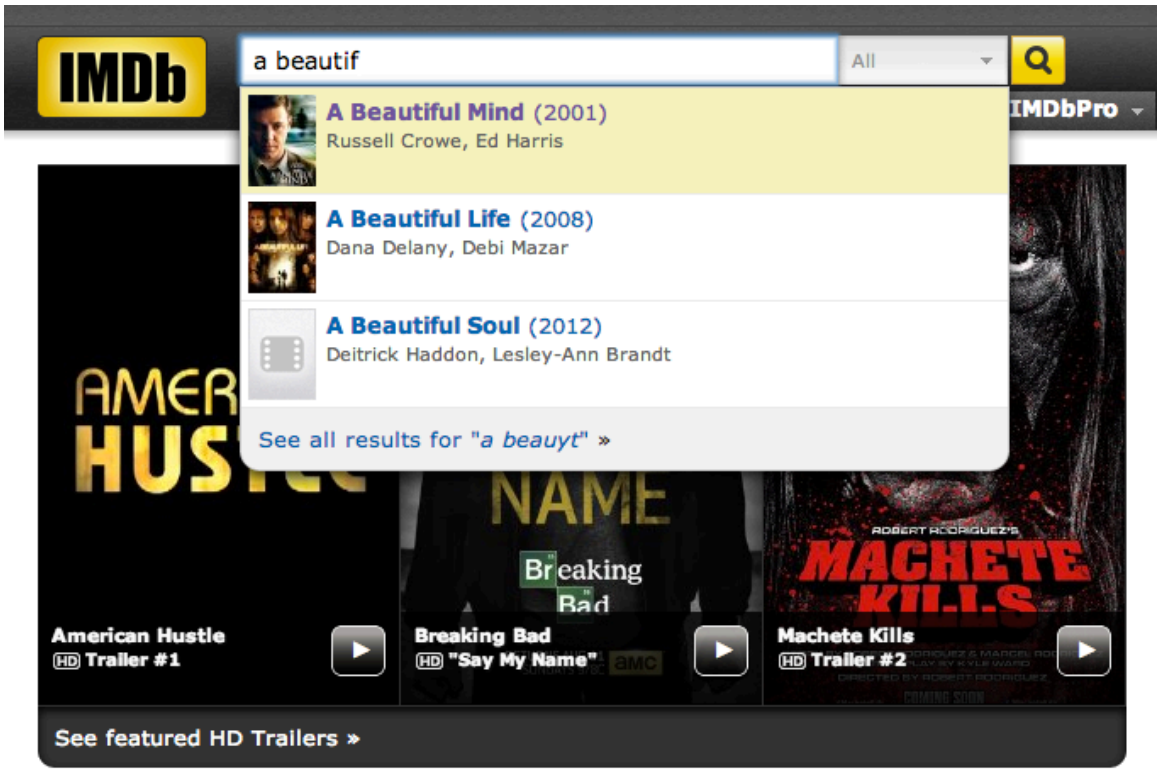
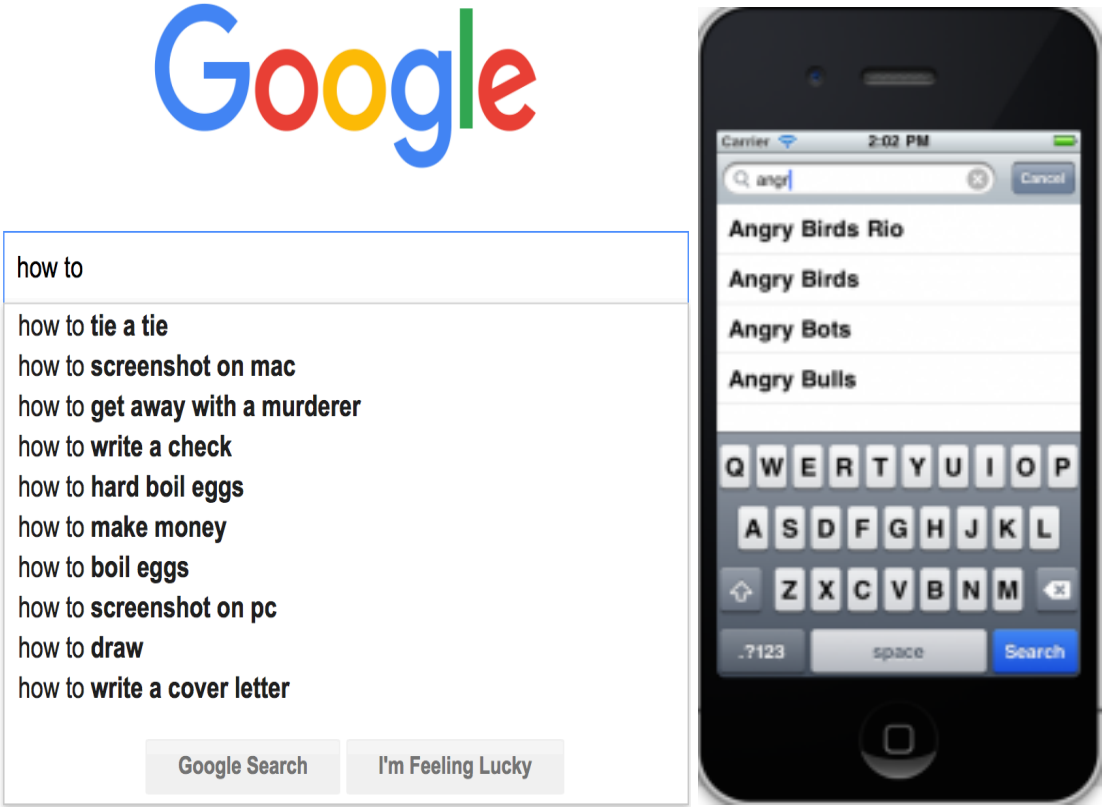
CS216: Introduction to Software Engineering Techniques (Fall, 2018)
Programming Assignment 2
(100 points)
Today's Date: Friday, October 26
Due Date: Sunday, November 11

Problem Statement

Write a C++ program to implement ***autocomplete me*** for a given set of *N terms*, where a term is a query string and an associated nonnegative weight. That is, given a prefix, find all queries that start with the given prefix, in descending order of weight.
Autocomplete is pervasive in modern applications. As the user types, the program predicts the complete *query* (typically a word or phrase) that the user intends to type. Autocomplete is most effective when there are a limited number of likely queries. For example, the Internet Movie Database uses it to display the names of movies as the user types; search engines use it to display suggestions as the user enters web search queries; cell phones use it to speed up text input.





In these examples, the application predicts how likely it is that the user is typing each query and presents to the user a list of the top-matching queries, in descending order of weight. These weights are determined by historical data, such as box office revenue for movies, frequencies of search queries from other Google users, or the typing history of a cell phone user. For the purposes of this assignment, you will have access to a set of all possible queries and associated weights (and these queries and weights will not change).

In this project assignment, you will implement "*Autocomplete Me*" by *sorting* the terms by query string first; then use *binary search* to find all matched terms whose query strings starting with a given prefix; and then generate the list of output by *sorting* the matched terms, in descending order by weight.

`Part 1: autocomplete term.` Define a class, named **`Term`** that represents an autocomplete term: (a query string and an associated integer weight). Then you need to provide three member functions, which support comparing terms by three different sorting criteria respectively: **(1) in lexicographic order by query string; (2) in descending order by weight;** and **(3) in lexicographic order by query string but using only the first *r* characters.** The last order may seem a bit odd, but you will use it in ***Part 2*** to find all query strings that start with a given prefix (of length *r*). The first sorting criterion is defined through operator overloading for operator "**`<`**". Other two sorting criteria are both defined as **`static`** member functions, and each of them returns an integer value: **`1, 0 or -1`**. When comparing two terms, if the first term and the second term are in the right order according to the sorting

criterion, it returns **1**; if they are in the opposite order according to the sorting criterion, it returns **-1**; if they are the same, it returns **0**.

You should start with the following declaration of the class Term:

```
class Term
{
   public:
     // default constructor
      Term();

     // initialize with the given query string and weight
      Term(string query, long weight);

     // compare two terms in descending order by weight
     // if t1 and t2 are in descending order by weight, return 1
     // if they are of the same weight, return 0;
     // otherwise, return -1
      static int compareByWeight(Term t1, Term t2);

     // compares two terms in lexicographic order but using only
     // the first r characters of each query
     // if the first r characters of t1 and t2 are in lexicographic order, return 1
     // if they are of the same r characters, return 0;
     // otherwise, return -1
      static int compareByPrefix(Term t1, Term t2, int r);

     // define the operator "<" for Term class (as friend function)
      friend bool operator<(Term t1, Term t2);

     // define the operator "<<" for Term class (as friend function)
     // so that it can send the Term object directly to cout, in the following format:
     // the weight followed by a tab key, then followed by the query
      friend ostream& operator<<(ostream& out, const Term& t);

     // assign "friendship" to the class named Autocomplete
     // so that Autocomplete class can directly access the private data members
     //                                of Term class. Not the other way around.
      friend class Autocomplete;

   private:
      string query;
      long weight;
};
```

Part 2: autocomplete. Define a class, named **Autocomplete** that provides autocomplete functionality for a given set of **Term** objects. For this assignment, use **the binary search** to efficiently find all query strings that start with a given prefix (they are the matched terms), and then sort the matched terms in descending order by weight. To do so, you need to **(1) sort the terms in lexicographic order by query first,** then **(2) use binary search to search for a given prefix, from a sorted sequence.** Since for a given prefix, the matched terms can be more than one, your program needs to get the first and the last such terms in the sequence, you may implement a member function named `search()` as follows:

```
void search(string prefix, int& firstIndex, int& lastIndex);
```

`firstIndex` and `lastIndex` are reference parameters, and they represent the index of the first query that matches the given prefix and the index of the last query that matches the given prefix respectively; or `-1` if no such prefix can be found.

For example, assume the following shows part of the sorted sequence of term objects (note that part of the first column (shown in orange color below) represents the index number of each Term object in the sequence named **terms**, which is the private data member of **Autocomplete** class):

```
……
terms[956] 2809    Wackenhut
terms[957] 219812  Wal-Mart Stores
terms[958] 24623   Walgreen
terms[959] 1693    Wallace Computer Svcs.
terms[960] 25269   Walt Disney
terms[961] 1922    Walter Industries
terms[962] 17692   Washington Mutual
……
```

Assume that the given **prefix** is "**Wal**", after calling **binary_search()** inside **search()** function mentioned above, it should return one of the index numbers of whose query matches "**Wal**". Then from this index number, your program should be able to decide the values for **firstIndex** and **lastIndex**, in this case, **firstIndex** should hold index number 957, and **lastIndex** should hold index number 961 after calling **search()** function. Then, **(3) your program needs to sort the matched terms** (from index **firstIndex** to **lastIndex**) **in descending order by weight.** (note that using each index number, it can directly access the corresponding **Term** object)

You need to use the following declaration of the class **Autocomplete**:

```
class Autocomplete
{
   public:
    // insert the newterm to the sequence
    // note that a SortingList<Term> object, named terms,
    //                              represents the sequence
    void insert(Term newterm);
```

```
    // sort all the terms by query in lexicographical order
    // note that this function needs to be called before applying binary search
    void sort();

    // return the index number of the term whose query
    // prefix-matches the given prefix, using binary search algorithm
    // Note that binary search can only be applied to sorted sequence
    // Note that you may want a binary search helper function
    int binary_search(string prefix);

    // first: the index of the first query whose prefix matches
    //         the search key, or -1 if no such key
    // last: the index of the last query whose prefix matches
    //        the search key, or -1 if no such key
    // key: the given prefix to match
    // hints: you can call binary_search() first to find one matched index number,
    //       say hit, then look up and down from hit, to find first and last respectively
    void search(string key, int& first, int& last);

    // return all terms whose queries match the given prefix,
    // in descending order by weight
    SortingList<Term> allMatches(string prefix);

    // display all the terms in the sequence
    void print();

    // other member functions you may need…

  private:
    SortingList<Term> terms;
};
```

The performance of autocomplete functionality is critical in many systems. For example, consider a search engine, which runs an *autocomplete me* application on a server farm. According to one study, the application has only about 50ms to return a list of suggestions for it to be useful to the user. Moreover, in principle, it must perform this computation for every keystroke typed into the search bar and for every user! Hence the decision of which data structure should be chosen to store the sequence of term objects in Autocomplete class is very important. Since sorting operation (either in lexicographic order by query and/or in descending order by weight) is needed and essential to the sequence of Term objects in this project, we think about the class named SortingList , which should provide a few member functions using different sorting algorithms, and it should also be flexible enough to apply to

different types of data items, such as integers, Building objects, Term objects and so on. This leads to designing the SortingList class to provide the following two features:

- Provide the function pointer to each sorting member function so that the sorting criteria (defined by different function) can be passed in as a parameter;
- Make the SortingList class into a template class, so that it can be used to store any type of data members or objects.

You can download the definition of the SortingList class from the following link (note that it is almost complete, however it still needs a little help from you) and spend some time understanding how to implement the two major features above.

http://www.cs.uky.edu/~yipike/CS216/PA2_SortingList.zip

Input format. You can download a number of sample input files for testing. Each file consists of *N* pairs of nonnegative weights and query strings. There is one pair per line, with the weight and query separated by a tab. A weight can be any integer between 0 and (2^63 – 1). A query string can be an arbitrary sequence of Unicode characters, including spaces (but not newlines).

- Starting with smaller testing file first:
  - The file http://www.cs.uky.edu/~yipike/CS216/fortune.txt contains Fortune 1000 American largest companies, ranked by revenues. The weight equals to the company's revenue.
- When you think it is ready to challenge the sorting algorithm you choose for your project, you can test running your program with the following file:
  - The file http://www.cs.uky.edu/~yipike/CS216/actors.txt contains 100,000 famous American actors and actresses, with weights equal to their revenue.
  - The file http://www.cs.uky.edu/~yipike/CS216/imdb.txt contains over 80,000 movies from IMDB web site, with weights equal to the number of votes.

When you run your program, it should take the name of an input file and an integer **k** as command-line arguments. It reads the data from the file; then it repeatedly asks the user to enter autocomplete queries, and prints out the top **k** matching terms in descending order by weight, until the user enters "exit" to quit. If there are only *j* matching terms, where (`0 < j < k`), then it prints out the top *j* matching terms in descending order by weight.

Here are a few sample executions:

```
$ ./CS216PA2
Usage: ./CS216PA3 <filename>  number

$ ./CS216PA2  Fortune.txt  6
```

Cannot open the file named Fortune.txt

$ ./CS216PA2 fortune.txt -1
The number of matching terms needs to be a positive number!

$ ./CS216PA2 fortune.txt 6
Please input the search query(type "exit" to quit):
Am↵
62402 American Intl. Group
61257 American Electric Power
22582 American Express
15823 AmerisourceBergen
13413 Amerada Hess
7465 American Standard
Please input the search query(type "exit" to quit):
Wal↵
219812 Wal-Mart Stores
25269 Walt Disney
24623 Walgreen
1922 Walter Industries
1693 Wallace Computer Svcs.
Please input the search query(type "exit" to quit):
National↵
9093 National City Corp.
2113 National Semiconductor
2100 National Fuel Gas
1748 National Oilwell
1548 National Commerce Finan.
1388 National Rural Utilities Coop.
Please input the search query(type "exit" to quit):
National C↵
9093 National City Corp.
1548 National Commerce Finan.
Please input the search query(type "exit" to quit):
Micro↵
25296 Microsoft
4516 Micron Technology
Please input the search query(type "exit" to quit):
Mic↵

```
25296    Microsoft
4516     Micron Technology
2531     Michaels Stores
Please input the search query(type "exit" to quit):
Mi↵
31502    Mirant
25296    Microsoft
16079    Minnesota Mining & Mfg.
4516     Micron Technology
2531     Michaels Stores
1801     Minnesota Life
Please input the search query(type "exit" to quit):
M↵
47716    Merck
43727    Morgan Stanley
42010    McKesson
38793    Merrill Lynch
35041    Marathon Oil
31928    MetLife
Please input the search query(type "exit" to quit):
exit↵

$ ./CS216PA2   fortune.txt actors.txt   5
Usage: ./CS216PA2 <filename>  number

$ ./CS216PA2   actors.txt   5
Please input the search query(type "exit" to quit):
Emma↵
2580192375  Emma Watson (II)
2171702719  Emma Thompson (I)
1209880667  Emma Stone (III)
623476184   Emma Raimi
595576013   Emma Roberts (II)
Please input the search query(type "exit" to quit):
Tom H↵
4386200484  Tom Hanks
1351430588  Tom Hollander
1110051890  Tom Hiddleston
1047104583  Tom Hardy (I)
```

493598615 Tom Hines (I)
Please input the search query(type "exit" to quit):
Charles↵
2035936358 Charles Adler (I)
1178796823 Charles Napier
1088884483 Charles Pendelton
1056408407 Charles Durning
839990239 Charles S. Dutton
Please input the search query(type "exit" to quit):
Harr↵
3564727945 Harrison Ford (I)
1731635944 Harry Dean Stanton
1708880015 Harry Melling
1540402196 Harry Taylor (I)
1461858964 Harry Fielder (I)
Please input the search query(type "exit" to quit):
Ha rr
No matched query!
Please input the search query(type "exit" to quit):
Zoe↵
2065969383 Zoe Saldana
422783777 Zoe Leader
366160217 Zoe Bell (I)
275520247 Zoe Lister Jones
257704099 Zoe Chernov
Please input the search query(type "exit" to quit):
Zoe S↵
2065969383 Zoe Saldana
78058929 Zoe Simek
60923326 Zoe Salmon (I)
35707327 Zoe Sharpe (II)
34900024 Zoe Sallis
Please input the search query(type "exit" to quit):
Zoe Sa↵
2065969383 Zoe Saldana
60923326 Zoe Salmon (I)
34900024 Zoe Sallis
Please input the search query(type "exit" to quit):
Yi Pike↵

No matched query!
Please input the search query(type "exit" to quit):
exit↵

$ ./CS216PA2 imdb.txt 7
Please input the search query(type "exit" to quit):
Star Wa↵
675029 Star Wars (1977)
605021 Star Wars: Episode V - The Empire Strikes Back (1980)
480393 Star Wars: Episode VI - Return of the Jedi (1983)
401231 Star Wars: Episode I - The Phantom Menace (1999)
387721 Star Wars: Episode III - Revenge of the Sith (2005)
343420 Star Wars: Episode II - Attack of the Clones (2002)
33212 Star Wars: The Clone Wars (2008)
Please input the search query(type "exit" to quit):
Stars↵
189541 Starship Troopers (1997)
106850 Starsky & Hutch (2004)
17297 Starship Troopers 3: Marauder (2008)
16284 Starship Troopers 2: Hero of the Federation (2004) (V)
7250 Starship Troopers: Invasion (2012)
998 Stars in My Crown (1950)
693 Starstruck (1982)
Please input the search query(type "exit" to quit):
Harry Potter↵
384098 Harry Potter and the Deathly Hallows: Part 2 (2011)
347615 Harry Potter and the Sorcerer's Stone (2001)
305471 Harry Potter and the Chamber of Secrets (2002)
304275 Harry Potter and the Goblet of Fire (2005)
298817 Harry Potter and the Prisoner of Azkaban (2004)
279603 Harry Potter and the Order of the Phoenix (2007)
253710 Harry Potter and the Deathly Hallows: Part 1 (2010)
Please input the search query(type "exit" to quit):
Harry K↵
116 Harry Knuckles and the Treasure of the Aztec Mummy (1999) (V)
73 Harry Knuckles and the Pearl Necklace (2004) (V)
Please input the search query(type "exit" to quit):
Gone↵

```
183587  Gone in Sixty Seconds (2000)
171036  Gone with the Wind (1939)
158166  Gone Baby Gone (2007)
28906   Gone (2012/I)
7328    Gone Fishin' (1997)
5457    Gone in 60 Seconds (1974)
3127    Gone (2006/VI)
Please input the search query(type "exit" to quit):
Gone w↵
171036  Gone with the Wind (1939)
230     Gone with the West (1975)
Please input the search query(type "exit" to quit):
U↵
467779  Up (2009)
233936  Up in the Air (2009/I)
219497  Unforgiven (1992)
212352  Unbreakable (2000)
174197  Underworld (2003)
171818  Unknown (2011/I)
130209  Underworld: Evolution (2006)
Please input the search query(type "exit" to quit):
Thanks↵
12675   Thanks for Sharing (2012)
2236    ThanksKilling (2009)
290     ThanksKilling 3 (2012)
241     Thanks, Smokey! (2011)
136     Thanks to Gravity (2006)
118     Thanks for the Memory (1938/II)
45      Thanks (2011)
Please input the search query(type "exit" to quit):
Thanksgiving↵
No matched query!
Please input the search query(type "exit" to quit):
exit↵
```

Note the blue part is what you type from the keyboard, ↵ represents the "return" key.

Notes on implementation:

- You are required to use the declaration of `Term` class described in this document,

and binary search algorithm to find all query strings that start with a given prefix.

- You are also required to use the declaration of Autocomplete class described in this document, however, you are allowed to add other member functions if you want to. Do not modify the declaration of **SortingList** class you download from the instructor's web site. If you use other data structure to store the Term objects, you will get 10 points deduction. While making your decisions about data structure and algorithms, it should be based on the application: the size of the data items and the mix of operations you will perform on those data items.
- Your program should not prompt for the user input at the beginning, and it should just start reading the data from the input file before asking the user for input.
- Since there are a few sorting algorithms implemented as member functions in the SortingList class, it is your choice to decide which function to call to fit the sorting operations you apply to the sequence of data items. Which one is the most efficient one (to sort *a large number* of Term objects fast enough)? Which one allows me to pass in sorting criteria (defined by my own function) as a parameter so that I can sort the matched Term objects *in decreasing order by weight*?

Part 3 makefile. Create a make file (name it makefile) to build your C++ program from **Part 1** and **Part 2**. In the makefile, name the executable program **CS216PA2** (NO .exe).

For all files:

- Test on your Virtual Machine. If (for whatever reason) it doesn't work there, you lose points. If your program cannot pass compilation, then you get zero credit.
- Your program is unzipped then compiled/linked by executing the command:

  $ make

  therefore:
  - All needed files are supplied and assumed to be on the current directory
  - No hardcoded directory names in the .h file names
  - Create your makefile (file name makefile)
  - In the makefile, name the created program CS216PA2
- Each class has a separate specification file (classname.h) and implementation file (classname.cpp) unless two classes have very closed relationship
- Comments for every file you write, lay out your source file in a readable style
- Name your zip file Project2.zip and zip using the following command:

```
$ zip Project2.zip makefile *.cpp *.h *.txt
                    (or list the individual files)
```

  NOT

```
$zip Project2.zip /PA2/*
```

  Zip from the directory where the files are, not a parent directory.

Extra Part: (Bonus 16 points)

In order to gain Bonus points for Project 2, you have to make sure that your program has already generated the correct output according to the Problem Statement in this document. Just as your own experience with *autocomplete me* feature, such as Google search engine or

imdb movie searching:

- The prefix matching should be **case-insensitive**. For example, with the input file of actors' information, if the user types "toM", or "tOm", or "TOM", your program should be able to match the actors whose first name is "Tom". (10 points)
- If the user-input starts with quite a few blank spaces or tab key, your program should ignore them and start to match the prefix from the first non-space character. (3 points)
- If the user-input contains spaces or tab key in between two non-space characters, your program should consider it as a single space. For example, if the user types " tom H", your program should use "tom H" as the prefix to match. (3 points)

For the Extra Part of this project, based on what you have done for the Project 2, challenge your program with the above three features, correctly implement each feature helps you gain at maximum of (10+3+3 = 16) bonus points. (Note that you are allowed to use any source code the instructor provided during this semester).

Here is a sample execution of your program with extra features from Bonus Part:

```
$ ./CS216PA2  imdb.txt  3↵
Please input the search query(type "exit" to quit):
      the      D↵
1245679 The Dark Knight (2008)
831456  The Dark Knight Rises (2012)
647808  The Departed (2006)
Please input the search query(type "exit" to quit):
   amEr↵
643948  American Beauty (1999)
596022  American History X (1998)
283491  American Psycho (2000)
Please input the search query(type "exit" to quit):
  Zoo↵
154239  Zoolander (2001)
36625   Zookeeper (2011)
12506   Zoom (2006)
Please input the search query(type "exit" to quit):
    harry    potter  ANd↵
384098  Harry Potter and the Deathly Hallows: Part 2 (2011)
347615  Harry Potter and the Sorcerer's Stone (2001)
305471  Harry Potter and the Chamber of Secrets (2002)
Please input the search query(type "exit" to quit):
    winter↵
94960   Winter's Bone (2010)
```

```
23255    Winter's Tale (2014)
6662     Winter Passing (2005)
Please input the search query(type "exit" to quit):
   winter    p↵
6662     Winter Passing (2005)
812      Winter People (1989)
Please input the search query(type "exit" to quit):
   winter    B↵
999      Winter Break (2003)
Please input the search query(type "exit" to quit):
    winter's↵
94960    Winter's Bone (2010)
23255    Winter's Tale (2014)
48       Winter's End (2005)
Please input the search query(type "exit" to quit):
Yi Pike↵
No matched query!
Please input the search query(type "exit" to quit):
exit↵
```

Note the blue part is what you type from the keyboard, ↵ represents the "return" key. You need to run your program from your Virtual Machine instead of your home computer.

Submission:

Open the link to Canvas LMS (https://uk.instructure.com/), and login to your account using your linkblue user id and password. Please submit your file (Project2.zip) through the submission link for "Project 2". Note that only one file is allowed to upload and it should be your zip file. It is a good idea to check that your file is already uploaded successfully. If not, go back and submit again.

The grading sheet is on the next page.

Grading Sheet for Project Assignment 2

Total: 100 points (Bonus 16 points)

These are example errors. There are other ways to lose points. C++ programs must compile in order to be graded	Points	Deducted Points
C++ Program	80	
Provide the command line argument check while running your program	5	
Check whether the file can be open successful	5	
Correctly read pairs of (weight, query) from the input file and store into an Autocomplete object	5	
Correctly and efficiently sort all Term objects in lexicographical order by query (if your program cannot pass the test from imdb.txt as the input file, you lose 7 points for this category)	10	
Repeatedly allow the user to type the search query until the user enters "exit" to quit	2	
Generate N terms matched with the search query by prefix, where N is the smaller value between the second command line argument and the total number of matcher terms)	3	
Correctly display the matched terms in the decreasing order by weight	8	
Provide the function which uses binary search to find the matched term	6	
Correctly provide the complete definition of Term class	10	
Correctly provide the complete definition of Autocomplete class	15	
Complete the definition of SortingList class as a template class	6	
Provide separate .cpp files and header files for class(es).	3	
Quit the program correctly	2	
makefile	7	
Generate the executable file correctly	2	
The dependency lines correctly track of files' dependencies	2	
The command lines correctly create targets	2	
Allow "make clean" to clean up the mess	1	
Miscellaneous errors, or did not follow the directions in the program assignment, examples: (-10 if your program uses other data structure than SortingList class to store the sequence of Term objects in the Autocomplete class)	5	
-2 did not zip file or used tar or gzip instead	2	
-2 created subdirectory when unzipped	2	
-1 wrong names	1	

There may be other errors in the is category		
Style Lay out your program in a readable style C++ program comments -6 non -4 only a few	8 2 6	
Bonus Extra Part Your program also provides the following feature: The prefix matching should be **case-insensitive** The prefix matching should start from the first non-space character(ignore space or tab at the beginning) The prefix matching should only keep one space between the non-space characters from user-input	16 10 3 3	
Your Score for Project 2		