

```

1  //////////////////////////////////////
2  //
3  // Authors:      David Hawkins
4  // Email:        david.james@hawkinsonline.us
5  // Label:        P02
6  // Title:        Assignment 5 - Processing in Linear Time
7  // Course:       3013
8  // Semester:     Spring 2020
9  //
10 // Description:
11 //      This program implements a linked list class that holds values read in
12 //      from a JSON file. This program also times how long it takes to load
13 //      the data from the JSON file into the linked list. Furthermore, it
14 //      incorporates getch functionality that reads input from the keyboard
15 //      and searches for the user's entry in the linked list (this is timed
16 //      as well).
17 //
18 // Files:
19 //      main.cpp          : Driver program
20 //      json.hpp          : File containing Json implementation
21 //      JsonFacade.hpp    : File containing simplified Json implementation
22 //      mygetch.hpp       : File containing getch implementation
23 //      Timer.hpp         : File used to implement timer
24 //
25  //////////////////////////////////////
26
27  #include <iostream>
28  #include "JsonFacade.hpp"
29  #include <time.h>
30  #include <chrono>
31  #include "Timer.hpp"
32  #include "mygetch.hpp"
33
34  using namespace std;
35
36  /**
37   * wordNode
38   *
39   * @description:
40   *      This is the struct that defines the nodes that are contained in the
41   *      linked list class below.
42   *
43   * @methods:
44   *      constructors:
45   *          wordNode(string w, string def) : Parameterized Constructor
46   */
47  struct wordNode{
48      string word;
49      string definition;
50      wordNode* next;
51
52      wordNode(string w, string def){
53          word = w;
54          definition = def;
55          next = NULL;
56      }
57  };
58
59  /**
60   * LinkedList
61   *
62   * @description:
63   *      This class implements a singly-linked list of wordNodes.
64   *
65   * @methods:
66   *      constructors:
67   *          LinkedList() : Default Constructor
68   *      public:
69   *          GetSize() : Returns size of the linked list

```

```

70      *          Insert(string, string)          : Inserts a new node in list
71      *          Remove()                        : Removes one node from list
72      *          PrintList()                     : Prints words in the list
73      *          PrintTail()                     : Prints tail at end of list
74      *          Search(string, int&, string&)    : Searches for item in list
75      *
76  */
77  class LinkedList {
78  private:
79      wordNode* head;
80      wordNode* tail;
81      int size;
82
83  public:
84      /**
85       * Public : LinkedList
86       *
87       * @description:
88       *     Creates instances of the Linked List class.
89       *
90       * @param
91       *     None
92       *
93       * @return
94       *     None
95       */
96      LinkedList() {
97          head = tail = NULL;
98          size = 0;
99      }
100
101      /**
102       * Public : GetSize
103       *
104       * @description:
105       *     Returns the current size of the list.
106       *
107       * @param
108       *     None
109       *
110       * @return
111       *     [int] : Size of the linked list
112       */
113      int GetSize() {
114          return size;
115      }
116
117      /**
118       * Public : Insert
119       *
120       * @description:
121       *     Inserts a new node into the linked list with the word
122       *     and definition.
123       *
124       * @param
125       *     [string] : word being inserted
126       *     [string] : definition being inserted
127       *
128       * @return
129       *     None
130       */
131      void Insert(string w, string def) {
132          if (head == NULL) { //if list is empty, insert at beginning
133              head = tail = new wordNode(w, def);
134              size++;
135          }
136          else { //if list is not empty
137              tail->next = new wordNode(w, def);
138              tail = tail->next;

```

```

139         size++;
140     }
141 }
142
143 /**
144  * Public : Remove
145  *
146  * @description:
147  *     Removes one wordNode from the list.
148  *
149  * @param
150  *     None
151  *
152  * @return
153  *     [string] : Word removed from the list
154  */
155 string Remove() {
156     if (head != NULL) {
157         wordNode* temp = head;
158         string data;
159
160         data = temp->word;
161         head = head->next;
162         delete temp;
163         temp = NULL;
164         size--;
165
166         return data;
167     }
168     else {
169         return "Can't remove from empty list.";
170     }
171 }
172
173 /**
174  * Public : PrintList
175  *
176  * @description:
177  *     Creates a string representation of the linked list that can
178  *     be printed to the screen.
179  *
180  * @param
181  *     None
182  *
183  * @return
184  *     [string] : String representation of the linked list
185  */
186 string PrintList() {
187     wordNode* temp = head;
188     string list;
189
190     while (temp != NULL) {
191         list += temp->word + "->";
192         temp = temp->next;
193     }
194
195     return list;
196 }
197
198 /**
199  * Public : PrintTail
200  *
201  * @description:
202  *     Prints the word contained in the very last wordNode in the list.
203  *
204  * @param
205  *     None
206  *
207  * @return

```

```

208     *         [string]      : Word in the last node
209 */
210 string PrintTail(){
211     string list;
212
213     list += tail->word + ": " + tail->definition;
214
215     return list;
216 }
217
218 /**
219  * Public : Search
220  *
221  * @description:
222  *     Searches the list for the item described using the .substr method
223  *     from the string class. When a match is found, the number of matches
224  *     is increased by one and the first ten results are compiled into
225  *     one string.
226  *
227  * @param
228  *     [string]      : Item being searched for
229  *     [int&]        : Number of matches found
230  *     [string&]     : First ten results found in the list
231  *
232  * @return
233  *     None
234 */
235 void Search(string item, int & match, string& results){
236     wordNode* temp = head;
237     string test;
238
239     while (temp != NULL){
240         test = temp->word;
241         if(item == test.substr(0, item.length())){ //Checks substring
242             match++;                                //If found, increment
243             if (match <= 10){
244                 results += temp->word + ", ";        //Then concatenate
245             }
246         }
247         temp = temp->next;                          //Traverse
248     }
249 }
250
251
252
253 };
254
255 /*****
256  * Function Prototypes
257  *****/
258 string GetInput();
259 void LoadList(Timer& T, LinkedList& L, JsonFacade& J, double& sec, long& mill);
260 void PrintResults(int& match, string& results, Timer& S);
261
262 int main(){
263
264     Timer T;                                         //Timer object for loading list
265     Timer S;                                         //Timer object for searching
266     LinkedList L;                                   //Linked list object
267     double Sec;                                     //num of secs to load list
268     long Milli;                                     //num of millisecs to load list
269     int match = 0;                                  //used to record number of matches
270     string result;                                  //records first ten results from search
271     string item;                                    //holds what we're searching for
272
273     JsonFacade J("dict_w_defs.json");              //Json object holding list of words
274
275     LoadList(T, L, J, Sec, Milli);                 //Loads list & times it
276

```

```

277     item = GetchInput();
278
279     S.Start();
280     L.Search(item, match, result);
281     S.End();
282
283     PrintResults(match, result, S);
284
285     return 0;
286 }
287
288 /**
289  * Public : GetchInput
290  *
291  * @description:
292  *     Captures user input from the keyboard and concatenates characters
293  *     into a string.
294  *
295  * @param
296  *     None
297  *
298  * @return
299  *     [string] : String captured at the keyboard
300 */
301 string GetchInput(){
302     char k;           // holder for character being typed
303     string word = ""; // var to concatenate letters to
304
305     cout << "Type what you're searching for, then press enter: ";
306
307     while ((k = getch()) != 13) {
308         word += k;
309         cout << k;
310     }
311
312     cout << endl;
313
314     return word;
315 }
316
317 /**
318  * Public : LoadList
319  *
320  * @description:
321  *     Reads values from the Json object and stores them in the linked
322  *     list object.
323  *
324  * @param
325  *     [Timer&] : Timer object used for timing
326  *     [LinkedList&] : Linked List object that is loaded into
327  *     [JsonFacade&] : Json object that is read from
328  *     [double&] : Number of seconds required to load
329  *     [long&] : Number of milliseconds required to load
330  *
331  * @return
332  *     None
333 */
334 void LoadList(Timer& T, LinkedList& L, JsonFacade& J, double& sec, long& mill){
335     T.Start(); //Timer starts
336     for (int i = 0; i < J.GetSize(); i++){ //Insert values into list
337         L.Insert(J.getKey(i), J.getValue(J.getKey(i)));
338     }
339     T.End(); //Timer stops
340
341     sec = T.Seconds(); //Time logged
342     mill = T.MilliSeconds();
343 }
344
345 /**

```

```

346     * Public : PrintResults
347     *
348     * @description:
349     *     Prints the results from searching the linked list for the item
350     *
351     * @param
352     *     [int&]      : Number of matches found in linked list
353     *     [string&]   : Top results found in list
354     *     [Timer&]    : Timer object used for timing
355     *
356     * @return
357     *     None
358     */
359 void PrintResults(int& match, string& results, Timer& S){
360     cout << endl << match << " words found in " << S.Seconds()
361         << " seconds" << endl << endl;
362     cout << results << endl;
363 }

```