MINI PROJECT

Problem Statement:

The project implements a modern day dictionary using an AVL Tree data structure.

Data Structure Used:

AVL Tree.

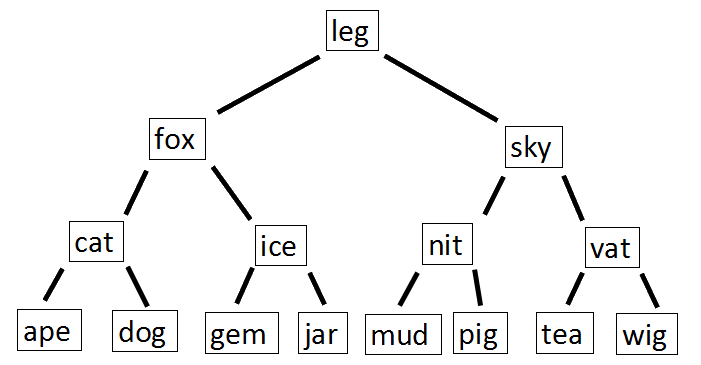
Implementation:

Each node contains a structure, containing the English word, and its meaning, along with the pointer to the left and right subtrees.

Traversal is done Inorder, to match the word with its respective meaning.

Once the required word has been found, and matched, it is printed along with its meaning.

Example of the Tree



Modules:

* **struct avlnode**

char english[MAX],mean[MAX];

avltree left;

avltree right;

int height;

* **typedef struct avlnode\* position**
* **typedef struct avlnode\* avltree**
* **int max(int a,int b)**

finds and returns the max element.

* **static int height(position p)**

returns the height of the tree

* **position singlerotatewithleft(position k2)**

Single rotates to the left.

* **position singlerotatewithright(position k2)**

Single rotates to the right.

* **position doublerotatewithleft(position k3)**

Double rotates to the left.

* **position doublerotatewithright(position k3)**

Double rotates to the right.

* **avltree insert(avltree t, char english[], char mean[])**

Inserts node into the AVL Tree.

* **int getBalance(avltree N)**

Retuns the balance value of the node.

If this value = -2 or 2 the necessary rotations are performed to balence the tree.

* **avltree minValueNode( avltree node)**

Retunrs a pointer to the node with the minimum value in its subtree.

* **avltree deleteNode(avltree root, char key[])**

Deletes a node.

* **position find(avltree t,char word[])**

Searches for a word, and returns it if found.

* **void inorder(avltree t,char op)**

Inorder traversal of the AVL tree.

* **int findMeaning(avltree tree, char s[])**

Finds the meaning of the word, returns -1 if the word is not in the dictionary.

* **avltree add(avltree t, char word[])**

Adds a new word, along with the meaning into the AVL tree.

* **avltree search(avltree t)**

Searches for a word in the dictionary.

* **void dispall(avltree t)**

Displays the whole dictionary.

* **void display(avltree t)**

Displays the dictionary for a particular alphabet that the user wants.

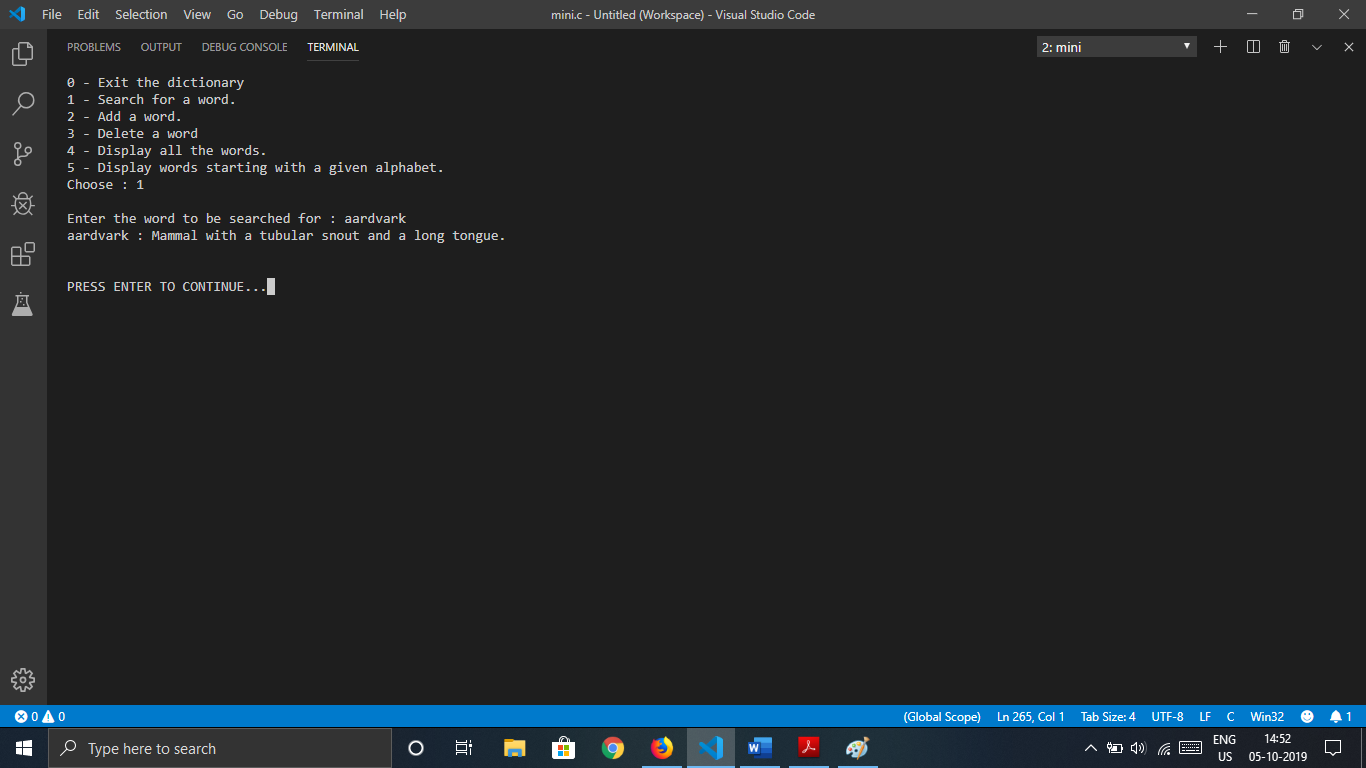
* **avltree createdict(avltree tree)**

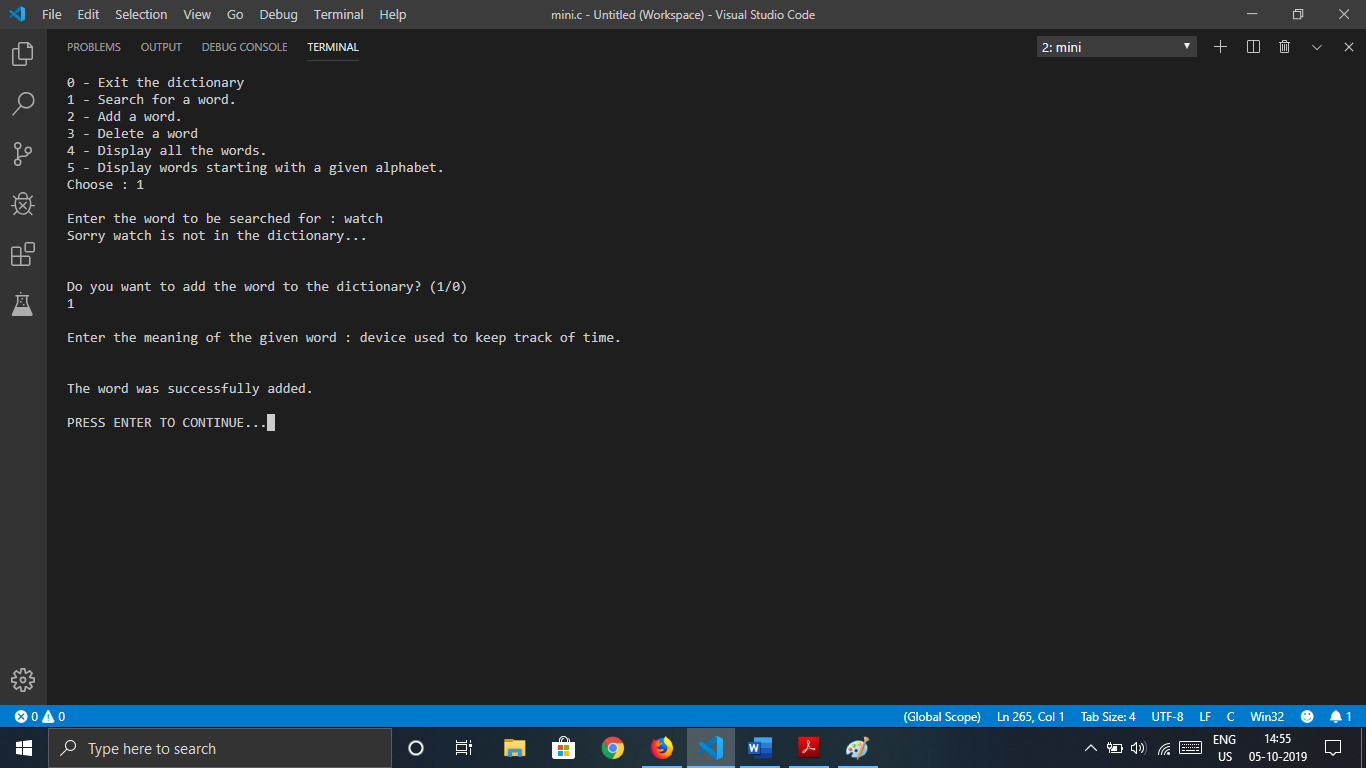
Creates the dictionary.

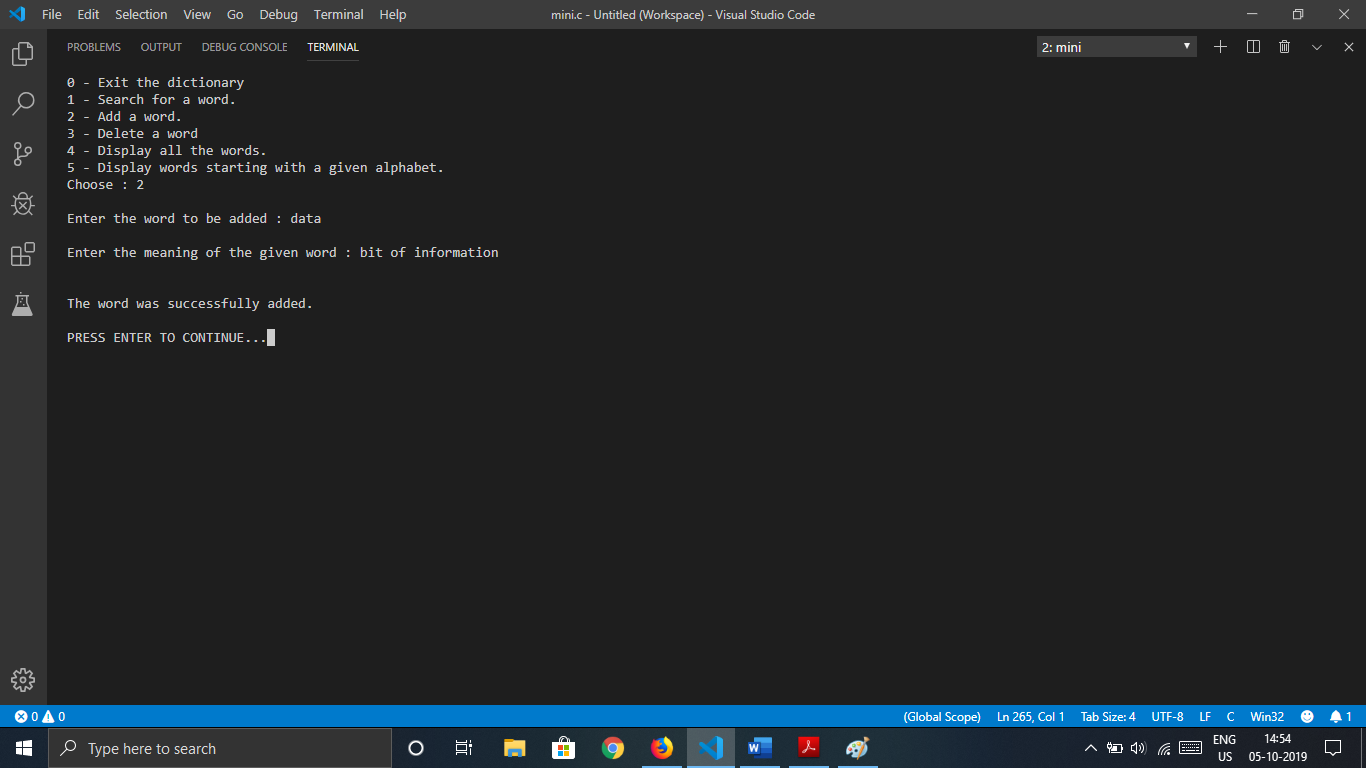
* **void writeout(avltree t,FILE \*fp)**

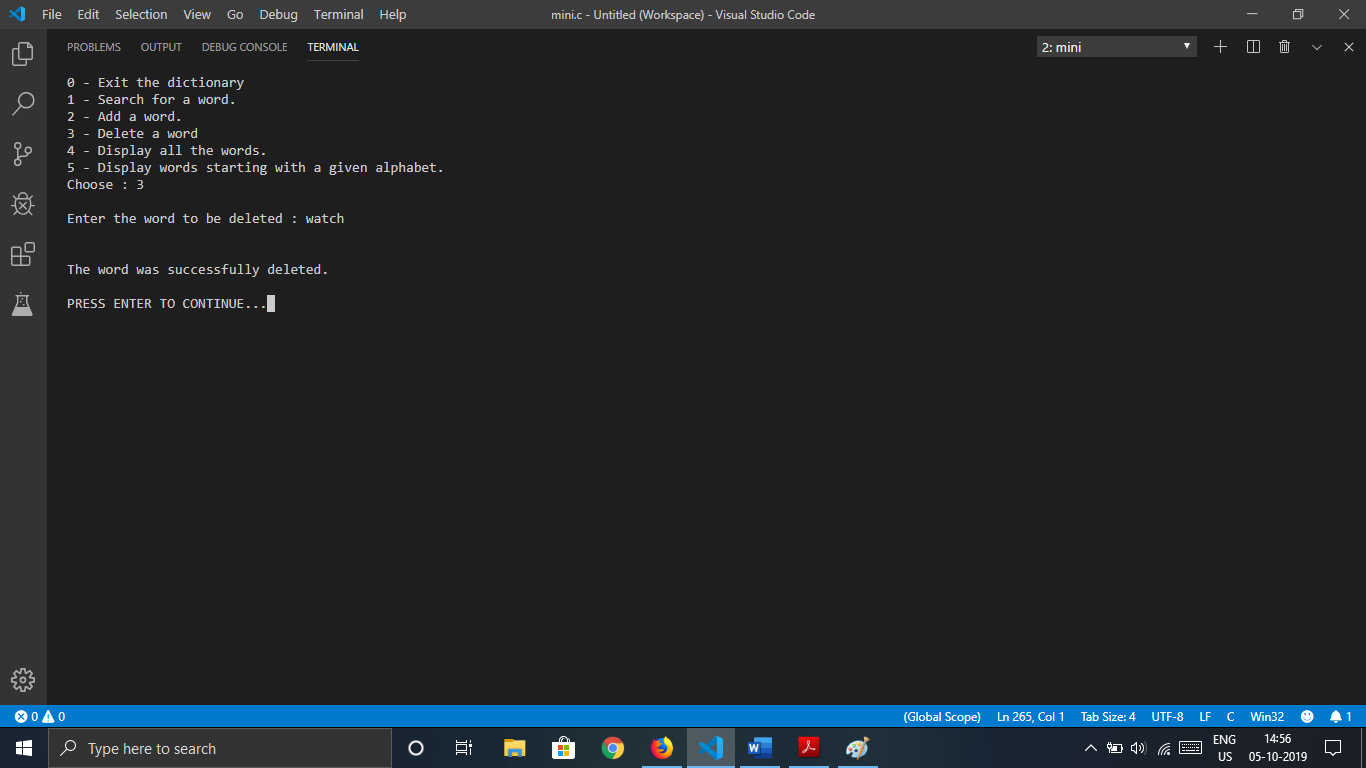
Writes new words and meanings into the file.

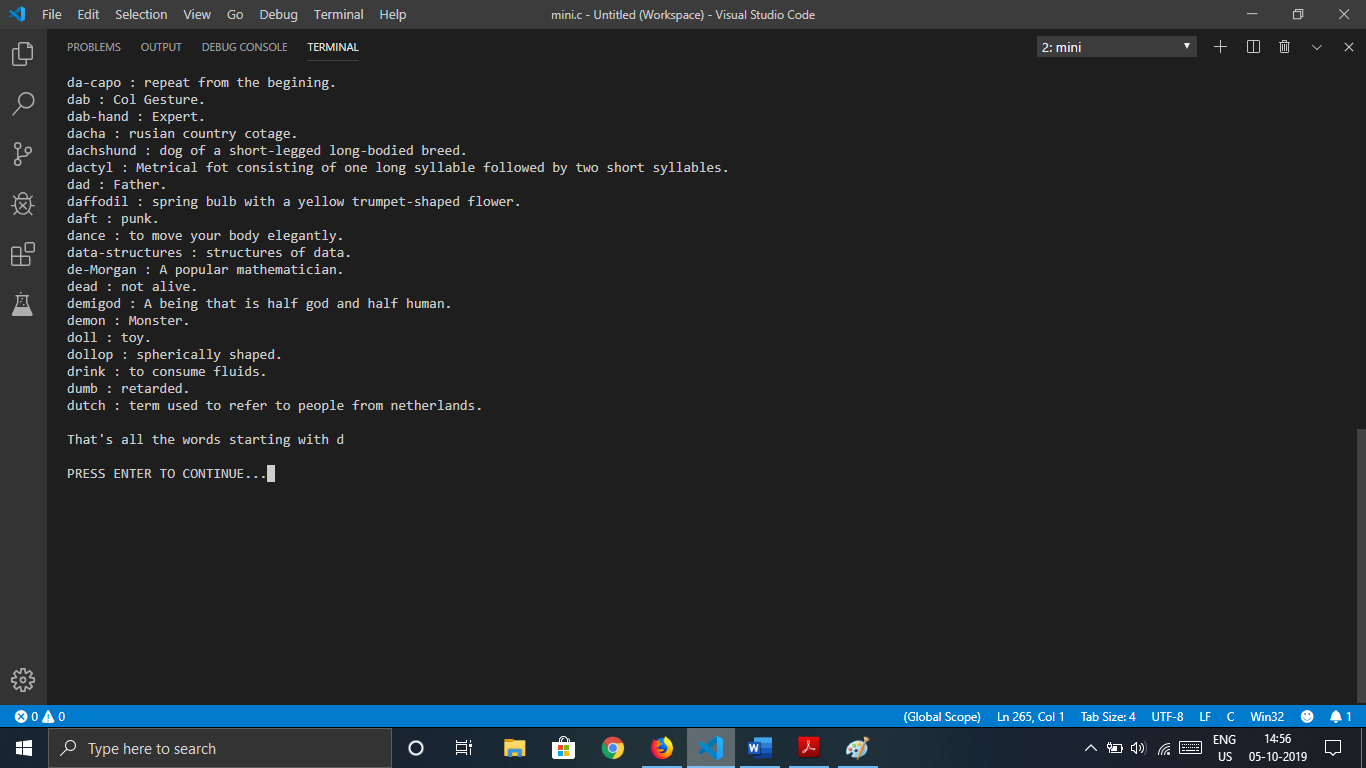
**Sample I/O:**

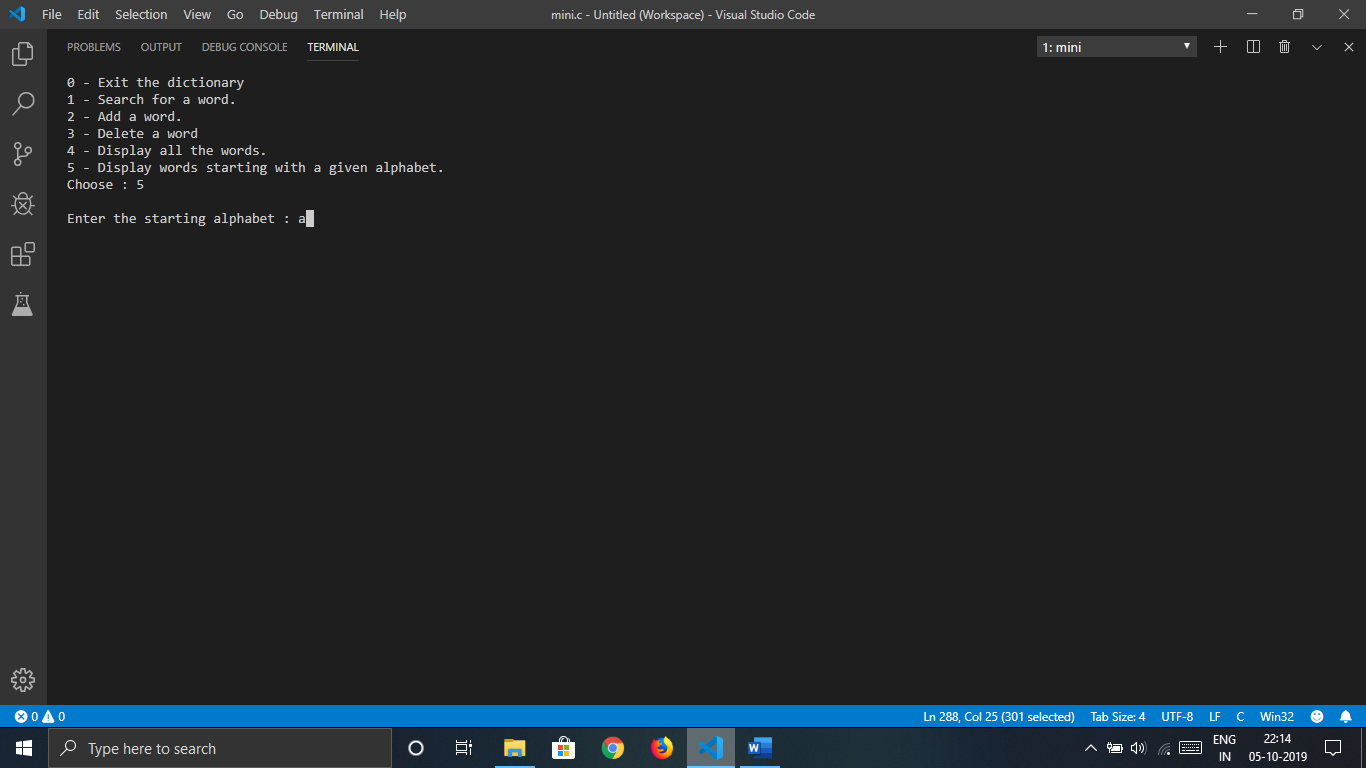


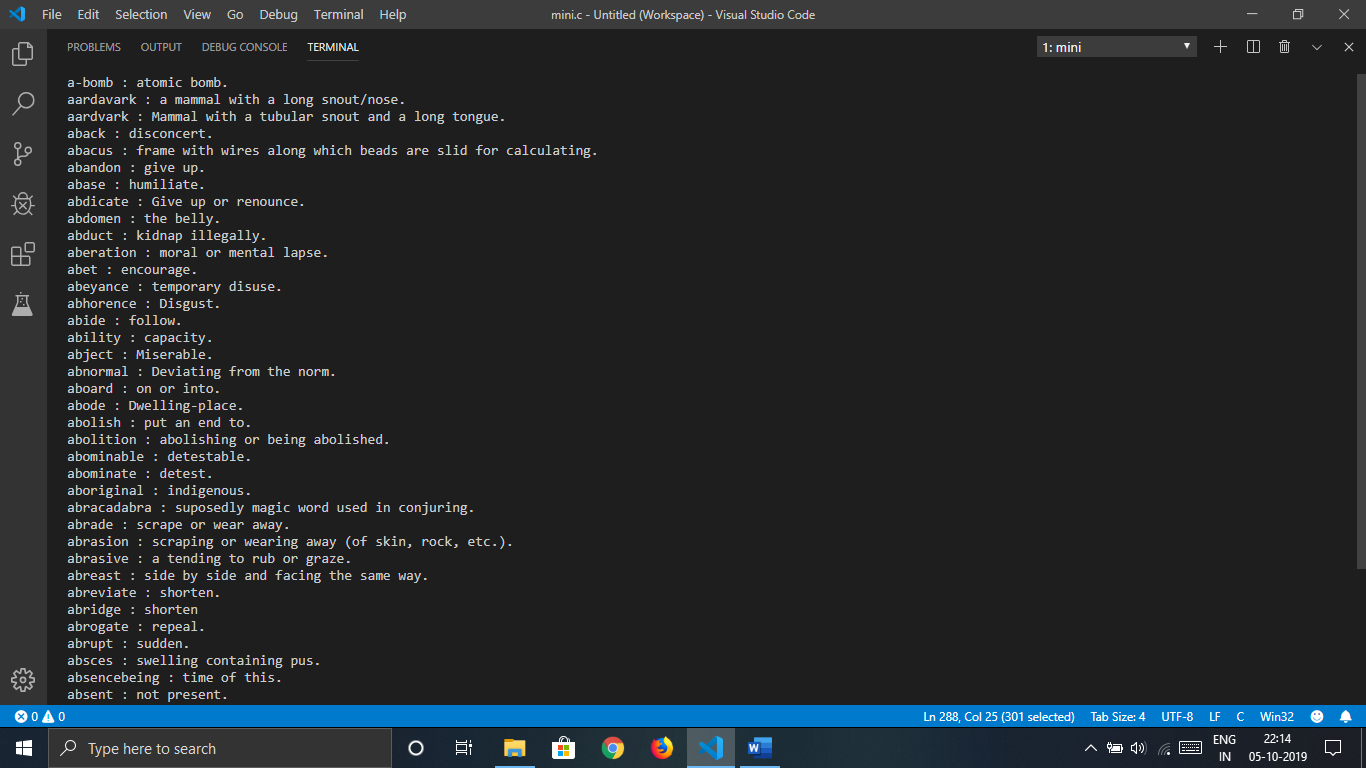


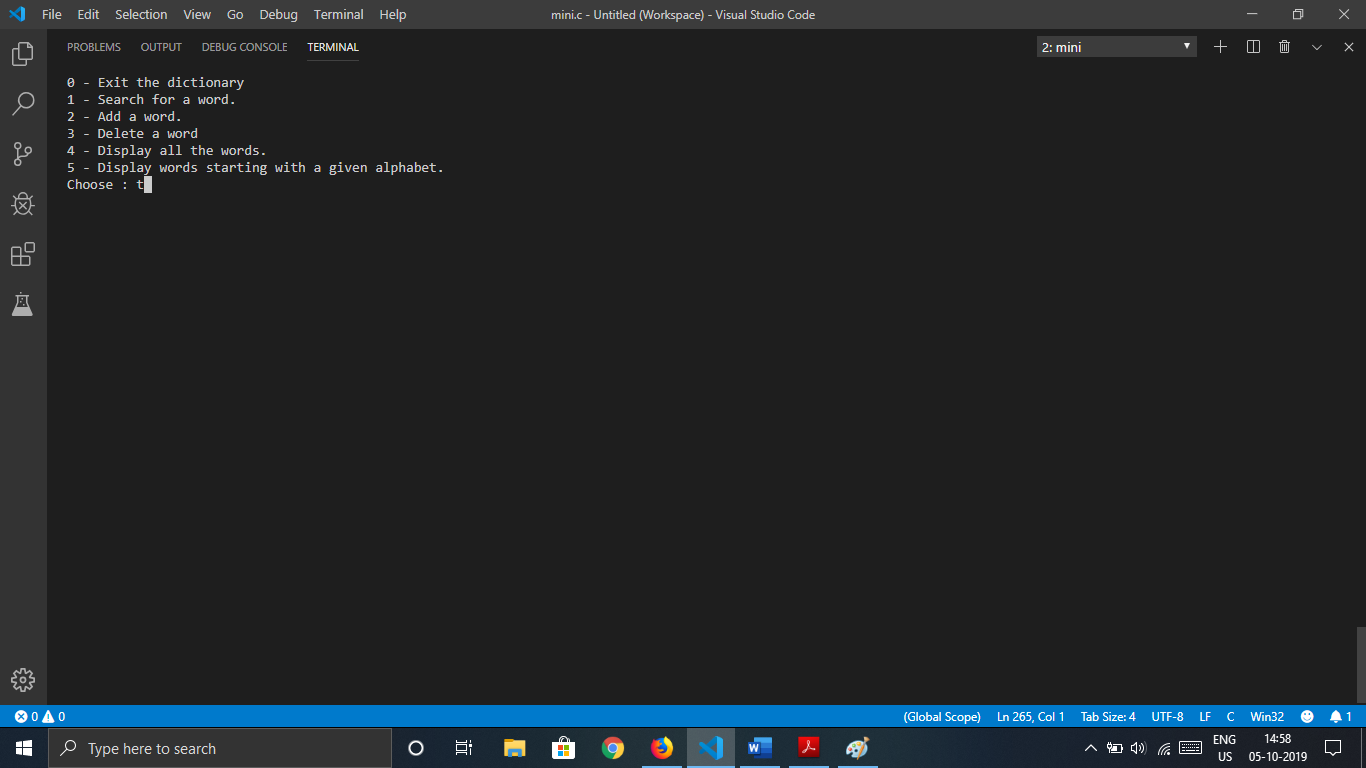


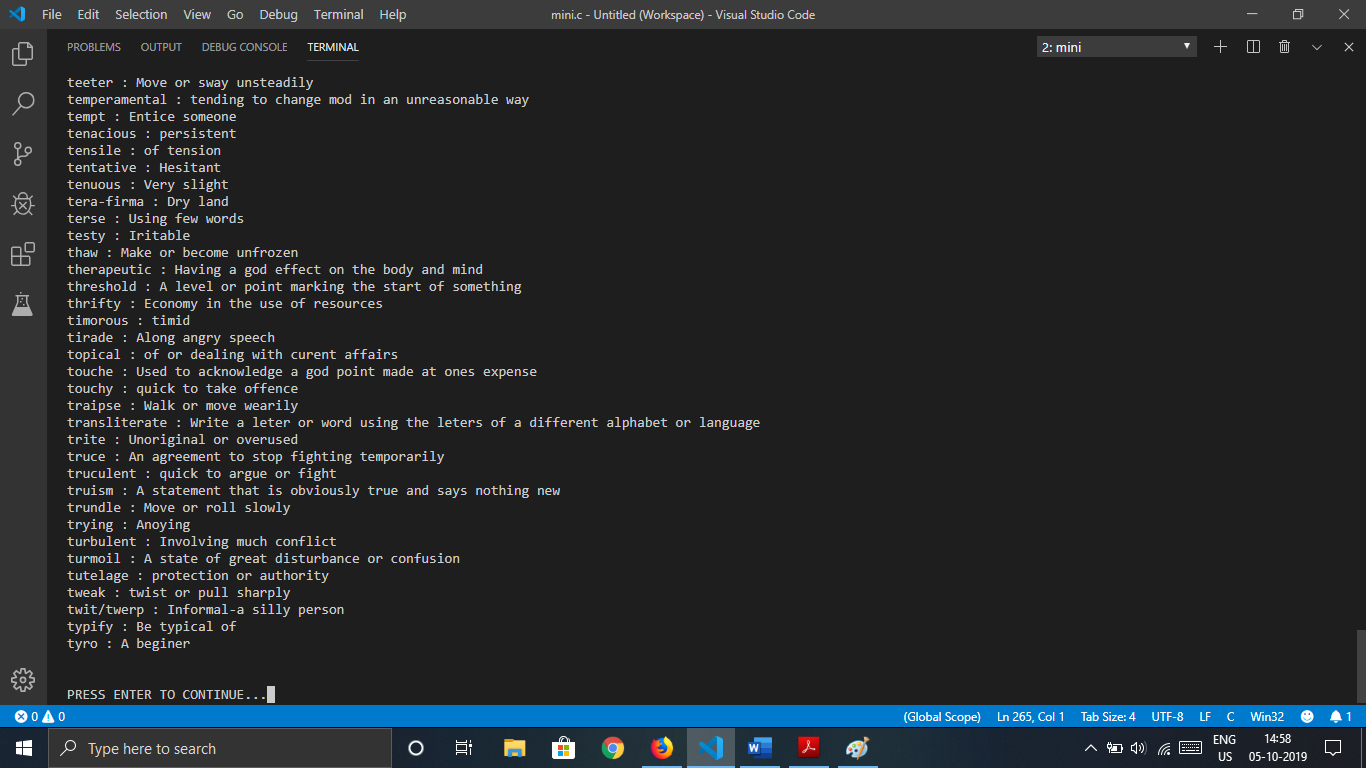


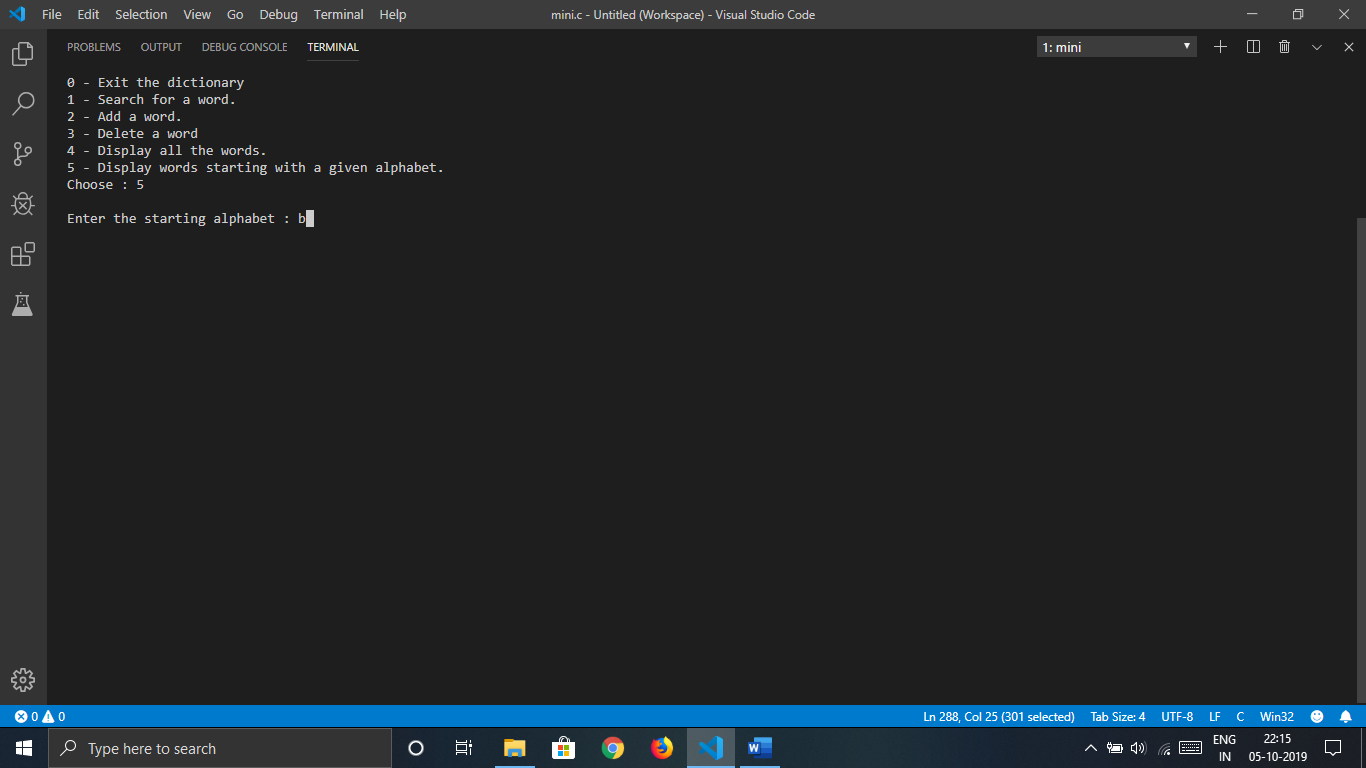


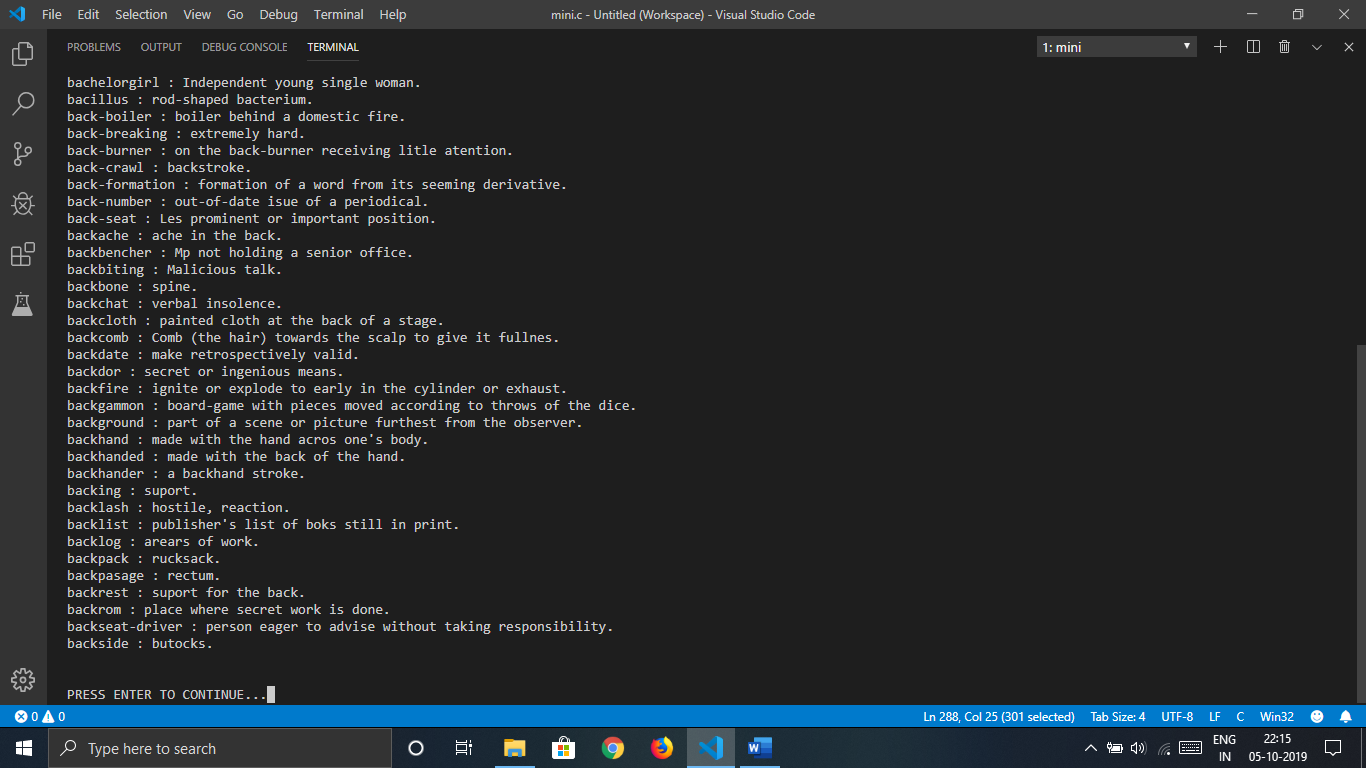












Learning Outcomes :

1. We have learnt the use of modular programming and how it can be helpful when constructing large programs.
2. We have learnt the real time use of an AVL tree.
3. We have learnt that the time complexity for the search operation on an AVL is O(Log(n)).
4. We have learnt to implement a dictionry using an AVL tree.
5. We have learnt how to read/write into a file in C.