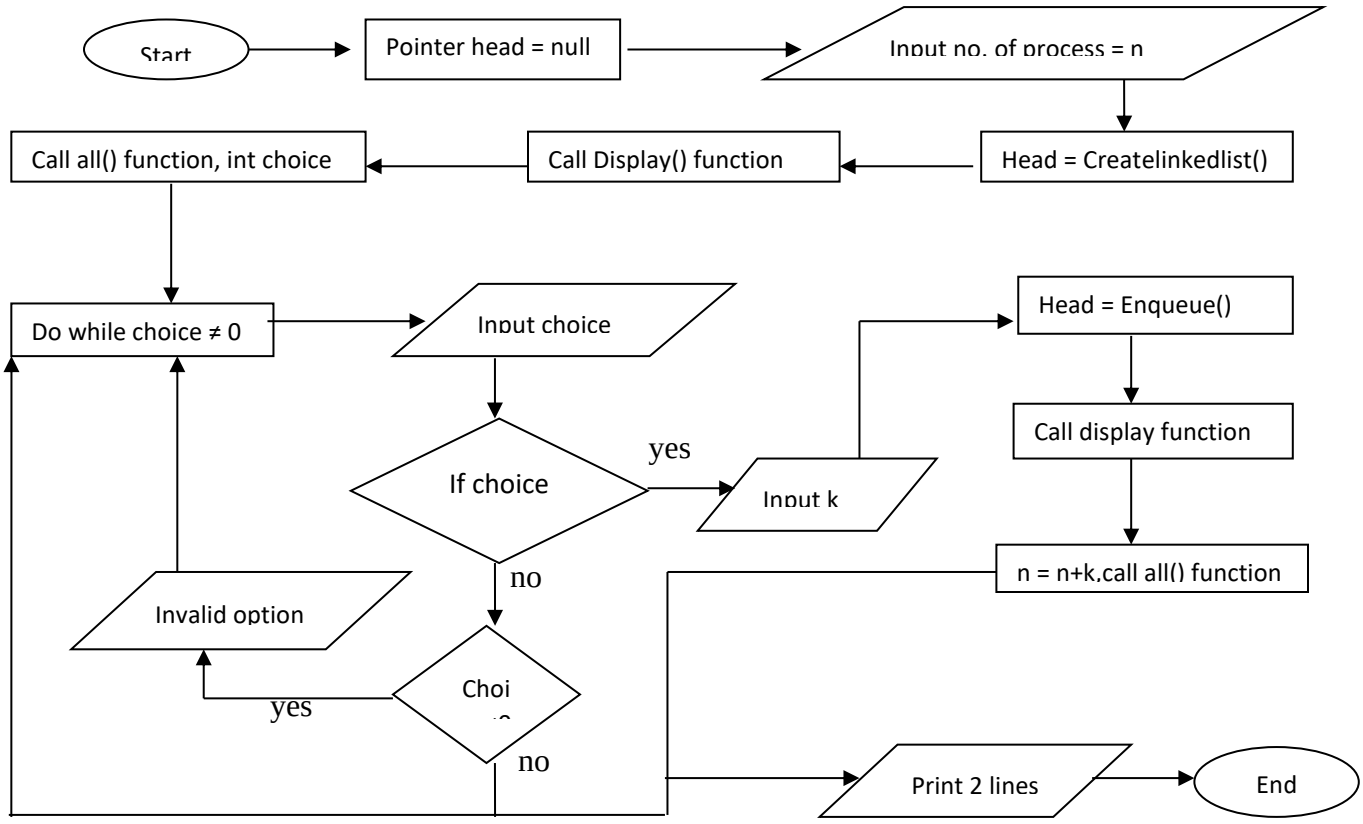
After all the processes, if the user wants to add more processes in the list then this function is used. This function is choice depended if user wants it to add or not. Basically this recalls all the functions discussed above.

7.
```
int main(){
    return 0;
}
```

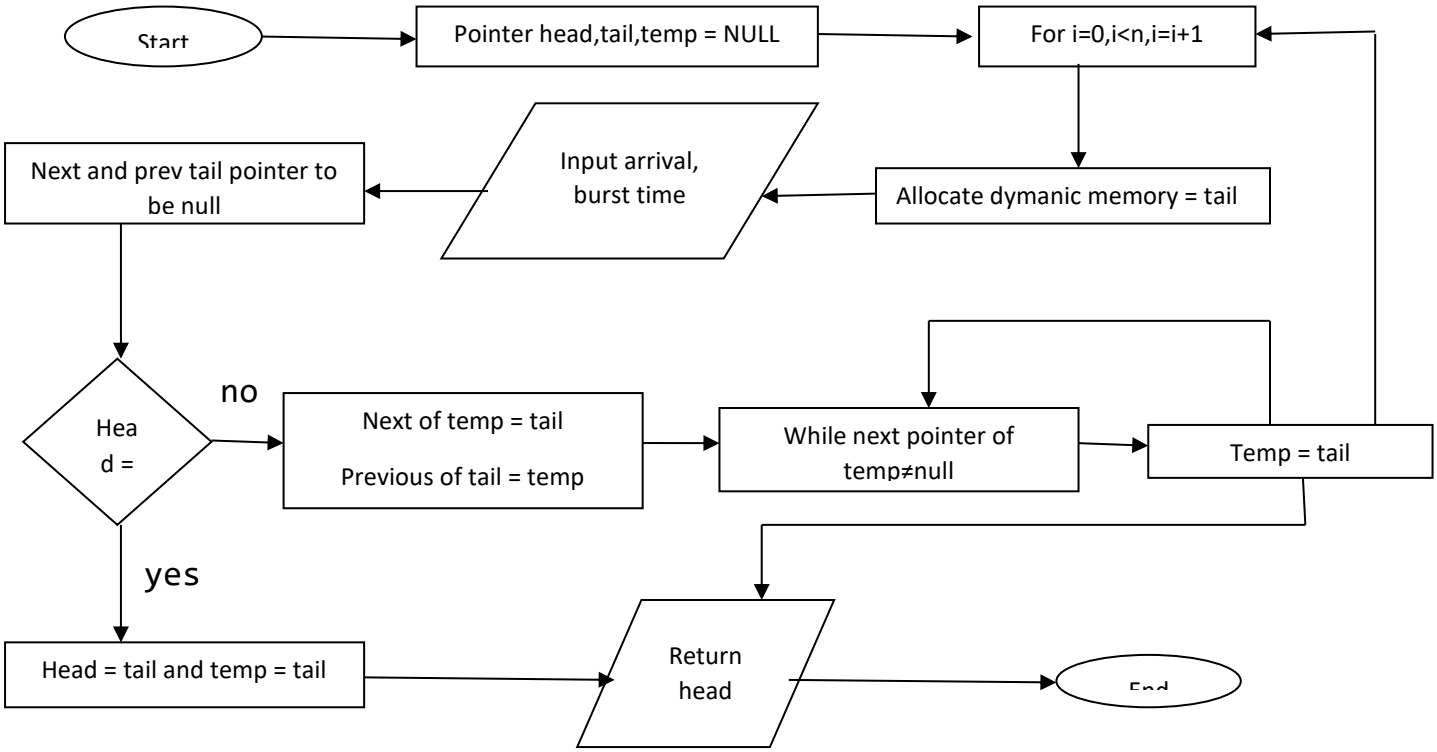
This function does the designing part of the project and also calling of functions is done in this part only.

- **Flowchart of each module/function**

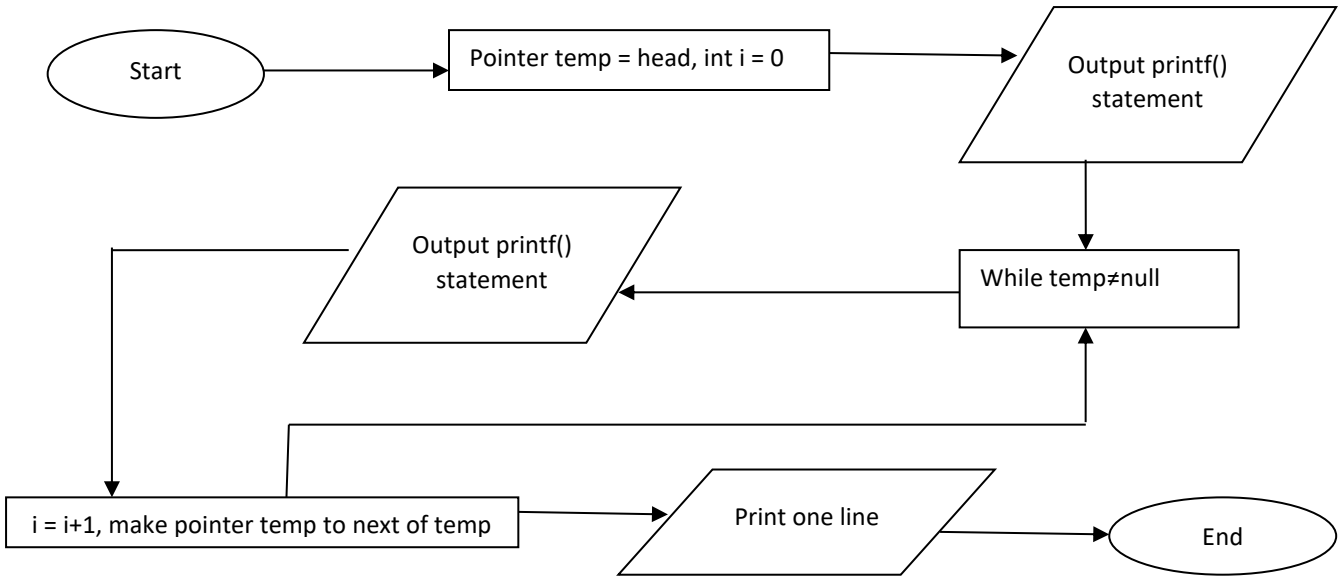
1. `int main()`



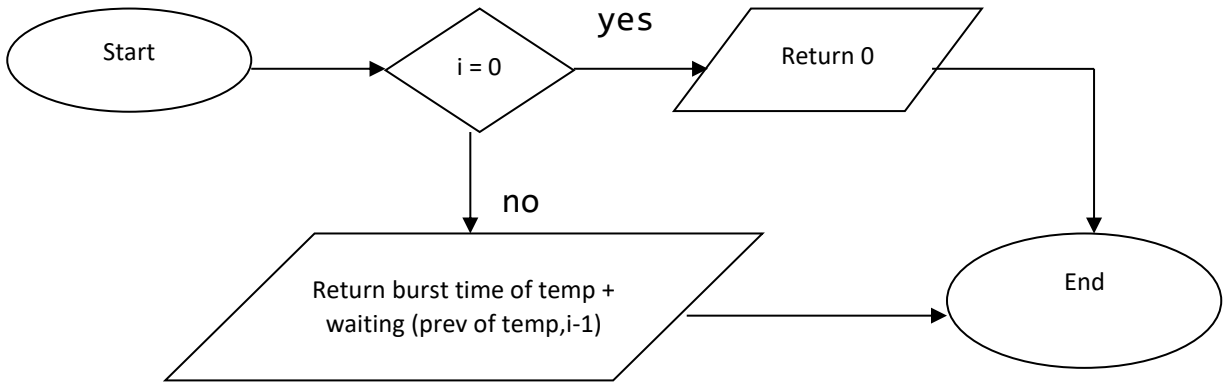
2. node * createLinkedList(int n);



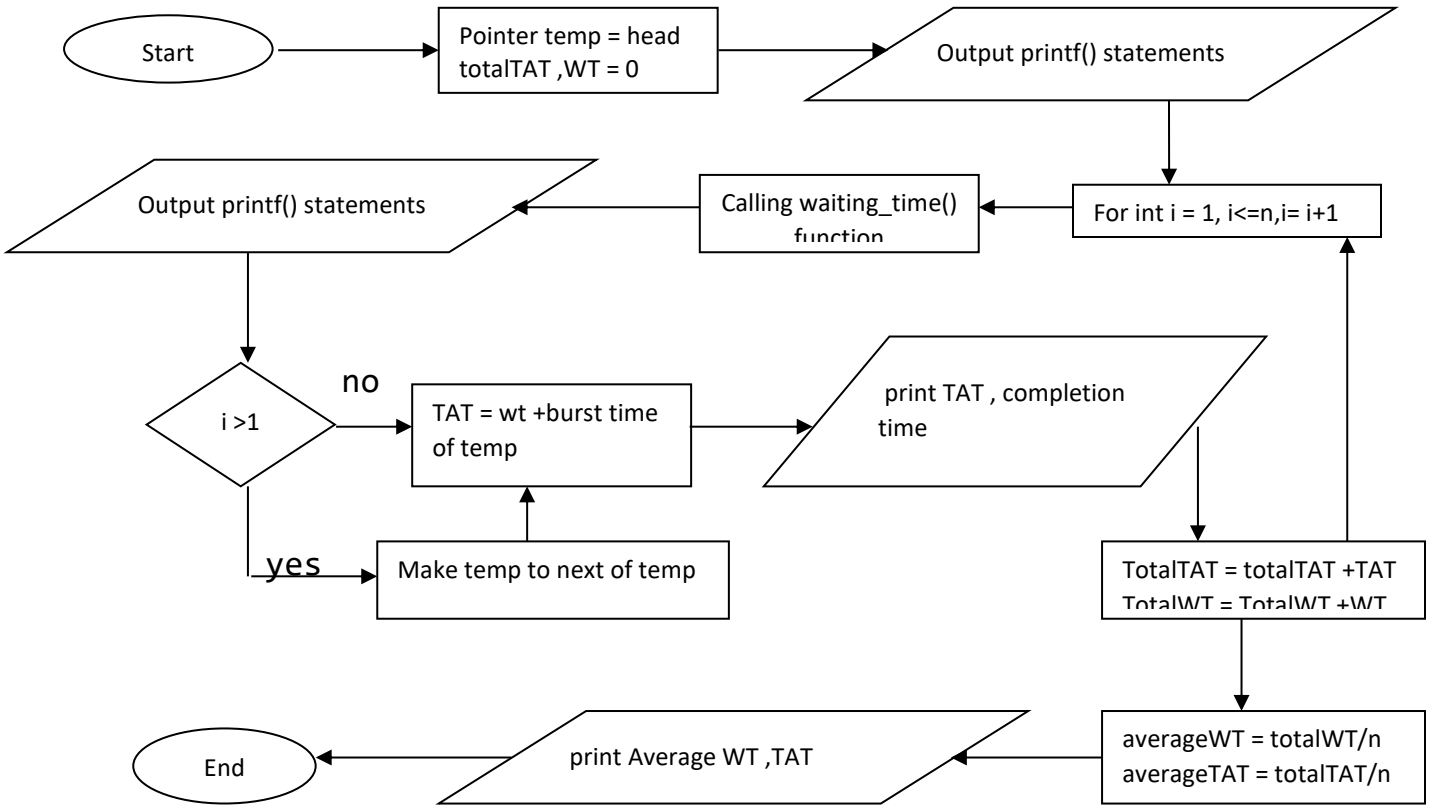
3. void display(node * head);



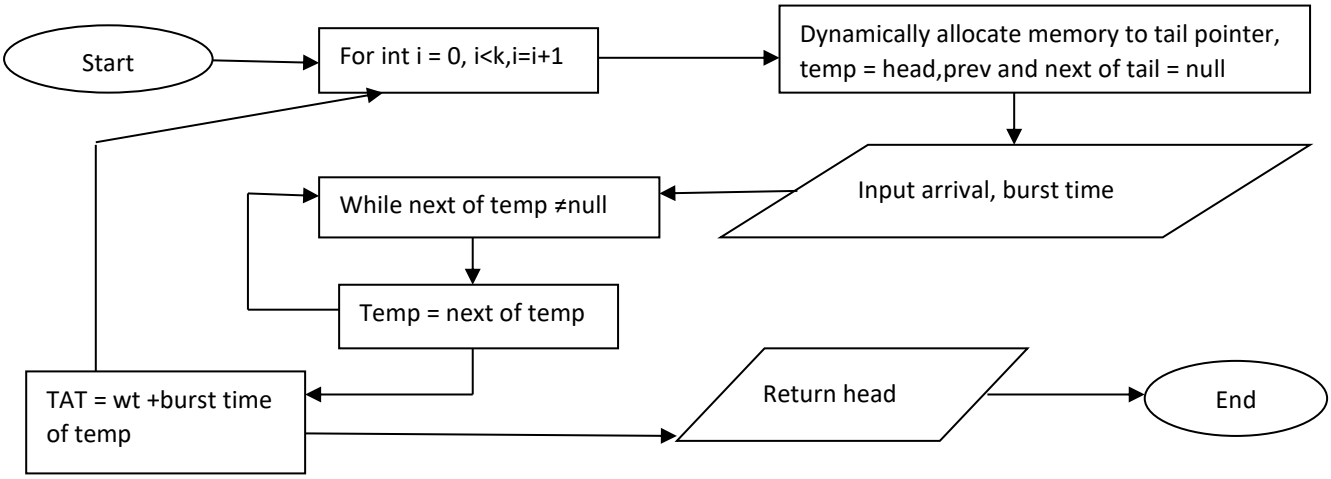
4. `int waiting_time(node * temp,int i);`



5. `void all(node * head , int n);`



6. node * enqueue(node * head ,int n, int k);



- **Data Structure used in the Project**

  In this project, I have used implementation of Queue using doubly linked list. Queue is a linear data structure in which insertion is done from rear and deletion is performed from front end. A queue is a FIFO based data structure. Doubly linked list is more efficient to implement a queue because of easy traversal.

- **Source Code**

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

typedef struct node
{
    int burst_time;
    int arrival_time;
    struct node * next;
    struct node * prev;
}node;
```

```
node * createLinkedList(int n){
    node * head = NULL;
    node * tail = NULL;
    node * temp = NULL;

 for (int i = 0; i < n; i++)
 {
     tail = (node *)malloc(sizeof(node));
     printf("\n\nPROCESS %d : ",i+1);
     printf("\nEnter the Arrival Time of Process %d ('0'
RECOMMENDED) : ",i+1);
     scanf("%d",&(tail->arrival_time));
     printf("Enter the Burst Time of Process %d : ",i+1);
     scanf("%d",&(tail->burst_time));
     tail->prev = NULL;
     tail->next = NULL;

      if(head == NULL){
          head = tail;
          temp = tail;
      }
      else{
        temp->next = tail;
          tail->prev = temp;

          while(temp->next!=NULL){
           temp = tail;
          }
        }
      }
 return head;
 }

 void display(node * head){
```

```
    node * temp = head;
   int i = 0;
   printf("\nTraversing the list now :\n");
          printf("\n  Processes\t\t  Arrival Time\t\t    Burst
Time\n");
      while (temp!=NULL)
      {
          printf("Process: %d \t\t\t %d \t\t\t %d\n",i+1,temp-
>arrival_time,temp->burst_time);
          i++;
          temp = temp->next;
      }
      printf("\n");
  }

  int waiting_time(node * temp,int i){


      if (i == 1)
      {
        return 0;
      }
      else
      {
          return temp->burst_time + waiting_time(temp->prev,i-1);
      }

  }

  void all(node * head , int n){
     node * temp = head;
    int total_TAT = 0;
    int total_WT = 0;
    int wt;
```

```
    printf("\n\n\t\t   SHOWING CHART OF DIFFERENT CALCULATED
TIMES\n");
    printf("\n  Processes\t\t  Waiting Time\t\tTurnAround
Time\t\tCompletion Time\n");
    for (int i = 1; i <= n; i++)
    {
    wt = waiting_time(temp,i);
    printf("Process: %d \t\t\t %d ",i,wt);

    if(i > 1){
        temp = temp->next;
    }

    int TAT = wt + temp->burst_time;
    printf("\t\t\t %d ",TAT);
    printf("\t\t\t %d\n",TAT + temp->arrival_time);
    total_WT += wt;
    total_TAT += TAT;
    }
    float avg_WT = (float)total_WT / n;
    float avg_TAT = (float)total_TAT / n;

    printf("\nAverage Waiting Time    =\t%.2f",avg_WT);
    printf("\nAverage Turnaround Time
=\t\t\t\t%.2f\n\n",avg_TAT);

  }

  node * enqueue(node * head ,int n, int k){

      for (int i = 0; i < k; i++)
      {
      node * tail = (node *)malloc(sizeof(node));
      node * temp = head;
      tail->prev = NULL;
```

```
      tail->next = NULL;
      printf("\n\nPROCESS %d : ",n+i+1);
     printf("\nEnter the Arrival Time of Process %d ('0'
RECOMMENDED) : ",n+i+1);
     scanf("%d",&(tail->arrival_time));
     printf("Enter the Burst Time of Process %d : ",n+i+1);
     scanf("%d",&(tail->burst_time));

     while (temp->next !=NULL)
     {
         temp = temp->next;
     }
     tail->prev = temp;
     temp->next = tail;

      }

      return head;

  }

  int main(){
    node * head = NULL;
    int n;

    printf("\n\n\n\n/////////**********    Implementation of the
FCFS scheduling algorithm    **********/////////\n");
            printf("/////////**********
using                          **********/////////\n");
            printf("/////////**********             Queue in
Doubly Linked List               **********/////////\n");
    printf("\nEnter the number of Processes : ");
    scanf("%d",&n);
    head = createLinkedList(n);
    display(head);
```

```
    all(head,n);
    int choice;
    do{
    printf("Want to Enqueue more processes \n('1' Proceed  ||
'0' Reject): ");
    scanf("%d",&choice);

    if(choice == 1){
        int k;
      printf("Enter the number of Processes to Enqueue more : ");
      scanf("%d",&k);
        head = enqueue(head ,n, k);
        display(head);
        n = n + k;
        all(head,n);
    }
    else if(choice != 1 && choice != 0)
    {
        printf("Invalid option !!");
    }
    }
    while(choice != 0);
    printf("\n\n");

return 0;
}
```

- **Screen shots of codes and output**

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

typedef struct node
{
    int burst_time;
    int arrival_time;
    struct node * next;
    struct node * prev;
}node;

node * createLinkedList(int n){
    node * head = NULL;
    node * tail = NULL;
    node * temp = NULL;

 for (int i = 0; i < n; i++)
 {
     tail = (node *)malloc(sizeof(node));
     printf("\n\nPROCESS %d : ",i+1);
     printf("\nEnter the Arrival Time of Process %d ('0' RECOMMENDED) : ",i+1);
     scanf("%d",&(tail->arrival_time));
     printf("Enter the Burst Time of Process %d : ",i+1);
     scanf("%d",&(tail->burst_time));
     tail->prev = NULL;
     tail->next = NULL;

      if(head == NULL){
          head = tail;
          temp = tail;
      }
```

```c
      else{
        temp->next = tail;
          tail->prev = temp;

          while(temp->next!=NULL){
           temp = tail;
          }
        }
      }
return head;
}

void display(node * head){
    node * temp = head;
  int i = 0;
  printf("\nTraversing the list now :\n");
          printf("\n  Processes\t\t  Arrival Time\t\t    Burst Time\n");
      while (temp!=NULL)
      {
          printf("Process: %d \t\t\t %d \t\t\t %d\n",i+1,temp->arrival_time,temp->burst_time);
         // printf("Process: %d \t\t\t %d\n\n",i+1,temp->burst_time);
          i++;
          temp = temp->next;
      }
      printf("\n");
 }

 int waiting_time(node * temp,int i){
```

Source code 1

```
62        if (i == 1)
63        {
64          return 0;
65        }
66        else
67        {
68            return temp->burst_time + waiting_time(temp->prev,i-1);
69        }
70
71    }
72
73    void all(node * head , int n){
74       node * temp = head;
75      int total_TAT = 0;
76      int total_WT = 0;
77      int wt;
78      printf("\n\n\t\t    SHOWING CHART OF DIFFERENT CALCULATED TIMES\n");
79      printf("\n  Processes\t\t  Waiting Time\t\tTurnAround Time\t\tCompletion Time\n");
80      for (int i = 1; i <= n; i++)
81      {
82      wt = waiting_time(temp,i);
83      printf("Process: %d \t\t\t %d ",i,wt);
84
85      if(i > 1){
86          temp = temp->next;
87      }
88
89      int TAT = wt + temp->burst_time;
90      printf("\t\t\t %d ",TAT);
91      printf("\t\t\t %d\n",TAT + temp->arrival_time);
92      total_WT += wt;
93      total_TAT += TAT;
```

```
94      }
95      float avg_WT = (float)total_WT / n;
96      float avg_TAT = (float)total_TAT / n;
97
98      printf("\nAverage Waiting Time     =\t%.2f",avg_WT);
99      printf("\nAverage Turnaround Time =\t\t\t%.2f\n\n",avg_TAT);
100
101   }
102
103   node * enqueue(node * head ,int n, int k){
104
105       for (int i = 0; i < k; i++)
106       {
107       node * tail = (node *)malloc(sizeof(node));
108       node * temp = head;
109       tail->prev = NULL;
110       tail->next = NULL;
111       printf("\n\nPROCESS %d : ",n+i+1);
112      printf("\nEnter the Arrival Time of Process %d ('0' RECOMMENDED) : ",n+i+1);
113      scanf("%d",&(tail->arrival_time));
114      printf("Enter the Burst Time of Process %d : ",n+i+1);
115      scanf("%d",&(tail->burst_time));
116
117      while (temp->next !=NULL)
118      {
119          temp = temp->next;
120      }
121      tail->prev = temp;
122      temp->next = tail;
123
124       }
125
```

Source Code 2

```
    return head;
      }
int main(){
  node * head = NULL;
  int n;
  printf("\n\n\n\n\n//////////**********    Implementation of the FCFS scheduling algorithm    **********//////////\n");
        printf("//////////**********                       using                           **********//////////\n");
        printf("//////////**********            Queue in Doubly Linked List                **********//////////\n");
  printf("\nEnter the number of Processes : ");
  scanf("%d",&n);
  head = createLinkedList(n);
  display(head);
  all(head,n);
  int choice;
  do{
  printf("Want to Enqueue more processes \n('1' Proceed  ||  '0' Reject): ");
  scanf("%d",&choice);
  if(choice == 1){
      int k;
    printf("Enter the number of Processes to Enqueue more : ");
    scanf("%d",&k);
      head = enqueue(head ,n, k);
      display(head);
      n = n + k;
      all(head,n);
  }
  else if(choice != 1 && choice != 0) {
      printf("Invalid option !!");
    }
  }while(choice != 0);
  printf("\n\n");
  return 0;
```

Source Code 3

```
//////////**********    Implementation of the FCFS scheduling algorithm    **********//////////
//////////**********                       using                           **********//////////
//////////**********            Queue in Doubly Linked List                **********//////////

Enter the number of Processes : 3


PROCESS 1 :
Enter the Arrival Time of Process 1 ('0' RECOMMENDED) : 1
Enter the Burst Time of Process 1 : 5


PROCESS 2 :
Enter the Arrival Time of Process 2 ('0' RECOMMENDED) : 1
Enter the Burst Time of Process 2 : 4


PROCESS 3 :
Enter the Arrival Time of Process 3 ('0' RECOMMENDED) : 1
Enter the Burst Time of Process 3 : 9

Traversing the list now :

  Processes               Arrival Time              Burst Time
Process: 1                      1                         5
Process: 2                      1                         4
Process: 3                      1                         9
```

Output 1

```
                SHOWING CHART OF DIFFERENT CALCULATED TIMES

  Processes              Waiting Time          TurnAround Time          Completion Time
Process: 1                     0                      5                       6
Process: 2                     5                      9                       10
Process: 3                     9                      18                      19

Average Waiting Time    =       4.67
Average Turnaround Time =                          10.67

Want to Enqueue more processes
('1' Proceed  ||  '0' Reject): 1
Enter the number of Processes to Enqueue more : 2


PROCESS 4 :
Enter the Arrival Time of Process 4 ('0' RECOMMENDED) : 1
Enter the Burst Time of Process 4 : 12


PROCESS 5 :
Enter the Arrival Time of Process 5 ('0' RECOMMENDED) : 1
Enter the Burst Time of Process 5 : 6

Traversing the list now :

  Processes              Arrival Time             Burst Time
Process: 1                     1                      5
Process: 2                     1                      4
Process: 3                     1                      9
Process: 4                     1                      12
Process: 5                     1                      6
```

```
                SHOWING CHART OF DIFFERENT CALCULATED TIMES

  Processes              Waiting Time          TurnAround Time          Completion Time
Process: 1                     0                      5                       6
Process: 2                     5                      9                       10
Process: 3                     9                      18                      19
Process: 4                     18                     30                      31
Process: 5                     30                     36                      37

Average Waiting Time    =       12.40
Average Turnaround Time =                          19.60

Want to Enqueue more processes
('1' Proceed  ||  '0' Reject): 0


PS C:\Users\hp\Desktop\CSE code with harry\vs code\mydirectory>
```

Output 2

- **Result analysis**

1. According to this result, we can say that the waiting time of the 1st process will always remain 0 as 1st process that comes need not to wait for any other.
2. As the other processes come they will be queued one after the other and hence waiting time will increase for further consecutive processes.
3. We know TAT = waiting time + burst time(input), hence TAT and also completion time will also increase for further consecutive processes.