Practice Assignment # 03

SJF

Shortest Job First Scheduling (SJF)

Shortest Job First (SJF) or Shortest Process Next (SPN), is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-preemptive algorithm. Shortest remaining time is a preemptive variant of SJN.

SJF is advantageous because of its simplicity and because it minimizes the average amount of time each process has to wait until its execution is complete. However, it has the potential for process starvation for processes which will require a long time to complete if short processes are continually added. Another disadvantage of using SJF is that the total execution time of a job must be known before execution. While it is not possible to perfectly predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times.

QUESTION

Implement SJF (Shortest Job First) CPU Scheduling Algorithm using C,C++. Also attach the snapshot of Code and output window.

Implementation and Algorithm

AIM: To write a C program to implement the CPU scheduling algorithm for shortest job first.

PROBLEM DESCRIPTION:

Cpu scheduler will decide which process should be given the CPU for its execution. For this it uses different algorithm to choose among the process. One among that algorithm is SJF algorithm.

In this algorithm the process which has less service time given the cpu after finishing its request only it will allow cpu to execute next other process.

Algorithm

ALGORITHM:

```
Step 1: Get the number of process.
Step 2: Get the id and service time for each process.
Step 3: Initially the waiting time of first short process as 0 and total time of
        first short is process the service time of that process.
Step 4: Calculate the total time and waiting time of remaining process.
Step 5: Waiting time of one process is the total time of the previous process.
Step 6: Total time of process is calculated by adding the waiting time and service
        time of each process.
Step 7: Total waiting time calculated by adding the waiting time of each process.
Step 8: Total turn around time calculated by adding all total time of each process. Step 9: Calculate average waiting time by dividing the total waiting time by total
        number of process.
Step 10: Calculate average turn around time by dividing the total waiting time
         by total number of process.
Step 11: Display the result.
```

Implementation

PROGRAM CODING:

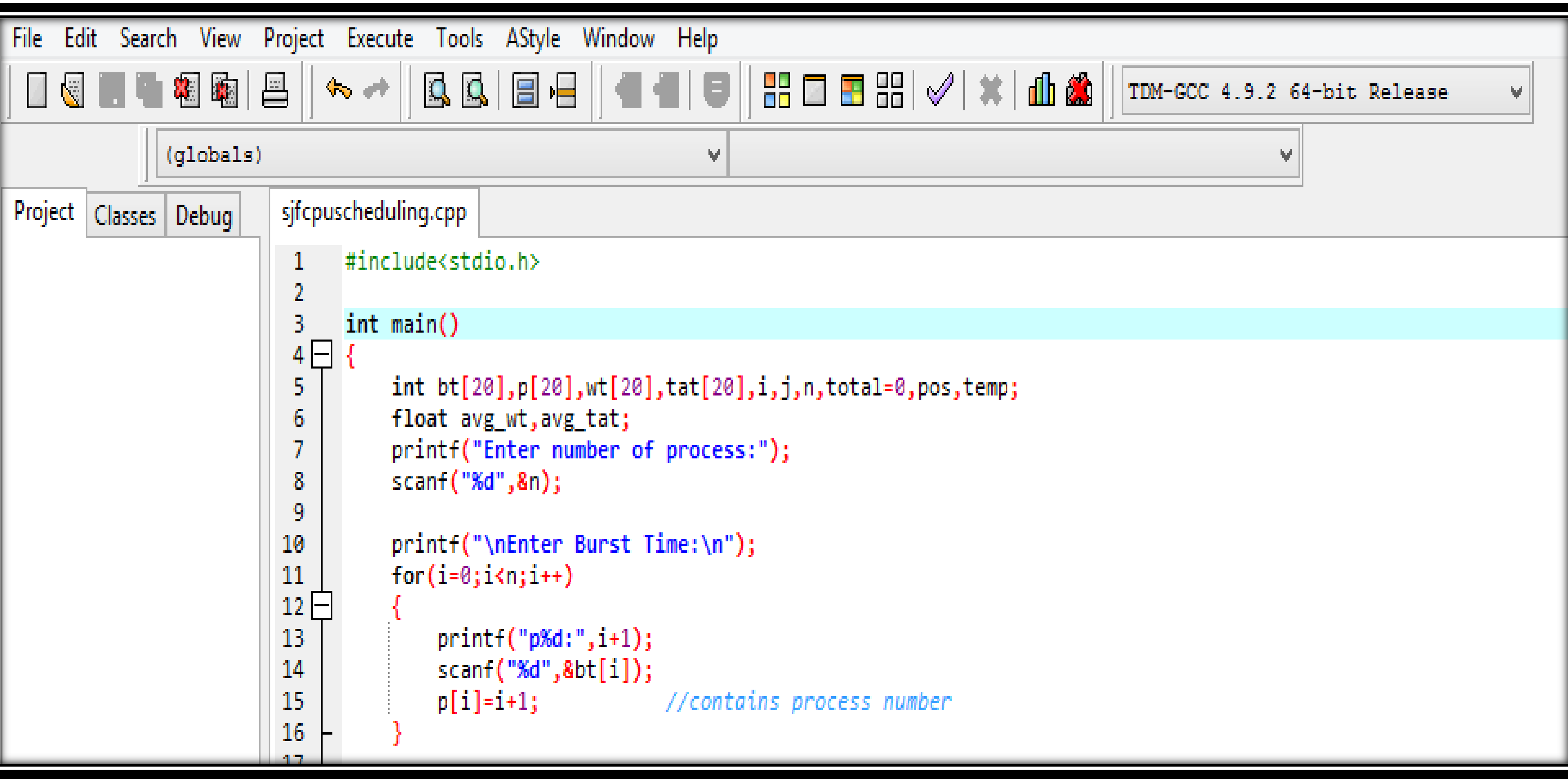
```
#include<stdio.h> int main()
{
        int n,w[100],tot[100],i,j,awt,atot; float avwt,avtot;
        struct
        {
                int p,bt; }sjf[10],temp;
        printf("Enter the number of Processes:"); scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                printf("Enter the Burst time for Process%d : ",i); scanf("%d",&sjf[i].bt);
                sjf[i].p=i;
        }
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                        if(sjf[j].bt>sjf[i].bt)
                        {
```

```
                        temp=sjf[i];
                        sjf[i]=sjf[j];
                        sjf[j]=temp;
                }
w[1]=0;
tot[1]=sjf[1].bt;
for(i=2;i<=n;i++) tot[i]=tot[i-1]+sjf[i].bt;
awt=0;
atot=0;
for(i=1;i<=n;i++)
{
        w[i]=tot[i]-sjf[i].bt; awt+=w[i]; atot+=tot[i];
}
avwt=(float)awt/n;
avtot=(float)atot/n;
printf("\n\nProcessId\tWaiting time\t TurnaroundTime");
 for(i=1;i<=n;i++)
printf("\n\t%d\t\t%d\t\t%d",sjf[i].p,w[i],tot[i]);
 printf("\n\nTotal Waiting Time :%d",awt);
printf("\n\nTotal Turnaround Time :%d",atot);
printf("\n\nAverage Waiting Time :%.2f",avwt);
printf("\n\nAverage Turnaround Time :%.2f",avtot); }
```

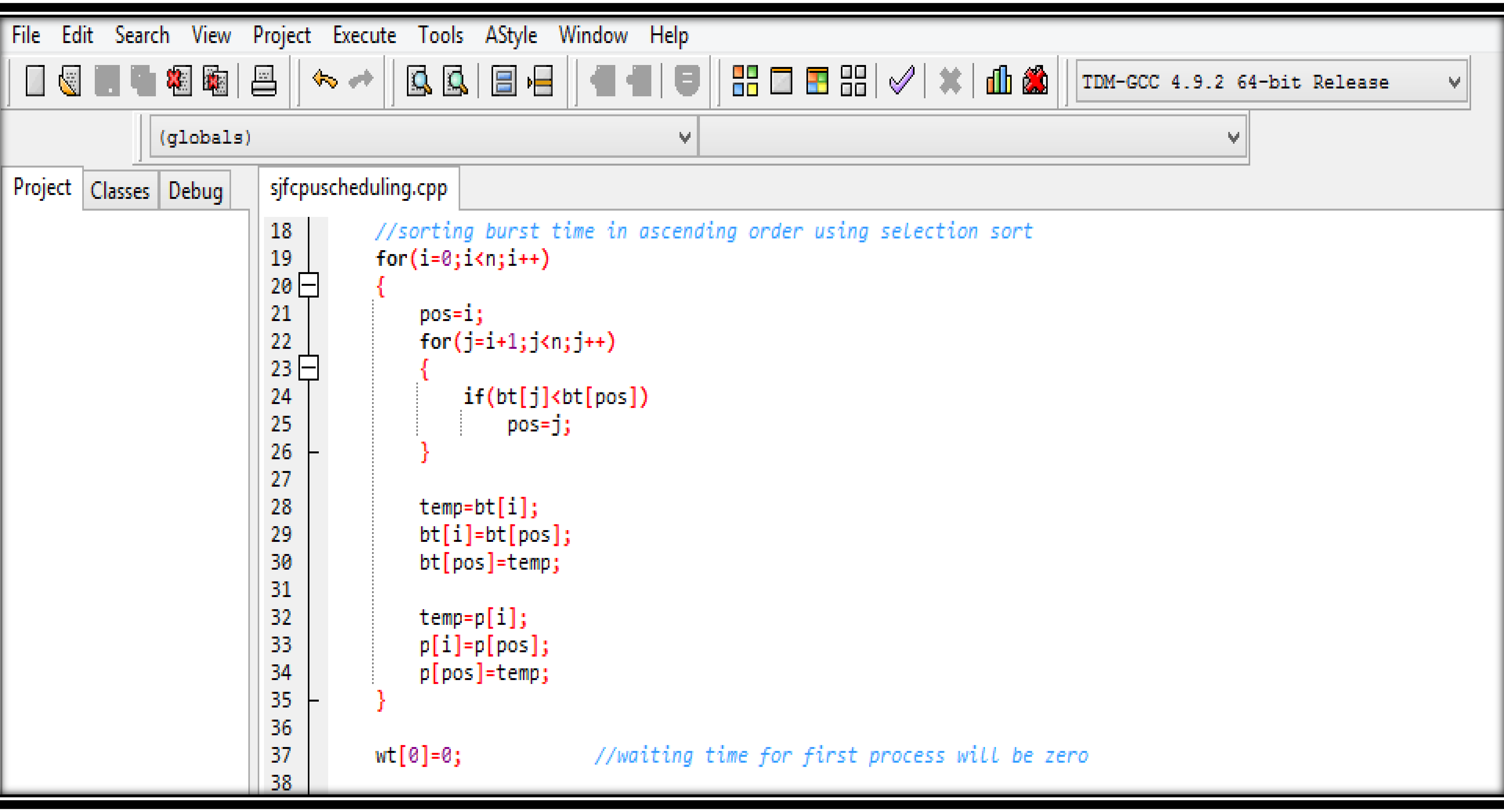
Output

OUTPUT:

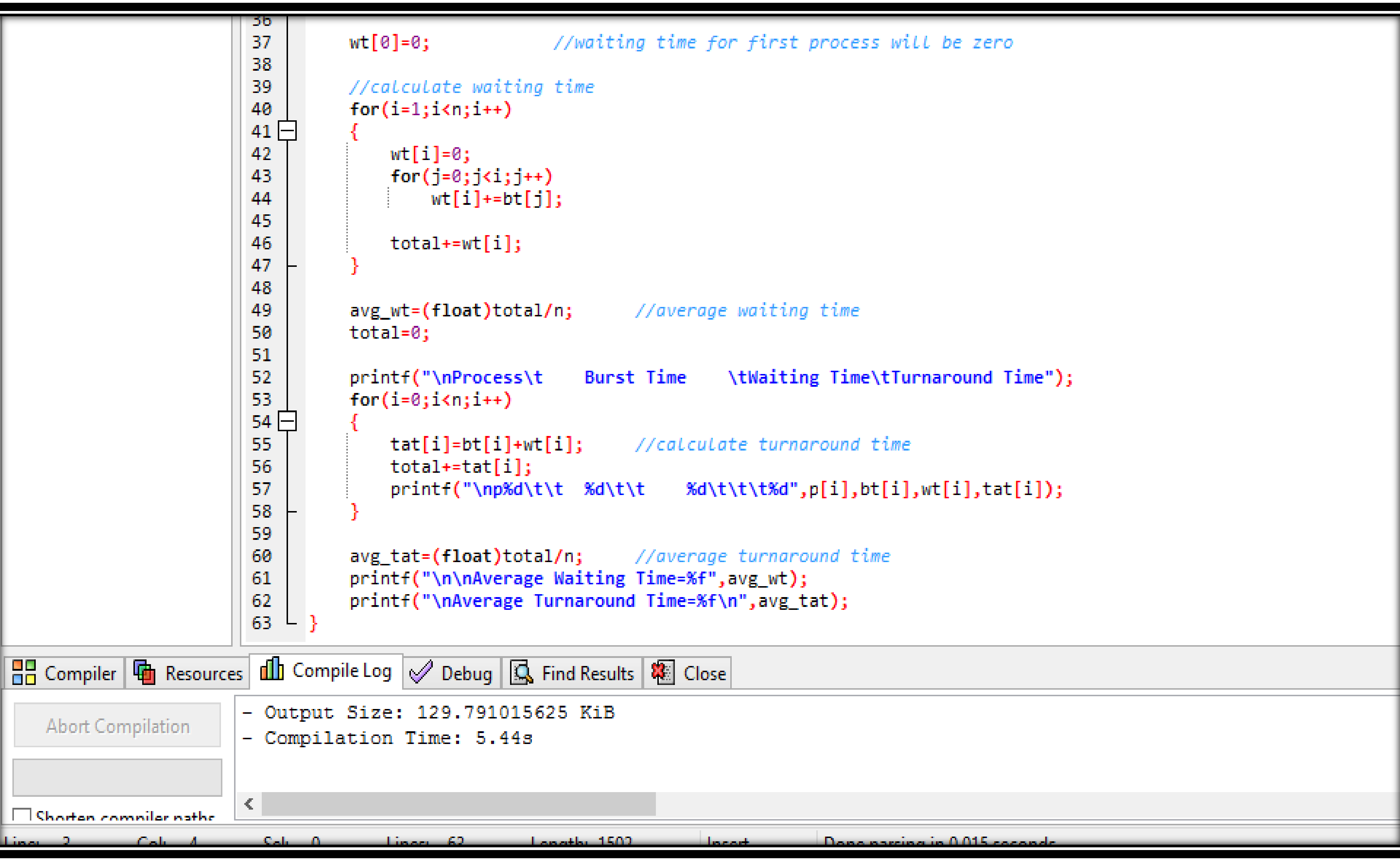
```
[cse6@localhost Pgm]$ cc prog9b.c
[cse6@localhost Pgm]$ ./a.out
Enter the number of Processes:3
Enter the Burst time for Process1 : 24
Enter the Burst time for Process2 : 5
Enter the Burst time for Process3 : 3
ProcessId                Waiting time                TurnaroundTime
    3                          0                            3
    2                          3                            8
    1                          8                            32
Total Waiting Time :11
Total Turnaround Time :43
Average Waiting Time :3.67
Average Turnaround Time :14.33
```


File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
sjfcpuscheduling.cpp

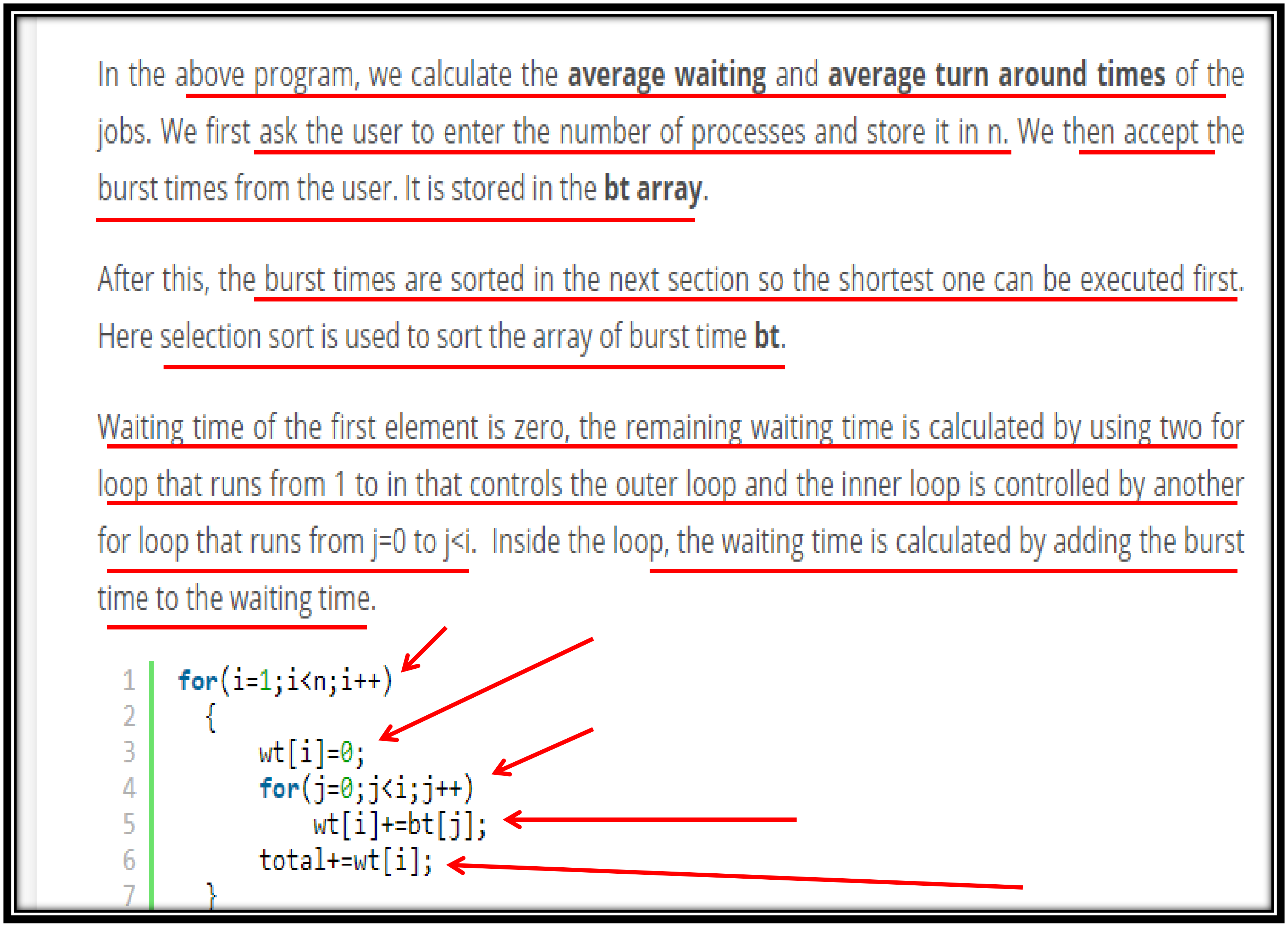
```
#include<stdio.h>

int main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter number of process:");
    scanf("%d",&n);

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


File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
sjfcpuscheduling.cpp

```
    //sorting burst time in ascending order using selection sort
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }

        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 will be zero

```


```c
    wt[0]=0;            //waiting time for first process will be 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=(float)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=(float)total/n;     //average turnaround time
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

```
- Output Size: 129.791015625 KiB
- Compilation Time: 5.44s
```



In the above program, we calculate the **average waiting** and **average turn around times** of the jobs. We first ask the user to enter the number of processes and store it in n. We then accept the burst times from the user. It is stored in the **bt array**.

After this, the burst times are sorted in the next section so the shortest one can be executed first. Here selection sort is used to sort the array of burst time **bt**.

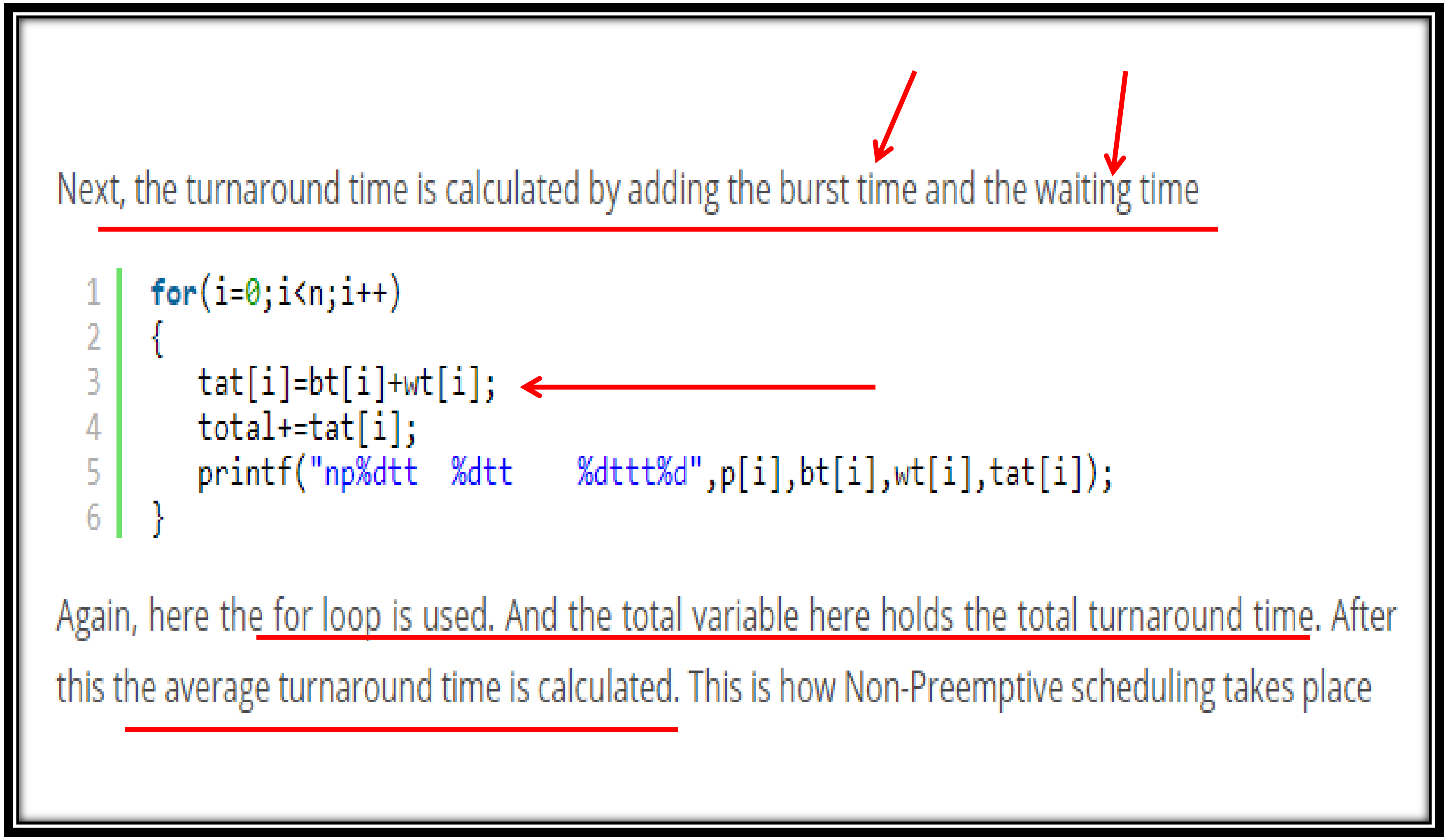
Waiting time of the first element is zero, the remaining waiting time is calculated by using two for loop that runs from 1 to in that controls the outer loop and the inner loop is controlled by another for loop that runs from j=0 to j<i. Inside the loop, the waiting time is calculated by adding the burst time to the waiting time.

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

Total is the addition of all the waiting time together. The average waiting time is calculated:

avg_wt=(float)total/n;

and it is printed.



Next, the turnaround time is calculated by adding the burst time and the waiting time

```
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];
    total+=tat[i];
    printf("np%dtt  %dtt    %dttt%d",p[i],bt[i],wt[i],tat[i]);
}
```

Again, here the for loop is used. And the total variable here holds the total turnaround time. After this the average turnaround time is calculated. This is how Non-Preemptive scheduling takes place

Task

Implement Pre-emptive SJF (Shortest Job First) CPU Scheduling Algorithm using C,C++. Also attach the snapshot of Code and output window.

Attach Task Output