Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

Experiment Title: File Organization

Aim/Objective: The student should be able to understand, how the file system manages the storage and retrieval of data on a physical storage device such as a hard drive, solid-state drive, or flash drive. Also concentrates on File Structure and different file models such as Ordinary Files, Directory Files, and Special Files.

Description:

The most common form of file structure is the sequential file in this type of file, a fixed format is used for records. All records (of the system) have the same length, consisting of the same number of fixed length fields in a particular order because the length and position of each field are known, only the values of fields need to be stored, the field name and length for each field are attributes of the file structure.

Prerequisite:

- **Basic functionality of Files.**
- **Complete idea of file structure, file types, and file access mechanisms.**

Pre-Lab Task:

File Organization terms	**FUNCTIONALITY**
File structure	Defines how data is stored and accessed in a file.
Ordinary files	Store data for general use, like text or binary files.

File Organization terms	FUNCTIONALITY
Directory files	Contain information about files in the system for organization.
Special files	Represent devices or system resources, not regular data storage.
Single-Level Directory	Contains all files in one directory, no subdirectories.

In lab task

1. Write a C program to organize the file using the single level directory.

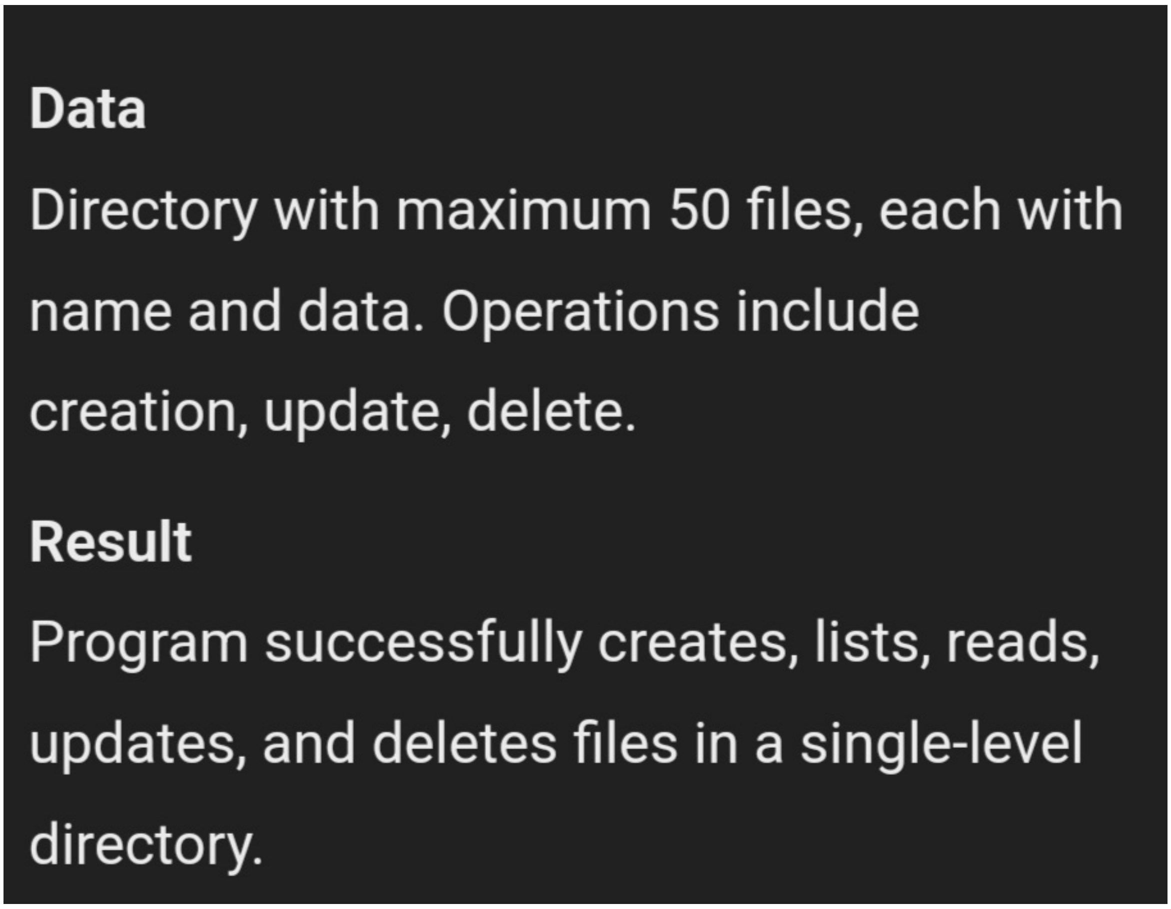
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX_FILES 50
#define MAX_NAME_LENGTH 50

struct File {
    char name[MAX_NAME_LENGTH];
    char data[MAX_NAME_LENGTH];
};

struct Directory {
    struct File files[MAX_FILES];
    int fileCount;
};

int main() {
    struct Directory directory;
    directory.fileCount = 0;
    int choice;
    do {
        printf("\nSingle-Level Directory Menu:\n");
        printf("1. Create a file\n");
        printf("2. List all files\n");
        printf("3. Read a file\n");
        printf("4. Update a file\n");
        printf("5. Delete a file\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                if (directory.fileCount < MAX_FILES) {
                    printf("Enter the name of the file: ");
                    scanf("%s", directory.files[directory.fileCount].name);
                    printf("Enter the data for the file: ");
                    scanf("%s", directory.files[directory.fileCount].data);
                    directory.fileCount++;
                    printf("File created successfully.\n");
                } else {
                    printf("Directory is full. Cannot create more files.\n");
```

```
}
        break;

      case 2:
        if (directory.fileCount == 0) {
          printf("Directory is empty.\n");
        } else {
          printf("List of files in the directory:\n");
          for (int i = 0; i < directory.fileCount; i++) {
            printf("%s\n", directory.files[i].name);
          }
        }
        break;

      case 3:
        if (directory.fileCount == 0) {
          printf("Directory is empty.\n");
        } else {
          char searchName[MAX_NAME_LENGTH];
          printf("Enter the name of the file to read: ");
          scanf("%s", searchName);
          int found = 0;
          for (int i = 0; i < directory.fileCount; i++) {
            if (strcmp(searchName, directory.files[i].name) == 0) {
              printf("Data in %s: %s\n", searchName, directory.files[i].data);
              found = 1;
              break;
            }
          }
          if (!found) {
            printf("File not found.\n");
          }
        }
        break;

      case 4:
        if (directory.fileCount == 0) {
          printf("Directory is empty.\n");
        } else {
          char updateName[MAX_NAME_LENGTH];
          printf("Enter the name of the file to update: ");
          scanf("%s", updateName);
          int found = 0;
          for (int i = 0; i < directory.fileCount; i++) {
            if (strcmp(updateName, directory.files[i].name) == 0) {
```

```
printf("Enter new data for %s: ", updateName);
                    scanf("%s", directory.files[i].data);
                    printf("File updated successfully.\n");
                    found = 1;
                    break;
                }
            }
            if (!found) {
                printf("File not found.\n");
            }
        }
        break;

    case 5:
        if (directory.fileCount == 0) {
            printf("Directory is empty.\n");
        } else {
            char deleteName[MAX_NAME_LENGTH];
            printf("Enter the name of the file to delete: ");
            scanf("%s", deleteName);
            int found = 0;
            for (int i = 0; i < directory.fileCount; i++) {
                if (strcmp(deleteName, directory.files[i].name) == 0) {
                    for (int j = i; j < directory.fileCount - 1; j++) {
                        strcpy(directory.files[j].name, directory.files[j + 1].name);
                        strcpy(directory.files[j].data, directory.files[j + 1].data);
                    }
                    directory.fileCount--;
                    printf("File %s deleted successfully.\n", deleteName);
                    found = 1;
                    break;
                }
            }
            if (!found) {
                printf("File not found.\n");
            }
        }
        break;

    case 6:
        printf("Exiting the program.\n");
        break;

    default:
```

```
printf("Invalid choice. Please enter a valid option.\n");
            break;
      }
   } while (choice != 6);
   return 0;
}
```

Data and Results



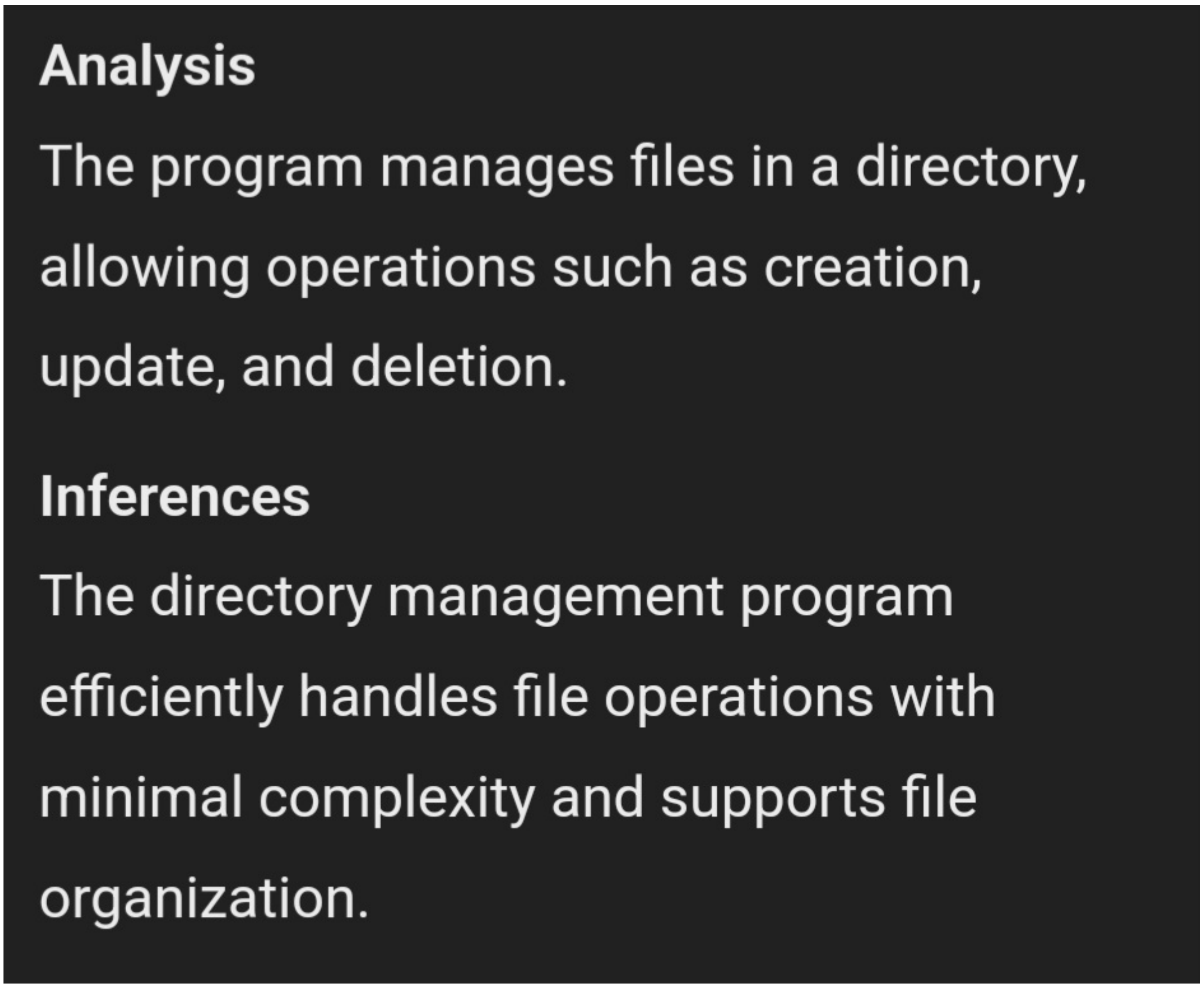
Data

Directory with maximum 50 files, each with name and data. Operations include creation, update, delete.

Result

Program successfully creates, lists, reads, updates, and deletes files in a single-level directory.

Analysis and Inferences



Analysis

The program manages files in a directory, allowing operations such as creation, update, and deletion.

Inferences

The directory management program efficiently handles file operations with minimal complexity and supports file organization.

2. Write a C program to organize the file using a level directory.

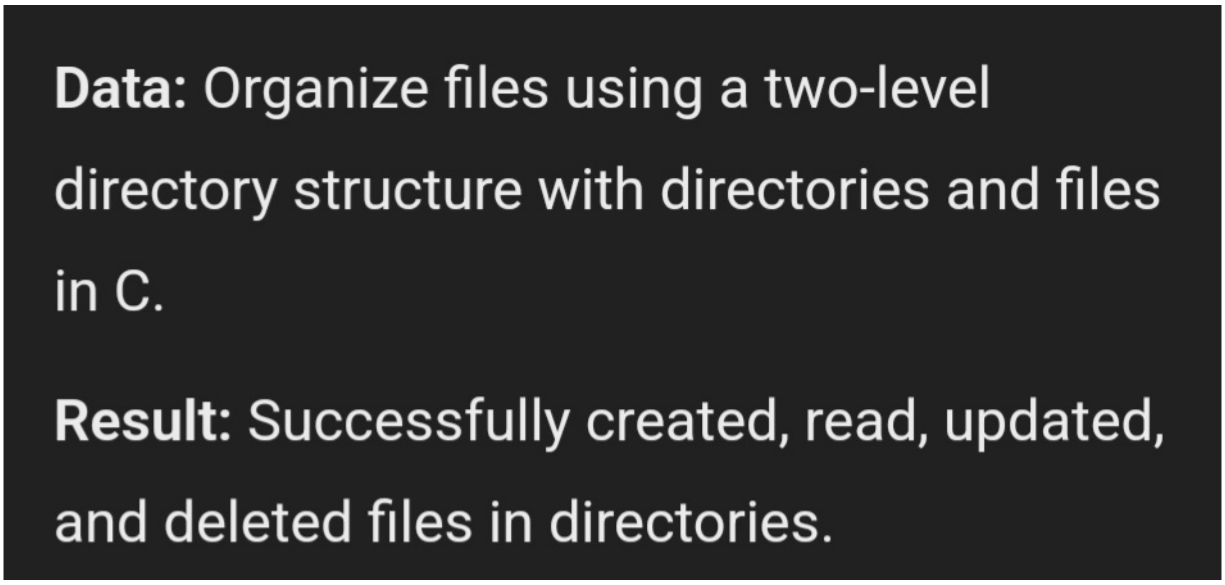
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_FILES 50
#define MAX_NAME_LENGTH 50

struct File {
    char name[MAX_NAME_LENGTH];
    char data[MAX_NAME_LENGTH];
};

struct Directory {
    char name[MAX_NAME_LENGTH];
    struct File files[MAX_FILES];
    int fileCount;
};

int main() {
    struct Directory rootDirectory;
    rootDirectory.fileCount = 0;
    strcpy(rootDirectory.name, "root");

    int choice;
    do {
        printf("\nTwo-Level Directory Menu:\n");
        printf("1. Create a directory\n");
        printf("2. Create a file\n");
        printf("3. List files in a directory\n");
        printf("4. Read a file\n");
        printf("5. Update a file\n");
        printf("6. Delete a file\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                if (rootDirectory.fileCount < MAX_FILES) {
                    char newDirectoryName[MAX_NAME_LENGTH];
                    printf("Enter the name of the new directory: ");
                    scanf("%s", newDirectoryName);
                    struct Directory newDirectory;
                    newDirectory.fileCount = 0;
                    strcpy(newDirectory.name, newDirectoryName);
```

```
 rootDirectory.files[rootDirectory.fileCount++] = newDirectory;
                printf("Directory '%s' created successfully.\n", newDirectoryName);
            } else {
                printf("Maximum directory limit reached.\n");
            }
            break;

        case 2:
            if (rootDirectory.fileCount < MAX_FILES) {
                char directoryName[MAX_NAME_LENGTH];
                printf("Enter the name of the directory to create the file in: ");
                scanf("%s", directoryName);
                int found = 0;
                for (int i = 0; i < rootDirectory.fileCount; i++) {
                    if (strcmp(directoryName, rootDirectory.files[i].name) == 0) {
                        if (rootDirectory.files[i].fileCount < MAX_FILES) {
                            char newFileName[MAX_NAME_LENGTH];
                            printf("Enter the name of the new file: ");
                            scanf("%s", newFileName);
                            struct File newFile;
                            strcpy(newFile.name, newFileName);
                            printf("Enter data for the file: ");
                            scanf("%s", newFile.data);
                            rootDirectory.files[i].files[rootDirectory.files[i].fileCount++] = newFile;
                            printf("File '%s' created successfully in directory '%s'.\n", newFileName,
directoryName);
                        } else {
                            printf("Maximum file limit reached in directory '%s'.\n", directoryName);
                        }
                        found = 1;
                        break;
                    }
                }
                if (!found) {
                    printf("Directory '%s' not found.\n", directoryName);
                }
            } else {
                printf("Maximum directory limit reached.\n");
            }
            break;

        case 3:
            printf("List of directories in the root directory:\n");
            for (int i = 0; i < rootDirectory.fileCount; i++) {
                printf("Directory: %s\n", rootDirectory.files[i].name);
```

```
  }
          break;

        case 4:
          printf("Enter the name of the directory containing the file: ");
          char searchDirectoryName[MAX_NAME_LENGTH];
          scanf("%s", searchDirectoryName);
          printf("Enter the name of the file to read: ");
          char searchFileName[MAX_NAME_LENGTH];
          scanf("%s", searchFileName);
          int found = 0;
          for (int i = 0; i < rootDirectory.fileCount; i++) {
            if (strcmp(searchDirectoryName, rootDirectory.files[i].name) == 0) {
              struct Directory directory = rootDirectory.files[i];
              for (int j = 0; j < directory.fileCount; j++) {
                if (strcmp(searchFileName, directory.files[j].name) == 0) {
                  printf("Data in %s/%s: %s\n", searchDirectoryName, searchFileName,
directory.files[j].data);
                  found = 1;
                  break;
                }
              }
            }
          }
          if (!found) {
            printf("File not found.\n");
          }
          break;

        case 5:
          printf("Enter the name of the directory containing the file to update: ");
          char updateDirectoryName[MAX_NAME_LENGTH];
          scanf("%s", updateDirectoryName);
          printf("Enter the name of the file to update: ");
          char updateFileName[MAX_NAME_LENGTH];
          scanf("%s", updateFileName);
          found = 0;
          for (int i = 0; i < rootDirectory.fileCount; i++) {
            if (strcmp(updateDirectoryName, rootDirectory.files[i].name) == 0) {
              struct Directory directory = rootDirectory.files[i];
              for (int j = 0; j < directory.fileCount; j++) {
                if (strcmp(updateFileName, directory.files[j].name) == 0) {
                  printf("Enter new data for %s/%s: ", updateDirectoryName, updateFileName);
                  scanf("%s", directory.files[j].data);
printf("File '%s/%s' updated successfully.\n", updateDirectoryName, updateFileName);
                        found = 1;
                        break;
                    }
                }
            }
        }
        if (!found) {
            printf("File not found.\n");
        }
        break;

    case 6:
        printf("Enter the name of the directory containing the file to delete: ");
        char deleteDirectoryName[MAX_NAME_LENGTH];
        scanf("%s", deleteDirectoryName);
        printf("Enter the name of the file to delete: ");
        char deleteFileName[MAX_NAME_LENGTH];
        scanf("%s", deleteFileName);
        found = 0;
        for (int i = 0; i < rootDirectory.fileCount; i++) {
            if (strcmp(deleteDirectoryName, rootDirectory.files[i].name) == 0) {
                struct Directory directory = rootDirectory.files[i];
                for (int j = 0; j < directory.fileCount; j++) {
                    if (strcmp(deleteFileName, directory.files[j].name) == 0) {
                        for (int k = j; k < directory.fileCount - 1; k++) {
                            strcpy(directory.files[k].name, directory.files[k + 1].name);
                            strcpy(directory.files[k].data, directory.files[k + 1].data);
                        }
                        directory.fileCount--;
                        printf("File '%s/%s' deleted successfully.\n", deleteDirectoryName,
deleteFileName);
                        found = 1;
                        break;
                    }
                }
            }
        }
        if (!found) {
            printf("File not found.\n");
        }
        break;

    case 7:
        printf("Exiting the program.\n");
```

```
            break;

             default:
                printf("Invalid choice. Please enter a valid option.\n");
                break;
        }
    } while (choice != 7);
    return 0;
}
```

Data and Results

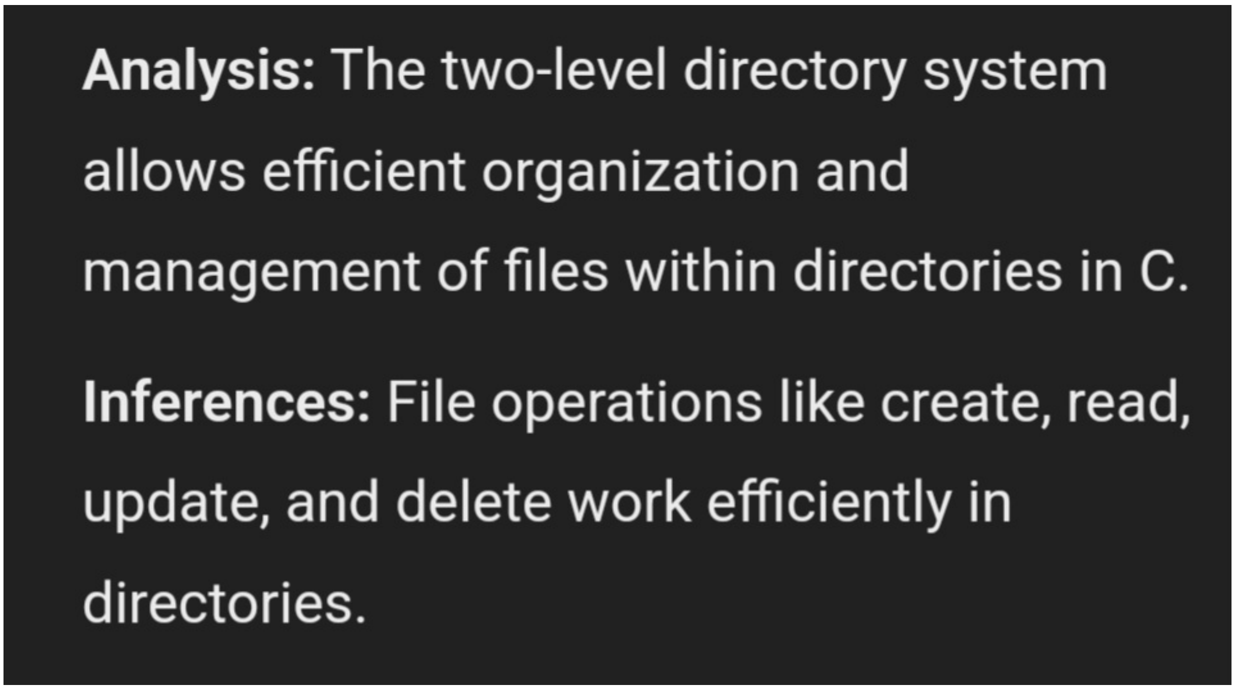


Data: Organize files using a two-level directory structure with directories and files in C.

Result: Successfully created, read, updated, and deleted files in directories.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

Analysis and Inferences:



Analysis: The two-level directory system allows efficient organization and management of files within directories in C.

Inferences: File operations like create, read, update, and delete work efficiently in directories.

Post lab task

1. Write program to Implementation of the following File Allocation
   Strategies
   a) Sequential
   b) Indexed
   c) Linked

a) Sequential

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_BLOCKS 100
int disk[MAX_BLOCKS];

int allocateSequential(int fileSize) {
    static int start = 0;
    if (start + fileSize <= MAX_BLOCKS) {
        int allocStart = start;
        start += fileSize;
        return allocStart;
    } else {
        return -1;
    }
}

int main() {
    int fileSize;
    int fileStartBlock;

    for (int i = 0; i < MAX_BLOCKS; i++) {
        disk[i] = 0;
    }

    printf("Enter file size: ");
    scanf("%d", &fileSize);
    fileStartBlock = allocateSequential(fileSize);
    if (fileStartBlock != -1) {
        printf("File allocated at blocks %d to %d.\n", fileStartBlock, fileStartBlock + fileSize - 1);
    } else {
        printf("File allocation failed. Not enough space.\n");
    }
    return 0;
}
```

b) Indexed

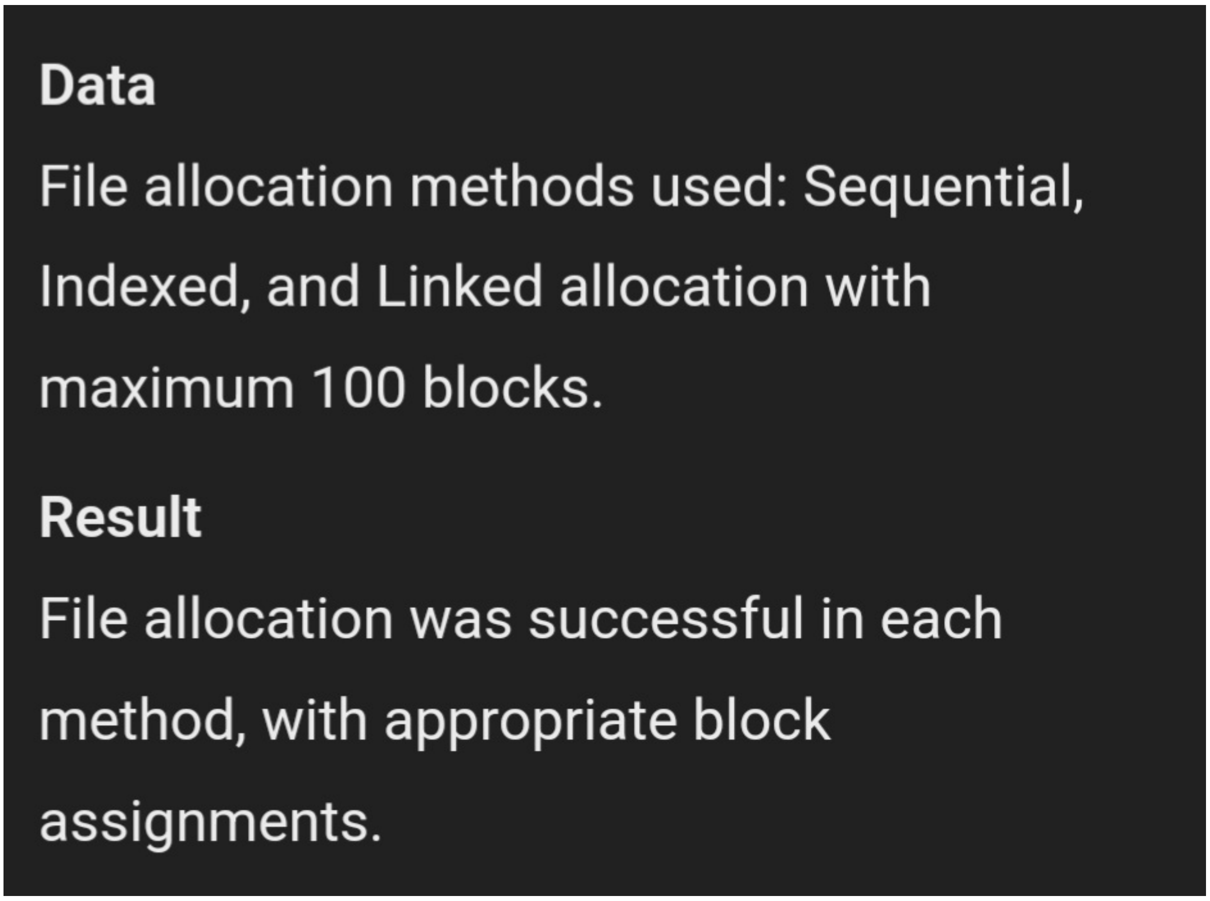
```
#include <stdio.h>
#include <stdlib.h>
#define MAX_BLOCKS 100
#define MAX_FILES 10
#define BLOCK_SIZE 4
struct IndexTable {
   int dataBlock[MAX_BLOCKS];
};
struct File {
   int indexBlock;
   int fileSize;
};
struct IndexTable indexTable;
struct File files[MAX_FILES];

int allocateIndexed(int fileSize) {
   static int nextIndexBlock = 0;
   if (nextIndexBlock < MAX_BLOCKS) {
      files[nextIndexBlock].indexBlock = nextIndexBlock;
      files[nextIndexBlock].fileSize = fileSize;
      nextIndexBlock++;
      return nextIndexBlock - 1;
   } else {
      return -1;
   }
}

int main() {
   int fileSize;
   int fileIndexBlock;

   for (int i = 0; i < MAX_BLOCKS; i++) {
      indexTable.dataBlock[i] = -1;
   }

   printf("Enter file size: ");
   scanf("%d", &fileSize);
   fileIndexBlock = allocateIndexed(fileSize);
   if (fileIndexBlock != -1) {
      printf("File allocated with index block: %d\n", fileIndexBlock);
   } else {
      printf("File allocation failed. Not enough space.\n");
```

```
    }
    return 0;
}
```

c) Linked

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_BLOCKS 100
#define BLOCK_SIZE 4
struct Block {
    int data;
    int nextBlock;
};
struct Block disk[MAX_BLOCKS];

int allocateLinked(int fileSize) {
    static int nextBlock = 0;
    int fileStartBlock = nextBlock;
    for (int i = 0; i < fileSize; i++) {
        if (nextBlock < MAX_BLOCKS) {
            disk[nextBlock].data = 0;
            disk[nextBlock].nextBlock = (nextBlock + 1) % MAX_BLOCKS;
            nextBlock++;
        } else {
            return -1;
        }
    }
    disk[fileStartBlock + fileSize - 1].nextBlock = -1;
    return fileStartBlock;
}

int main() {
    int fileSize;
    int fileStartBlock;

    printf("Enter file size: ");
    scanf("%d", &fileSize);
    fileStartBlock = allocateLinked(fileSize);
    if (fileStartBlock != -1) {
        printf("File allocated starting from block: %d\n", fileStartBlock);
    } else {
        printf("File allocation failed. Not enough space.\n");         }
    return 0;
}
```

Data and Results



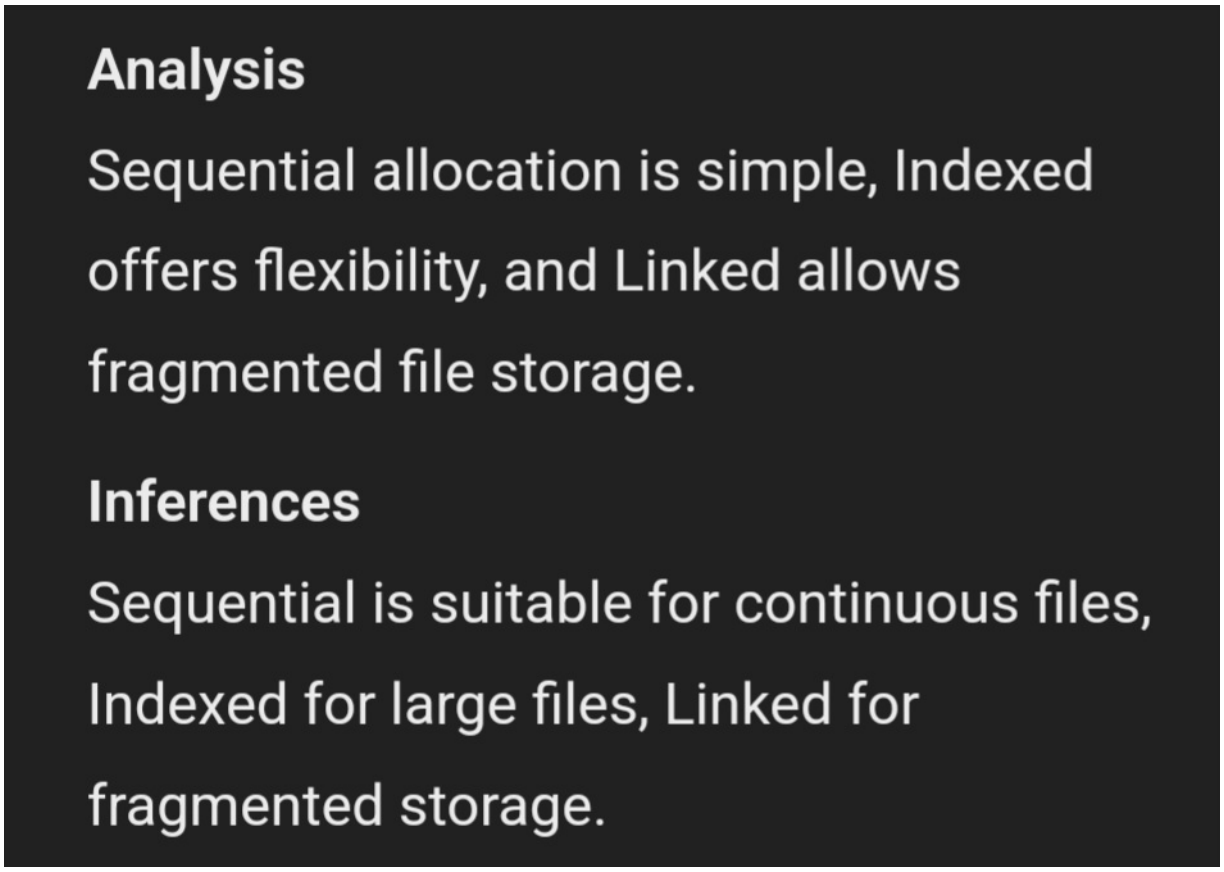
Data

File allocation methods used: Sequential, Indexed, and Linked allocation with maximum 100 blocks.

Result

File allocation was successful in each method, with appropriate block assignments.

Analysis and Inferences:



Analysis

Sequential allocation is simple, Indexed offers flexibility, and Linked allows fragmented file storage.

Inferences

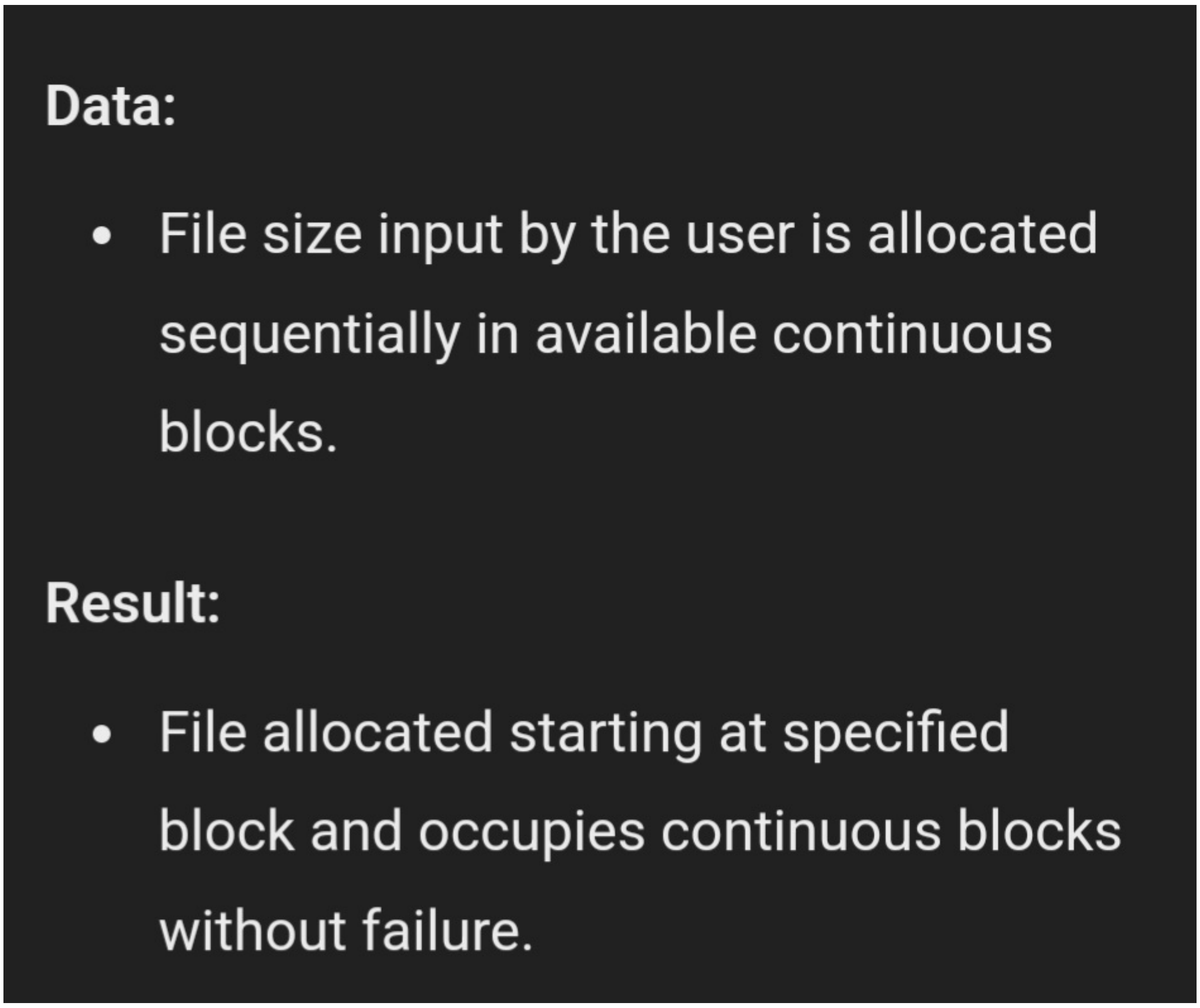
Sequential is suitable for continuous files, Indexed for large files, Linked for fragmented storage.

2. Write a C program to Implement a Continuous file allocationstrategy.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_BLOCKS 100

int disk[MAX_BLOCKS];

int allocateContinuous(int fileSize) {
    static int start = 0;

    if (start + fileSize <= MAX_BLOCKS) {
        int allocStart = start;
        start += fileSize;
        return allocStart;
    } else {
        return -1;
    }
}

int main() {
    int fileSize;
    int fileStartBlock;

    for (int i = 0; i < MAX_BLOCKS; i++) {
        disk[i] = 0;
    }

    printf("Enter file size: ");
    scanf("%d", &fileSize);

    fileStartBlock = allocateContinuous(fileSize);

    if (fileStartBlock != -1) {
        printf("File allocated starting at block %d.\n", fileStartBlock);
```

```
     printf("File occupies blocks from %d to %d.\n", fileStartBlock, fileStartBlock + fileSize - 1);
    } else {
        printf("File allocation failed. Not enough space.\n");
    }

    return 0;
}
```

Data and Results



Data:

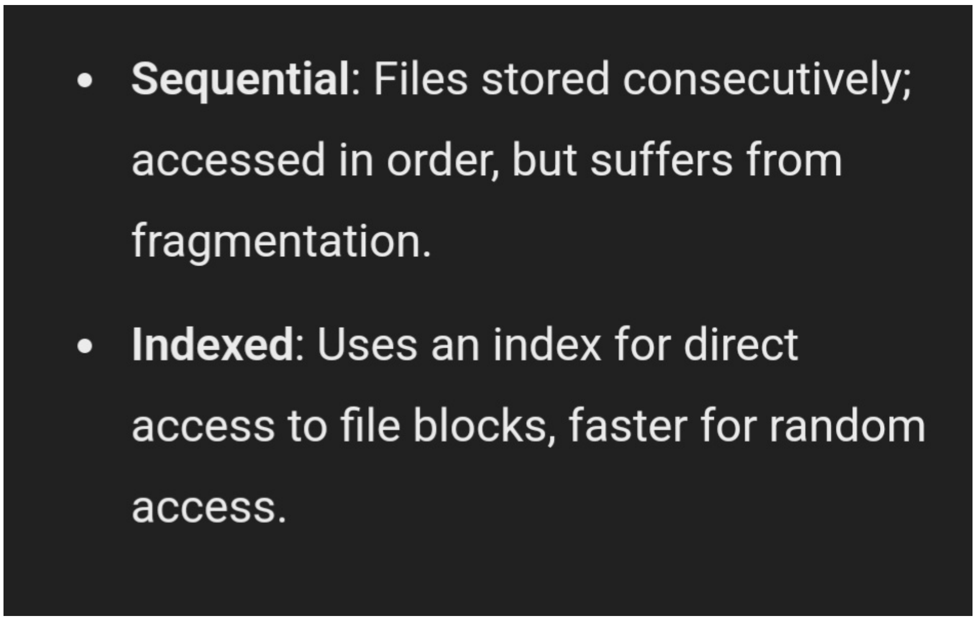
- File size input by the user is allocated sequentially in available continuous blocks.

Result:

- File allocated starting at specified block and occupies continuous blocks without failure.

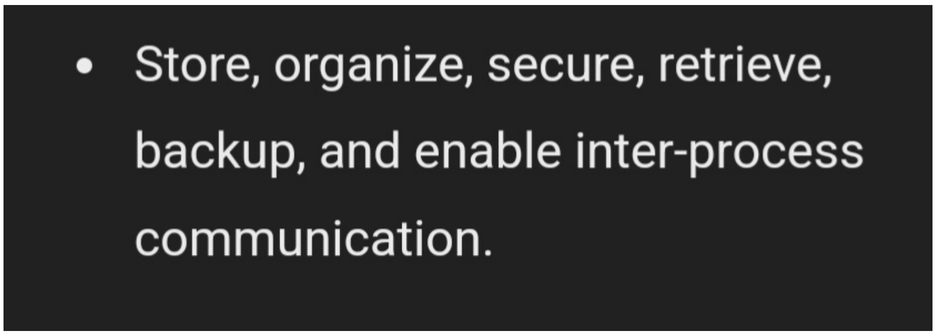
Sample VIVA-VOCE Questions (In-Lab):

6. Explain in detail about the Sequential File System and Indexed File System.



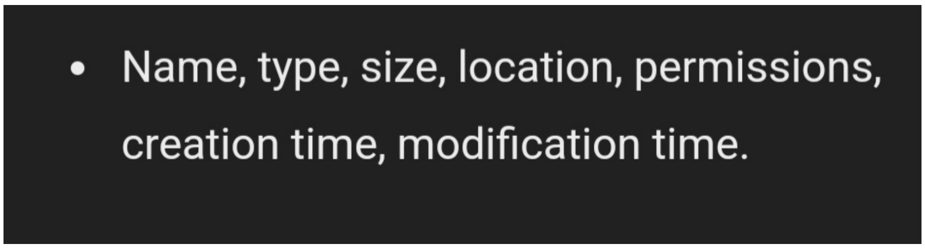
- **Sequential**: Files stored consecutively; accessed in order, but suffers from fragmentation.
- **Indexed**: Uses an index for direct access to file blocks, faster for random access.

7. Explain in detail about Functions of Files in the Operating System?



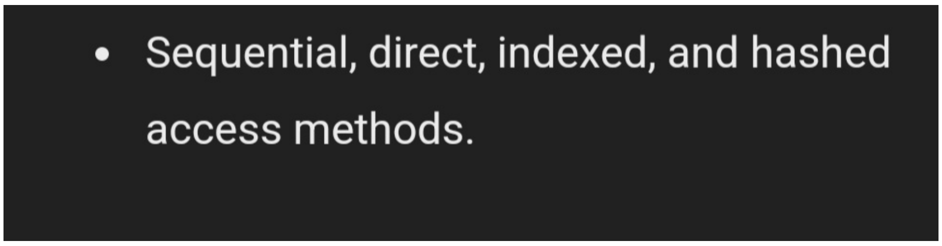
- Store, organize, secure, retrieve, backup, and enable inter-process communication.

8. What are File Attributes in OS?



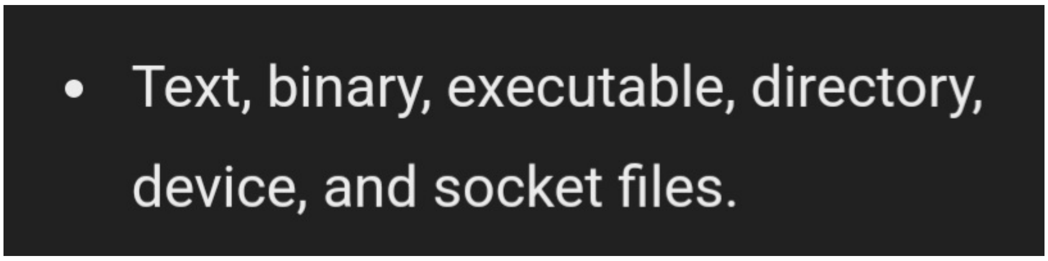
- Name, type, size, location, permissions, creation time, modification time.

9. Write File Access Mechanisms in OS.



- Sequential, direct, indexed, and hashed access methods.

10. Write down File Types in an OS.



- Text, binary, executable, directory, device, and socket files.

Evaluator Remark (if any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Note: Evaluator MUST ask Viva-voce before signing and posting marks for each experiment.