**CAPSTONE PROJECT (B-TECH SEM-2 IT205/SECTION B)**

**TEAM NAME: - Project A**

**PROBLEM: - P5 (Folder Cleaner)**

**STUDENTS: -**

**202301482 - Rushil Soni**

**202301488 - Manthan Gajera**

**202301457 - Soham Rathod**

**202301418 - Shivansh Shekhar**

**GITHUB REPOSITORY LINK: -** https://github.com/RushilSoni-UniAcc/Project-A-P5

**DESCRIPTION OF THE PROJECT: -**

In this problem we were required to make a Folder Cleaner which deleted files based on following criteria’s: -

* Files which are older than a particular date (input provided by the user).
* Empty files.
* Files that have not been accessed enough number of times (the required minimum number of accesses will be taken from the user).
* Redundant files (files which are of same size and content, so the newer one is kept in folder and older file is deleted).
* **ASSUMPTIONS: -- Input data of files is given in a .csv file by us which contains Folder name, Files in that folder with their name, File creation and modification date, Number of times accessed (program works for any number of access) , Size of file in bytes and URL of that file in our machine.   
  Input .csv file is attached in the GitHub repository.**

**PSEUDOCODE FOR PROPOSED ALOGORITHM: -**

Start

Linked list{

string name

string created\_date

string last\_accessed\_date

string last\_modificated\_date

int access\_count

long size\_of\_file

string URL

node\* next

EndClass}

Input( .CSV file )

{

{User confirmation(file) }

While( EOF)

{

Option

{

1) remove redundant files

2) remove old files

3) remove empty files

4) remove low access files

5) All of the above

}

Push(Row\_Data, &node)

{

If( Option 1)

{

Remove\_redundant(node \*file) //checks if file is redundant(same size and content)

//function is of time complexity Big-O- O(n^2)

}

If( Option 2)

{

Bool is\_old(node \* file, input date) //checks if file is older than a particular input date from user

// function is of time complexity Big-O- O(1).

If( is\_old) { Push(Bin\_linked\_list)}

Else { push Linked\_list}

}

If( Option 3)

{

Bool is\_empty(node \* file)// checks if file is empty or not.

// function is of time complexity Big-O- O(1)

If( is\_empty) { Push(Bin\_linked\_list)}

Else { push Linked\_list}

}

If( Option 4)

{

Bool is\_low\_accessed(node \* file)// checks if file is empty or not.

// function is of time complexity Big-O- O(1)

If( is\_low\_accessed) { Push(Bin\_linked\_list)}

Else { push Linked\_list}

}

If( Option 5)

{

Bool is\_valid(node \* file) //compares with all the functionalities mentioned in all

above 4 options.

//function is of time complexity Big-O- O(n^2)

If( is\_valid) { Push(Linked\_list)}

Else { push Bin\_linked\_list }

}

Endwhile

}

Void Create\_csv(node \*Linked\_list , node \* Bin\_Linked\_list)

{

Create( “To\_keep.csv”, Push(Linked\_list)) // function is of time complexity Big-O- O(n)

Create( “To\_delete.csv”, Push(Bin\_Linked\_list)) // function is of time complexity Big-O- O(n)

}

Confirmation

{

Display(“To\_delete.csv”)

Print “Are you sure you want to delete?”

Input (y || n)

If( y)

{

Delete(Bin\_Linked\_list)

}

Else

{

Linked\_list ( Push( Bin\_Linked\_list ) )

}

}

Print “Program was terminated successfully”

EndProgram

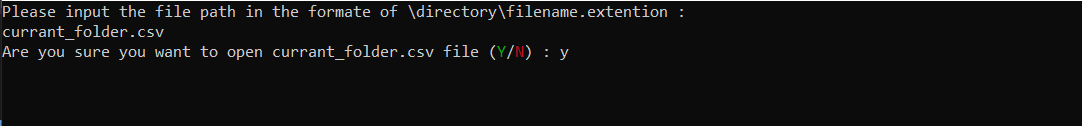
}

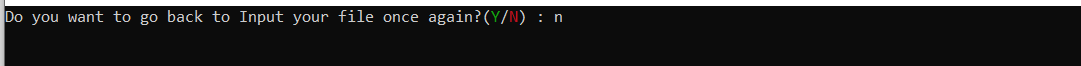
**Rationale for selecting Linked List with Dynamic allocation as data structure: -**

We have used Linked List with dynamic allocation as data structure as the input may be of 1,000,000 files or more. So, in storing large amount of file data may cause memory underflow if data is stored in static allocations. Also, the input csv file has multiple data types for which creating a self-defined linked list is more convenient to store data of a singular file.

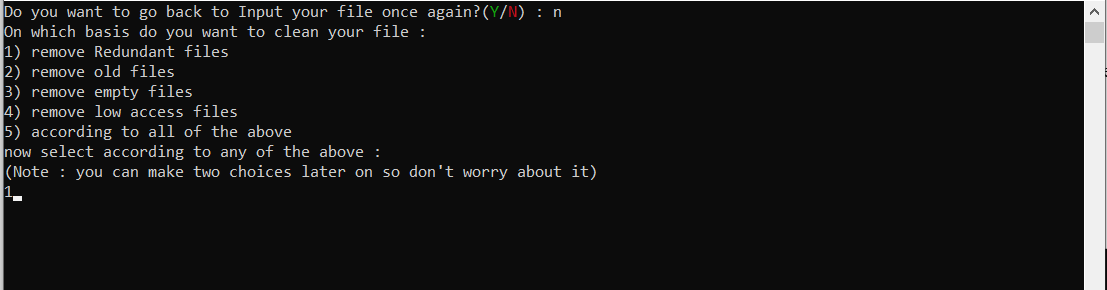
First when the data of a file is read, it is compared with other parameters, if it passes all the criterial only then it is pushed into the linked list otherwise its memory is deallocated and the file is removed.

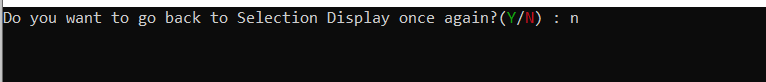
**OUTPUT: -**

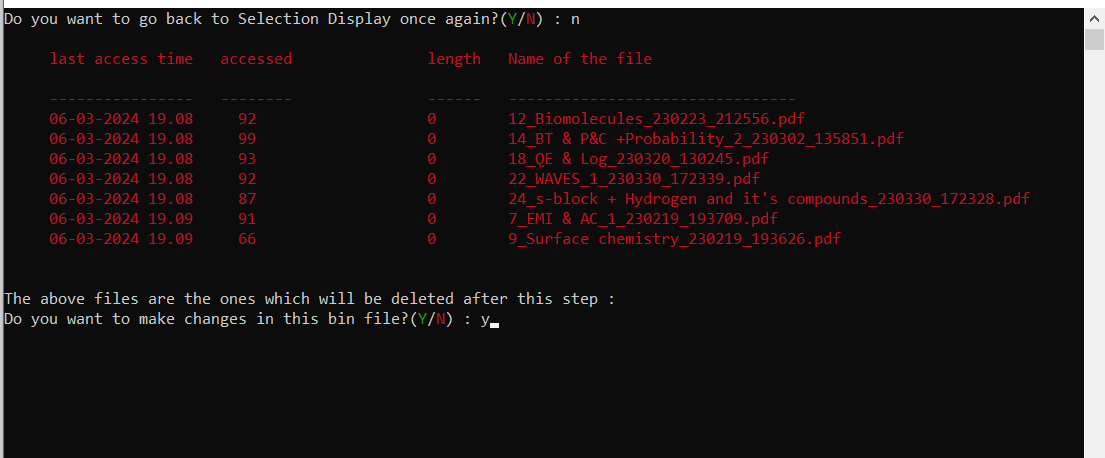




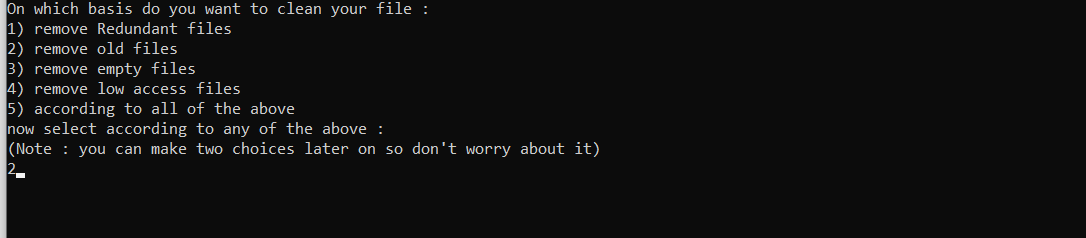
1st option:





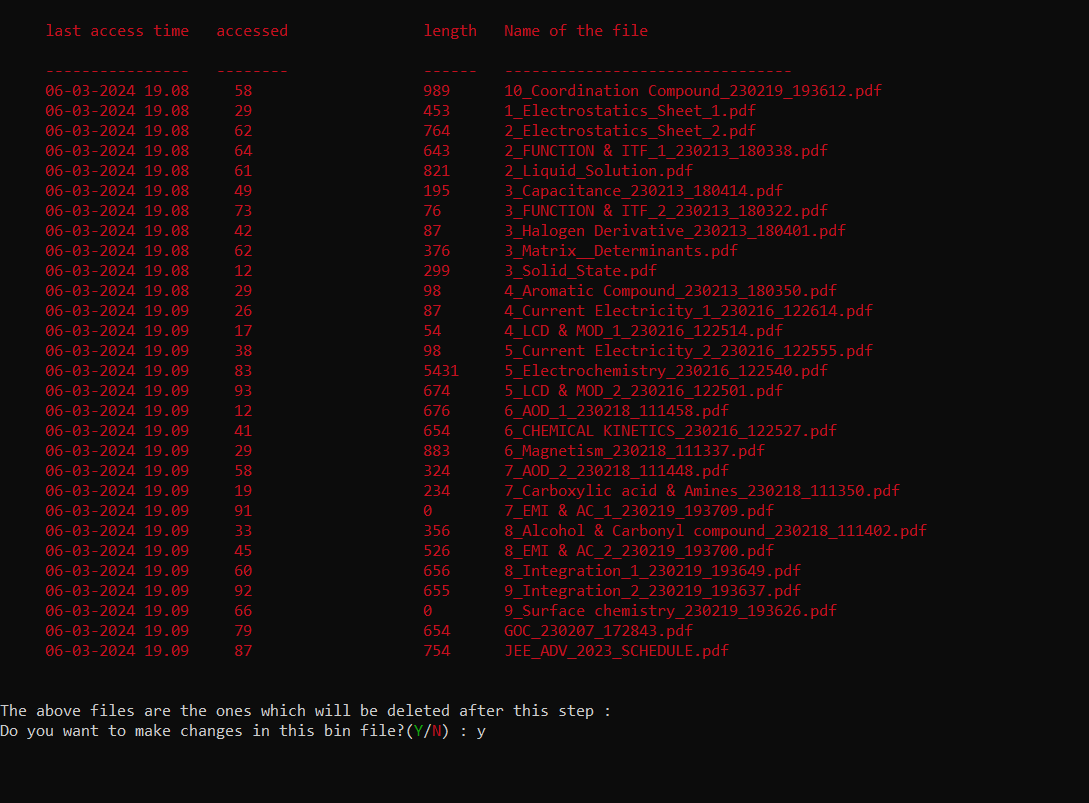


2nd option:

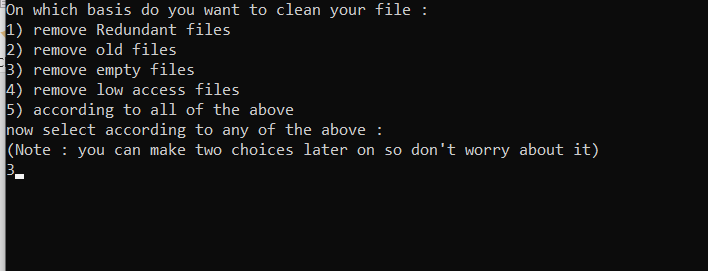


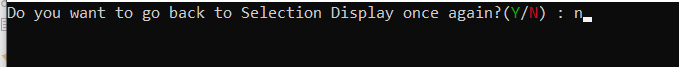


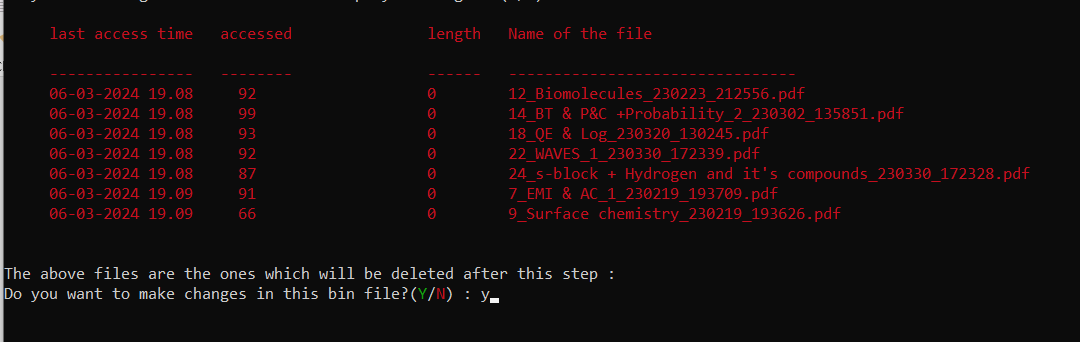




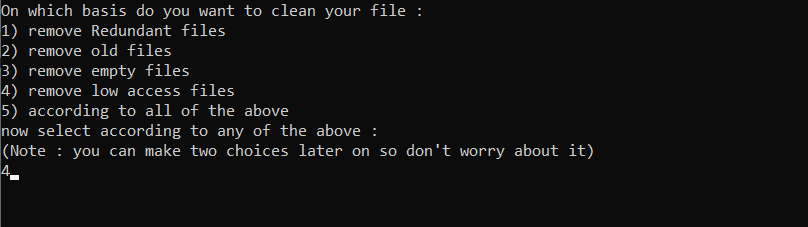
3rd option:

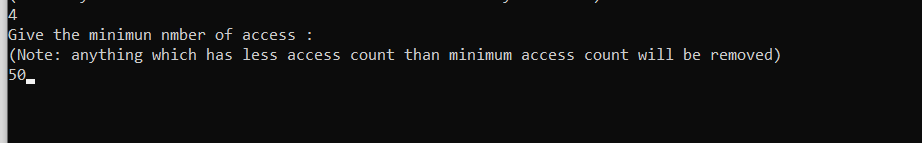




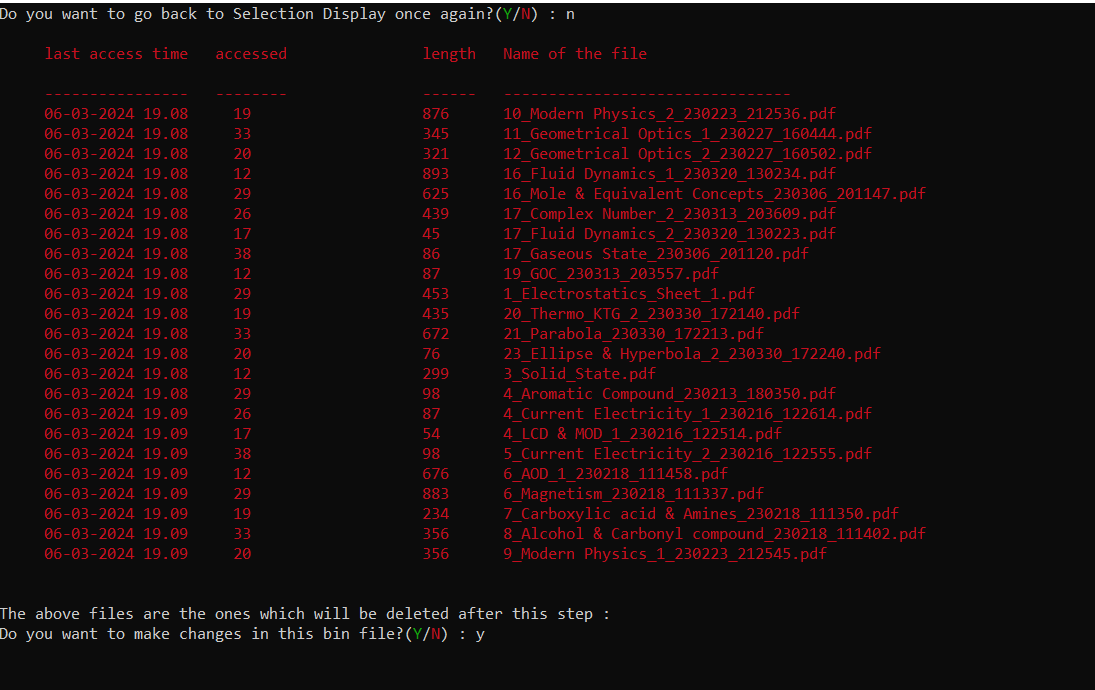


4th option:









5th option:

