PROGRAM-13

Aim: Write a program to implement Banker's algorithm for deadlock avoidance**.**

Software Used: Windows Subsystem for Linux (ubuntu)

Theory: The Banker's Algorithm, developed by Edsger Dijkstra, is a deadlock avoidance algorithm. It checks whether granting a requested resource will leave the system in a safe state. A *safe state* means there exists at least one sequence of process execution that allows all processes to complete without leading to a deadlock.
It uses three key data structures:

- Available: Vector of available instances of each resource.
- Max: Matrix representing the maximum demand of each process.
- Allocation: Matrix representing the resources currently allocated to each process.
- Need: Matrix derived as Need[i][j] = Max[i][j] - Allocation[i][j].

A system is safe if there exists a sequence of all processes such that each process can obtain its remaining needed resources and complete its execution.

Algorithm:

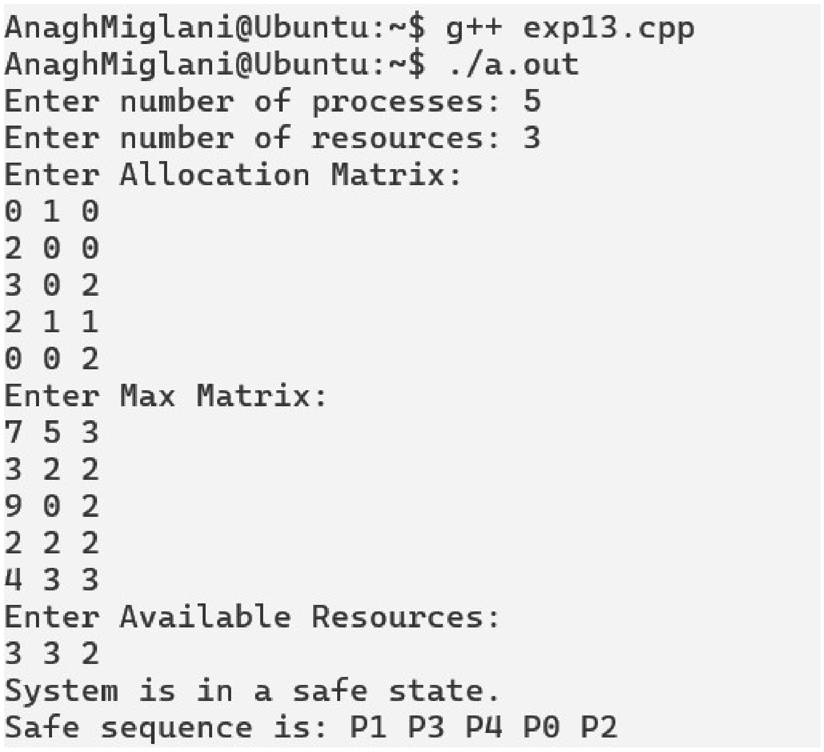
1. Input the number of processes and resources.
2. Input the matrices: Allocation, Max, and the vector Available.
3. Compute the Need matrix as Need[i][j] = Max[i][j] - Allocation[i][j].
4. Initialize Finish[i] = false for all processes.
5. Find a process P[i] such that:
   - Finish[i] == false
   - Need[i] <= Work (where Work is a copy of Available)
6. If such a process is found:
   - Allocate its resources back: Work += Allocation[i]
   - Mark Finish[i] = true
   - Add P[i] to the safe sequence
7. Repeat until all processes are finished or no such process exists.
8. If all processes can finish, the system is in a safe state; otherwise, it is unsafe.

Source Code:

```
#include  <iostream>
using namespace std;

int main() {
   int n, m;
   cout << "Enter number of processes: ";
   cin >> n;
```

```
cout << "Enter number of resources: ";
cin >> m;

int alloc[n][m], max[n][m], avail[m];
cout << "Enter Allocation Matrix:\n";
for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
        cin >> alloc[i][j];

cout << "Enter Max Matrix:\n";
for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
        cin >> max[i][j];

cout << "Enter Available Resources:\n";
for (int i = 0; i < m; i++)
    cin >> avail[i];

int need[n][m];
for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
        need[i][j] = max[i][j] - alloc[i][j];

bool finish[n] = {false};
int safeSeq[n], work[m];
for (int i = 0; i < m; i++)
    work[i] = avail[i];

int    count    =    0;
while (count < n) {
    bool found = false;
    for (int p = 0; p < n; p++) {
        if (!finish[p]) {
            int j;
            for (j = 0; j < m; j++)
                if  (need[p][j]  >  work[j])
                    break;
            if (j == m) {
                for  (int  k  =  0;  k  <  m;  k++)
                    work[k] += alloc[p][k];
                safeSeq[count++]       =       p;
                finish[p] = true;
                found = true;
                }
            }
        }
        if (!found) {
            cout << "System is not in a safe state.";
            return 0;
        }
    }

    cout << "System is in a safe state.\nSafe sequence is: ";
    for (int i = 0; i < n; i++)
        cout << "P" << safeSeq[i] << " ";
    return 0;
}
```

Output:



```
AnaghMiglani@Ubuntu:~$ g++ exp13.cpp
AnaghMiglani@Ubuntu:~$ ./a.out
Enter number of processes: 5
Enter number of resources: 3
Enter Allocation Matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter Max Matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter Available Resources:
3 3 2
System is in a safe state.
Safe sequence is: P1 P3 P4 P0 P2
```

PROGRAM-14 (a)

Aim: Write a C++ program to implement the various File Organization Techniques using Single level directory system

Software Used: Windows Subsystem for Linux (ubuntu)

Theory: The **Single-Level Directory** structure is the simplest form of file organization used in file systems. In this technique, all files are stored in a single directory. Each file must have a **unique name** since there is no concept of subdirectories or user separation. This structure is easy to implement and manage but becomes inefficient as the number of files grows because searching for files takes more time, and name conflicts are frequent.

Algorithm:

```
1. Start.
2. Create a structure File to store filename and related details.
3. Initialize an array of File to represent the directory.
4. Display a menu:
     o Create file
     o Search file
     o Delete file
     o Display files
     o Exit
5. For Create file:
     o Input filename.
     o Check if it already exists.
     o If not, add it to the directory list.
6. For Search file:
     o Input filename.
     o Traverse the list to find a match.
7. For Delete file:
     o Input filename and remove it if found.
8. For Display:
     o Print all filenames.
9. Repeat until user exits.
10. Stop.
```

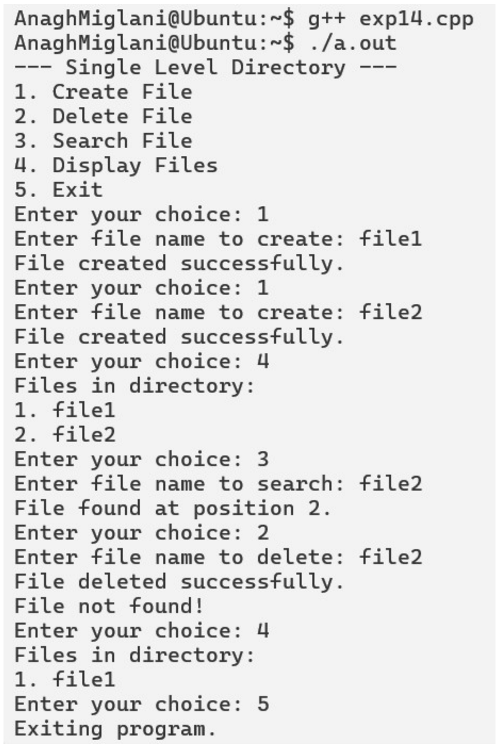
Source Code:

```
#include    <stdio.h>
#include    <string.h>
struct File {
   char name[30];
};
int main() {
```

```
struct File
directory[50]; int
count = 0, i, choice;
char filename[30];
while (1) {
  printf("\n--- Single Level Directory ---\n");
  printf("1. Create File\n");
  printf("2. Delete File\n");
  printf("3. Search File\n");
  printf("4. Display
  Files\n"); printf("5.
  Exit\n"); printf("Enter
  your choice: ");
  scanf("%d", &choice);
  switch (choice) {
    case 1:
      printf("Enter file name to create:
      "); scanf("%s", filename);
      for (i = 0; i < count; i++) {
        if (strcmp(directory[i].name, filename) == 0) {
          printf("File already exists!\n");
          break;
        }
      }
      if (i == count) {
        strcpy(directory[count].name,
        filename); count++;
        printf("File created successfully.\n");
      }
      break;
    case 2:
      printf("Enter file name to delete:
      "); scanf("%s", filename);
      for (i = 0; i < count; i++) {
        if (strcmp(directory[i].name, filename) == 0) {
          strcpy(directory[i].name, directory[count -
          1].name); count--;
          printf("File deleted
          successfully.\n"); break;
        }
      }
      if (i == count)
        printf("File not
      found!\n"); break;
    case 3:
      printf("Enter file name to search:
      "); scanf("%s", filename);
      for (i = 0; i < count; i++) {
        if (strcmp(directory[i].name, filename) ==
          0) { printf("File found at position
```

```
                    %d.\n", i + 1); break;
                }
            if (i == count)
                printf("File not found!\n");
            break;
        case 4:
            if (count == 0)
                printf("No            files            in
            directory.\n"); else {
                printf("Files                      in
                directory:\n"); for (i = 0; i
                < count; i++)
                    printf("%d. %s\n", i + 1, directory[i].name);
            }
            break;
        case 5:
            printf("Exiting  program.\n");
            return 0;
        default:
            printf("Invalid choice! Try again.\n");
        }
    }
    return 0;
}
```

Output:



```
AnaghMiglani@Ubuntu:~$ g++ exp14.cpp
AnaghMiglani@Ubuntu:~$ ./a.out
--- Single Level Directory ---
1. Create File
2. Delete File
3. Search File
4. Display Files
5. Exit
Enter your choice: 1
Enter file name to create: file1
File created successfully.
Enter your choice: 1
Enter file name to create: file2
File created successfully.
Enter your choice: 4
Files in directory:
1. file1
2. file2
Enter your choice: 3
Enter file name to search: file2
File found at position 2.
Enter your choice: 2
Enter file name to delete: file2
File deleted successfully.
File not found!
Enter your choice: 4
Files in directory:
1. file1
Enter your choice: 5
Exiting program.
```

PROGRAM-14(b)

Aim: Write a C++ program to implement the various File Organization Techniques using Two-level directory system

Software Used: Windows Subsystem for Linux (ubuntu)

Theory: The Two-Level Directory system is an improvement over the single-level directory. In this organization, each user has their own directory within a master directory. This allows different users to create files with the same name without conflict since files are organized per user.This model supports multi-user environments, where each user can manage their own files independently. Searching for a file first involves locating the user's directory and then searching within that directory.

Algorithm:

1. Start.
2. Create structures:
   - UserDirectory containing username and array of files.
   - File structure for each file.
3. Maintain an array of UserDirectory.
4. Display menu:
   - Create user directory
   - Create file under a user
   - Search file under a user
   - Delete file under a user
   - Display user directories and files
   - Exit
5. For each operation, locate the user directory first.
6. Perform file operations inside that user's file list.
7. Continue until user chooses to exit.
8. Stop.

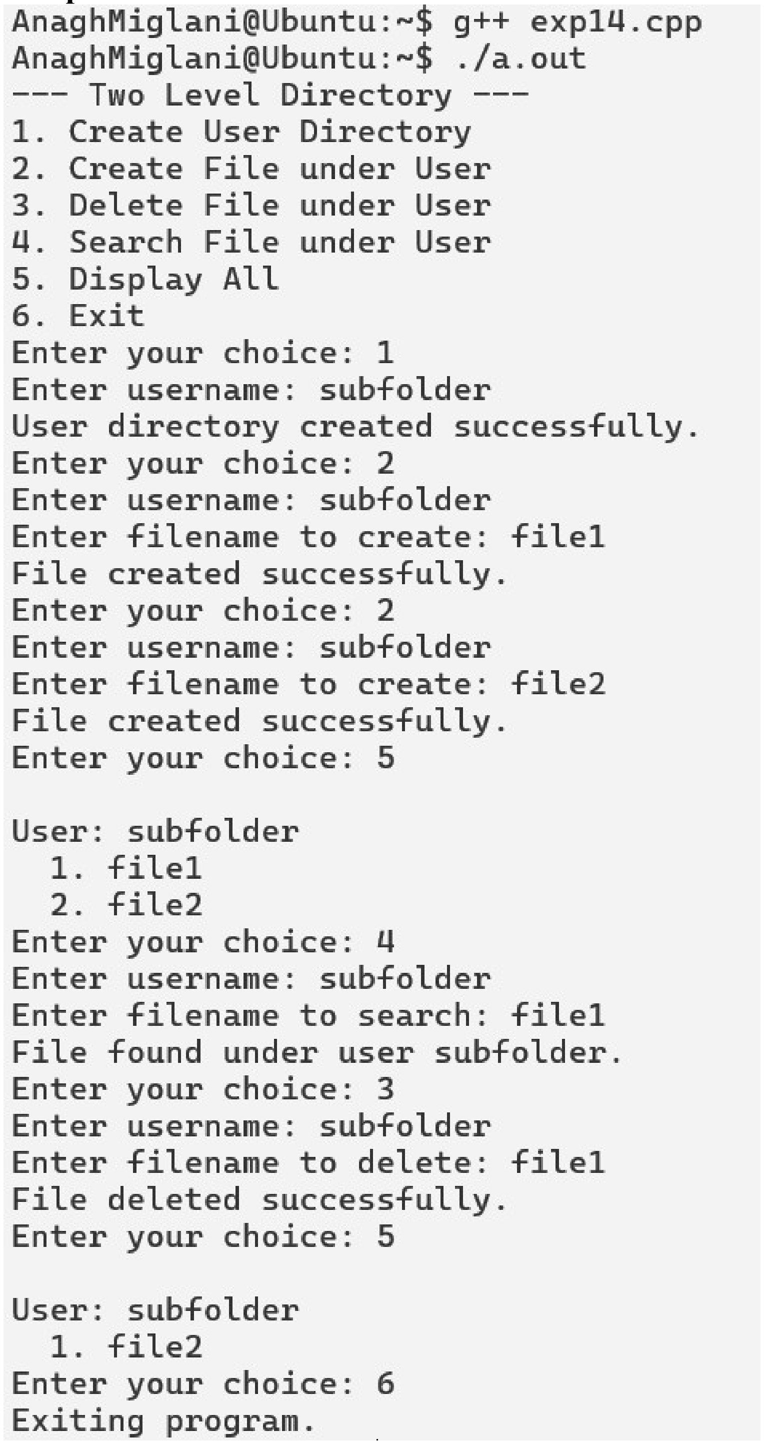
Source Code:

```
#include <stdio.h>
#include <string.h>
struct File {
  char name[30];
};
struct UserDirectory
  {               char
  username[30];
  struct          File
  files[20];        int
  fileCount;
```

```
};
int main() {
  struct UserDirectory users[10];
  int userCount = 0, i, j, choice;
  char username[30], filename[30];
  int userIndex, found;
  while (1) {
    printf("\n--- Two Level Directory ---
    \n"); printf("1. Create User
    Directory\n"); printf("2. Create File
    under User\n"); printf("3. Delete File
    under User\n"); printf("4. Search
    File under User\n"); printf("5.
    Display All\n");
    printf("6. Exit\n");
    printf("Enter your choice:
    "); scanf("%d", &choice);
    switch (choice) {
      case 1:
        printf("Enter username: ");
        scanf("%s", username);
        for (i = 0; i < userCount; i++) {
          if (strcmp(users[i].username, username) == 0)
            { printf("User already exists!\n");
            break;
          }
        }
        if (i == userCount) {
          strcpy(users[userCount].username, username);
          users[userCount].fileCount = 0;
          userCount++;
          printf("User directory created successfully.\n");
        }
        break;
      case 2:
        printf("Enter username: ");
        scanf("%s", username);
        for (i = 0; i < userCount; i++) {
          if (strcmp(users[i].username, username) == 0)
            break;
        }
        if (i == userCount) {
          printf("User not
          found!\n"); break;
        }
        userIndex = i;
        printf("Enter filename to create:
        "); scanf("%s", filename);
        for (j = 0; j < users[userIndex].fileCount; j++) {
          if (strcmp(users[userIndex].files[j].name, filename) == 0) {
            printf("File already exists!\n");
            break;
          }
        }
        if (j == users[userIndex].fileCount) {
          strcpy(users[userIndex].files[users[userIndex].fileCount].name, filename);
          users[userIndex].fileCount++;
          printf("File created successfully.\n");
        }
        break;
      case 3:
        printf("Enter username: ");
        scanf("%s", username);
        for (i = 0; i < userCount; i++) {
          if (strcmp(users[i].username, username) == 0)
            break;
        }
        if (i == userCount) {
          printf("User not
          found!\n"); break;
        }
        userIndex = i;
        printf("Enter filename to delete:
        "); scanf("%s", filename);
        found = 0;
        for (j = 0; j < users[userIndex].fileCount; j++) {
          if (strcmp(users[userIndex].files[j].name, filename) == 0) {
            strcpy(users[userIndex].files[j].name,
                users[userIndex].files[users[userIndex].fileCount -
            1].name); users[userIndex].fileCount--;
            found = 1;
            printf("File deleted
            successfully.\n"); break;
          }
        }
        if (!found)
          printf("File not found!\n");
        break;
      case 4:
        printf("Enter username: ");
        scanf("%s", username);
        printf("Enter filename to search:
        "); scanf("%s", filename);
        for (i = 0; i < userCount; i++) {
          if (strcmp(users[i].username, username) == 0)
            { found = 0;
            for (j = 0; j < users[i].fileCount; j++) {
              if (strcmp(users[i].files[j].name, filename) == 0) {
                printf("File found under user %s.\n", username);
                found = 1;
```

```
                    break;
                  }
                }
                if (!found)
                  printf("File not found under user %s.\n",
                username); break;
              }
            }
            if (i == userCount)
              printf("User not
              found!\n");
            break;
          case 5:
            if (userCount == 0)
              printf("No users
              found.\n");
            else {
              for (i = 0; i < userCount; i++) {
                printf("\nUser: %s\n",
                users[i].username); if
                (users[i].fileCount == 0)
                  printf("  No files.\n");
                else {
                  for (j = 0; j < users[i].fileCount; j++)
                    printf("  %d. %s\n", j + 1, users[i].files[j].name);
                }
              }
            }
            break;
          case 6:
            printf("Exiting program.\n");
            return 0;
          default:
            printf("Invalid choice! Try again.\n");
        }
      }
    }
```

Output:



```
AnaghMiglani@Ubuntu:~$ g++ exp14.cpp
AnaghMiglani@Ubuntu:~$ ./a.out
--- Two Level Directory ---
1. Create User Directory
2. Create File under User
3. Delete File under User
4. Search File under User
5. Display All
6. Exit
Enter your choice: 1
Enter username: subfolder
User directory created successfully.
Enter your choice: 2
Enter username: subfolder
Enter filename to create: file1
File created successfully.
Enter your choice: 2
Enter username: subfolder
Enter filename to create: file2
File created successfully.
Enter your choice: 5

User: subfolder
  1. file1
  2. file2
Enter your choice: 4
Enter username: subfolder
Enter filename to search: file1
File found under user subfolder.
Enter your choice: 3
Enter username: subfolder
Enter filename to delete: file1
File deleted successfully.
Enter your choice: 5

User: subfolder
  1. file2
Enter your choice: 6
Exiting program.
```

PROGRAM-14(C)

Aim: Write a C++ program to implement the various File Organization Techniques using Hierarchical (Tree-Structured) directory system

Software Used: Windows Subsystem for Linux (ubuntu)

Theory: The Hierarchical Directory Structure (also called a tree-structured directory) is the most advanced and commonly used organization method.In this structure, directories can contain both files and subdirectories, forming a tree-like hierarchy. The root directory acts as the topmost level, and users can navigate through paths (like root/user/docs/file.txt).This system allows efficient file organization, flexible grouping, and easy navigation between directories using operations like create, search, delete, and change directory (cd).

Algorithm:

1. Start.
2. Create a structure Directory containing:
    - Directory name
    - List of files
    - List (or pointer) to subdirectories
3. Initialize root directory.
4. Display menu:
    - Create subdirectory
    - Create file
    - Change directory (navigate into/out of subdirectories)
    - Display current directory contents
    - Exit
5. For Create subdirectory, attach a new directory node to the current directory.
6. For Change directory, traverse to the chosen subdirectory or back to parent.
7. For Create file, add file to the current directory's file list.
8. Display current directory and its contents when required.
9. Repeat until user exits.
10. Stop.

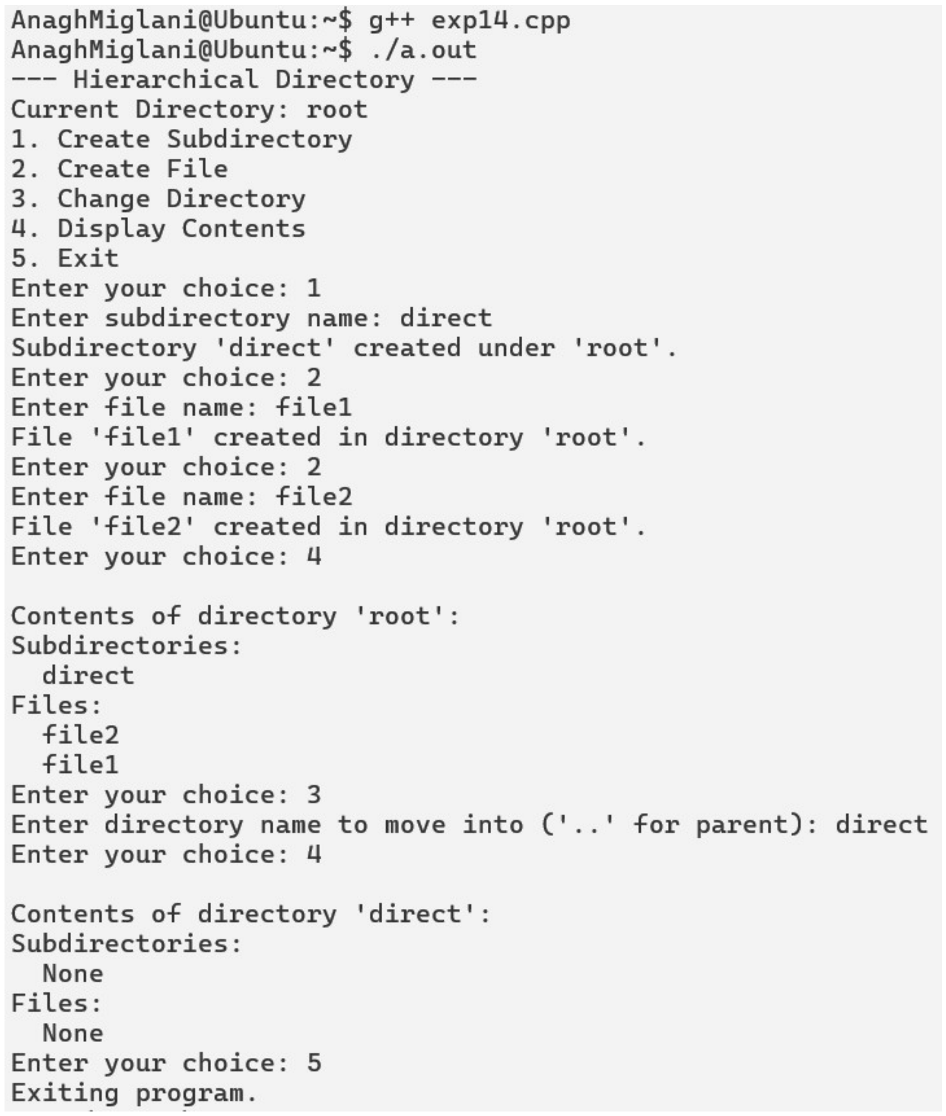
Source Code :

```
#include
<stdio.h>
#include
<stdlib.h>
#include
<string.h>   struct
File {
  char name[30];
```

```
    struct          File
    *next;
};
struct Directory {
    char name[30];
    struct Directory
    *subdirs; struct
    Directory *next; struct
    Directory *parent;
    struct File *files;
};
struct Directory *createDirectory(char name[], struct Directory *parent) {
    struct Directory *newDir = (struct Directory *)malloc(sizeof(struct
    Directory)); strcpy(newDir->name, name);
    newDir->subdirs =
    NULL; newDir->next
    = NULL; newDir-
    >parent = parent;
    newDir->files =
    NULL; return newDir;
}
void createFile(struct Directory *current, char name[]) {
    struct File *newFile = (struct File *)malloc(sizeof(struct
    File)); strcpy(newFile->name, name);
    newFile->next = current-
    >files; current->files =
    newFile;
    printf("File '%s' created in directory '%s'.\n", name, current->name);
}
void createSubdirectory(struct Directory *current, char
    name[]) { struct Directory *newDir =
    createDirectory(name, current); newDir->next = current-
    >subdirs;
    current->subdirs = newDir;
    printf("Subdirectory '%s' created under '%s'.\n", name, current->name);
}
struct Directory *changeDirectory(struct Directory *current, char
    name[]) { if (strcmp(name, "..") == 0) {
        if (current->parent != NULL)
            return current->parent;
        printf("Already at root
        directory.\n"); return current;
    }
    struct Directory *temp = current-
    >subdirs; while (temp != NULL) {
        if (strcmp(temp->name, name) == 0)
            return temp;
        temp = temp->next;
    }
    printf("Subdirectory '%s' not found.\n", name);
```

```
  return current;
}
void displayDirectory(struct Directory *current) {
  struct File *f;
  struct Directory *d;
  printf("\nContents of directory '%s':\n", current->name);
  printf("Subdirectories:\n");
  if (current->subdirs == NULL)
    printf("  None\n");
  else {
    d = current-
    >subdirs; while (d
    != NULL) {
      printf(" %s\n", d-
      >name); d = d->next;
    }
  }
  printf("Files:\n");
  if (current->files ==
    NULL) printf("
    None\n");
  else {
    f = current-
    >files; while (f
    != NULL) {
      printf(" %s\n", f-
      >name); f = f->next;
    }
  }
}
int main() {
  struct Directory *root = createDirectory("root",
  NULL); struct Directory *current = root;
  char
  name[30]; int
  choice; while
  (1) {
    printf("\n--- Hierarchical Directory ---\n");
    printf("Current Directory: %s\n", current-
    >name); printf("1. Create Subdirectory\n");
    printf("2. Create File\n");
    printf("3. Change
    Directory\n"); printf("4.
    Display Contents\n");
    printf("5. Exit\n");
    printf("Enter your choice:
    "); scanf("%d", &choice);
    switch (choice) {
      case 1:
        printf("Enter subdirectory name:
        "); scanf("%s", name);
        createSubdirectory(current,
        name); break;

      case 2:
        printf("Enter file name: ");
        scanf("%s", name);
        createFile(current,
        name); break;
      case 3:
        printf("Enter directory name to move into ('..' for parent): ");
        scanf("%s", name);
        current = changeDirectory(current,
        name); break;
      case 4:
        displayDirectory(current);
        break;
      case 5:
        printf("Exiting
        program.\n"); exit(0);
      default:
        printf("Invalid choice! Try again.\n");
    }
  }
  return 0;
}
```

Output:



```
AnaghMiglani@Ubuntu:~$ g++ exp14.cpp
AnaghMiglani@Ubuntu:~$ ./a.out
--- Hierarchical Directory ---
Current Directory: root
1. Create Subdirectory
2. Create File
3. Change Directory
4. Display Contents
5. Exit
Enter your choice: 1
Enter subdirectory name: direct
Subdirectory 'direct' created under 'root'.
Enter your choice: 2
Enter file name: file1
File 'file1' created in directory 'root'.
Enter your choice: 2
Enter file name: file2
File 'file2' created in directory 'root'.
Enter your choice: 4

Contents of directory 'root':
Subdirectories:
  direct
Files:
  file2
  file1
Enter your choice: 3
Enter directory name to move into ('..' for parent): direct
Enter your choice: 4

Contents of directory 'direct':
Subdirectories:
  None
Files:
  None
Enter your choice: 5
Exiting program.
```