CONTIGUOUS SEQUENTIAL

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

#define MAX 10

struct dir {
    char file_name[10];
    int file_size;
    int start_block;
}directory[MAX];

int bit_vector[MAX];
int  directory_size = 0;
int i;

void initialize() {
    for (i = 0; i < MAX; i++) {
        bit_vector[i] = 1;
    }
}
void diplay_bit_vector()
{
   for(i=0;i<10;i++)
   {
     printf("%d",bit_vector[i]);
   }
   printf("\n");
}

int check_if_free(int size) {
    int start=-1,count=0;

    for (i = 0;i<MAX;i++) {
        if (bit_vector[i] == 1) {
            if (start == -1) start = i;
            count++;
            if (count == size) return start;
        } else {
            start = -1;
            count = 0;
        }
    }
    return -1;
}

void update_bit_vector(int start, int size, int allocate) {
    int value = allocate ? 0 : 1;
    for (i = start; i < start + size; i++) {
        if (i < MAX) {
            bit_vector[i] = value;
        }
    }
}
```

```
void update_directory(const char *name, int size, int start, int add)
    {
 if (add) {
        if (directory_size < MAX) {
            strcpy(directory[directory_size].file_name, name);
            directory[directory_size].file_size = size;
            directory[directory_size].start_block = start;
            directory_size++;
        } else {
            printf("Directory is full.\n");
        }
    } else {
        int j;
        for (j = 0; j < directory_size; j++) {
            if (strcmp(directory[j].file_name, name) == 0) {
                break;
            }
        }
        if (j < directory_size) {
            for (; j < directory_size - 1; j++) {
                directory[j] = directory[j + 1];
            }
            directory_size--;
        } else {
            printf("File not found.\n");
        }
    }
}

void create_file() {
    char file_name[10];
    int file_size;

    printf("Enter file name: ");
    scanf("%s", file_name);
    printf("Enter file size: ");
    scanf("%d", &file_size);

    int start = check_if_free(file_size);
    if (start != -1) {
        update_bit_vector(start, file_size, 1); //1 to allocate
        update_directory(file_name, file_size, start, 1);
        printf("File ’%s’ create successfully.\n", file_name);
    } else {
        printf("Not enough space.\n");
    }
}

void displayDirectory() {
     printf("Bit vector: ");
     for (i = 0; i < MAX; i++) {
         printf("%d", bit_vector[i]);
     }
      printf("\n");
     printf("\nDirectory contents:\n");
     printf("File Name  Start Block  File Size\n");
     for (i = 0; i < directory_size; i++) {
         printf("%s          %d             %d\n", directory[i].file_name, directory[i].start_block,
directory[i].file_size);
     }
}

void delete() {
     char file_name[10];
     printf("Enter file name to delete: ");
     scanf("%s", file_name);

     int j;
     for (j = 0; j < directory_size; j++) {
         if (strcmp(directory[j].file_name, file_name) == 0) {
             break;
         }
     }
      if (j < directory_size) {
          update_bit_vector(directory[j].start_block, directory[j].file_size, 0);
          update_directory(file_name, 0, 0, 0);
          printf("File ’%s’ deleted successfully.\n", file_name);
     } else {
         printf("File not found.\n");
    }
}
int main() {
    initialize();

    while (1) {
        int choice;
        printf("\nMenu");
        printf("\n\1. Display bit vector\n2. Display directory\n3. create new file\n4. Delete file\n5. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: diplay_bit_vector();

                break;
            case 2:displayDirectory();

                break;
            case 3: create_file();
                break;
            case 4:delete();
  break;
            case 5:exit(0);
            default:
                printf("Invalid option.\n");
        }
    }
    return 0;
```

```
}
```

INDEXED

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

#define MAX 10
#define INDEX_BLOCK_SIZE 10


struct Directory {
    char file_name[10];
    int file_size;
    int index_block;
    int index_blocks[INDEX_BLOCK_SIZE];
} directory[MAX];

int Bit_vector[MAX], count = 0, i, j;

void initial() {
    for (i = 0; i < MAX; i++) {
        Bit_vector[i] = 1;
    }
}

void Display_bit_vector() {
    printf("\nBit Vector Status: ");
    int d;
    for (i = 0; i < MAX; i++) {
        if (Bit_vector[i] == 0) {
            // Check if this is an index block
            int is_index_block = 0;
            for (d = 0; d < count; d++) {
                if (directory[d].index_block == i) {

                    printf("{%d{", i);
                    for (j = 0; j < directory[d].file_size; j++) {
                        printf("%d", directory[d].index_blocks[j]);
                        if (j < directory[d].file_size - 1) {
                            printf(",");
                        }
                    }
                    printf("}}, ");
                    is_index_block = 1;
                    break;
                }
            }
            if (!is_index_block) {

                for (d = 0; d < count; d++) {
                    for (j = 0; j < directory[d].file_size; j++) {
                        if (directory[d].index_blocks[j] == i) {
                            printf("{%d{}}, ", directory[d].index_block);
                    }
                }
            }
        }
    } else {

        printf("1 ");
    }
  }
  printf("\n");
}

int check_if_free(int size) {
    int start = -1, count = 0;

    for (i = 0; i < MAX; i++) {
        if (Bit_vector[i] == 1) {
            if (start == -1) start = i;
            count++;
            if (count == size) return start;
        } else {
            start = -1;
            count = 0;
        }
    }
    return -1;
}

void update_bit_vector(int start, int size, int allocate) {
    int value = allocate ? 0 : 1;
    for (i = start; i < start + size; i++) {
        if (i < MAX) {
            Bit_vector[i] = value;
        }
    }
}

void update_directory(const char *name, int size, int index_block, int add) {
    if (add) {
        if (count < MAX) {
            strcpy(directory[count].file_name, name);
            directory[count].file_size = size;
            directory[count].index_block = index_block;
            count++;
        } else {
            printf("Directory is full.\n");
        }
    } else {
        int j;
        for (j = 0; j < count; j++) {
            if (strcmp(directory[j].file_name, name) == 0) {
                break;
            }
        }
        if (j < count) {
            for (; j < count - 1; j++) {
                directory[j] = directory[j + 1];
            }
            count--;
        } else {
            printf("File not found.\n");
        }
    }
}

void create_file() {
    char file_name[10];
    int file_size;

    printf("Enter file name: ");
    scanf("%s", file_name);
    printf("Enter file size: ");
    scanf("%d", &file_size);

    if (file_size <= 0 || file_size > INDEX_BLOCK_SIZE) {
        printf("Invalid file size. Must be between 1 and %d.\n", INDEX_BLOCK_SIZE);
        return;
    }

    int start = check_if_free(file_size + 1);
    if (start != -1) {

        update_bit_vector(start, file_size + 1, 1);

        int index_block = start;

        for (i = 0; i < file_size; i++) {
            directory[count].index_blocks[i] = start + i + 1;
        }

        update_directory(file_name, file_size, index_block, 1);

        printf("File '%s' created successfully.\n", file_name);
    } else {
        printf("Not enough space.\n");
    }
}

void Display_directory() {
    printf("\nDirectory:\n");
    printf("Name\tIndex Block\tLength\n");
    for (i = 0; i < count; i++) {
        printf("%s\t%d\t\t%d\t\t", directory[i].file_name, directory[i].index_block, directory[i].file_size);

        printf("\n");
    }
}

int File_is_exist_or_not(char temp[]) {
    for (i = 0; i < count; i++) {
        if (strcmp(directory[i].file_name, temp) == 0) {
            return i;
        }
    }
    return -1;
}

void Delete_File() {
    char name[10];
    printf("\nEnter name to delete the file: ");
    scanf("%s", name);
    int file_index = File_is_exist_or_not(name);

    if (file_index != -1) {
        int length = directory[file_index].file_size;
        int index_block = directory[file_index].index_block;

        Bit_vector[index_block] = 1;

        for (i = 0; i < length; i++) {
            Bit_vector[directory[file_index].index_blocks[i]] = 1;
        }

        for (i = file_index; i < count - 1; i++) {
            directory[i] = directory[i + 1];
        }
        count--;

        update_directory(name, 0, 0, 0);
        printf("File deleted successfully!\n");
    } else {
        printf("File not found!\n");
    }
}

int main() {
    int choice;

    initial();

    while (1) {
        printf("\n1. 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("\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                Display_bit_vector();
                break;
            case 2:
                create_file();
                break;
            case 3:
                Display_directory();
                break;
            case 4:
                Delete_File();
                break;
            case 5:
                exit(0);
            default:
                printf("\nInvalid Choice\n");
        }
    }
}
```

1 & 2

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

int Allocation[10][10] , Max[10][10] , Need[10][10] , Available[10] , Work[10] , Finish[10] , Request[10] ,
Safe[10];
int n , m , proc;

void accept_matrix(int A[10][10])
{
   int i, j;

   for(i = 0 ; i < n ; i++)
   {
      for(j = 0 ; j < m ; j++)
      {
         scanf("%d" , &A[i][j]);
      }
   }
}

void display()
{
  int i , j;

  printf("\nAllocation\tMax\t\tNeed\n");

  for(i = 0 ; i < n ; i++)
  {
    for(j = 0 ; j < m ; j++)
        printf("%d ", Allocation[i][j]);
    printf("\t");

    for(j = 0 ; j < m ; j++)
        printf("%d ", Max[i][j]);
    printf("\t");
    for(j = 0 ; j < m ; j++)
        printf("%d ", Need[i][j]);
    printf("\n");
  }

  printf("\nAvailable\n");
  for(j = 0 ; j < m ; j++)
     printf("%d ",Available[j]);
  printf("\t");
}


void find_need()
{
  int i , j;
  for(i = 0 ; i < n ; i++)
  {
    for(j = 0 ; j < m ; j++)
    {
       Need[i][j] = Max[i][j] - Allocation[i][j];
    }
  }
}


void accept_request()
{
   int i;
   printf("\nEnter process no. ");
   scanf("%d",&proc);
   printf("Enter Request :\n");
   for(i = 0 ; i < m ; i++)
      scanf("%d",&Request[i]);
}


int compare_need(int p)
{
   int i, j, flag = 0;
   for(j = 0 ; j < m ; j++)
   {
        if(Need[p][j] > Work[j])
        {
           flag = 1;
           break;
        }
        if(flag == 0)
            return p;
        return -1;
  }
}

void safety_alg()
{
   int over = 0 , i, j,k, l = 0 , flag , pno;
   for(i = 0 ; i < m ; i++)
   {
       Work[i] = Available[i];
   }
   for(i = 0 ; i < n ; i++)
   {
       Finish[i] = 0;
   }
   while(!over)
   {
      for(i = 0 ; i < n ; i++)
      {
         if(Finish[i] == 0)
         {
             flag = 0;
             pno = compare_need(i);
             if(pno > -1)
                break;
         }
      }
      if(i == n)
      {
         printf("System is safe");
         exit(1);
      }

      if(i < n && pno >= 0)
      {
          for(k = 0 ; k < m ; k++)
              Work[k] += Allocation[pno][k];

          Finish[pno] = 1;
          Safe[l++] = pno;

          if(l >= n)
          {
              printf("\nSafe Sequence is : ");
              for(l = 0; l < n ; l++)
                   printf("P%d\t",Safe[l]);
                   over = 1;
          }
       }
     }
}

void resource_request_alg()
{
   int i;
   for(i = 0 ; i < m ; i++)
   {
       if(Request[i] > Need[proc][i])
       {
          printf("Error : process exceeds its max demand");
          exit(1);
      }
   }

   for(i = 0 ; i < m ; i++)
   {
      if(Request[i] > Available[i])
      {
         printf("Process must wait , resources not available");
         exit(1);
      }
   }

   for(i = 0 ; i < m ; i++)
   {
      Available[i] = Available[i] - Request[i];
      Allocation[proc][i] = Allocation[proc][i] + Request[i];
      Need[proc][i] = Need[proc][i] - Request[i];
   }

   safety_alg();
}


int bankers_alg()
{
   resource_request_alg();
}


int main()
{
   int choice;

   printf("How many processes? ");
   scanf("%d",&n);

   printf("How many resources? ");
   scanf("%d",&m);

   do
   {
      printf("\nMenu\n1. Accept Allocation\n2. Accept Max\n3. Calculate Need\n4. Accept Available\n5.
Display matrices\n6. Accept request & use Bankers Algorithm\n7. Exit");
      printf("\nEnter your choice : ");
      scanf("%d",&choice);

      switch(choice)
      {
         case 1 : printf("Enter Allocation Matrix :\n");
                  accept_matrix(Allocation);
                  break;

         case 2 : printf("Enter Max Matrix :\n");
                  accept_matrix(Max);
                  break;
```

```
        case 3 : find_need();
                printf("Need is calculated which is displayed below");
                break;

        case 4 : printf("Enter Available Matrix :\n");
                int i;
                for(i = 0; i < m ; i++)
                   scanf("%d",&Available[i]);

                break;

        case 5 : printf("DATA STRUCTURES : ");
                display();
                break;

        case 6 : accept_request();
                bankers_alg();
                break;

        case 7 : printf("EXIT");
                break;

        default : printf("Invalid choice. Choose option between 1 to 6");
                 break;
      }
    }while (choice!= 7);
}
```