```
/* 6. Write a C program to simulate the following file organization techniques:
     a) Single level directory    b) Two level directory    c) Hierarchical  */

// Single level directory - All files contained in same directory
// Easy to support and understand, large number of names requirend,
// But how to support different user?
// When represented as tree structure, tree height = 1
// Tree height = max ( number of edges encountered when traversing from root to  leaf/last
//                                 file/directory )

// Two level directory-one Master File Directory(MFD), supports separate directory for
//   each user, User has own User File Directory(UFD), MFD has User name pointing to UFD
// When represented as tree structure, tree height = 2
// Root = MFD , direct descendants / sub directories = UFD

// Hierarchical, Tree Structure - Directory / Tree structure with arbitary height
// Single and Two level directory are specific case of Hierarchical directory with
//   height 1 and 2 respectively; Users may create their own subdirectories
// One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1)
// Path to a file in a tree-strucured directory can be longer than in a two-level directory

// Single level : All files have have parent as root directory, inode 0
// sub directory can not be created, only regular files, file with same names not allowed
//                            /  Root directory , the only directory
//   _________________|_________________
// |      |  |      |    |        |          |    |      |
// cat  bo  a  test  data  mail  cont  hex  records  ; all are regular files

// Two level : Root directory with indoe 0 , can have file or sub directory, not sub directories
//                                   /  Root directory , can have sub directory
//              ______________|______________
//             |            |                 |              |
//           user1      user2         user3        user4    Directories, they can have files, but no subdir
//    ________|__        __|__        __|__       ____|_____
//    |     |     |      |        |       |       |       |       |       |       |
//   cat  bo  a  test     a  data      a    test     x    data      a  ; all are regular files

/* Hierarchical, Tree Structure - Directory / Tree , will have Root directory, no restrictions on
files or sub directories
                                                 /  Root directory
                               ______________|______________
                              |                  |                        |
                            spell              bin                 program
                     ________|__       _____|__________           _|_______
                     |          |       |       |        |        |        |            |     |      |
                   stat  mail  dist   find  count  hex  reorder   p    e    mail
                ___________|____                       ____________|      |______
               |      |          |      |                         |          |        |             |      |
          prog   copy  prt   exp                   reorder  list  find     hex  count
     ______|__    |_     |_____
     |     |       |        |        |    |
list  obj  spell  all  last  first  */

// Generic program that can maintain details of file organization of arbitary height.
//          tree height of  1 = Single level directory
//                          2 = Two level directory
//                    Arbitary = Hierarchical directory
 #include <stdio.h>
 #include <string.h>
 #define numberOfINodes 256
 // For simplicity consider collection of following data structures to simulate file organisation
 int  iNode[numberOfINodes];// Assume only 256 inodes available, inode from 0 to 255
 int  freeInode[numberOfINodes];//list informing used/unused inode numbers, 1=free, 0=used
 struct metadataOfFile
  {
   int  parent; // Parent directory inode, -1 if no parent, example root node
   int  type; // 0 for regular, 1 for directory
   int  level; // level of file, root directory level is 0
   char name[9]; // File name, maximum 8 characters
  } fileMetadata[numberOfINodes];
// iNode + metadataOfFile structure rougly implementing Inode table structure and directory
// file structure with details inode number , type of file , level the file is on and name of file
 int level;
 int displayQueue[numberOfINodes + 1]; // Queue to hold list of directories to display

 void printSpaceBasedOnLevel( int inode)
  { // Print space based on level of file , space before file name
   for(int i=0; i<fileMetadata[inode].level ; i++ )
     printf(" ");
  }

 void displayFileOrganisation( )
  { // Function to display Directory structure and files contained in directories
   printf("\n\n File organisation\n");
   int qFront=0;    int qRear=0; // Initialize queue front and rear
   int frontDirInDisplayQueue; // Variable to save inode of first directory in display queue
   int i;

   displayQueue[qRear++] = 0;   // Enqueue Root directory , inode 0 added to queue
   while( qFront < qRear  ) // queue front not equal to queue rear
    {
     frontDirInDisplayQueue = displayQueue[qFront++]; // Select from queue front
     printSpaceBasedOnLevel(frontDirInDisplayQueue);
     printf(" Dir = \"%s\", inode = %d\n",fileMetadata[frontDirInDisplayQueue].name, frontD
irInDisplayQueue);
     printSpaceBasedOnLevel(frontDirInDisplayQueue);
     printf(" Contains = ");
     for(i=0; i<numberOfINodes; i++) // print all files and folder name
      { // And also inodes for sub directory
       if( fileMetadata[i].parent == frontDirInDisplayQueue )
        { // If the parent of file is same as the directory
         if( fileMetadata[i].type == 1 ) // if this is a directory file, i.e. sub directory
          {
           printf("  subdir=%s, inode=%d;", fileMetadata[i].name, i);
           displayQueue[qRear++] = i; // Add directory in Queue
          }
         else
           printf("  %s", fileMetadata[i].name); // Print name of file
```

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

int firstFreeInode() // function returns first free/unused inode number
 {
   for( int i=0; i<numberOfINodes; i++ )
     if ( freeInode[i] == 1 )  return i;

   return -1;     // if no free/unused inode numbers, return -1
 }

// Function to update file metadata with parameters as address of new file fileMetadata
void updateFileMetadata( struct metadataOfFile *ptr, int parent,
                         int type, int level, const char *fileName )
 { // which is to be updated, parent dir inode, file type, file level and name
   ptr -> parent = parent;
   ptr -> type = type;
   ptr -> level = level;
   strcpy( ptr -> name, fileName );
 }

void createFile( )
 { // Function to makes note of new file
   char fileName[9]; // variable to read new file name
   int parentDirInode; // variable to save parent directory inode number
   printf("\n Enter Under which directory new file is to be created \n ");
   displayFileOrganisation();
   printf("\n Please enter inode number of directory where you want to create file= ");
   scanf("%d",&parentDirInode); // Lets assume valid inode of directory is entered
   printf("\n Enter file name = "); // How would you check if file name already exists
   scanf("%s", fileName);//Assume non existing name of eight characters is entered

   int free = firstFreeInode(); // Check for first free inode
   freeInode[free] = 0; // inode number used, free = 0
   // address of fileMetadata , parent dir inode, file type, parent level + 1 and name
   updateFileMetadata( &fileMetadata[free], parentDirInode, 0, fileMetadata[parentDirInode].
level+1, fileName );
   printf("\n File %s added", fileName);
 }

void createDirectory( )
 { // Function makes note of created file
   if( level == 1 )
     { // Creating new directory not allowed in Sinlge level
       printf("\n Creating sub directory not possible in Single Level");
       return;
     }

   char dirName[9]; // variable to read new directory name
   int parentDirInode; // variable to save parent directory inode number
   printf("\n Enter Under which directory the new sub directory is to be created \n ");
   displayFileOrganisation();
   printf("\n Please enter inode number of directory under which you want to create new (sub)
directory = ");
   scanf("%d",&parentDirInode);

   if( fileMetadata[parentDirInode].level == 1 && level == 2 )
     {// In two level, further sub directory not allowed in sub directory of root
      printf("\n Creating further sub directories not possible in two level");
      return;
     }

   // Assume a valid name is entered and Check if file name already exists
   printf("\n Enter directory name = ");
   scanf("%s", dirName);

   int free = firstFreeInode(); // Check freeInode , assume free inode number is available
   freeInode[free] = 0; // inode number used, free = 0
   // address of fileMetadata , parent dir inode, file type, parent level+1 and name
   updateFileMetadata( &fileMetadata[free], parentDirInode, 1, fileMetadata[parentDirInode].
level+1 , dirName );
   printf("\n Directory %s created", dirName);
  }

 int main()
  {
   for( int i=0; i<numberOfINodes; i++ )
    { // Initialize
      iNode[i] = i;
      freeInode[i] = 1;
      fileMetadata[i].parent = -1;
    }
   freeInode[0] = 0; // inode number 0 will be used for Root Directory, hence free = 0
   // address of fileMetadata , parent dir inode, file type, level and name
   updateFileMetadata( &fileMetadata[0], -1, 1, 0, "rootDir" );
   printf("\n\n Root directory \"rootDir\" , with inode = 0 created\n");

   int operation;
   printf("\n Enter\n 1 to simulate Single level directory,\n 2 ");
   printf("for Two level directory, or\n 3 for Hierarchical\n ");
   scanf("%d",&level);
   while(1)
    {
      printf("\n\n Enter\n 1 to Create file\t 2 to Create Directory\t 3 to Display\t 4 to Exit\n ");
      scanf("%d",&operation);
      switch( operation )
       {
         case 1: createFile(); break;
         case 2: createDirectory(); break;
         case 3: displayFileOrganisation(); break;
         case 4: return 0;
       }
    }
   return(0);//Can data structure design be improved, what is the run time for file add, display
  } // With above data structure : How will you implement delete file and directory
```