Q.1 Implement the Menu driven Banker's algorithm for accepting Allocation, Max From user.

a)Accept Available

b)Display Allocation, Max

c)Find Need and display It,

d)Display Available

Consider the system with 3 resources types A,B, and C with 7,2,6 instances respectively.Consider the following snapshot:

Process Allocation Request

A B C A B C

P0 0 1 0 0 0 0

P1 4 0 0 5 2 2

P2 5 0 4 1 0 4

P3 4 3 3 4 4 4

P4 2 2 4 6 5 5

Ans:-

```
#include<stdio.h>
#include<unistd.h>
int main()
{
int i,j,process,r,p,finish[10],count=0,safeSequence[10];
printf("Enter the no of processes :");
scanf("%d",&p);
for(i=0;i<p;i++)
 {
finish[i]=0;
}
printf("Enter the no of resourses:");
scanf("%d",&r);
int available[r];
printf("Enter the available matrix ");
for(i=0;i<r;i++)
{
scanf("%d",&available[i]);
}
printf("\nEnter the allocated matrix :"); int allocation[10][10];
for(i=0;i<p;i++)
{
printf("\nfor the process %d :",i+1);
 for(j=0;j<r;j++)
{
scanf(" %d",&allocation[i][j]);
}
}
printf("\nEnter the maximum matrix :"); int Max[10][10];
for(i=0;i<p;i++)
```

```
{
printf("\nfor the process %d:\n",i+1);
for(j=0;j<r;j++)
{
scanf(" %d",&Max[i][j]);
}
}
 printf("\nThe available matrix is\n ");
for(i=0;i<r;i++)
{
printf("\t r[%d]",i);
}
printf("\n");
for(i=0;i<r;i++)
{
printf("\t %d",available[i]);
}
printf("\nThe allocation matrix is :\n");
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
{
printf(" \t %d ",allocation[i][j]);
}
 printf("\n");
}
printf("\nThe max matrix is :\n");
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
{
printf("\t %d",Max[i][j]);
}
printf("\n");
}
int need[10][10];
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
{
 need[i][j]=Max[i][j]-allocation[i][j]
}
}
printf("The need matrix is :\n");
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
```

```
{
printf("\t %d",need[i][j]);
}
printf("\n");
}
do{
int process=-1;
for(i=0;i<p;i++)
{
if(finish[i]==0)
 {
process=i;
for(j=0;j<r;j++)
{
if(available[j]<need[i][j])
{
process=-1;
break;
}
}
if(process!=-1)
{
printf("\nprocess p%d granted required resources ",process);
safeSequence[count]=process; count++;
for(j=0;j<r;j++)
{
 available[j]+=allocation[process][j];
allocation[process][j]=0;
Max[process][j]=0;
finish[process]=1;
}
}
}
}
}
while(count!=p && process!=-1);
if(count==p)
{
printf("\nThe system is in safe state :\n"); printf("safe sequence :<"); for(i=0;i<p;i+
+)
printf("p%d ",safeSequence[i]);
printf(">\n");
}
else{
 printf("\n The system is in an unsafe state :");
}
int req[10],proc;
```

```
printf("Enter the requested process :\n");
scanf("%d",&proc);
printf("Enter the request\n");
for( i=0;i<r;i++)
{
scanf("%d ",&req[i]);
}
if(req[i]>available[i])
{
printf("The request is not granted immediately \n");
}
else
{
printf("The request is granted immediately\n ");
 }
```

Q.1) Write a C Menu driven Program to implement following functionality

a) Accept Available

b) Display Allocation, Max

c) Display the contents of need matrix

d) Display Available

Process Allocation Max Available

A B C A B C A B C

P0 2 3 2 9 7 5 3 3 2

P1 4 0 0 5 2 2

P2 5 0 4 1 0 4

P3 4 3 3 4 4 4

P4 2 2 4

Ans:-

```
#include<stdio.h>
#include<unistd.h>
int main()
{
int i,j,k,res,pno;
printf("Enter the no of resourses:");
scanf("%d",&res);
printf("Enter the no of processes :");
scanf("%d",&pno);
int available[res];
printf("Enter the available matrix ");
for(i=0;i<res;i++)
{
scanf("%d",&available[i]);
}
printf("The available matrix is\n ");
for(i=0;i<res;i++)
```

```
{
printf("\t r[%d]",i);
}
printf("\n");
for(i=0;i<res;i++)
{
printf("\t %d",available[i]);
}
printf("\nEnter the allocated matrix :");
 int allocation[10][10];
for(i=0;i<pno;i++)
{
for(j=0;j<res;j++)
{
scanf(" %d",&allocation[i][j]);
}
}
printf("The allocation matrix is :\n");
for(i=0;i<pno;i++)
{
for(j=0;j<res;j++)
{
printf(" \t %d ",allocation[i][j]);
}
printf("\n");
}
printf("\nEnter the maximum matrix :"); int Max[10][10];
for(i=0;i<pno;i++)
{
for(j=0;j<res;j++)
{
scanf(" %d",&Max[i][j]);
}
}
rintf("The max matrix is :\n"); for(i=0;i<pno;i++)
{
for(j=0;j<res;j++)
{
printf("\t %d",Max[i][j]);
}
printf("\n");
}
int need[10][10];
for(i=0;i<pno;i++)
{
for(j=0;j<res;j++)
{
```

```
need[i][j]=Max[i][j]-allocation[i][j];
}
}
printf("The need matrix is :\n"); for(i=0;i<pno;i++)
{
for(j=0;j<res;j++)
{
printf("\t %d",need[i][j]);
}
printf("\n");
}
```

Q.1 Write a program to simulate Linked file allocation method. Assume disk with n number of blocks. Give value of n as input. Randomly mark some block as allocated and accordingly maintain the list of free blocks Write menu driver program with menu options as mentioned below and implement each option.

- Show Bit Vector
- Create New File
- Show Directory
- Exit

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX 200
typedef struct dir
{
char fname[20];
int start;
struct dir *next;
}NODE;
NODE *first,*last;
int n,fb,bit[MAX];
void init()
{
int i;
printf("Enter total no.of disk blocks:");
 scanf("%d",&n);
fb = n;
for(i=0;i<fb;i++)
bit[i] = 0;
printf("\n");
}
void show_bitvector()
{
int i;
for(i=0;i<n;i++)
```

```
printf("%d ",bit[i]);
printf("\n");
}
void show_dir()
{
NODE *p;
int i;
D
Doprintf("File\t\t\tNumber of blocks\n");
p = first;
while(p!=NULL)
{
printf("%s\t",p->fname);
i = p->start; while(i!=-1)
{
printf("%d->",i); i=bit[i];
}
printf("NULL\n");
p=p->next;
}
}
void create()
{
NODE *p;
char fname[20];
int i,j,k,nob;
printf("Enter file name:");
scanf("%s",fname);
printf("Enter no.of blocks:");
scanf("%d",&nob);
if(nob>fb)
{
printf("Failed to create file %s\n",fname);
return;
}
for(i=0;i<n;i++)
{
if(bit[i]==0) break;
}
p = (NODE*)malloc(sizeof(NODE));
strcpy(p->fname,fname);
p->start=i;
p->next=NULL;
if(first==NULL)
first=p;
else
last->next=p;
```

```
last=p;
fb-=nob;
j=i+1;
nob--;
while(nob>0)
{
if(bit[j]==0)
{
bit[i]=j;
i=j;
nob--;
}
j++;
}
bit[i]=-1;
printf("File %s created successully.\n",fname);
}
void delete()
{
char fname[20];
NODE *p,*q;
int nob=0,i,j;
printf("Enter file name to be deleted:"); scanf("%s",fname);
p = q = first;
while(p!=NULL)
{
if(strcmp(p->fname,fname)==0)
break;
q=p;
p=p->next;
}
if(p==NULL)
{
printf("File %s not found.\n",fname);
return;
}
i = p->start;
while(i!=-1)
{
nob++; j = i;
i = bit[i]; bit[j] = 0;
}
fb+=nob;
if(p==first)
first=first->next;
else if(p==last)
{
```

```
last=q;
last->next=NULL;
}
else
q->next = p->next;
free(p);
printf("File %s deleted successfully.\n",fname);
}
int main()
{
int ch;
init();
while(1)
{
printf("1.Show bit vector\n");
printf("2.Create new file\n");
printf("3.Show directory\n");
printf("4.Delete file\n");
 printf("5.Exit\n");
printf("Enter your choice (1-5):");
scanf("%d",&ch);
switch(ch)
{
case 1:
show_bitvector();
break;
case 2:
create();
break;
case 3:
show_dir();
break;
case 4:
delete();
break;
case 5:
exit(0);
}
}
return 0;
}
```

Q.1 Write a program to simulate Contiguous file allocation method. Assume disk with n number of blocks. Give value of n as input. Randomly mark some block as allocated and accordingly maintain the list of free blocks Write menu driver program with menu options as mentioned above and implement each option.

• Show Bit Vector
• Create New File
• Show Directory
• Exit

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX_BLOCKS 100
int disk[MAX_BLOCKS];
int num_blocks;
int free_blocks;
int *directory;
void init_disk() {
 for (int i = 0; i < num_blocks; i++) {
 disk[i] = rand() % 2; // randomly mark blocks as allocated or free
 if (disk[i] == 0) {
 free_blocks++;
 }
 }
}
void init_directory() {
 directory = (int *)malloc(num_blocks * sizeof(int));
 for (int i = 0; i < num_blocks; i++) {
 directory[i] = -1; // initialize directory with -1
 }
}
void show_bit_vector() {
 printf("Bit Vector:\n");
 for (int i = 0; i < num_blocks; i++) {
 printf("%d ", disk[i]);
 }
 printf("\n");
}
void show_directory() {
 printf("Directory:\n");
 for (int i = 0; i < num_blocks; i++) {
 printf("%d ", directory[i]);
 }
 printf("\n");
}
void delete_file(int start_block) {
 int current_block = start_block;
 while (current_block != -1) {
 disk[current_block] = 0;
 free_blocks++;
 current_block = directory[current_block];
 directory[current_block] = -1;
```

```
 }
 printf("File deleted successfully.\n");
}
int main() {
 printf("Enter the number of blocks in the disk: ");
 scanf("%d", &num_blocks);
 init_disk();
 init_directory();

int choice, start_block;
 bool running = true;
 while (running) {
 printf("\n1. Show Bit Vector\n2. Show Directory\n3. Delete File\n4. Exit\n");
 printf("Enter your choice: ");
 scanf("%d", &choice);
 switch (choice) {
 case 1:
 show_bit_vector();
 break;
 case 2:
 show_directory();
 break;
 case 3:
 printf("Enter the starting block of the file: ");
 scanf("%d", &start_block);
 delete_file(start_block);
 break;
 case 4:
 running = false;
 break;
 default:
 printf("Invalid choice. Try again.\n");
 }
 }
 return Exit
```

.1 Write a program to simulate Indexed file allocation method. Assume disk with n number of blocks. Give value of n as input. Randomly mark some block as allocated and accordingly maintain the list of free blocks Write menu driver program with menu options as mentioned above and implement each option.

- Show Bit Vector
- Show Directory
- Delete Already File
- Exit

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

```
#include <stdlib.h>
#include <stdbool.h>
#define MAX_BLOCKS 100
typedef struct {
 bool allocated;
 int index;
} Block;
int main() {
 int n;
 printf("Enter the number of blocks on the disk: ");
 scanf("%d", &n);
 // Initialize the bit vector and directory
 Block bit_vector[MAX_BLOCKS];
 int directory[MAX_BLOCKS];
 for (int i = 0; i < n; i++) {
 bit_vector[i].allocated = false;
 bit_vector[i].index = -1;
 directory[i] = -1;
 }
 // Mark some blocks as allocated
 for (int i = 0; i < n / 2; i++) {
 int block_index = rand() % n;
 bit_vector[block_index].allocated = true;
 bit_vector[block_index].index = i;
 directory[i] = block_index;
 }
 int choice;
 do {

printf("\n1. Show Bit Vector\n");
 printf("2. Show Directory\n");
 printf("3. Delete Already File\n");
 printf("4. Exit\n");
 printf("Enter your choice: ");
 scanf("%d", &choice);
 switch (choice) {
 case 1:
 // Show the bit vector
 printf("Bit Vector:\n");
 for (int i = 0; i < n; i++) {
 printf("%d ", bit_vector[i].allocated);
 }
 printf("\n");
 break;
 case 2:
 // Show the directory
 printf("Directory:\n");
```

```
 for (int i = 0; i < n; i++) {
 if (directory[i] != -1) {
 printf("%d -> %d\n", i, directory[i]);
 }
 }
 break;
 case 3:
 // Delete a file
 int file_index;
 printf("Enter the file index to delete: ");
 scanf("%d", &file_index);

if (directory[file_index] == -1) {
 printf("File not found.\n");
 } else {
 int block_index = directory[file_index];
 bit_vector[block_index].allocated = false;
 bit_vector[block_index].index = -1;
 directory[file_index] = -1;
 printf("File deleted successfully.\n");
 }
 break;
 case 4:
 // Exit the program
 printf("Exiting...\n");
 break;
 default:
 printf("Invalid choice. Please try again.\n");
 }
 } while (choice != 4);
 return 0;
}
```

Q.2 Write a simulation program for disk scheduling using FCFS algorithm.
Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.

55, 58, 39, 18, 90, 160, 150, 38, 184
Start Head Position: 50

Ans:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int RQ[100],i,n,TotalHeadMovement=0,initial;
printf("Enter the number of Requests\n");
scanf("%d",&n);
```

```
printf("Enter the Requests sequence\n");
for(i=0;i<n;i++)
scanf("%d",&RQ[i]);
printf("Enter initial head position\n");
scanf("%d",&initial);
for(i=0;i<n;i++)
{
TotalHeadMovement=TotalHeadMovement+abs(RQ[i]-initial);
initial=RQ[i];
}
printf("Total head Movement is %d\n",TotalHeadMovement);
return 0;
}
```

Q.2 Write a simulation program for disk scheduling using SSTF algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.

186, 89, 44, 70, 102, 22, 51, 124

Start Head Position: 70

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int RQ[100],i,n,TotalHeadMoment=0,initial,count=0; printf("Enter the number of
Requests\n"); scanf("%d",&n);
printf("Enter the Requests sequence\n");
for(i=0;i<n;i++)
scanf("%d",&RQ[i]);
printf("Enter initial head position\n"); scanf("%d",&initial);
while(count!=n)
{
int min=1000,d,index;
for(i=0;i<n;i++)
{
d=abs(RQ[i]-initial);
if(min>d)
{
min=d;
 index=i;
}
}
TotalHeadMoment=TotalHeadMoment+min;
initial=RQ[index];
RQ[index]=1000;
count++;
```

```
}
printf("Total head movement is %d",TotalHeadMoment); return 0;
}
```

1)Scan

Write a simulation program for disk scheduling using SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.

86, 147, 91, 170, 95, 130, 102, 70

Starting Head position= 125

Direction: Left

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
 printf("Enter the number of Requests\n");
scanf("%d",&n);
printf("Enter the Requests sequence\n");
for(i=0;i<n;i++)
scanf("%d",&RQ[i]);
printf("Enter initial head position\n");
 scanf("%d",&initial);
printf("Enter total disk size\n");
scanf("%d",&size);
printf("Enter the head movement direction for high 1 and for low 0\n");
scanf("%d",&move);
for(i=0;i<n;i++)
{
for(j=0;j<n-i-1;j++)
{
if(RQ[j]>RQ[j+1])
 {
int temp;
temp=RQ[j];
RQ[j]=RQ[j+1];
RQ[j+1]=temp;
}
}
}
int index;
for(i=0;i<n;i++)
{
if(initial<RQ[i])
{
index=i;
```

```
break;
}
}
 if(move==1)
{
for(i=index;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];
}
TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
initial = size-1;
for(i=index-1;i>=0;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];
}
}
 else
{
for(i=index-1;i>=0;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];
}
TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
initial =0;
for(i=index;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];
}
}
 printf("Total head movement is %d",TotalHeadMoment);
return 0;
```

Q.2 Write a simulation program for disk scheduling using C-SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments..

80, 150, 60,135, 40, 35, 170

Starting Head Position: 70

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move; printf("Enter the number
of Requests\n"); scanf("%d",&n);
printf("Enter the Requests sequence\n");
for(i=0;i<n;i++)
```

```
scanf("%d",&RQ[i]);
printf("Enter initial head position\n");
scanf("%d",&initial);
printf("Enter total disk size\n");
scanf("%d",&size);
printf("Enter the head movement direction for high 1 and for low 0\n");
scanf("%d",&move);
for(i=0;i<n;i++)
{
for( j=0;j<n-i-1;j++)
{
if(RQ[j]>RQ[j+1])
{
int temp;
temp=RQ[j];
RQ[j]=RQ[j+1];
RQ[j+1]=temp;
}
}
}
int index;
for(i=0;i<n;i++)
{
if(initial<RQ[i])
{
index=i;

break;
}
}
// if movement is towards high value if(move==1)
{
for(i=index;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];
}
// last movement for max size
TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
 /*movement max to min disk */
 TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
 initial=0;
for( i=0;i<index;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];
}
}
// if movement is towards low value else
```

```
{
for(i=index-1;i>=0;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
// last movement for min size
TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
 /*movement min to max disk */ TotalHeadMoment=TotalHeadMoment+abs(size-
1-0);
 initial =size-1;
for(i=n-1;i>=index;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];
}
}
printf("Total head movement is %d",TotalHeadMoment); return 0;
}
```