FINAL REPORT

SOFTWARE DEVELOPMENT – I

Overview:

The provided code implements a basic **console-based file management system** in C++, allowing users to create and manage users, directories, and files. It demonstrates object-oriented programming principles and efficient resource management through custom copy and move semantics.

Features of Project:

- Create User: Allows users to be created to the system.
- Create Directory: Enables the creation of directories within the system.
- Create File: Allows users to create files and associate them with a specific directory and owner.
- Move File: Facilitates the movement of files between directories within the system.
- List All Files: Displays a list of all files in the system, along with their owners and directories.
- Delete File: Allows users to delete files from the system.
- Quit: Gives an option to exit the file management system.

Applications:

- **C++**
- **Object-Oriented Programming (OOP) Principles**
- **Dynamic Function**

Implementations:

- **User Class:** Encapsulates a user with a name
- **File Class:** Represents a file with attributes
- **Directory Class:** Represents a directory that can contain multiple files
- **SimpleFileManager Class**: It orchestrates the creation of project and management of users, directories and files.

Achievements:

The project has successfully implemented a basic file management system in C++ as creating users, managing directories, handling file operations (creation, moving, and deletion), and displaying file details. The project effectively utilizes object-oriented programming concepts, on a high proficiency in using classes, objects, and various methods such as constructors (both copy and move), destructors, and operator overloads to manage resources efficiently.

Conclusion:

In conclusion, the development of basic file management system project provided valuable insights into C++ programming and OOP principles. Despite challenges, the project achieved its objectives and laid a foundation for future enhancements and improvements.

Final Code & Output:

```
#include <bits/stdc++.h>
using namespace std;

class User
{
public:
   User(string name) : name(name) {}

   // Copy constructor
   User(const User &other) : name(other.name) {}

   // Copy assignment operator
   User &operator=(const User &other)
   {
      if (this != &other)
      {
         name = other.name;
      }
      return *this;
   }

   // Move constructor
   User(User &&other) noexcept : name(std::move(other.name)) {}

   // Move assignment operator
   User &operator=(User &&other) noexcept
   {
      if (this != &other)
      {
         name = std::move(other.name);
      }
      return *this;
   }

   string getName() const
   {
      return name;
   }

private:
   string name;
};

class File
```

```
{
public:
    File(string name, string content, const User &owner, const string &directory) : name(name), content(content), owner(owner), directory(directory) {}

    // Copy constructor
    File(const File &other)
        : name(other.name), content(other.content), owner(other.owner), directory(other.directory) {}

    // Move constructor
    File(File &&other) noexcept
        : name(std::move(other.name)), content(std::move(other.content)), owner(other.owner), directory(std::move(other.directory)) {}

    // Move assignment operator
    File &operator=(File &&other) noexcept
    {
        if (this != &other)
        {
            name = std::move(other.name);
            content = std::move(other.content);
            // owner = other.owner;
            directory = std::move(other.directory);
        }
        return *this;
    }

    string getName() const
    {
        return name;
    }

    string getContent() const
    {
        return content;
    }

    const User &getOwner() const
    {
        return owner;
    }

    string getDirectory() const
    {
        return directory;
    }
```

```
    void updateDirectory(const string &newDirectory)
    {
        directory = newDirectory;
    }

private:
    string name;
    string content;
    const User &owner;
    string directory;
};

class Directory
{
public:
    Directory(string name) : name(name) {}

    void moveFile(const string &fileName, Directory &destination)
    {
        auto it = find_if(files.begin(), files.end(),
                          [fileName](const File &file)
                          { return file.getName() == fileName; });

        if (it != files.end())
        {
            destination.addFile(*it);
            it->updateDirectory(destination.getName()); // Update the directory information in the moved file
            files.erase(it);
            cout << "File " << fileName << " moved from " << name << " directory to " <<
destination.getName() << " directory." << endl;
        }
        else
        {
            cout << "File " << fileName << " not found in " << name << " directory." << endl;
        }
    }

    void addFile(const File &file)
    {
        files.push_back(file);
    }

    string getName() const
    {
        return name;
    }
```

```
    const vector<File> &getFiles() const
    {
        return files;
    }

private:
    string name;
    vector<File> files;
};

class SimpleFileManager
{
public:
    vector<User> users;
    vector<Directory> directories;
    vector<File> files;

    void createUser()
    {
        int numUsers;
        while (true)
        {
            cout << "Number of Users: ";
            if (cin >> numUsers && numUsers > 0)
            {
                break;
            }
            else
            {
                cout << "Invalid input. Please enter a positive integer." << endl;
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
            }
        }

        for (int i = 0; i < numUsers; ++i)
        {
            string userName;
            cout << "Enter user name: ";
            cin >> userName;
            users.push_back(User(userName));
            cout << "User " << userName << " created." << endl;
        }
    }

    void createDirectory()
    {
```

```
    int numDirs;
    while (true)
    {
        cout << "Number of Directories: ";
        if (cin >> numDirs && numDirs > 0)
        {
            break;
        }
        else
        {
            cout << "Invalid input. Please enter a positive integer." << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
    }

    for (int i = 0; i < numDirs; ++i)
    {
        string dirName;
        cout << "Enter directory name: ";
        cin >> dirName;
        directories.push_back(Directory(dirName));
        cout << "Directory " << dirName << " created." << endl;
    }
}

void createFile()
{
    if (directories.empty())
    {
        cout << "No directories created. Please create a directory first." << endl;
        return;
    }

    cout << "Current Owners: ";
    for (const auto &user : users)
    {
        cout << user.getName() << " ";
    }
    cout << endl;

    cout << "Current Directories: ";
    for (const auto &dir : directories)
    {
        cout << dir.getName() << " ";
    }
    cout << endl;
```

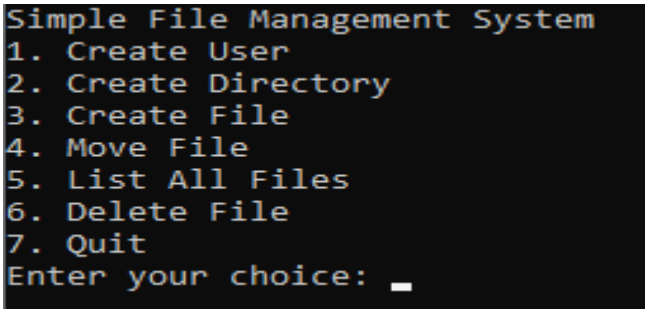
```
        string fileName, content, ownerName, dirName;

        cout << "Enter name of file: ";
        cin >> fileName;

        cout << "Enter details for the file: ";
        cin.ignore(); // Ignore newline character left in the buffer
        getline(cin, content);

        cout << "Enter owner's name: ";
        cin >> ownerName;

        auto userIt = find_if(users.begin(), users.end(),
                              [ownerName](const User &user)
                              { return user.getName() == ownerName; });

        if (userIt == users.end())
        {
            cout << "User " << ownerName << " not found. Cannot create file." << endl;
            return;
        }

        User &owner = *userIt;

        cout << "Enter directory name to create the file under: ";
        cin >> dirName;

        auto dirIt = find_if(directories.begin(), directories.end(),
                             [dirName](const Directory &dir)
                             { return dir.getName() == dirName; });

        if (dirIt != directories.end())
        {
            Directory &dir = *dirIt;
            files.push_back(File(fileName, content, owner, dirName));
            dir.addFile(File(fileName, content, owner, dirName));
            cout << "File " << fileName << " created by " << ownerName << " in directory " << dirName <<
"." << endl;
        }
        else
        {
            cout << "Directory " << dirName << " not found. Cannot create file." << endl;
        }
    }

    void moveFile()
```

```
{
    cout << "Current Files: ";
    for (const auto &file : files)
    {
        cout << file.getName() << " ";
    }
    cout << endl;

    cout << "Current Directories: ";
    for (const auto &dir : directories)
    {
        cout << dir.getName() << " ";
    }
    cout << endl;

    string fileName, sourceDirName, destDirName;
    cout << "Enter name of file to move: ";
    cin >> fileName;

    cout << "Enter source directory: ";
    cin >> sourceDirName;

    cout << "Enter destination directory: ";
    cin >> destDirName;

    auto sourceDirIt = find_if(directories.begin(), directories.end(),
                               [sourceDirName](const Directory &dir)
                               { return dir.getName() == sourceDirName; });

    auto destDirIt = find_if(directories.begin(), directories.end(),
                             [destDirName](const Directory &dir)
                             { return dir.getName() == destDirName; });

    if (sourceDirIt != directories.end() && destDirIt != directories.end())
    {
        Directory &sourceDir = *sourceDirIt;
        Directory &destDir = *destDirIt;
        sourceDir.moveFile(fileName, destDir);
    }
    else
    {
        cout << "Source or destination directory not found." << endl;
    }
}

void deleteFile()
{
```

```
        cout << "Current Files: ";
        for (const auto &file : files)
        {
            cout << file.getName() << " ";
        }
        cout << endl;

        string fileName;
        cout << "Enter name of file to delete: ";
        cin >> fileName;

        auto fileIt = find_if(files.begin(), files.end(),
                              [fileName](const File &file)
                              { return file.getName() == fileName; });

        if (fileIt != files.end())
        {
            files.erase(fileIt);
            cout << "File " << fileName << " deleted." << endl;
        }
        else
        {
            cout << "File " << fileName << " not found." << endl;
        }
    }

    void listAllFiles() const
    {
        cout << "All files in the system:" << endl;
        for (const auto &file : files)
        {
            auto dirIt = find_if(directories.begin(), directories.end(),
                                 [dirName = file.getDirectory()](const Directory &dir)
                                 { return dir.getName() == dirName; });

            if (dirIt != directories.end())
            {
                const Directory &dir = *dirIt;
                cout << "- " << file.getName() << " (Owner: " << file.getOwner().getName() << ", Directory: "
<< file.getDirectory() << ")" << endl;
                cout << "  Content: " << file.getContent() << endl;
            }
            else
            {
                cout << "File " << file.getName() << " has an invalid directory." << endl;
            }
        }
```

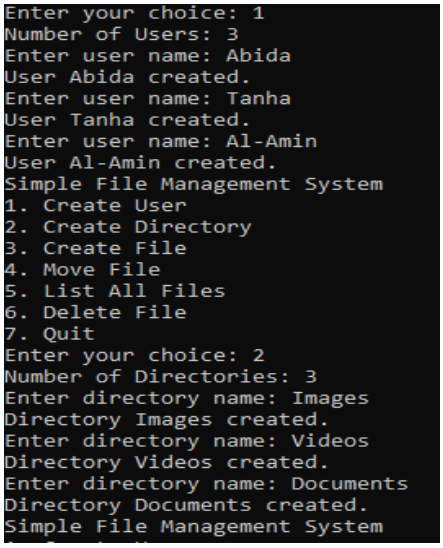
```
    }
};

int main()
{
    SimpleFileManager fileManager;

    while (true)
    {
        cout << "Simple File Management System" << endl;
        cout << "1. Create User" << endl;
        cout << "2. Create Directory" << endl;
        cout << "3. Create File" << endl;
        cout << "4. Move File" << endl;
        cout << "5. List All Files" << endl;
        cout << "6. Delete File" << endl;
        cout << "7. Quit" << endl;

        int choice;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            fileManager.createUser();
            break;
        case 2:
            fileManager.createDirectory();
            break;
        case 3:
            fileManager.createFile();
            break;
        case 4:
            fileManager.moveFile();
            break;
        case 5:
            fileManager.listAllFiles();
            break;
        case 6:
            fileManager.deleteFile();
            break;
        case 7:
            cout << "Quitting the program." << endl;
            return 0;
        default:
            cout << "Invalid choice. Please enter a valid option." << endl;
```

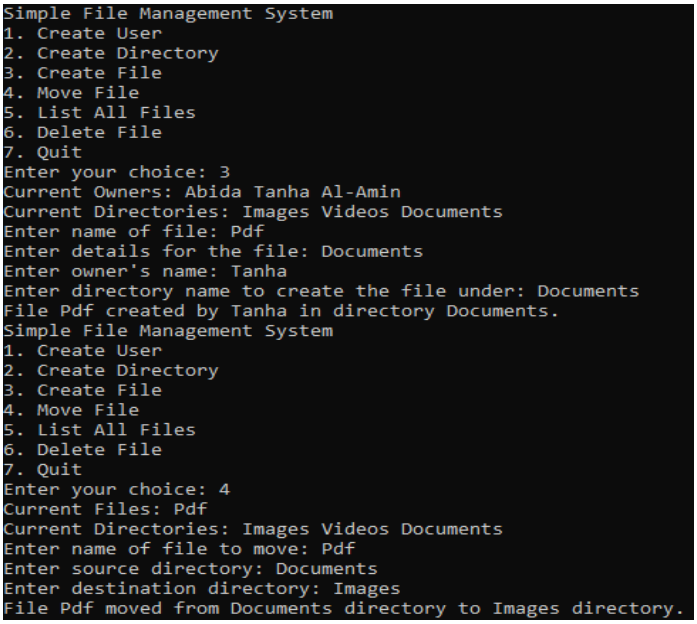
```
    }
  }

  return 0;
}
```



User gets final interface of the code and begins the procedure. This is the initial process where user select the list and choose number of users according to his wants.

Outputs:





Here is the process how user choose how many user should be created and enter the number of directories as much as user wants. By following the upcoming steps, user creates multiple files and choose their directories according the type of data it stores. User gradually finalize his work and can list all the files according to his users and directories to check all are ok or not. If everything is ok, user can quit the process. This is how the project actually works. Thank you.