```
cat -b p7.cpp
     1      /*
     2       PROGRAM NAME: Program 7: Hashing (Seperate Chaining)
     3
     4       PROGRAMMER:   James Francis
     5
     6       CLASS:        CSC 331.001, Fall 2014
     7
     8       INSTRUCTOR:   Dr. Robert Strader
     9
    10       DATE STARTED: November 17, 2014
    11
    12       DUE DATE:     November 18, 2014
    13
    14       PROGRAM PURPOSE:
    15       This program is used to implement to Hashtable class, and Node class, for use in a dictionary-like Structure.
    16       This program will read in data from prog7.dat and search for words that may be in the hashtable based upon input
from the file.
    17
    18       VARIABLE DICTIONARY:
    19
    20       t: Reference to a Hashtable Object
    21       infile: fstream object, input file
    22
    23
    24       ADTs: Hashtable
    25
    26
    27       FILES USED: Hashtable.h prog7.dat
    28
    29
    30       SAMPLE INPUTS: (from prog7.dat)
    31
    32       the relative lack of acceptance
    33       of these products in the
    34       corporate marketplace is
    35       due less to technical than
    36       to political factors the
    37       availability of this technology
    38       threatens the perks privileges
    39       and traditions of corporate
    40       management
    41       **********
    42       the
    43       political
```

```
44       lack
45       relative
46       less
47       forgive
48       tradition
49       factors
50       more
51
52       SAMPLE OUTPUTS:
53
54       Found 'the' with 1 access(es)
55       the appears on line(s): 1 2 5 7
56
57       Found 'political' with 2 access(es)
58       political appears on line(s): 5
59
60       Found 'lack' with 1 access(es)
61       lack appears on line(s): 1
62
63       Found 'relative' with 1 access(es)
64       relative appears on line(s): 1
65
66       Found 'less' with 1 access(es)
67       less appears on line(s): 4
68
69       Unable to find 'forgive' with 2 access(es).
70
71       Unable to find 'tradition' with 4 access(es).
72
73       Found 'factors' with 3 access(es)
74       factors appears on line(s): 5
75
76       Unable to find 'more' with 5 access(es).
77
78       ---Averages rounded to the nearest integer---
79       Average number of accesses for successful retrieval: 1
80
81       Average number of accesses for unsuccessful retrieval: 3
82
83       Average number of accesses over total retrievals: 2
84
85       ------------------------------------------------------------------------*/


86      #include "HashTable.h"
```

```
    87      int main(int argc, const char * argv[]) {
    88
    89          HashTable t = *new HashTable();
    90
    91          fstream infile("../instr/prog7.dat", ios::in);
    92          if (!infile.is_open()) {
    93              cout<<"File not found."<<endl;
    94              return -1;
    95          }
    96
    97          t.populate(infile);
    98
    99          infile.close();
   100
   101          t.hashingReport();
   102
   103          return 0;
   104      }
printf \\n

cat -b Node.h
     1      /*
     2       PROGRAM NAME: Program 7: Hashing (Seperate Chaining)
     3
     4       PROGRAMMER:   James Francis
     5
     6       CLASS:        CSC 331.001, Fall 2014
     7
     8       INSTRUCTOR:   Dr. Robert Strader
     9
    10       DATE STARTED: November 17, 2014
    11
    12       DUE DATE:     November 18, 2014
    13
    14       PROGRAM PURPOSE:
    15       Declaration of the Node Class
    16
    17
    18       VARIABLE DICTIONARY: in functions
    19
    20       ADTs: none
    21
    22       FILES USED: none
    23
```

```
    24       SAMPLE INPUTS: none
    25
    26       SAMPLE OUTPUTS: none
    27
    28
    29       ------------------------------------------------------------------------*/

    30      #ifndef __p7__Node__
    31      #define __p7__Node__

    32      #include <stdio.h>
    33      #include <string>
    34      #include <iomanip>
    35      #include <fstream>
    36      #include <iostream>
    37      #include <sstream>

    38      using namespace std;


    39      class Node{
    40
    41      public:
    42          Node();
    43          Node(int n, string str, string line);
    44          int getSize();
    45          string getWord();
    46          Node* getNext();
    47          void setNext(Node* next);
    48          string getLines();
    49          void addLine(string line);
    50
    51      private:
    52          int size;
    53          string word;
    54          string lines;
    55          Node* next;
    56          void setSize(int n);
    57          void setWord(string str);
    58          void setLine(string line);
    59
    60      };

    61      #endif /* defined(__p7__Node__) */
printf \\n\\n
```

```
cat -b Node.cpp
     1      /*
     2       PROGRAM NAME: Program 7: Hashing (Seperate Chaining)
     3
     4       PROGRAMMER:   James Francis
     5
     6       CLASS:        CSC 331.001, Fall 2014
     7
     8       INSTRUCTOR:   Dr. Robert Strader
     9
    10       DATE STARTED: November 17, 2014
    11
    12       DUE DATE:     November 18, 2014
    13
    14       PROGRAM PURPOSE:
    15       Definition of the Node Class
    16
    17
    18       VARIABLE DICTIONARY: in functions
    19
    20       ADTs: none
    21
    22       FILES USED: none
    23
    24       SAMPLE INPUTS: none
    25
    26       SAMPLE OUTPUTS: none
    27
    28
    29       ------------------------------------------------------------------------*/

    30      #include "Node.h"
    31      Node::Node()
    32      : next(NULL) {
    33          //---------------------------
    34          // Default Constructor
    35          //---------------------------
    36          setSize(0);
    37          setWord("");
    38      }

    39      Node::Node(int n, string str, string line)
    40      : next(NULL)
```

```
41      {
42          //---------------------------
43          // Initializing Constructor
44          //---------------------------
45          setSize(n);
46          setWord(str);
47          setLine(line);
48      }

49      //--------------
50      //Begin Setters
51      //--------------
52      void Node::setSize(int n){
53          size = n;
54      }

55      void Node::setWord(string str){
56          word = str;
57      }

58      void Node::setNext(Node* nextNode){
59          next = nextNode;
60      }

61      void Node::setLine(string line){
62          lines=(line+" ");
63      }
64      void Node::addLine(string line){
65          lines+=(line+" ");
66      }


67      //--------------
68      //Begin Getters
69      //--------------
70      int Node::getSize(){
71          return size;
72      }

73      string Node::getWord(){
74          return word;
75      }
76      Node* Node::getNext(){
77          return next;
78      }
```

```
    79      string Node::getLines(){
    80          return lines;
    81      }



printf \\n

cat -b HashTable.h
     1      /*
     2       PROGRAM NAME: Program 7: Hashing (Seperate Chaining)
     3
     4       PROGRAMMER:   James Francis
     5
     6       CLASS:        CSC 331.001, Fall 2014
     7
     8       INSTRUCTOR:   Dr. Robert Strader
     9
    10       DATE STARTED: November 17, 2014
    11
    12       DUE DATE:     November 18, 2014
    13
    14       PROGRAM PURPOSE:
    15       Declaration of the HashTable Class

    16
    17       VARIABLE DICTIONARY: in functions
    18
    19       ADTs: none
    20
    21       FILES USED: none
    22
    23       SAMPLE INPUTS: none
    24
    25       SAMPLE OUTPUTS: none
    26
    27
    28       ------------------------------------------------------------------------*/

    29      #ifndef p7_HashTable_h
    30      #define p7_HashTable_h
    31      #include <string>
    32      #include <iomanip>
    33      #include <fstream>
    34      #include <iostream>
```

```
    35      #include <sstream>
    36      #include <cmath>
    37      #include <cstring>

    38      #include "Node.h"


    39      using namespace std;


    40      class HashTable{
    41      public:
    42          HashTable();
    43          void populate(fstream& infile);
    44          void hashingReport();
    45
    46      private:
    47
    48          int success;
    49          int failed;
    50          int successCount;
    51          int failedCount;
    52
    53          int hashValue(string str);
    54          void printTable(Node* table[]);
    55          string NumberToString(int t);
    56          void insert(Node* newNode, int index, Node* table[], int count);
    57          bool scanChain(Node* current, string word);
    58          void search(string line, Node* table[]);
    59          int getSuccess();
    60          int getFailed();
    61          void setSuccess(int n);
    62          void setFailed(int n);
    63      };

    64      #endif
```

```
printf \\n\\n


cat -b HashTable.cpp
     1      /*
     2       PROGRAM NAME: Program 7: Hashing (Seperate Chaining)
     3
     4       PROGRAMMER:   James Francis
```

```
 5
 6       CLASS:        CSC 331.001, Fall 2014
 7
 8       INSTRUCTOR:   Dr. Robert Strader
 9
10       DATE STARTED: November 17, 2014
11
12       DUE DATE:     November 18, 2014
13
14       PROGRAM PURPOSE:
15       Definition of the Card Class

16
17       VARIABLE DICTIONARY: in functions
18
19       ADTs: none
20
21       FILES USED: none
22
23       SAMPLE INPUTS: none
24
25       SAMPLE OUTPUTS: none
26
27
28       ----------------------------------------------------------------------*/



29      #include "HashTable.h"

30      HashTable::HashTable(){
31          //---------------------------
32          // Default Constructor
33          //---------------------------
34
35          setSuccess(0);
36          setFailed(0);
37      }


38      void HashTable::populate(fstream& infile) {
39          //-----------------------------------------------------------------------
40          //Preconditions: Calling code calls the HashTable populate function
41          //
42          //Postconditions: The hashtable object has an array of Node pointers populated
```

```
43          //                  with NULL Nodes or Nodes containing values.
44          //
45          //Variables used:
46          //              table: an array of Node pointers
47          //              stars: delimiting string
48          //              word, line: strings that contain a line of input and a word
49          //              size, count: integer variables used for the size of a word
50          //              and the current line count
51          //-------------------------------------------------------------------
52
53
54          Node* table[23];
55
56          for (int i = 0; i < 23; i++) {
57              table[i] = NULL;
58          }
59
60          string stars = "**********";
61
62
63          string word;
64          string line;
65          int size;
66          int count = 0;
67
68          while (line.compare(stars)!=0) {
69              getline(infile, line);
70              count++;
71              word='\0';
72              size = 0;
73
74
75              stringstream linestream(line);
76              while (linestream>>word && word.compare(stars)!=0) {
77
78                  int index = HashTable::hashValue(word);
79                  Node* newNode = new Node((int)word.length(), word, NumberToString(count));
80                  insert(newNode, index, table, count);
81
82              }
83          }
84
85
86          while (getline(infile, line)) {
87              search(line, table);
```

```
 88              cout << endl;
 89          }
 90
 91      }

 92      int HashTable::hashValue(string str){
 93          //-----------------------------------------------------------------------
 94          //Preconditions: Calling code has passed a word, as a string, to this function
 95          //
 96          //Postconditions: An integer value is returned to the calling code, representing
 97          //                  the passed word's hash value.
 98          //
 99          //Variables used:
100          //              first: integer, offset of first character in the word from 'a'
101          //              last: integer, offset of last character in the word from 'a'
102          //              position: hashvalue of the passed string
103          //-----------------------------------------------------------------------
104
105          int first = str.at(0) - 97;
106          int last = str.at(str.size()-1) - 97;
107
108
109          int position = (int) (first*256+last)%23;
110
111          return position;
112
113      }

114      void HashTable::printTable(Node* table[]){
115          //-----------------------------------------------------------------------
116          //Preconditions: calling code passed an array of Node pointers to this function
117          //
118          //Postconditions: The hashtable is printed along with its seperate chains
119          //
120          //Variables used:
121          //              current: Node pointer
122          //
123          //-----------------------------------------------------------------------
124
125
126          for(int i = 0; i<23; i++){
127              Node* current = table[i];
128
129              if (current == NULL) {
130                  cout<<endl;
```

```
131             }
132             else{
133                 while (current!= NULL) {
134                     cout<<current->getWord()<<" ";
135                     current = current->getNext();
136                 }
137                 cout<<endl;
138                 cout<<endl;
139             }
140         }
141 
142 
143     }

144     void HashTable::insert(Node* newNode, int index, Node* table[], int count){
145         //-----------------------------------------------------------------------
146         //Preconditions: calling code has populated the passed parameters properly
147         //
148         //Postconditions: The passed node has been hashed into the table, or the
149         //                line that the word was on was added to the a previously
150         //                hashed Node containing the same word.
151         //
152         //Variables used:
153         //              duplicate: boolean flag to determine if the passed value is
154         //                         found in the hash table
155         //              current: Node pointer
156         //              newNode: node Pointer that contains a word to be hashed into
157         //                         the table
158         //              index: integer containing the hash value for the passed word
159         //
160         //              count: integer containing the line the word was input from
161         //
162         //-----------------------------------------------------------------------
163 
164 
165 
166         bool duplicate = false;
167 
168         if (table[index]==NULL) {
169             table[index] = newNode;
170 
171         } else {
172             Node *current = table[index];
173 
174 
```

```
175             if (scanChain(current, newNode->getWord())) {
176 
177                 current->addLine(NumberToString(count));
178                 duplicate = true;
179 
180             }
181             else{
182 
183                 while (current->getNext() != NULL) {
184                     if (scanChain(current, newNode->getWord())) {
185 
186                         current->addLine(NumberToString(count));
187                         duplicate = true;
188                         break;
189                     }
190                     else
191                         current = current->getNext();
192 
193                 }
194             }
195 
196             if (duplicate) {
197                 newNode = NULL;
198                 current = NULL;
199             }else{
200                 current->setNext(newNode);
201             }
202         }
203 
204     }

205     string HashTable::NumberToString ( int t )
206     {
207         //-------------------------------------------------------------------------
208         //Preconditions: calling code has passed an integer, t, to this function
209         //
210         //Postconditions: a string representation of the passed integer is returned
211         //
212         //Variables used:
213         //              ss: stringstream object used to convert the passed integer
214         //
215         //-------------------------------------------------------------------------
216         ostringstream ss;
217         ss << t;
218         return ss.str();
```

```
219     }

220     bool HashTable::scanChain(Node* current, string word){
221         //-------------------------------------------------------------------------
222         //Preconditions: Calling code has encountered a seperate chain in the
223         //               hash table
224         //
225         //Postconditions: a string representation of the passed integer is returned
226         //
227         //Variables used:
228         //              result: boolean flag that identifies if a duplicate word is
229         //                      encountered
230         //
231         //-------------------------------------------------------------------------
232
233         bool result = false;
234         while (current!=NULL) {
235
236             if (word.compare(current->getWord())==0) {
237                 result = true;
238                 break;
239             }
240             else current = current->getNext();
241         }
242         return result;
243     }

244     void HashTable::search(string line, Node* table[]){
245         //-------------------------------------------------------------------------
246         //Preconditions: calling code has encountered a word to search the table for
247         //
248         //Postconditions: A success, along with lines the word was found on in the
249         //               input file, or a failure to find, with the number of
250         //               accesses to do so is returned to console
251         //
252         //Variables used:
253         //              accesses: integer that counted the number of accesses to
254         //                        the hash table
255         //              word: string, word to be searched for in the table
256         //              linestream: stringstream object, used to ensure escape
257         //                          charcaters are removed from input string
258         //              searchValue: integer, hash value of the passed word
259         //-------------------------------------------------------------------------
260
261
```

```
262         int accesses= 1;
263         string word;
264         stringstream linestream (line);
265         linestream >>word;
266         int searchValue = hashValue(word);
267         Node *current = table[searchValue];
268 
269         while (current!=NULL) {
270 
271             if (word.compare(current->getWord())==0) {
272                 setSuccess(accesses);
273                 cout<< "Found '"<<word<<"' with "<<accesses<<" access(es)"<<endl;
274                 cout<< word <<" appears on line(s): "<< current->getLines()<<endl;
275                 return;
276             }
277             else {
278                 current = current->getNext();
279                 accesses++;
280             }
281         }
282 
283         cout<< "Unable to find '"<<word<<"' with "<<accesses<<" access(es)."<<endl;
284 
285         setFailed(accesses);
286         return;
287     }


288     void HashTable::hashingReport() {
289         //-------------------------------------------------------------------------
290         //Preconditions: calling code has finished searching
291         //
292         //Postconditions: Average number of successful accesses, failed accesses,
293         //                  and avwrage total accessess are printed to screen
294         //
295         //Variables used:
296         //              successRate: double, stores the average success rate
297         //              failRate: double, stores the average fail rate
298         //              totalRetrievalRate: double, stores the overall average
299         //-------------------------------------------------------------------------
300 
301         cout<<"---Averages rounded to the nearest integer---"<<endl;
302 
303         double successRate = floor(((double)this->getSuccess()/(double)successCount)+.5);
304 
```

```
305           cout<<"Average number of accesses for successful retrieval: "<< successRate<<endl;
306
307
308           cout<<endl;
309
310           double failRate = floor(((double)this->getFailed()/(double)this->failedCount)+.5);
311
312           cout<<"Average number of accesses for unsuccessful retrieval: "<< failRate <<endl;
313
314           cout<<endl;
315           double totalRetrievalRate = floor(((this->getSuccess()+this->getFailed())/(this->successCount))+.5);
316
317           cout<<"Average number of accesses over total retrievals: "<< totalRetrievalRate <<endl;
318       }


319       //--------------------
320       //   SETTERS
321       //--------------------
322       void HashTable::setSuccess(int n){
323           success +=n;
324           successCount++;
325       }

326       void HashTable::setFailed(int n){
327           failed +=n;
328           failedCount++;
329       }

330       //--------------------
331       //   GETTERS
332       //--------------------
333       int HashTable::getSuccess(){
334           return success;
335       }

336       int HashTable::getFailed(){
337           return failed;
338       }


:
g++ Node.cpp HashTable.cpp p7.cpp -o prog7
printf \\n\\n
```

```
:

prog7
Found 'the' with 1 access(es)
the appears on line(s): 1 2 5 7

Found 'political' with 2 access(es)
political appears on line(s): 5

Found 'lack' with 1 access(es)
lack appears on line(s): 1

Found 'relative' with 1 access(es)
relative appears on line(s): 1

Found 'less' with 1 access(es)
less appears on line(s): 4

Unable to find 'forgive' with 2 access(es).

Unable to find 'tradition' with 4 access(es).

Found 'factors' with 3 access(es)
factors appears on line(s): 5

Unable to find 'more' with 5 access(es).

---Averages rounded to the nearest integer---
Average number of accesses for successful retrieval: 1

Average number of accesses for unsuccessful retrieval: 3

Average number of accesses over total retrievals: 2
```