UNIVERSITY OF DELHI

BHASKARACHARYA COLLEGE OF APPLIED SCIENCES

BSC (HONS) COMPUTER SCIENCE

SEMESTER - 3

OPERATING SYSTEM

ANAND KUMAR MISHRA

2102006

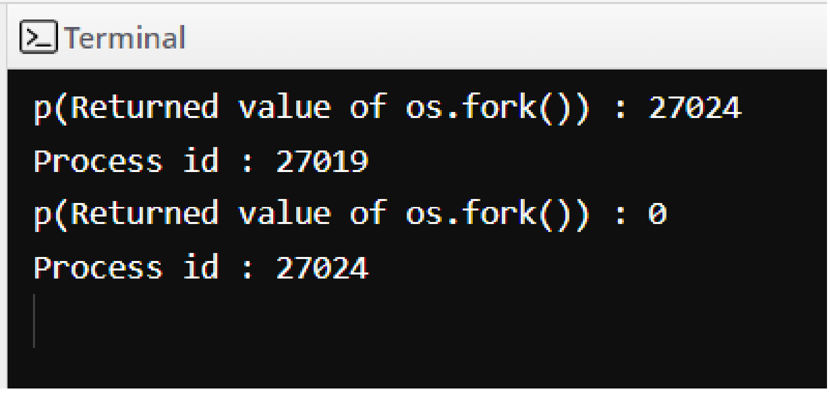
Question 1 :

Write a program (using fork() and/or exec() commands) where parent and child execute:
a) same program, same code.
b) same program, different code.
c) before terminating, the parent waits for the child to finish its task.

Solution 1(a) :

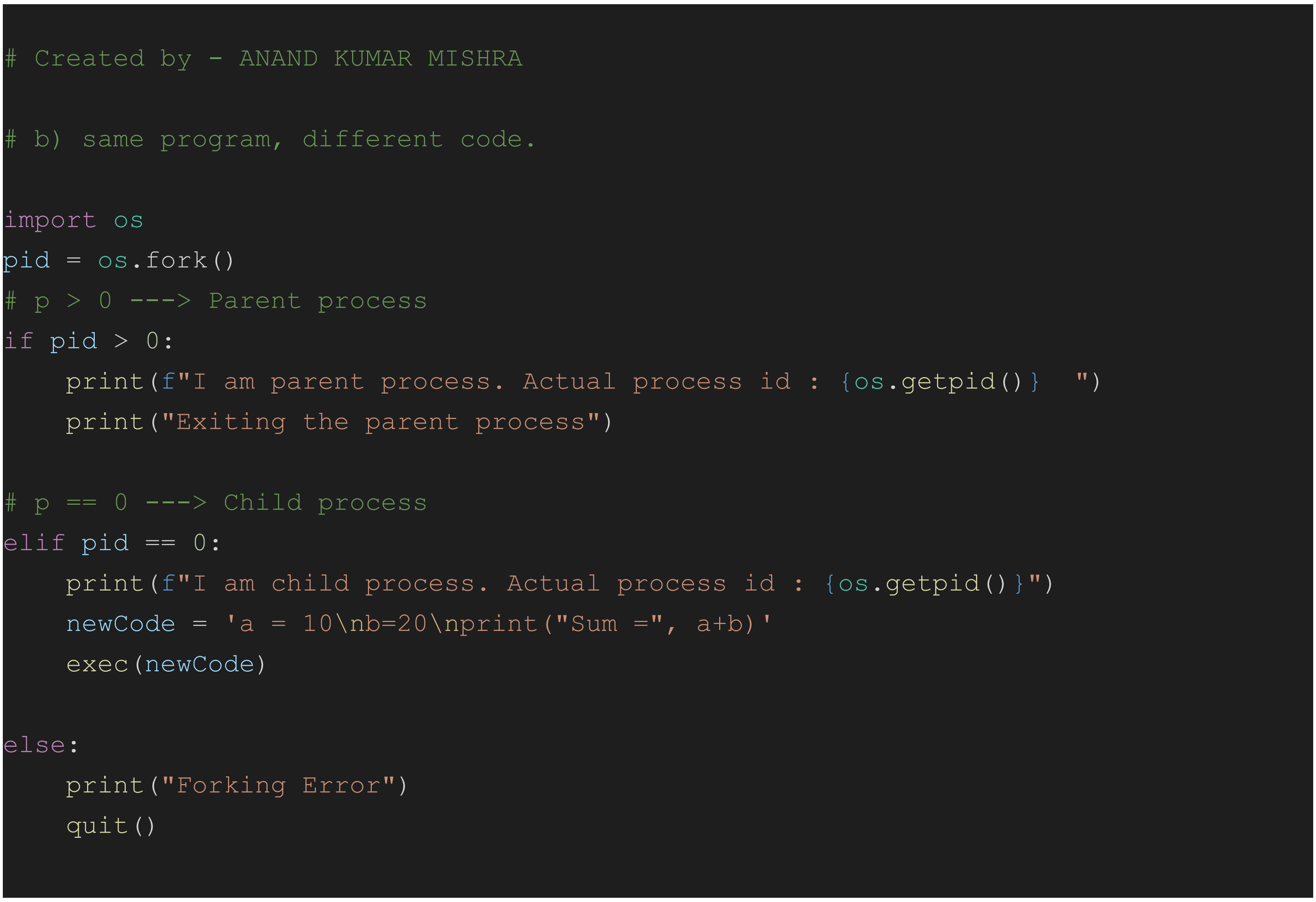
```python
"""
os.fork() is used to create child process Returns 0 in child process and child's process id in parent process

"""
# This code won't work on Windows, use online compiler to execute

# Created by - ANAND KUMAR MISHRA

# a) same program, same code.

import os
pid = os.fork()

if pid < 0:
    print("Fork failed")
    quit()

print(f"p(Returned value of os.fork()) : {pid}")
print(f"Process id : {os.getpid()}")
# If returned value of os.fork() is 0, child process has been executed
# Returned value of os.fork() in parent process will match os.getpid() of child process
```

Output 1(a) :



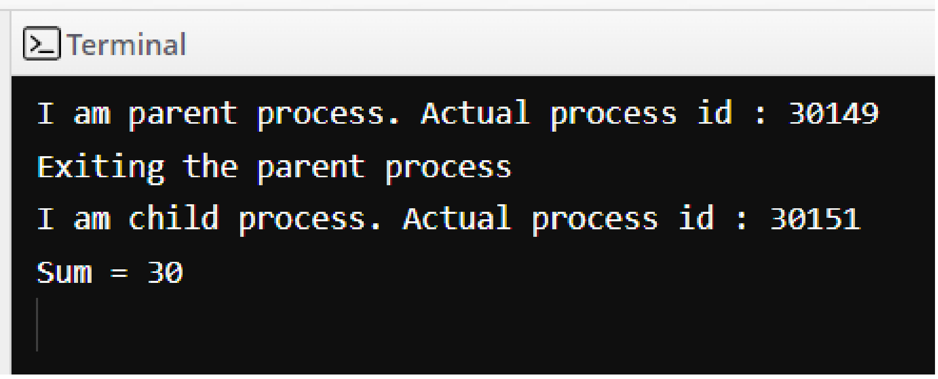
```
p(Returned value of os.fork()) : 27024
Process id : 27019
p(Returned value of os.fork()) : 0
Process id : 27024
```

Solution 1(b) :



```
# Created by - ANAND KUMAR MISHRA

# b) same program, different code.

import os
pid = os.fork()
# p > 0 ---> Parent process
if pid > 0:
    print(f"I am parent process. Actual process id : {os.getpid()}  ")
    print("Exiting the parent process")

# p == 0 ---> Child process
elif pid == 0:
    print(f"I am child process. Actual process id : {os.getpid()}")
    newCode = 'a = 10\nb=20\nprint("Sum =", a+b)'
    exec(newCode)

else:
    print("Forking Error")
    quit()
```

Output 1(b) :

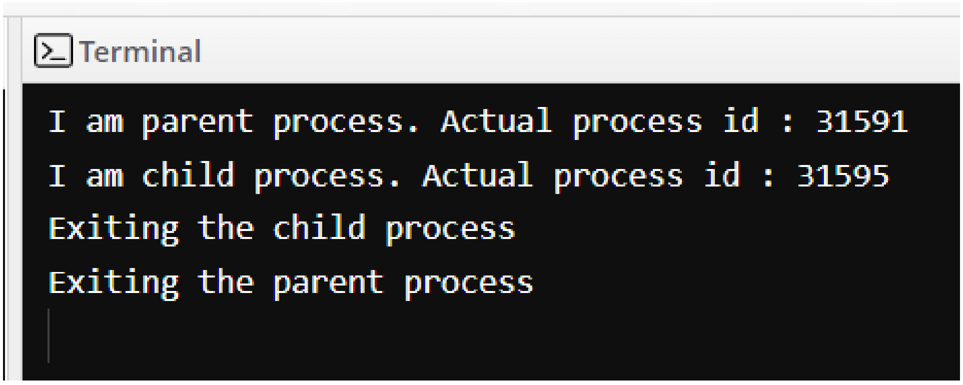


```
I am parent process. Actual process id : 30149
Exiting the parent process
I am child process. Actual process id : 30151
Sum = 30
```

Solution 1(c) :

```
# Created by - ANAND KUMAR MISHRA

# c) before terminating, the parent waits for the child to finish its task.

import os
pid = os.fork()
# p > 0 ---> Parent process
if pid > 0:
    print(f"I am parent process. Actual process id : {os.getpid()}  ")
    os.waitpid(-1, 0)
    print("Exiting the parent process")

# p == 0 ---> Child process
elif pid == 0:
    print(f"I am child process. Actual process id : {os.getpid()}")
    print("Exiting the child process")

else:
    print("Forking Error")
    quit()
```

Output 1(c) :



```
I am parent process. Actual process id : 31591
I am child process. Actual process id : 31595
Exiting the child process
Exiting the parent process
```

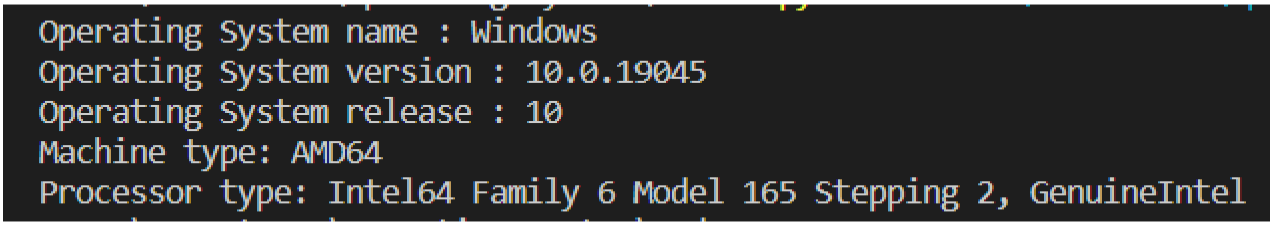
Question 2 :

Write a program to report behaviour of Linux kernel including kernel version, CPU type and model. (CPU information)

Solution 2 :

```
# Created by - ANAND KUMAR MISHRA

import platform

print(f"Operating System name : {platform.system()}")
print(f"Operating System version : {platform.version()}")
print(f"Operating System release : {platform.release()}")
print(f"Machine type: {platform.machine()}")
print(f"Processor type: {platform.processor()}")
```

Output 2 :



```
Operating System name : Windows
Operating System version : 10.0.19045
Operating System release : 10
Machine type: AMD64
Processor type: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```

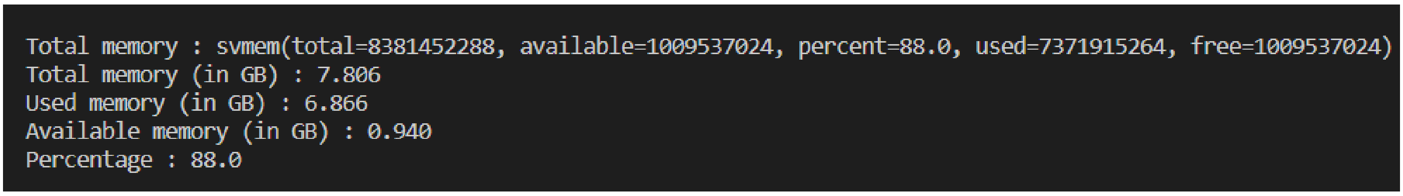
Question 3 :

Write a program to report behaviour of Linux kernel including information on 19 configured memory, amount of free and used memory. (memory information)

Solution 3 :

```
# Created by - ANAND KUMAR MISHRA

import psutil
print(f"Total memory : {psutil.virtual_memory()}")
print(f"Total memory (in GB) : {psutil.virtual_memory().total / (1024.0 ** 3):.3f}")
print(f"Used memory (in GB) : {psutil.virtual_memory().used / (1024.0 ** 3):.3f}")
print(f"Available memory (in GB) : {psutil.virtual_memory().available / (1024.0 ** 3):.3f}")
print(f"Percentage : {psutil.virtual_memory().percent}")
```

Output 3 :



```
Total memory : svmem(total=8381452288, available=1009537024, percent=88.0, used=7371915264, free=1009537024)
Total memory (in GB) : 7.806
Used memory (in GB) : 6.866
Available memory (in GB) : 0.940
Percentage : 88.0
```

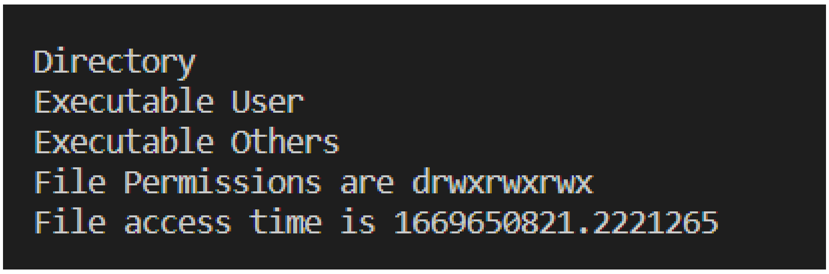
Question 4 :

Write a program to print file details including owner access permissions, file access time, where file name is given as argument

Solution 4 :

```
# Created by - ANAND KUMAR MISHRA

import os
from stat import *

statinfo = os.stat('Downloads')

mode = statinfo.st_mode

if S_ISDIR(mode):
    print("Directory")

elif S_ISREG(mode):
    print("Regular File")

if (mode & S_IXUSR):
    print("Executable User")
elif (mode & S_IWUSR):
    print("Writable User")
elif (mode & S_IRUSR):
    print("Readable User")

if (mode & S_IXOTH):
    print("Executable Others")
elif (mode & S_IWOTH):
    print("Writable Others")
elif (mode & S_IROTH):
    print("Readable Others")

filePerm = filemode(mode)

print(f"File Permissions are {filePerm}")

print(f"File access time is {statinfo.st_atime}")
```

Output 4 :

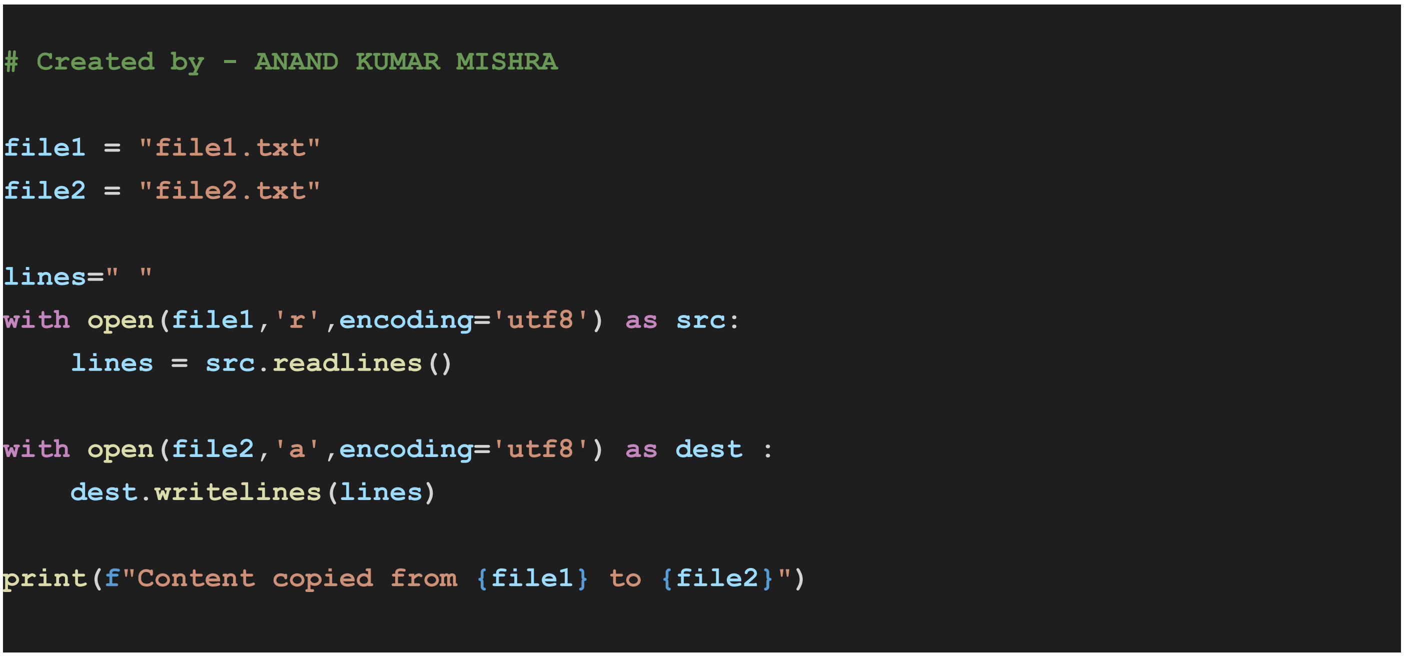


```
Directory
Executable User
Executable Others
File Permissions are drwxrwxrwx
File access time is 1669650821.2221265
```

Question 5 :

Write a program to copy files using system calls.

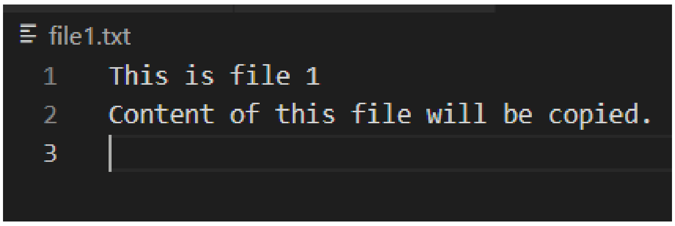
Solution 5 :



```python
# Created by - ANAND KUMAR MISHRA

file1 = "file1.txt"
file2 = "file2.txt"

lines=" "
with open(file1,'r',encoding='utf8') as src:
    lines = src.readlines()

with open(file2,'a',encoding='utf8') as dest :
    dest.writelines(lines)

print(f"Content copied from {file1} to {file2}")
```

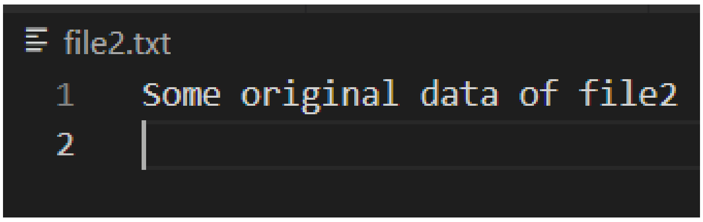
Output 5 :

file1.txt (before) executing the code :

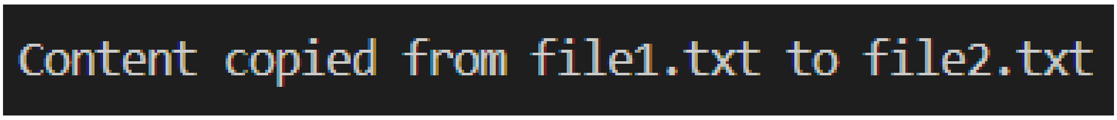


```
This is file 1
Content of this file will be copied.

```

file2.txt (before) executing the code :

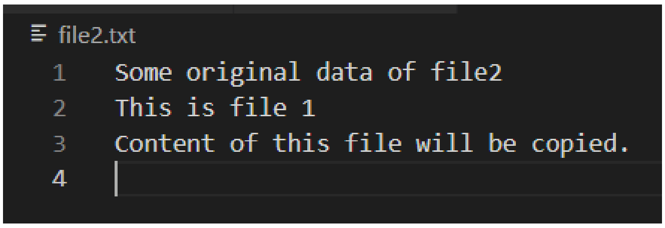


```
Some original data of file2

```



```
Content copied from file1.txt to file2.txt
```

file2.txt (after) executing the code :



```
Some original data of file2
This is file 1
Content of this file will be copied.

```

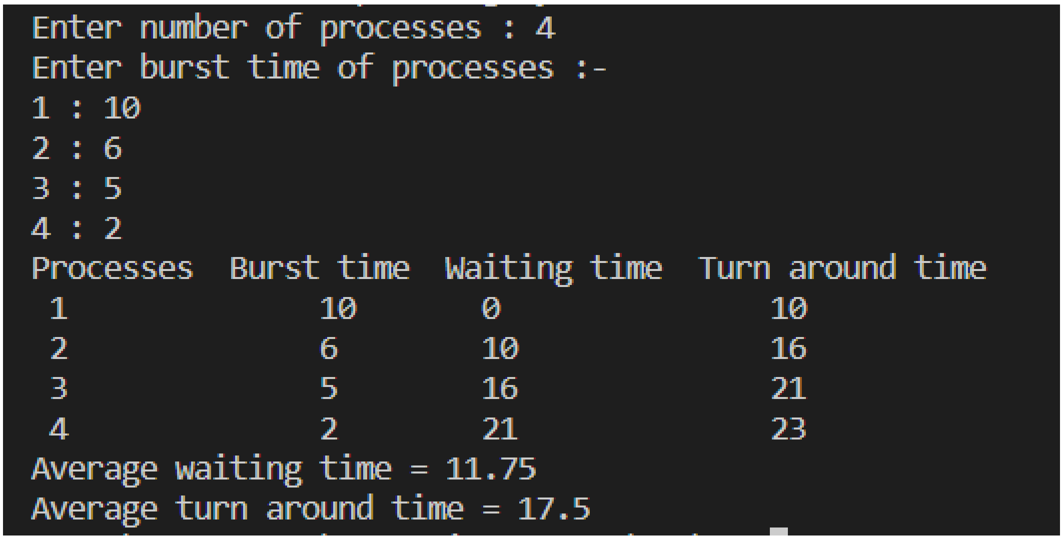
Question 6 :

Write a program to implement FCFS scheduling algorithm.

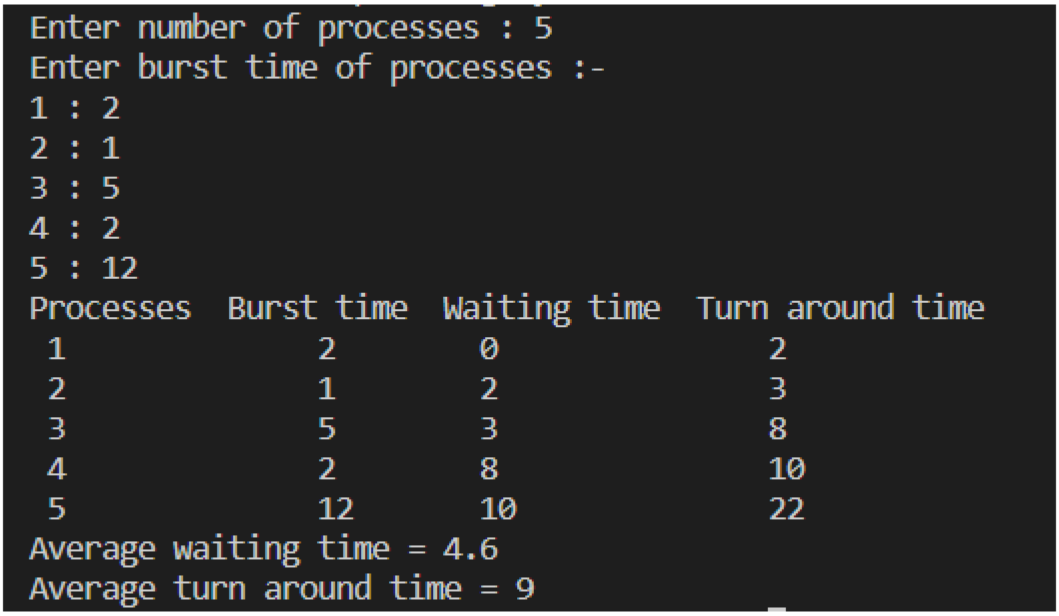
Solution 6 :

```
/* Created by - ANAND KUMAR MISHRA */

#include <iostream>
using namespace std;

void findWaitingTime(int processes[], int n, int bt[], int wt[])
{
    // waiting time for first process is 0
    wt[0] = 0;

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

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[])
{
    // calculating turnaround time by adding bt[i] + wt[i]
    for (int i = 0; i < n; i++)
    {
        tat[i] = bt[i] + wt[i];
    }
}

void findavgTime(int processes[], int n, int bt[])
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    findWaitingTime(processes, n, bt, wt);
    findTurnAroundTime(processes, n, bt, wt, tat);
    cout << "Processes "
         << " Burst time "
         << " Waiting time "
         << " Turn around time\n";

    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << i + 1 << "\t\t" << bt[i] << "\t "
             << wt[i] << "\t\t " << tat[i] << endl;
    }

    cout << "Average waiting time = "
         << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
         << (float)total_tat / (float)n;
}

int main()
{
    int n;
    cout << "Enter number of processes : ";
    cin >> n;
    int processes[n];
    for (int i = 0; i < n; i++)
    {
        processes[i] = i + 1;
    }

    int burst_time[n];
    cout << "Enter burst time of processes :- " << endl;
    for (int i = 0; i < n; i++)

    {
        cout << i + 1 << " : ";
        cin >> burst_time[i];
    }

    findavgTime(processes, n, burst_time);
    return 0;
}
```

Output 6 :



```
Enter number of processes : 4
Enter burst time of processes :-
1 : 10
2 : 6
3 : 5
4 : 2
Processes  Burst time  Waiting time  Turn around time
 1              10       0               10
 2              6        10              16
 3              5        16              21
 4              2        21              23
Average waiting time = 11.75
Average turn around time = 17.5
```



```
Enter number of processes : 5
Enter burst time of processes :-
1 : 2
2 : 1
3 : 5
4 : 2
5 : 12
Processes  Burst time  Waiting time  Turn around time
 1              2        0               2
 2              1        2               3
 3              5        3               8
 4              2        8               10
 5              12       10              22
Average waiting time = 4.6
Average turn around time = 9
```

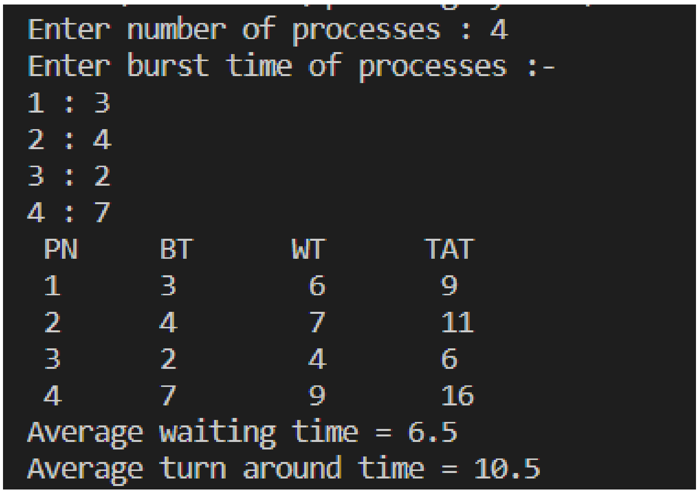
Question 7 :

Write a program to implement Round Robin scheduling algorithm.

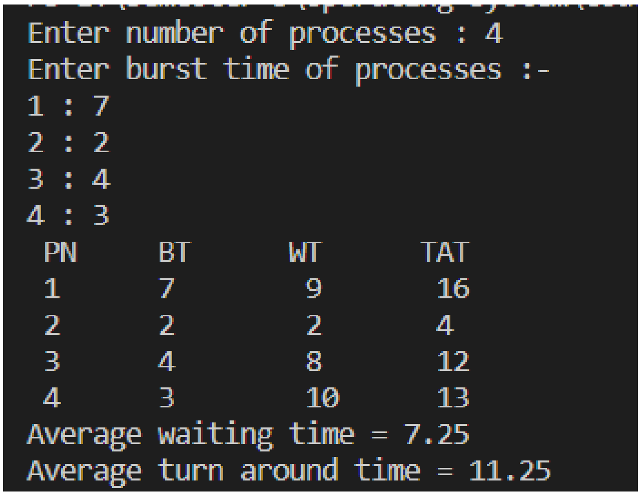
Solution 7 :

```
/* Created by - ANAND KUMAR MISHRA */

#include <iostream>
using namespace std;

void findWaitingTime(int processes[], int n,
                     int bt[], int wt[], int quantum)
{
    int rem_bt[n];
    for (int i = 0; i < n; i++)
        rem_bt[i] = bt[i];

    int t = 0; // Current time

    while (1)
    {
        bool done = true;

        for (int i = 0; i < n; i++)
        {
            // If burst time of a process is greater than 0
            // then only need to process further
            if (rem_bt[i] > 0)
            {
                done = false; // There is a pending process

                if (rem_bt[i] > quantum)
                {
                    // Increase the value of t i.e. shows
                    // how much time a process has been processed
                    t += quantum;

                    // Decrease the burst_time of current process
                    // by quantum
                    rem_bt[i] -= quantum;
                }

                // If burst time is smaller than or equal to
                // quantum. Last cycle for this process
                else
                {
                    // Increase the value of t i.e. shows
                    // how much time a process has been processed
                    t = t + rem_bt[i];

                    // Waiting time is current time minus time
                    // used by this process
                    wt[i] = t - bt[i];

                    // As the process gets fully executed
                    // make its remaining burst time = 0
                    rem_bt[i] = 0;
                }
            }
        }

        // If all processes are done
        if (done == true)
            break;
    }
}

// Function to calculate turn around time
void findTurnAroundTime(int processes[], int n,
                        int bt[], int wt[], int tat[])
{
    // calculating turnaround time by adding
    // bt[i] + wt[i]
    for (int i = 0; i < n; i++)
        tat[i] = bt[i] + wt[i];
}

// Function to calculate average time
void findavgTime(int processes[], int n, int bt[],
                 int quantum)
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    // Function to find waiting time of all processes
    findWaitingTime(processes, n, bt, wt, quantum);

    // Function to find turn around time for all processes
    findTurnAroundTime(processes, n, bt, wt, tat);

    // Display processes along with all details
    cout << " PN "
         << " \tBT "
         << " \tWT "
         << " \tTAT\n";

    // Calculate total waiting time and total turn
    // around time
    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << i + 1 << "\t" << bt[i] << "\t "
             << wt[i] << "\t " << tat[i] << endl;
    }

    cout << "Average waiting time = "
         << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
         << (float)total_tat / (float)n;
}

// Driver code
int main()
{
    int n;
    cout << "Enter number of processes : ";
    cin >> n;
    int processes[n];
    for (int i = 0; i < n; i++)
    {
        processes[i] = i + 1;
    }

    int burst_time[n];
    cout << "Enter burst time of processes :- " << endl;
    for (int i = 0; i < n; i++)
    {
        cout << i + 1 << " : ";
        cin >> burst_time[i];
    }

    // Time quantum
    int quantum = 2;
    findavgTime(processes, n, burst_time, quantum);
    return 0;
}
```

Output 7 :



```
Enter number of processes : 4
Enter burst time of processes :-
1 : 3
2 : 4
3 : 2
4 : 7
 PN      BT       WT       TAT
 1       3         6        9
 2       4         7        11
 3       2         4        6
 4       7         9        16
Average waiting time = 6.5
Average turn around time = 10.5
```



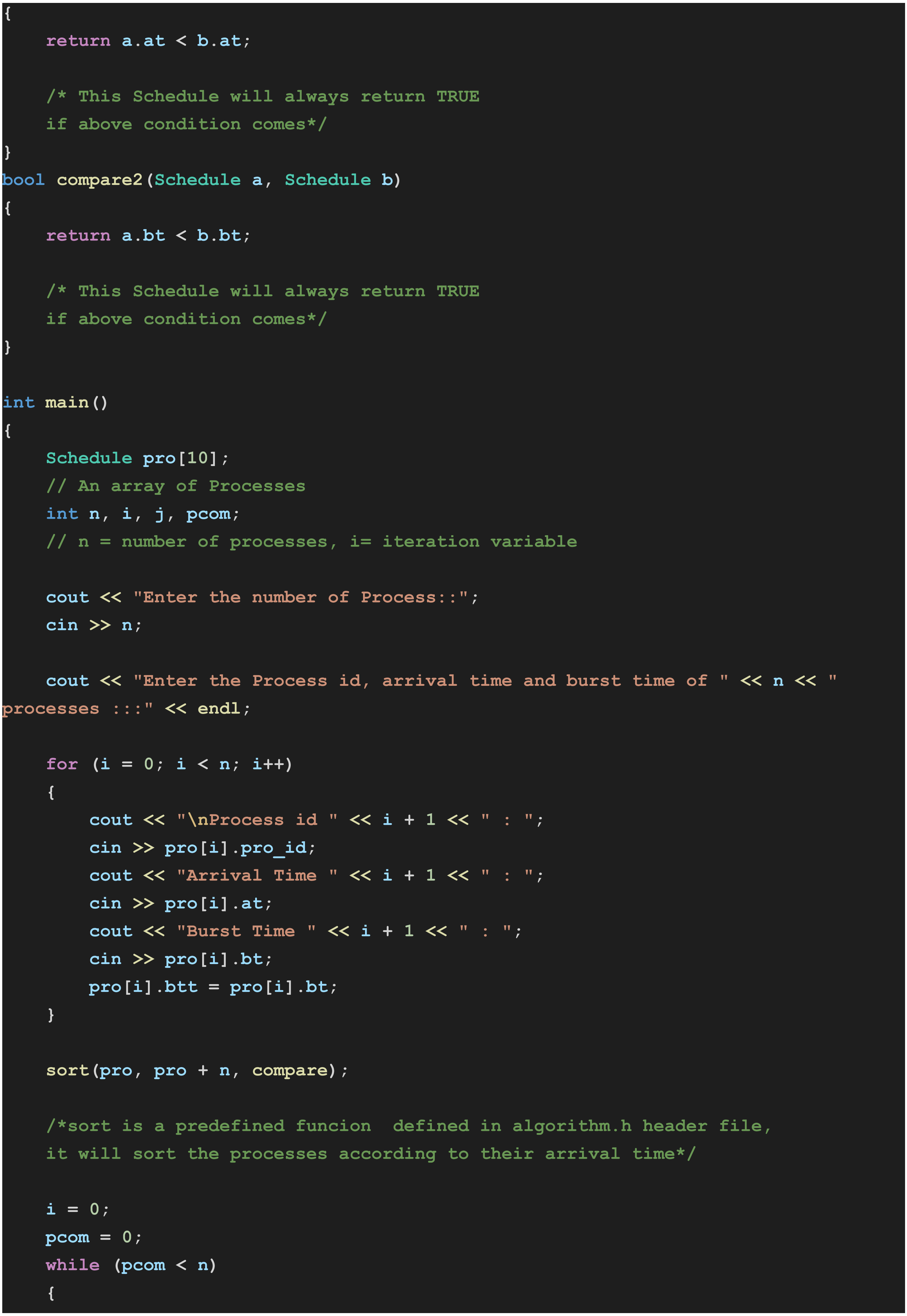
```
Enter number of processes : 4
Enter burst time of processes :-
1 : 7
2 : 2
3 : 4
4 : 3
 PN      BT       WT       TAT
 1       7         9        16
 2       2         2        4
 3       4         8        12
 4       3         10       13
Average waiting time = 7.25
Average turn around time = 11.25
```

Question 8 :

Write a program to implement SJF scheduling algorithm.

Solution 8 :

```
/* Created by - ANAND KUMAR MISHRA */

#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;

typedef struct proccess
{
    int at, bt, ct, ta, wt, btt;
    string pro_id;

    /*
at = Arrival time,
bt = Burst time,
ct = Completion time,
ta = Turn around time,
wt = Waiting time
*/

} Schedule;

bool compare(Schedule a, Schedule b)
```

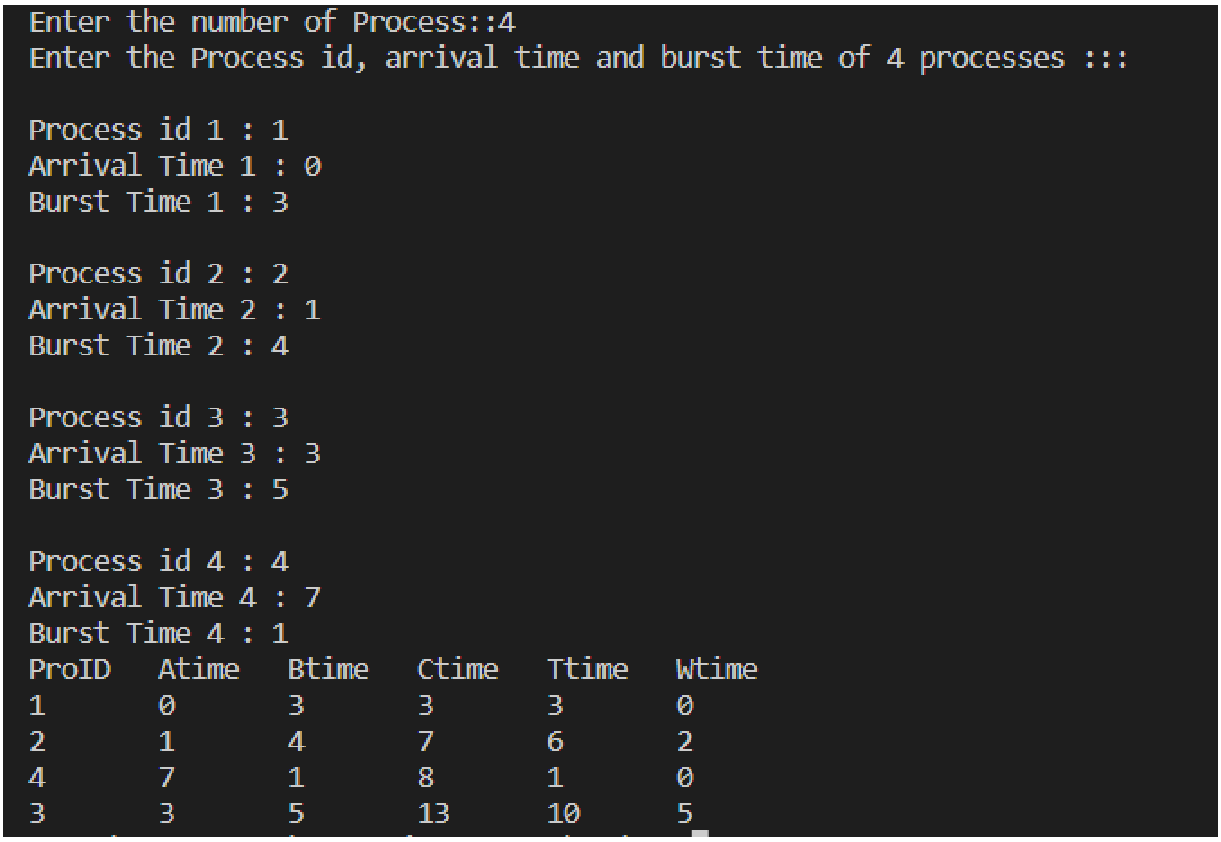


```
{
    return a.at < b.at;

    /* This Schedule will always return TRUE
    if above condition comes*/
}
bool compare2(Schedule a, Schedule b)
{
    return a.bt < b.bt;

    /* This Schedule will always return TRUE
    if above condition comes*/
}

int main()
{
    Schedule pro[10];
    // An array of Processes
    int n, i, j, pcom;
    // n = number of processes, i= iteration variable

    cout << "Enter the number of Process::";
    cin >> n;

    cout << "Enter the Process id, arrival time and burst time of " << n << "
processes :::" << endl;

    for (i = 0; i < n; i++)
    {
        cout << "\nProcess id " << i + 1 << " : ";
        cin >> pro[i].pro_id;
        cout << "Arrival Time " << i + 1 << " : ";
        cin >> pro[i].at;
        cout << "Burst Time " << i + 1 << " : ";
        cin >> pro[i].bt;
        pro[i].btt = pro[i].bt;
    }

    sort(pro, pro + n, compare);

    /*sort is a predefined funcion  defined in algorithm.h header file,
    it will sort the processes according to their arrival time*/

    i = 0;
    pcom = 0;
    while (pcom < n)
    {
```

```
    for (j = 0; j < n; j++)
    {
        if (pro[j].at > i)
            break;
    }

    sort(pro, pro + j, compare2);

    /*sort is a predefined funcion  defined in algorithm.h header file,
it will sort the processes according to their burst time*/

    if (j > 0)
    {

        for (j = 0; j < n; j++)
        {
            if (pro[j].bt != 0)
                break;
        }
        if (pro[j].at > i)

        {
            i = pro[j].at;
        }
        pro[j].ct = i + 1;
        pro[j].bt--;
    }
    i++;
    pcom = 0;
    for (j = 0; j < n; j++)
    {
        if (pro[j].bt == 0)
            pcom++;
    }
}

cout << "ProID\tAtime\tBtime\tCtime\tTtime\tWtime\n";

for (i = 0; i < n; i++)
{
    pro[i].ta = pro[i].ct - pro[i].at;
    pro[i].wt = pro[i].ta - pro[i].btt;

    /*Printing the Process id, arrival time, burst time,
    completion time, turn around time, waiting time*/
```

```
        cout << pro[i].pro_id << "\t" << pro[i].at << "\t" << pro[i].btt << "\t" <<
pro[i].ct << "\t" << pro[i].ta << "\t" << pro[i].wt;
        cout << endl;
    }
    return 0;
}
```

Output 8 :



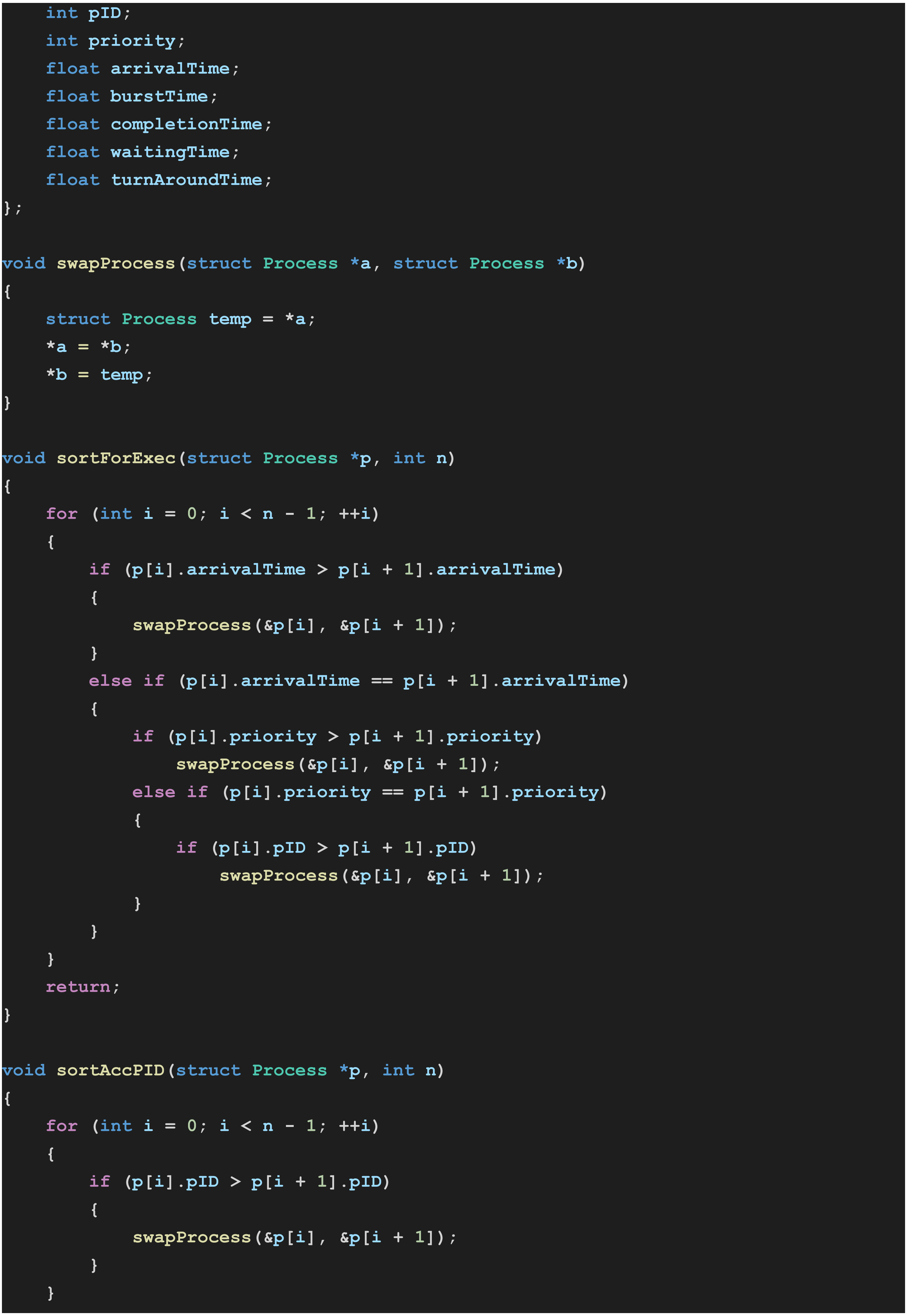
```
Enter the number of Process::4
Enter the Process id, arrival time and burst time of 4 processes :::

Process id 1 : 1
Arrival Time 1 : 0
Burst Time 1 : 3

Process id 2 : 2
Arrival Time 2 : 1
Burst Time 2 : 4

Process id 3 : 3
Arrival Time 3 : 3
Burst Time 3 : 5

Process id 4 : 4
Arrival Time 4 : 7
Burst Time 4 : 1
ProID   Atime   Btime   Ctime   Ttime   Wtime
1       0       3       3       3       0
2       1       4       7       6       2
4       7       1       8       1       0
3       3       5       13      10      5
```

Question 9 :

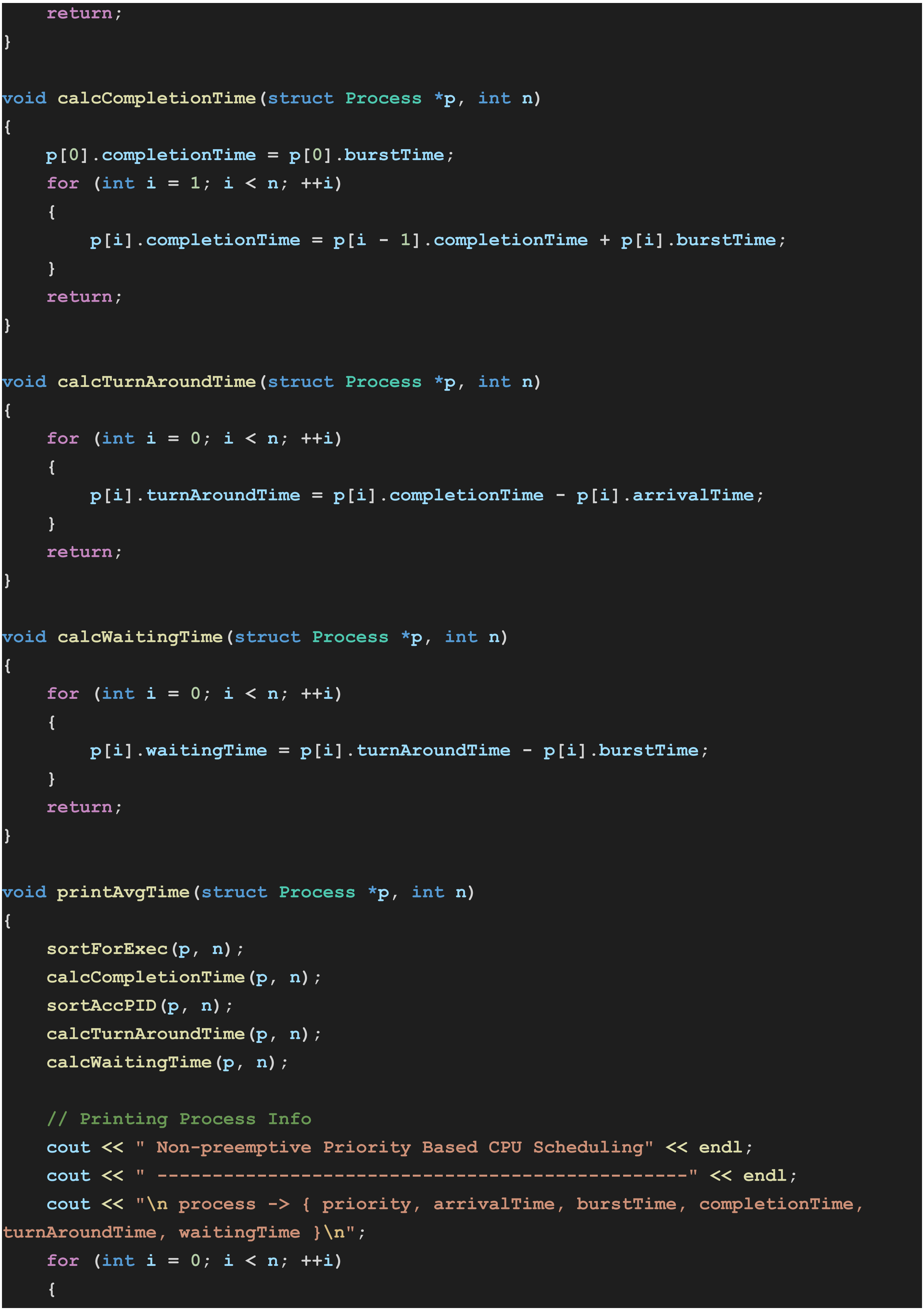
Write a program to implement non-preemptive priority based scheduling algorithm.

Solution 9 :

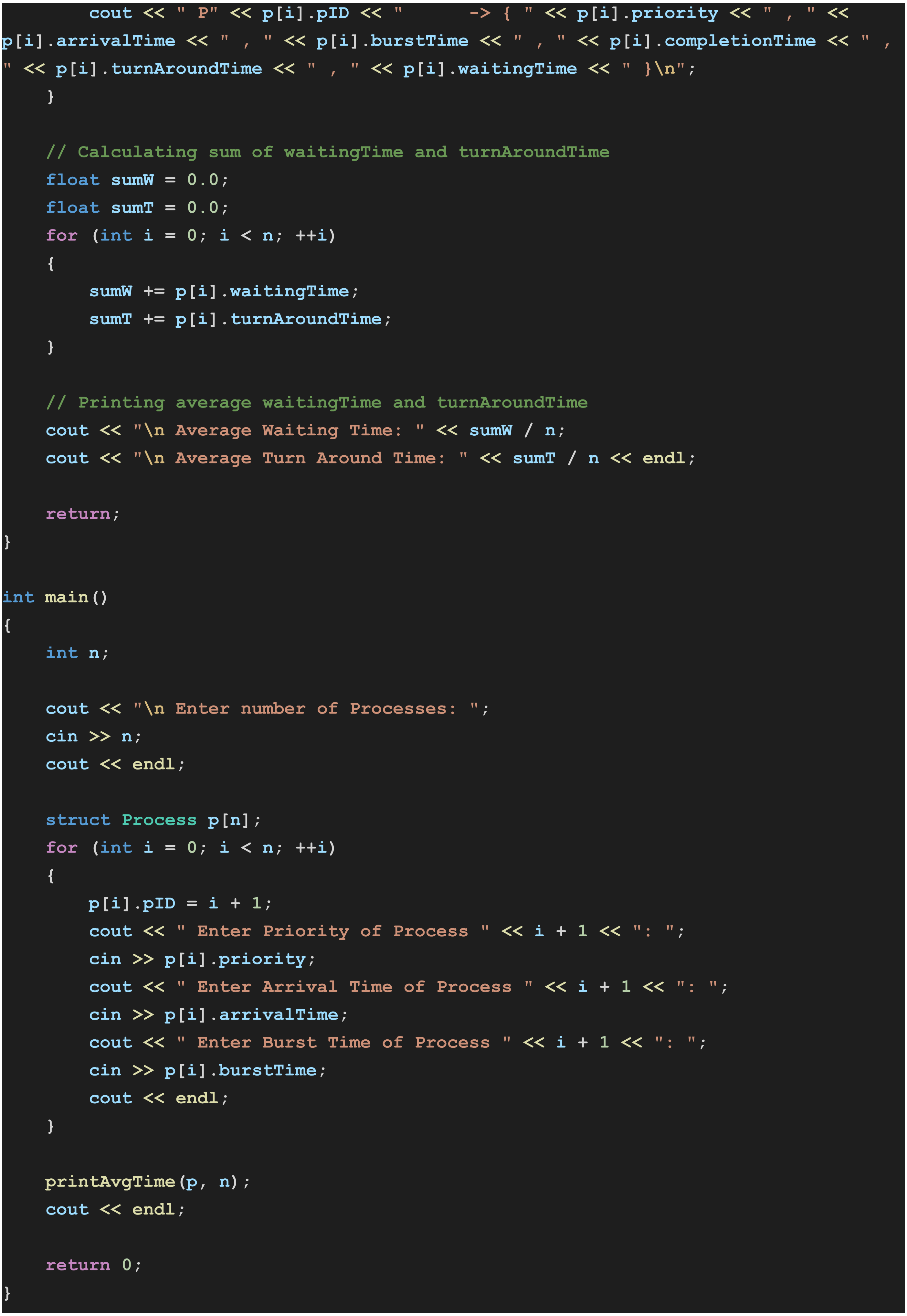
```
/* Created by - ANAND KUMAR MISHRA */

#include <iostream>
#include <stdlib.h>
using namespace std;

struct Process
{
```



```c
    int pID;
    int priority;
    float arrivalTime;
    float burstTime;
    float completionTime;
    float waitingTime;
    float turnAroundTime;
};

void swapProcess(struct Process *a, struct Process *b)
{
    struct Process temp = *a;
    *a = *b;
    *b = temp;
}

void sortForExec(struct Process *p, int n)
{
    for (int i = 0; i < n - 1; ++i)
    {
        if (p[i].arrivalTime > p[i + 1].arrivalTime)
        {
            swapProcess(&p[i], &p[i + 1]);
        }
        else if (p[i].arrivalTime == p[i + 1].arrivalTime)
        {
            if (p[i].priority > p[i + 1].priority)
                swapProcess(&p[i], &p[i + 1]);
            else if (p[i].priority == p[i + 1].priority)
            {
                if (p[i].pID > p[i + 1].pID)
                    swapProcess(&p[i], &p[i + 1]);
            }
        }
    }
    return;
}

void sortAccPID(struct Process *p, int n)
{
    for (int i = 0; i < n - 1; ++i)
    {
        if (p[i].pID > p[i + 1].pID)
        {
            swapProcess(&p[i], &p[i + 1]);
        }
    }
```

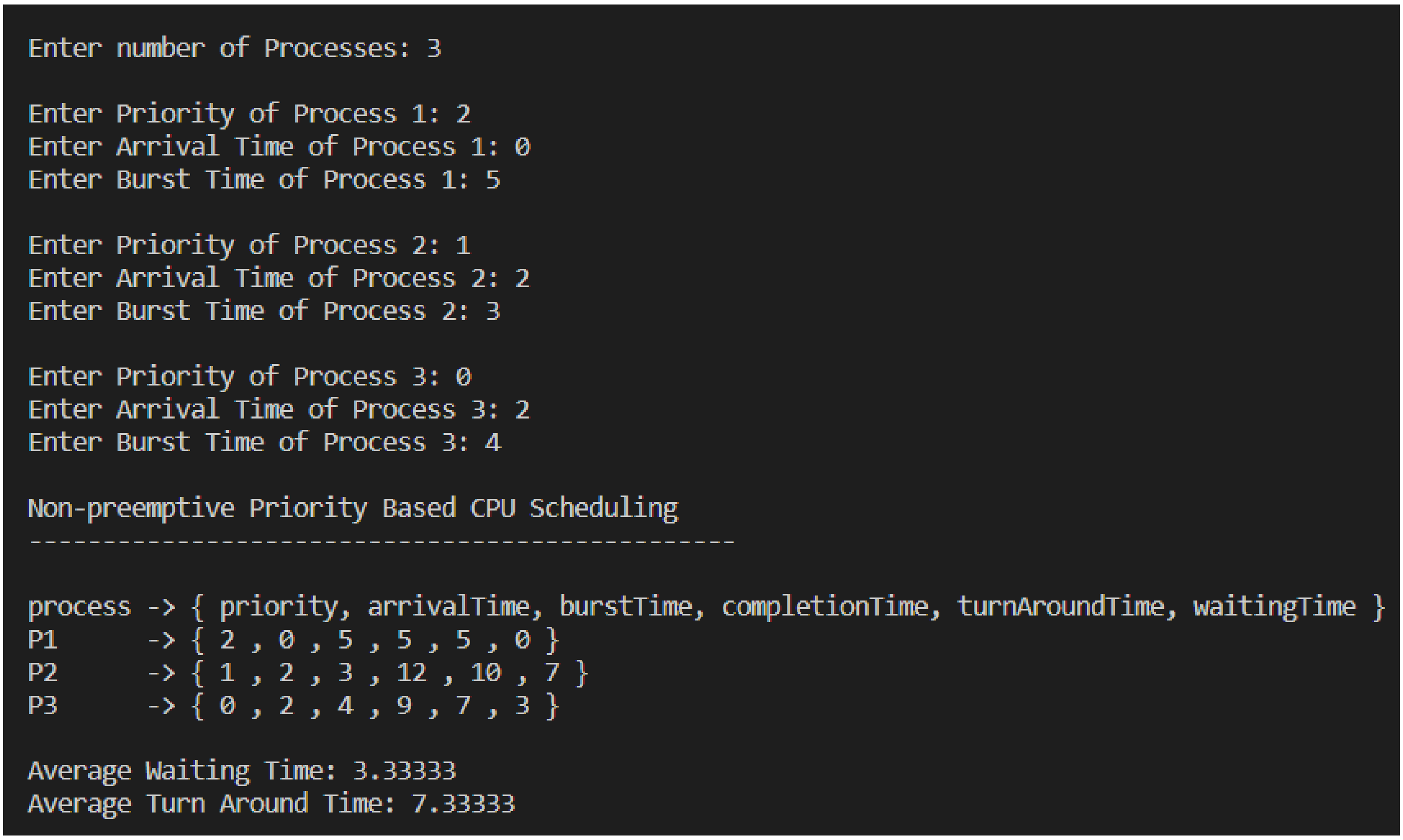


```
    return;
}

void calcCompletionTime(struct Process *p, int n)
{
    p[0].completionTime = p[0].burstTime;
    for (int i = 1; i < n; ++i)
    {
        p[i].completionTime = p[i - 1].completionTime + p[i].burstTime;
    }
    return;
}

void calcTurnAroundTime(struct Process *p, int n)
{
    for (int i = 0; i < n; ++i)
    {
        p[i].turnAroundTime = p[i].completionTime - p[i].arrivalTime;
    }
    return;
}

void calcWaitingTime(struct Process *p, int n)
{
    for (int i = 0; i < n; ++i)
    {
        p[i].waitingTime = p[i].turnAroundTime - p[i].burstTime;
    }
    return;
}

void printAvgTime(struct Process *p, int n)
{
    sortForExec(p, n);
    calcCompletionTime(p, n);
    sortAccPID(p, n);
    calcTurnAroundTime(p, n);
    calcWaitingTime(p, n);

    // Printing Process Info
    cout << " Non-preemptive Priority Based CPU Scheduling" << endl;
    cout << " ------------------------------------------------" << endl;
    cout << "\n process -> { priority, arrivalTime, burstTime, completionTime,
turnAroundTime, waitingTime }\n";
    for (int i = 0; i < n; ++i)
    {
```



```
        cout << " P" << p[i].pID << "      -> { " << p[i].priority << " , " <<
p[i].arrivalTime << " , " << p[i].burstTime << " , " << p[i].completionTime << " ,
" << p[i].turnAroundTime << " , " << p[i].waitingTime << " }\n";
    }

    // Calculating sum of waitingTime and turnAroundTime
    float sumW = 0.0;
    float sumT = 0.0;
    for (int i = 0; i < n; ++i)
    {
        sumW += p[i].waitingTime;
        sumT += p[i].turnAroundTime;
    }

    // Printing average waitingTime and turnAroundTime
    cout << "\n Average Waiting Time: " << sumW / n;
    cout << "\n Average Turn Around Time: " << sumT / n << endl;

    return;
}

int main()
{
    int n;

    cout << "\n Enter number of Processes: ";
    cin >> n;
    cout << endl;

    struct Process p[n];
    for (int i = 0; i < n; ++i)
    {
        p[i].pID = i + 1;
        cout << " Enter Priority of Process " << i + 1 << ": ";
        cin >> p[i].priority;
        cout << " Enter Arrival Time of Process " << i + 1 << ": ";
        cin >> p[i].arrivalTime;
        cout << " Enter Burst Time of Process " << i + 1 << ": ";
        cin >> p[i].burstTime;
        cout << endl;
    }

    printAvgTime(p, n);
    cout << endl;

    return 0;
}
```

Output 9 :

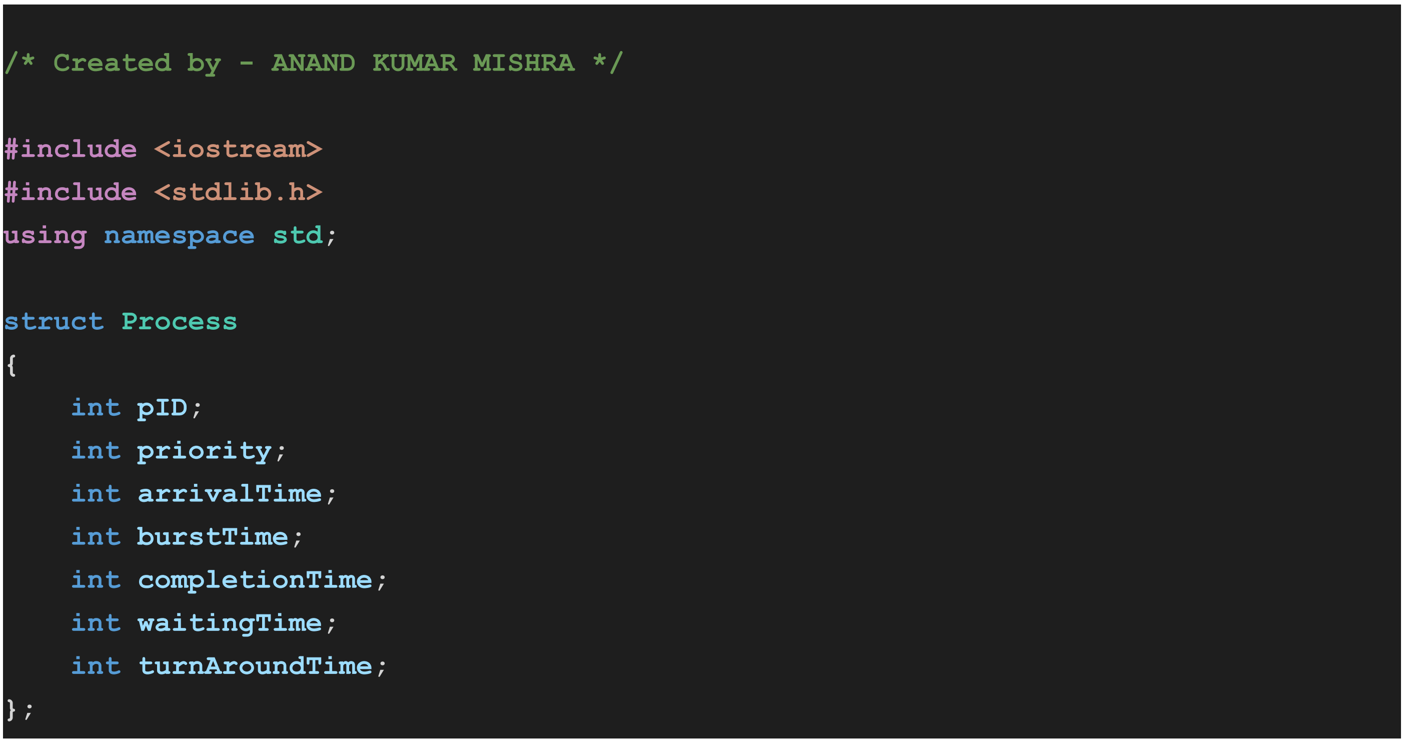


```
Enter number of Processes: 3

Enter Priority of Process 1: 2
Enter Arrival Time of Process 1: 0
Enter Burst Time of Process 1: 5

Enter Priority of Process 2: 1
Enter Arrival Time of Process 2: 2
Enter Burst Time of Process 2: 3

Enter Priority of Process 3: 0
Enter Arrival Time of Process 3: 2
Enter Burst Time of Process 3: 4

Non-preemptive Priority Based CPU Scheduling
------------------------------------------------

process -> { priority, arrivalTime, burstTime, completionTime, turnAroundTime, waitingTime }
P1      -> { 2 , 0 , 5 , 5 , 5 , 0 }
P2      -> { 1 , 2 , 3 , 12 , 10 , 7 }
P3      -> { 0 , 2 , 4 , 9 , 7 , 3 }

Average Waiting Time: 3.33333
Average Turn Around Time: 7.33333
```

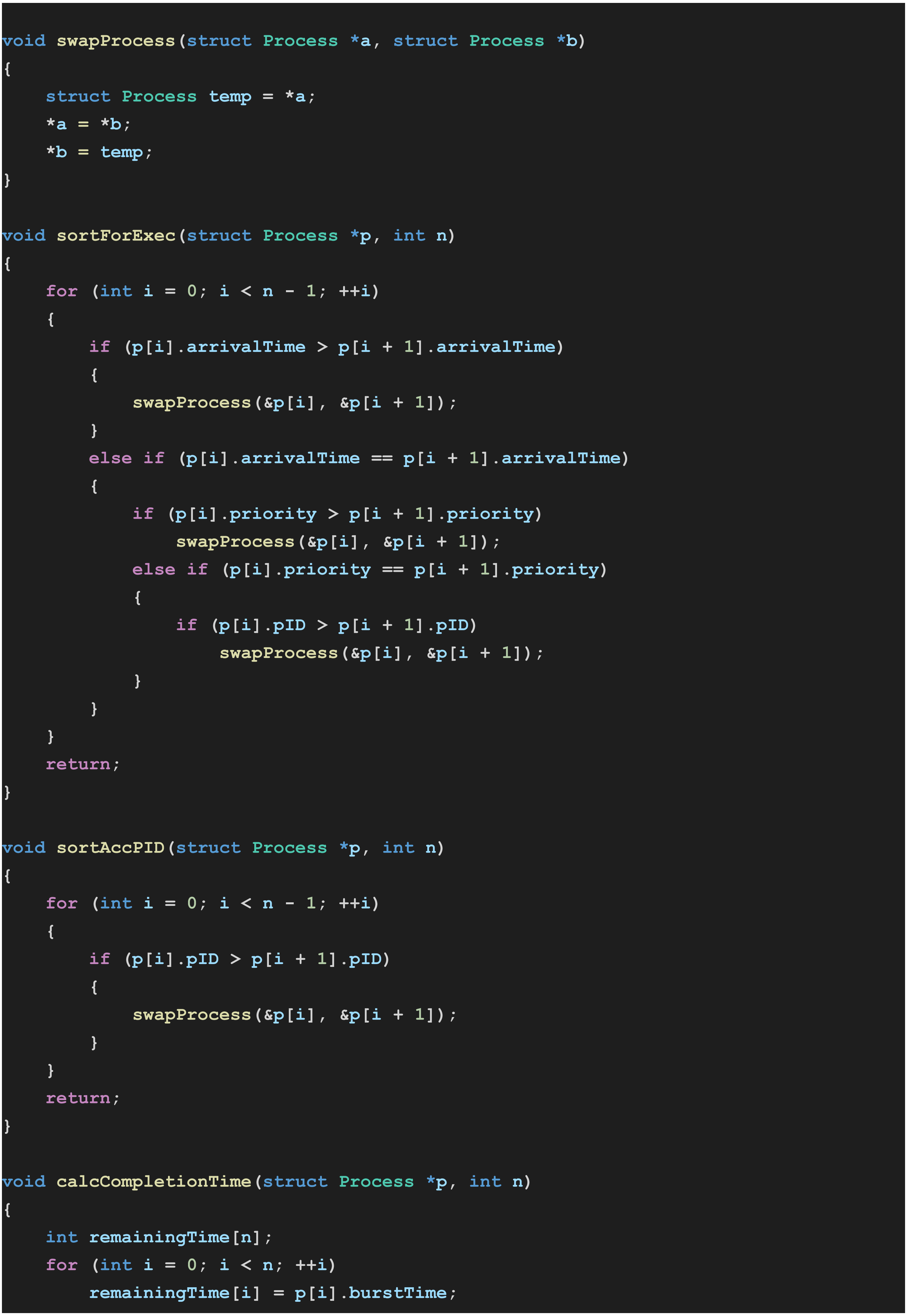
Question 10 :

Write a program to implement a preemptive priority based scheduling algorithm.

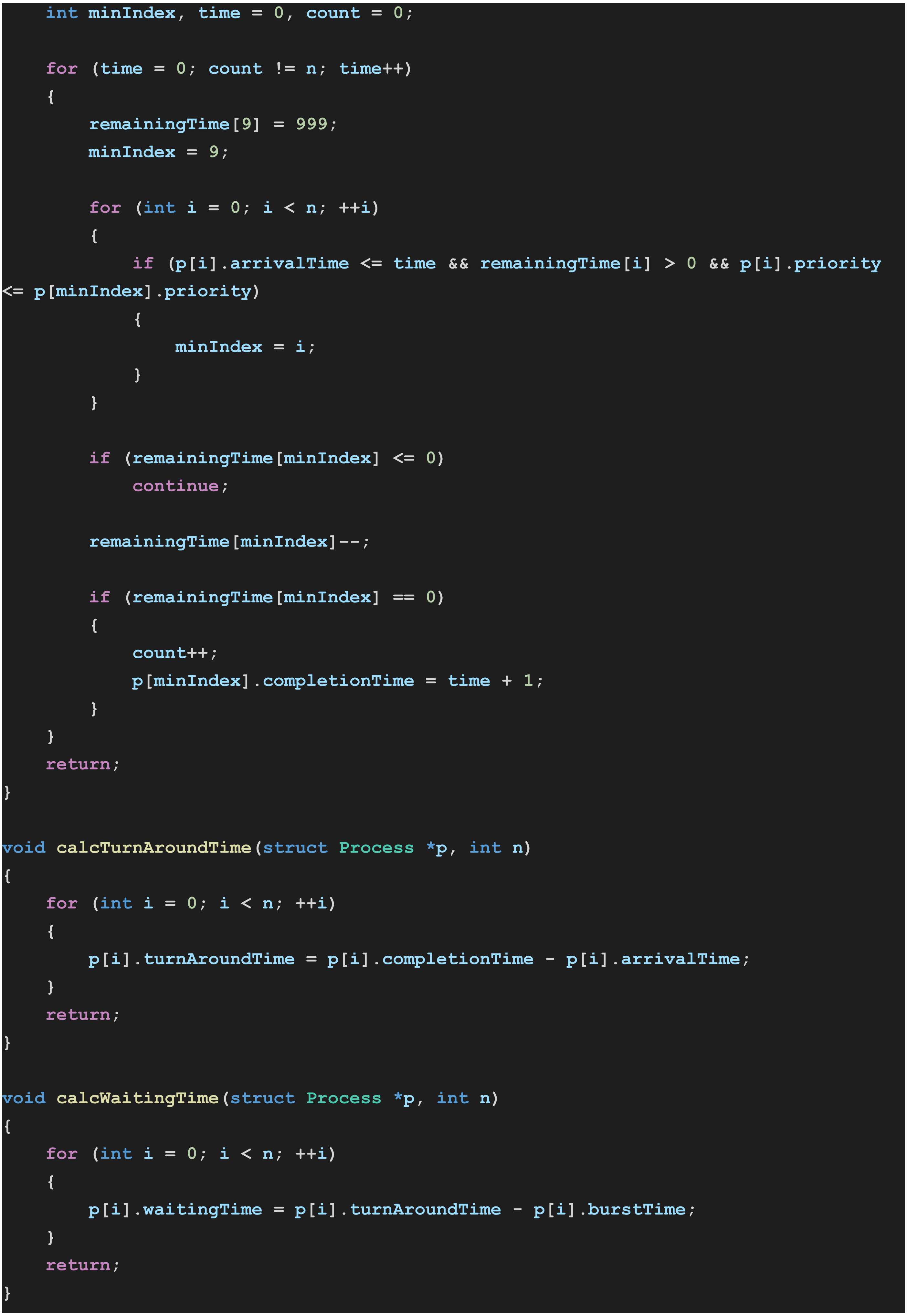
Solution 10 :



```cpp
/* Created by - ANAND KUMAR MISHRA */

#include <iostream>
#include <stdlib.h>
using namespace std;

struct Process
{
    int pID;
    int priority;
    int arrivalTime;
    int burstTime;
    int completionTime;
    int waitingTime;
    int turnAroundTime;
};
```

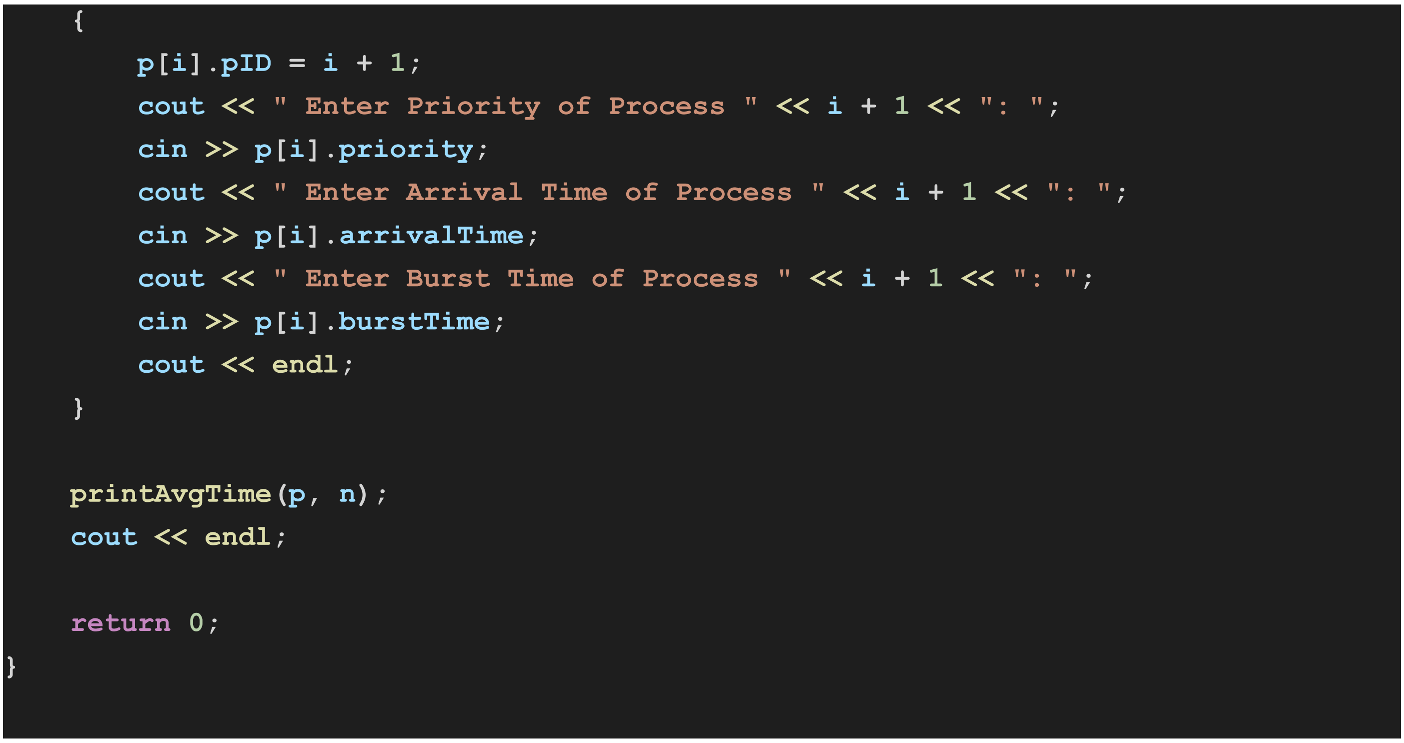


```c
void swapProcess(struct Process *a, struct Process *b)
{
    struct Process temp = *a;
    *a = *b;
    *b = temp;
}

void sortForExec(struct Process *p, int n)
{
    for (int i = 0; i < n - 1; ++i)
    {
        if (p[i].arrivalTime > p[i + 1].arrivalTime)
        {
            swapProcess(&p[i], &p[i + 1]);
        }
        else if (p[i].arrivalTime == p[i + 1].arrivalTime)
        {
            if (p[i].priority > p[i + 1].priority)
                swapProcess(&p[i], &p[i + 1]);
            else if (p[i].priority == p[i + 1].priority)
            {
                if (p[i].pID > p[i + 1].pID)
                    swapProcess(&p[i], &p[i + 1]);
            }
        }
    }
    return;
}

void sortAccPID(struct Process *p, int n)
{
    for (int i = 0; i < n - 1; ++i)
    {
        if (p[i].pID > p[i + 1].pID)
        {
            swapProcess(&p[i], &p[i + 1]);
        }
    }
    return;
}

void calcCompletionTime(struct Process *p, int n)
{
    int remainingTime[n];
    for (int i = 0; i < n; ++i)
        remainingTime[i] = p[i].burstTime;
```



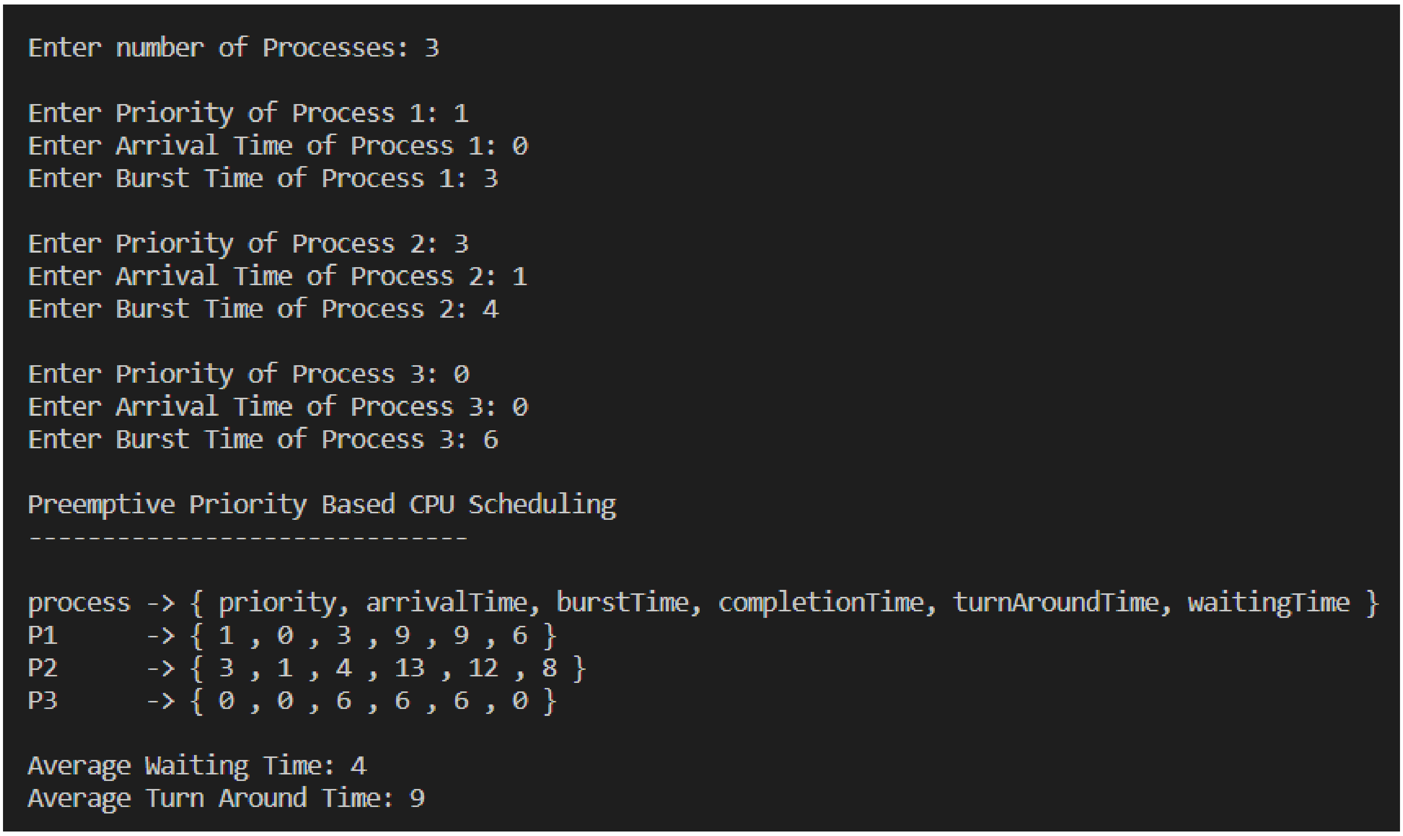
```
    int minIndex, time = 0, count = 0;

    for (time = 0; count != n; time++)
    {
        remainingTime[9] = 999;
        minIndex = 9;

        for (int i = 0; i < n; ++i)
        {
            if (p[i].arrivalTime <= time && remainingTime[i] > 0 && p[i].priority
<= p[minIndex].priority)
            {
                minIndex = i;
            }
        }

        if (remainingTime[minIndex] <= 0)
            continue;

        remainingTime[minIndex]--;

        if (remainingTime[minIndex] == 0)
        {
            count++;
            p[minIndex].completionTime = time + 1;
        }
    }
    return;
}

void calcTurnAroundTime(struct Process *p, int n)
{
    for (int i = 0; i < n; ++i)
    {
        p[i].turnAroundTime = p[i].completionTime - p[i].arrivalTime;
    }
    return;
}

void calcWaitingTime(struct Process *p, int n)
{
    for (int i = 0; i < n; ++i)
    {
        p[i].waitingTime = p[i].turnAroundTime - p[i].burstTime;
    }
    return;
}
```

```
void printAvgTime(struct Process *p, int n)
{
    sortForExec(p, n);
    calcCompletionTime(p, n);
    calcTurnAroundTime(p, n);
    calcWaitingTime(p, n);
    sortAccPID(p, n);

    // Printing Process Info
    cout << " Preemptive Priority Based CPU Scheduling" << endl;
    cout << " ------------------------------" << endl;
    cout << "\n process -> { priority, arrivalTime, burstTime, completionTime, turnAroundTime, waitingTime }\n";
    for (int i = 0; i < n; ++i)
    {
        cout << " P" << p[i].pID << "      -> { " << p[i].priority << " , " << p[i].arrivalTime << " , " << p[i].burstTime << " , " << p[i].completionTime << " , " << p[i].turnAroundTime << " , " << p[i].waitingTime << " }\n";
    }

    // Calculating sum of waitingTime and turnAroundTime
    int sumW = 0.0;
    int sumT = 0.0;
    for (int i = 0; i < n; ++i)
    {
        sumW += p[i].waitingTime;
        sumT += p[i].turnAroundTime;
    }

    // Printing average waitingTime and turnAroundTime
    cout << "\n Average Waiting Time: " << sumW / n;
    cout << "\n Average Turn Around Time: " << sumT / n << endl;

    return;
}

int main()
{
    int n;

    cout << "\n Enter number of Processes: ";
    cin >> n;
    cout << endl;

    struct Process p[n];
    for (int i = 0; i < n; ++i)
```



```cpp
    {
        p[i].pID = i + 1;
        cout << " Enter Priority of Process " << i + 1 << ": ";
        cin >> p[i].priority;
        cout << " Enter Arrival Time of Process " << i + 1 << ": ";
        cin >> p[i].arrivalTime;
        cout << " Enter Burst Time of Process " << i + 1 << ": ";
        cin >> p[i].burstTime;
        cout << endl;
    }

    printAvgTime(p, n);
    cout << endl;

    return 0;
}
```

Output 10 :



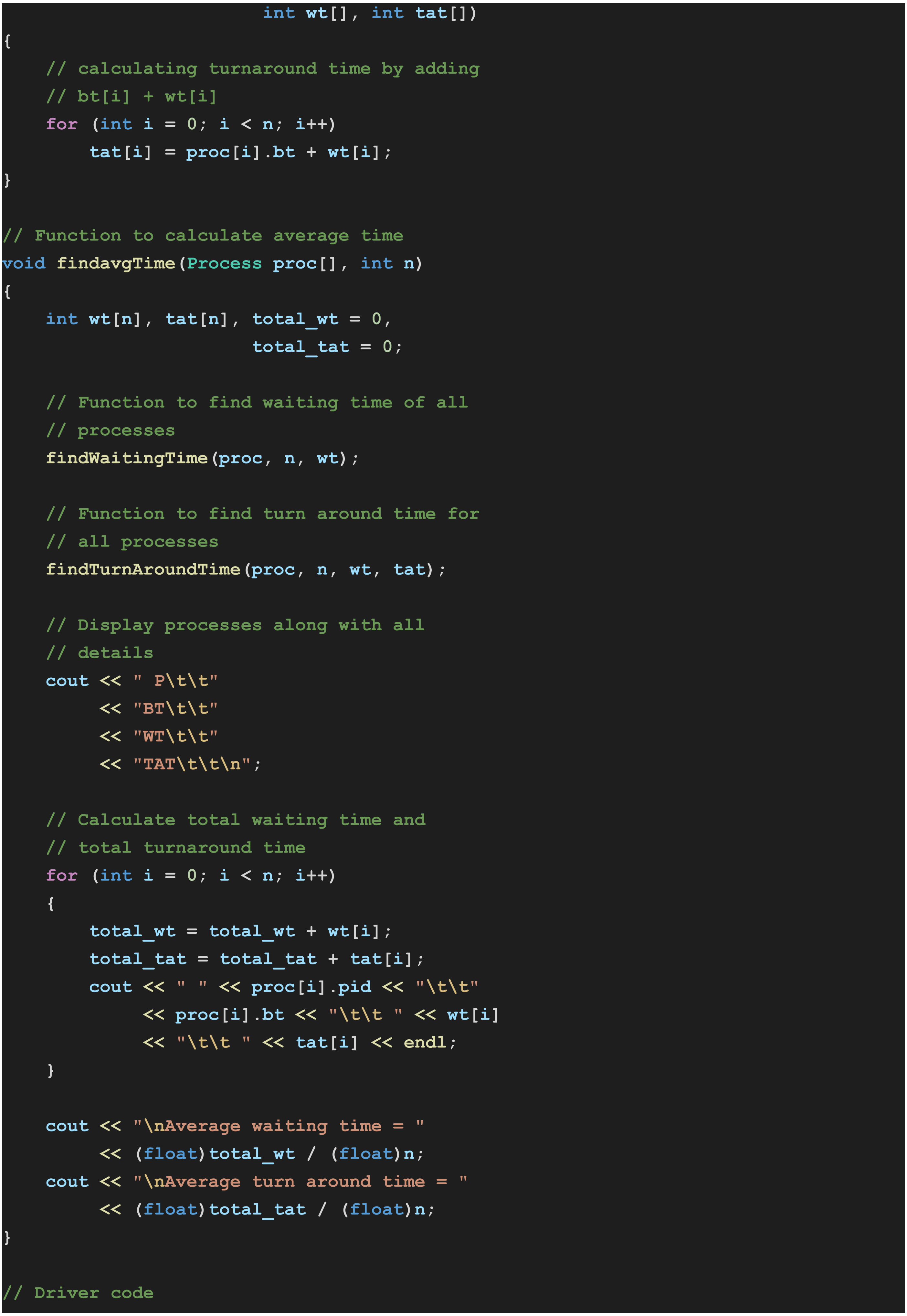
```
Enter number of Processes: 3

Enter Priority of Process 1: 1
Enter Arrival Time of Process 1: 0
Enter Burst Time of Process 1: 3

Enter Priority of Process 2: 3
Enter Arrival Time of Process 2: 1
Enter Burst Time of Process 2: 4

Enter Priority of Process 3: 0
Enter Arrival Time of Process 3: 0
Enter Burst Time of Process 3: 6

Preemptive Priority Based CPU Scheduling
------------------------------

process -> { priority, arrivalTime, burstTime, completionTime, turnAroundTime, waitingTime }
P1      -> { 1 , 0 , 3 , 9 , 9 , 6 }
P2      -> { 3 , 1 , 4 , 13 , 12 , 8 }
P3      -> { 0 , 0 , 6 , 6 , 6 , 0 }

Average Waiting Time: 4
Average Turn Around Time: 9
```

Question 11 :

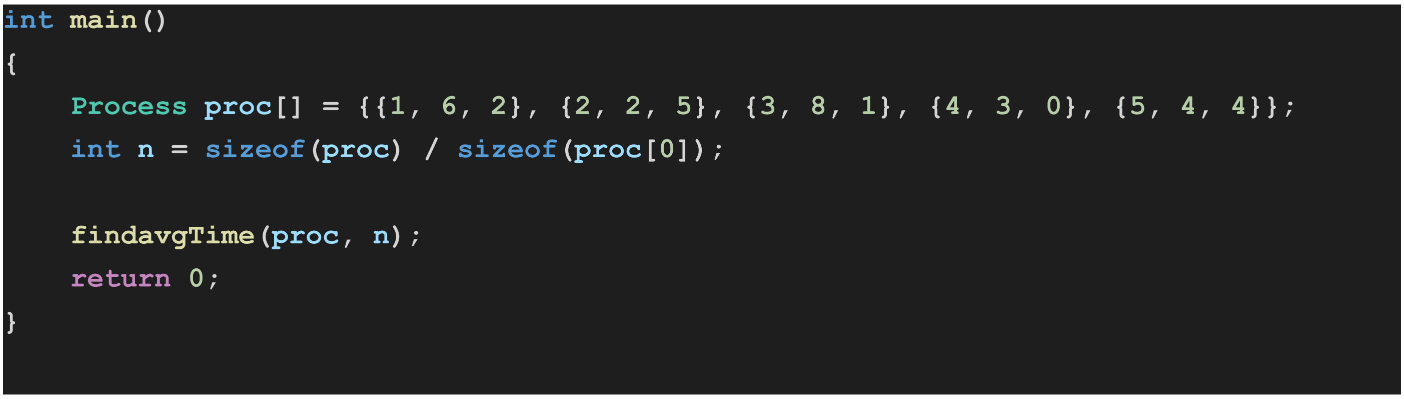
Write a program to implement SRJF scheduling algorithm.

Solution 11 :

```
/* Created by - ANAND KUMAR MISHRA */

#include <bits/stdc++.h>
using namespace std;

struct Process
{
    int pid; // Process ID
    int bt;  // Burst Time
    int art; // Arrival Time
};

// Function to find the waiting time for all processes
void findWaitingTime(Process proc[], int n, int wt[])
{
    int rt[n];

    // Copy the burst time into rt[]
    for (int i = 0; i < n; i++)
        rt[i] = proc[i].bt;

    int complete = 0, t = 0, minm = INT_MAX;
    int shortest = 0, finish_time;
    bool check = false;

    // Process until all processes gets
    // completed
    while (complete != n)
    {

        // Find process with minimum
        // remaining time among the
        // processes that arrives till the
        // current time`
        for (int j = 0; j < n; j++)
        {
            if ((proc[j].art <= t) &&
                (rt[j] < minm) && rt[j] > 0)
            {
                minm = rt[j];
                shortest = j;
                check = true;
            }
        }

        if (check == false)
        {
            t++;
            continue;
        }

        // Reduce remaining time by one
        rt[shortest]--;

        // Update minimum
        minm = rt[shortest];
        if (minm == 0)
            minm = INT_MAX;

        // If a process gets completely
        // executed
        if (rt[shortest] == 0)
        {

            // Increment complete
            complete++;
            check = false;

            // Find finish time of current
            // process
            finish_time = t + 1;

            // Calculate waiting time
            wt[shortest] = finish_time -
                           proc[shortest].bt -
                           proc[shortest].art;

            if (wt[shortest] < 0)
                wt[shortest] = 0;
        }
        // Increment time
        t++;
    }
}

// Function to calculate turn around time
void findTurnAroundTime(Process proc[], int n,
```

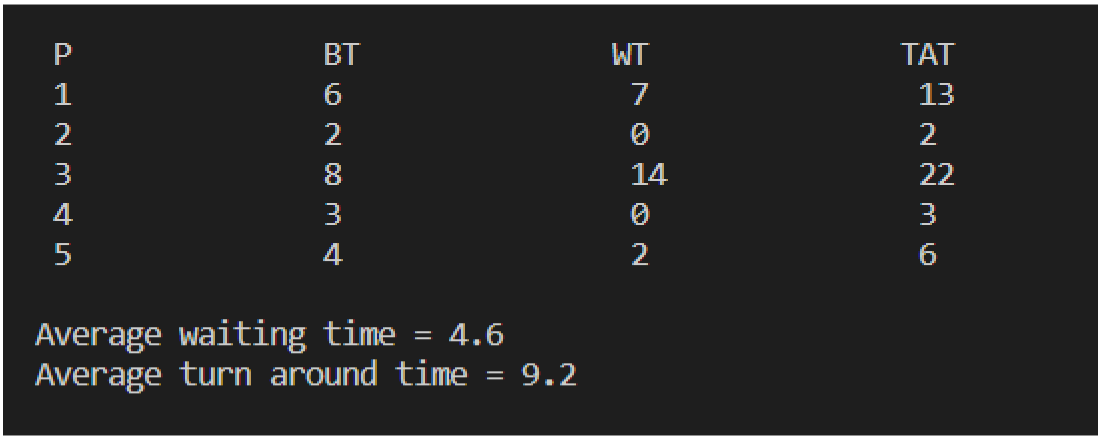


```cpp
                        int wt[], int tat[])
{
    // calculating turnaround time by adding
    // bt[i] + wt[i]
    for (int i = 0; i < n; i++)
        tat[i] = proc[i].bt + wt[i];
}

// Function to calculate average time
void findavgTime(Process proc[], int n)
{
    int wt[n], tat[n], total_wt = 0,
                       total_tat = 0;

    // Function to find waiting time of all
    // processes
    findWaitingTime(proc, n, wt);

    // Function to find turn around time for
    // all processes
    findTurnAroundTime(proc, n, wt, tat);

    // Display processes along with all
    // details
    cout << " P\t\t"
         << "BT\t\t"
         << "WT\t\t"
         << "TAT\t\t\n";

    // Calculate total waiting time and
    // total turnaround time
    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << proc[i].pid << "\t\t"
             << proc[i].bt << "\t\t " << wt[i]
             << "\t\t " << tat[i] << endl;
    }

    cout << "\nAverage waiting time = "
         << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
         << (float)total_tat / (float)n;
}

// Driver code
```



```
int main()
{
    Process proc[] = {{1, 6, 2}, {2, 2, 5}, {3, 8, 1}, {4, 3, 0}, {5, 4, 4}};
    int n = sizeof(proc) / sizeof(proc[0]);

    findavgTime(proc, n);
    return 0;
}
```

Output 11 :



```
 P               BT               WT               TAT
 1               6                 7                13
 2               2                 0                2
 3               8                 14               22
 4               3                 0                3
 5               4                 2                6

Average waiting time = 4.6
Average turn around time = 9.2
```

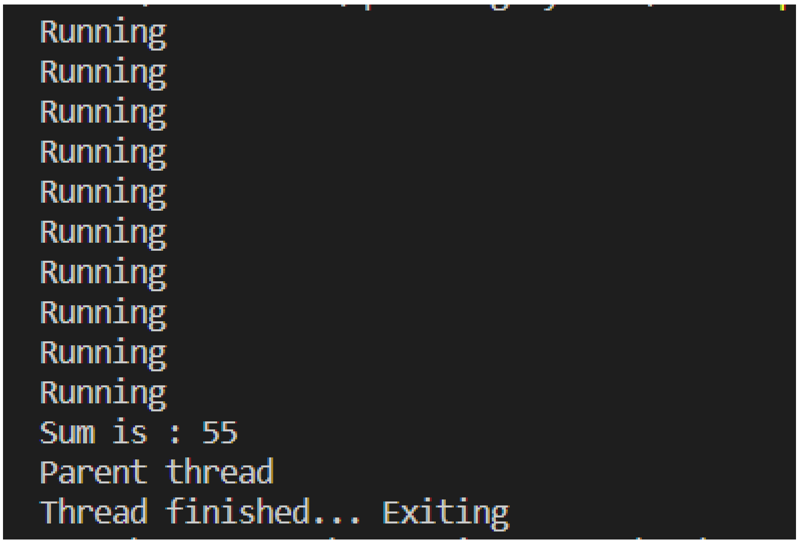
Question 12 :

Write a program to calculate sum of n numbers using thread library.

Solution 12 :

```python
# Created by - ANAND KUMAR MISHRA

from threading import Thread

# function to create threads
def callThread(arg):
    sumVal = 0
    for i in range(1, arg+1):
        print("Running")
        sumVal += i
    print(f"Sum is : {sumVal}")


if __name__ == "__main__":
    thread = Thread(target=callThread, args=(10, ))
    thread.start()
    thread.join()
    print("Parent thread")
    print("Thread finished... Exiting")
```

Output 12 :



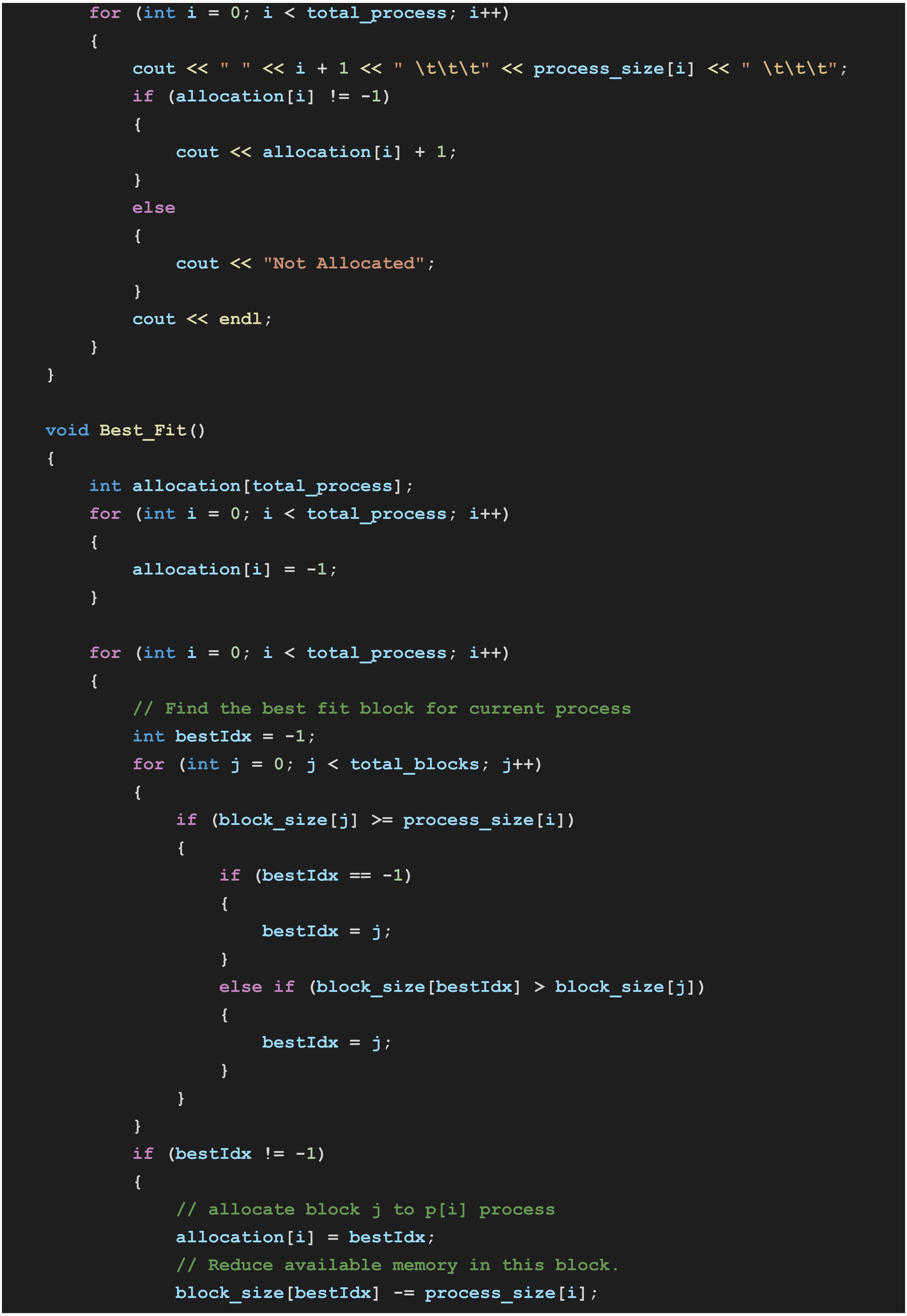
```
Running
Running
Running
Running
Running
Running
Running
Running
Running
Running
Sum is : 55
Parent thread
Thread finished... Exiting
```

Question 13 :

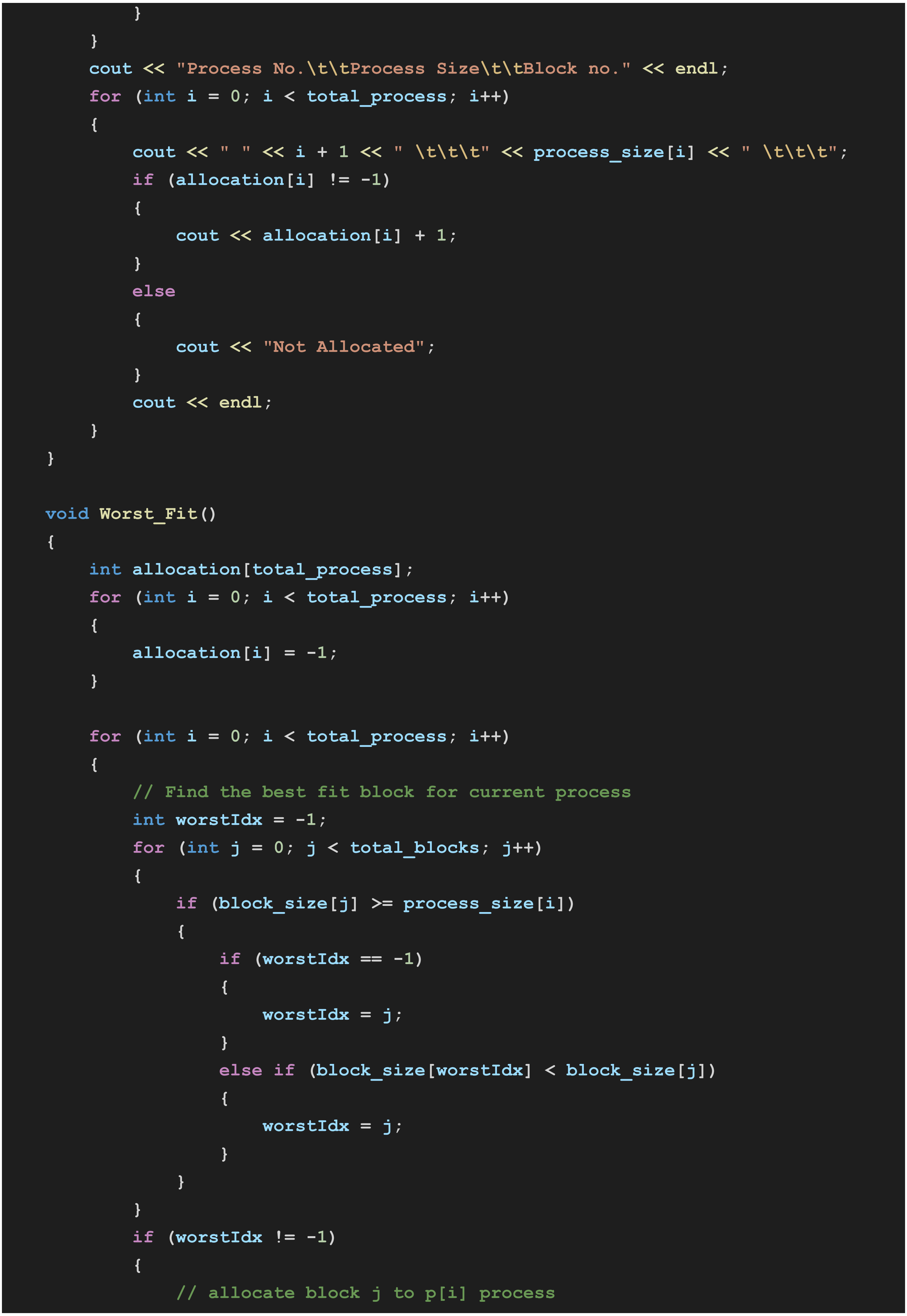
Write a program to implement first-fit, best-fit and worst-fit allocation strategies.

Solution 13 :

```
/* Created by - ANAND KUMAR MISHRA */

#include <iostream>
using namespace std;

class MemoryManagementAlgo
{
public:
    int *block_size;
    int total_blocks;
    int *process_size;
    int total_process;
    MemoryManagementAlgo(int blkSize[], int tBlocks, int prSize[], int tProcess)
    {
        block_size = blkSize;
        total_blocks = tBlocks;
        process_size = prSize;
        total_process = tProcess;
    }
    void First_Fit()
    {
        int allocation[total_process];
        for (int i = 0; i < total_process; i++)
        {
            allocation[i] = -1;
        }

        for (int i = 0; i < total_process; i++)
        {
            for (int j = 0; j < total_blocks; j++)
            {
                if (block_size[j] >= process_size[i])
                {
                    allocation[i] = j;
                    block_size[j] -= process_size[i];
                    break;
                }
            }
        }
        cout << "Process No.\t\tProcess Size\t\tBlock no." << endl;
```



```
    for (int i = 0; i < total_process; i++)
    {
        cout << " " << i + 1 << " \t\t\t" << process_size[i] << " \t\t\t";
        if (allocation[i] != -1)
        {
            cout << allocation[i] + 1;
        }
        else
        {
            cout << "Not Allocated";
        }
        cout << endl;
    }
}

void Best_Fit()
{
    int allocation[total_process];
    for (int i = 0; i < total_process; i++)
    {
        allocation[i] = -1;
    }

    for (int i = 0; i < total_process; i++)
    {
        // Find the best fit block for current process
        int bestIdx = -1;
        for (int j = 0; j < total_blocks; j++)
        {
            if (block_size[j] >= process_size[i])
            {
                if (bestIdx == -1)
                {
                    bestIdx = j;
                }
                else if (block_size[bestIdx] > block_size[j])
                {
                    bestIdx = j;
                }
            }
        }
        if (bestIdx != -1)
        {
            // allocate block j to p[i] process
            allocation[i] = bestIdx;
            // Reduce available memory in this block.
            block_size[bestIdx] -= process_size[i];
```

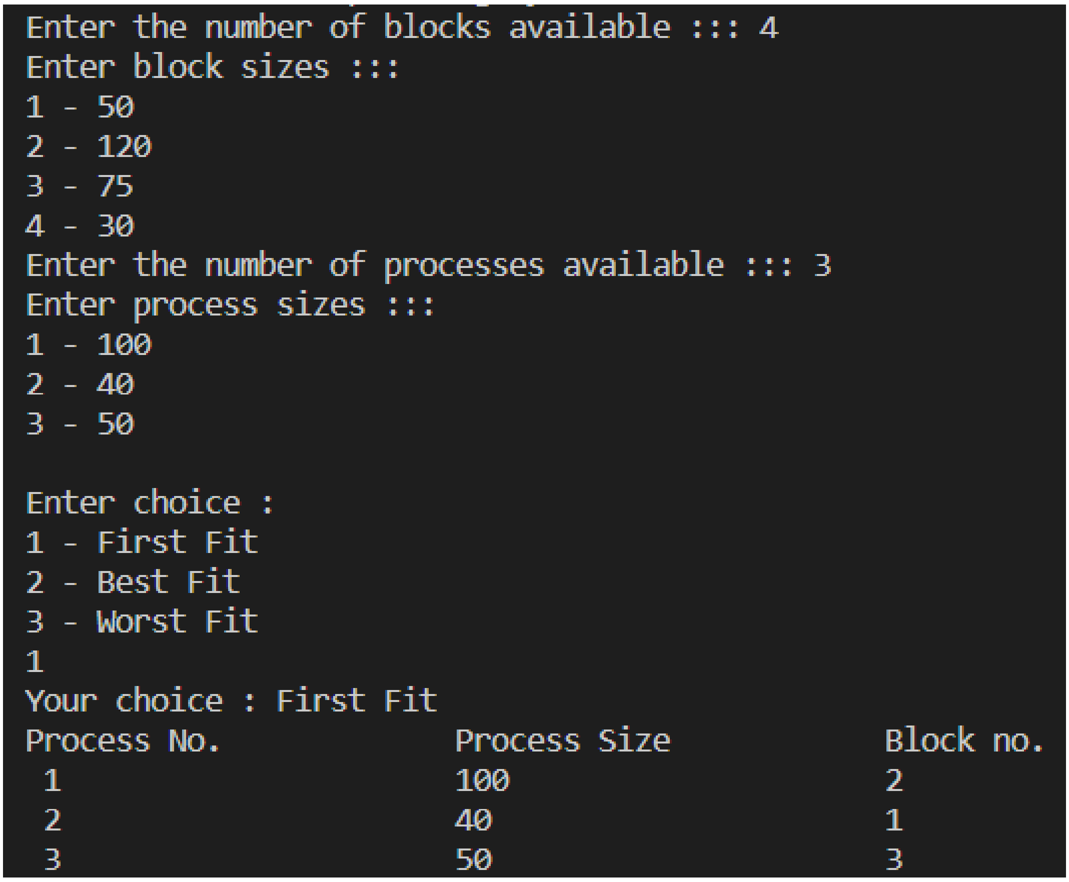


```cpp
        }
    }
    cout << "Process No.\t\tProcess Size\t\tBlock no." << endl;
    for (int i = 0; i < total_process; i++)
    {
        cout << " " << i + 1 << " \t\t\t" << process_size[i] << " \t\t\t";
        if (allocation[i] != -1)
        {
            cout << allocation[i] + 1;
        }
        else
        {
            cout << "Not Allocated";
        }
        cout << endl;
    }
}

void Worst_Fit()
{
    int allocation[total_process];
    for (int i = 0; i < total_process; i++)
    {
        allocation[i] = -1;
    }

    for (int i = 0; i < total_process; i++)
    {
        // Find the best fit block for current process
        int worstIdx = -1;
        for (int j = 0; j < total_blocks; j++)
        {
            if (block_size[j] >= process_size[i])
            {
                if (worstIdx == -1)
                {
                    worstIdx = j;
                }
                else if (block_size[worstIdx] < block_size[j])
                {
                    worstIdx = j;
                }
            }
        }
        if (worstIdx != -1)
        {
            // allocate block j to p[i] process
```

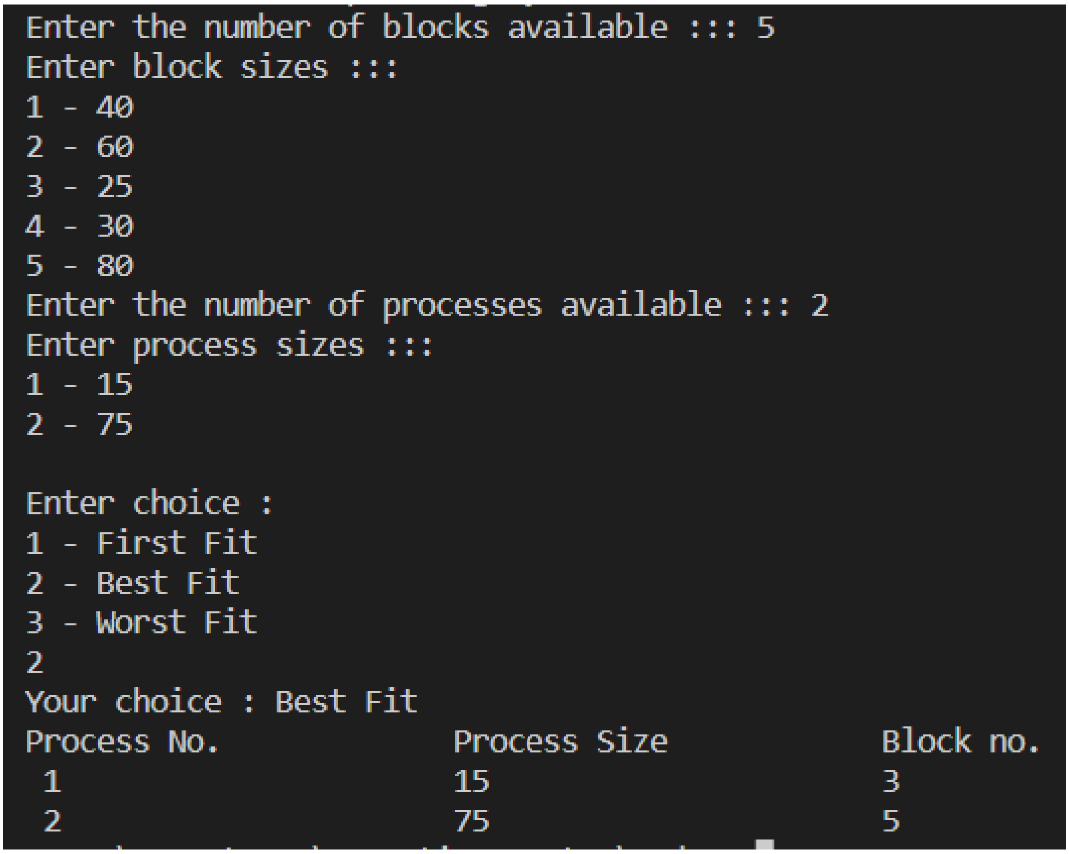
```
                allocation[i] = worstIdx;
                // Reduce available memory in this block.
                block_size[worstIdx] -= process_size[i];
            }
        }
        cout << "Process No.\t\tProcess Size\t\tBlock no." << endl;
        for (int i = 0; i < total_process; i++)
        {
            cout << " " << i + 1 << " \t\t\t" << process_size[i] << " \t\t\t";
            if (allocation[i] != -1)
            {
                cout << allocation[i] + 1;
            }
            else
            {
                cout << "Not Allocated";
            }
            cout << endl;
        }
    }
};

int main()
{
    /*
    blkSize - Array to store Block Sizes
    prcSize - Array to store Process Size
    tblocks - Total number of blocks
    tprc - Total number of process
    */

    int tblocks, tprc;
    cout << "Enter the number of blocks available ::: ";
    cin >> tblocks;

    int blkSize[tblocks];
    cout << "Enter block sizes :::" << endl;
    for (int i = 0; i < tblocks; i++)
    {
        cout << i + 1 << " - ";
        cin >> blkSize[i];
    }

    cout << "Enter the number of processes available ::: ";
    cin >> tprc;

    int prcSize[tprc];
```

```cpp
    cout << "Enter process sizes :::" << endl;
    for (int i = 0; i < tprc; i++)
    {
        cout << i + 1 << " - ";
        cin >> prcSize[i];
    }

    cout << "\nEnter choice : \n1 - First Fit \n2 - Best Fit \n3 - Worst Fit\n";
    int choice;
    cin >> choice;
    MemoryManagementAlgo ob(blkSize, tblocks, prcSize, tprc);
    switch (choice)
    {
    case 1:
    {
        cout << "Your choice : First Fit" << endl;
        ob.First_Fit();
        break;
    }
    case 2:
    {
        cout << "Your choice : Best Fit" << endl;
        ob.Best_Fit();
        break;
    }
    case 3:
    {
        cout << "Your choice : Worst Fit" << endl;
        ob.Worst_Fit();
        break;
    }
    default:
    {
        cout << "Invalid choice" << endl;
        break;
    }
    }

    return 0;
}
```

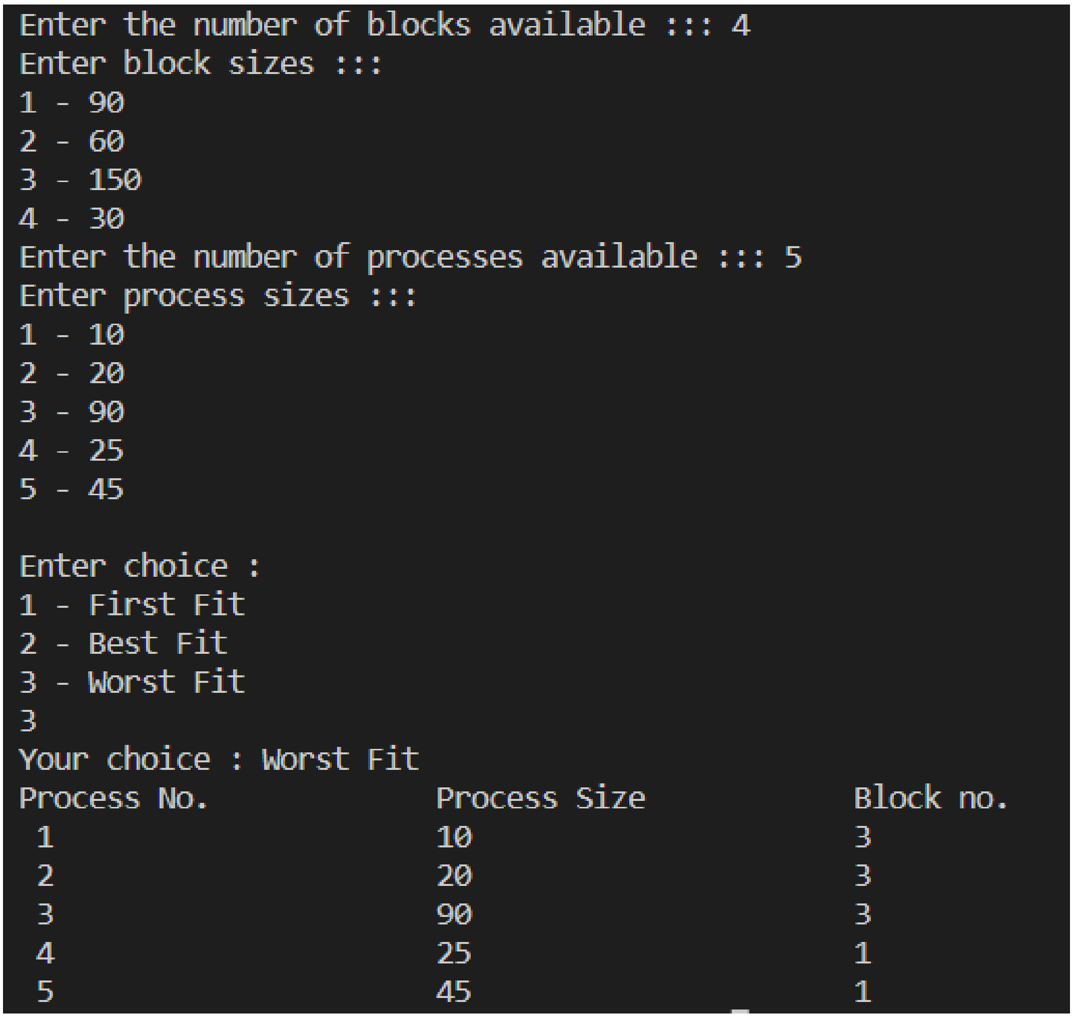
Output 13 :



```
Enter the number of blocks available ::: 4
Enter block sizes :::
1 - 50
2 - 120
3 - 75
4 - 30
Enter the number of processes available ::: 3
Enter process sizes :::
1 - 100
2 - 40
3 - 50

Enter choice :
1 - First Fit
2 - Best Fit
3 - Worst Fit
1
Your choice : First Fit
Process No.             Process Size            Block no.
 1                      100                     2
 2                      40                      1
 3                      50                      3
```



```
Enter the number of blocks available ::: 5
Enter block sizes :::
1 - 40
2 - 60
3 - 25
4 - 30
5 - 80
Enter the number of processes available ::: 2
Enter process sizes :::
1 - 15
2 - 75

Enter choice :
1 - First Fit
2 - Best Fit
3 - Worst Fit
2
Your choice : Best Fit
Process No.             Process Size            Block no.
 1                      15                      3
 2                      75                      5
```



```
Enter the number of blocks available ::: 4
Enter block sizes :::
1 - 90
2 - 60
3 - 150
4 - 30
Enter the number of processes available ::: 5
Enter process sizes :::
1 - 10
2 - 20
3 - 90
4 - 25
5 - 45

Enter choice :
1 - First Fit
2 - Best Fit
3 - Worst Fit
3
Your choice : Worst Fit
Process No.             Process Size            Block no.
 1                      10                      3
 2                      20                      3
 3                      90                      3
 4                      25                      1
 5                      45                      1
```

END OF ASSIGNMENT