आत्मा राम सनातन धर्म महाविद्यालय

Atma Ram Sanatan Dharma College

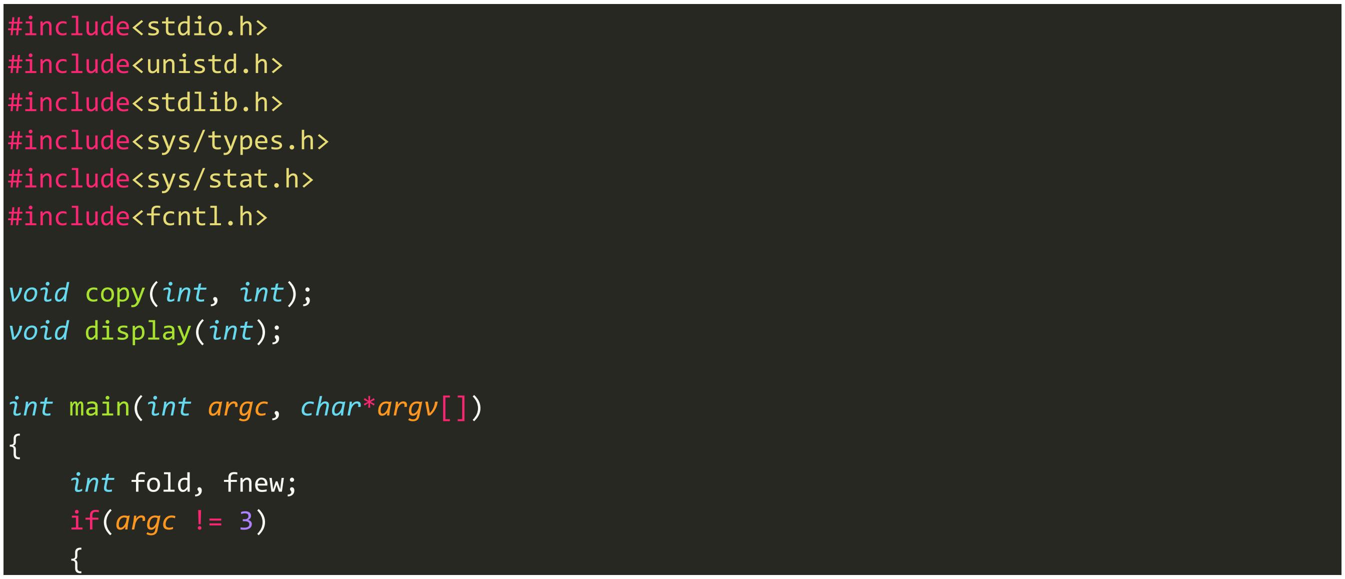
University of Delhi

Atma Ram Sanatan Dharma College
University Of Delhi

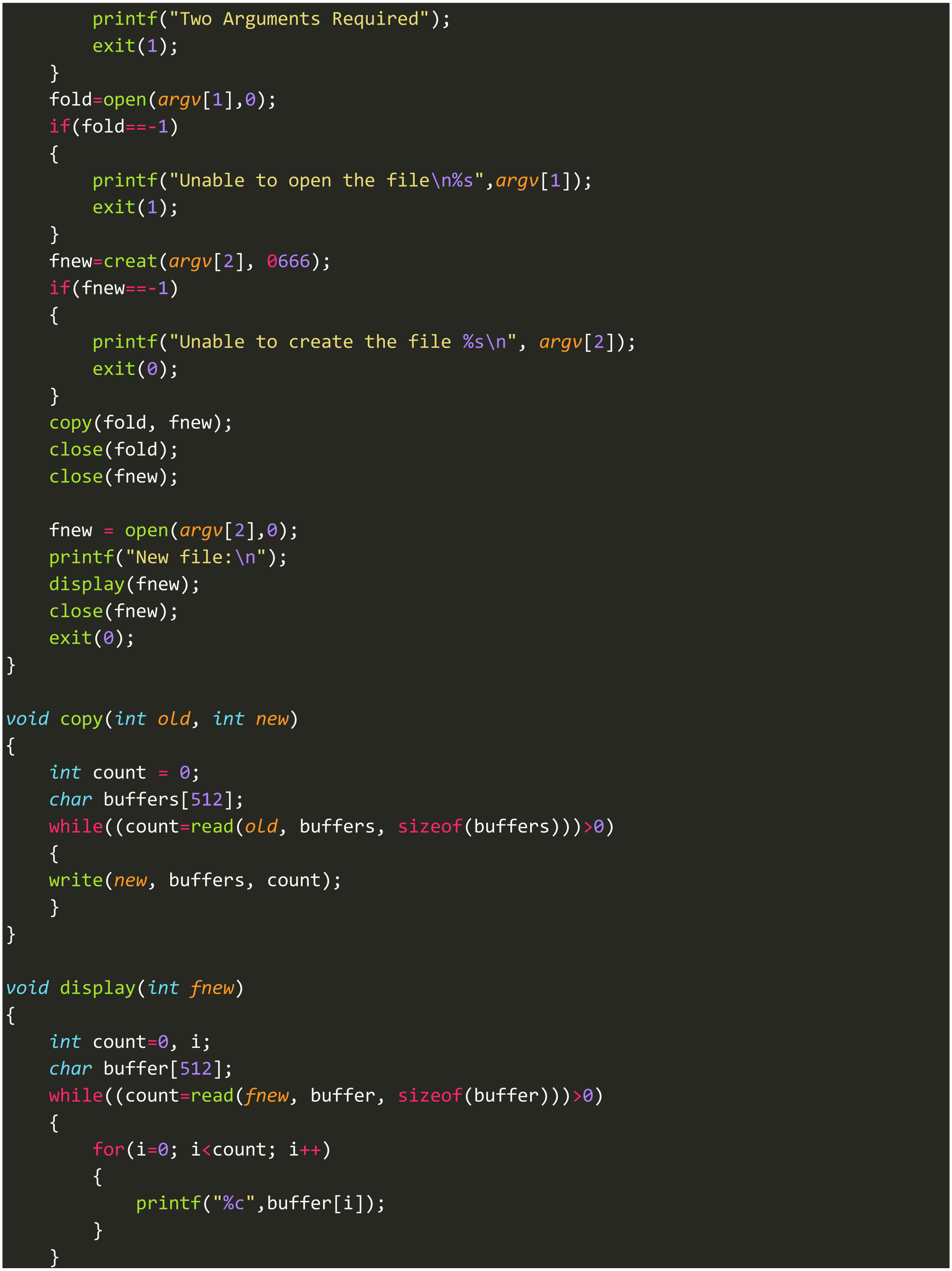
Name	Neeraj
Roll no.	18088
Course	Bsc(H)Computer Science
Submitted to	**Dr. Parul Jain**

Q5. Write a program to copy files using system calls.

Answer

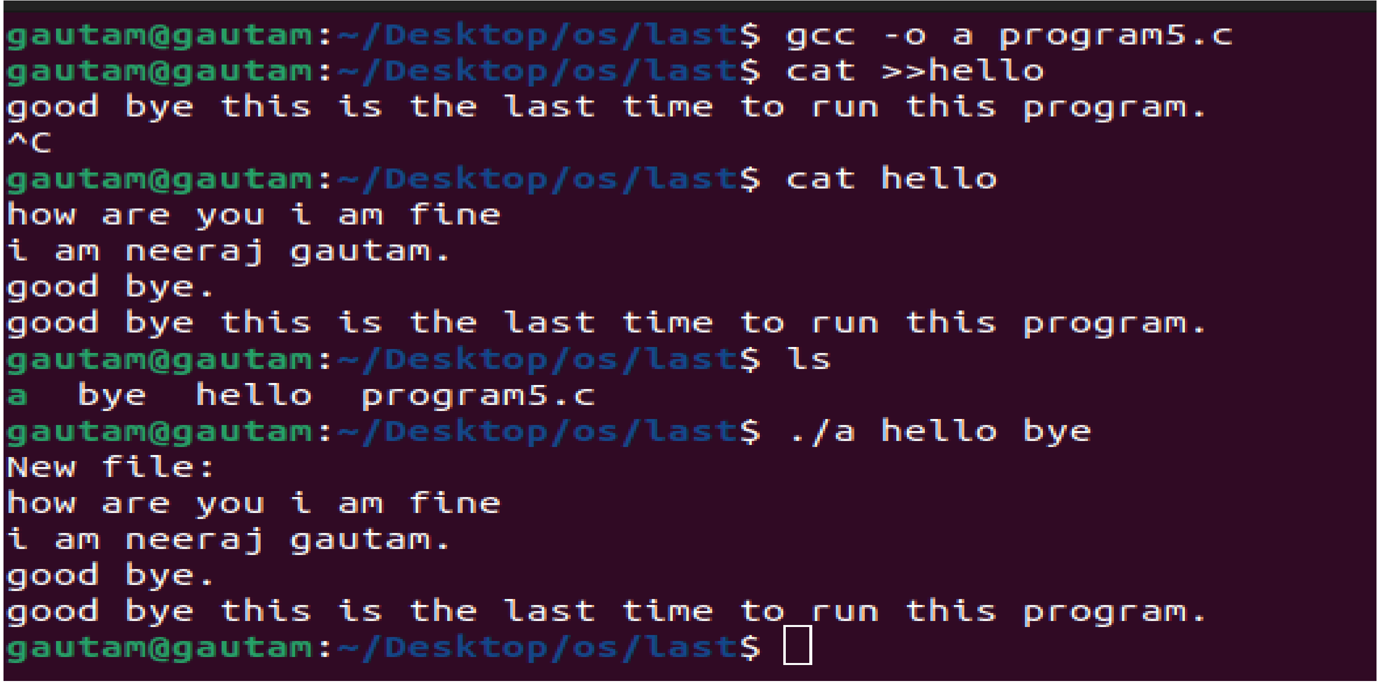


```c
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>

void copy(int, int);
void display(int);

int main(int argc, char*argv[])
{
    int fold, fnew;
    if(argc != 3)
    {
```



```
        printf("Two Arguments Required");
        exit(1);
    }
    fold=open(argv[1],0);
    if(fold==-1)
    {
        printf("Unable to open the file\n%s",argv[1]);
        exit(1);
    }
    fnew=creat(argv[2], 0666);
    if(fnew==-1)
    {
        printf("Unable to create the file %s\n", argv[2]);
        exit(0);
    }
    copy(fold, fnew);
    close(fold);
    close(fnew);

    fnew = open(argv[2],0);
    printf("New file:\n");
    display(fnew);
    close(fnew);
    exit(0);
}

void copy(int old, int new)
{
    int count = 0;
    char buffers[512];
    while((count=read(old, buffers, sizeof(buffers)))>0)
    {
    write(new, buffers, count);
    }
}

void display(int fnew)
{
    int count=0, i;
    char buffer[512];
    while((count=read(fnew, buffer, sizeof(buffer)))>0)
    {
        for(i=0; i<count; i++)
        {
            printf("%c",buffer[i]);
        }
    }
    for(i=0;i<count;i++)
    {
        printf("%c",buffer[i]);
    }

}
```

Output:-

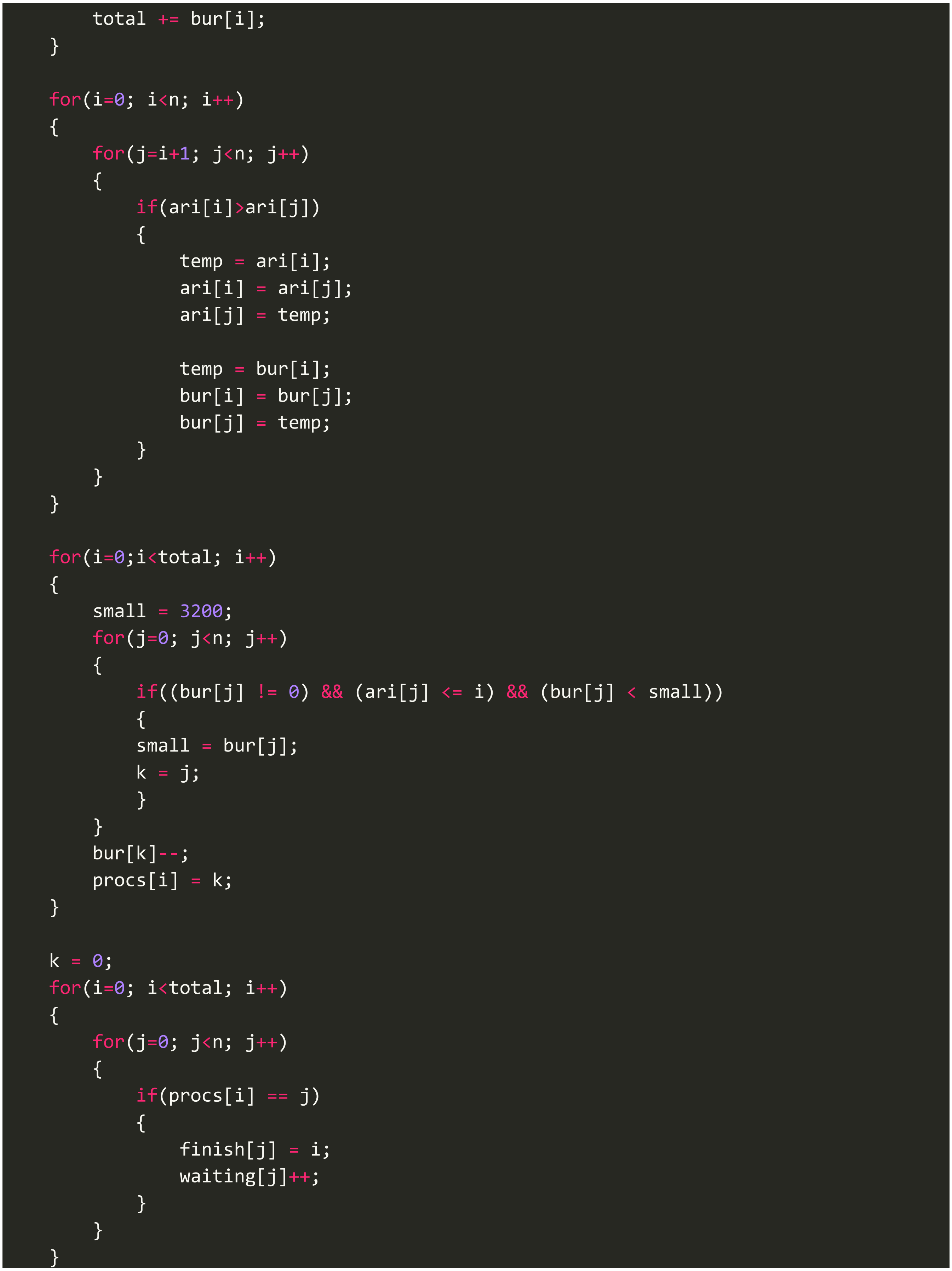


```
gautam@gautam:~/Desktop/os/last$ gcc -o a program5.c
gautam@gautam:~/Desktop/os/last$ cat >>hello
good bye this is the last time to run this program.
^C
gautam@gautam:~/Desktop/os/last$ cat hello
how are you i am fine
i am neeraj gautam.
good bye.
good bye this is the last time to run this program.
gautam@gautam:~/Desktop/os/last$ ls
a  bye  hello  program5.c
gautam@gautam:~/Desktop/os/last$ ./a hello bye
New file:
how are you i am fine
i am neeraj gautam.
good bye.
good bye this is the last time to run this program.
gautam@gautam:~/Desktop/os/last$ 
```

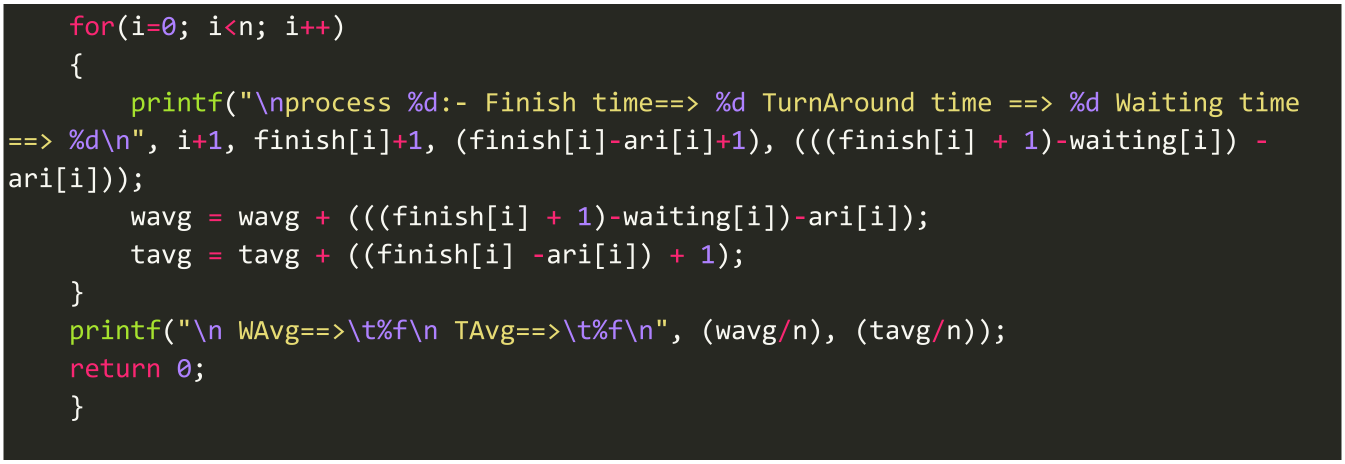
Q11 write a program to implement SRJF scheduling algorithm.

Answer

```
#include<stdio.h>

int main()
{
    int n, ari[10], bur[10], total = 0, i, j, small, temp, procs[100], k, waiting[10],
finish[10];
    float tavg=0.0, wavg=0.0;
    printf("Enter the number of process: ");
    scanf("%d", &n);
    for(i = 0; i<n; i++)
    {
        printf("Enter the arrival time of process %d:\t", i);
        scanf("%d", &ari[i]);
        printf("Enter the burst time of process %d:\t",i);
        scanf("%d", &bur[i]);
        waiting[i] = 0;
```

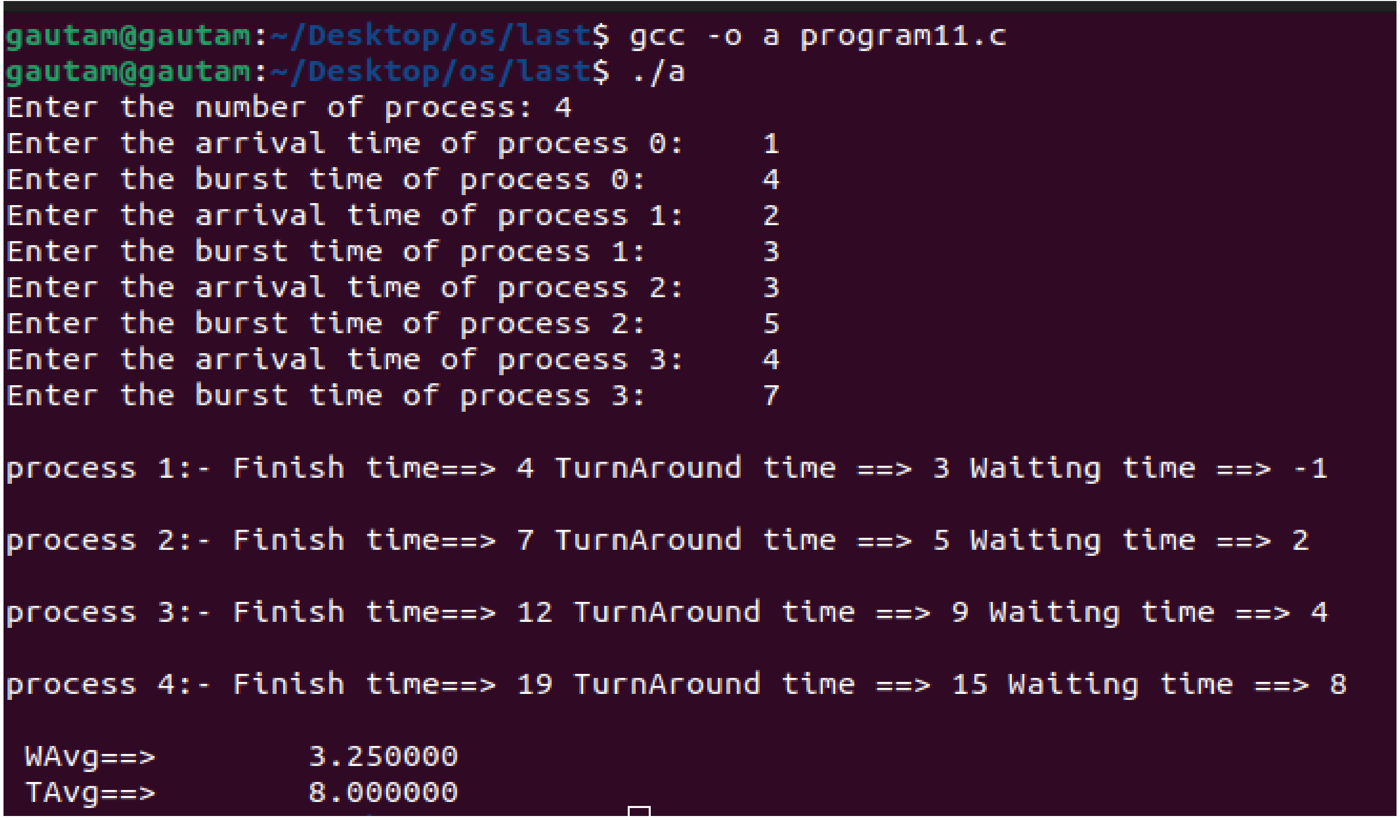


```
        total += bur[i];
    }

    for(i=0; i<n; i++)
    {
        for(j=i+1; j<n; j++)
        {
            if(ari[i]>ari[j])
            {
                temp = ari[i];
                ari[i] = ari[j];
                ari[j] = temp;

                temp = bur[i];
                bur[i] = bur[j];
                bur[j] = temp;
            }
        }
    }

    for(i=0;i<total; i++)
    {
        small = 3200;
        for(j=0; j<n; j++)
        {
            if((bur[j] != 0) && (ari[j] <= i) && (bur[j] < small))
            {
            small = bur[j];
            k = j;
            }
        }
        bur[k]--;
        procs[i] = k;
    }

    k = 0;
    for(i=0; i<total; i++)
    {
        for(j=0; j<n; j++)
        {
            if(procs[i] == j)
            {
                finish[j] = i;
                waiting[j]++;
            }
        }
    }
```



```
    for(i=0; i<n; i++)
    {
        printf("\nprocess %d:- Finish time==> %d TurnAround time ==> %d Waiting time
==> %d\n", i+1, finish[i]+1, (finish[i]-ari[i]+1), (((finish[i] + 1)-waiting[i]) -
ari[i]));
        wavg = wavg + (((finish[i] + 1)-waiting[i])-ari[i]);
        tavg = tavg + ((finish[i] -ari[i]) + 1);
    }
    printf("\n WAvg==>\t%f\n TAvg==>\t%f\n", (wavg/n), (tavg/n));
    return 0;
    }
```

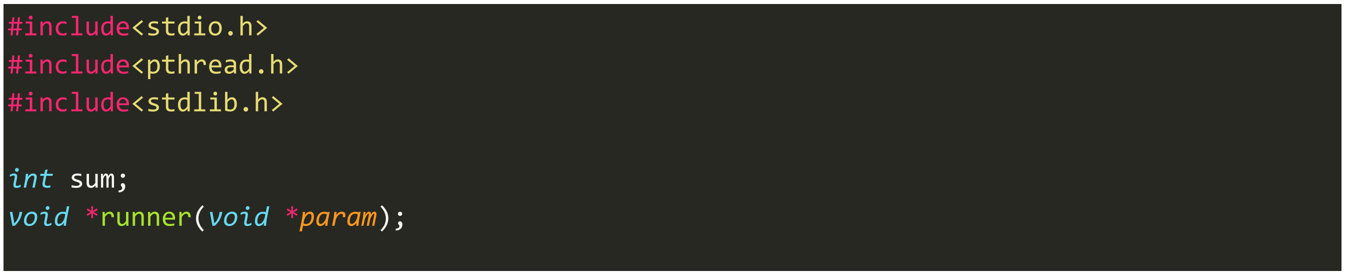
Output:-



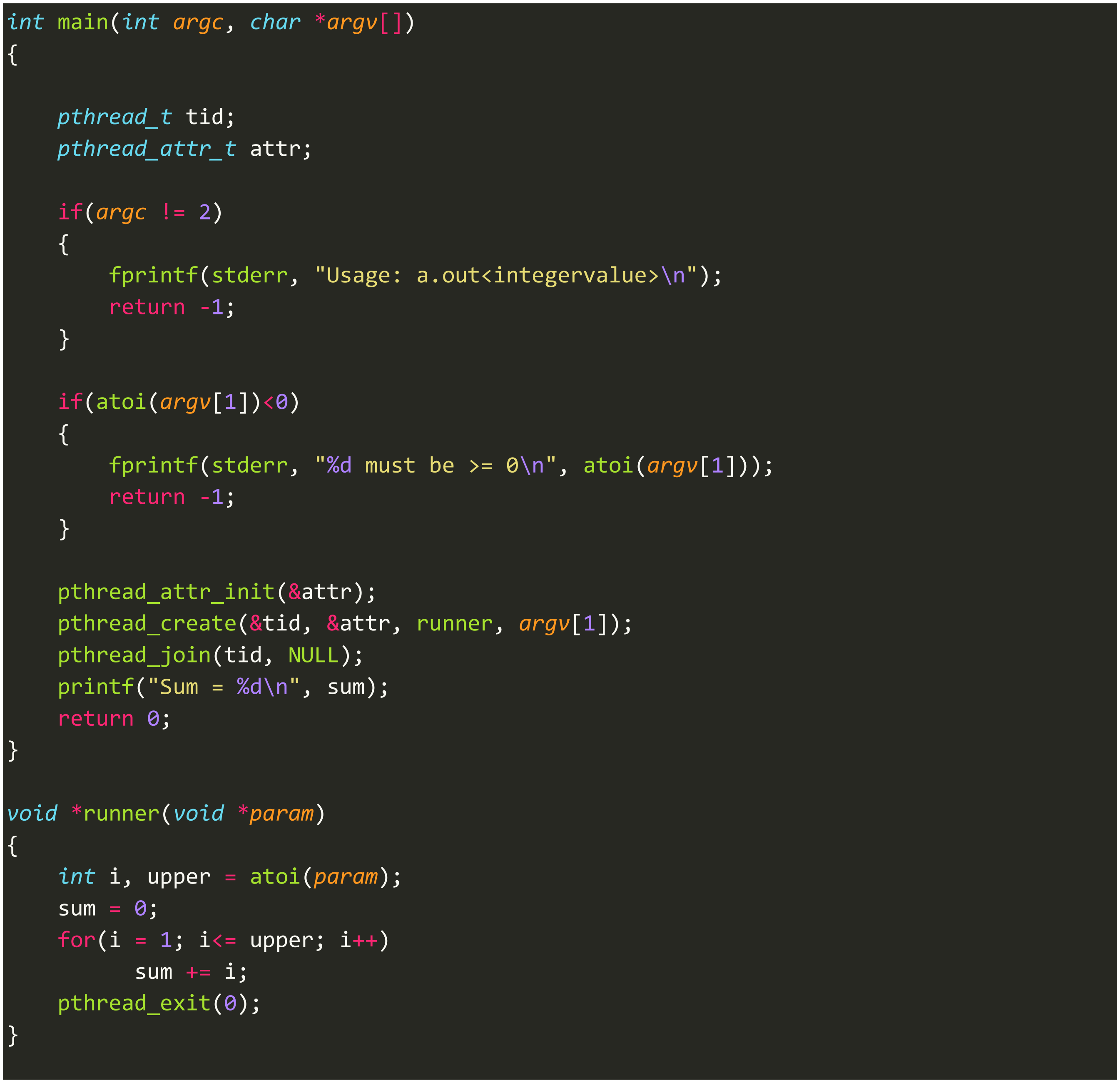
```
gautam@gautam:~/Desktop/os/last$ gcc -o a program11.c
gautam@gautam:~/Desktop/os/last$ ./a
Enter the number of process: 4
Enter the arrival time of process 0:    1
Enter the burst time of process 0:      4
Enter the arrival time of process 1:    2
Enter the burst time of process 1:      3
Enter the arrival time of process 2:    3
Enter the burst time of process 2:      5
Enter the arrival time of process 3:    4
Enter the burst time of process 3:      7

process 1:- Finish time==> 4 TurnAround time ==> 3 Waiting time ==> -1

process 2:- Finish time==> 7 TurnAround time ==> 5 Waiting time ==> 2

process 3:- Finish time==> 12 TurnAround time ==> 9 Waiting time ==> 4

process 4:- Finish time==> 19 TurnAround time ==> 15 Waiting time ==> 8

 WAvg==>        3.250000
 TAvg==>        8.000000
```

Q12 write a program to calculate sum of n numbers using thread library.

Answer

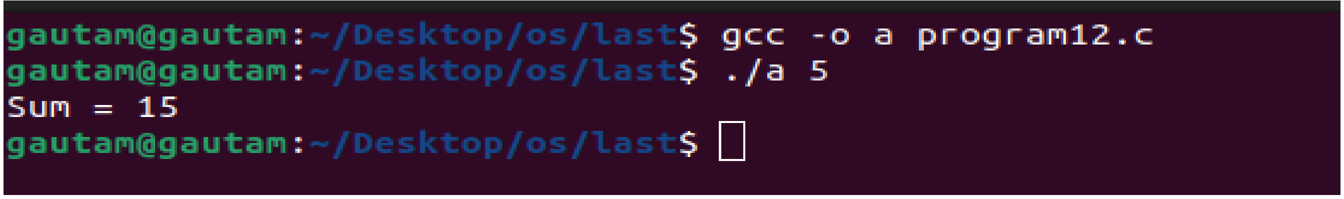


```
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>

int sum;
void *runner(void *param);
```



```c
int main(int argc, char *argv[])
{

    pthread_t tid;
    pthread_attr_t attr;

    if(argc != 2)
    {
        fprintf(stderr, "Usage: a.out<integervalue>\n");
        return -1;
    }

    if(atoi(argv[1])<0)
    {
        fprintf(stderr, "%d must be >= 0\n", atoi(argv[1]));
        return -1;
    }

    pthread_attr_init(&attr);
    pthread_create(&tid, &attr, runner, argv[1]);
    pthread_join(tid, NULL);
    printf("Sum = %d\n", sum);
    return 0;
}

void *runner(void *param)
{
    int i, upper = atoi(param);
    sum = 0;
    for(i = 1; i<= upper; i++)
          sum += i;
    pthread_exit(0);
}
```

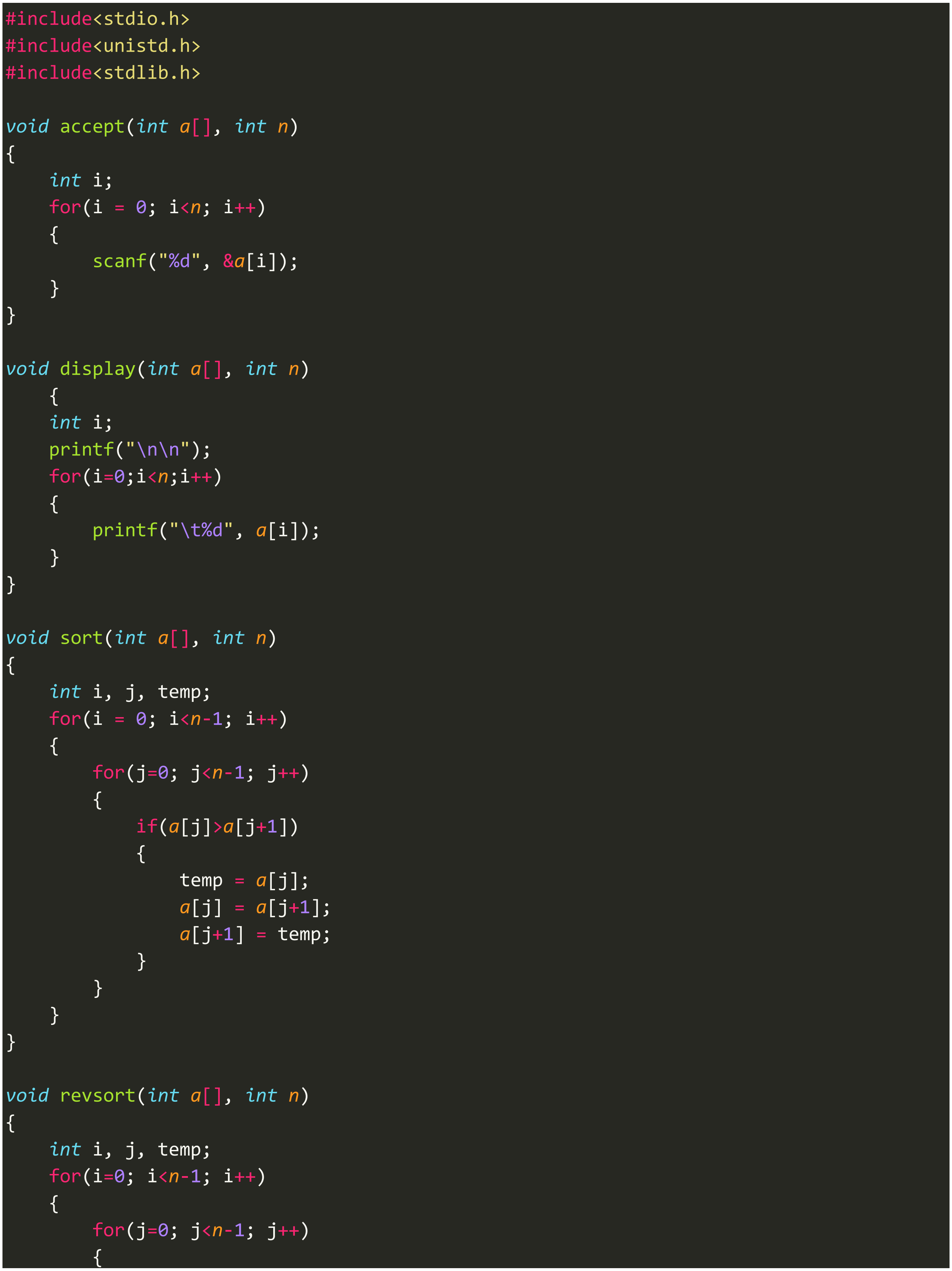
Output:-



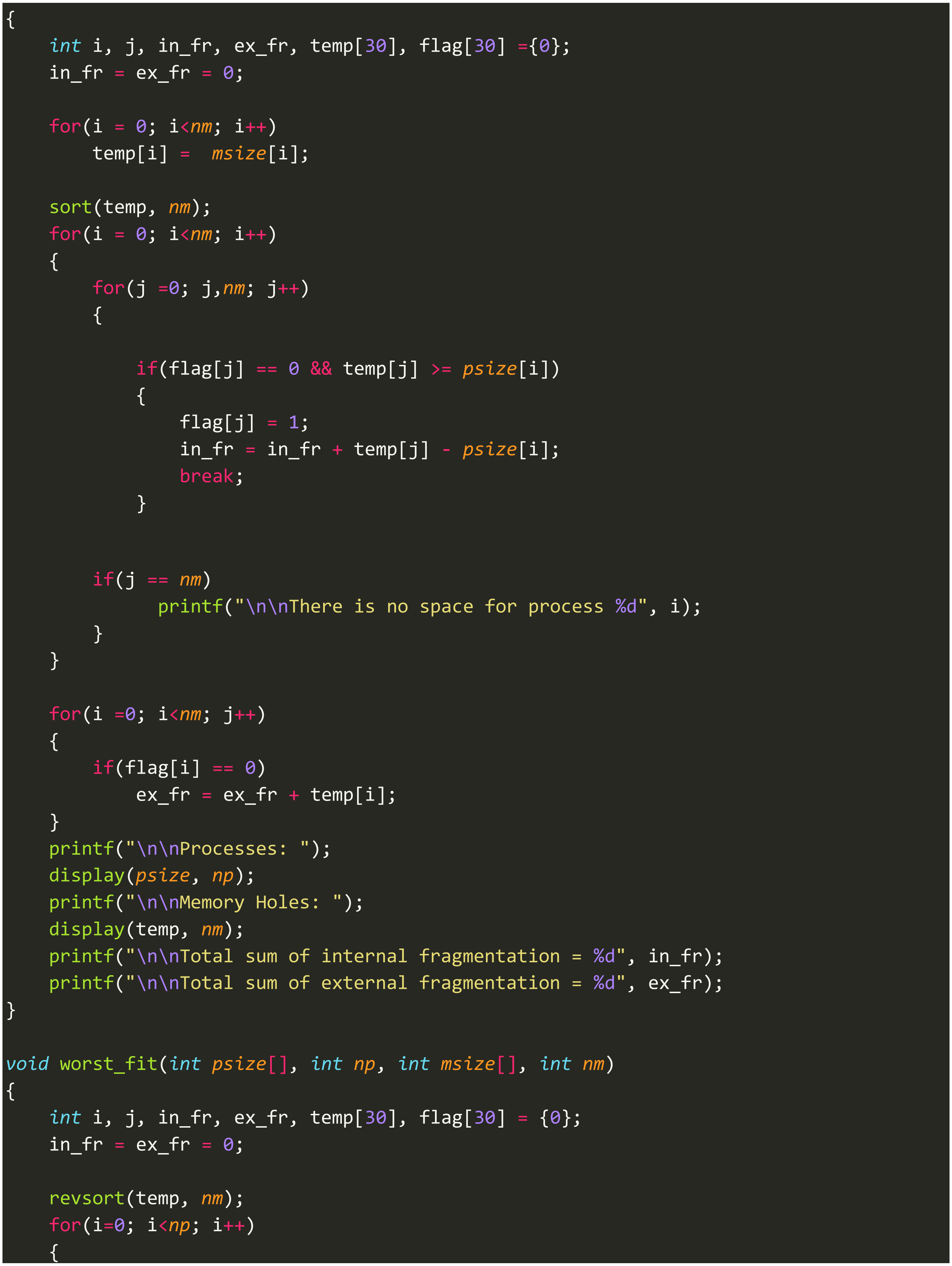
```
gautam@gautam:~/Desktop/os/last$ gcc -o a program12.c
gautam@gautam:~/Desktop/os/last$ ./a 5
Sum = 15
gautam@gautam:~/Desktop/os/last$ 
```

Q13 write a program to implement first-fit, best-fit and worst-fit allocation strategies.

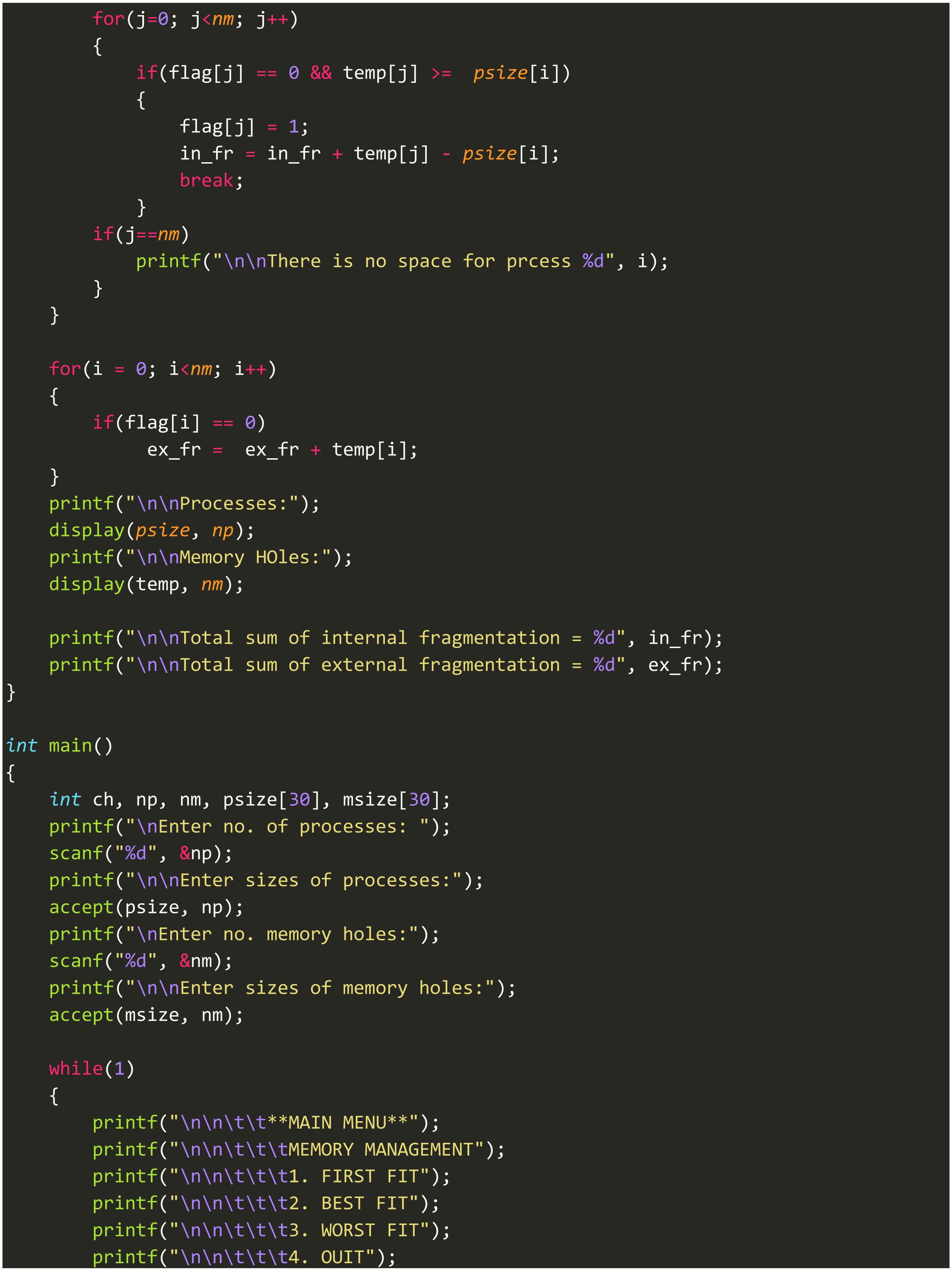
Answer



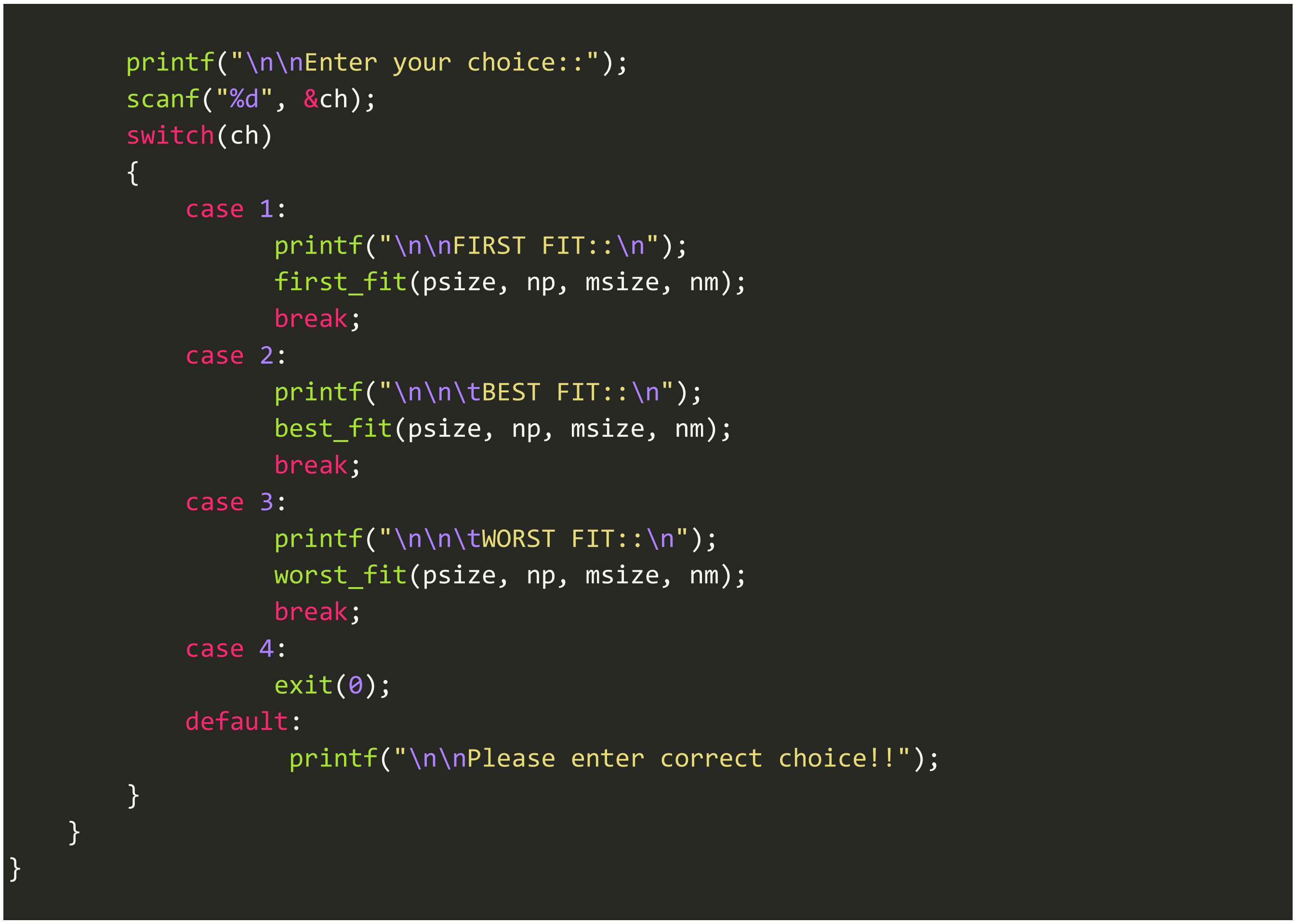
```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>

void accept(int a[], int n)
{
    int i;
    for(i = 0; i<n; i++)
    {
        scanf("%d", &a[i]);
    }
}

void display(int a[], int n)
    {
    int i;
    printf("\n\n");
    for(i=0;i<n;i++)
    {
        printf("\t%d", a[i]);
    }
}

void sort(int a[], int n)
{
    int i, j, temp;
    for(i = 0; i<n-1; i++)
    {
        for(j=0; j<n-1; j++)
        {
            if(a[j]>a[j+1])
            {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
}

void revsort(int a[], int n)
{
    int i, j, temp;
    for(i=0; i<n-1; i++)
    {
        for(j=0; j<n-1; j++)
        {
            if(a[j]<a[j+1])
            {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
}

void first_fit(int psize[], int np, int msize[], int nm)
{
    int i, j, in_fr, ex_fr, flag[30]={0};
    in_fr = ex_fr = 0;

    for(i=0; i<np; i++)
    {
        for(j=0; j<nm; j++)
        {
            if(flag[j] == 0 && msize[j] >= psize[i])
            {
                flag[j] = 1;
                in_fr = in_fr+msize[j]-psize[i];
                break;
            }


        if(j==nm)
             printf("\n\nThere is no space for process %d", i);
        }
    }

    for(i = 0; i<nm; i++)
    {
        if(flag[i] == 0)
            ex_fr = ex_fr + msize[i];
    }

    printf("\n\nProcesses: ");
    display(psize, np);
    printf("\n\nMemory Holes: ");
    display(msize, nm);
    printf("\n\nTotal sum of internal fragmentation = %d", in_fr);
    printf("\n\nTotal sum of external fragmentation = %d", ex_fr);
}

void best_fit(int psize[], int np, int msize[], int nm)
```



```
{
    int i, j, in_fr, ex_fr, temp[30], flag[30] ={0};
    in_fr = ex_fr = 0;

    for(i = 0; i<nm; i++)
        temp[i] =  msize[i];

    sort(temp, nm);
    for(i = 0; i<nm; i++)
    {
        for(j =0; j,nm; j++)
        {

            if(flag[j] == 0 && temp[j] >= psize[i])
            {
                flag[j] = 1;
                in_fr = in_fr + temp[j] - psize[i];
                break;
            }


        if(j == nm)
              printf("\n\nThere is no space for process %d", i);
        }
    }

    for(i =0; i<nm; j++)
    {
        if(flag[i] == 0)
            ex_fr = ex_fr + temp[i];
    }
    printf("\n\nProcesses: ");
    display(psize, np);
    printf("\n\nMemory Holes: ");
    display(temp, nm);
    printf("\n\nTotal sum of internal fragmentation = %d", in_fr);
    printf("\n\nTotal sum of external fragmentation = %d", ex_fr);
}

void worst_fit(int psize[], int np, int msize[], int nm)
{
    int i, j, in_fr, ex_fr, temp[30], flag[30] = {0};
    in_fr = ex_fr = 0;

    revsort(temp, nm);
    for(i=0; i<np; i++)
    {
```

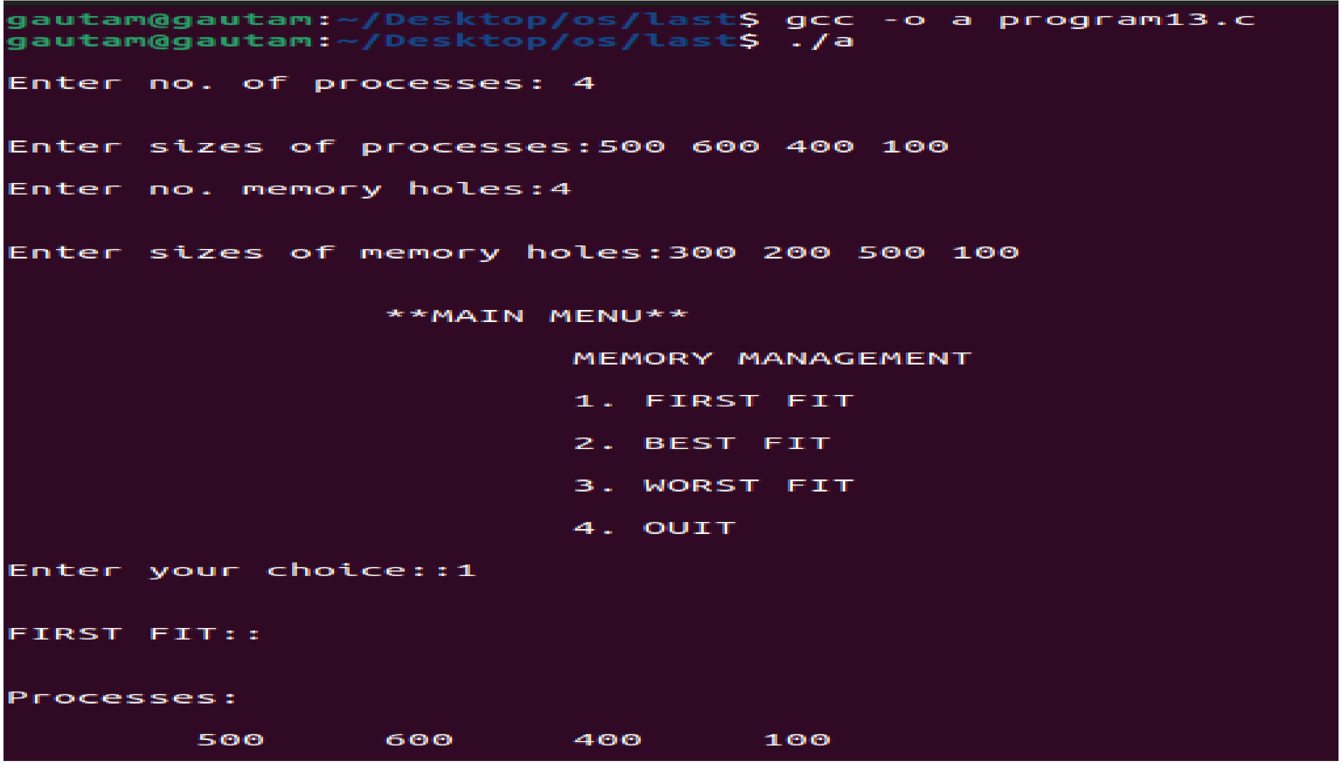


```
        for(j=0; j<nm; j++)
        {
            if(flag[j] == 0 && temp[j] >=  psize[i])
            {
                flag[j] = 1;
                in_fr = in_fr + temp[j] - psize[i];
                break;
            }
        if(j==nm)
            printf("\n\nThere is no space for prcess %d", i);
        }
    }

    for(i = 0; i<nm; i++)
    {
        if(flag[i] == 0)
             ex_fr =  ex_fr + temp[i];
    }
    printf("\n\nProcesses:");
    display(psize, np);
    printf("\n\nMemory HOles:");
    display(temp, nm);

    printf("\n\nTotal sum of internal fragmentation = %d", in_fr);
    printf("\n\nTotal sum of external fragmentation = %d", ex_fr);
}

int main()
{
    int ch, np, nm, psize[30], msize[30];
    printf("\nEnter no. of processes: ");
    scanf("%d", &np);
    printf("\n\nEnter sizes of processes:");
    accept(psize, np);
    printf("\nEnter no. memory holes:");
    scanf("%d", &nm);
    printf("\n\nEnter sizes of memory holes:");
    accept(msize, nm);

    while(1)
    {
        printf("\n\n\t\t**MAIN MENU**");
        printf("\n\n\t\t\tMEMORY MANAGEMENT");
        printf("\n\n\t\t\t1. FIRST FIT");
        printf("\n\n\t\t\t2. BEST FIT");
        printf("\n\n\t\t\t3. WORST FIT");
        printf("\n\n\t\t\t4. QUIT");
```

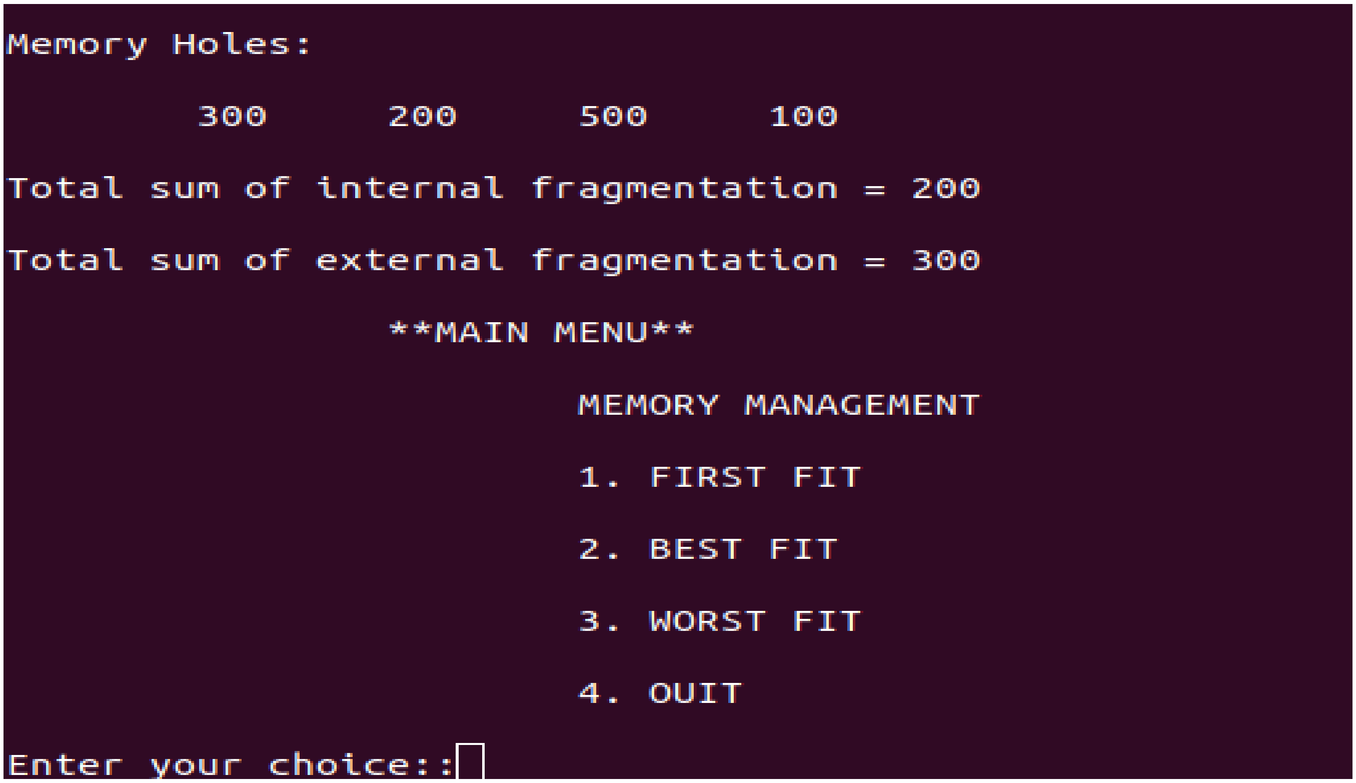


```
        printf("\n\nEnter your choice::");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1:
                  printf("\n\nFIRST FIT::\n");
                  first_fit(psize, np, msize, nm);
                  break;
            case 2:
                  printf("\n\n\tBEST FIT::\n");
                  best_fit(psize, np, msize, nm);
                  break;
            case 3:
                  printf("\n\n\tWORST FIT::\n");
                  worst_fit(psize, np, msize, nm);
                  break;
            case 4:
                  exit(0);
            default:
                   printf("\n\nPlease enter correct choice!!");
        }
    }
}
```

Output:-



```
gautam@gautam:~/Desktop/os/last$ gcc -o a program13.c
gautam@gautam:~/Desktop/os/last$ ./a

Enter no. of processes: 4

Enter sizes of processes:500 600 400 100
Enter no. memory holes:4

Enter sizes of memory holes:300 200 500 100

                **MAIN MENU**
                        MEMORY MANAGEMENT
                        1. FIRST FIT
                        2. BEST FIT
                        3. WORST FIT
                        4. QUIT
Enter your choice::1

FIRST FIT::

Processes:
        500     600     400     100
```



```
Memory Holes:

        300     200     500     100

Total sum of internal fragmentation = 200

Total sum of external fragmentation = 300

                **MAIN MENU**

                        MEMORY MANAGEMENT

                        1. FIRST FIT

                        2. BEST FIT

                        3. WORST FIT

                        4. OUIT

Enter your choice::
```