```
/*****************************************************************************
*
*    Author:           Miklos Moreno
*    Label:            P02
*    Title:            Processing in Linear Time
*    Course:           CMPS 3013
*    Semester:         Spring 2022
*    Description:
*
*       A linked list search program that stores a file with wordsin it. Then
allowes
*       the user to type in a series of character. Everytime a user
*       enters a character the program will search through the list to find all
the words
*       with a substring of the chararcter entered and returns the top ten results
plus
*       the time it took to search the list.
*
*    Files:
*         main.cpp
*         Timer.hpp
*         mygetch.hpp
*         termcolor.hpp
*         dictionary.txt
*
*    Usage:
*           main.cpp          : driver program
*           dictionary.txt    : Input file
*
*           output will be display on the console in color.
*
*****************************************************************************/
#include <iostream>
#include <time.h>
#include <chrono>
#include "Timer.hpp"
#include "my_getch.hpp"
#include <string>
#include <vector>
#include <fstream>
#include "termcolor.hpp"

using namespace std;
/*
    Struct Name: wordNode

    Description:
        - A node that holds a string word and a pointer next.

    Public Methods:
        - string word
        - wordNode* Next
    Private Methods:
        - None

    Usage:
        - Creates node for a Linked List.

 */

struct wordNode
{
    wordNode *Next;
    string word;

    wordNode()
    {
        Next = NULL;
        word = "";
    }
};

/*
    Class Name: LinkedList

    Description:
        - Implements Linked List consisting of wordNodes.
        - Head and Tail wordNode pointers.
        - Size variable.

    Public Methods:
        - LinkedList()              :default constructor
        - int Get_Size()
        - void Insert_Data(wordNode* entry)
        - vector<string> Find(string typed)
        - void Print()

    Private Methods:
        - void

    Usage:
        - Load linked list of wordNodes.
        - Print the results
 */

class LinkedList
{
protected:
    wordNode *Head;
    wordNode *Tail;
    int Size;

public:
    /*
    Constructor : LinkedList
    Description:

        - Initialize with default values.

    Params:
        - None

    Returns:
        - None
 */
LinkedList()
{
    Head = NULL;
    Tail = NULL;
    Size = 0;
}
    /*
    Public : Get_Size()

    Description:
        - returns the size of the Linked List.

    Params:
        - None

    Returns:
        - int
 */
int Get_Size()
{
    return Size;
}

/*
   Public : Insert_Data(wordNode* entry)

   Description:
        - receives a wordNode.
        - insert the node.

   Params:
       - wordNode* entry

   Returns:
        - void
*/
    void Insert_Data(wordNode *entry)
{
    if (!Head)
    {
        Head = Tail = entry;
    }
    else
    {
        Tail->Next = entry;
        Tail = entry;
    }

    Size++;
}
    /*
   Public : Print()

   Description:
        - prints the results of the Linked List.

   Params:
        - None/Member Variables

   Returns:
        - void
*/
void Print()
{
    wordNode *Current = Head;

    while (Current)                         // Standard traversal
    {
        cout << Current->word;              // Print name in node
        cout << endl;
        cout << "->";
        Current = Current->Next;            // Point to the next node
    }
    cout << "Done" << endl;
}
    /*
    Public : Find(string typed)

    Description:
        - Receives the a character from the user.
        - Compare it with the animals data.
        - If a match is found, it is pushed to the Vector Results.
    Params:
        - string typed

    Returns:
        - vector<string> Results
*/
vector<string> Find(string typed)
{

    vector<string> Results;

    wordNode *Current = Head;

    while (Current)
    {
        string found = "";

        found = Current->word;              // Temp variable for the word of the
current wordNode stored

        int len = typed.length();           // length variable for the length of
the word typed/passed in

        if (found.substr(0, len) == typed)  // if the length of the word from
index 0 to the length of the
        {                                   // typed word is equal then it is
pushed to Results
            Results.push_back(found);
        }

        Current = Current->Next;            // traverse to next wordNode
    }

    return Results;                         // return the vector of results
}
};

/**
 * Main Driver
 *
 * For this program
 * *
 */
int main()
{
    LinkedList L1;                      // Linked List object
    vector<string> data;                // Placeholder data to read in the
dictionary.txt data

    ifstream infile;
    infile.open("dictionary.txt");

    Timer time;                          // Create a timer.
    time.Start();                        // Start the timer.

    while (!infile.eof())                // If the file is not empty.
    {
        string Temp;

        infile >> Temp;

        data.push_back(Temp);
    }

    time.End();

    cout << termcolor::green << time.Seconds() << termcolor::reset
         << " seconds to read in the 1st data."    << endl;
    Timer Load_Words;                       // Time to load the words into the
Linked List

    Load_Words.Start();

    for (int j = 0; j < data.size(); j++)
    {                                       // Loop through the vector.
        wordNode *Temp = new wordNode;      // Allocate new memories.

        string item = data[j];

        Temp->word = item;

        L1.Insert_Data(Temp);
    }

    Load_Words.End();

    cout << termcolor::green << Load_Words.Seconds() << termcolor::reset
         << " seconds to read in the 2nd data." << termcolor::reset << endl;

    char k;                                 // Hold the character being typed.
    string word = "";                       // Use to Concatenate letters.
    vector<string> Matches;                 // Any matches found in vector of data
Words.

    string Top_Results[10];                 // Initializing 10 words to print.
    int SearchResults;                      // Initializing the integer
SearchResults.

    cout << "Type keys and watch what happens. Type capital"
         << termcolor::red << " Z to quit." << termcolor::reset << endl;

    while ((k = getch()) != 'Z')            // While capital Z is not typed keep
looping.
    {
        if ((int)k == 127)                  // Tests for a backspace and if
pressed deletes.
        {
            if (word.size() > 0)
            {
                word = word.substr(0, word.size() - 1);
            }
        }

        else
        {
            if (!isalpha(k))                // Making sure a letter was pressed.
            {
                cout << "Letters only!\n";
                continue;
            }
            if ((int)k >= 97)                // Making sure its lowercase.
            {
                k -= 32;                     // Make the input word  capital
letters.
            }
        }
        word += k;                           // Append character to word.

        Timer Auto_Suggestion;               // Timer for (word suggestions and
total words found).

        Auto_Suggestion.Start();
        Matches = L1.Find(word);
        Auto_Suggestion.End();

        SearchResults = Matches.size();

        if ((int)k != 32)                    // When the key pressed is not "Space
bar".
        {
            cout << "Keypressed: "     << termcolor::red   << k      << " = "
                 << termcolor::green   << (int)k << termcolor::reset << endl;
            cout << "Current Substr: " << termcolor::red   << word
                 << termcolor::reset   << endl;
            cout << termcolor::red     << SearchResults    << termcolor::reset
                 << " words found in " << termcolor::green <<
Auto_Suggestion.Seconds()
                 << termcolor::reset   << " seconds"       << termcolor::reset <<
endl;

            if (Matches.size() >= 10)        // Prints out the top 10 results.
            {
                for (int i = 0; i < 10; i++)
                {
                    Top_Results[i] = Matches[i];
                    cout << Top_Results[i] << " ";
                }
            }
            else
            {
                for (int j = 0; j < Matches.size(); j++)
                {
                    Top_Results[j] = Matches[j];
                    cout << Top_Results[j] << " ";
                }
            }

            cout << termcolor::reset << endl << endl;
        }
    }
    return 0;
}
```