UITS

UNIVERSITY OF INFORMATION
TECHNOLOGY AND SCIENCES

NAME	:- KHALED SAIFULLAH
ID	:- 1814351102
BATCH	:- 43
DEPARTMENT	:- CSE
COURSE TITLE	:-OPERATING SYSTEM
COURSE TEACHER	:-CSE-321
COURSE TEACHER	:- MALIHA HOSSIAN

Lab Report No :- 01

Experiment Name :- c/c++ Programming to implement FCFS and as well as use that programming language to draw grand.

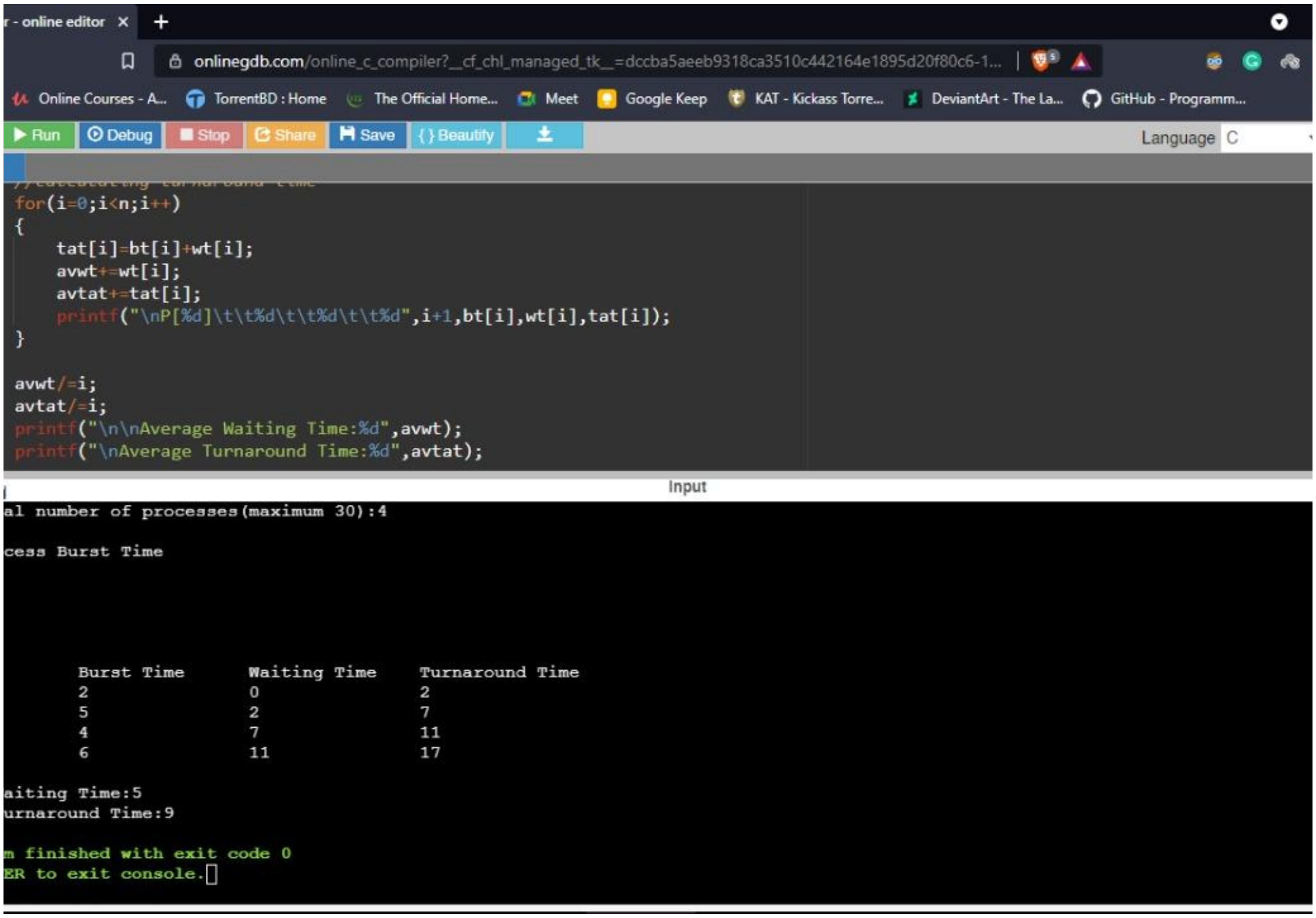
Theory

First come first serve (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time. The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU. FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

Source Code

```
#include<stdio.h>
int main()
{
    int n,bt[25],wt[30],tat[30],avwt=0,avtat=0,i,j;
    printf("Enter total number of processes(maximum 30):");
    scanf("%d",&n);

    printf("\nEnter Process Burst Time\n");
    for(i=0;i<n;i++)
    {
        printf("P[%d]:",i+1);
        scanf("%d",&bt[i]);
    }
}

wt[0]=0;    //waiting time for first process is 0

//calculating waiting time
for(i=1;i<n;i++)
{
    wt[i]=0;
    for(j=0;j<i;j++)
        wt[i]+=bt[j];
}

printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

//calculating turnaround time
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];
    avwt+=wt[i];
    avtat+=tat[i];
    printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
}
```

```
    avwt/=i;

    avtat/=i;

    printf("\n\nAverage Waiting Time:%d",avwt);

    printf("\nAverage Turnaround Time:%d",avtat);



    return 0;

}
```



```
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];
    avwt+=wt[i];
    avtat+=tat[i];
    printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
}

avwt/=i;
avtat/=i;
printf("\n\nAverage Waiting Time:%d",avwt);
printf("\nAverage Turnaround Time:%d",avtat);
```

```
al number of processes(maximum 30):4

cess Burst Time





       Burst Time       Waiting Time      Turnaround Time
       2                0                 2
       5                2                 7
       4                7                 11
       6                11                17

aiting Time:5
urnaround Time:9

m finished with exit code 0
ER to exit console.
```

Lab Report No :- 02

Experiment Name :- c/c++ Programming to implement SJF and as well as use that programming language to draw grand.

Theory

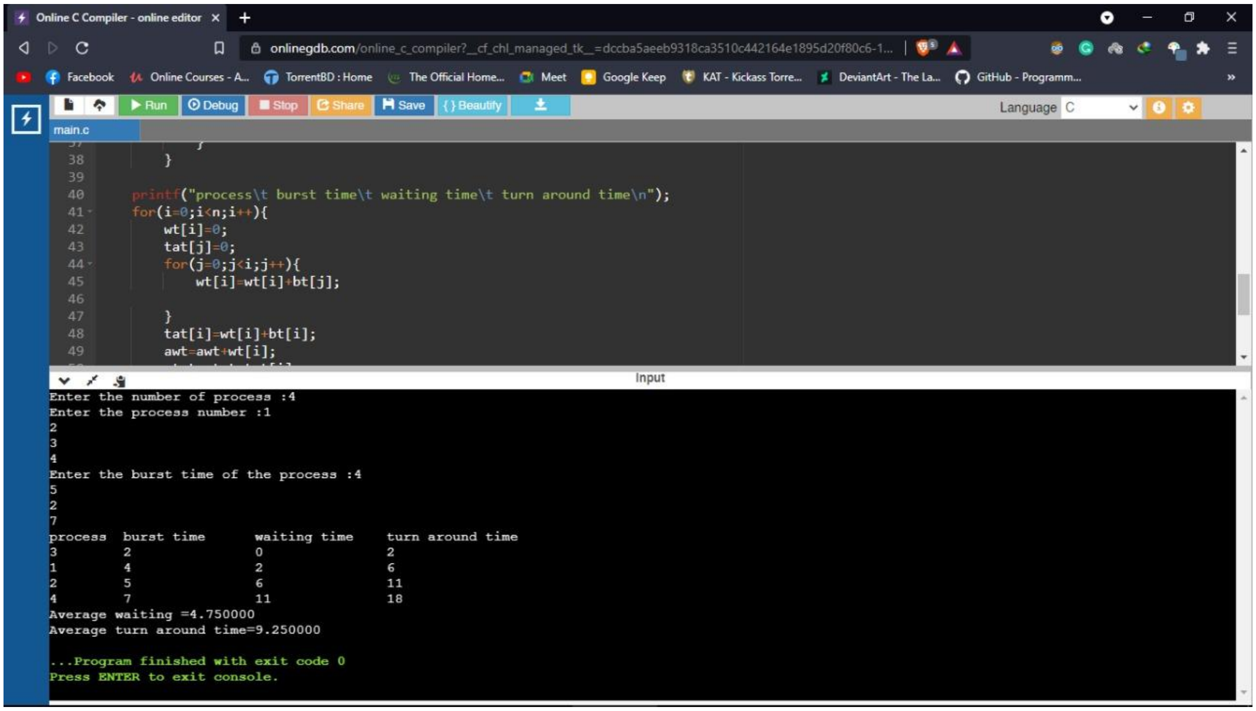
Shortest job first (SJF) is a scheduling algorithm, that is used to schedule processes in an operating system. It is a very important topic in Scheduling when compared to round-robin and FCFS Scheduling

Source Code

```
#include<stdio.h>

#include<conio.h>

# define max 30

void main(){

   int i,j,n,t,p[max],bt[max],wt[max],tat[max];

   float awt=0,atat=0;


   printf("Enter the number of process :");
```

```
scanf("%d",&n);
printf("Enter the process number :");
for(i=0;i<n;i++)
    {
    scanf("%d",&p[i]);
    }
    printf("Enter the burst time of the process :");
    for(int i=0;i<n;i++)
        {
        scanf("%d",&bt[i]);

    }
    //bubble sort according to their burst time
    for(i=0;i<n;i++)
    {
        for(j=0;j<n-i-1;j++)
        {

            if(bt[j]>bt[j+1])
            {
                t=bt[j];
                bt[j]=bt[j+1];
                bt[j+1]=t;
                t=p[j];
                p[j]=p[j+1];
                p[j+1]=t;
            }
        }
    }

printf("process\t burst time\t waiting time\t turn around time\n");
for(i=0;i<n;i++){
    wt[i]=0;
    tat[j]=0;
    for(j=0;j<i;j++){
        wt[i]=wt[i]+bt[j];

    }
    tat[i]=wt[i]+bt[i];
```

```
        awt=awt+wt[i];

        atat=atat+tat[i];

        printf("%d\t %d\t\t %d\t\t %d\n", p[i],bt[i],wt[i],tat[i]);

    }

    awt=awt/n;

    atat=atat/n;

    printf("Average waiting =%f\n" ,awt);

    printf("Average turn around time=%f",atat);

    getch();


}
```



```
        }

    printf("process\t burst time\t waiting time\t turn around time\n");
    for(i=0;i<n;i++){
        wt[i]=0;
        tat[j]=0;
        for(j=0;j<i;j++){
            wt[i]=wt[i]+bt[j];

        }
        tat[i]=wt[i]+bt[i];
        awt=awt+wt[i];
```

```
Enter the number of process :4
Enter the process number :1
2
3
4
Enter the burst time of the process :4
5
2
7
process  burst time       waiting time     turn around time
3        2                0                2
1        4                2                6
2        5                6                11
4        7                11               18
Average waiting =4.750000
Average turn around time=9.250000

...Program finished with exit code 0
Press ENTER to exit console.
```

Lab Report No :- 03

Experiment Name :- c/c++ Programming to implement Round Robin and as well as use that programming language to draw grand.

Theory

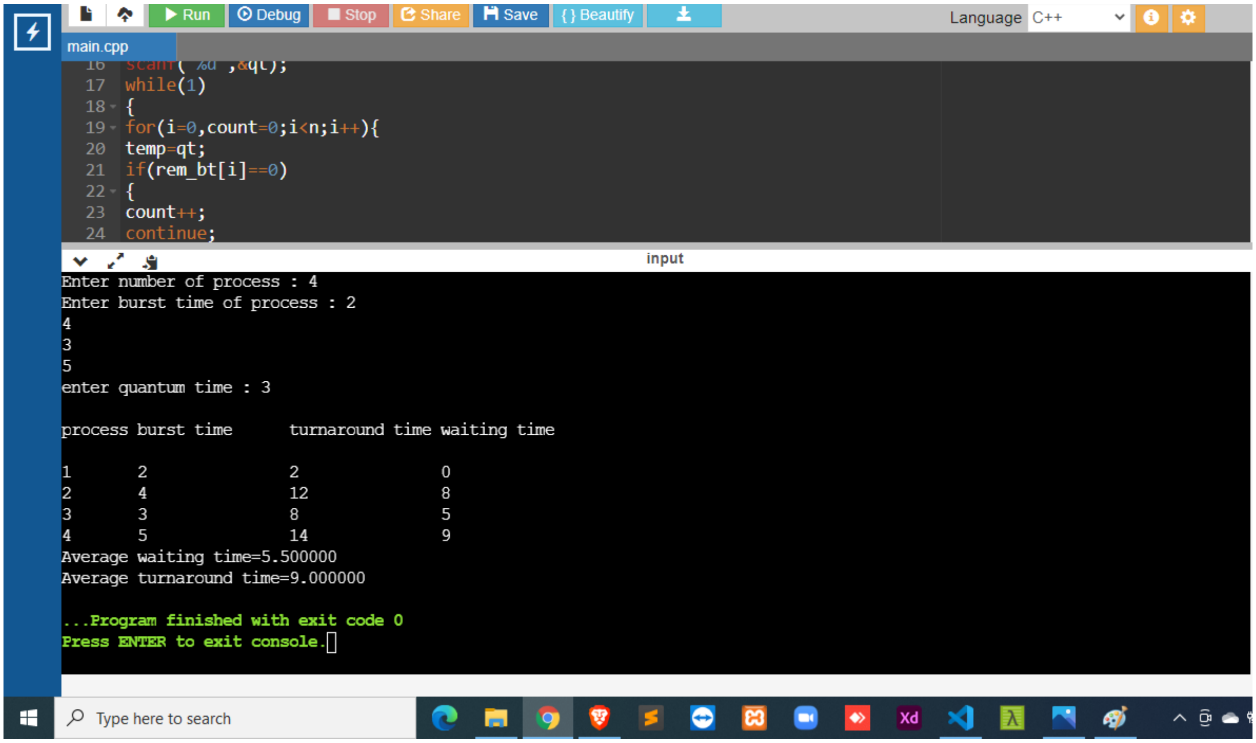
Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. It is simple, easy to implement, and starvation-free as all processes get fair share of CPU. It is preemptive as processes are assigned CPU only for a fixed slice of time at most.

Source Code

```
#include<stdio.h>

#include<conio.h>


void main()

{

   int n, i, qt,count=0,temp,sq=0,bt[10],wt[10],tat[10],rem_bt[10];
```

```
float awt=0,atat=0;


printf("Enter number of process :-");
scanf("%d",&n);
printf("Enter burst time of process :-");
for(i=0;i<n;i++){
    scanf("%d",&bt[i]);
    rem_bt[i]=bt[i];
}
printf("enter quantum time :-");
scanf("%d",&qt);
while(1)
{


    for(i=0,count=0;i<n;i++){
        temp=qt;
        if(rem_bt[i]==0)
        {
            count++;
            continue;
        }
        if(rem_bt[i]>qt)
            rem_bt[i]= rem_bt[i]-qt;
        else
            if(rem_bt[i]>=0)
        {
            temp=rem_bt[i];
            rem_bt[i]=0;
        }
        sq=sq+temp;
        tat[i]=sq;
    }
    if(n==count)
        break;
}
printf("\nprocess\tburst time\tturnaround time\twaiting time\n ");
for(i=0;i<n;i++)
{
    wt[i]=tat[i]-bt[i];
    awt=awt+wt[i];
```

```
        atat=atat+tat[i];

        printf("\n%d\t%d\t\t%d\t\t%d",i+1,bt[i],tat[i],wt[i]);

    }

    awt=awt/n;

    atat=atat/n;

    {

    printf("\nAverage waiting time=%f",awt);

    printf("\nAverage turnaround time=%f",atat);

    getch();

    }

}
```



```
scanf("%d",&qt);
while(1)
{
for(i=0,count=0;i<n;i++){
temp=qt;
if(rem_bt[i]==0)
{
count++;
continue;
```

```
Enter number of process : 4
Enter burst time of process : 2
4
3
5
enter quantum time : 3

process burst time      turnaround time waiting time

1       2               2               0
2       4               12              8
3       3               8               5
4       5               14              9
Average waiting time=5.500000
Average turnaround time=9.000000

...Program finished with exit code 0
Press ENTER to exit console.
```

Lab Report No :- 04

Experiment Name :- c/c++ Programming to implement Priority Scheduling and as well as use that programming language to draw grand.

Theory

Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned a priority. Process with highest priority is to be executed first and so on. Processes with same priority are executed on first come first served basis.

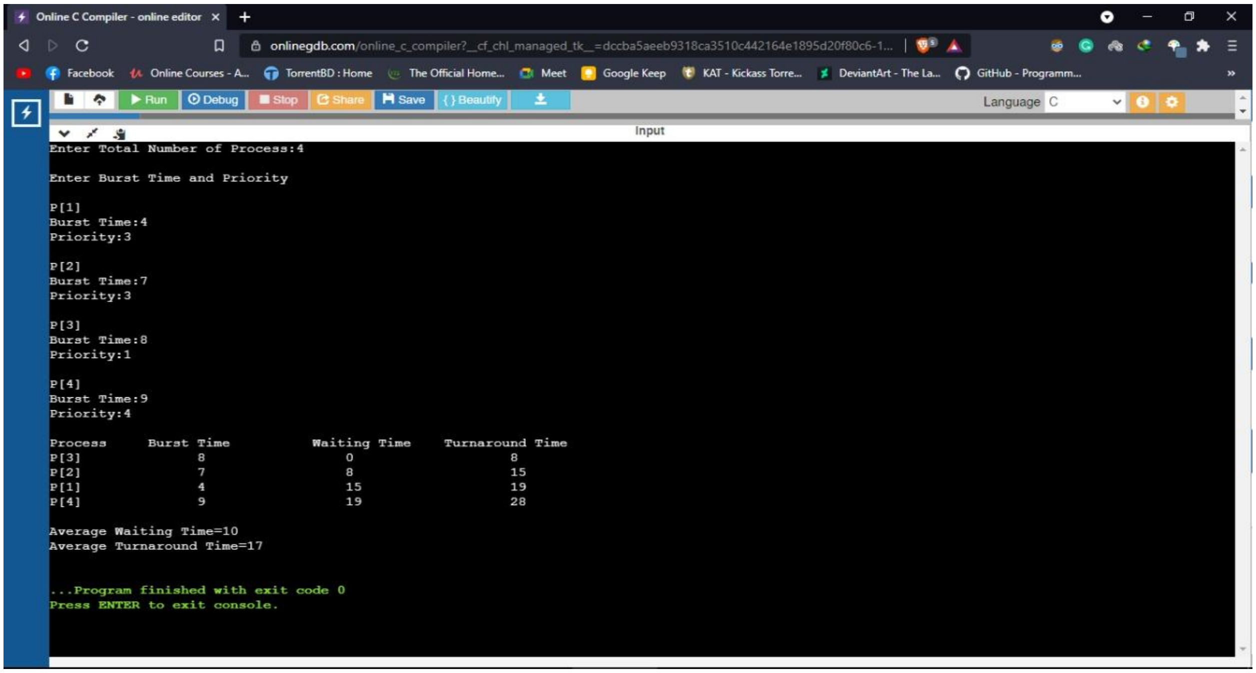
Source Code

```
#include<stdio.h>


int main()

{

   int bt[25],p[26],wt[25],tat[25],pr[25],i,j,n,total=0,pos,temp,avg_wt,avg_tat;

   printf("Enter Total Number of Process:");
```

```
    scanf("%d",&n);

    printf("\nEnter Burst Time and Priority\n");
    for(i=0;i<n;i++)
    {
        printf("\nP[%d]\n",i+1);
        printf("Burst Time:");
        scanf("%d",&bt[i]);
        printf("Priority:");
        scanf("%d",&pr[i]);
        p[i]=i+1;           //contains process number
    }

    //sorting burst time, priority and process number in ascending order
using selection sort
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
```

```
    if(pr[j]<pr[pos])

      pos=j;

  }


  temp=pr[i];

  pr[i]=pr[pos];

  pr[pos]=temp;


  temp=bt[i];

  bt[i]=bt[pos];

  bt[pos]=temp;


  temp=p[i];

  p[i]=p[pos];

  p[pos]=temp;

}


wt[0]=0; //waiting time for first process is zero


//calculate waiting time
```

```
    }

    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    }

    avg_wt=total/n;      //average waiting time
    total=0;

    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];     //calculate turnaround time
        total+=tat[i];
        printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }
    avg_tat=total/n;     //average turnaround time

    printf("\n\nAverage Waiting Time=%d",avg_wt);

    printf("\nAverage Turnaround Time=%d\n",avg_tat);


return 0;

}
```



```
Enter Total Number of Process:4

Enter Burst Time and Priority

P[1]
Burst Time:4
Priority:3

P[2]
Burst Time:7
Priority:3

P[3]
Burst Time:8
Priority:1

P[4]
Burst Time:9
Priority:4

Process     Burst Time          Waiting Time     Turnaround Time
P[3]              8                  0                   8
P[2]              7                  8                   15
P[1]              4                  15                  19
P[4]              9                  19                  28

Average Waiting Time=10
Average Turnaround Time=17


...Program finished with exit code 0
Press ENTER to exit console.
```