A TWO WAY TRANSLATOR

BY

RAM GUNASEKARAN A 18BCE1234
SIDDHARTH M NAIR 18BCE1238
VARSHA R 18BCE1344

A PROJECT REPORT SUBMITTED TO

RAJALAKSHMI R

Vellore Institute of Technology, Chennai
Vandalur – Kelambakkam Road Chennai – 600127

NOVEMBER 2019

CHENNAI

BONAFIDE CERTIFICATE

Certified that this project report entitled "**TWO WAY TRANSLATOR"** is a bonafide work of **RAM GUNASEKARAN (18BCE)
,SIDDHARTH M NAIR(18BCE1238) AND VARSHA R(18BCE1344)** who carried out the project work under my supervision and guidance.

Guide Signature

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide,RAJALAKSHMI R , for her consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

RAM GUNASEKARAN

SIDDHARTH M NAIR

VARSHA R

Table of Contents

PROBLEM STATEMENT

Machine translation has been in the process of development since the early 1940's. Our project, aims to simplify and accelerate the process of translation using data structures like stacks, trie and binary trees, coded using C language.

Our project stores English and French words and translates a given word to the opposite language. Auto-complete of stored words has been implemented to accelerate the process, thus making it more efficient. A history tab has been implemented which stores the recently searched words. It displays them in the order of most-recently used to least-recently used. A 'Word of the Day' functionality has been implemented so that the user can learn a new English word along with it's French translation everyday.

ABSTRACT

Our projects goal is to develop a set of tools for translating texts between multiple languages in real time with high quality.

Languages are in separate modules in the tool and can be varied using different prototypes

The principle of translating is simple: a system decides an appropriate translation of an input word by analysing the pre-translated word in the database. Therefore the input English sentence is first checked with database and if it exists , it is then translated into bilingual corpora for an exact match. Hence the corresponding French word is retrieved and displayed as output.

A history tab has been developed in our project to ensure that our user can access his previously accessed words so that he doesn't have to waste time in researching for the same words. It thus saves time and memory

The random word of the day increases our users knowledge and keeps his engaged to our project. It is a fun and an interactive part of the code.

A BRIEF EXPLANATION OF THE DATA STRUCTURES:

Binary search trees:

A binary search tree (BST) is a binary tree where each node has a Comparable key (and an associated value) and satisfies the restriction that the key in any node is larger than the keys in all nodes in that node's left subtree and smaller than the keys in all nodes in that node's right subtree. Each node contains a key, a value, a left link, a right link, and a node count. The left link

points to a BST for items with smaller keys, and the right link points to a BST for items with larger keys.

Search: A recursive algorithm to search for a key in a BST follows immediately from the recursive structure: If the tree is empty, we have a search miss; if the search key is equal to the key at the root, we have a search hit. Otherwise, we search (recursively) in the appropriate subtree. The recursive get() method implements this algorithm directly. It takes a node (root of a subtree) as first argument and a key as second argument, starting with the root of the tree and the search key.

Insert.: Insert is not much more difficult to implement than search. Indeed, a search for a key not in the tree ends at a null link, and all that we need to do is replace that link with a new node containing the key. The recursive put() method accomplishes this task using logic similar to that we used for the recursive search: If the tree is empty, we return a new node containing the key and value; if the search key is less than the key at the root, we set the left link to the result of inserting the key into the left subtree; otherwise, we set the right link to the result of inserting the key into the right subtree.

Trie:

A trie (from retrieval), is a multi-way tree structure useful for storing strings over an alphabet.
It has been used to store large dictionaries of English (say) words in spelling-checking programs and in natural-language "understanding" programs.
A trie tree uses the property that if two strings have a common prefix of length n then these two will have a common path in the trie tree till the length n.
The end_string flag is set if there exists a string that ends at this node.

The next_char[0-26] represents the next character of the string.

Time complexity:
Insert operation takes a time of O(n) where n is the length of string to be inserted.

Space complexity:
The total space taken is O(n * 26) where n is the number of nodes in a tree.

Linked list

A linked list is a set of dynamically allocated nodes, arranged in such a way that each node contains one value and one pointer.

The pointer always points to the next member of the list. If the pointer is NULL, then it is the last node in the list.

A linked list is held using a local pointer variable which points to the first item of the list. If that pointer is also NULL, then the list is considered to be empty.

Searching a node means finding the node that contains the value being searched. This is in fact a very simple task if we talk about linear search (Note that there can be many search algorithms). One just needs to start with the first node and then compare the value which is being searched with the value contained in this node. If the value does not match then through the ‘next’ pointer (which contains the address of next node) the next node is accessed and same value comparison is done there. The search goes on until last node is accessed or node is found whose value is equal to the value being searched

Stack

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).
Mainly the following three basic operations are performed in the stack:

- **Push:** Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
- **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
- **Peek or Top:** Returns top element of stack.
- **isEmpty:** Returns true if stack is empty, else false.

METHODOLOGY:

Our is designed in such a way to perform a series of operations using the above mentioned data structures

- Insert a new word in the translator
- Display all the words
- Search for a English word
- Search for a French word
- Delete a word
- Search Words under the given prefix both English and French
- Display Search History
- Random word of the day
- Index – Both English and French

The Translator we designed is basically stored in a Binary Search Tree. Which has two strings in it's structure. And words are inserted in the dictionary following the constraints in a regular BST thereby maintaining its structure.

Searching is also done in a similar way as we traverse through the nodes of the BST based on the element to be searched

Display all words – This module is implemented by traversing through all the nodes of the translator structure and displaying the English words along with it's French equivalent.

Delete a word – This module is implemented by removing the node to be removed and filling up the hole formed in the tree such that the structure of the BST is kept intact.

Prefix search – This is done using trie data structure. Trie displays all the words in the given translator which has the given prefix.

Search History – This is done using a stack. All the words searched by the user are pushed into a stack and are displayed along with the search frequency

Random Word of the day – This is done using a time function which stops at random as we traverse through the nodes of the translator structure and displays the data stored in the particular node.

LOADING INITIAL DATA:

Data is loaded into our translator structure using file handling. A set of English words along with their French equivalents. These words are inserted into the structures to form a basic translator with some words. However the words which aren't present in the translator can be inserted using the insert function.

CODE:

```
1.  # include <iostream>
2.  # include <iomanip>
3.  # include <string>
4.  # include <math.h>
5.  # include <fstream>
6.  # include <bits/stdc++.h>
7.  # include <time.h>

8.  using namespace std;

9.  #define ALPHABET_SIZE (26)
10. #define CHAR_TO_INDEX(c) ((int)c - (int)'a')

11. string history_stack[100];
12. int top=-1;

13. struct Node
14. {
15. string eng;
16. string fre;
17. struct Node* next;
18. };
```

```
19. class node
20. {
21. public:
22. string data;
23. string syn;
24. node *parent;
25. node *leftChild;
26. node *rightChild;

27. node()
28. {
                a.  data = '0';
                b.  syn = '0';
                c.  parent = NULL;
                d.  leftChild = NULL;
                e.  rightChild = NULL; 29. }

30. };

31. class bst
32. {
33. public:
34. node* root;

35. bst()
36. {
37. root = NULL;
38. }
39. void insert(string data, string);
40. void display(node*, int);
41. node* search1(string data);
42. node* search2(string syn);
43. void deletion(node *z);
44. void transplant(node* u, node* v);
45. node* successor(node* currentNode);

46. };

47. void bst::insert(string data, string syn)
48. {
49. node* newNode = new node();
50. newNode->data = data;
51. newNode->syn = syn;
52. if (root == NULL)
53. {
        a.  root = newNode;
54. }
55. else
56. {
        a.  node* tempNode = new node();
        b.  node* backTempNode = new node();
        c.  tempNode = root;

        d.  while (tempNode != NULL)
        e.  {
                i.    backTempNode = tempNode;
                ii.   if (tempNode->data <= newNode->data)
                iii.  {
                                1. tempNode = tempNode-
                          >rightChild; iv. }
               v.   else
               vi.  {
                        1. tempNode = tempNode->leftChild; vii.
             }
        f.  }

        g.  newNode->parent = backTempNode;
        h.  if (backTempNode->data <= newNode->data)
        i.  {
        i.  backTempNode->rightChild = newNode;
        j.  }
        k.  else
        l.  {
                i.    backTempNode->leftChild = newNode;
        m. }
57. }
58. }

59. void bst::display(node *Node, int level)
60. {
```

```
61. if (root == NULL)
62. {
        a.  cout << "Dictionary is empty." << endl;
63. }
64. else
65. {
        a.  if (Node->rightChild != NULL)
        b.  {
               i.    display(Node->rightChild, level + 1);
        c.  }

        d.  if (Node != root)
        e.  {
               i.    for (int i = 0; i < level + 1; i++)
               ii.   {
                                1.cout << "      ";
                      iii.}
        f.  }
        g.  else
        h.  {
        i.  cout << "Root->";
        i.  }

        j.  cout << Node->data << ":" << Node->syn << endl;

        k.  if (Node->leftChild != NULL)
        l.  {
               i.    display(Node->leftChild, level + 1);
       m.} 66.
}

    67.     }

    68.     node* bst::search1(string Data)
    69.     {
    70.     node* tempNode = new node();
    71.     tempNode = root;
    72.     while (tempNode != NULL)
    73.     {
        a.  if (tempNode->data == Data)
        b.  {
              i.     return tempNode;
        c.  }
        d.  else
        e.  {
              i.     if (tempNode->data <= Data)
              ii.    {
                              1. tempNode = tempNode->rightChild;
                      iii.} iv.else
                v.{
                       1. tempNode = tempNode->leftChild; vi.
              }
        f.  }
    74.     }
    75.     return NULL;
    76.     }

    77.     node* bst::search2(string syn)
    78.     {
    79.     node* tempNode = new node();
    80.     tempNode = root;
    81.     while (tempNode != NULL)
    82.     {
        a.  if (tempNode->syn == syn)
        b.  {
              i.     return tempNode;
        c.  }
        d.  else
        e.  {
              i.     if (tempNode->syn <= syn)
              ii.    {
                              1. tempNode = tempNode->rightChild;
                      iii.} iv.else
                v.{
                       1. tempNode = tempNode->leftChild; vi.
              }
        f.  }
    83.     }
    84.     return NULL;
    85.     }
```

```
86.     node* bst::successor(node* currentNode)
87.     {
88.     node* tempNode = new node();
89.     node* backTempNode = new node();
90.     tempNode = currentNode;
91.     if (tempNode->rightChild != NULL)
92.     {
    a.  tempNode = tempNode->rightChild;
    b.  while (tempNode != NULL)
    c.  {
          i.     backTempNode = tempNode;
          ii.    tempNode = tempNode->leftChild;
    d.  }
    e.  return backTempNode;
93.     }
94.     else
95.     {
    a.  backTempNode = tempNode;
    b.  tempNode = tempNode->parent;
    c.  while (tempNode != NULL && tempNode->rightChild == backTempNode)
    d.  {
          i.     backTempNode = tempNode;
          ii.    tempNode = tempNode->parent;
    e.  }
    f.  return tempNode;
96.     }
97.     }


98.     void bst::transplant(node* u, node* v)
99.     {
100.    if (u->parent == NULL)
101.    {
    a. root = v;
102.    }
103.    else
104.    {
    a.  if (u == u->parent->leftChild)
    b.  {
           i.  u->parent->leftChild = v;
    c.  }
    d.  else
    e.  {
           i.  u->parent->rightChild = v;
    f.  }
105.    }
106.    if (v != NULL)
107.    {
    a. v->parent = u->parent;
108.    }
109.    }


110.    void bst::deletion(node *z)
111.    {
112.    if (z->leftChild == NULL)
113.    {
    a. transplant(z, z->rightChild);
114.    }
115.    else
116.    {
    a.  if (z->rightChild == NULL)
    b.  {
          i. transplant(z, z->leftChild);
    c.  }
    d.  else
    e.  {
          i.     node* succesor = new node();
          ii.    succesor = successor(z);
          iii.   if (succesor->parent != z)
          iv.    {
                          1. transplant(succesor, succesor-
                 >rightChild);
                          2. succesor->rightChild = z-
                 >rightChild;
                          3. succesor->rightChild->parent =
                 succesor;
          v.     }
          vi.    transplant(z, succesor);
          vii.   succesor->leftChild = z->leftChild;
```

```
            viii.  succesor->leftChild->parent = succesor;
    f. }
117.    }
118.    delete z;
119.    }

120.    struct TrieNode
121.    {
122.    struct TrieNode *children[ALPHABET_SIZE];
123.    bool isWordEnd;
124.    };

125.    struct TrieNode *getNode(void)
126.    {
127.    struct TrieNode *pNode = new TrieNode;
128.    pNode->isWordEnd = false;

129.    for (int i = 0; i < ALPHABET_SIZE; i++)
    a.  pNode->children[i] = NULL;

130.    return pNode;
131.    }

132.    void insert(struct TrieNode *root, const string key)
133.    {
134.    struct TrieNode *pCrawl = root;
135.    for (int level = 0; level < key.length(); level++)
136.    {
    a.  int index = CHAR_TO_INDEX(key[level]);
    b.  if (!pCrawl->children[index])
            i. pCrawl->children[index] = getNode();

    c.  pCrawl = pCrawl->children[index];
137.    }

138.    pCrawl->isWordEnd = true;
139.    }

140.    bool search(struct TrieNode *root, const string key)
141.    {
142.    int length = key.length();
143.    struct TrieNode *pCrawl = root;
144.    for (int level = 0; level < length; level++)
145.    {
    a.  int index = CHAR_TO_INDEX(key[level]);

    b.  if (!pCrawl->children[index])
            i. return false;

    c.  pCrawl = pCrawl->children[index];
146.    }

147.    return (pCrawl != NULL && pCrawl->isWordEnd);
148.    }

149.    bool isLastNode(struct TrieNode* root)
150.    {
151.    for (int i = 0; i < ALPHABET_SIZE; i++)
    a.  if (root->children[i])
            i. return 0;
152.    return 1;
153.    }

154.    void suggestionsRec(struct TrieNode* root, string currPrefix)
155.    {
156.    if (root->isWordEnd)
157.    {
    a.  cout << currPrefix;
    b.  cout << endl;
158.    }

159.    if (isLastNode(root))
    a.  return;

160.    for (int i = 0; i < ALPHABET_SIZE; i++)
161.    {
    a.  if (root->children[i])
    b.  {

            i. currPrefix.push_back(97 + i);
           ii. suggestionsRec(root->children[i], currPrefix);
```

```
    c.  }
162.    }
163.    }

164.    int printAutoSuggestions(TrieNode* root, const string query)
165.    {
166.    struct TrieNode* pCrawl = root;
167.    int level;
168.    int n = query.length();
169.    for (level = 0; level < n; level++)
170.    {
    a.  int index = CHAR_TO_INDEX(query[level]);

    b.  if (!pCrawl->children[index])
            i. return 0;

    c.  pCrawl = pCrawl->children[index];
171.    }

172.    bool isWord = (pCrawl->isWordEnd == true);


173.    bool isLast = isLastNode(pCrawl);

174.    if (isWord && isLast)
175.    {
    a.  cout << query << endl;
    b.  return -1;
176.    }
177.    if (!isLast)
178.    {
    a.  string prefix = query;
    b.  suggestionsRec(pCrawl, prefix);
    c.  return 1;
179.    }
180.    }

181.    struct TrieNode2
182.    {
183.    struct TrieNode2 *children[ALPHABET_SIZE];
184.    bool isWordEnd;
185.    };

186.    struct TrieNode2 *getNode2(void)
187.    {
188.    struct TrieNode2 *pNode = new TrieNode2;
189.    pNode->isWordEnd = false;
190.    for (int i = 0; i < ALPHABET_SIZE; i++)
    a.  pNode->children[i] = NULL;
191.    return pNode;
192.    }

193.    void insert2(struct TrieNode2 *root, const string key)
194.    {
195.    struct TrieNode2 *pCrawl = root;
196.    for (int level = 0; level < key.length(); level++)
197.    {
    a.  int index = CHAR_TO_INDEX(key[level]);
    b.  if (!pCrawl->children[index])
            i. pCrawl->children[index] = getNode2();

    c.  pCrawl = pCrawl->children[index];
198.    }

199.    pCrawl->isWordEnd = true;
200.    }

201.    bool search2(struct TrieNode2 *root, const string key)
202.    {
203.    int length = key.length();
204.    struct TrieNode2 *pCrawl = root;
205.    for (int level = 0; level < length; level++)
206.    {
    a.  int index = CHAR_TO_INDEX(key[level]);

    b.  if (!pCrawl->children[index])
            i. return false;

    c.  pCrawl = pCrawl->children[index];
207.    }
```

```
208.    return (pCrawl != NULL && pCrawl->isWordEnd);
209.    }

210.    bool isLastNode2(struct TrieNode2* root)
211.    {
212.    for (int i = 0; i < ALPHABET_SIZE; i++)
    a.  if (root->children[i])
            i. return 0;
213.    return 1;
214.    }

215.    void suggestionsRec2(struct TrieNode2* root, string currPrefix)
216.    {
217.    if (root->isWordEnd)
218.    {
    a.  cout << currPrefix;
    b.  cout << endl;
219.    }

220.    if (isLastNode2(root))
    a.  return;
221.    for (int i = 0; i < ALPHABET_SIZE; i++)
222.    {
    a.  if (root->children[i])
    b.  {

            i. currPrefix.push_back(97 + i);
           ii. suggestionsRec2(root->children[i], currPrefix);
    c.  }
223.    }
224.    }

225.    int printAutoSuggestions2(TrieNode2* root, const string query)
226.    {
227.    struct TrieNode2* pCrawl = root;

228.    int level;
229.    int n = query.length();
230.    for (level = 0; level < n; level++)
231.    {
    a.  int index = CHAR_TO_INDEX(query[level]);

    b.  if (!pCrawl->children[index])
            i. return 0;

    c.  pCrawl = pCrawl->children[index];
232.    }

233.    bool isWord = (pCrawl->isWordEnd == true);


234.    bool isLast = isLastNode2(pCrawl);

235.    if (isWord && isLast)
236.    {
    a.  cout << query << endl;
    b.  return -1;
237.    }
238.    if (!isLast)
239.    {
    a.  string prefix = query;
    b.  suggestionsRec2(pCrawl, prefix);
    c.  return 1;
240.    }
241.    }

242.    void push(string word)
243.    {
244.    history_stack[++top].assign(word);
245.    }

246.    void display()
247.    {
248.    cout<<top<<endl;
249.    if(top==-1)
    a.  printf("No elements have been searched");
250.    else
251.    {
    a.  printf("Your search history is:\n");
    b.  for(int i=0;i<=top;i++)
            i. cout<<history_stack[i]<<endl;
```

```
    }
    }

    void count()
    {
    string tem[100],tem1[100],w,x;
    int i,j,k=0,flag=0,coun=0;
    for(i=0;i<=top;i++)
    a.  tem[i]=history_stack[i];
    for(i=0;i<=top;i++)
    {
    a.  flag=0;
    b.  w=tem[i];
    c.  for(j=0;j<i;j++)
    d.  {
           i.   x=tem[j];
           ii.  if(w.compare(x)==0)
                   1. flag=1;
    e.  }
    f.  if(flag==0)
    g.  {
            i. tem1[k++]=w;
    h.  }
    }
    printf("Search Frequency is:\n");
    printf("Word\t\tCount\n");
    for(i=0;i<k;i++)
    {
    a.  coun=0;
    b.  w=tem1[i];
    c.  for(j=0;j<=top;j++)
    d.  {
          i.    x=tem[j];
          ii.   if(w.compare(x)==0)
          iii.  {
                   1. coun++; iv.
          }
    e.  }
    f.  cout<<w<<"\t\t"<<coun<<endl;
    }
    }

    void printRandom(struct Node *head)
    {
    if (head == NULL)
    return;

    srand(time(NULL));

    string result = head->eng;
    string result2 = head->fre;

    struct Node *current = head;
    int n;
    for (n=2; current!=NULL; n++)
    {
    a.  if (rand() % n == 0)
    b.  {
    c.  result.assign(current->eng);
    d.  result2.assign(current->fre);
    e.  }
    f.  // Move to next node
    g.  current = current->next;
    }

    cout<<result<<"\t"<<result2<<endl;
    }


    struct Node *newNode(string en,string fr)
    {

    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));

    new_node->eng.assign(en);
    new_node->fre.assign(fr);

    new_node->next =  NULL;

    return new_node;
    }
```

```
290.    void ins_list(struct Node** head_ref, string en,string fr)
291.    {
292.    struct Node* new_node = new Node();

293.    new_node->eng.assign(en);
294.    new_node->fre.assign(fr);

295.    new_node->next = (*head_ref);

296.    (*head_ref) = new_node;
297.    }

298.    string getString(char x)
299.    {
300.    string s(1, x);
301.    return s;
302.    }

303.    void menu()
304.    {
305.    char ch;
306.    string inttemp;
307.    string p;
308.    node *temp = new node();
309.    struct Node *head = NULL;
310.    bst myTree;
311.    int select;
312.    struct TrieNode* root = getNode();
313.    struct TrieNode2* root2 = getNode2();
314.    fstream newfile;
315.    newfile.open("words.txt",ios::in); //open a file to perform read operation
using file object
316.    if (newfile.is_open())
317.    {   //checking whether the file is open
    a.  string tp[2];
    b.  int i=0;
    c.  while(getline(newfile, tp[(i++)%2])){ //read data from file object and put it
        into string.
          i.    if(i%2==0)
          ii.   {
          iii.  myTree.insert(tp[0],tp[1]);
                              1. insert(root,tp[0]);
                              2. insert2(root2,tp[1]);
                              3. ins_list(&head,tp[0],tp[1]); iv. }
    d.  }
    e.  newfile.close(); //close the file object.
318.    }
319.    do
320.    {
    a.  cout<<endl<<endl<<"\t\t\t\tTRANSLATOR"<<endl<<endl;
    b.  cout << "0. Exit" << endl;
    c.  cout << "1. Insert" << endl;
    d.  cout << "2. Display" << endl;
    e.  cout << "3. English Search" << endl;
    f.  cout << "4. French Search" << endl;
    g.  cout << "5. Delete" << endl;
    h.  cout << "6. Search words under given prefix(ENG)" << endl;
    i.  cout << "7. Search words under given prefix(FRE)" << endl;
    j.  cout << "8. Display Search History" << endl;
    k.  cout << "9. Random Word of the day" << endl;
    l.  cout << "10. Index(ENG)"<<endl;
    m.  cout << "11. Index(FRE)"<<endl;
    n.  cout << endl << "Enter your selection:" << endl;

    o.  cin >> select;

    p.  switch (select)
    q.  {
    r.  case 0:
    s.  {
           i. break;
    t.  }
    u.  case 1:
    v.  {
           i. do ii.
          {
                              1.       string data;
                              2.       string syn;
                              3.       cout << "Enter the word (English,French meaning):
                  ";
```

```
                                    4.       cin >> data >> syn;
                                    5.       myTree.insert(data, syn);
                                    6.       insert(root,data);
                                    7.       ins_list(&head,data,syn);
                                    8.       cout << endl << "Do you want another word? [y or
                           n]" << endl;
                                    9.       cin >> ch; iii. } while (ch != 'n');
               iv. break;
          w.  }
          x.  case 2:
          y.  {
                 i.    myTree.display(myTree.root, 0);
                 ii.   cout << endl << "Press 0 to continue!" << endl;
                 iii.  cin >> ch;
                 iv.   break;
          z.  } aa. case 3:
          bb. {
                 i.    cout << "Enter English word:";
                 ii.   cin >> inttemp;
                 iii.  push(inttemp);
                 iv.   temp = myTree.search1(inttemp);
                 v.    if (temp != NULL)
                 vi.   {
                          1. cout << temp->data << ": " << temp->syn << endl; vii.
               }
               viii.  else
               ix.    {
                          1. cout << "It isn't in Dictionary!" << endl;
                 x. }
xi. cout << endl << "Press 0 to continue!" << endl;
              xii.   cin >> ch;
              xiii.  break;
          cc. } dd.
          case 4:
          ee. {
                 i.    cout << "Enter French word:";
                 ii.   cin >> p;
                 iii.  push(p);
                 iv.   temp = myTree.search2(p);
                 v.    if (temp != NULL)
                 vi.   {
                          1. cout << temp->syn << ": " << temp->data << endl; vii.
               }
               viii.  else
               ix.    {
                          1. cout << "It isn't in Dictionary!" << endl;
                 x. }
xi. cout << endl << "Press 0 to continue!" << endl;
              xii.   cin >> ch;
              xiii.  break;
          ff. } gg.
          case 5:
          hh. {
                 i.    cout << "Enter the word that you want delete: ";
                 ii.   cin >> inttemp;
                 iii.  temp = myTree.search1(inttemp);
                 iv.   if (temp == NULL)
                 v.    {
                          1. cout << inttemp << " isn't in Dictionary" << endl; vi.
                 }
               vii.   else
               viii.  {
                                    1.       myTree.deletion(temp);
                                    2.       cout << inttemp << " removed from
                              Dictionary." << endl; ix. }
                 x.    cout << endl << "Press 0 to continue!" << endl;
                 xi.   cin >> ch;
                 xii.  break;
          ii. } jj.
          case 6:
          kk. {
                 i.    string pre;
                 ii.   cout<<"Enter the prefix(ENG): ";
                 iii.  cin>>pre;
                 iv.   int comp = printAutoSuggestions(root, pre);
                 v.    if (comp == -1)
                          1. cout << "No other strings found with this prefix\n";
               vi. else if (comp == 0)
                          1. cout << "No string found with this prefix\n";
vii. break;
          ll. } mm.
          case 7:
```

```
            nn. {
                  i.    string pre;
                  ii.   cout<<"Enter the prefix(FRE): ";
                  iii.  cin>>pre;
                  iv.   int comp = printAutoSuggestions2(root2, pre);
                  v.    if (comp == -1)
                           1. cout << "No other strings found with this prefix\n";
                      vi. else if (comp == 0)
                                1. cout << "No string found with this prefix\n";
vii. break;
            oo. } pp.
            case 8:
            qq. {
                  i.    display();
                  ii.   count();
                  iii.  break;
            rr. } ss.
            case 9:
            tt. {
                  i.    cout<<"\t\t\tWORD OF THE DAY"<<endl;;
                  ii.   cout<<"ENG\t\tFRE"<<endl;
                  iii.  printRandom(head);
                  iv.   break;
            uu. } vv.
            case 10:
            ww. {
                   i. int cho;
                 ii. cout<<"Display English words starting with: "<<endl;
                  iii. cout<<"1.\ta"<<endl; iv.
                 cout<<"2.\tb"<<endl; v.
                 cout<<"3.\tc"<<endl; vi.
                 cout<<"4.\td"<<endl; vii.
                 cout<<"5.\te"<<endl; viii.
                 cout<<"6.\tf"<<endl; ix.
                 cout<<"7.\tg"<<endl; x.
                 cout<<"8.\th"<<endl; xi.
                 cout<<"9.\ti"<<endl;
                  xii. cout<<"10.\tj"<<endl;
               xiii. cout<<"11.\tk"<<endl; xiv.
               cout<<"12.\tl"<<endl; xv.
               cout<<"13.\tm"<<endl; xvi.
               cout<<"14.\tn"<<endl; xvii.
               cout<<"15.\to"<<endl; xviii.
               cout<<"16.\tp"<<endl; xix.
               cout<<"17.\tq"<<endl; xx.
               cout<<"18.\tr"<<endl; xxi.
               cout<<"19.\ts"<<endl; xxii.
               cout<<"20.\tt"<<endl; xxiii.
               cout<<"21.\tu"<<endl; xxiv.
               cout<<"22.\tv"<<endl; xxv.
               cout<<"23.\tw"<<endl; xxvi.
               cout<<"24.\tx"<<endl; xxvii.
               cout<<"25.\ty"<<endl; xxviii.
               cout<<"26.\tz"<<endl;
                  xxix. cout<<"Enter your choice: ";
                   xxx. cin>>cho;
                  xxxi. if((cho>26)||(cho<=0))
                 xxxii.  {
                              1. cout<<"Invalid Input!"<<endl;
                 xxxiii. } xxxiv. else
                   xxxv. {
                                1.  int comp = printAutoSuggestions(root, getString(cho+96));
                                2.  if (comp == -1)
                                       a. cout << "No other strings found with this prefix\n";

                              3. else if (comp == 0)
                                       a. cout << "No string found with this prefix\n";
                    xxxvi. }

               xxxvii. break;
            xx. } yy.
            case 11:
            zz. {
                   i. int cho;
                 ii. cout<<"Display French words starting with: "<<endl;
                  iii. cout<<"1.\ta"<<endl; iv.
                 cout<<"2.\tb"<<endl; v.
                 cout<<"3.\tc"<<endl; vi.
                 cout<<"4.\td"<<endl; vii.
                 cout<<"5.\te"<<endl; viii.
                 cout<<"6.\tf"<<endl; ix.
                 cout<<"7.\tg"<<endl; x.
```

```
         cout<<"8.\th"<<endl; xi.
         cout<<"9.\ti"<<endl;
          xii. cout<<"10.\tj"<<endl;
         xiii. cout<<"11.\tk"<<endl; xiv.
         cout<<"12.\tl"<<endl; xv.
         cout<<"13.\tm"<<endl; xvi.
         cout<<"14.\tn"<<endl; xvii.
         cout<<"15.\to"<<endl; xviii.
         cout<<"16.\tp"<<endl; xix.
         cout<<"17.\tq"<<endl; xx.
         cout<<"18.\tr"<<endl; xxi.
         cout<<"19.\ts"<<endl; xxii.
         cout<<"20.\tt"<<endl; xxiii.
         cout<<"21.\tu"<<endl; xxiv.
         cout<<"22.\tv"<<endl; xxv.
         cout<<"23.\tw"<<endl; xxvi.
         cout<<"24.\tx"<<endl; xxvii.
         cout<<"25.\ty"<<endl; xxviii.
         cout<<"26.\tz"<<endl;
          xxix. cout<<"Enter your choice: ";
           xxx. cin>>cho;
          xxxi. if((cho>26)||(cho<=0))
        xxxii. {
                        1. cout<<"Invalid Input!"<<endl;
      xxxiii. } xxxiv.
        else
         xxxv. {
                        1.  int comp = printAutoSuggestions2(root2, getString(cho+96));
                        2.  if (comp == -1)
                               a. cout << "No other strings found with this prefix\n";

                    3. else if (comp == 0)
                               a. cout << "No string found with this prefix\n";
        xxxvi. }

       xxxvii. break;
    aaa.        } bbb.
        default:
            i. break;
    ccc.        }

321.    } while (select != 0);
322.    }



323.    int main()
324.    {

325.    menu();
326.    return 0; 327. }
```

OUTPUT: Here are the output screeenshots for the above described functions.

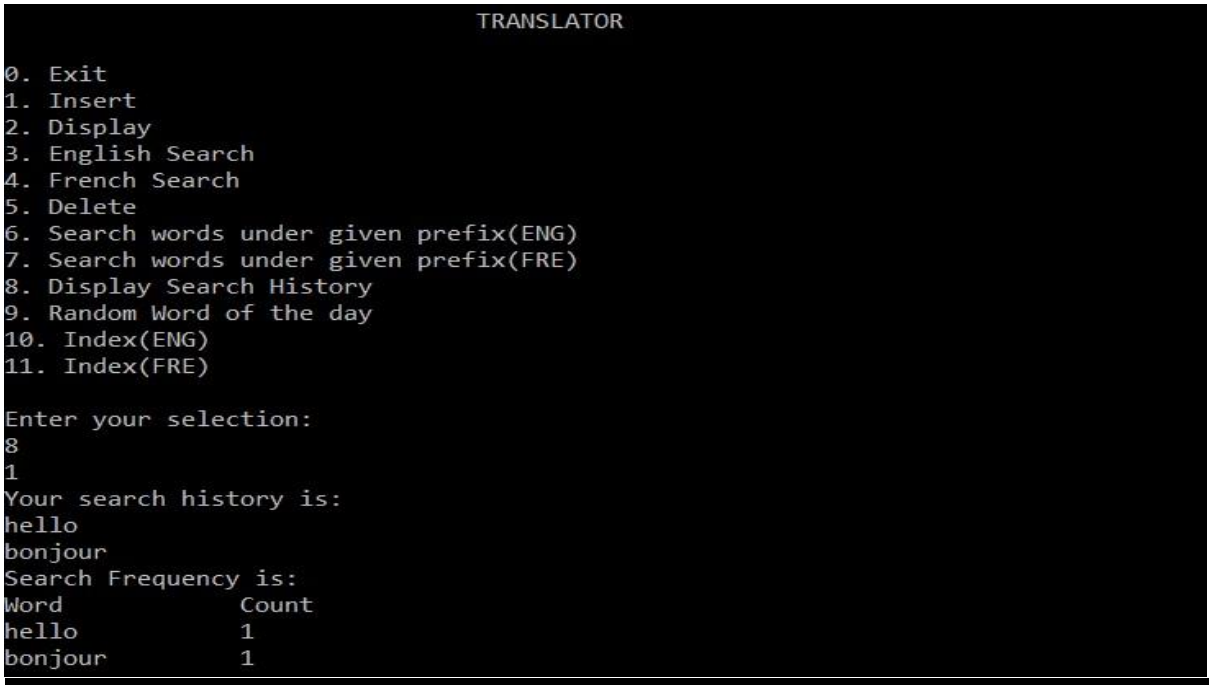
```
                                TRANSLATOR

0. Exit
1. Insert
2. Display
3. English Search
4. French Search
5. Delete
6. Search words under given prefix(ENG)
7. Search words under given prefix(FRE)
8. Display Search History
9. Random Word of the day
10. Index(ENG)
11. Index(FRE)

Enter your selection:
3
Enter English word:hello
hello: bonjour

Press 0 to continue!
```

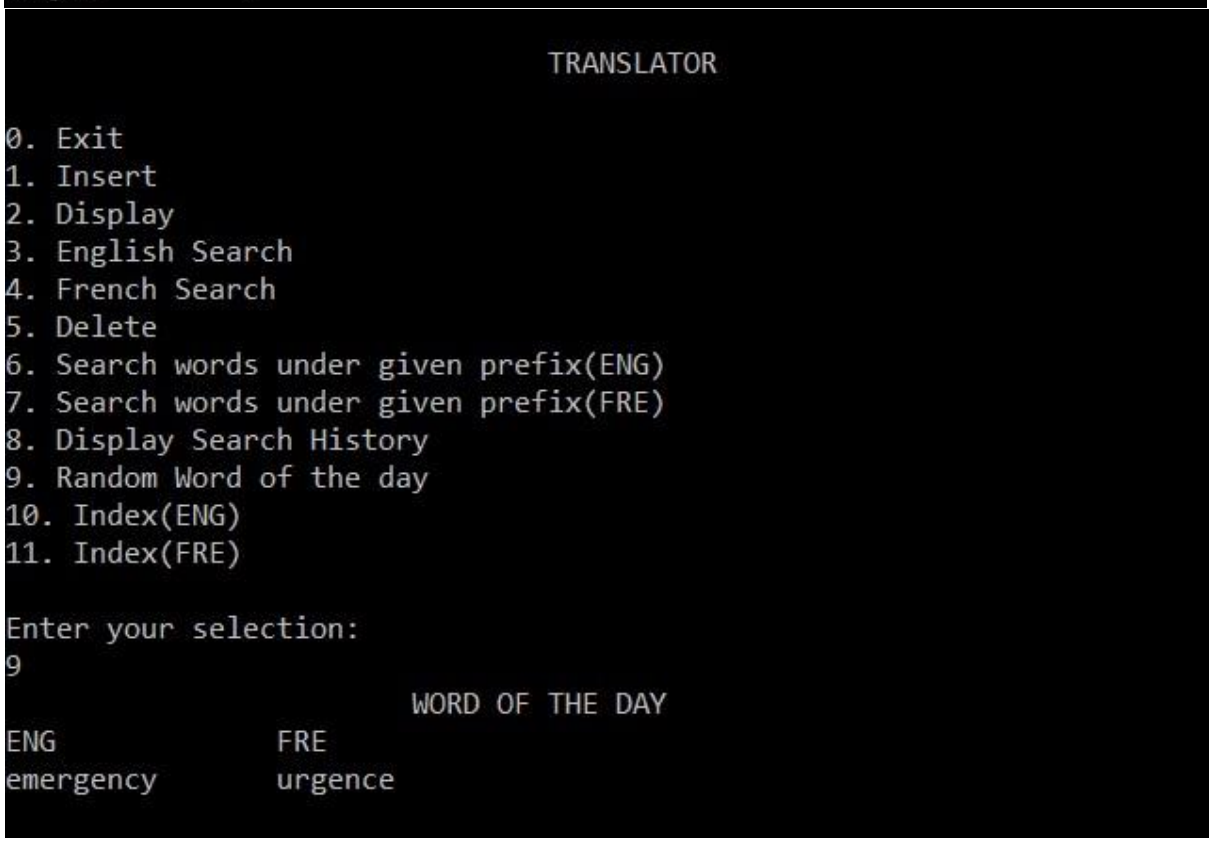
```
                                TRANSLATOR

0. Exit
1. Insert
2. Display
3. English Search
4. French Search
5. Delete
6. Search words under given prefix(ENG)
7. Search words under given prefix(FRE)
8. Display Search History
9. Random Word of the day
10. Index(ENG)
11. Index(FRE)

Enter your selection:
4
Enter French word:bonjour
bonjour: hello

Press 0 to continue!
```

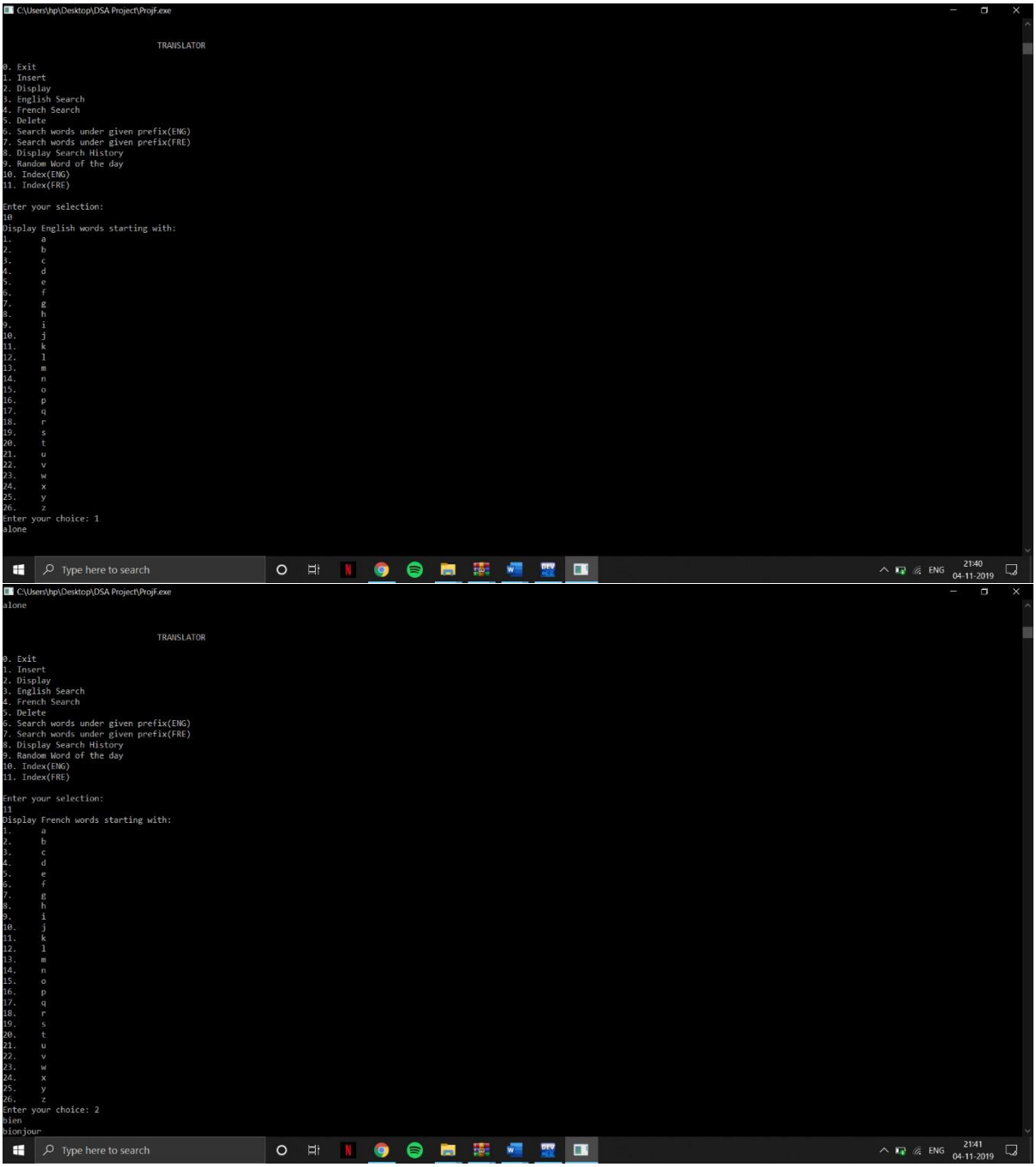
```
                                TRANSLATOR

0. Exit
1. Insert
2. Display
3. English Search
4. French Search
5. Delete
6. Search words under given prefix(ENG)
7. Search words under given prefix(FRE)
8. Display Search History
9. Random Word of the day
10. Index(ENG)
11. Index(FRE)

Enter your selection:
5
Enter the word that you want delete: four
four removed from Dictionary.
```

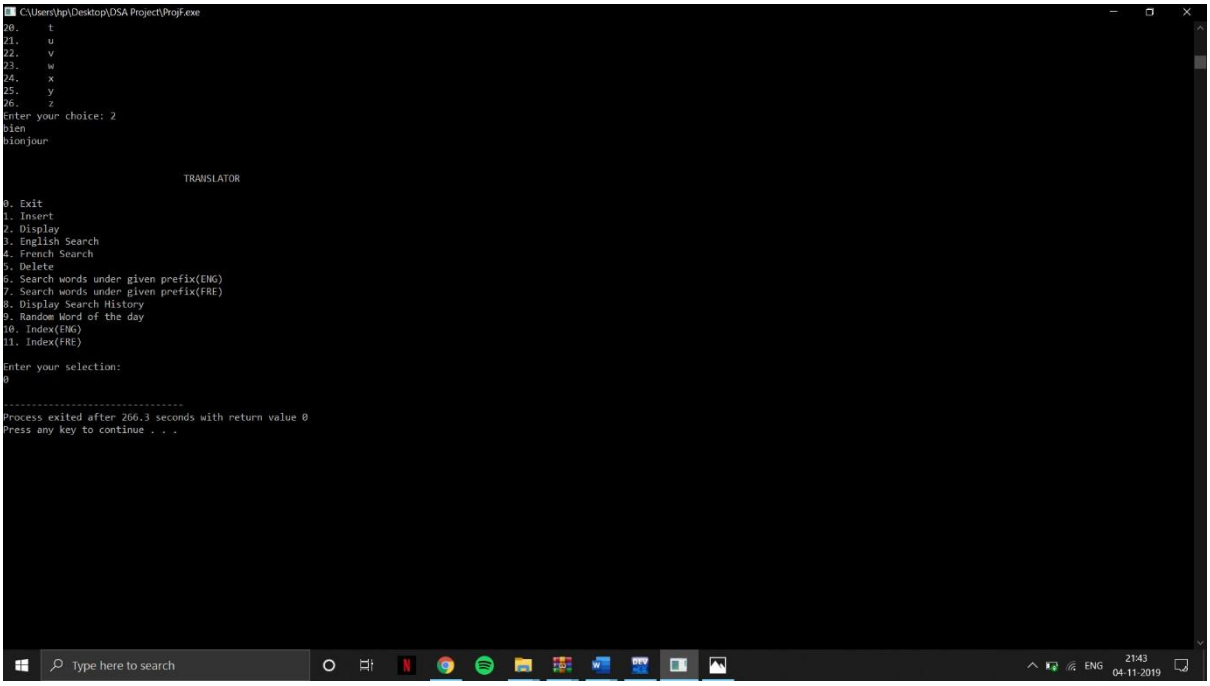
```
                                TRANSLATOR

0. Exit
1. Insert
2. Display
3. English Search
4. French Search
5. Delete
6. Search words under given prefix(ENG)
7. Search words under given prefix(FRE)
8. Display Search History
9. Random Word of the day
10. Index(ENG)
11. Index(FRE)

Enter your selection:
6
Enter the prefix(ENG): he
hello
helre
```

```
                                TRANSLATOR

0. Exit
1. Insert
2. Display
3. English Search
4. French Search
5. Delete
6. Search words under given prefix(ENG)
7. Search words under given prefix(FRE)
8. Display Search History
9. Random Word of the day
10. Index(ENG)
11. Index(FRE)

Enter your selection:
7
Enter the prefix(FRE): b
bien
bionjour
```


TRANSLATOR
0. Exit
1. Insert
2. Display
3. English Search
4. French Search
5. Delete
6. Search words under given prefix(ENG)
7. Search words under given prefix(FRE)
8. Display Search History
9. Random Word of the day
10. Index(ENG)
11. Index(FRE)
Enter your selection:
8
1
Your search history is:
hello
bonjour
Search Frequency is:
Word Count
hello 1
bonjour 1


TRANSLATOR
0. Exit
1. Insert
2. Display
3. English Search
4. French Search
5. Delete
6. Search words under given prefix(ENG)
7. Search words under given prefix(FRE)
8. Display Search History
9. Random Word of the day
10. Index(ENG)
11. Index(FRE)
Enter your selection:
9
WORD OF THE DAY
ENG FRE
emergency urgence


C:\Users\hp\Desktop\DSA Project\ProjF.exe
TRANSLATOR
0. Exit
1. Insert
2. Display
3. English Search
4. French Search
5. Delete
6. Search words under given prefix(ENG)
7. Search words under given prefix(FRE)
8. Display Search History
9. Random Word of the day
10. Index(ENG)
11. Index(FRE)
Enter your selection:
10
Display English words starting with:
1. a
2. b
3. c
4. d
5. e
6. f
7. g
8. h
9. i
10. j
11. k
12. l
13. m
14. n
15. o
16. p
17. q
18. r
19. s
20. t
21. u
22. v
23. w
24. x
25. y
26. z
Enter your choice: 1
alone
Type here to search
ENG
21:40
04-11-2019
C:\Users\hp\Desktop\DSA Project\ProjF.exe
alone
TRANSLATOR
0. Exit
1. Insert
2. Display
3. English Search
4. French Search
5. Delete
6. Search words under given prefix(ENG)
7. Search words under given prefix(FRE)
8. Display Search History
9. Random Word of the day
10. Index(ENG)
11. Index(FRE)
Enter your selection:
11
Display French words starting with:
1. a
2. b
3. c
4. d
5. e
6. f
7. g
8. h
9. i
10. j
11. k
12. l
13. m
14. n
15. o
16. p
17. q
18. r
19. s
20. t
21. u
22. v
23. w
24. x
25. y
26. z
Enter your choice: 2
bien
bienjour
Type here to search
ENG
21:41
04-11-2019



REFERENCES:

- GeeksforGeeks
- StackExchange
    - Youtube