DATA STRUCTURES AND ALGORITHMS

-PROJECT

AUTO PREDICTIVE DICTIONARY

Problem Statement:

Create an auto predictive dictionary application in C++ having advanced features such as adding new words, deleting and editing existing words only by the administrator password. They should be able to see the list of changes made in the dictionary.

The dictionary should be accessed only by the users registered in the application

Data Structures and Algorithms Used:

**Trie (digital tree):**

It is an ordered tree data structure that is used to store a dynamic set or associative array where the keys are usually strings.

**Binary Search Tree:**

Theyare a particular type of containers: data structures that store "items" (such as numbers, names etc.) in memory. They allow fast lookup, addition and removal of item.

**Boyer- Moore Horspool Algorithm:**

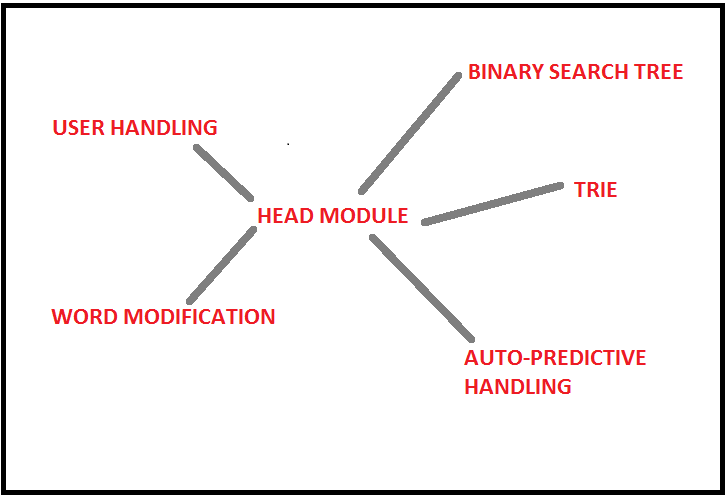
In computer science, the Boyer–Moore string search algorithm is an efficient string searching algorithm that is the standard benchmark for practical string search literature. It is simplification to Boyer Moore Algorithm .The algorithm trades space for time in order to obtain an average-case complexity of O(N) on random text, although it has O(MN) in the worst case, where the length of the pattern is M and the length of the search string is N.

**Introduction to Application:**

The application “Auto Predictive Dictionary” is made for the modern users. This dictionary contains around 5000 words which are generally needed to be looked in a dictionary rather than the simple words. Additionally users can add new words, delete and edit existing words in the dictionary if they know the administrative password. The application also shows the history of all words searched by all the users.

This application used the advanced algorithms to make possible the processing of the large amount of data very fast and runs smoothly.

**MODULE DESIGN**

****

**MODULE DESCRIPTION:**

**User handling:** This module deals with the user processing such as registration, login and logout. Also keep the count of users.

**Auto Prediction and error correction:** It comes into picture whenever a given word is not founded in the dictionary. It suggests user the similar words according to the word given as input.

It can also be used to predict a particular word when the user only knows the prefix or subpart of a word.

**Trie**: It stores all the existing words and the words given as input by the users in the form of a digital tree. New words can be added and any word can be searched.

**Binary Search Tree**: It stores all the words with their meaning in a binary search tree. Words can be searched quickly with their meanings. It automatically destroys itself at the end of the program.

**Word Modification**: This module helps in the modification of the dictionary. Upon administrative access new words can be added and existing words can be edited and deleted.

**Head Module**: It controls all other modules and makes them function accordingly. It is also responsible for the presentation of user interface.

**Implementation Details with Sample Code:**

**User Handling**:

**class user**

**{**

**char name[20];**

**char password[20];**

**int id;**

**int state;**

**int ti;**

**public:**

**int getreg();**

**void reg();**

**int getstate();**

**void getname()**

**void setid(int x);**

**int getid();**

**int login();**

**void logout();**

**};**

**Trie :**

**trie\_node \*create\_node()**

It helps in the creation of a trie node and assigns the node values to NULL.

**void insert\_node(char word[])**

//It helps in the insertion of a new word in the nodes.

{

int length=strlen(word);

int index;

trie\_node \*a;

count++;

a=trie\_root;

for(int i=0 ; i<length ; ++i)

{

index = INDEX(word[i]);

if(!a->children[index])

{

a->children[index]=create\_node();

}

a=a->children[index];

}

a->value=count;

}

**void copystr(char b[], char a[])**

It inputs the given word and helps to convert it in the format which can be used for the trie.

**int search\_node(char word[])**

This function accepts a word and finds it in the given trie. If the word is found it returns 1 else returns 0.

**Binary Search Tree:**

**void make\_tree(string x,string y)**

This function takes input as two strings i.e a word and its meaning and makes a binary search tree according to the words with the dynamically allocated nodes.

**void show\_word(node \*n, string x)**

This function takes root node as input and a word to be searched in the binary search tree.

**node\* delete\_tree(node\* r)**

This function destroys the binary search tree upon the termination of the program.

**Word modification :**

**void add\_new\_words()**

This function adds new words after accepting as input by the user and adds it to the dictionary.

**void edit\_word()**

This function edits a given word present in the dictionary. And displays error message if the word is not found in the dictionary.

**void delete\_word()**

**void delete\_word()**

**{**

**int x=0,i;**

**char word[20],s[200],w[20];**

**cout<<"\nEnter the word you want to delete\n";**

**cin>>word;**

**ofstream fo;**

**fo.open("adv.txt",ios::out | ios::app);**

**fo<<"DELETED: "<<word<<endl;**

**fo.close();**

**ifstream fin;**

**fin.open("OfficialDictionary.txt",ios::in);**

**ofstream fout;**

**fout.open("newfile2.txt",ios::out);**

**while(!fin.eof())**

**{**

**fin.getline(s,200);**

**for(i=0;s[i]!=' ';++i)**

**{**

**w[i]=s[i];**

**}**

**while(i<20)**

**{**

**w[i]='\0';**

**i++;**

**}**

**if(!strcmp(word,w))**

**x=1;**

**if(strcmp(word,w))**

**{**

**fout<<s<<endl;**

**}**

**}**

**fin.close();**

**fout.close();**

**if(x==1)**

**cout<<"The word successfully deleted";**

**else**

**cout<<"The word was not found in the dictionary";**

**getch();**

**}**

This function deletes an existing word in the dictionary. And displays error message if the word is not found in the dictionary.

**Auto prediction and correction**

**void fail\_handle(char word[])//This helps to find the**

**//suggesting words when it is not found in the dictionary**

**{**

**trie\_node \*tp;**

**tp=trie\_root;**

**int k=0,i;**

**for(i=0; i<strlen(word); ++i)**

**{**

**k=int(word[i])-97;**

**if(tp->children[k]==NULL)**

**return ;**

**else**

**tp=tp->children[k];**

**}**

**fail\_search(tp,word);**

**return ;**

**}**

**void fail\_control(char p[])**

**int fail\_search(trie\_node \*x,char word[])**

**//This is Boyer Moore Horspool Algorithm**

**int max(int a, int b)**

**{**

**return (a > b) ? a : b;**

**}**

**void badCharHeuristic(char \*str, int size, int badchar[NO\_OF\_CHARS])**

**{**

**int i;**

**for (i = 0; i < NO\_OF\_CHARS; i++)**

**badchar[i] = -1;**

**for (i = 0; i < size; i++)**

**badchar[(int) str[i]] = i;**

**}**

**void searchw(char \*txt, char \*pat)**

**{**

**int m = strlen(pat);**

**int n = strlen(txt);**

**int badchar[NO\_OF\_CHARS];**

**badCharHeuristic(pat, m, badchar);**

**int s = 0;**

**while (s <= (n - m))**

**{**

**int j = m - 1;**

**while (j >= 0 && pat[j] == txt[s + j])**

**j--;**

**if (j < 0)**

**{**

**cout<<txt<<endl;**

**s += (s + m < n) ? m - badchar[txt[s + m]] : 1;**

**}**

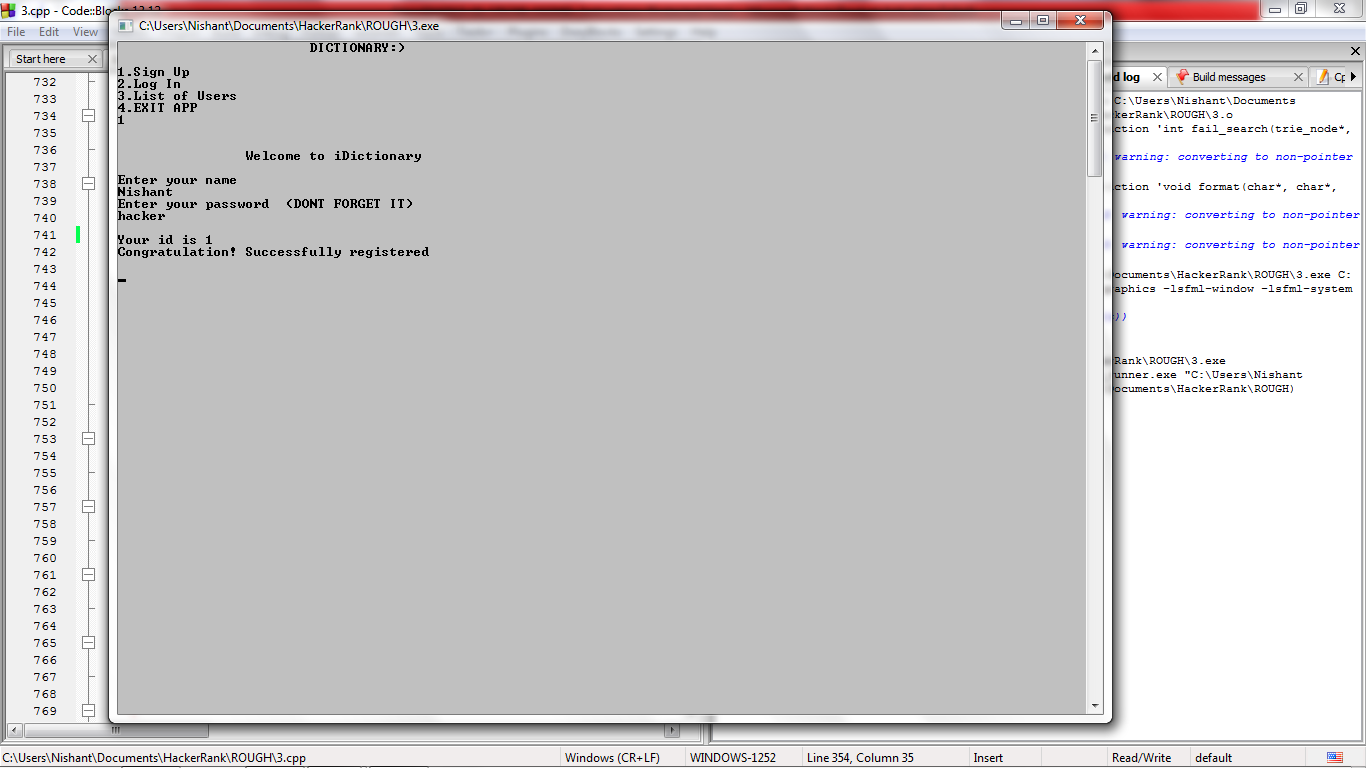
**else**

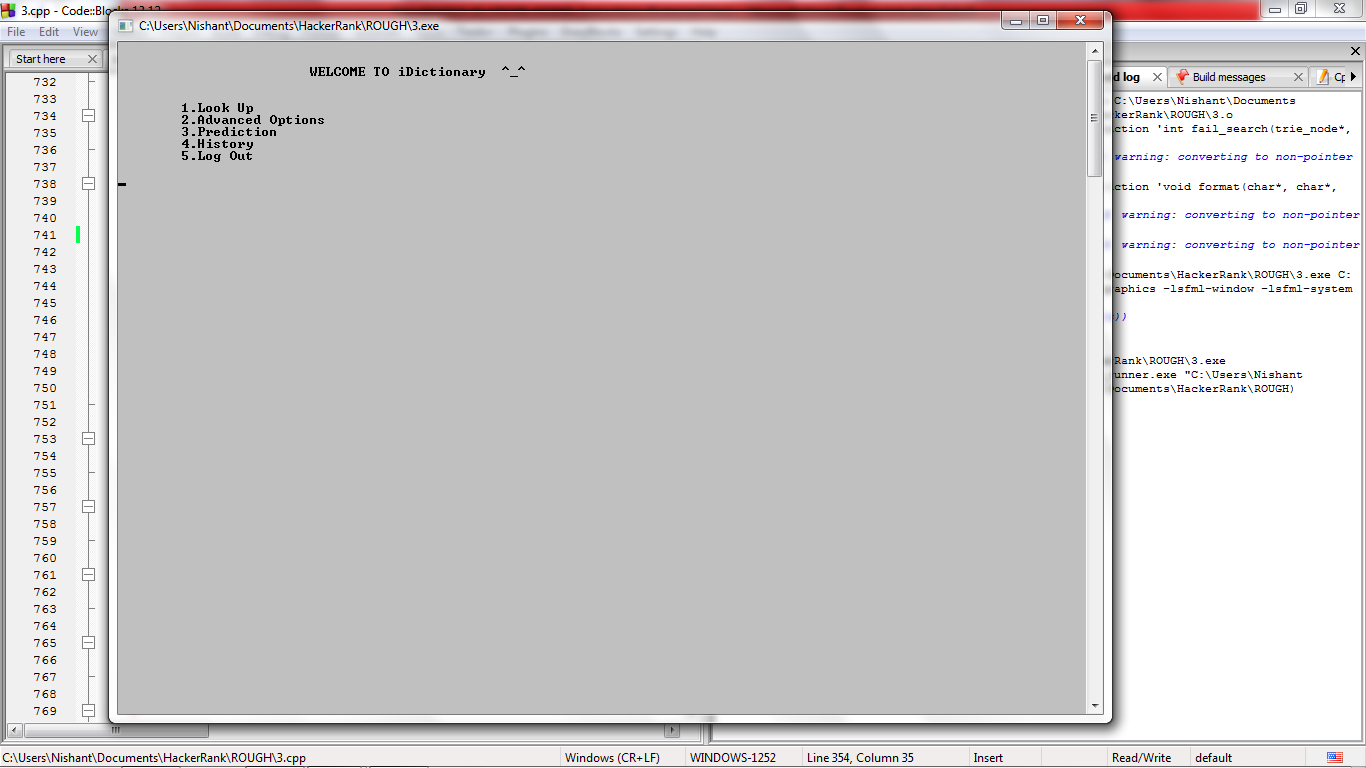
**s += max(1, j - badchar[txt[s + j]]);**

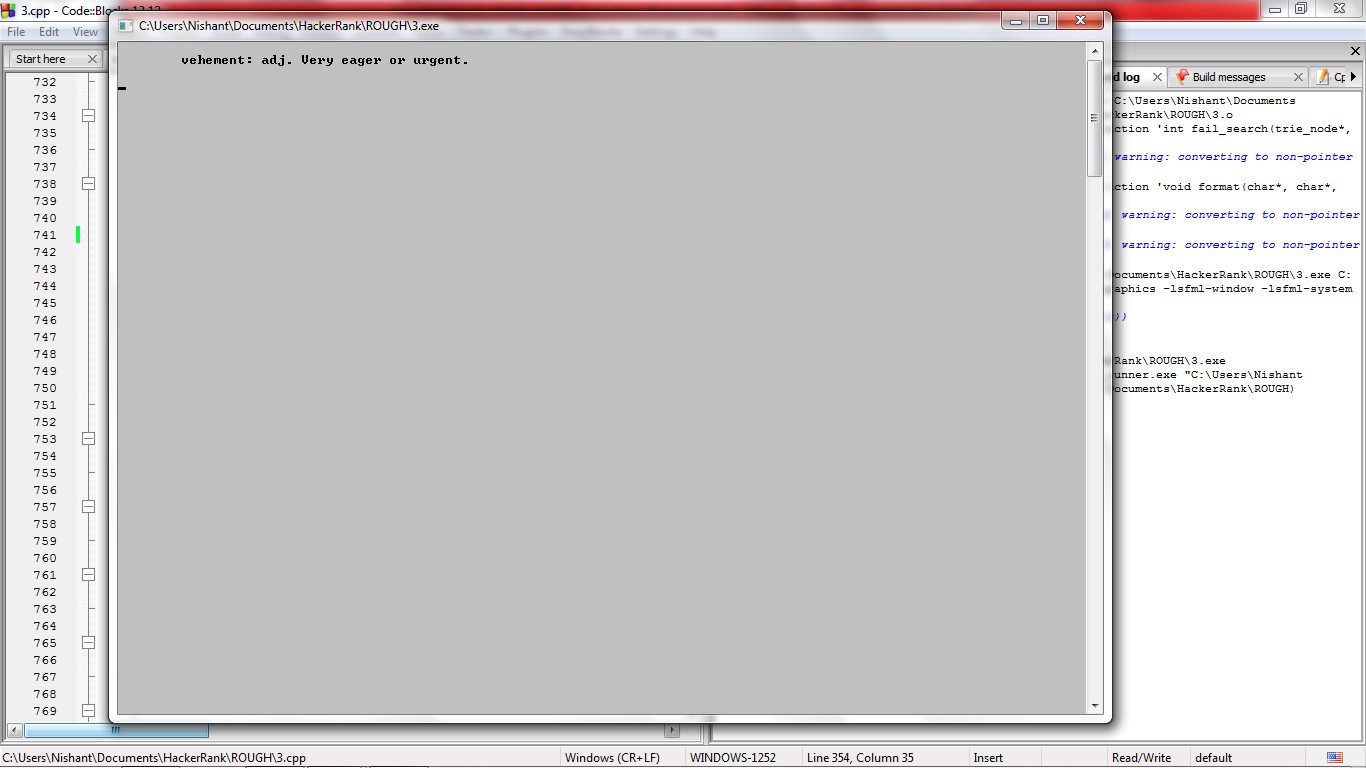
**}**

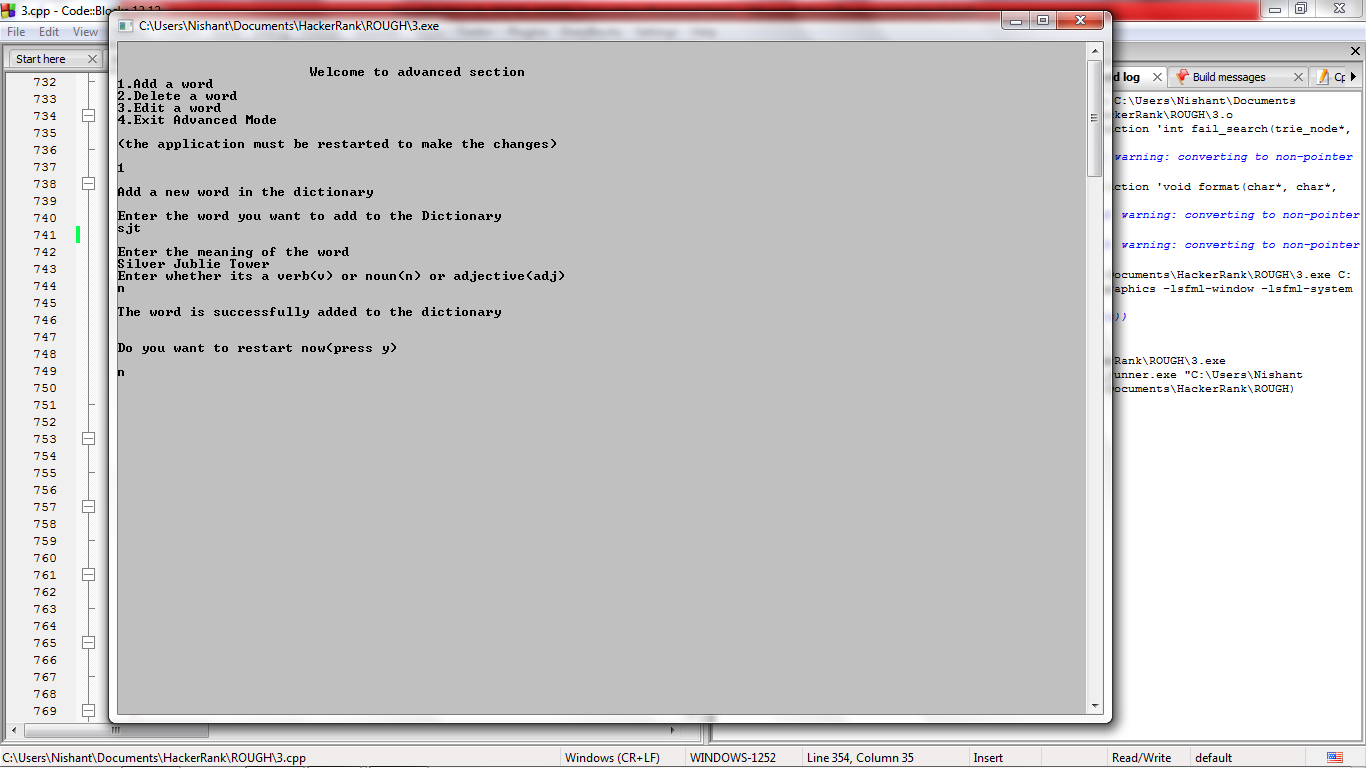
**}**

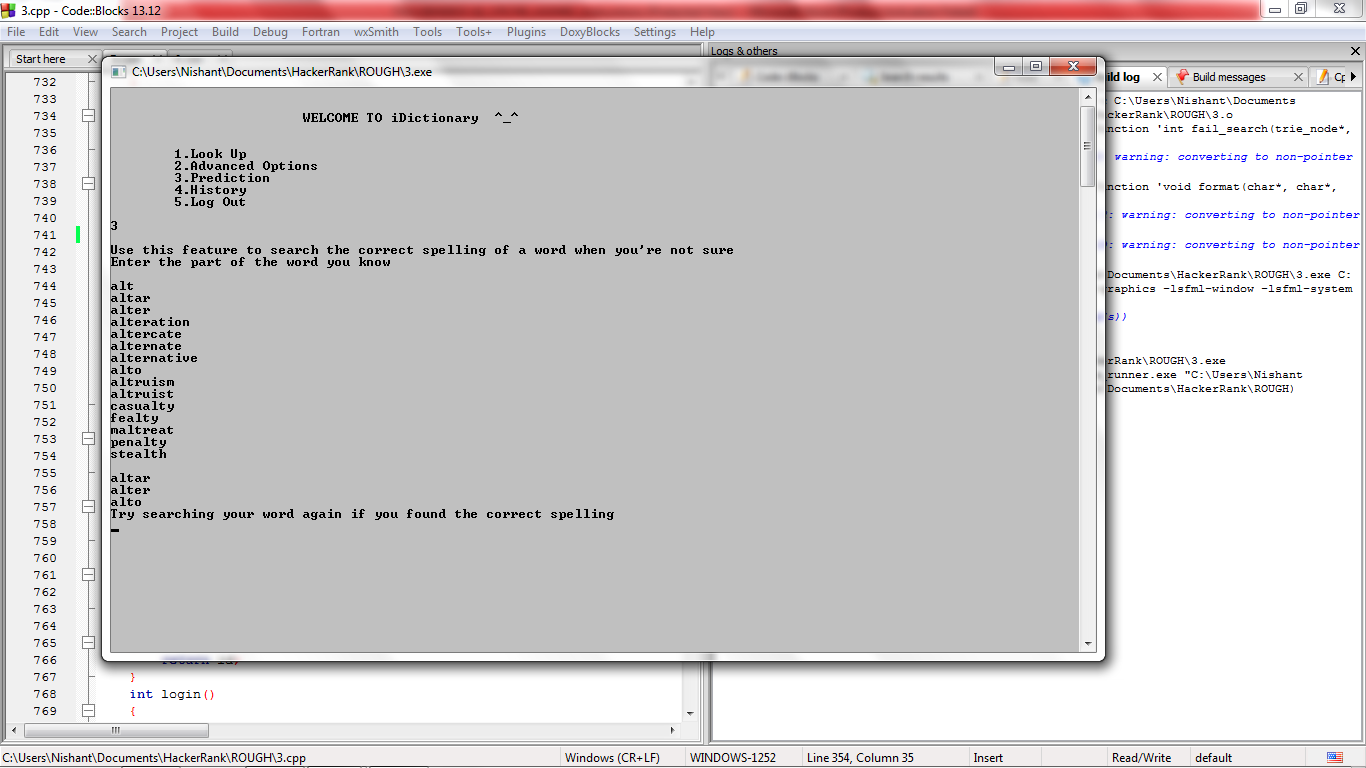
Screen Shots

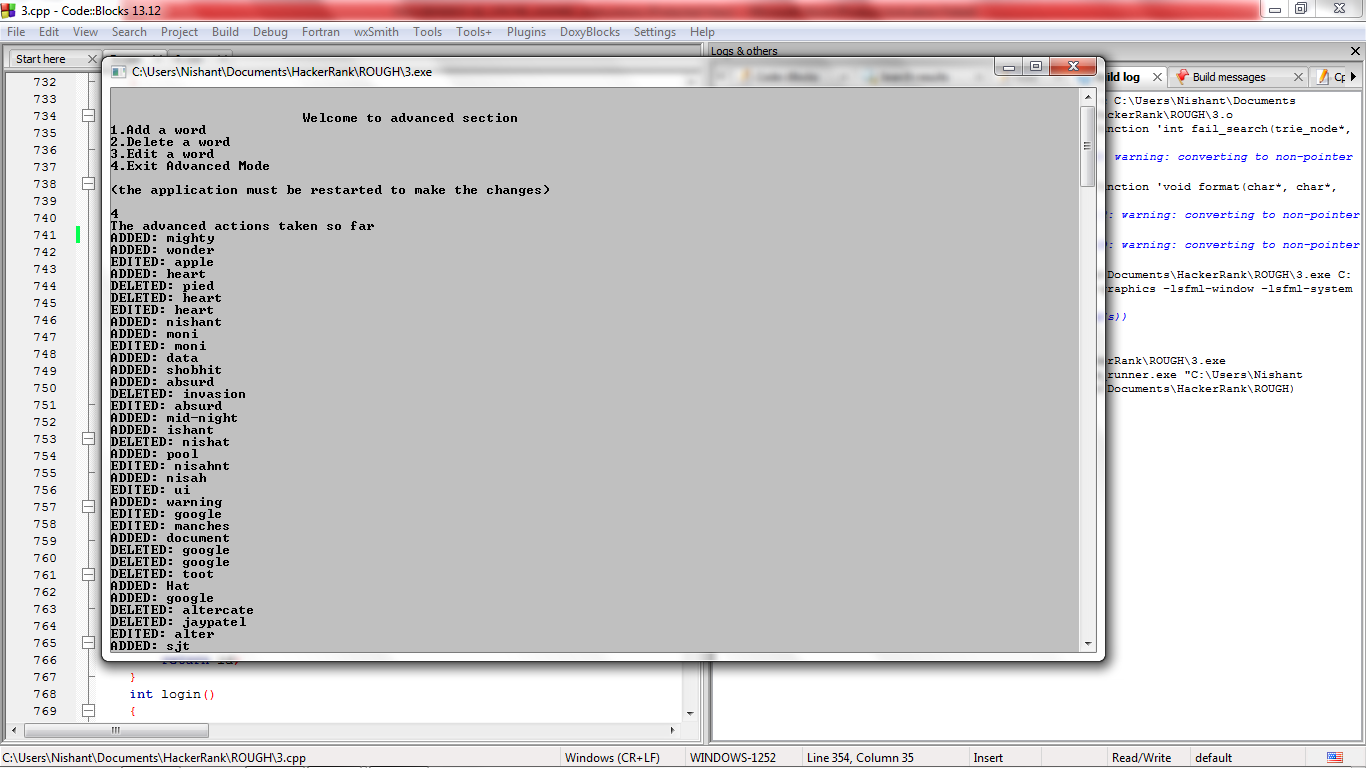


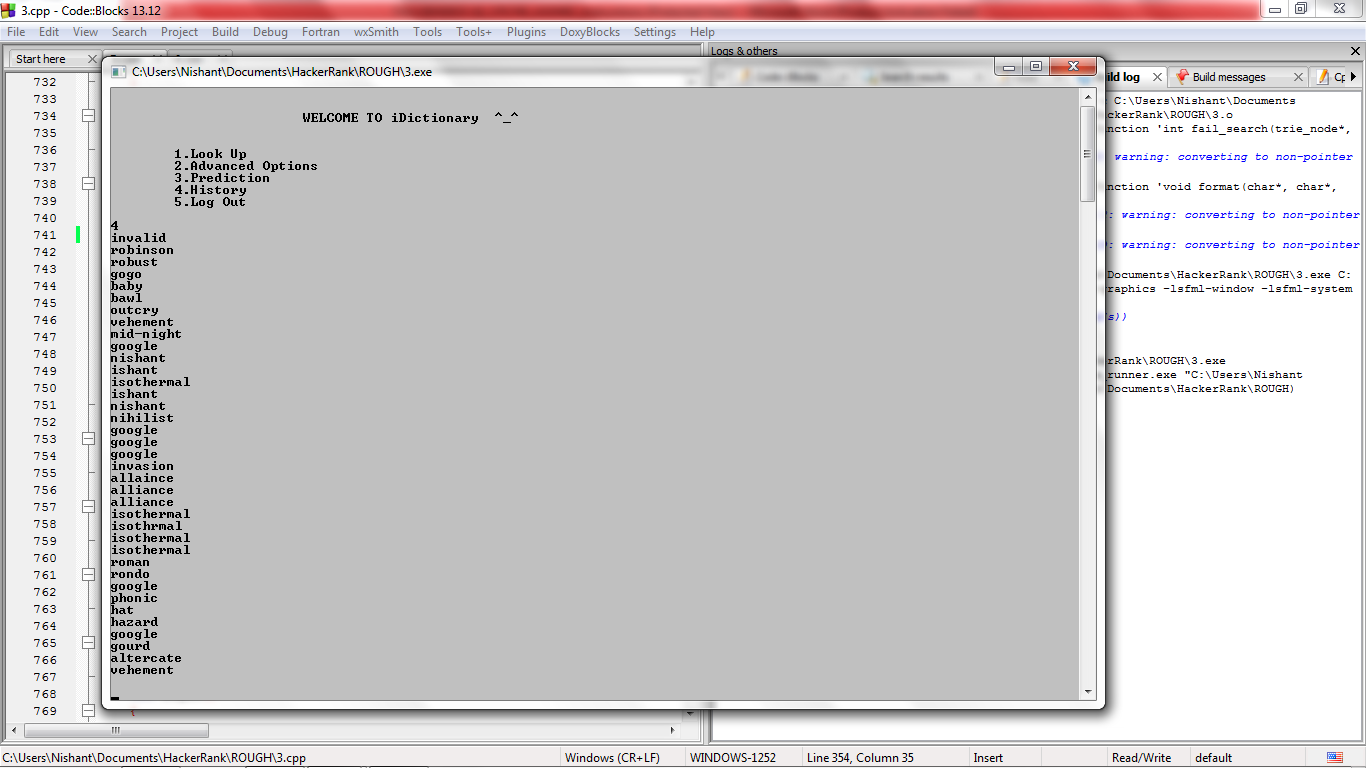












**CONCLUSION**

It was a wonderful learning experience for our team. Throughout the making of this application we explored the various phase of project development and it gave us real insight into the world of software engineering.

The various problems that we tackled and rectified gave us joy and the feel of developer’s industry.

As of now our application “Auto-predictive dictionary” is successfully built and all the features are working as planned. All the modules are tested many times to make sure all the bugs has been removed.

**REFERENCES**

* *[Cormen, Thomas H.](https://en.wikipedia.org/wiki/Thomas_H._Cormen" \o "Thomas H. Cormen)*;*[Leiserson, Charles E.](https://en.wikipedia.org/wiki/Charles_E._Leiserson" \o "Charles E. Leiserson)*,*[Rivest, Ronald L.](https://en.wikipedia.org/wiki/Ron_Rivest" \o "Ron Rivest)*,[*Stein, Clifford*](https://en.wikipedia.org/wiki/Clifford_Stein)(2009) [1990].[*Introduction to Algorithms*](https://en.wikipedia.org/wiki/Introduction_to_Algorithms)(3rd ed.). MIT Press and McGraw-Hill. p. 287.[*ISBN*](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [*0-262-03384-4*](https://en.wikipedia.org/wiki/Special:BookSources/0-262-03384-4).
* [*Knuth, Donald*](https://en.wikipedia.org/wiki/Donald_Knuth)(1997). "6.3: Digital Searching".The Art of Computer Programming Volume 3: Sorting and Searching(2nd ed.). Addison-Wesley. p. 492.[*ISBN*](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [*0-201-89685-0*](https://en.wikipedia.org/wiki/Special:BookSources/0-201-89685-0).
* [www.geeksforgeeks.org/data-structures/#BinarySearchTree](http://www.geeksforgeeks.org/data-structures/#BinarySearchTree)
* www.stackoverflow.com/questions/....
* <https://en.wikipedia.org/wiki/Boyer–Moore–Horspool_algorithm>