CSE2003 - Data Structure And Algorithms
Embedded Project – C++/C
Winter Semester – 2019 ~ 2020
L41+L42 Slot

E- Record:

AutoComplete Word Using Trie Data Structure

Submitted By
Name of the Student : Aman Goel
Registration Number : 18BCE0895
BTECH (CSE) – 2ND YEAR
SCOPE

VELLORE INSTITUTE OF TECHNOLOGY
VELLORE – 632014
TAMIL NADU INDIA
DATE: 2nd—June– 2019

Note: (https://drive.google.com/drive/folders/1MtQQMRhKwC1fJj-RBmV5he0gXOTUStmy?usp=sharing) Link containing the word_alpha text file which contains all the words in a dictionary and cpp,readme file.

Word AutoComplete :

Autocomplete, or word completion, is a feature in which an application predicts the rest of a word a user is typing. In graphical user interfaces, users can typically press the tab key to accept a suggestion or the down arrow key to accept one of several.

Autocomplete speeds up human-computer interactions when it correctly predicts the word a user intends to enter after only a few characters have been typed into a text input field. It works best in domains with a limited number of possible words (such as in command line interpreters), when some words are much more common (such as when addressing an e-mail), or writing structured and predictable text (as in source code editors).

Many autocomplete algorithms learn new words after the user has written them a few times, and can suggest alternatives based on the learned habits of the individual user.

Original purpose

The original purpose of word prediction software was to help people with physical disabilities increase their typing speed,] as well as to help them decrease the number of keystrokes needed in order to complete a word or a sentence. The need to increase speed is noted by the fact that people who use speech-generating devices generally produce speech at a rate that is less than 10% as fast as people who use oral speech. But the function is also very useful for anybody who writes text, particularly people–such as medical doctors–who frequently use long, hard-to-spell terminology that may be technical or medical in nature.

Description

Autocomplete or word completion works so that when the writer writes the first letter or letters of a word, the program predicts one or more possible words as choices. If the word he intends to write is included in the list he can select it, for example by using the number keys. If the word that the user wants is not predicted, the writer must enter the next letter of the word. At this time, the word choice(s) is altered so that the words provided begin with the same letters as those that have been selected. When the word that the user wants appears it is selected, and the word is inserted into the text.[4][5] In another form of word prediction, words most likely to follow the just written one are predicted, based on recent word pairs used.[5] Word prediction uses language modeling, where within a set vocabulary the words are most likely to occur are calculated.[6] Along with language modeling, basic word prediction on AAC devices is often coupled with a frecency model, where words the AAC user has used recently and frequently are more likely to be predicted.[3] Word prediction software often also allows the user to enter their own words into the word prediction dictionaries either directly, or by "learning" words that have been written.[4][5] Some search returns related to genitals or other vulgar terms are often omitted from autocompletion technologies, as are morbid terms.

Code:

```
#include<bits/stdc++.h>
#include<iostream>
#include<string>

#include<fstream>
using namespace std;


void delay(int number_of_seconds)
{
   // Converting time into milli_seconds
   int milli_seconds = 1000 * number_of_seconds;

   // Storing start time
   clock_t start_time = clock();

   // looping till required time is not achieved
   while (clock() < start_time + milli_seconds)
       ;
}

class TrieNode {
	public :
	char data;
	TrieNode **children;
	bool isTerminal;

	TrieNode(char data) {
		this -> data = data;
		children = new TrieNode*[26];
		for(int i = 0; i < 26; i++) {
```

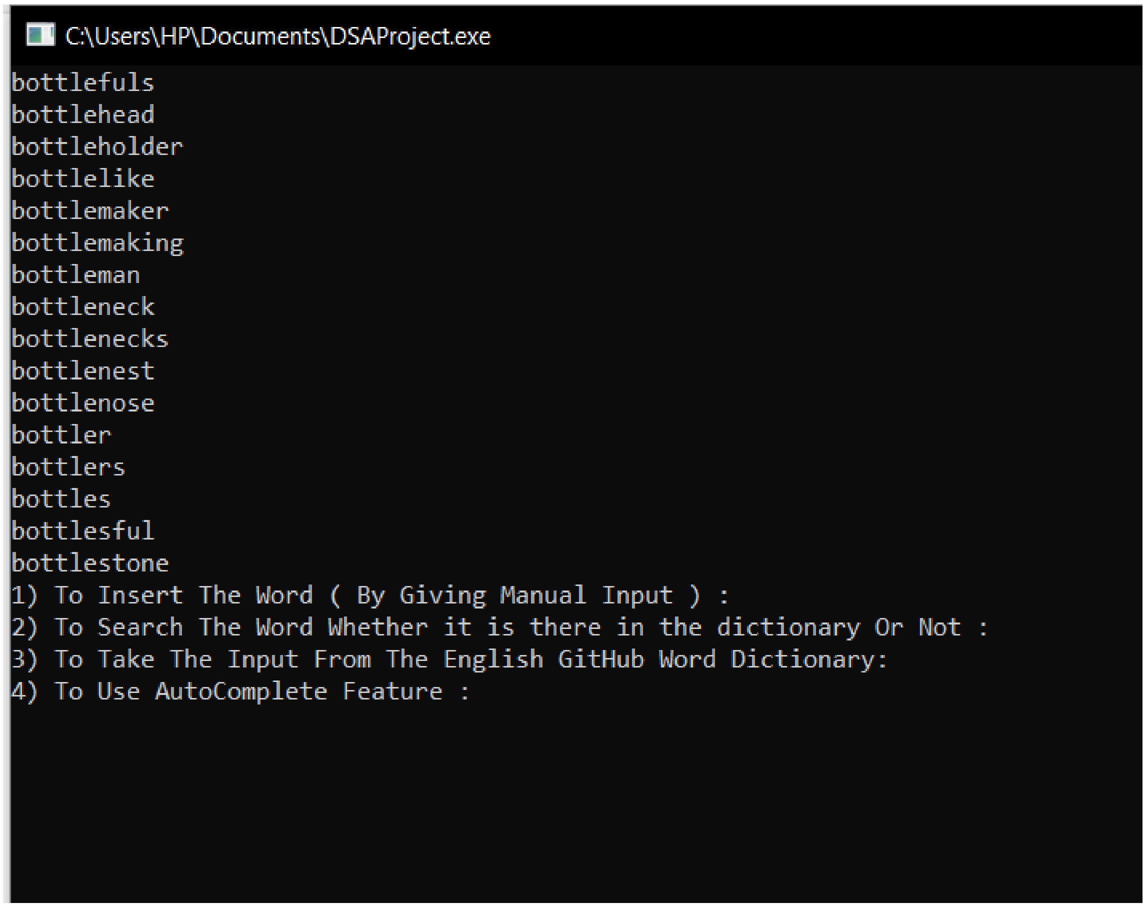
```
                                    children[i] = NULL;
                            }
                            isTerminal = false;
                    }
};

class Trie {
    TrieNode *root;

    public :

    Trie() {
        root = new TrieNode('\0');
    }

    void insertWord(TrieNode *root, string word) {
        // Base case
        if(word.size() == 0) {
            root -> isTerminal = true;
            return;
        }

        // Small Calculation
        int index = word[0] - 'a';
        TrieNode *child;
        if(root -> children[index] != NULL) {
            child = root -> children[index];
        }
        else {
            child = new TrieNode(word[0]);
            root -> children[index] = child;
        }

        // Recursive call
        insertWord(child, word.substr(1));
    }

    // For user
    void insertWord(string word) {
        insertWord(root, word);
    }


    bool search(string word) {
        // Write your code here
        return search(root,word);

    }
```

```
bool search(TrieNode* root,string word){
    if(word.size() == 0){
        if(root->isTerminal == true){
            return true;
        }else{
            return false;
        }
    }
    TrieNode* child;
    int index = word[0] - 'a';
    if(root->children[index] != NULL){
        child = root->children[index];
        return search(child,word.substr(1));
    }else{
        return false;
    }
}

  void print(TrieNode* root,string output){
    if(root == NULL){
        return;
    }

    if(root->isTerminal){
        cout<<output<<endl;
    }

    for(int i=0;i<26;i++){
        if(root->children[i] != NULL){
            string mm = output  + root->children[i]->data;
            print(root->children[i],mm);
        }
    }
}


void helper(TrieNode* root,string pattern,string output){
    if(root == NULL){
        return ;
    }

    if(pattern.size() == 0){
        if(root->isTerminal){
            cout<<output<<endl;
        }
        for(int i=0;i<26;i++){
            if(root->children[i] != NULL){
                string mm = output + root->children[i]->data;
                print(root->children[i],mm);
              }
            }
            return ;
          }

          int index = pattern[0] - 'a';
          if(root->children[index]){
             output += root->children[index]->data;
             helper(root->children[index],pattern.substr(1),output);
          }
          return ;


      }

      void autoComplete(/*vector<string> input,*/string pattern) {
          // Complete this function
          // Print the output as specified in question
       //   for(int i=0;i<input.size();i++){
         //    insertWord(input[i]);
         // }

          string output = "";
          helper(root,pattern,output);
      }




   };

   int main(){
               cout<<endl<<endl<<endl<<endl;
               cout<< "\t\t\t\t\t\t\t Name : Aman Goel  "<<endl;
               cout<< "\t\t\t\t\t\t\t Registration Number : 18BCE0895 "<<endl;
               cout<< "\t\t\t\t\t\t\t Subject Course Code: CSE2003"<<endl;
               cout<< "\t\t\t\t\t\t\t Slot : B1"<<endl;
               cout<< "\t\t\t\t\t\t\t Faculty Name : Gopinath MP"<<endl;
               delay(5);
               system("cls");

               cout<<"1) To Insert The Word ( By Giving Manual Input ) : "<<endl;
               //cout<<"2) To Search The Word Whether it is there in the dictionary Or Not : "<<endl;
               cout<<"3) To Take The Input From The English GitHub Word Dictionary: "<<endl;
               //cout<<"4) To Autocomplete a word : "<<endl;
```

```
    int choice;
    cin >> choice;
    Trie t;
    while(choice != -1){
        string word;
        bool ans;
        switch(choice) {
            case 1 :
              cout<<"Please Enter The Word To Append In The Dictionary : "<<endl;
               cin >> word;
               transform(word.begin(),word.end(),word.begin(),::tolower);
               t.insertWord(word);
               cout<<"Word Got Inserted Successfully : "<<endl;
               break;
            case 2 :
              cout<<"Please enter the word to search : "<<endl;
               cin >> word;
               ans = t.search(word);
               if (ans) {
                   cout << "true" << endl;
               } else {
                   cout << "false" << endl;
               }
               break;
            case 3:
              {
                                    ifstream file;
                                                file.open("words_alpha.txt");
                                                if(!file.is_open()){
                                                            cout<<"Error while opening the file";
                                                }else{
                                                            cout<<"Process Initiated"<<endl;
                                                            string line;
                                                            while(file.good()){
                                                                        getline(file,line);
                                                                        t.insertWord(line);
                                                                        //cout<<line.size()<<endl;
                                                            }
                                                            cout<<"Process Ended"<<endl;
                                                }

                                                }
                                                break;
                                    case 4:
                                                cout<<"Please enter the word for Autocomplete feature to work:
  "<<endl;
                                                cin>>word;
                                                transform(word.begin(),word.end(),word.begin(),::tolower);
```

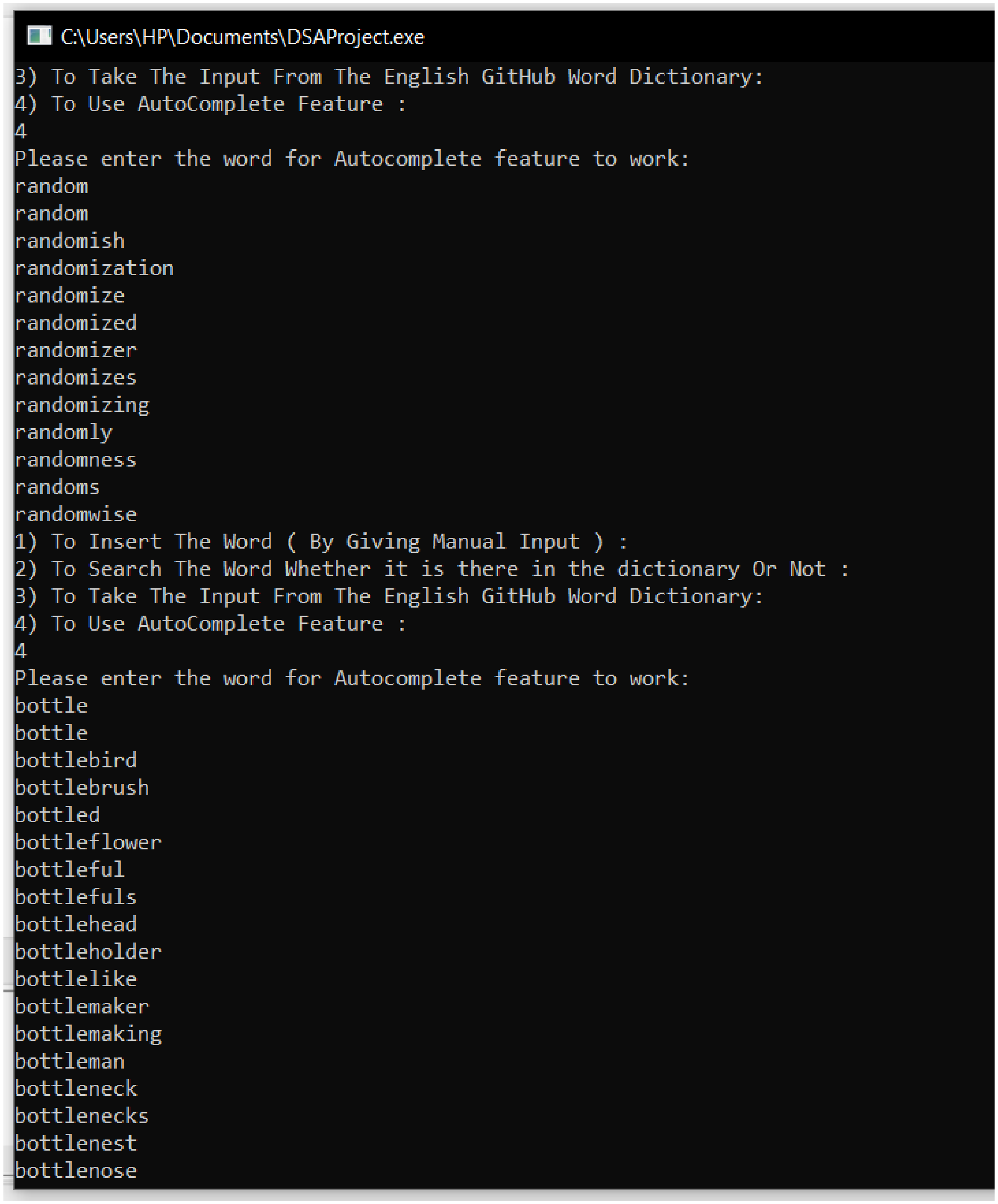
```
                                        t.autoComplete(word);
                                        break;
            default :
              return 0;
        }
        cout<<"1) To Insert The Word ( By Giving Manual Input ) : "<<endl;
                    cout<<"2) To Search The Word Whether it is there in the dictionary Or Not : "<<endl;
                    cout<<"3) To Take The Input From The English GitHub Word Dictionary: "<<endl;
                    cout<<"4) To Use AutoComplete Feature : "<<endl;
        cin >> choice;
    }
    return 0;
}
```

Output:



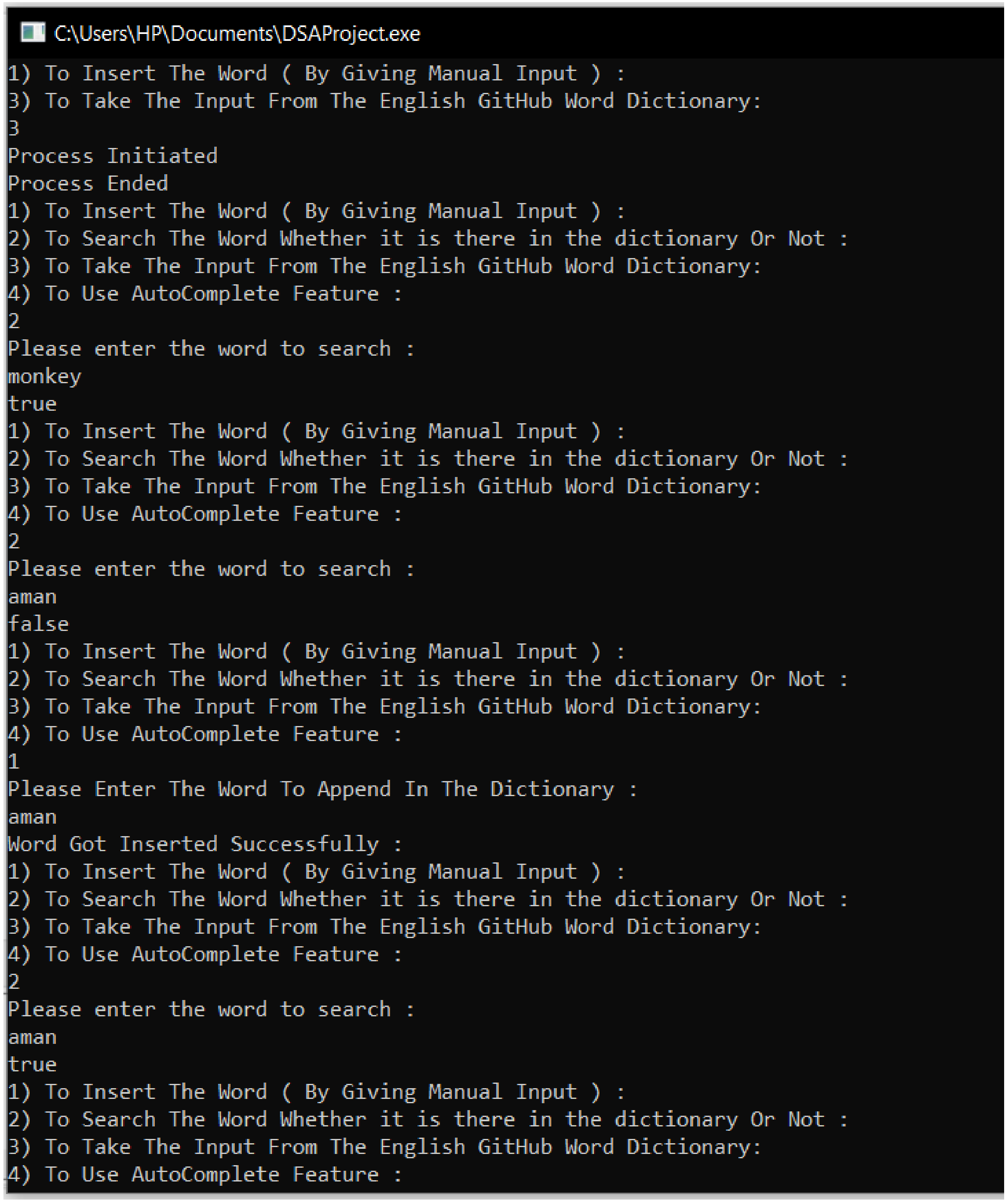
C:\Users\HP\Documents\DSAProject.exe

```
bottlefuls
bottlehead
bottleholder
bottlelike
bottlemaker
bottlemaking
bottleman
bottleneck
bottlenecks
bottlenest
bottlenose
bottler
bottlers
bottles
bottlesful
bottlestone
1) To Insert The Word ( By Giving Manual Input ) :
2) To Search The Word Whether it is there in the dictionary Or Not :
3) To Take The Input From The English GitHub Word Dictionary:
4) To Use AutoComplete Feature :
```



C:\Users\HP\Documents\DSAProject.exe

```
3) To Take The Input From The English GitHub Word Dictionary:
4) To Use AutoComplete Feature :
4
Please enter the word for Autocomplete feature to work:
random
random
randomish
randomization
randomize
randomized
randomizer
randomizes
randomizing
randomly
randomness
randoms
randomwise
1) To Insert The Word ( By Giving Manual Input ) :
2) To Search The Word Whether it is there in the dictionary Or Not :
3) To Take The Input From The English GitHub Word Dictionary:
4) To Use AutoComplete Feature :
4
Please enter the word for Autocomplete feature to work:
bottle
bottle
bottlebird
bottlebrush
bottled
bottleflower
bottleful
bottlefuls
bottlehead
bottleholder
bottlelike
bottlemaker
bottlemaking
bottleman
bottleneck
bottlenecks
bottlenest
bottlenose
```



```
C:\Users\HP\Documents\DSAProject.exe
1) To Insert The Word ( By Giving Manual Input ) :
3) To Take The Input From The English GitHub Word Dictionary:
3
Process Initiated
Process Ended
1) To Insert The Word ( By Giving Manual Input ) :
2) To Search The Word Whether it is there in the dictionary Or Not :
3) To Take The Input From The English GitHub Word Dictionary:
4) To Use AutoComplete Feature :
2
Please enter the word to search :
monkey
true
1) To Insert The Word ( By Giving Manual Input ) :
2) To Search The Word Whether it is there in the dictionary Or Not :
3) To Take The Input From The English GitHub Word Dictionary:
4) To Use AutoComplete Feature :
2
Please enter the word to search :
aman
false
1) To Insert The Word ( By Giving Manual Input ) :
2) To Search The Word Whether it is there in the dictionary Or Not :
3) To Take The Input From The English GitHub Word Dictionary:
4) To Use AutoComplete Feature :
1
Please Enter The Word To Append In The Dictionary :
aman
Word Got Inserted Successfully :
1) To Insert The Word ( By Giving Manual Input ) :
2) To Search The Word Whether it is there in the dictionary Or Not :
3) To Take The Input From The English GitHub Word Dictionary:
4) To Use AutoComplete Feature :
2
Please enter the word to search :
aman
true
1) To Insert The Word ( By Giving Manual Input ) :
2) To Search The Word Whether it is there in the dictionary Or Not :
3) To Take The Input From The English GitHub Word Dictionary:
4) To Use AutoComplete Feature :
```