# CS103 – Computer Programming – Spring 2015

**Semester Project**

**Abraxas Cleaner**



|  |  |
| --- | --- |
| **Developed by:** |  |
|  | **Ahsan Sohail – Section C – 14k-2085**  **Muhammad Feroz Uddin Ghauri – Section C – 14k-2182** |
| **Supervised by:** |  |
|  | **Miss Fareeha Sheikh** |

**Department of Computer Science**

**National University of Computer & Emerging Sciences (NU-FAST)**

Table of Content

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1. | Introduction | 3 |
| 2. | Need for the Project | 3 |
| 3. | Uniqueness of the Project | 3 |
| 4. | Implemented Classes | 4 |
| 5. | UML Class Diagram | 5 |
| 6. | Header Files | 5 |
| 7. | Algorithm | 7-9 |
| 8. | Flow Chart | 8 |
| 9. | Source Code | 10 |
| 10. | Screen Shots | 10 |
| 11. | Future Enhancements | 10 |
| 12. | References | 10 |

1. **Introduction:**

This application is a PC hard drive cleaner which scans for duplicate files in the given directory, and deletes them or moves the duplicates to a new directory given by user for further evaluation before deleting. This way data redundancy is lowered and hard drive stays free from useless files.

1. **Need of the project:**

This project is highly useful as most people don’t know that they should keep the hard drive clean to run the system faster. People download or save one file more than one time, accidentally or intentionally. This case is very common with pictures. People while making backups of their photos, save same pictures again and again which eats up extra space on hard disk. Not only pictures, videos, documents, or other files are kept with their duplicates. To overcome this problem AbraxasCleaner is developed. It scans the given directories for such duplicated files and deletes them or moves them to separate and delete later, and free up space on hard drive.

1. **Uniqueness of the project:**

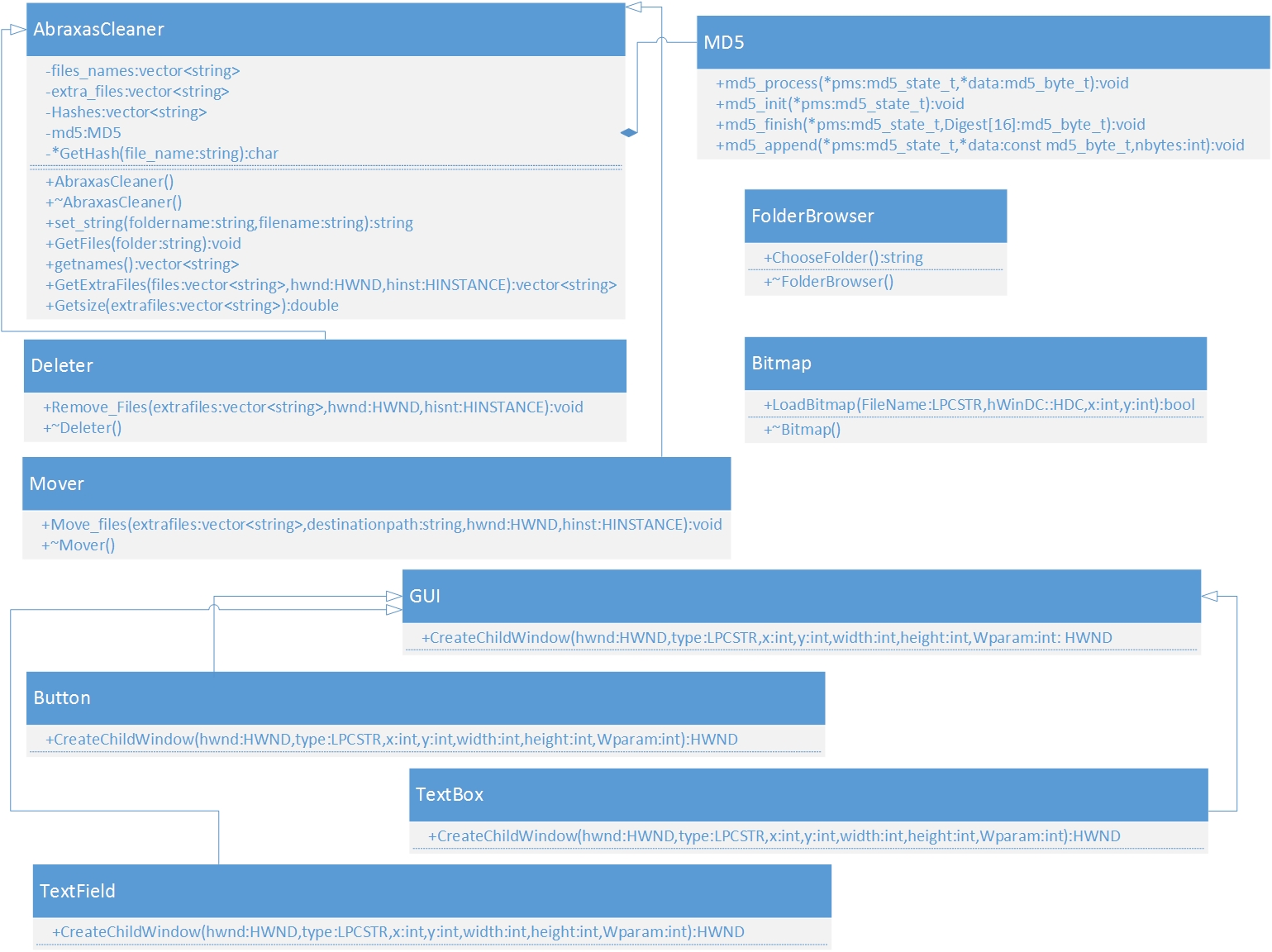
This software is run without any installation that means anyone can download and run this software who even do not know how to use a computer. This solution is very simple to use as it has very simple graphical user interface. User just have to select the directory with mouse, everything will be done automatically and user will just be prompted by options to decide what to do with files..

1. **Implemented Classes:**

We have implemented the following class:

|  |  |  |
| --- | --- | --- |
| **No.** | **Class** | **Purpose** |
| 1. | AbraxasCleaner | It is performing the functions of scanning directories, getting file names, storring duplicate files in a vector to perform desired operations on them later. A vector named *file\_names* of data type string which will store the file names after scanning the directory. A vector named *extra\_files* of data type string which will store the names of duplicate files along with their paths. A vector named hashes of data type string which stores the extracted hashes of all the files. An object md5 of class MD5. Our class contains a constructor and a destructor. A private function *Gethash* which will extract the hashes of all the files. A function *set\_string* of return type string which will append the filename at the end of its path to make a complete string which will be used to delete or move the file later. A mutator function *GetFiles* which will scan files in the directory given by user. An accessor function *getnames* which will return the names of the files. A function *GetExtraFiles* which will save the duplicate files in a vector. A function *Getsize* which will calculate the size of duplicate files scanned. |
| 2 | Deleter | *Deleter* class is inherited from *AbraxasCleaner*. It contains of a function *Remove\_files* which deletes all duplicate files. A destructor which will destroy the objects of deleter class when the program is terminated. |
| 3 | Mover | Mover class is inherited from our base class *AbraxasCleaner*. It contains a function *Move\_files* which will move the duplicate files to the directory/folder given by the user. |
| 4 | GUI | *GUI* is another base class from which three other classes are inherited. *GUI* class creates a window UI. |
| 5 | Button | *Button* class is inherited from our *GUI*. It creates five buttons, two browse buttons, one to delete files, one to move files, and one to exit. |
| 6 | Textbox | Textbox is inherited from our base class *GUI*. It creates two text boxes where user can input the path of the folder to be scanned, or where the files will be moved. |
| 7 | Textfield | *Textfield* is also inherited from *GUI*. It manages all the text which appears on our UI window. |
| 8 | Folderbrowser | This class has a function *choosefolder* of return type string. This function will open a window from which the user can choose the folder/directory which is to be scanned. A destructor which will destroy its objects after the program is terminated. |
| 9 | Bitmap | The class bitmap contains a function *LoadBitmap* of return type bool, which will tell if the bitmap image used in the application is successfully loaded or not. If not it will return error. A destructor which will destroy its objects after the program is terminated. |
| 10 | MD5 | It contains of a function md5\_process of return type void. A function md5\_init of return type void. A function md5\_finish of return type void. A function md5\_append of return type void. |

1. **UML Class Diagram:**



1. **Header Files:**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Header File** | **Functions** | **Purpose** |
| 1. | Shlobj.h | PROGRESS\_CLASS  PBS\_SMOOTH  PBS\_MARQUEE  PBM\_SETRANGE PBM\_GETPOS  PBM\_SETPOS | Used to create progress bar.  Used to make progress bar run smooth.  Used for the progress bar.  Used to set range of the progress bar.  Get the current position of the progress bar  Set the position of the progress bar. |
| 2. | vector | vector.push\_back()  vector.size() | Vector declaration.  Add element from back to a vector.  Size of the vector. |
| 3. | dirent.h | Opendir()  Readdir() | Open directory  To read data in the directory |
| 4. | Sys/stat.h | S\_ISREG(buf.st\_mode)  S\_ISDIR(buf.st\_mode) | To check if the data in directory is a regular file.  To check if the date in directory is a folder. |
| 5. | string | Strcat()  To\_string()  String.size()  Strcmp() | String declaration.  String catenation.  Convert int or double values to string.  Size of string  String compare |
| 6. | Windows.h | CreateWindow()/CreateWindowex()  SHbrowserforfolder()  MessageBox()  CreateFont()  SendMessage()  SetTextColor()  SetBkColor()  CreateSolidBrush()  BeginPaint()  DisableProcessWindowsGhosting()  GetWindowText()  LoadImage() | To create child and main window.  Allow user to browse for a folder in the computer.  Display popup messages.  Create custom font.  Send the specified message to window.  Set color of text of static and edit windows.  Set Background color of static and edit windows.  Paint Background color of main window using RGB values.  To paint bitmap and other images.  Disable window while performing actions.  Get text of edit window(get address of the directory given by user)  Load icons and bitmaps |

# Algorithm:

To find same files in the directory and delete them or move to specified folder to save memory, we require following steps:

* + - * First of all a main window is created by using CreateWindowex() function in windows.h library.
* Then our GUI class will create the child windows, this class is inherited by Button class, TextField class and TextBox class.
* Button class will create 5 buttons as shown in the screenshots below, two browse buttons ,one button to move files , one to delete the file and one to exit the program.
* TextField class creates the headings and sub-heading or all the static text shown in the screen shots below.
* TextBox class creates two edit windows, where user will enter the path of the directory to be scan or path of the directory where the duplicate files will move.
* A bitmap is loaded which is also our logo of the application using LoadBitmap function in our bitmap class.
* Our programs consist of parts one to deletes duplicate files in a directory and other is to move files in the directory provided by the user.
* To delete duplicate files user will first choose directory to be clean using first browse button which allow user to select any folder in his/her computer. This is done by the function SHBrowserForFolder() in library windows.h. This function is in our FolderBrowser Class.
* User will hit delete button to delete extrafiles from the directory provided by him/her.
* After taking the directory from the user we can extract files and sub directory from the base directory using dirent.h library. Functions S\_ISREG(buf.st\_mode) and S\_ISDIR(buf.st\_mode) are used to distinguish between files and folder. The files will be inserted in a vector; recursive function will be used to extract files from the subdirectories. These functions are present in the function GetFiles() present in our main class AbraxasCleaner.
* Once the vector is filled with the files in the directory provide by the user, function getname() will return the vector containing the path of the files.
* Function GetExtraFiles() present in our main class, will extra the extra/duplicate files and save the path of these files in another vector.
* Extracting duplicate files is our main task, it is done by comparing the MD5 hashes of each files with each other.
* First we have run a loop till the vector containing path of all files, extract hash of the file by using GetHash() function also present in our main class, this function uses the md5 algorithm and return the hash of corresponding file.
* Once we have hash of a file we will compare this hash with the hashes of all the other files present in the vector, if hashes of two files are same then these are duplicate of each other, one of them will be saved in a vector extrafiles.
* We have another vector Hashes which will save hashes of files, it is done to check that if the files of same hashes have already been checked or not, it is done to avoid extra processing.
  + For Example: A vector contains 7 files. 3 of them are duplicate of each other; the first loop will get the hash of file at position 0 and save it in a string and also in a vector Hashes.
  + The second loop will run from position 1 to the end of the vector, let say file at position 5 and 6 were duplicate of file at position 0, these files will be moved to a vector extrafiles.
  + When the first loop extract the hash of file at position 5, it will check in the vector Hashes, the hash of this file will be already in the vector Hashes, there will no need to check for the duplicate file of this file because it is already a duplicate of file at position 0 and have already been moved to a vector extrafiles.
  + Same is for file at position 6.
  + Normal process will be followed for the files at other positions
  + If this is not done, then it will take extra time to check each file with each other, and may save the address of one duplicate file more than one which is not required.
* Now we have a vector extrafiles, which contain the path of all the duplicate files.
* The program will confirm the user whatever to deletes these files or not.
  + This is done because files will be parmentally deleted from the system and cannot be recovered.
* This vector is passed in the function Romove\_Files() in our Deleter class. This function will delete all the extrafiles one by one in a loop.
* The directory is now cleaned from the extra/duplicate files.
* The application will display message informing the user about number of files deleted and the how much MB’s of memory is cleared, which is calculated by Getsize() function in our main class.
* For the second part, user will have to provide two path of the directories, one path of the directory to be scan, other the path of the directory where extra files will be moved.
* When user will hit the move button, all the functions call is same as it was for the delete button till we get a vector containing duplicates files.
* Then this vector will passed in the function Move\_files() in our Mover class. This function uses predefine function MoveFile() to move files from the origin to destination. But we only have the origin address of the files, and we have the address of the folder where the files is to be moved.
* We will the extract files name from the origin address by using following code:

while (fullpath[k]!='\\')

{

nametemp=nametemp+fullpath[k]; //get files name(it will be reverse)

k--;

}

K is the last element of the address.

For example if the origin address of a file is as follow:

C:\user\folder\filename.exe

The loop will began from the last element till ‘\’ and the file name saved in the string will be like this “exe.emanelif”.

We will reverse it by using following code:

for(int a=nametemp.size()-1;a>=0;a--)

{

name=name+nametemp[a]; // reverse name

}

We will now distinguish each filename with a counter number.

This is done because file with same name cannot be copied or moved to the same folder, to avoid this collision we have distinguish each filename from each other by the attaching a counter number to the file name.

Now we have the file name as “(some number)filename.exe”. We will attach this file name with the address of the destination folder provided by user using function set\_string() which contain following code:

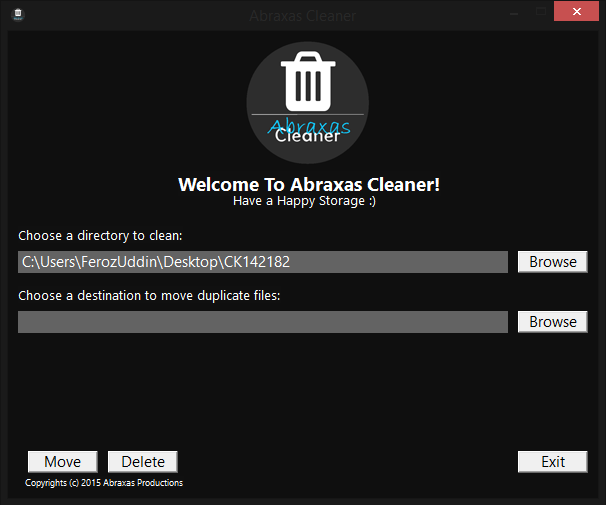
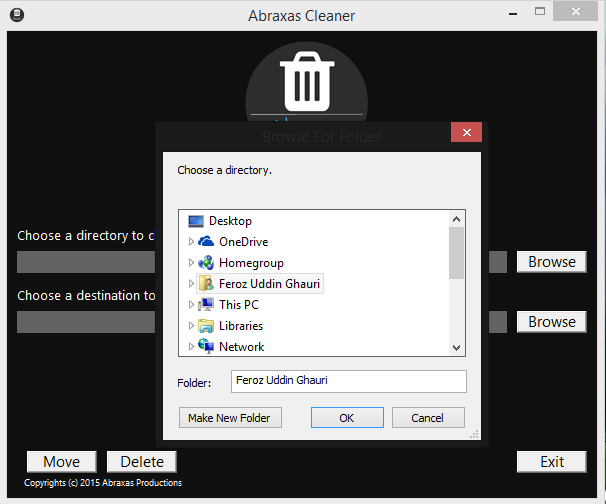
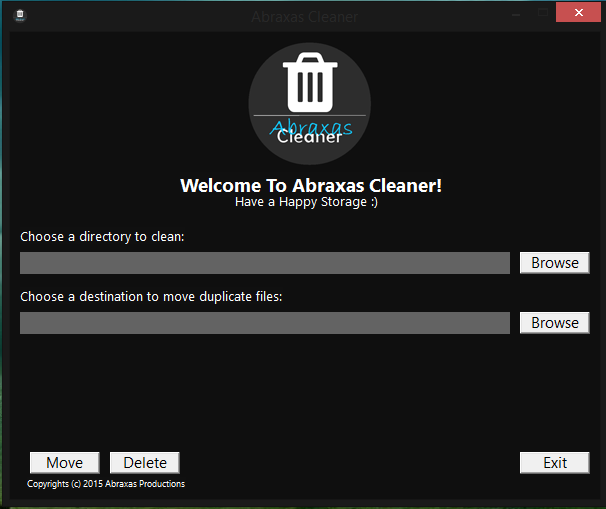
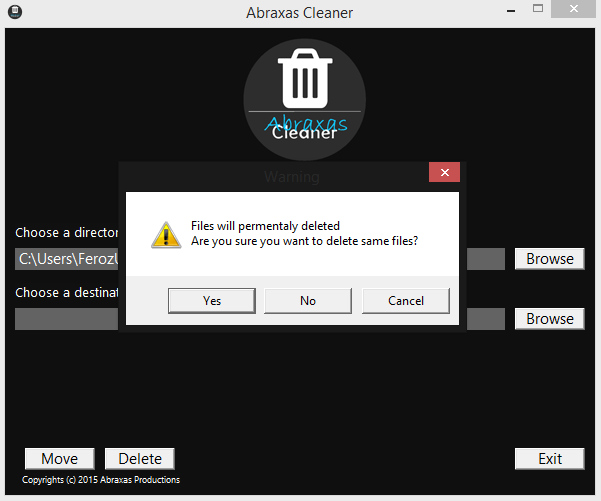
path=foldername + "\\" + filename;

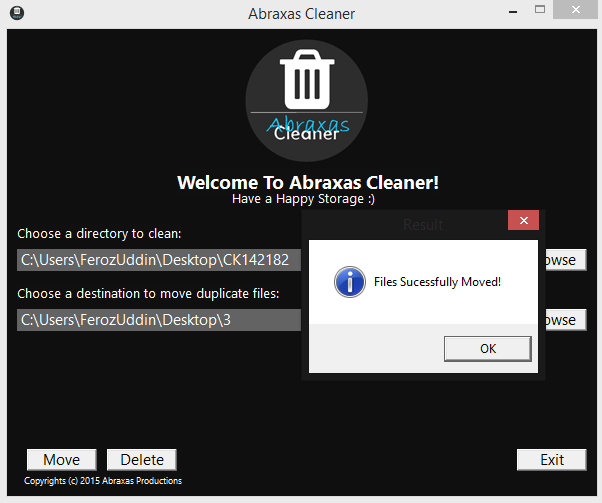
* Now we can use MoveFile() function to move duplicate files in the folder provided by user.
* The user can repeat the task by providing another directory till he/she hit exit button to exit the program.

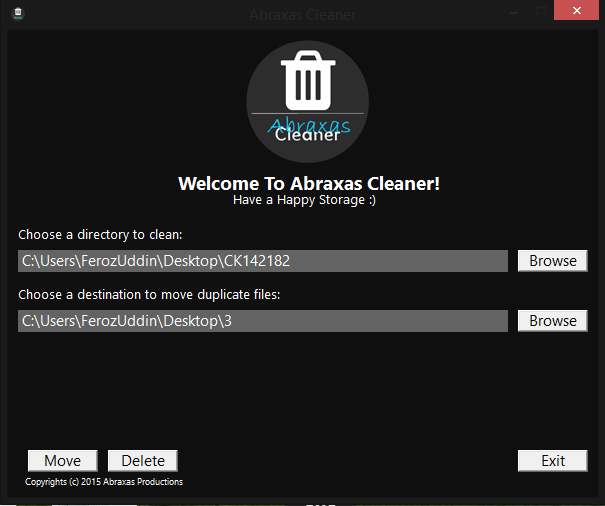
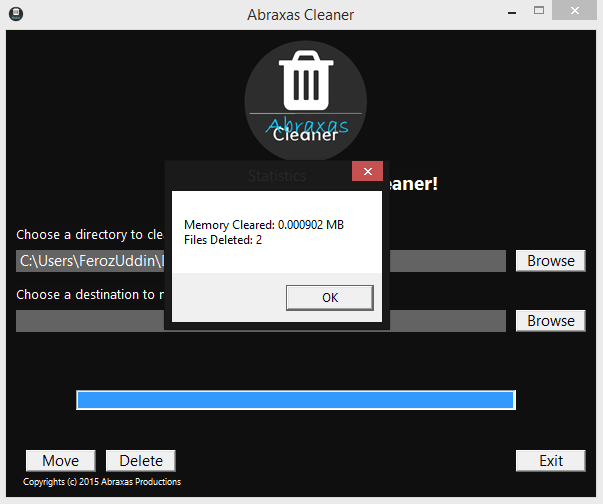
1. **Source Code:**

* Prayer\_Time\_&\_Qibla\_Direction.h

1. **Screen Shots:**

****



es. ­­­­­­­­

1. **Future Enhancements:**

This application can have more enhancements. We can speed up the processing time to scan larger directories in lesser time. We can include even more precise methods of compare files and give user more options to manage the files.

1. **References:**

The code for calculating md5 hashes was taken from a facebook group. The code contain the further details of copyright and fair usage with original reference