```
/* Write a C program to simulate the fie allocation strategies
   a. Sequential    b. Linked */

// Allocate space to files so that disk space is utilized effectively and files can be accessed
//   quickly

// Three major methods of allocating disk space: contiguous, linked and indexed

// Commonly, a system uses one method for all files within a file-system type

// List of free block of same size forming free space/hole is maintained
// And a request of size n from a list of free hole is allocated with one of method

// Sequential or Contiguous allocation - each file occupy a set of contiguous blocks on disk,
//   idea - no head movement on same sector, unless file saved across cylinders
// Disk seeks required for accessing contiguously allocated files is minimal
// Directory entry for each file indicates the address of the starting block and the length
//   of the area allocated for this file

// First fit and best fit are the most common strategies used to select a free hole from the
//   set of available holes
// All these algorithms suffer from the problem of external fragmentation - As files are
//   allocated and deleted, the free disk space is broken into little pieces
// Compacting , defragmentation

// Linked allocation solves all problems of contiguous allocation
// Each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on
//   the disk
// Directory contains a pointer to the first and last blocks of the file
// There is no external fragmentation with linked allocation
// The major problem : can be used effectively only for sequential-access files
// Some amount of block spcae will be used to save details of next block/pointer
// Collection of blocks as Clusters, but increases internal fragmentation

#include <stdio.h>
#include <string.h>
/* Sequential File Allocation : Directory entry for each file has address of the starting
    block and the length of the area allocated for this file, assume content in directory =
      file   start length
      count    0       2
      tr       14      3
      mail     19      6
      list     28      4
      f        6       2 */ // To visualize above allocation - Draw 4 * 7 grid,
// number grid entries from 0 to 31, assign file to respective blocks
struct directoryOfSequentialFileAllocation // Directory entry for each file indicates
 {
  char fileName[16]; // Name of file
  int startBlock;    // Address of the starting block
  int length;        // Length of the area allocated for this file
 };

int sNumberOfFiles = 5;
```

```
struct directoryOfSequentialFileAllocation sDirEntry[5] = { "count", 0, 2, "tr", 14, 3,
"mail", 19, 6, "list", 28, 4, "f", 6, 2 };//Read number of files and allocation details from user

/* Linked File Allocation : Directory contains a pointer to the first and last blocks of
   the file, assume content in directory =
       file    start   end
       jeep    9       25    */
struct directoryOfLinkedFileAllocation // Directory entry for each file indicates
 {
   char fileName[16]; // Name of file
   int startBlock;    // Address of the starting block
   int endBlock;      // Address of the ending block
 };

struct directoryOfLinkedFileAllocation lDirEntry[1] = { "jeep", 9, 25 };

int lNumberOfFiles = 1;

struct block  // Some amount of block spcae will be used to
 {              //    save details of next block/pointer
   int blockNumber;
   struct block *next;
 };

struct block blockEntry[32]; // Assume there are 32 blocks, structure initialization in main()

int i;
int j;
char fileName[20];

void searchInSequentialFileAllocation()
 { // Since allocation was completed, search for a file
   printf("\n Enter the file name to be searched : ");
   scanf("%s",fileName);// count, tr, mail, list, f

   for( i=0; i < sNumberOfFiles; i++ ) // For each file in directory
     if( strcmp( fileName, sDirEntry[i].fileName) == 0 ) // compare with fileName
       { // File is present, print file allocation details
         printf("\nFile name   Start block   Number of Blocks   Blocks occupied");
         printf("\n %s \t\t %d \t\t %d \t\t ", sDirEntry[i].fileName,
                                  sDirEntry[i].startBlock, sDirEntry[i].length );
         j=0;
         do
          {
           printf("%d ", sDirEntry[i].startBlock + j);
           j++;
          } while( j < sDirEntry[i].length && printf(", ") );

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

void searchInLinkedFileAllocation()
```

```
{
  int numberOfBlocks = 0;
  printf("\n Enter the file name to be searched : ");//search for a file since allocation
  scanf("%s",fileName); // was completed, search for jeep

  for( i=0; i < lNumberOfFiles; i++ ) // For each file in directory
   if( strcmp( fileName, lDirEntry[i].fileName) == 0 ) // compare with fileName
     { // File is present, print file allocation details
      printf("\nFile name   Start block   End Blocks   Blocks occupied");
      printf("\n %s \t\t %d \t\t %d \t\t", lDirEntry[i].fileName,
                              lDirEntry[i].startBlock, lDirEntry[i].endBlock );
      // Using the start location of block, continue till next block is NULL
      struct block *blockPtr =  &blockEntry[ lDirEntry[i].startBlock ];
      do
        {
         printf(" %d", blockPtr -> blockNumber ); // print block number
         blockPtr = blockPtr -> next; // get address next block
         numberOfBlocks++;
        } while ( blockPtr != NULL && printf(" -> ") ); // Or stopping condition
      // can also be blockPtr -> blockNumber != lDirEntry[i].endBlock
      printf("\n Number of blocks occupied = %d\n",numberOfBlocks); return;
     }
  return;
}

int main()
 {
  searchInSequentialFileAllocation();

// Linked Allocation : struct block blockEntry[32];
// Suppose, a file of five blocks might start at block 9 and continue at block 16
//   then block 1, then block 10, and finally block 25
// Visualize this by drawing 4*7 grid, number blocks from 0 to 31, let 9 -> 16 ...
// Each block contains a pointer to the next block, nil (the end-of-list pointer value),
//   signify an empty file, size field is 0
  blockEntry[9].blockNumber  = 9;  blockEntry[9].next = &blockEntry[16];
  blockEntry[16].blockNumber = 16; blockEntry[16].next = &blockEntry[1];
  blockEntry[1].blockNumber  = 1;  blockEntry[1].next = &blockEntry[10];
  blockEntry[10].blockNumber = 10; blockEntry[10].next = NULL;

  searchInLinkedFileAllocation();

  return(0);
}
```