Day -3

11.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
void print_odd_numbers() {
    printf("Odd numbers: ");
    for (int i = 1; i <= 10; i += 2) {
        printf("%d ", i);
    }
    printf("\n");
}

void print_even_numbers() {
    printf("Even numbers: ");
    for (int i = 2; i <= 10; i += 2) {
        printf("%d ", i);
    }
    printf("\n");
}

void print_multiples_of_3() {
    printf("Multiples of 3: ");
    for (int i = 3; i <= 30; i += 3) {
        printf("%d ", i);
    }
    printf("\n");
}

void print_multiples_of_5() {
    printf("Multiples of 5: ");
    for (int i = 5; i <= 50; i += 5) {
```

```
        printf("%d ", i);
    }
    printf("\n");
}

int main() {
    pid_t pid1, pid2, pid3, pid4;

    // create first child process
    pid1 = fork();

    if (pid1 < 0) {
        fprintf(stderr, "Error creating first child process\n");
        exit(EXIT_FAILURE);
    } else if (pid1 == 0) {
        // first child process
        printf("First child process (pid=%d):\n", getpid());
        print_odd_numbers();
        exit(EXIT_SUCCESS);
    }

    // create second child process
    pid2 = fork();

    if (pid2 < 0) {
        fprintf(stderr, "Error creating second child process\n");
        exit(EXIT_FAILURE);
    } else if (pid2 == 0) {
        // second child process
        printf("Second child process (pid=%d):\n", getpid());
        print_even_numbers();
```

```
    exit(EXIT_SUCCESS);
}

// create third child process
pid3 = fork();

if (pid3 < 0) {
    fprintf(stderr, "Error creating third child process\n");
    exit(EXIT_FAILURE);
} else if (pid3 == 0) {
    // third child process
    printf("Third child process (pid=%d):\n", getpid());
    print_multiples_of_3();
    exit(EXIT_SUCCESS);
}

// create fourth child process
pid4 = fork();

if (pid4 < 0) {
    fprintf(stderr, "Error creating fourth child process\n");
    exit(EXIT_FAILURE);
} else if (pid4 == 0) {
    // fourth child process
    printf("Fourth child process (pid=%d):\n", getpid());
    print_multiples_of_5();
    exit(EXIT_SUCCESS);
}

// wait for all child processes to finish
wait(NULL);
```

```
    wait(NULL);
    wait(NULL);
    wait(NULL);

    return 0;
}
```

Output

```
First child process (pid=9847):
Odd numbers: 1 3 5 7 9
Third child process (pid=9849):
Multiples of 3: 3 6 9 12 15 18 21 24 27 30
Second child process (pid=9848):
Even numbers: 2 4 6 8 10
Fourth child process (pid=9850):
Multiples of 5: 5 10 15 20 25 30 35 40 45 50


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

12. #include<iostream>

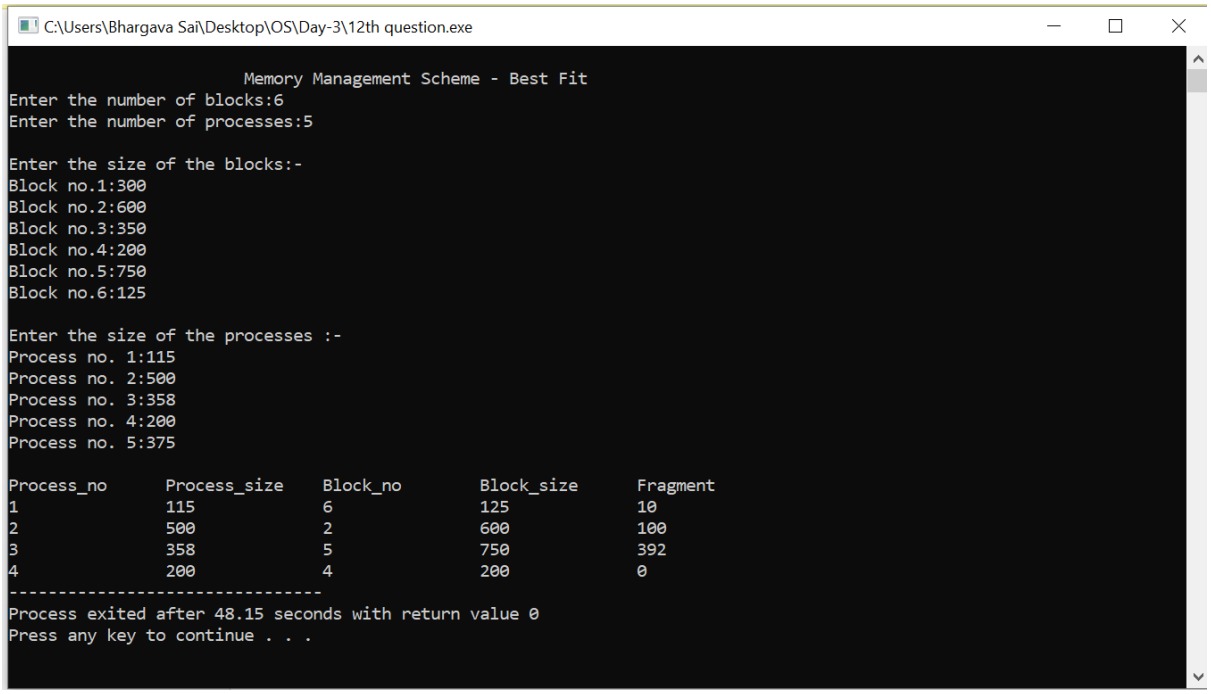
```
using namespace std;

int main()
{
int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
static int barray[20],parray[20];
cout<<"\n\t\t\tMemory Management Scheme - Best Fit";
cout<<"\nEnter the number of blocks:";
cin>>nb;
cout<<"Enter the number of processes:";
cin>>np;
cout<<"\nEnter the size of the blocks:-\n";
for(i=1;i<=nb;i++)
  {
```

```
      cout<<"Block no."<<i<<":";
      cin>>b[i];
   }
cout<<"\nEnter the size of the processes :-\n";
for(i=1;i<=np;i++)
   {
      cout<<"Process no. "<<i<<":";
      cin>>p[i];
   }
for(i=1;i<=np;i++)
{
for(j=1;j<=nb;j++)
{
if(barray[j]!=1)
{
temp=b[j]-p[i];
if(temp>=0)
if(lowest>temp)
{
parray[i]=j;
lowest=temp;
}
}
}
fragment[i]=lowest;
barray[parray[i]]=1;
lowest=10000;
}
cout<<"\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment";
for(i=1;i<=np && parray[i]!=0;i++)
cout<<"\n"<<i<<"\t\t"<<p[i]<<"\t\t"<<parray[i]<<"\t\t"<<b[parray[i]]<<"\t\t"<<fragment[i];
```

return 0;

}

Output



13. #include<stdio.h>

#include<conio.h>

#include<string.h>

int main()

{

int nf=0,i=0,j=0,ch;

char mdname[10],fname[10][10],name[10];

printf("Enter the directory name:");

scanf("%s",mdname);

printf("Enter the number of files:");

scanf("%d",&nf);

do

{

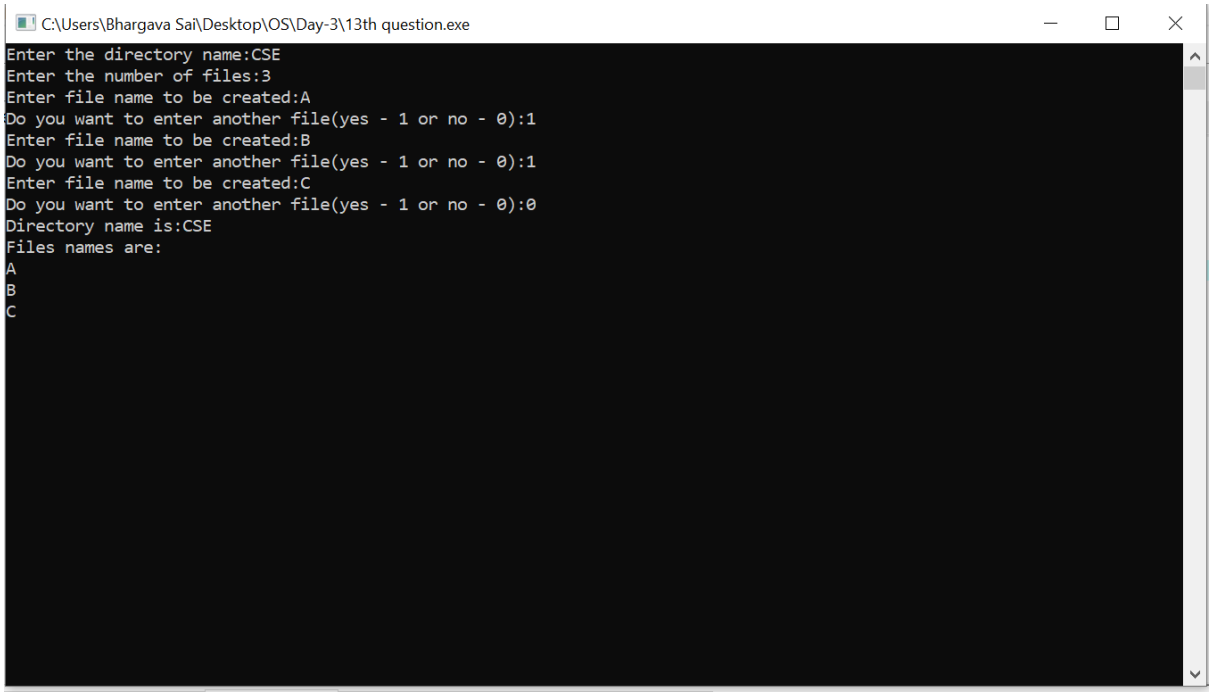
printf("Enter file name to be created:");

scanf("%s",name);

for(i=0;i<nf;i++)

```
{
if(!strcmp(name,fname[i]))
break;
}
if(i==nf)
{
strcpy(fname[j++],name);
nf++;
}
else
printf("There is already %s\n",name);
printf("Do you want to enter another file(yes - 1 or no - 0):");
scanf("%d",&ch);
}
while(ch==1);
printf("Directory name is:%s\n",mdname);
printf("Files names are:");
for(i=0;i<j;i++)
printf("\n%s",fname[i]);
getch();
}
```

Output



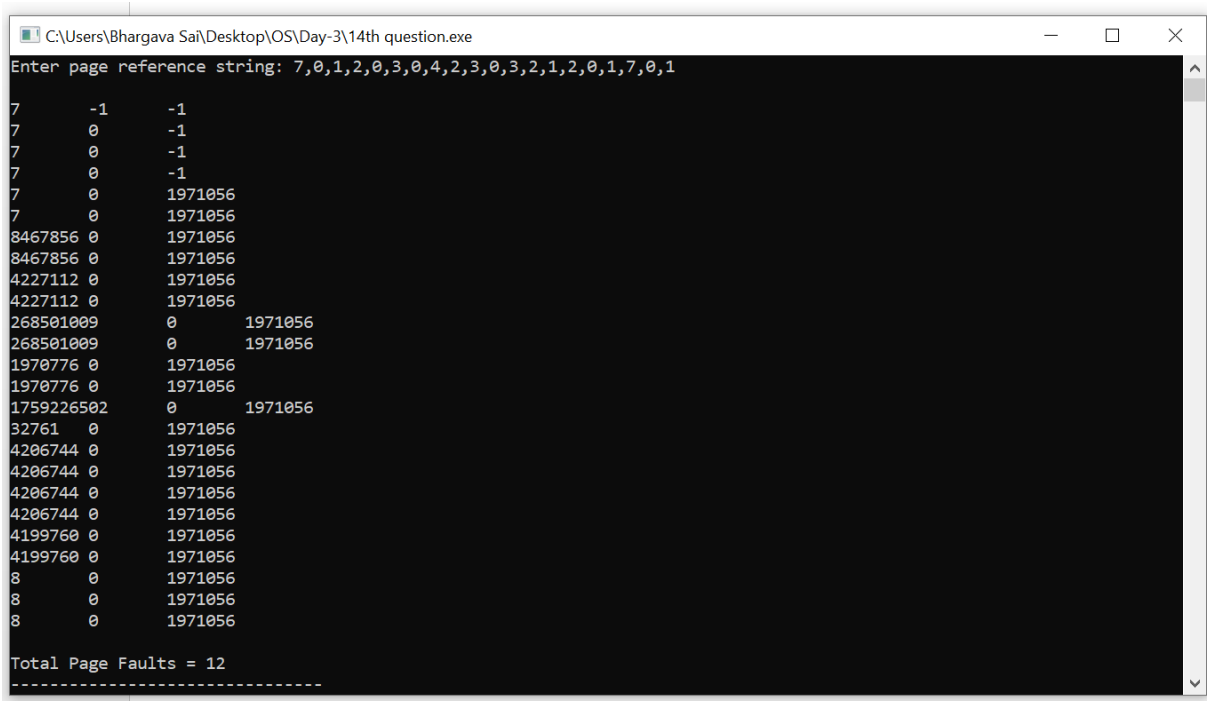
14. #include<stdio.h>

```
int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;
   printf("Enter number of frames: ");
   scanf("%d", &no_of_frames);
   printf("Enter number of pages: ");
   scanf("%d", &no_of_pages);
   printf("Enter page reference string: ");
   for(i = 0; i < no_of_pages; ++i){
      scanf("%d", &pages[i]);
   }
   for(i = 0; i < no_of_frames; ++i){
      frames[i] = -1;
   }
   for(i = 0; i < no_of_pages; ++i){
      flag1 = flag2 = 0;
```

```
for(j = 0; j < no_of_frames; ++j){
   if(frames[j] == pages[i]){
         flag1 = flag2 = 1;
         break;
      }
}
if(flag1 == 0){
   for(j = 0; j < no_of_frames; ++j){
      if(frames[j] == -1){
         faults++;
         frames[j] = pages[i];
         flag2 = 1;
         break;
      }
   }
}
if(flag2 == 0){
 flag3 =0;
   for(j = 0; j < no_of_frames; ++j){
    temp[j] = -1;
    for(k = i + 1; k < no_of_pages; ++k){
    if(frames[j] == pages[k]){
    temp[j] = k;
    break;
    }
    }
   }
   for(j = 0; j < no_of_frames; ++j){
    if(temp[j] == -1){
    pos = j;
    flag3 = 1;
```

```
            break;
            }
           }
           if(flag3 ==0){
            max = temp[0];
            pos = 0;
            for(j = 1; j < no_of_frames; ++j){
            if(temp[j] > max){
            max = temp[j];
            pos = j;
            }
            }
           }
frames[pos] = pages[i];
faults++;
      }

      printf("\n");

      for(j = 0; j < no_of_frames; ++j){
         printf("%d\t", frames[j]);
      }
   }
   printf("\n\nTotal Page Faults = %d", faults);
   return 0;
}
```

Output



15. #include <stdio.h>

#include <math.h>

int size = 9;

void FCFS(int arr[],int head)

{

int seek_count = 0;

int cur_track, distance;

for(int i=0;i<size;i++)

{

cur_track = arr[i];

distance = fabs(head - cur_track);

seek_count += distance;

head = cur_track;

}

printf("Total number of seek operations: %d\n",seek_count);

printf("Seek Sequence is\n");

for (int i = 0; i < size; i++)

{

printf("%d\n",arr[i]);

```
        }
}
int main()
{
        int arr[9] = {55,58,60,70,18,90,150,160,184};
        int head = 50;

        FCFS(arr,head);

        return 0;
}
```

Output