Oytun Kuday Duran 28357 PA 5 Report:

My process while solving the problem (I didn't understand if it was asked or not so I wanted to write):

To solve this problem, first of all, I watched the lectures about the content. Then, I made lots of Google researches about c programming, file pointers, directories and more. CodeVault youtube channel really helped me while writing this code. It took me many attempts and hours of research on youtube, stack overflow, other coding websites about c programming but eventually I finished it. As I tried, it compiles without problem using "gcc corrector.c -o corrector; ./corrector" command on terminal.

My C implementation:

First of all I defined a struct Node named Dictionarynode that has parameters for title of a person (Mr. or Mrs.), name and surname. Since I wanted to make a linkedlist, it also has a next pointer as usual.

To enlarge my linkedlist, I wrote a function named Addtodictionary which takes necessary information as parameters, then creates a node and allocates space from heap using malloc for informations. If a person is female, I directly give her the title "Ms.". I used strcpy() function to copy the parameter strings to the destination strings in new node. If the list is empty, I assign newNode as head. Else, I iterate until the next node is NULL and then I assign newNode to the tail.

In ReadDatabase function, I open the databasetxt, and using a file pointer, I iterate over all the words in the txt. I have 2 boolean variables, whenever I get an input from the database, I assign one of them as true so that I know what the next parameter will be. By checking bool variables, if I see I read the name and title, after taking the surname, i assign boolean variables as false so that we know a line in database.txt is over. Then, I copy input to my char array variable named surname just like I did with others (I took the gender as a char since its either 'f' or 'm'). I initialize head using Addtodictionary function using the information we gathered. There were no problem since I always strcpy() the strings to the necessary areas in my nodes.

In bool returning function Dictionaryfind, I simply iterate using a ptr initialized to head at the beginning. Until I see if the char* name parameter is in Dictionarynode's name area, i keep going to the next Dictionarynode. If I can't find the name (Word in txt in this case), the ptr becomes NULL since we iterated over all the list. If it is not null, i return true as I want to inform that the given Word is in our dictionary list. I have a global variable named findnode, If we can find the name in the dictionary, I assign the corresponding node to this to use in other functions. If we can't, I return false and make findnode point to null.

In correctfile function, I take directory name and the filename as parameters and use concenate function strcat() to merge them to one char array named txtname. I then use a file pointer again and check Word by Word. If Word is in our dictionary (Using above function), I make file pointer point to beginning of title and surname using fseek() one by one and use fputs() to extract true information from my findnode. So, the title and surname gets corrected. Here, I need to use " -lengthofname-4" to make pointer to reach to beginning of title from the name using file pointer, Similarly, I need to use “lengthofname+2” to reach the beginning of the surname with file pointer. I close the file using filepointer as parameter.

In destructor, I simply iterate over the dictionary linked list using 2 pointers, and free the spaces allocated using free() function until all the pointers are deallocated. The second pointer here named temp keeps the next node information since we won’t be able to access the next node when current node is deleted. I assign temp as the current pointer to keep iterating until temp(next node) is null.

In searcher function which takes a char * parameter named dir, I first create a directory entity pointer named directent. I then use a directory pointer and try to open the directory if possible. If not, i close the directory using directory pointer and end the function. If we can open the directory, I use our directory entity to read the directory. While it is possible for it to read (not null), firstly, I create a char array to see the extension of the file. I use memcpy() function to copy the directory entity’s last 4 characters to our extension string and then add the string ending char ‘\0’ at the end. Here, I assumed that the length of any directory entity won’t be less than 4, otherwise, It may not be possible to check if it is a txt or not. After getting the extension of the directory entity, I check whether if it is another directory or a file.

If it is a file (If the d_type is DT_REG), I be sure that it is not our initial database.txt file. I also be sure that It has the extension “.txt”. Then, I create a char array to copy the directory entity’s name. I then send the current directory and the directory entity names to the corre ctfile() function.

If its another directory (If the d_type is DT_DIR) and not “.” and not “..” (I learned this from the CodeVaults youtube channel, to not be stuck in a loop), I create a char array named recursion. Using sprintf() function, I copy our current directory + the slash to go to our next direction(/) + the directory entity’s name. Since it is another directory, we most probably will be able to search in it. So i use a recursion call here and use the Searcher function using this recursion string.

After we go through the function, I close the directory. Also in while loop, we keep updating directory entity using readdir() function in current directory to check all files in the directory.

In main, I simply assign NULL to head, call ReadDatabase(), Searcher() and Destructor() functions. I send Searcher() the “.” As a parameter since we want to start the directory in which the c file is located.