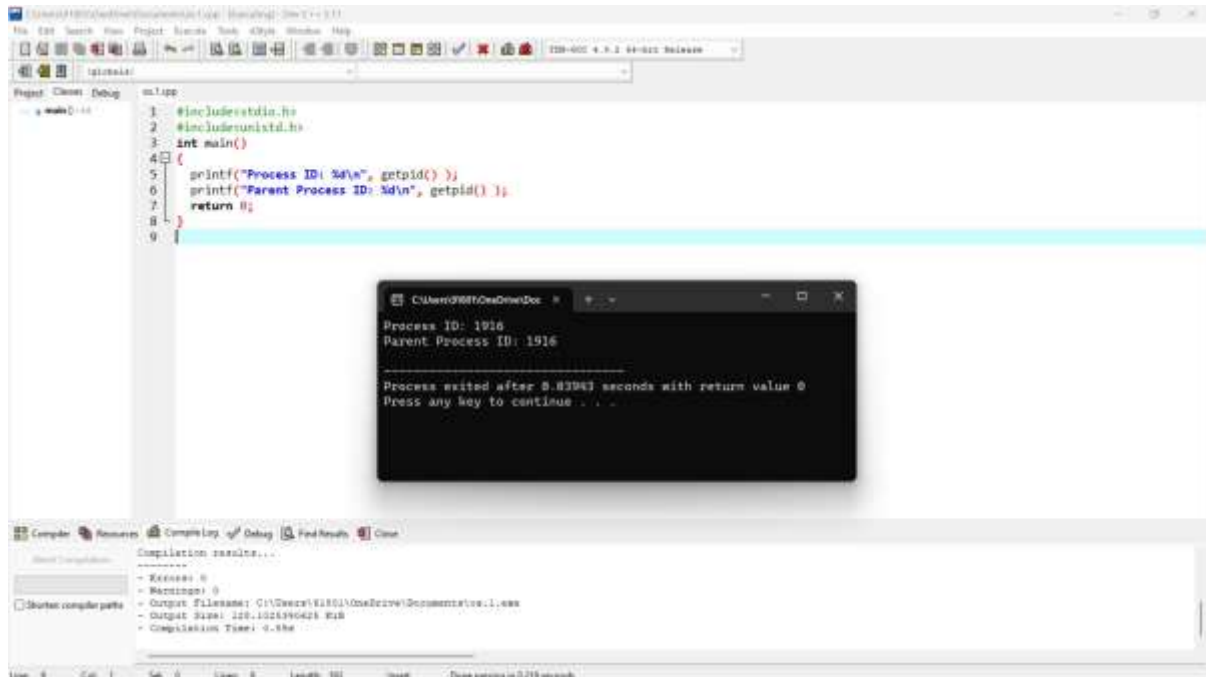


EXERCISE 1

1. Create a new process by invoking the appropriate system call. Get the process identifier of the currently running process and its respective parent using system calls and display the same using a C program.



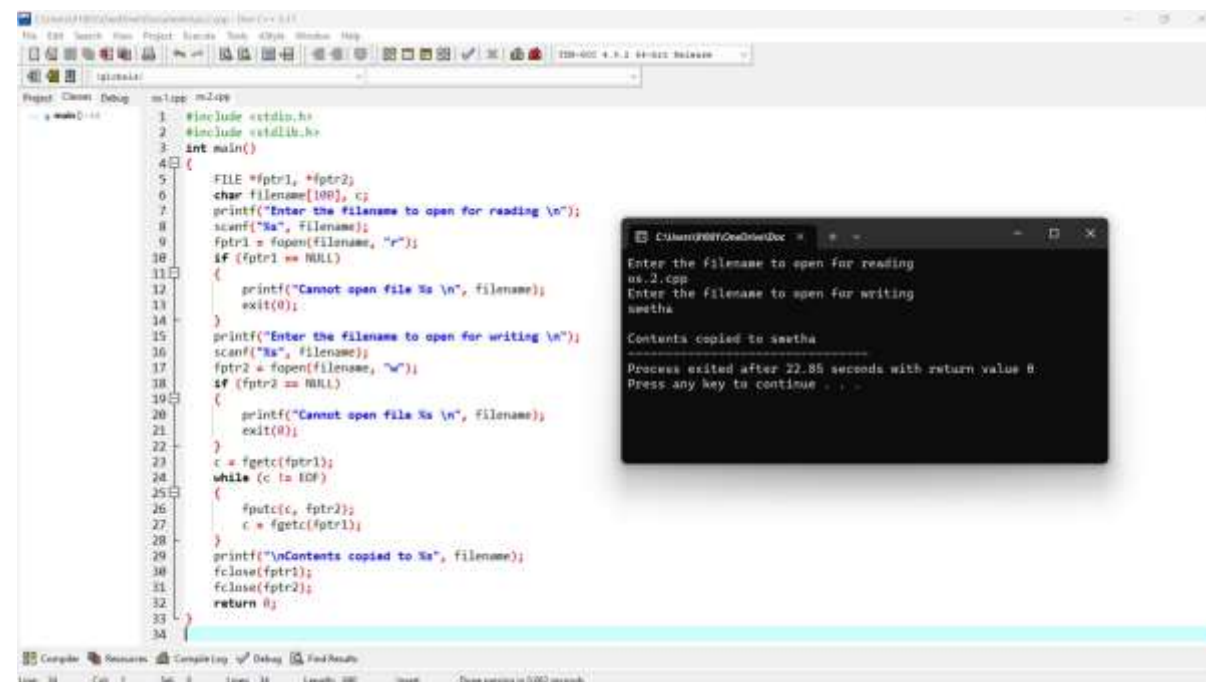
The screenshot shows a C program in a code editor. The code includes `<stdio.h>` and `<unistd.h>`. The `main` function prints the process ID and parent process ID using `getpid()` and `getppid()`. A terminal window shows the output: "Process ID: 1916", "Parent Process ID: 1916", and "Process exited after 0.03041 seconds with return value 0".

```
1 #include <stdio.h>
2 #include <unistd.h>
3 int main()
4 {
5     printf("Process ID: %d\n", getpid());
6     printf("Parent Process ID: %d\n", getppid());
7     return 0;
8 }
9
```

Process ID: 1916
Parent Process ID: 1916

Process exited after 0.03041 seconds with return value 0
Press any key to continue . . .

2. Identify the system calls to copy the content of one file to another and illustrate The same using a C program.



The screenshot shows a C program in a code editor. The code includes `<stdio.h>` and `<stdlib.h>`. The `main` function prompts the user for a filename to read and a filename to write. It uses `fopen` to open the files, `fgetc` to read characters, and `fputc` to write characters. A terminal window shows the input: "Enter the filename to open for reading: ex2.cpp", "Enter the filename to open for writing: swetha", and the output: "Contents copied to swetha".

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     FILE *fptr1, *fptr2;
6     char filename[100], c;
7     printf("Enter the filename to open for reading \n");
8     scanf("%s", filename);
9     fptr1 = fopen(filename, "r");
10    if (fptr1 == NULL)
11    {
12        printf("Cannot open file %s \n", filename);
13        exit(0);
14    }
15    printf("Enter the filename to open for writing \n");
16    scanf("%s", filename);
17    fptr2 = fopen(filename, "w");
18    if (fptr2 == NULL)
19    {
20        printf("Cannot open file %s \n", filename);
21        exit(0);
22    }
23    c = fgetc(fptr1);
24    while (c != EOF)
25    {
26        fputc(c, fptr2);
27        c = fgetc(fptr1);
28    }
29    printf("\nContents copied to %s", filename);
30    fclose(fptr1);
31    fclose(fptr2);
32    return 0;
33 }
34
```

Enter the filename to open for reading
ex2.cpp
Enter the filename to open for writing
swetha

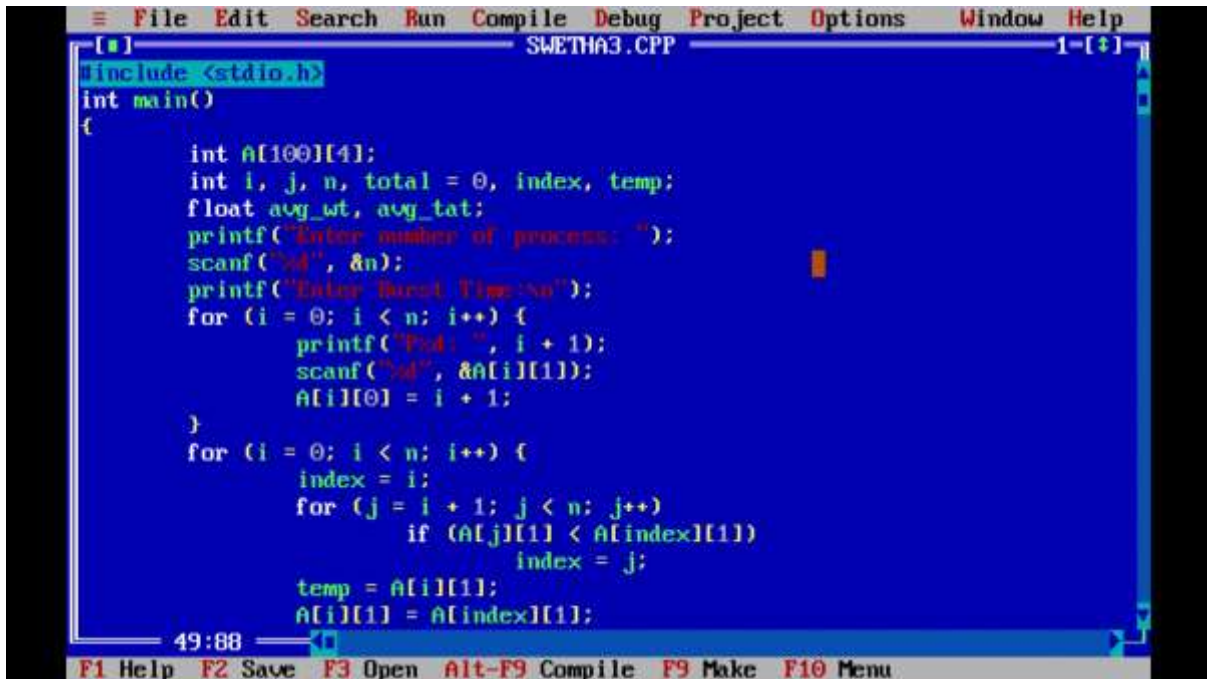
Contents copied to swetha

Process exited after 22.85 seconds with return value 0
Press any key to continue . . .

EXERCISE 1

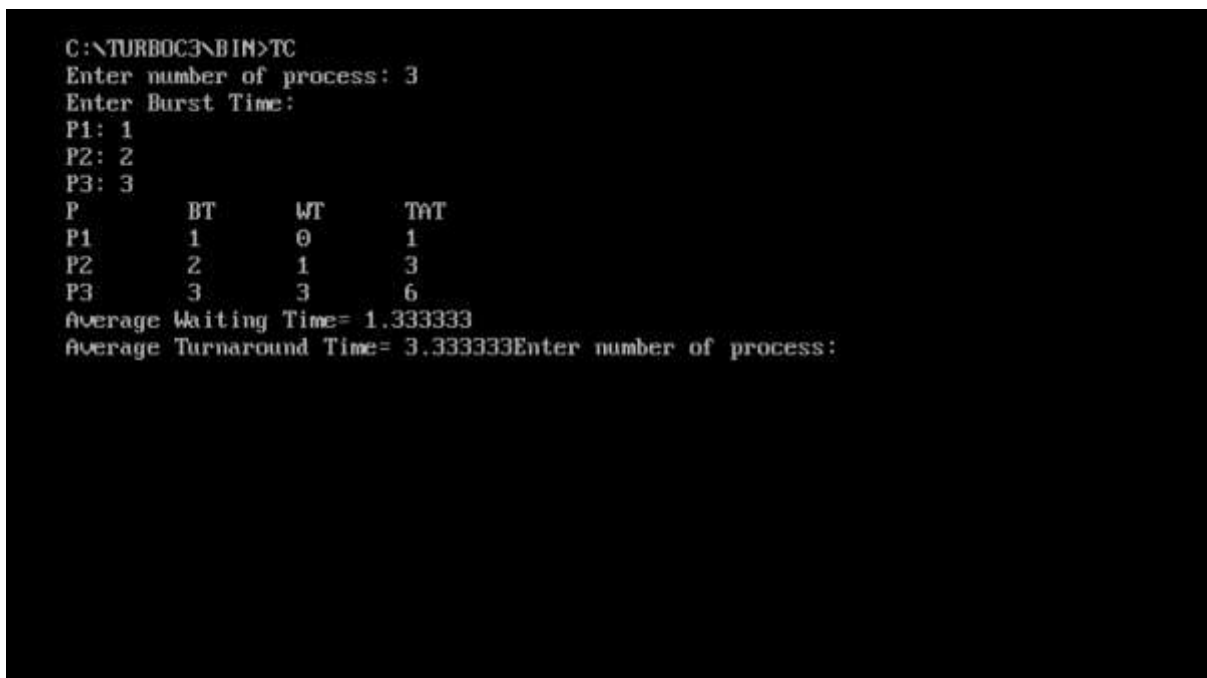
3. Design a CPU scheduling program with C using First Come First Served technique with the following considerations.

- All processes are activated at time 0.
- Assume that no process waits on I/O devices.



```
#include <stdio.h>
int main()
{
    int A[100][4];
    int i, j, n, total = 0, index, temp;
    float avg_wt, avg_tat;
    printf("Enter number of process: ");
    scanf("%d", &n);
    printf("Enter Burst Time:\n");
    for (i = 0; i < n; i++) {
        printf("P%d: ", i + 1);
        scanf("%d", &A[i][1]);
        A[i][0] = i + 1;
    }
    for (i = 0; i < n; i++) {
        index = i;
        for (j = i + 1; j < n; j++)
            if (A[j][1] < A[index][1])
                index = j;
        temp = A[i][1];
        A[i][1] = A[index][1];
    }
    49:88
    F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu
```

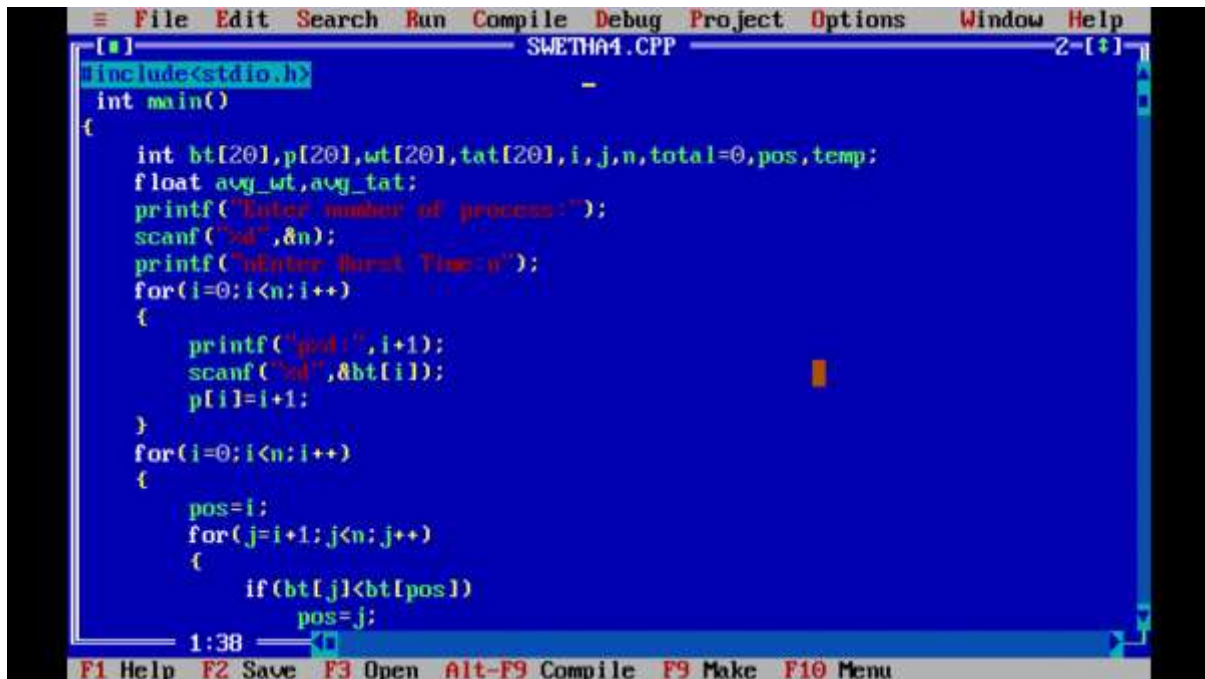
Output



```
C:\TURBOC3\BIN>TC
Enter number of process: 3
Enter Burst Time:
P1: 1
P2: 2
P3: 3
P    BT    WT    TAT
P1   1     0     1
P2   2     1     3
P3   3     3     6
Average Waiting Time= 1.333333
Average Turnaround Time= 3.333333Enter number of process:
```

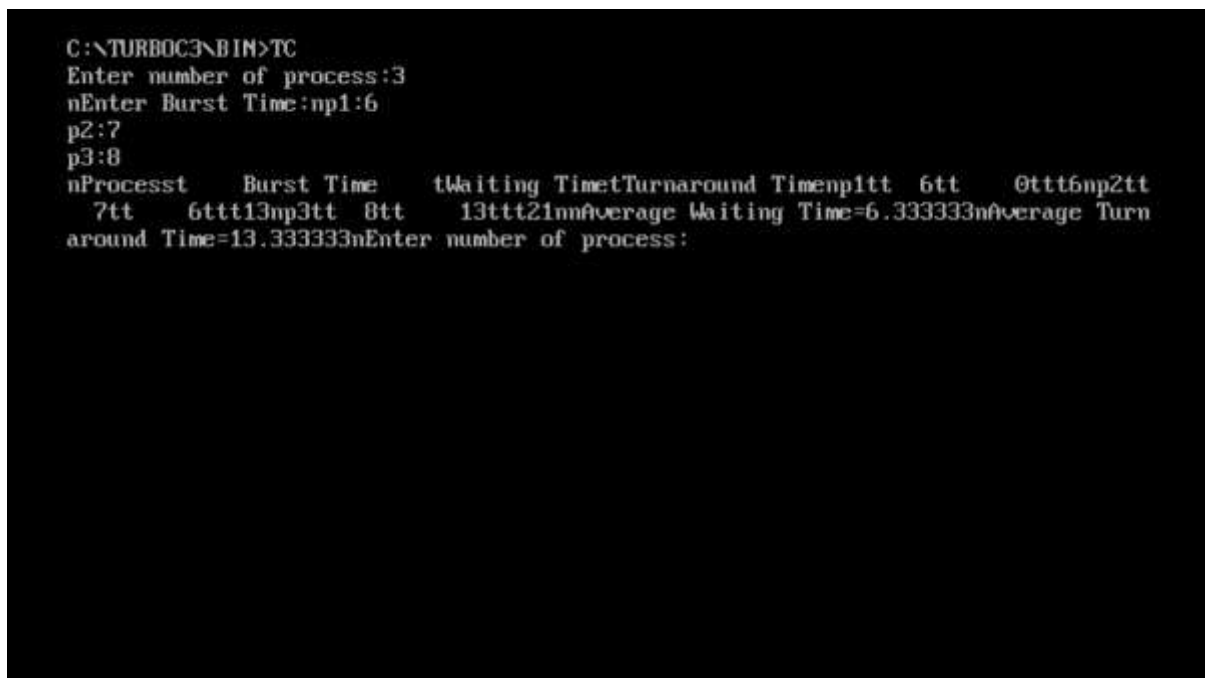
EXERCISE 1

4. Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next.



```
File Edit Search Run Compile Debug Project Options Window Help
SWETHA4.CPP 2-[+1]
#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("Enter Burst Time:n");
    for(i=0;i<n;i++)
    {
        printf("p%d:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1;
    }
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }
    }
}
```

Output



```
C:\TURBOC3\BIN>TC
Enter number of process:3
nEnter Burst Time:np1:6
p2:7
p3:8
nProcesst    Burst Time    tWaiting TimetTurnaround Timenp1tt    6tt    0ttt6np2tt
7tt    6ttt13np3tt    8tt    13ttt21nnAverage Waiting Time=6.333333nAverage Turn
around Time=13.333333nEnter number of process:
```

EXERCISE 1

5. Construct a scheduling program with C that selects the waiting process with the highest priority to execute next.

The screenshot shows a C program in a code editor and its execution output in a terminal window. The program implements a priority scheduling algorithm. It prompts the user to enter the priority, details of process C, details of process D, and the priority of process D. The output displays a table of process details and the average waiting and turnaround times.

```
#include <stdio.h>
2 struct priority_scheduling {
3     char process_name;
4     int burst_time;
5     int waiting_time;
6     int turn_around_time;
7     int priority;
8 }
9
10 int main() {
11     int number_of_processes;
12     int total = 0;
13     struct priority_scheduling temp_process;
14     int ASCII_number = 0;
15     int position;
16     float average_waiting_time;
17     float average_turn_around_time;
18     printf("Enter the total number of Processes: ");
19     scanf("%d", &number_of_processes);
20     struct priority_scheduling process[number_of_processes];
21     printf("Please enter the Burst Time and Priority of each process:\n");
22     for (int i = 0; i < number_of_processes; i++) {
23         process[i].process_name = (char) ASCII_number;
24         printf("Enter the details of the process %c: ", process[i].process_name);
25         printf("Enter the burst time: ");
26         scanf("%d", &process[i].burst_time);
27         printf("Enter the priority: ");
28         scanf("%d", &process[i].priority);
29         ASCII_number++;
30     }
31     for (int i = 0; i < number_of_processes; i++) {
32         position = i;
33         for (int j = i + 1; j < number_of_processes; j++) {
34             if (process[i].priority > process[j].priority)
35                 position = j;
36         }
37         temp_process = process[i];
38         process[i] = process[position];
39         process[position] = temp_process;
40     }
41     process[0].waiting_time = 0;
42     for (int i = 1; i < number_of_processes; i++) {
43         process[i].waiting_time = 0;
44         for (int j = 0; j < i; j++) {
45             process[i].waiting_time += process[j].burst_time;
46         }
47     }
48     total = 0;
49     average_waiting_time = 0;
50     average_turn_around_time = 0;
51     for (int i = 0; i < number_of_processes; i++) {
52         total += process[i].waiting_time;
53         average_waiting_time += process[i].waiting_time;
54         average_turn_around_time += process[i].turn_around_time;
55     }
56     printf("Average Waiting Time : %.750000\n");
57     printf("Average Turnaround Time : 8.250000\n");
58     printf("Process exited after 29.83 seconds with return value 0\n");
59     printf("Press any key to continue . . . ");
60 }
```

Process_name	Burst Time	Waiting Time	Turnaround Time
A	4	0	4
B	2	4	6
C	3	6	9
D	5	9	14

Average Waiting Time : 8.750000
Average Turnaround Time : 8.250000
Process exited after 29.83 seconds with return value 0
Press any key to continue . . .

6. Construct a C program to implement pre-emptive priority scheduling algorithm.

The screenshot shows a C program in a code editor and its execution output in a terminal window. The program implements a pre-emptive priority scheduling algorithm. It prompts the user to enter the number of processes, arrival time, burst time, and priority of each process. The output displays a table of process details and the average waiting and turnaround times.

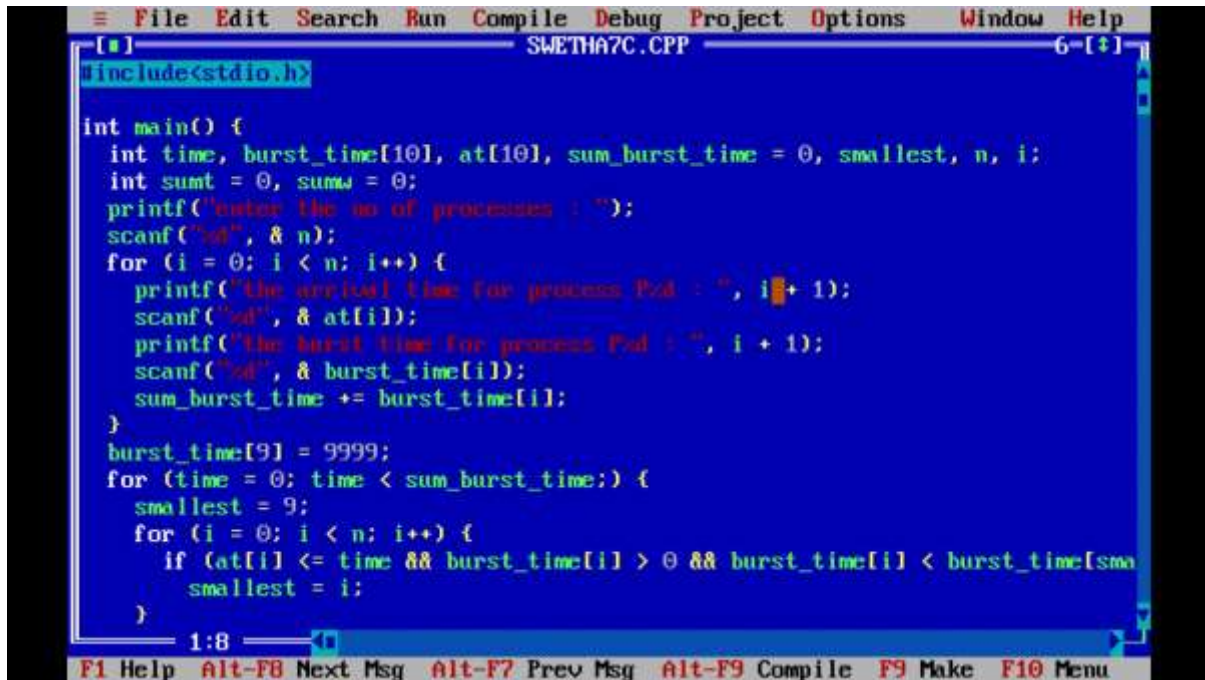
```
#include <stdio.h>
2 struct process {
3     int AT, BT, PT;
4 }
5
6 struct process a[10];
7
8 int main() {
9     int n, temp[10], t, count=0, short_p;
10    float total_AT=0, total_BT=0, Avg_AT, Avg_BT;
11    printf("Enter the number of the process:\n");
12    scanf("%d", &n);
13    printf("Enter the arrival time, burst time and priority of the process:\n");
14    printf("AT BT PT\n");
15    for (int i=0; i<n; i++) {
16        printf("Enter the arrival time, burst time and priority of the process:\n");
17        scanf("%d%d%d", &a[i].AT, &a[i].BT, &a[i].PT);
18    }
19    temp[0] = a[0].BT;
20    a[0].PT = 10000;
21    for (int i=0; count!=n; i++) {
22        short_p = 0;
23        for (int j=0; j<n; j++) {
24            if (a[short_p].PT > a[j].PT || a[j].AT < a[short_p].AT)
25                short_p = j;
26        }
27        count++;
28        total_AT += a[short_p].AT;
29        total_BT += a[short_p].BT;
30        a[short_p].PT = 0;
31    }
32    Avg_AT = total_AT / n;
33    Avg_BT = total_BT / n;
34    printf("Avg waiting time of the process is %.666667\n");
35    printf("Avg turn around time of the process is 5.666667\n");
36    printf("Process exited after 9.482 seconds with return value 0\n");
37    printf("Press any key to continue . . . ");
38 }
```

ID	AT	TAT
1	0	2
2	0	5
3	2	18

Avg waiting time of the process is 8.666667
Avg turn around time of the process is 5.666667
Process exited after 9.482 seconds with return value 0
Press any key to continue . . .

EXERCISE 1

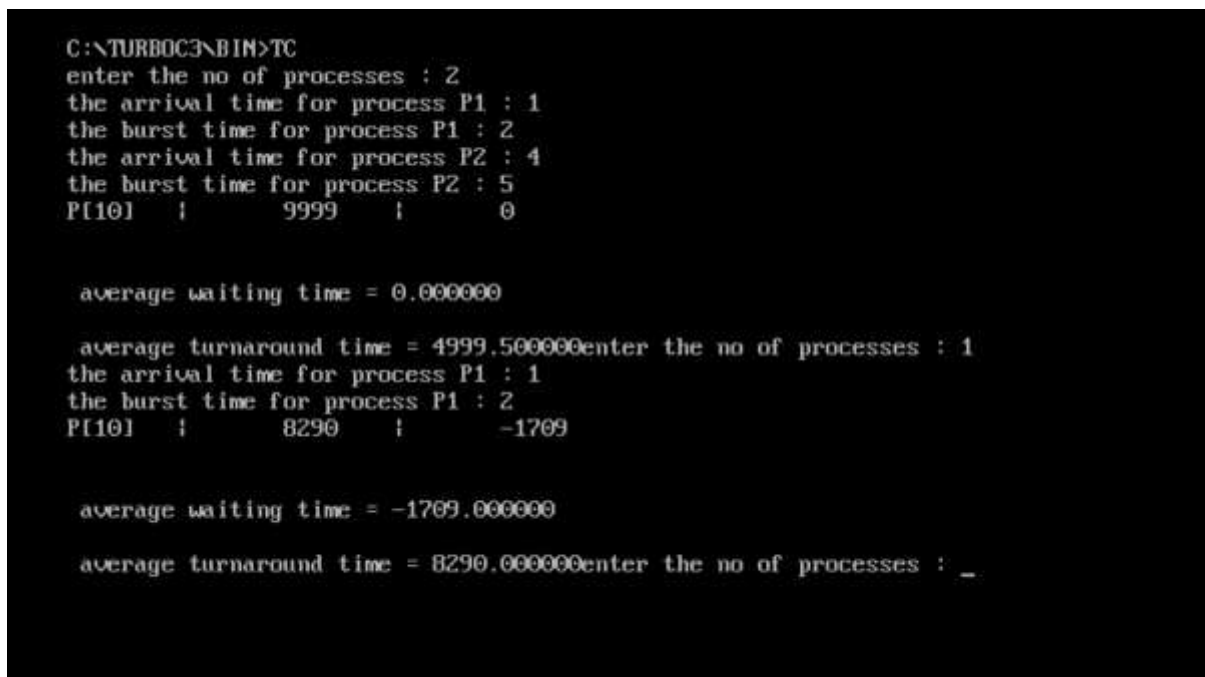
7. Construct a C program to implement non-preemptive SJF algorithm.



```
File Edit Search Run Compile Debug Project Options Window Help
[.] SWETHA7C.CPP 6-[.]
#include<stdio.h>

int main() {
    int time, burst_time[10], at[10], sum_burst_time = 0, smallest, n, i;
    int sumt = 0, sumw = 0;
    printf("enter the no of processes : ");
    scanf("%d", & n);
    for (i = 0; i < n; i++) {
        printf("the arrival time for process P%d : ", i + 1);
        scanf("%d", & at[i]);
        printf("the burst time for process P%d : ", i + 1);
        scanf("%d", & burst_time[i]);
        sum_burst_time += burst_time[i];
    }
    burst_time[9] = 9999;
    for (time = 0; time < sum_burst_time;) {
        smallest = 9;
        for (i = 0; i < n; i++) {
            if (at[i] <= time && burst_time[i] > 0 && burst_time[i] < burst_time[smallest])
                smallest = i;
        }
    }
}
```

Output



```
C:\TURBOC3\BIN>TC
enter the no of processes : 2
the arrival time for process P1 : 1
the burst time for process P1 : 2
the arrival time for process P2 : 4
the burst time for process P2 : 5
P[10] | 9999 | 0

average waiting time = 0.000000

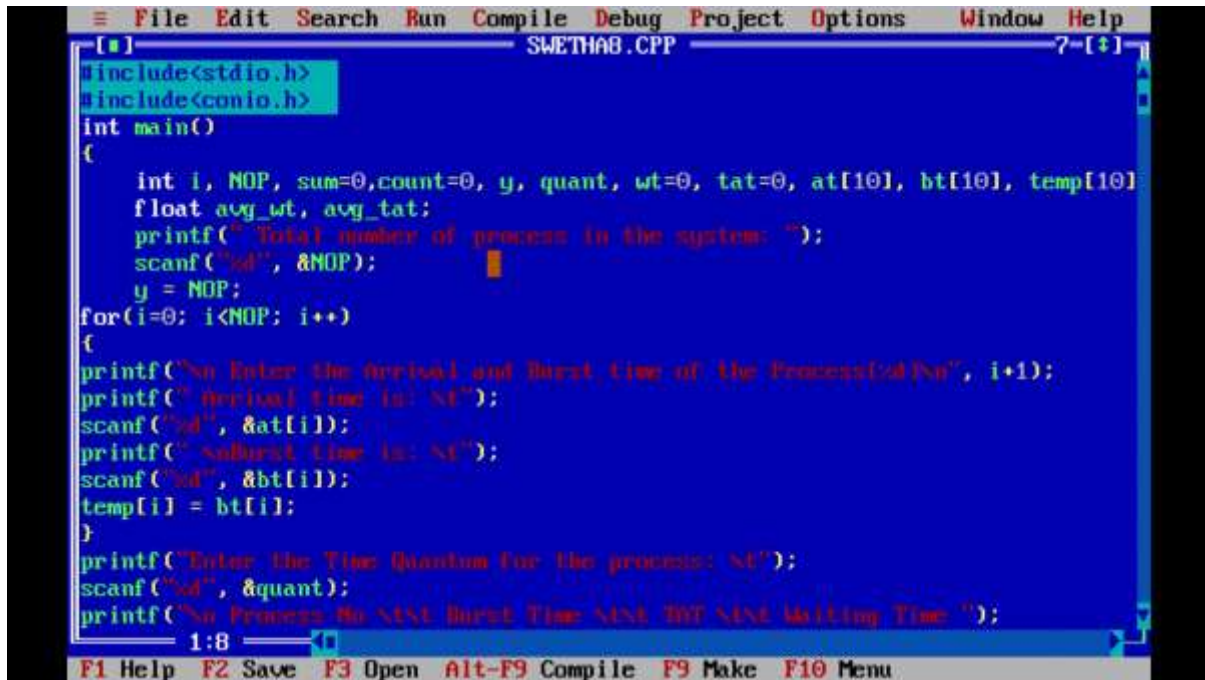
average turnaround time = 4999.500000enter the no of processes : 1
the arrival time for process P1 : 1
the burst time for process P1 : 2
P[10] | 8290 | -1709

average waiting time = -1709.000000

average turnaround time = 8290.000000enter the no of processes : 5
```

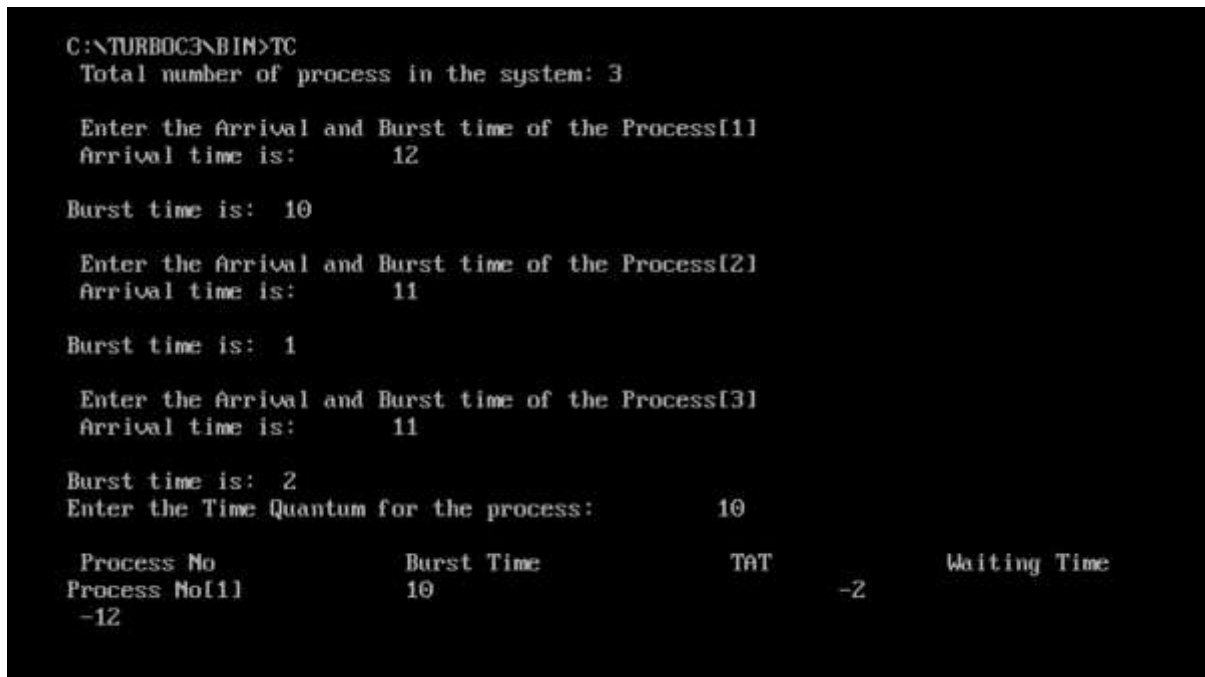

EXERCISE 1

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



```
File Edit Search Run Compile Debug Project Options Window Help
SWETHAB.CPP 7-11
#include<stdio.h>
#include<conio.h>
int main()
{
    int i, NOP, sum=0, count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
    float avg_wt, avg_tat;
    printf("Total number of process in the system: ");
    scanf("%d", &NOP);
    y = NOP;
    for(i=0; i<NOP; i++)
    {
        printf("\nEnter the Arrival and Burst time of the Process[%d]", i+1);
        printf("Arrival time is: ");
        scanf("%d", &at[i]);
        printf("Burst time is: ");
        scanf("%d", &bt[i]);
        temp[i] = bt[i];
    }
    printf("Enter the Time Quantum for the process: ");
    scanf("%d", &quant);
    printf("\nProcess No\tArrival Time\tBurst Time\tWaiting Time:");
    1:8
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu
```

Output



```
C:\TURBOC3\BIN>TC
Total number of process in the system: 3

Enter the Arrival and Burst time of the Process[1]
Arrival time is: 12

Burst time is: 10

Enter the Arrival and Burst time of the Process[2]
Arrival time is: 11

Burst time is: 1

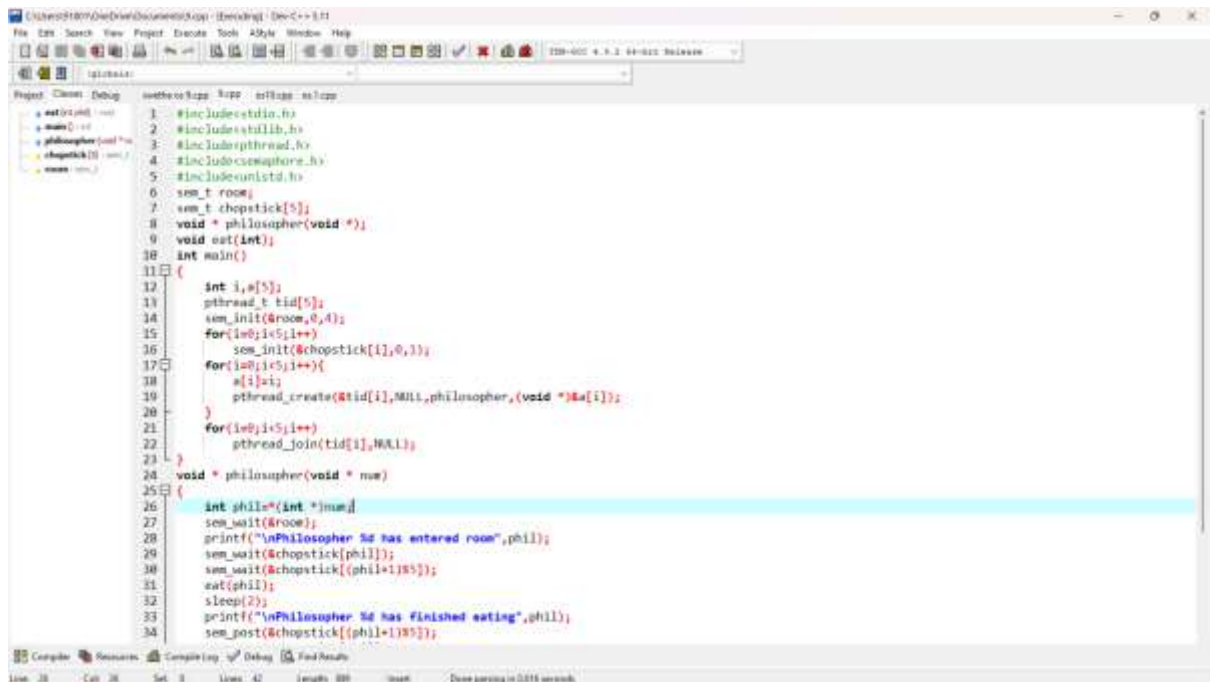
Enter the Arrival and Burst time of the Process[3]
Arrival time is: 11

Burst time is: 2
Enter the Time Quantum for the process: 10

Process No      Burst Time      TAT      Waiting Time
Process No[1]   10              -2
-12
```

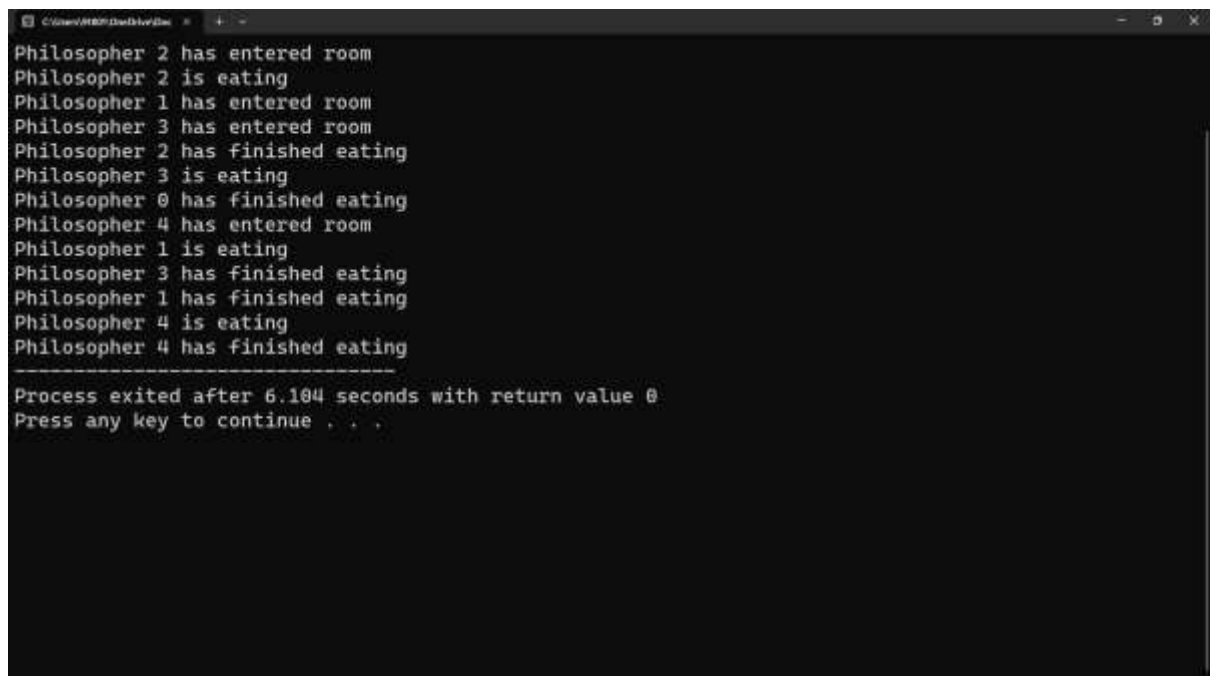
EXERCISE 1

9. Design a C program to simulate the concept of Dining-Philosophers problem



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <semaphore.h>
5 #include <unistd.h>
6 sem_t room;
7 sem_t chopstick[5];
8 void * philosopher(void *);
9 void eat(int);
10
11 int main()
12 {
13     int i, n[5];
14     pthread_t tid[5];
15     sem_init(&room, 0, 4);
16     for(i=0; i<5; i++)
17         sem_init(&chopstick[i], 0, 1);
18     for(i=0; i<5; i++){
19         n[i]=i;
20         pthread_create(&tid[i], NULL, philosopher, (void *)n[i]);
21     }
22     for(i=0; i<5; i++)
23         pthread_join(tid[i], NULL);
24 }
25 void * philosopher(void * num)
26 {
27     int phil=(int *)num;
28     sem_wait(&room);
29     printf("\nPhilosopher %d has entered room", phil);
30     sem_wait(&chopstick[phil]);
31     sem_wait(&chopstick[(phil+1)%5]);
32     eat(phil);
33     sleep(2);
34     printf("\nPhilosopher %d has finished eating", phil);
35     sem_post(&chopstick[(phil+1)%5]);
36 }
```

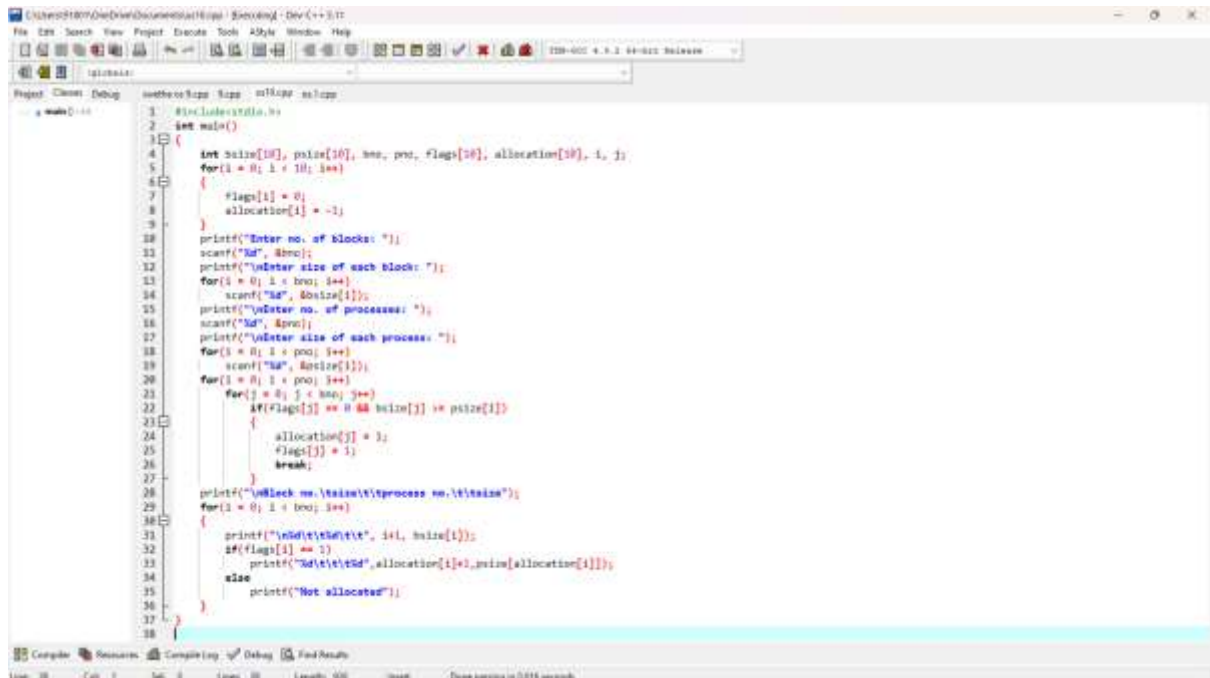
Output



```
Philosopher 2 has entered room
Philosopher 2 is eating
Philosopher 1 has entered room
Philosopher 3 has entered room
Philosopher 2 has finished eating
Philosopher 3 is eating
Philosopher 0 has finished eating
Philosopher 4 has entered room
Philosopher 1 is eating
Philosopher 3 has finished eating
Philosopher 1 has finished eating
Philosopher 4 is eating
Philosopher 4 has finished eating
-----
Process exited after 6.104 seconds with return value 0
Press any key to continue . . .
```

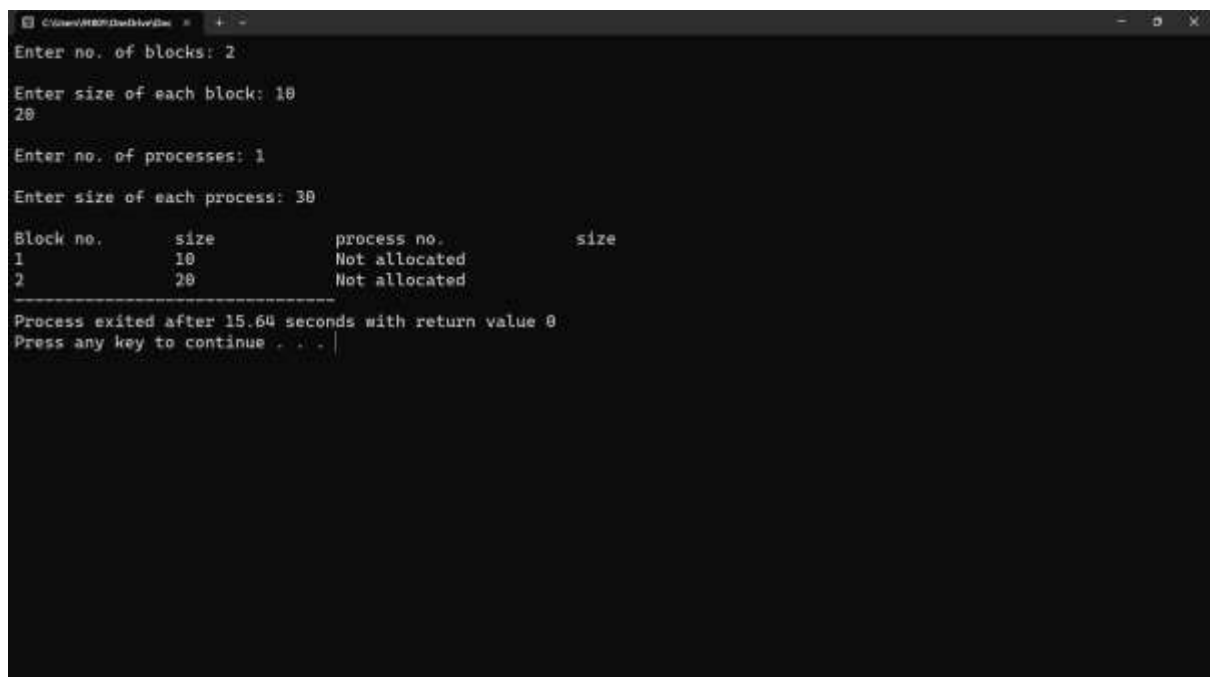
EXERCISE 1

10. Construct a C program for implementation of memory allocation using first fit strategy.



```
1 #include <stdio.h>
2 int main()
3 {
4     int size[10], psize[10], bno, pno, flags[20], allocation[30], i, j;
5     for(i = 0; i < 10; i++)
6     {
7         flags[i] = 0;
8         allocation[i] = -1;
9     }
10    printf("Enter no. of blocks: ");
11    scanf("%d", &bno);
12    printf("Enter size of each block: ");
13    for(i = 0; i < bno; i++)
14        scanf("%d", &size[i]);
15    printf("Enter no. of processes: ");
16    scanf("%d", &pno);
17    printf("Enter size of each process: ");
18    for(i = 0; i < pno; i++)
19        scanf("%d", &psize[i]);
20    for(i = 0; i < pno; i++)
21        for(j = 0; j < bno; j++)
22            if(flags[j] == 0 && size[j] >= psize[i])
23            {
24                allocation[i] = j;
25                flags[j] = 1;
26                break;
27            }
28    printf("\nBlock no.\tsize\tprocess no.\tsize");
29    for(i = 0; i < bno; i++)
30    {
31        printf("\n%d\t%d\t\t", i+1, size[i]);
32        if(flags[i] == 1)
33            printf("\t\t\t\t\t", allocation[i]+1, psize[allocation[i]]);
34        else
35            printf("\t\t\t\t\t", "Not allocated");
36    }
37}
```

Output



```
Enter no. of blocks: 2
Enter size of each block: 10
20
Enter no. of processes: 1
Enter size of each process: 30
Block no.      size      process no.      size
1              10      Not allocated
2              20      Not allocated
-----
Process exited after 15.64 seconds with return value 0
Press any key to continue . . .
```