```cpp
///////////////////////////////////////////////////////////////////////////////
//
// Authors:          David Hawkins
// Email:            david.james@hawkinsonline.us
// Label:            P02
// Title:            Assignment 5 - Processing in Linear Time
// Course:           3013
// Semester:         Spring 2020
//
// Description:
//      This program implements a linked list class that holds values read in
//      from a JSON file. This program also times how long it takes to load
//      the data from the JSON file into the linked list. Furthermore, it
//      incorporates getch functionality that reads input from the keyboard
//      and searches for the user's entry in the linked list (this is timed
//      as well).
//
// Files:
//      main.cpp             : Driver program
//      json.hpp             : File containing Json implementation
//      JsonFacade.hpp       : File containing simplified Json implementation
//      mygetch.hpp          : File containing getch implementation
//      Timer.hpp            : File used to implement timer
//
///////////////////////////////////////////////////////////////////////////////

#include <iostream>
#include "JsonFacade.hpp"
#include <time.h>
#include <chrono>
#include "Timer.hpp"
#include "mygetch.hpp"

using namespace std;

/**
 * wordNode
 *
 * @description:
 *      This is the struct that defines the nodes that are contained in the
 *      linked list class below.
 *
 * @methods:
 *      constructors:
 *          wordNode(string w, string def)   : Parameterized Constructor
*/
struct wordNode{
    string word;
    string definition;
    wordNode* next;

    wordNode(string w, string def){
        word = w;
        definition = def;
        next = NULL;
    }
};

/**
 * LinkedList
 *
 * @description:
 *      This class implements a singly-linked list of wordNodes.
 *
 * @methods:
 *      constructors:
 *          LinkedList()                     : Default Constructor
 *      public:
 *          GetSize()                        : Returns size of the linked list
 *          Insert(string, string)          : Inserts a new node in list
 *          Remove()                        : Removes one node from list
 *          PrintList()                     : Prints words in the list
 *          PrintTail()                     : Prints tail at end of list
 *          Search(string, int&, string&)   : Searches for item in list
 *
*/
class LinkedList {
private:
    wordNode* head;
    wordNode* tail;
    int size;

public:
    /**
     * Public : LinkedList
     *
     * @description:
     *      Creates instances of the Linked List class.
     *
     * @param
     *      None
     *
     * @return
     *      None
    */
    LinkedList(){
        head = tail = NULL;
        size = 0;
    }

    /**
     * Public : GetSize
     *
     * @description:
     *      Returns the current size of the list.
     *
     * @param
     *      None
     *
     * @return
     *      [int]   : Size of the linked list
    */
    int GetSize(){
        return size;
    }

    /**
     * Public : Insert
     *
     * @description:
     *      Inserts a new node into the linked list with the word
     *      and definition.
     *
     * @param
     *      [string]    : word being inserted
     *      [string]    : definition being inserted
     *
     * @return
     *      None
    */
    void Insert(string w, string def){
        if (head == NULL){       //if list is empty, insert at beginning
            head = tail = new wordNode(w, def);
            size++;
        }
        else{                    //if list is not empty
            tail->next = new wordNode(w, def);
            tail = tail->next;
```

```
                size++;
            }
        }

        /**
         * Public : Remove
         *
         * @description:
         *      Removes one wordNode from the list.
         *
         * @param
         *      None
         *
         * @return
         *      [string]    : Word removed from the list
        */
        string Remove(){
            if (head != NULL){
                wordNode* temp = head;
                string data;

                data = temp->word;
                head = head->next;
                delete temp;
                temp = NULL;
                size--;

                return data;
            }
            else{
                return "Can't remove from empty list.";
            }
        }

        /**
         * Public : PrintList
         *
         * @description:
         *      Creates a string representation of the linked list that can
         *      be printed to the screen.
         *
         * @param
         *      None
         *
         * @return
         *      [string]    : String representation of the linked list
        */
        string PrintList(){
            wordNode* temp = head;
            string list;

            while (temp != NULL){
                list += temp->word + "->";
                temp = temp->next;
            }

            return list;
        }

        /**
         * Public : PrintTail
         *
         * @description:
         *      Prints the word contained in the very last wordNode in the list.
         *
         * @param
         *      None
         *
         * @return
```

```cpp
     *      [string]   : Word in the last node
    */
    string PrintTail(){
        string list;

        list += tail->word + ": " + tail->definition;

        return list;
    }

    /**
     * Public : Search
     *
     * @description:
     *      Searches the list for the item described using the .substr method
     *      from the string class. When a match is found, the number of matches
     *      is increased by one and the first ten results are compiled into
     *      one string.
     *
     * @param
     *      [string]    : Item being searched for
     *      [int&]      : Number of matches found
     *      [string&]   : First ten results found in the list
     *
     * @return
     *      None
    */
    void Search(string item, int & match, string& results){
        wordNode* temp = head;
        string test;

        while (temp != NULL){
            test = temp->word;
            if(item == test.substr(0, item.length())){  //Checks substring
                match++;                                 //If found, increment
                if (match <= 10){
                    results += temp->word + ", ";        //Then concatenate
                }

            }
            temp = temp->next;                           //Traverse
        }
    }


};

/*******************************************************
 * Function Prototypes
 ******************************************************/
string GetchInput();
void LoadList(Timer& T, LinkedList& L, JsonFacade& J, double& sec, long& mill);
void PrintResults(int& match, string& results, Timer& S);

int main(){

    Timer T;                            //Timer object for loading list
    Timer S;                            //Timer object for searching
    LinkedList L;                       //Linked list object
    double Sec;                         //num of secs to load list
    long Milli;                         //num of millisecs to load list
    int match = 0;                      //used to record number of matches
    string result;                      //records first ten results from search
    string item;                        //holds what we're searching for

    JsonFacade J("dict_w_defs.json");   //Json object holding list of words

    LoadList(T, L, J, Sec, Milli);      //Loads list & times it

    item = GetchInput();

    S.Start();
    L.Search(item, match, result);
    S.End();

    PrintResults(match, result, S);

    return 0;
}

/**
     * Public : GetchInput
     *
     * @description:
     *      Captures user input from the keyboard and concatenates characters
     *      into a string.
     *
     * @param
     *      None
     *
     * @return
     *      [string]   : String captured at the keyboard
    */
string GetchInput(){
    char k;             // holder for character being typed
    string word = "";   // var to concatenate letters to

    cout << "Type what you're searching for, then press enter: ";

    while ((k = getch()) != 13) {
        word += k;
        cout << k;
    }

    cout << endl;

    return word;
}

/**
     * Public : LoadList
     *
     * @description:
     *      Reads values from the Json object and stores them in the linked
     *      list object.
     *
     * @param
     *      [Timer&]        : Timer object used for timing
     *      [LinkedList&]   : Linked List object that is loaded into
     *      [JsonFacade&]   : Json object that is read from
     *      [double&]       : Number of seconds required to load
     *      [long&]         : Number of milliseconds required to load
     *
     * @return
     *      None
    */
void LoadList(Timer& T, LinkedList& L, JsonFacade& J, double& sec, long& mill){
    T.Start();                                //Timer starts
    for (int i = 0; i < J.getSize(); i++){   //Insert values into list
        L.Insert(J.getKey(i), J.getValue(J.getKey(i)));
    }
    T.End();                                  //Timer stops

    sec = T.Seconds();                        //Time logged
    mill = T.MilliSeconds();
}

/**
```

```cpp
     * Public : PrintResults
     *
     * @description:
     *      Prints the results from searching the linked list for the item
     *
     * @param
     *      [int&]      : Number of matches found in linked list
     *      [string&]   : Top results found in list
     *      [Timer&]    : Timer object used for timing
     *
     * @return
     *      None
    */
void PrintResults(int& match, string& results, Timer& S){
    cout << endl << match << " words found in " << S.Seconds()
        << " seconds" << endl << endl;
    cout << results << endl;
}
```