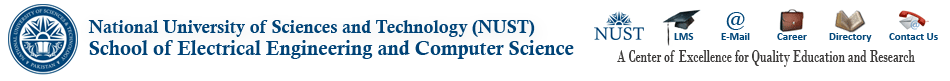
****

## 

## **Department of Computing**

## **CS 330: Operating Systems**

## **BSCS: 10ABC**

## **Assignment: File Management**

### 

## **Instructor: Dr. Farzana Jabeen (AB) & Dr. Sana Qadir (C)**

1. **System design and Implementation details**
   * **File and directory implementation**

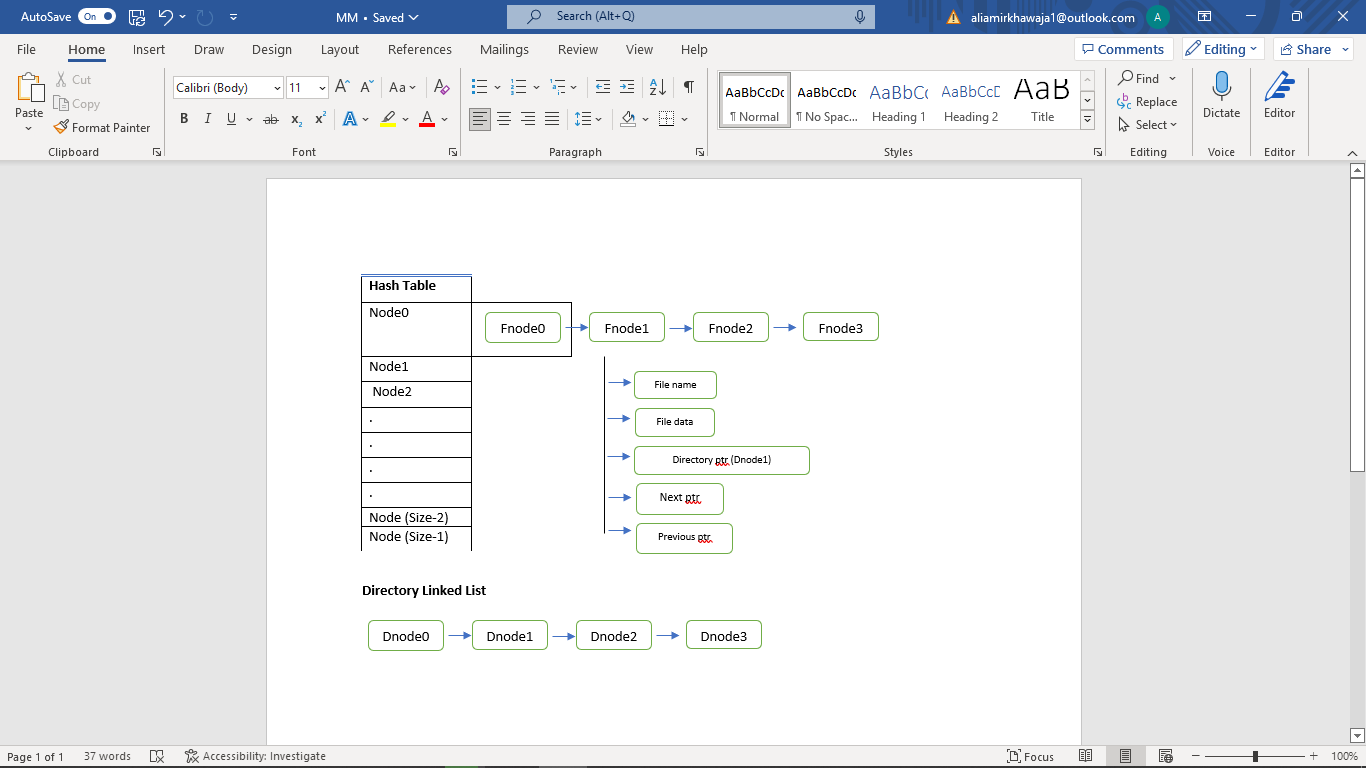
The file management system consists of allocation for the directories and files. For this assignment, a linked list of nodes is maintained, and a hash table is created for maintaining the files. The hash table of nodes is designed to store the contents of the files. Each index can have a chained list attached to avoid collisions.

Each node has five components. The node has two string variables and three-pointers. The string variables contain a file name and file data. And, the pointers include the file directory address from the directory list, the next pointer pointing to the next node, and the previous pointer pointing to the previous node (only used in the deletion operation).

* + **Memory allocation**

For this system, the size of the number memory locations available for the storage of files is 1024 (scalable). Each file in the table has a variable size. Each directory/folder will contain one or multiple files, so the size allocated for each file or directory in memory will be variable.

1. **Memory map**



1. **Data structure**

We use Hash tables, lists, and arrays. In a hash table, data is stored in an array format, where each data value has its unique index value. Access to data becomes very fast if we know the index of the desired data. Hashtables have O (1) times the complexity giving them an edge over other data structures.

The list is the basic structure that is the most used. It has the best-case time complexity of O(n). Lists are used to store files that are hashed to the same location.

1. **System functions**

The system can open a file, edit it, and make changes to it. We can create a directory, and make changes to it. We can change the location of a file without changing its physical location.

* Create A File.
* Delete A File.
* Move the File To Directory.
* Open A File(For Read or Write).
* Move Contents Within A File.
* Create Directory
* Truncate a File.
* Write Memory Map To File.

1. **System requirements**

The system allows opening a file. It allows the storage of content in the file. We can make changes to the File. We can add additional data in the file, and append data. we can also free up the node to truncate files. Also, can create folders and place files in directories.

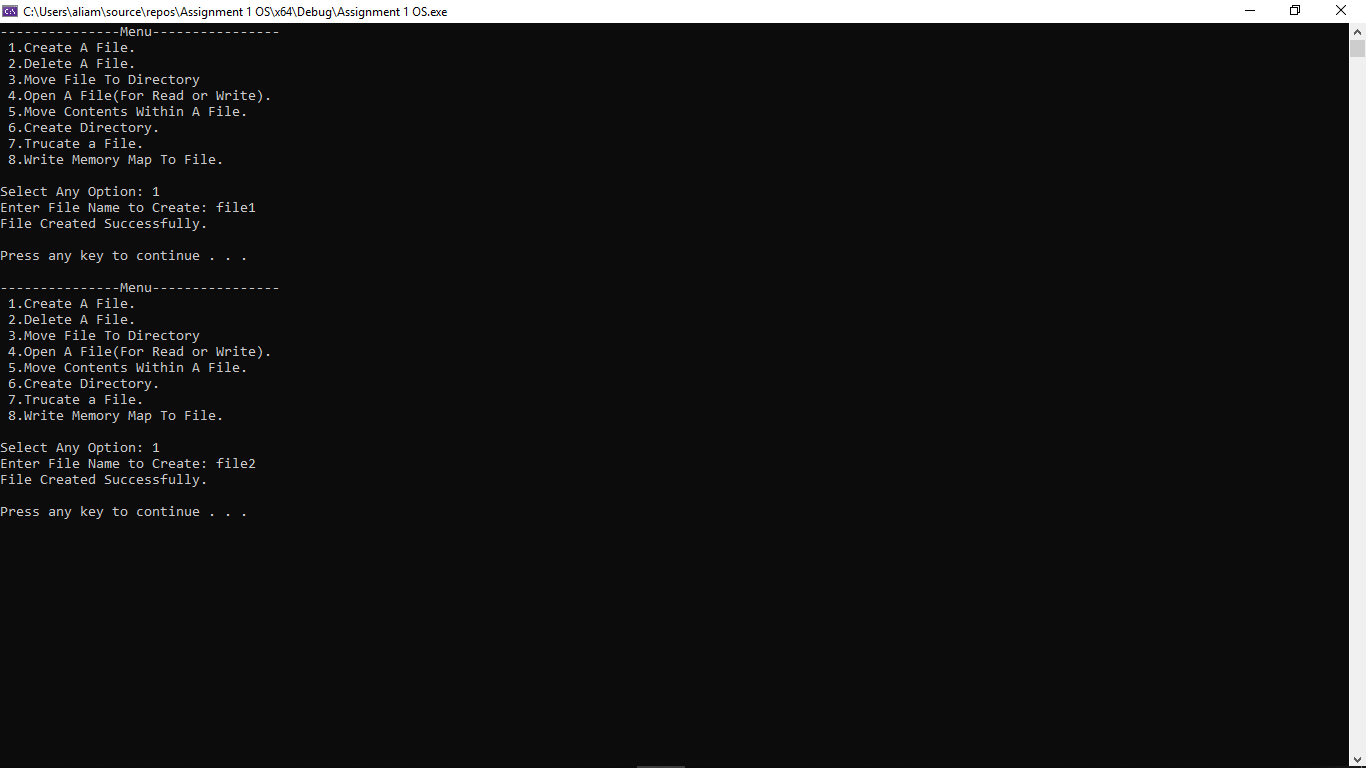
1. **Limitations**

The limitation of the system is that the file name created, whose ASCII value maybe by chance map to the same hash table index over and over. This would create a collision each time and would result in a long chain list with other indexes of the table relatively free.

The file system has a size limited to 1024 indexes. Therefore, 1024 unique indexes are available to store data efficiently, to a complexity of O(1), other files will be stored in the linked list with the complexity of O(n) after all indexes are full.

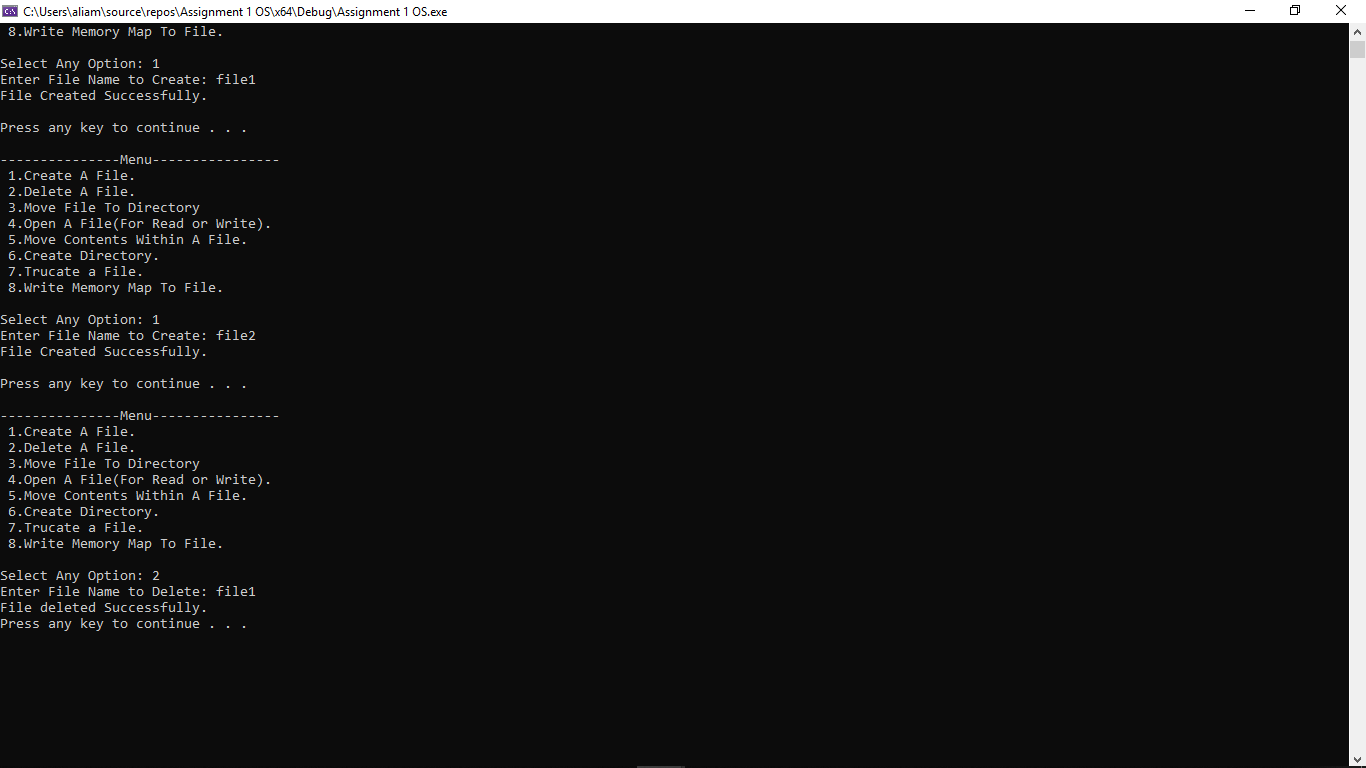
1. **Outputs (screenshots)**

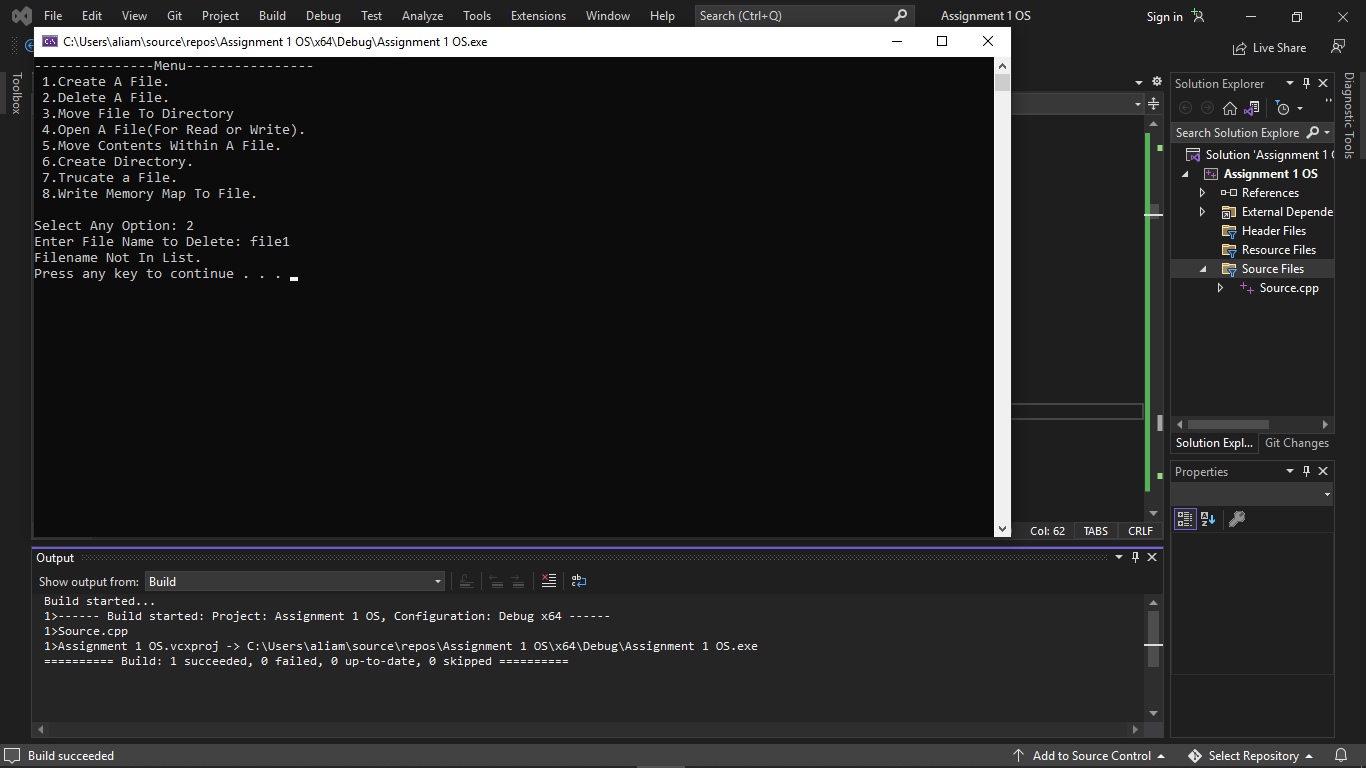
* Create A File.



Two files were created named **file1** and **file2**.

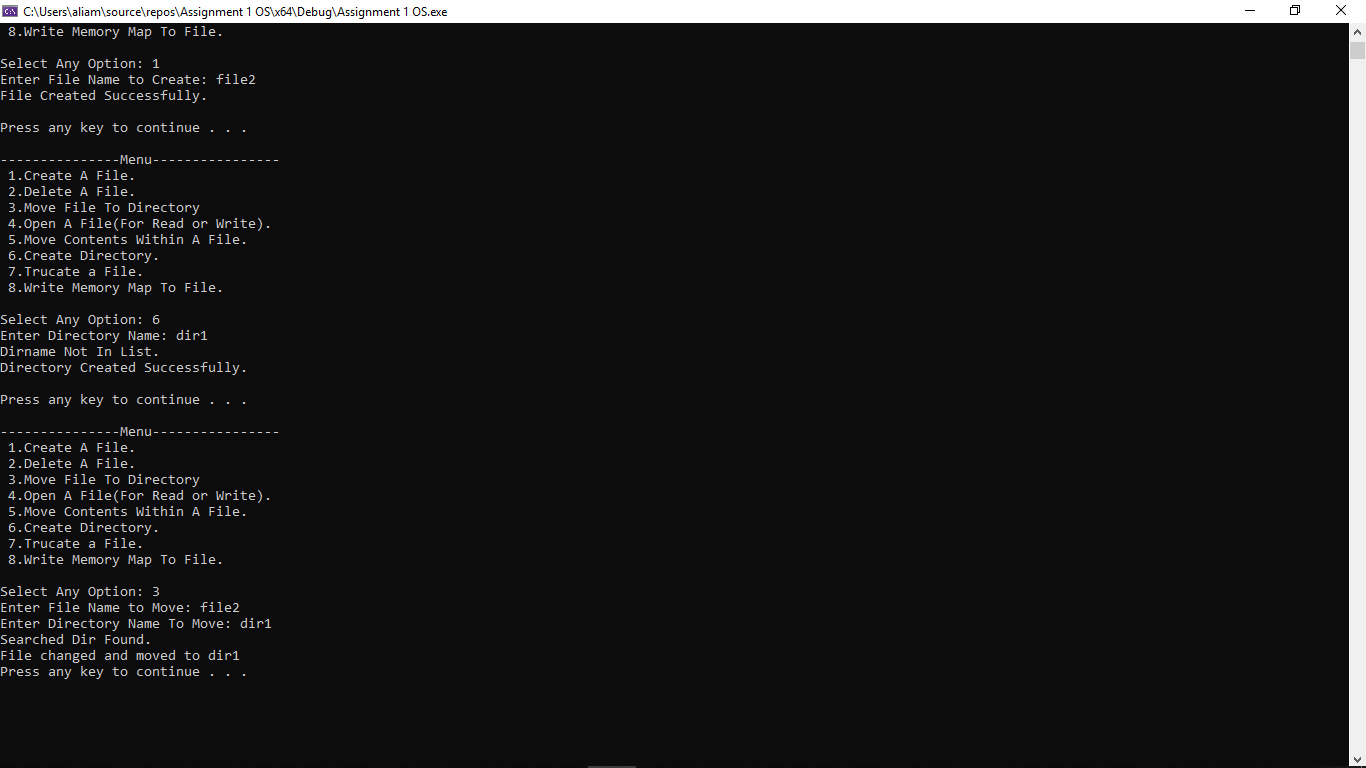
* Delete A File.





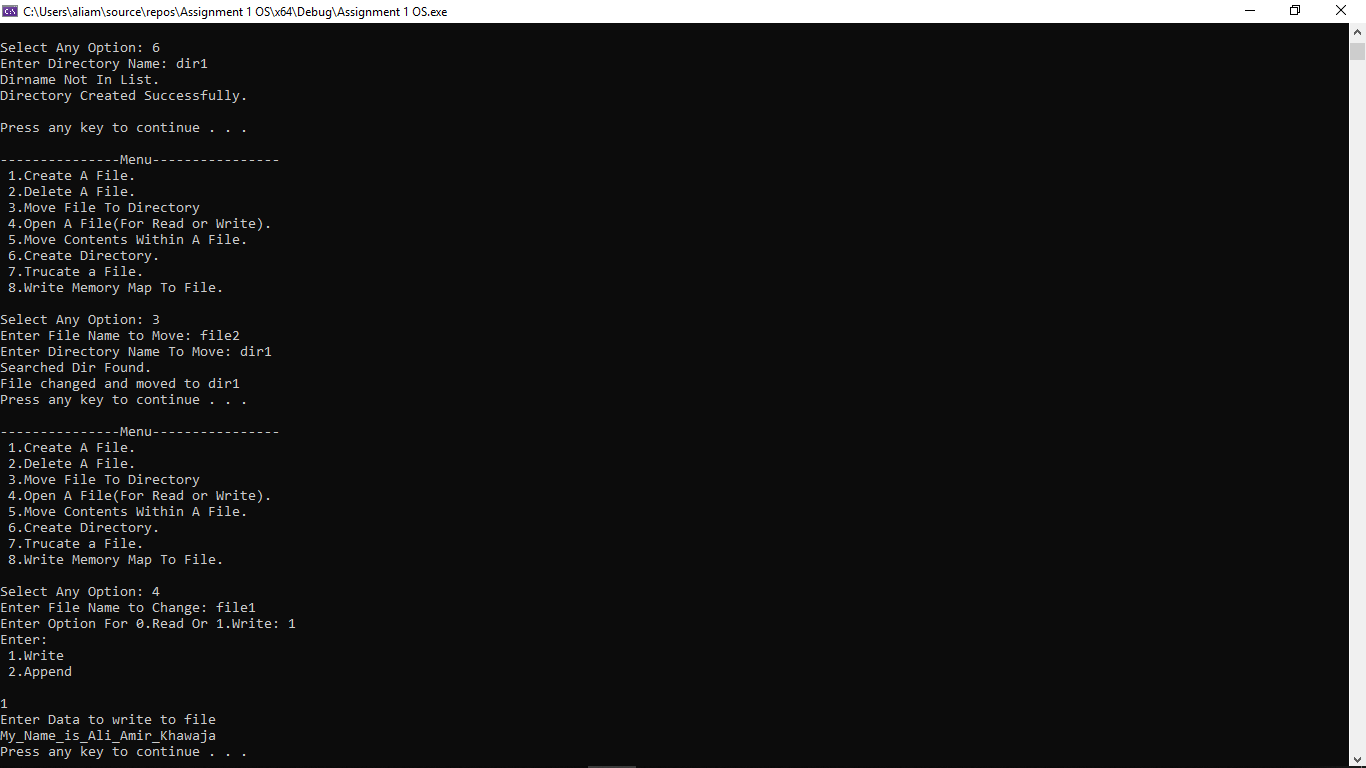
If not in memory then output relevant data.

* Move the File To Directory.

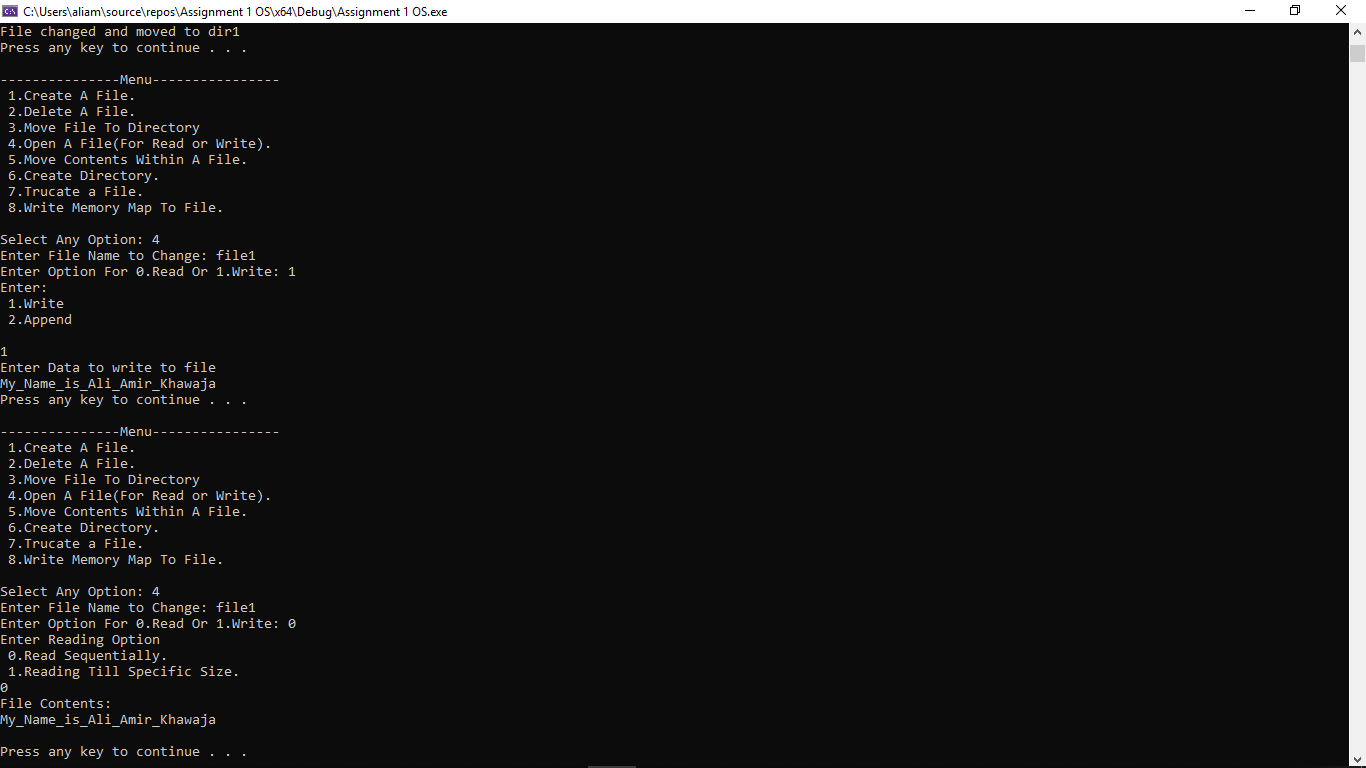


* Open A File(For Read or Write).

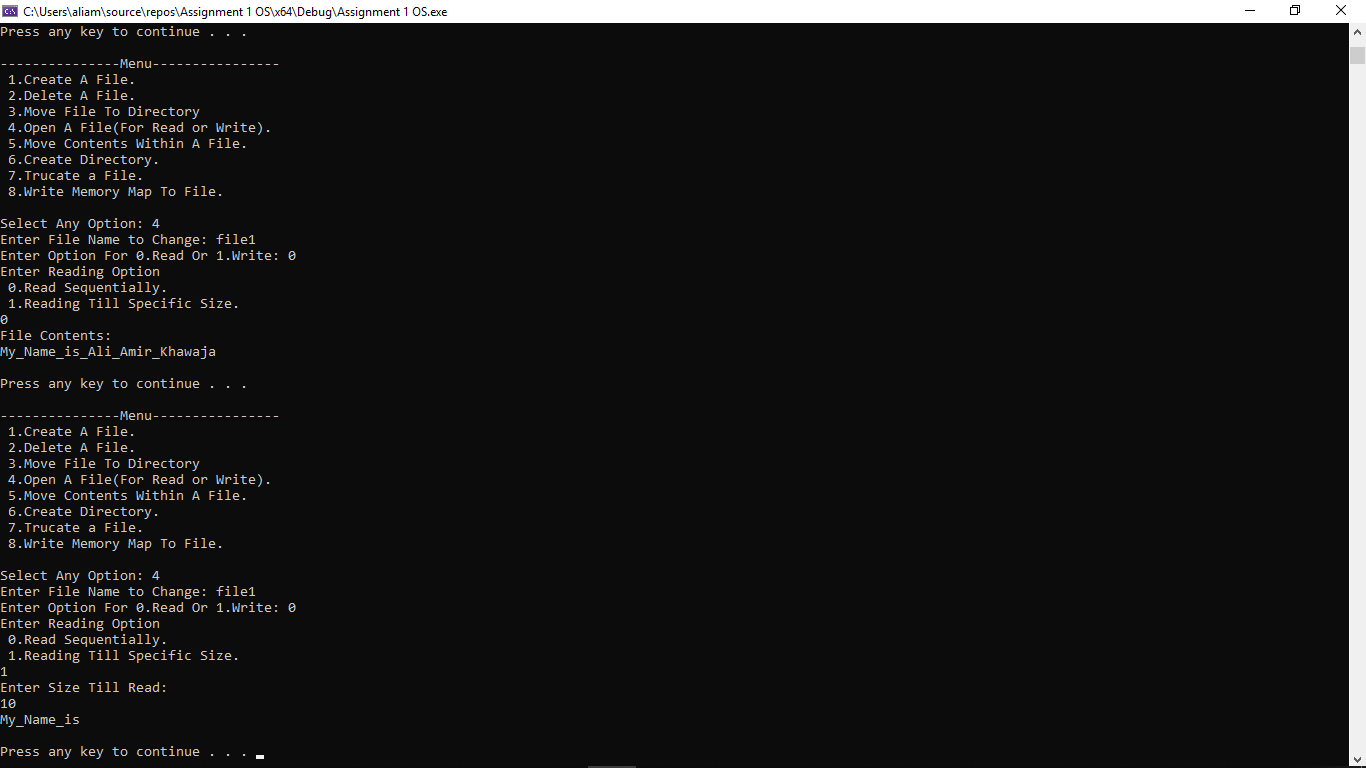
Writing to File1:



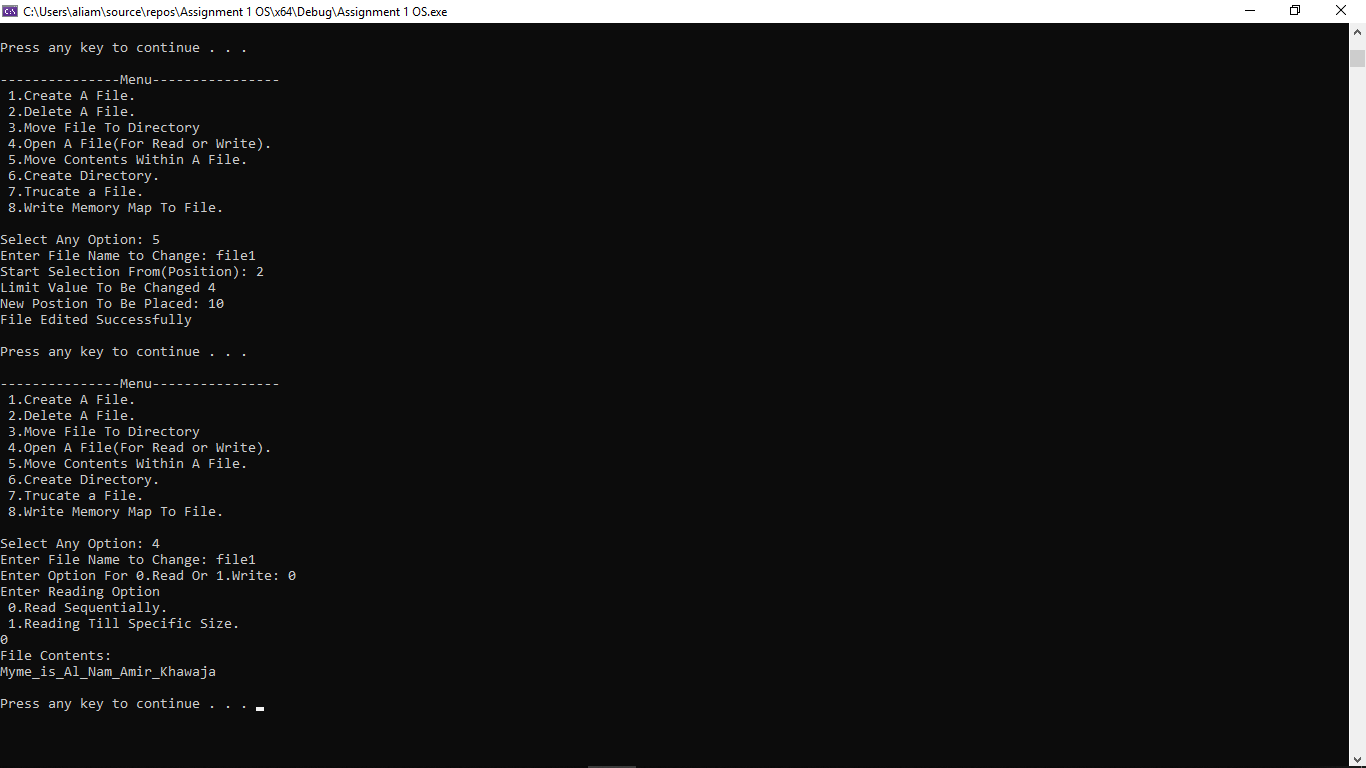
Reading the contents of the file1:



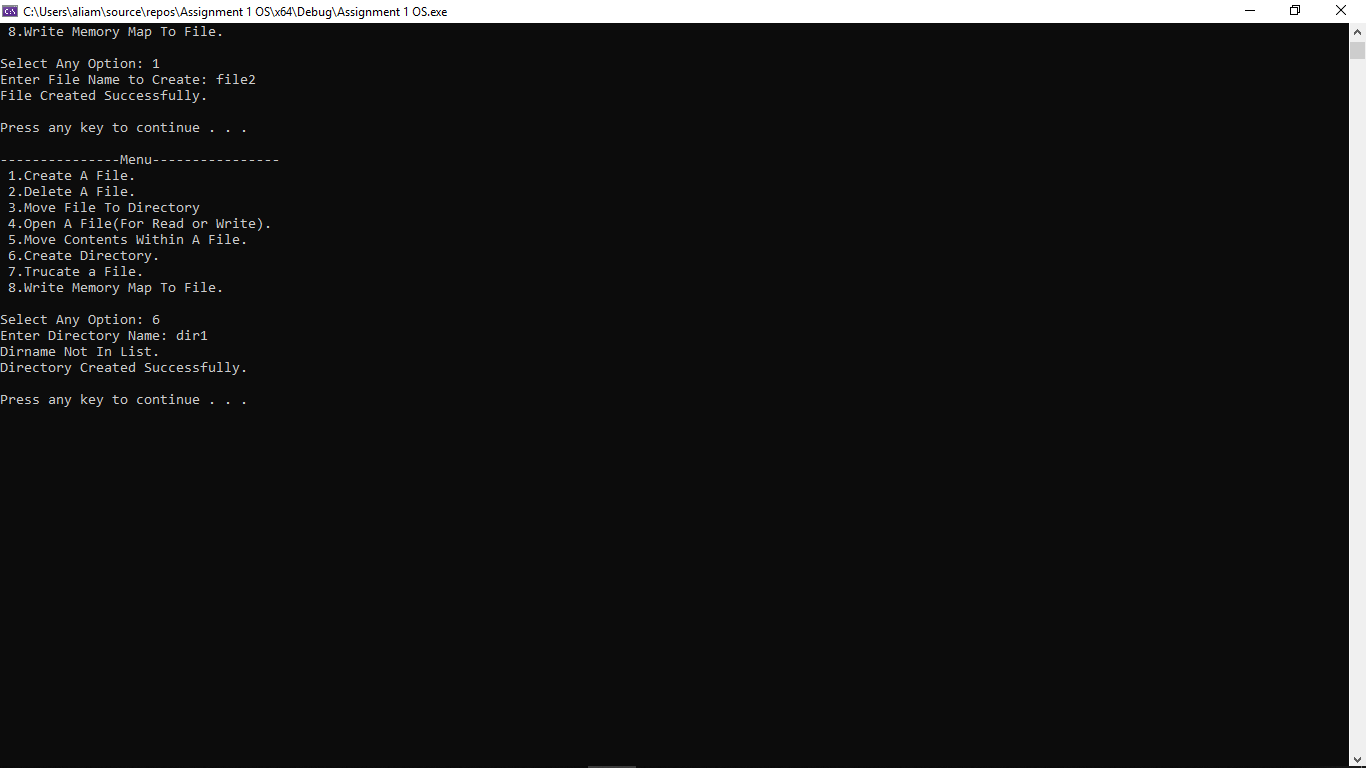
Reading Sequentially:



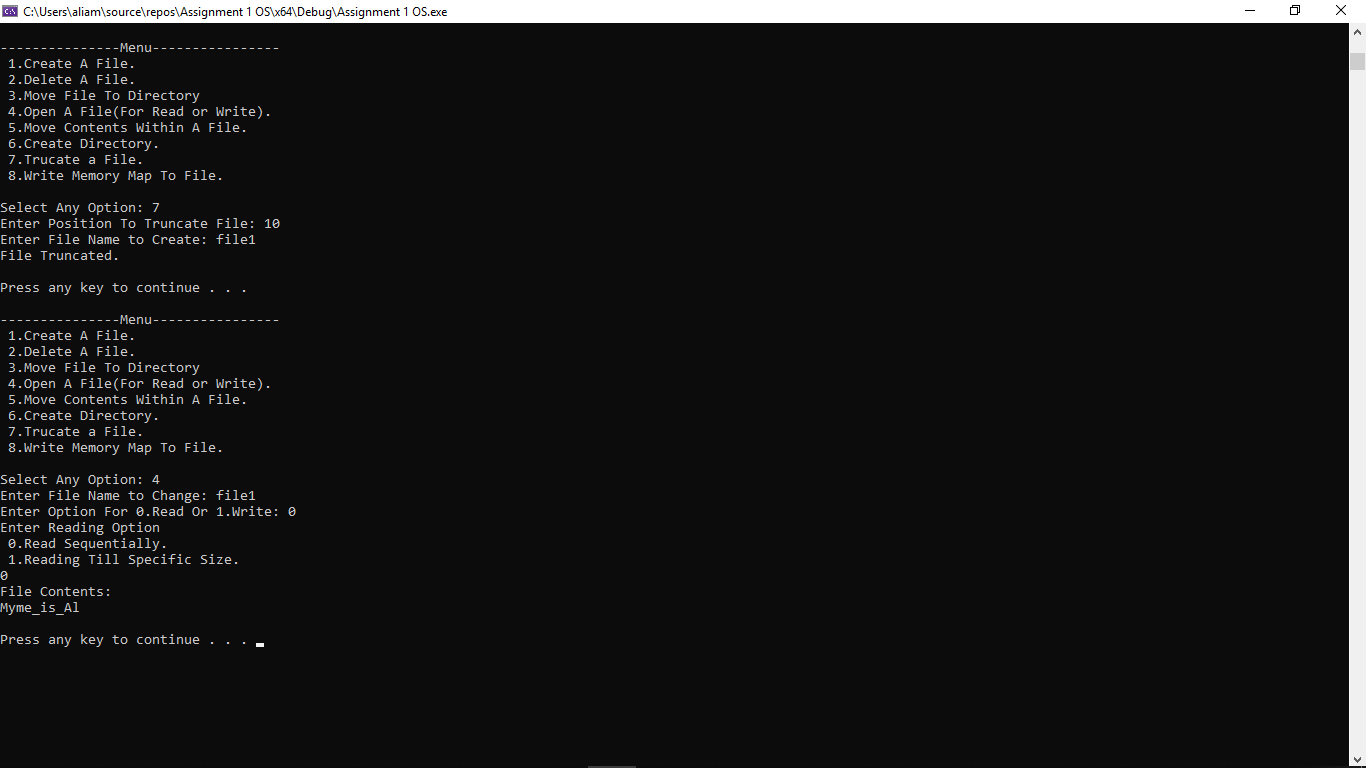
* Move Contents Within A File.



* Create Directory

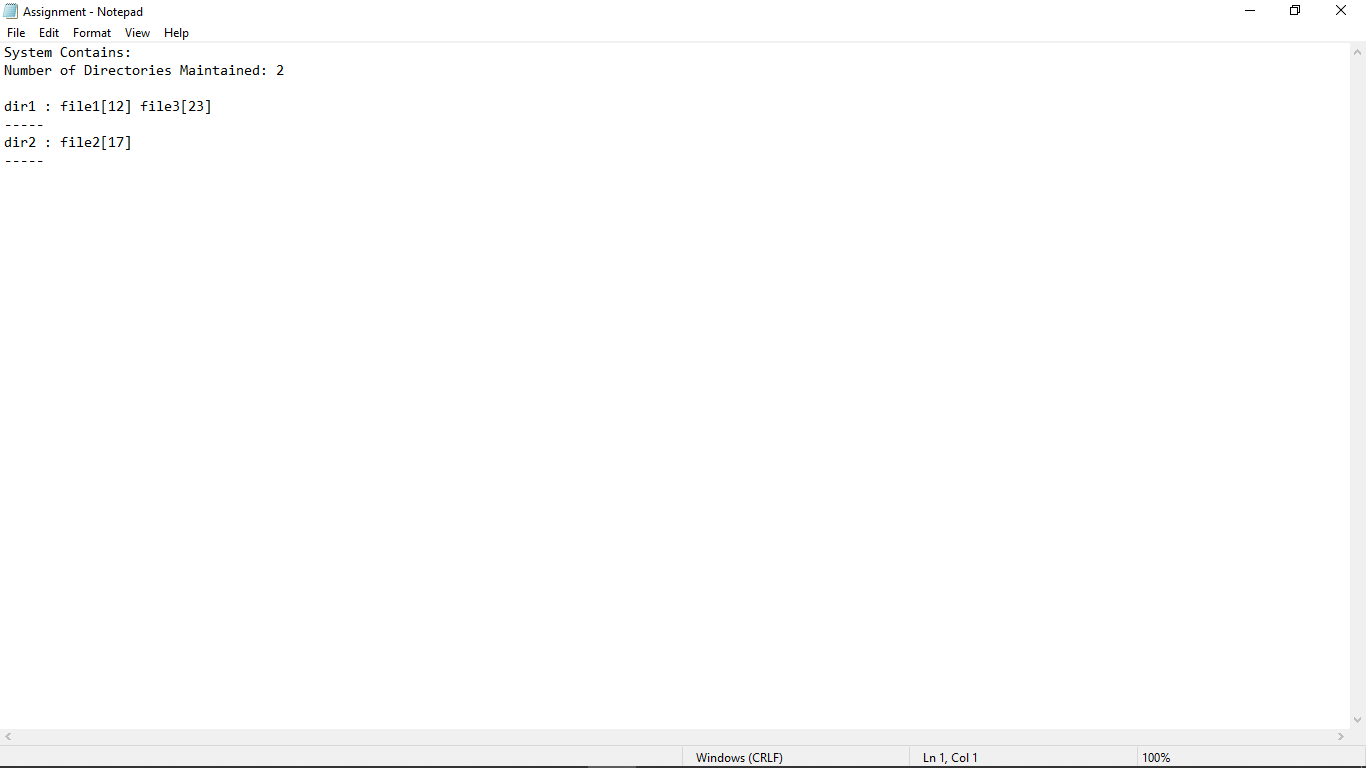


* Truncate a File.



* Write Memory Map To File.

Successfully Written to Dat file. Displays the number of directories. The files with their location in a specific folder and the size of each file are written in the .dat format file.

****

1. **References**

References that we used are:

* Geeks for Geeks
* Tutorial point
* HackerEarth