

# Design Report

王雨阳 1809853Z-I011-0045

## ***1.Problem Specification:***

We need to write a program that can implement text search and we expect it could help users to finish the word puzzle. Compared to original one, we make some modification. According to a given text file or CSV dictionary in particular format, our program will generate a new dictionary(ordered in alphabet) simultaneously. This dictionary will record words and its rank of all the imported files, additionally, this dictionary can be updated more times(can be combined with other dictionaries). When users want to search words, they only need to input a fixed query pattern(like particular character and corresponding position number), our program will find all the words which satisfy this pattern and arrange them in rank order.

## ***2.Contribution of Project:***

The project is written by Mr. Wang Yuyang (Kennard Wang), who owns the copyright and the right of explanation. The source code will be upload to GitHub, you can visit it through link <https://github.com/KennardWang?tab=repositories>

## ***3.Project Specification:***

I write 2 versions: one is based on console, another is based on GUI. There are a few differences between them.

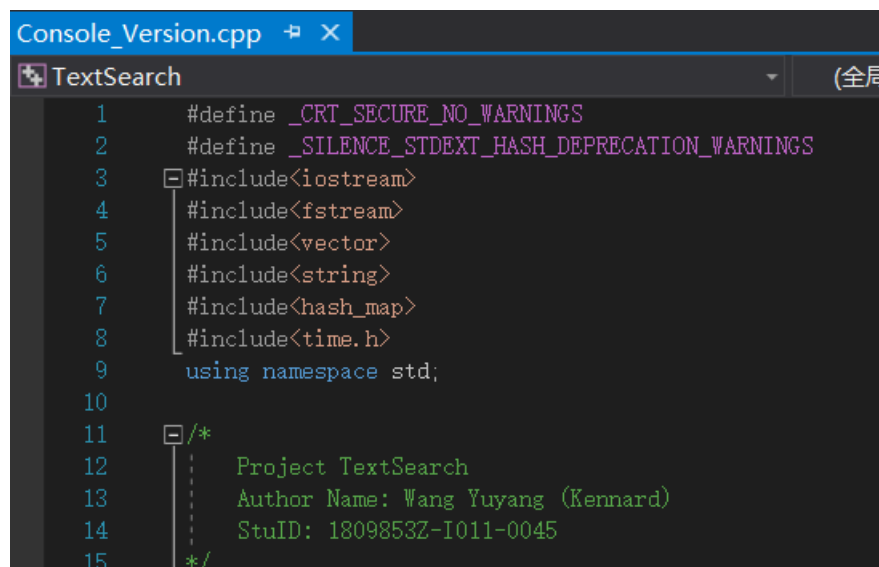
## Console Version:

### User Manual:

1. Please put the test file under the same root of **Console\_Version.cpp**.

名称	修改日期	类型	大小
Debug	2020/05/25 15:13	文件夹	
Console_Version.cpp	2020/05/25 15:13	C++ Source	7 KB
newsDict	2020/05/25 15:13	Microsoft Excel ...	146 KB
test	2020/04/08 11:53	文本文档	498 KB
testDict	2020/05/08 10:06	Microsoft Excel ...	98 KB
TextSearch.vcxproj	2020/05/25 15:13	VC++ Project	7 KB
TextSearch.vcxproj.filters	2020/05/25 15:13	VC++ Project Fil...	1 KB
TextSearch.vcxproj.user	2020/05/08 10:51	Per-User Project...	1 KB

2. Open source code via IDE or text editor. (Recommend Visual Studio 2017)



```
Console_Version.cpp
TextSearch
1  #define _CRT_SECURE_NO_WARNINGS
2  #define _SILENCE_STDTEXT_HASH_DEPRECATION_WARNINGS
3  #include<iostream>
4  #include<fstream>
5  #include<vector>
6  #include<string>
7  #include<hash_map>
8  #include<time.h>
9  using namespace std;
10
11  /*
12   Project TextSearch
13   Author Name: Wang Yuyang (Kennard)
14   StuID: 1809853Z-I011-0045
15  */
```

3. Run it and enter text file with ".txt" suffix.

```
Please enter the file directory you want to import:
test.txt
Load test.txt successfully! Time cost: 1339 ms.

Dictionary Completed ! Time cost: 140 ms.
```

4. Enter CSV dictionary with ".csv" suffix.

```
Please enter the file directory you want to import:
testDict.csv
Import testDict.csv successfully! Time cost: 357 ms.
Dictionary Completed ! Time cost: 157 ms.
```

5. If your directory is not correct, you will be allowed to input again.

```
Please enter the file directory you want to import:
hello.csv
Fail to load hello.csv. Please check your directory!
Please enter the file directory you want to import:
```

6. Then you can search words in fixed patterns. Using particular character and position number, also a comma between them, please do not use space.

```
Please input character and its position like 'u,3'
u,3
Continue ? (Y/N)
Y
Completed ! Time cost: 39 ms.

Please input character and its position like 'u,3'
a,4
Continue ? (Y/N)
N
```

7. The result will be ordered in rank.

```
usually, 199
square, 72
equally, 66
usual, 66
equal, 65
squad, 26
equation, 25
stuart, 21
equality, 15
squadron, 13
equations, 11
eduard, 7
Ecuador-born, 1
eduardo, 1
Equadorean-born, 1
```

## Data Structure:

Class Name/ main()	Element /function name	Modifier	Element type	Function return type	Function Parameter	Description
class completion	/	private	struct Word	/	/	Defined struct with 2 members: <b>string word</b> & <b>int rk</b>
	display	private	vector< Word>	/	/	Used for displaying both words and its rank
	c_dict	private	hash_map<string, int>	/	/	Dictionary, used to store word data
	askForFile()	public	/	string	void	Read the file directory users have entered
	splitStr()	public	/	string*	string str, string token	Split a string to a string array by token
	modifyStr()	public	/	void	string &str	Modify string words by trimming redundant prefix & suffix and transform abbreviation to full word

						Change Upper case character to lower case character
	fill_com pletions( )	public	/	void	string fd	Read text file through given file directory, then modify them and store in the dictionary
	importC SV()	public	/	void	string fd	Read CSV dictionary through given file directory, then modify them and store in the dictionary
	findWor d()	public	/	hash_ma p<strin g,int>	void	According to inputted pattern to search correct words from dictionary and return these words
	rankSort ()	public	/	int	hash_m ap<stri ng,int> hash	Rearrange those words searched by <b>findWord()</b> in rank order,

						then push them into vector <b>display</b> and output them
	fileWrite ()	public	/	void	void	Generate new dictionary based on <b>c_dict</b>
/	main()	/	/	int	void	Test function

### Core Algorithm:

#### 1. Storage:

I use **hash\_map<string,int> c\_dict** as a dictionary, this data structure could record key-value pair without duplicate, it is based on hash value and linked list, which means the insertion procedure is very fast. When a new word comes in, we compute the hash value of the key, and then link the key-value pair to the corresponding node, so the complexity is **O(1)**.

#### 2. Word Search:

We let users input fixed patterns including character and its exact position. Like 'u,3', it means we will find the word whose third character is U. I don't want to design a complicated algorithm so I use **Sequential Search** here, which has **O(N)** complexity average. [best case O(1), worst case O(N)]

#### 3. Rank Sort: ("First Search Last Sort")

We have learned about complexity in Java programming.

Selection Sort & Insertion Sort:  $O(N^2)$

Merge Sort & Heap Sort:  $O(N \log N)$

Quick Sort:  $O(N \ln N)$

All in all, time consumption is correlated to sort complexity, also based on size of N. The smaller N is, the faster the program will be. I remembered that in mid-term project I made a ranked dictionary which cost 36 sec. At this time, I decide to search words first because if you confirm the position of some characters, it will exclude most cases of the words.

I did an experiment. For example, we know that 'E' has the highest

appearing rate (12.25%), so we assume we know the exact position of 'E' and count the number of remaining words (allow repetition).

Here are the results:

Position of E	1	2	3	4	5	6	7	8	9
Total words	550	1925	875	1542	1714	1134	1027	756	527
12328	4.46%	15.6%	7.1%	12.5%	13.9%	9.2%	8.3%	6.13%	4.27%

Although it is an example, but it demonstrates clearly that the percentage of matching words has decreased to 10% around. We can compute the overall complexity is  $O(N)$  [search]+  $O(0.01N^2)$  [use selection sort]. However, if we use the former algorithm, the complexity will be  $O(N^2)$  [sort] +  $O(N)$  [search]. When  $N \sim 10000$ , our new algorithm is probably 99 times faster than the old one.

### System Feature:

#### 1. "Kit Class" Concept:

Based on OOP concept, I design completion class and encapsulate correlated method into it. To solve sustainable problem, I prefer to use "kit class" concept. The class is regarded as a kit, the operation to object could be executed by calling class method, which you do not define function parameters and your test main function will look briefly, too.

#### 2. Complexity:

The total complexity is  $N * O(1)$  [read file] +  $N * O(1)$  [insert key] +  $N * O(1)$  [write dictionary] +  $O(N)$  [search] +  $O(0.01N^2)$  [sort]  $\sim O(0.01N^2)$ . Although, I do not use quicker sort algorithm (Merge Sort) and search algorithm (BST), the high efficiency will be revealed most by "First Search Last Sort" algorithm. Compared to this, I think using quicker sort or search algorithms just has little influence.

#### 3. Dictionary Processing:

Compared to mid-term project, we retain all the words with hyphen and we also parse most abbreviations to words. For example, 're will be regarded as are, 's will be read as is.

#### 4. Timer:

A time consumption record will be displayed after each process ends up, so

that you could know how much time this section costs.

#### 5. Exception Detection:

While running program, we will face this situation usually: There is something wrong with program and the cmd will stop working as dead. Fortunately, the try-catch method in main function resolves this problem, the program will continue to run even if you enter wrong file directory, simultaneously while loop will ask you to enter again.

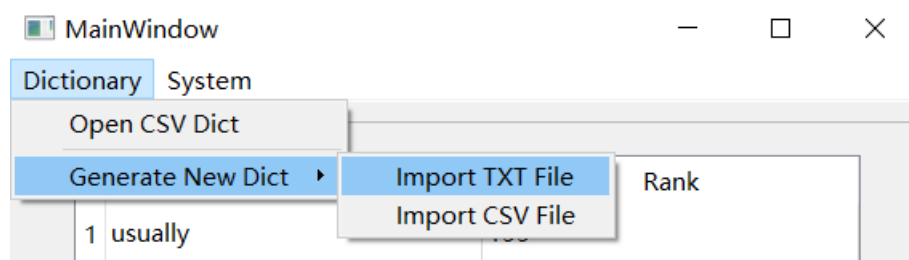
## GUI Version:

### User Manual:

1. Open "GUI\_Version" file and run "TextSearch\_GUI.exe".

iconengines	2020/05/20 22:18	文件夹	
imageformats	2020/05/20 22:18	文件夹	
platforms	2020/05/20 22:18	文件夹	
styles	2020/05/20 22:18	文件夹	
translations	2020/05/20 22:18	文件夹	
D3Dcompiler_47.dll	2014/03/11 18:54	应用程序扩展	3,386 KB
libEGL.dll	2018/12/03 19:29	应用程序扩展	16 KB
libGLSV2.dll	2018/12/03 19:29	应用程序扩展	2,722 KB
newsDict	2020/05/24 20:29	Microsoft Excel ...	146 KB
opengl32sw.dll	2016/06/14 21:08	应用程序扩展	15,621 KB
Qt5Core.dll	2020/05/20 22:18	应用程序扩展	4,967 KB
Qt5Gui.dll	2018/12/03 19:35	应用程序扩展	5,213 KB
Qt5Svg.dll	2018/12/03 22:19	应用程序扩展	258 KB
Qt5Widgets.dll	2018/12/03 19:40	应用程序扩展	4,420 KB
TextSearch_GUI	2020/05/23 1:23	应用程序	127 KB

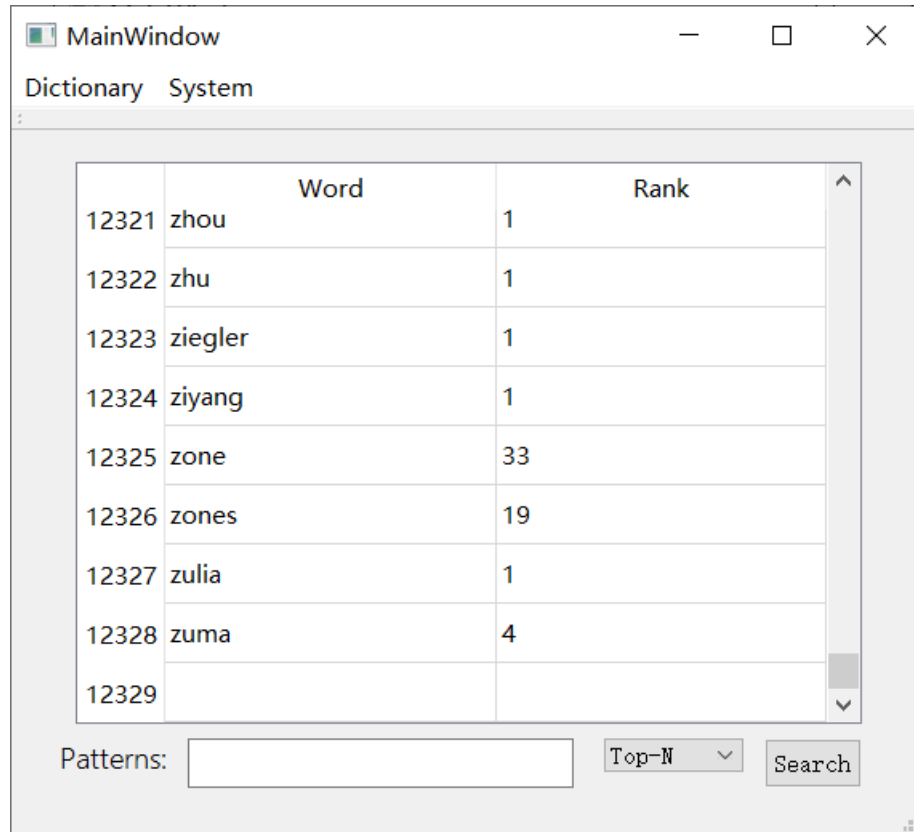
2. Import TXT or CSV file to generate a new dictionary.



3. **Attention: please import the same file type as you select. And all the file directories must be English!**

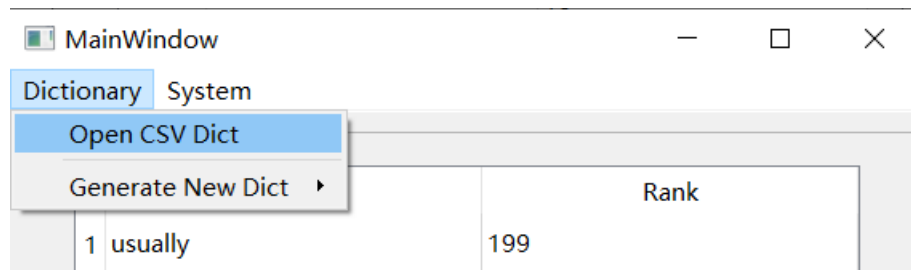


- After importing, you will see a display of generated **newDict** here. And next time, you can use your generated dictionary by importing **newDict** directly.

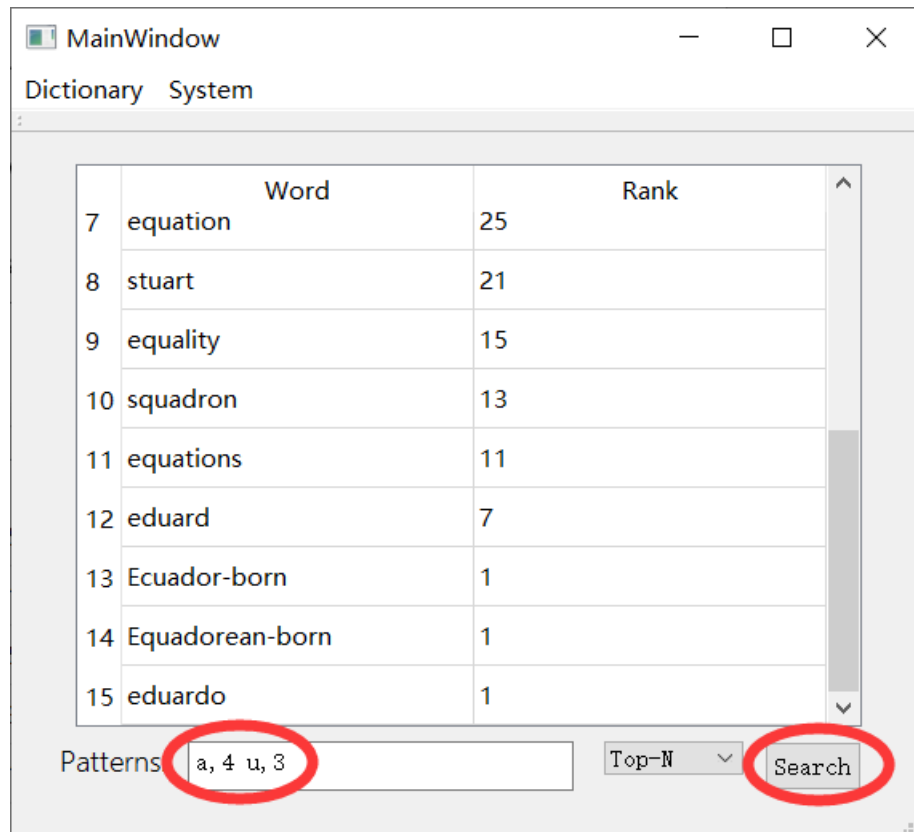


iconengines	2020/05/20 22:18	文件夹	
imageformats	2020/05/20 22:18	文件夹	
platforms	2020/05/20 22:18	文件夹	
styles	2020/05/20 22:18	文件夹	
translations	2020/05/20 22:18	文件夹	
D3Dcompiler_47.dll	2014/03/11 18:54	应用程序扩展	3,386 KB
libEGL.dll	2018/12/03 19:29	应用程序扩展	16 KB
libGLESV2.dll	2018/12/03 19:29	应用程序扩展	2,722 KB
<b>newsDict</b>	2020/05/31 13:18	Microsoft Excel ...	146 KB
opengl32sw.dll	2016/06/14 21:08	应用程序扩展	15,621 KB
Qt5Core.dll	2020/05/20 22:18	应用程序扩展	4,967 KB
Qt5Gui.dll	2018/12/03 19:35	应用程序扩展	5,213 KB
Qt5Svg.dll	2018/12/03 22:19	应用程序扩展	258 KB
Qt5Widgets.dll	2018/12/03 19:40	应用程序扩展	4,420 KB
TextSearch_GUI	2020/05/23 1:23	应用程序	127 KB

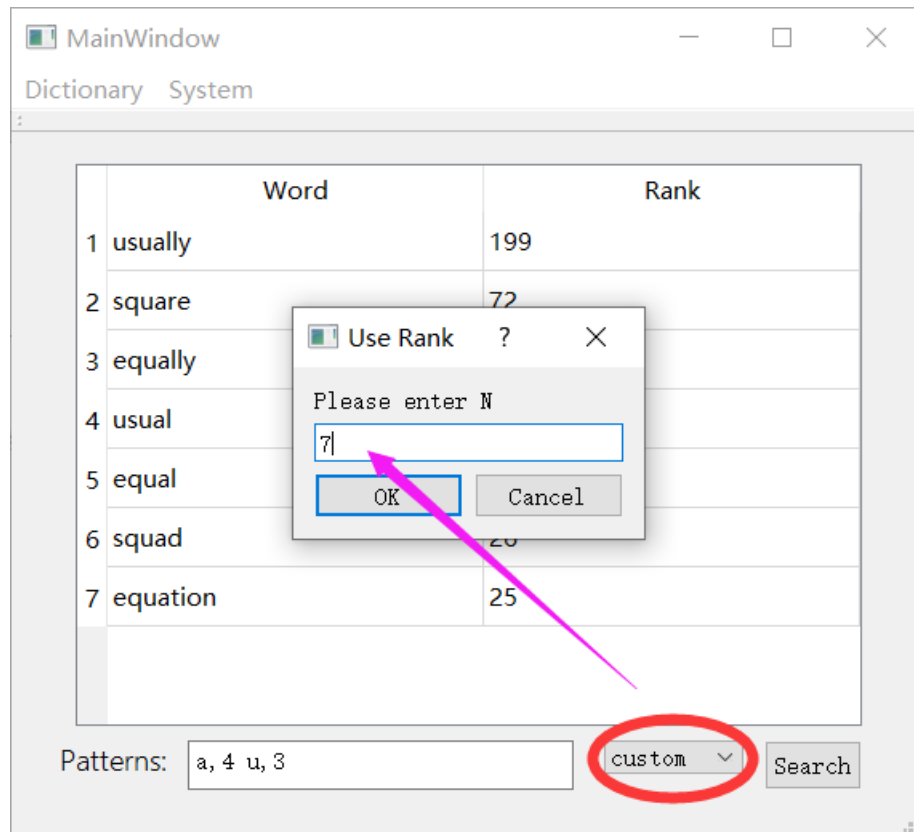
5. You can also browse it through “Open CSV Dict” function.



6. Search: you can enter a query string in a fixed format. For each pattern, please use character and integer number to represent which character in which position, use a comma to split it. Patterns are separated by space. For example, if you want to find the words like **\*\*ua\*\*\*** (third character is u, fourth character is a), you need write **u,3 a,4** (regardless of order between 2 patterns) in the text field, and click search button.



7. Top-N rank selection: Before searching, you can select to display the Top N results of all the searched words. “Custom” means you can determine any number N by yourself.

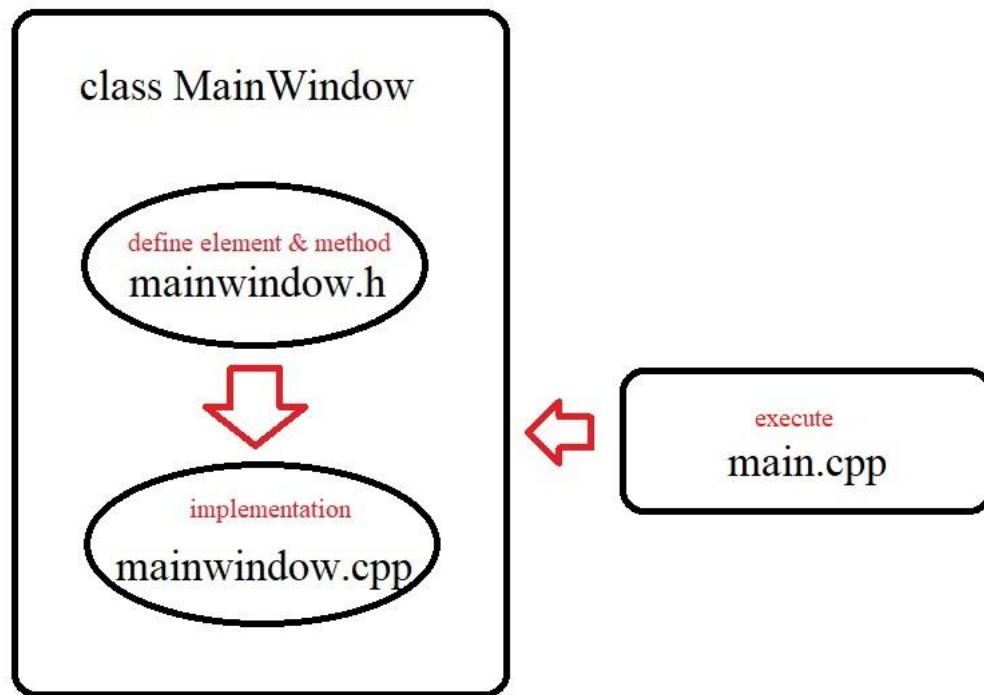


8. Exit: select "Exit" to end up the program.



9. Tips: if your query operation is improper, the display window will be cleared. But don't worry, if you have imported data file and you still don't end up this program, the content of **c\_dict** is still alive, which means you just need to do your query operation again in a correct way.

## Object Diagram:



## Data Structure with Adaptation:

Class Name/ main()	Element /function name	Modifier	Element type	Function return type	Function Parameter	Adaptation
class MainWin dow	UI function					
	MainWin dow()	public	/	/	Qwidge t *parent	Constructor
	~MainW indow()	public	/	/	/	Destructor
	displayC SVFile()	private	/	void	void	Slot: display the content

						of <b>newDict.csv</b>
<b>openFile</b> ( )	private	/	void	string suff		Slot: open file with correspondin g suffix (If suff is csv, then open csv file)
<b>on_actio</b> <b>nOpen_t</b> <b>riggered</b> ( )	private	/	void	void		Slot: Execute openFile("csv ")
<b>on_actio</b> <b>nExit_tri</b> <b>ggered</b> ( )	private	/	void	void		Slot: End up the program
<b>on_actio</b> <b>nImport</b> <b>_TXT_fi</b> <b>le_trigge</b> <b>red</b> ( )	private	/	void	void		Slot: read TXT file into the new dictionary and display the newDict
<b>on_actio</b> <b>nImport</b> <b>_CSV_F</b> <b>ile_trigg</b> <b>ered</b> ( )	private	/	void	void		Slot: read CSV file into the new dictionary and display the newDict
<b>displayD</b> <b>ict</b> ( )	private	/	void	unsigne d int topN		Slot: display the final result of searched words
<b>On_push</b> <b>Button_c</b>	private	/	void	void		Slot: confirm to search

	licked()					
	ui	private	Ui:Main Window*	/	/	UI pointer, can be used to point to any components of the UI
Original class completion						
	/	private	struct Word	/	/	/
	display	private	vector< Word>	/	/	/
	c_dict	private	map<string, int>	/	/	Use map data structure: dictionary will be ordered by alphabet of key
	splitStr()	public	/	string*	string str, string token	/
	splitQuery()	public	/	vector< string>	string str, string token	Used for separating multi-word string like query string line
	modifyStr()	public	/	void	string &str	Dealing with problems of unsigned integer: replace -1 with

						UINT_MAX
	fill_completions() ( )	public	/	void	string fd	/
	importCSV() SV()	public	/	void	string fd	/
	findWord() d()	public	/	map<string,int> >	string str	According to query string (more than one patterns) to search correct words
	rankSort() ( )	public	/	int	map<string,int> > hash	/
	fileWrite() ( )	public	/	void	void	/
/	main() ( )	/	/	int	void	Execute program and show the main window

### Core Algorithm:

Similar to Console Version, but int storage part, I replace hash\_map with map<string,int> because map uses Red Black Tree to sort key in ASCII order automatically. The complexity of N-word insertion is  $O(N\log N)$ , still less than sort  $O(0.01N^2)$ , so the overall complexity is:

$$N*O(1) [\text{read file}] + O(N\log N) [\text{insert key}] + N*O(1) [\text{write dictionary}] + O(N) [\text{search}] + O(0.01N^2) [\text{sort}] \sim O(0.01N^2)$$

## System Feature:

### 1. GUI:

GUI means Graphical User Interface, it is easier for user to use program (without using command). All the function can be implemented by mouse event and key board event, and it looks nice as well.

### 2. ASCII order dictionary:

Most normal dictionaries are in ascii order. Compared to console version, this version has implemented this feature at a very low cost of time.

### 3. Dictionary display:

My program implements open file function to let you browse your dictionary, and at importing part, the real-time **newDict** will be displayed when you import a new dictionary to combine with the old one.

### 4. Top N rank:

This function is from mid-term project, users can choose to display results with N-top rank or without N-top rank, or in custom. However, if the total number is less than N, it will display all the results. If there are no results, the program will mention you with a message dialog.

### 5. Exception Detection:

A message dialog will show up to help you while you are doing an incorrect operation.

## 4. Need Improvement

**1:** The input file type of Console Version is fixed (1 for TXT and 1 for CSV), and if your directory is incorrect, you need to reinput both 2 directories again.

**2:** The Console Version probably cannot implement dictionary combination function more than once, but the GUI Version can.

**3:** Both 2 versions are not perfect in word pre-processing, there are still some meaningless words remaining in the dictionary.