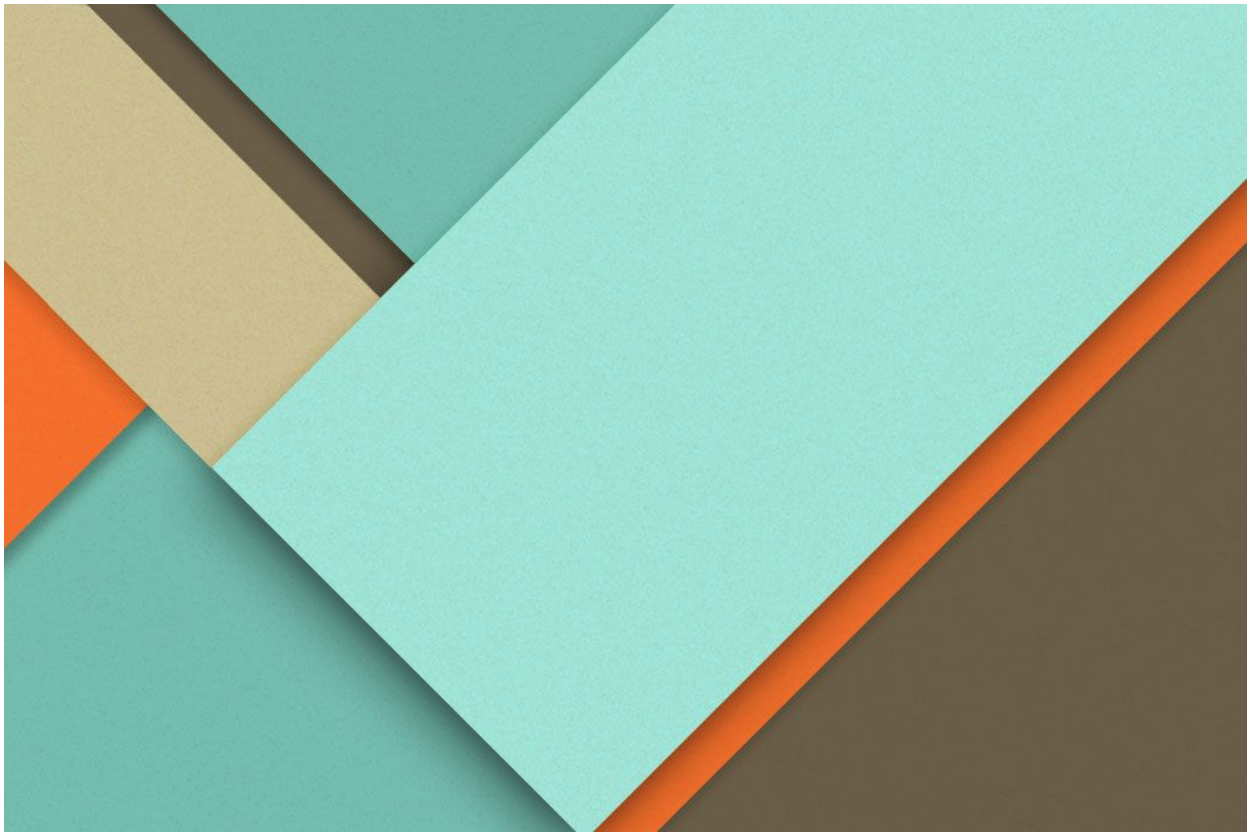

The Personal Dictionary

Winter Semester 2018

Group Members

Achyut Tripathi - 17BCE0954

Poulami Paul - 17BIS0082

Kirubakaran Nithiya Soundari - 17BCE2244

Sridhar Suraj - 17BCE2245

Problem Statement

Creating a personal dictionary using trie data structure and file handling.

Data Structure(s) Used

In this project, we focused mainly on the Trie data structure. Trie is an efficient information re***Trie***val data structure. Every node of Trie consists of multiple branches. Each branch represents a possible character of keys. We need to mark the last node of every key as end of word node. A Trie node field *isEndOfWord* is used to distinguish the node as end of word node.

It is said that a binary search tree data is the most efficient one of them all, and it is true. But in this application of formulating a program for a personal dictionary, the type of data which has to be stored is strings. This storing and retrieval of string data types are very efficient with Trie data structures. It is also to be noticed that with a Trie, we can insert and find strings in O(L) time where L represents the length of a single word. This is obviously faster than BST. This is also faster than Hashing because of the ways it is implemented. We do not need to compute any hash function. No collision handling is required (like we do in open addressing and separate chaining). Another advantage of Trie is, we can easily print all words in alphabetical order which is not easily possible with hashing. It is also possible to do prefix search or auto complete efficiently with a Trie data structure.

In this program, the displaying of search history is done due to the use of a stack data structure using a linked list.

List of Functions / Features

In this program the following functions are used:

(a) Inserting data into the dictionary
(b) Deleting data from the dictionary
(c) Searching data from the dictionary
(d) Viewing the search history
(e) File Handling

Insert Function

This characteristic of this function is to write data, that is here in this case the word and the meaning for this word to be stored in the Trie data structure as well as the “dictionary.txt” file which has been created to store the data. This function accepts the word (in the program it is ‘key’) and the meaning of the word (in the program it is ‘mean’) as a character array. Then using a for loop, every character of the word is stored in the form of the Trie data structure. The last character of the word is stored as and its isEOW (i.e., end of word) is set to true indicating the end of the word. Then the meaning of the word is copied to the end of the word using a pointer. This is then written into the dictionary.txt file.

Delete Function

This function deletes the word and the meaning from the file. Here there will be two functions in the program, one to check the word is in the file and to delete it and the other to check whether the word is in the Trie data structure and to delete it. In the function to delete the word from the file it checks if the word is present in the file, if yes then it will extract the next word(meaning) also, thereby preventing the word and the meaning from being written in to the temporary copy file,thus essentially deleting them. If the word is in the trie data structure, a temporary pointer traverses the Trie tree and deletes the word by changing the EOW of last letter of delete word to false.

Search Function

This function searches for the word the user has entered. If the word exists, the word and the meaning get printed, else it returns saying that the searched word was not found. More is explained in the comments of the code.

Search History Function

This function displays the search history of the user. Here stack data structure was used. When the user is in the search function entering which word he/she wants to search we copy this onto another variable and this variable is called into the push function which is responsible for storing the search history. When the user wants to see the search history he/she will enter the particular option and it gets printed due to the display_history function.

List of Errors and it's Rectification

I. Storing of Data in the .txt File

When we first implemented the file handling in the insert function:

(a) Instead of 'ios::app' we used 'ios::out'. So whenever we ran the program the data which was previously stored was lost. After changing into 'ios::app' the file did not erase all the previously stored; it continued storing more data line by line below it.

(b) Both the word and the meaning of the word were stored in the same line with a space separating them. This did not enable the search function to operate properly. The error we faced was the search function printed the word but did not print the meaning of the word, instead printed the next word. So we changed the way we store the word and the meaning in the text file by storing the meaning just below (on the next line of) the word and leaving another line before the next word. Now the search function worked properly - it printed the word and the meaning.

II. Delete

We had initially planned on deleting the word only from the trie tree,but the problem was that in the search function would check for the word in both the file as well as the trie tree. So even if we deleted the word from the trie tree,when search operation was performed for the word,it would show a positive result along with its meaning because the word still existed in the file.Thus it was necessary to include file handling for deletion.Now to delete the word from the file we were considering many approaches,like using seeg, tellg and pointers but it was getting very complicated,so we thought of copying all the words except the delete word and its corresponding meaning into a temporary file and then renaming it with the name of the original file and simultaneously deleting the original file,thereby deleting the given word and its meaning.

Code

```
//Following are the header files used in our project
#include<iostream>
#include<stdio.h>
#include <stdlib.h>
#include <windows.h>
#include<string.h>
#include<fstream>

using namespace std;

#define total_alphabet (26)                    //Alphabet size (# of symbols)
#define char_to_index(c) ((int)c - (int)'a')                    //Macro definition to Find the index of a particular english letter

struct TrieNode
{
        struct TrieNode *children[total_alphabet];
        char meaning[50];                    //To store meaning of the word
        bool isEOW;                    //Non zero if word is completed
};

typedef TrieNode TrieNode;

TrieNode * newnode()        //// Returns new trie node (initialized to NULLs)
{
        int i=0;
        TrieNode *newnode = (TrieNode*)malloc(sizeof(TrieNode));
```

```
        newnode->isEOW = false;
        for(i = 0;i<total_alphabet;i++)
                        newnode->children[i] = NULL;
        memset (newnode->meaning,'\0',50);
        return newnode;
}

struct history                  //History node use to implement stack data structure for
Recent history function
{
        char data[50];
        struct history *next;
}*top=NULL;

void insert(TrieNode* root)                     //If not present, inserts key into trie If the key is
prefix of trie node, just marks leaf node
{
        ofstream file;
        file.open("C:\\Users\\Achyut Tripathi\\Desktop\\dictionary.txt", ios::app);
//Opening file to Write Words and meanings
        xy2:
        char key[50], ch, mean[50];
        memset (key,'\0',50);
        memset (mean,'\0',50);
        cout<<"\n\nEnter a word: ";
        gets(key);                      //Getiing Word
        fflush(stdin);
        cout<<"\n\nEnter Meaning: ";                  //Getting Meaning
        gets(mean);
        fflush(stdin);
```

```
        int i, j, k;
        j = strlen(key);
        TrieNode* temp = root;
        for(i = 0; i < j; i++)                  //Traversing through trie node
        {
                        k = char_to_index(key[i]);
                        if(!temp->children[k])
                                temp->children[k] = newnode();
                        temp = temp->children[k];
        }
        strcpy(temp->meaning, mean);
        temp->isEOW = true;                     // mark last node as leaf
        {
                        file.write(key,(strlen(key)));                  //Writing word on file
                        file<<"\n";
                        file.write(mean,strlen(mean));                  //Writing meaning on file
                        file<<"\n";
                        file<<"\n";
        }
        cout<<"\n\nDo you want to add more words?(y/n): ";
        cin >> ch;
        fflush(stdin);
        if(ch == 'y')
                        goto xy2;
        file.close();
}

void delette_file(TrieNode* root, char * key)           //Deleting word and meaning from
file
{
```

```
   char word[50];
   int temp = 0, temp1=0;
        fstream fil;
   ofstream o;
   o.open("C:\\Users\\Achyut Tripathi\\Desktop\\tempdictionary.txt",ios::app); //opening a temporary file to copy all of the data except the word to be deleted
   fil.open("C:\\Users\\Achyut Tripathi\\Desktop\\dictionary.txt",ios::in);//opening original file
   if(!fil)
        {
             cout<<"File is not found";
             exit(0);
        }
        else
        {
             while(!fil.eof())//iterating through the file  till it reaches the end
             {
             fil>>word;
                       if(strcmp(key, word)==0) /*checks if the word is present in the file,
                                                 if yes then it will extract the next word(meaning) also,
                                                thereby preventing the word and the meaning from being written into
                                                the temporary copy file,thus esentially deleting them*/
             {
            fil>>word;
             temp1=1;
             }
```

```
                else
                {
                                o.write(word,strlen(word)); //coping the rest data into
temporary file
                        temp++;
                        if(temp%2==0)
                                o<<"\n\n";
                        else
                                o<<"\n";
                }
                }
                if(temp1==0)
                        cout<<"\n\nWord NOT Found !!!!";
        }
        o.close();
        fil.close();
        remove("C:\\Users\\Achyut Tripathi\\Desktop\\dictionary.txt");
        rename("C:\\Users\\Achyut
Tripathi\\Desktop\\tempdictionary.txt","C:\\Users\\Achyut
Tripathi\\Desktop\\dictionary.txt");
}
void delette_tree(TrieNode* root, char * key)                //Deleting word an meaning from
trie tree
{
        int i, j, k;
        j = strlen(key);
        TrieNode* temp1 = root;
        for(i = 0; i < j; i++)
        {
                        k = char_to_index(key[i]);
```

```
                        if(!temp1->children[k])
                        {
                                cout<<"\n\nWord NOT Found !!!!";
                                break;
                        }
                        temp1 = temp1->children[k];
        }
        if(temp1 != NULL && temp1->isEOW)      //Traverses the trie tree and deletes the
word by changing the EOW of last letter of delete word to false
        {
                        temp1->isEOW = false;
        }
        else if(temp1==NULL)
        {
      free(temp1);
        }
}

void push(char * key)                   //Pushing Recent Searched Item in Stack
{

        struct history *newhistory;
        newhistory=(struct history*)malloc(sizeof(struct history));
        strcpy(newhistory->data,key);
        if(top==NULL)
        {
                        newhistory->next=NULL;
        }
        else
                        newhistory->next=top;
```

```
        top=newhistory;
}

void search(TrieNode* root)                    //Function to search a word in the file and
trie tree
{
   ifstream file;
        file.open("C:\\Users\\Achyut Tripathi\\Desktop\\dictionary.txt", ios::in);
        xy3:
        char key[50], ch,word[50],mean[50];
        int temp=0;
        cout<<"\n\nEnter Word to Search: ";
        gets(key);
        fflush(stdin);
        push(key);
        while(!file.eof())
        {
      file.getline(word,50);//Extracting each textline of the file
      if(strcmp(key,word)==0) //checking whether they match with the search word
      {
         temp=1; //So as to prevent it from searching the word in the trie tree as well
         break;

      }

        }
   file.getline(mean,50); //Extracting the meaning of the word from the file
   cout<<"\n\nMeaning: ";
   puts(mean);
   if(temp==0)                    //Search in the trie tree
```

```
        {

    int i, j, k;
        j = strlen(key);
        TrieNode* temp = root;
        for(i = 0; i < j; i++)
        {
                        k = char_to_index(key[i]);
                        if(!temp->children[k])
                        {
                                cout<<"\n\nWord NOT Found !!!!";
                                break;
                        }
                        temp = temp->children[k];
        }
        if(temp != NULL && temp->isEOW)
        {
                        cout<<"\n\nMeaning: ";
                        puts(temp->meaning);
        }
        }

        cout<<"\n\n\nDo you want to search more words?(y/n): ";
        cin >> ch;
        fflush(stdin);
        if(ch == 'y')
                        goto xy3;
    file.close();
}
```

```
void display_history()                    //Function to display the recent history
{
        if(top==NULL)
                        cout<<"\n\n\n\n\n\n\t\t\t\t\t\t        Stack history is empty!"<<"\n";
        else
        {
                        struct history *temp=top;
                        while(temp->next!=NULL)
                        {
                                cout<<"\n"<<temp->data<<"\n";
                                temp=temp->next;
                        }

        cout<<"\n"<<temp->data;
        }
}

void SetColor(int ForgC)                        //Function to set the font colour
 {
        WORD wColor;
        HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
        CONSOLE_SCREEN_BUFFER_INFO csbi;
        if(GetConsoleScreenBufferInfo(hStdOut, &csbi))
        {
                wColor = (csbi.wAttributes & 0xF0) + (ForgC & 0x0F);
                SetConsoleTextAttribute(hStdOut, wColor);
        }
}
```

```
void ClearConsoleToColors(int ForgC, int BackC)               //Function to clear the console and for colouring the console
{
	WORD wColor = ((BackC & 0x0F) << 4) + (ForgC & 0x0F);
	HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
	COORD coord = {0, 0};
	DWORD count;
	CONSOLE_SCREEN_BUFFER_INFO csbi;
	SetConsoleTextAttribute(hStdOut, wColor);
	if(GetConsoleScreenBufferInfo(hStdOut, &csbi))
	{
		FillConsoleOutputCharacter(hStdOut, (TCHAR) 32, csbi.dwSize.X * csbi.dwSize.Y, coord, &count);
		FillConsoleOutputAttribute(hStdOut, csbi.wAttributes, csbi.dwSize.X * csbi.dwSize.Y, coord, &count );
		SetConsoleCursorPosition(hStdOut, coord);
	}
	return;
}

void dec()                          //Display word DICTIONARY in the top
{
	ClearConsoleToColors(0,11);
	SetColor(0);
	printf("\n\t\t\t ****  *****  **** ***** *****  ****  **       *      ***   **** **");
	printf("\n\t\t\t *  *  *      *    * *      *  * ** *  *  *   * * *  *");
	printf("\n\t\t\t *  * * *    *    * *   * * * *  *******  ****    ***");
	printf("\n\t\t\t *  *  *      *    * *  *  * *  * * *    * *  * *");
```

```
	printf("\n\t\t\t  ****  *****  ****  *  *****  ****  *   ** *    * *     *  *");
}

int main()
{
	char ch,st[50];
	TrieNode* root = newnode();
	xy1:
	dec();
	SetColor(0);
	printf("\n\n\n\n\n\n\t\t\t\t\t\t1. Add a Word in the Dictionary\n\n\t\t\t\t\t\t2. Search a Word\n\n\t\t\t\t\t\t3. Delete a word from Dictionary\n\n\t\t\t\t\t\t4. Display Search History\n\n\t\t\t\t\t\t5. Exit\n\n\nYour Choice: ");
	scanf("%c", &ch);
	fflush(stdin);
	switch(ch)
	{
			case '1':
				dec();
				printf("\n\n\n\t\t\t\t\t\t--------Insert Word--------");
				insert(root);
				break;
			case '2':
				dec();
				printf("\n\n\n\t\t\t\t\t\t--------SEARCH WORD--------");
				search(root);
				break;
			case '3':
				dec();
				printf("\n\n\n\t\t\t\t\t\t--------DELETE WORD--------");
```

```
                cout<<"\n\n\nEnter a Word to delete: ";
                gets(st);
                fflush(stdin);
                delette_tree(root, st);
                cout<<"\n\nPress Enter for Delete from File";
                getchar();
                delette_file(root, st);
                break;
        case '4':
                dec();
                printf("\n\n\n\t\t\t\t\t\t-------- SEARCH HISTORY-------- ");
                display_history();
                break;
        case '5':
                exit(1);

    }
    printf("\n\n\nDo you want to do more operations ? (y/n): ");
    scanf("%c", &ch);
    fflush(stdin);
    if(ch == 'y')
                goto xy1;
    return 0;
}
```

References

https://www.geeksforgeeks.org/trie-insert-and-search/

https://www.geeksforgeeks.org/trie-delete/

https://www.youtube.com/watch?v=AXjmTQ8LEoI

https://www.tutorialspoint.com/cplusplus/cpp_files_streams.htm

http://www.mycplus.com/tutorials/c-programming-tutorials/file-handling/

https://stackoverflow.com/questions/5496192/deleting-a-string-from-a-particular-position-in-a-file

http://www.cplusplus.com/forum/beginner/220864/

https://stackoverflow.com/questions/28212863/delete-certain-content-from-txt-file-using-c

https://www.daniweb.com/programming/software-development/threads/161677/c-find-word-from-text-file

https://codereview.stackexchange.com/questions/52702/how-to-search-for-a-word-in-a-sorted-text-file-efficiently