```cpp
/*
 * Project 04
 * Working example of a templated binary search tree.  Loads a specially formatted commands file which,
 * when read by the program, constructs a binary search tree of type Word with the commanded configuration.
 * All commands are reported to the console.
 *
 * Programmer:  Chad Philip Johnson
 * Date Created:  Friday, April 26th, 2013
 * Last Date Modified:  Thursday, May 09th, 2013
 *
 * Sources Used:
 *      BSTree.h
 *          - to create binary search tree
 *      Word.h
 *          - the type used for this particular implementation of the binary search tree
 */

#pragma once

#include <iostream>
#include <cstdlib>
#include <string>
#include <fstream>

#include "BSTree.h"
#include "Word.h"

/**
 * Processes the program commands received from a formatted data file.
 * @param pobjBSTreeDriverTree The instance of the binary search tree to be structured by the commands.
 * @param charCommand The command to be issued to the binary search tree.
 * @param pobjWordSelectedWord The value of type Word to be added to the binary search tree.
 */
template<typename T>
void processInput( BSTree<T>*& pobjBSTreeDriverTree, char& charCommand, Word*& pobjWordSelectedWord );

int main( int argc, char* argv[] )
{
    BSTree<Word>* pobjBSTreeDriverTree  = NULL;
    bool boolPreserveCaseOfInput        = false;

    if( argc > 1 )
    {
        if( argc > 2)
        {
            if( argv[2][0] == 'P' || argv[2][0] == 'p' )
            {
                boolPreserveCaseOfInput = true;
            }
        }
```

```
        ifstream objifstreamInputFile( argv[1] );
        if( objifstreamInputFile.good() )
        {
            string strNextLine  = "";
            Word* pobjWordSelectedWord  = NULL;

            while( getline( objifstreamInputFile, strNextLine ) )
            {
                if( strNextLine.length() == 1 )
                {
                    processInput( pobjBSTreeDriverTree, strNextLine[0], pobjWordSelectedWord );
                }
                else if( strNextLine.length() > 1 )
                {
                    if( !boolPreserveCaseOfInput )
                    {
                        unsigned int uintWordLength = strNextLine.length();

                        for( unsigned int uintI = 0; uintI < uintWordLength; uintI++ )
                        {
                            strNextLine.at( uintI ) = toupper( strNextLine.at( uintI ));
                        }
                    }

                    pobjWordSelectedWord    = new Word( strNextLine.substr( 2 ) );
                    processInput( pobjBSTreeDriverTree, strNextLine[0], pobjWordSelectedWord );

                    delete pobjWordSelectedWord;
                }
            }

            objifstreamInputFile.close();
        }
        else
        {
            cout << "The file " << argv[1] << " does not exist." << endl;
        }
    }
    else
    {
        cout << "Usage of proj4.exe:" << endl;
        cout << "\t" << "proj4.exe NAME_OF_FILE" << endl;
        cout << "\t\tProcess commands in NAME_OF_FILE" << endl;
        cout << "\t" << "proj4.exe NAME_OF_FILE p" << endl;
        cout << "\t\tOptional command [p]:  Process commands in NAME_OF_FILE\n\t\tand preserve original case of input"  << endl;
    }

}
```

```
101  template<class T>
102  void processInput( BSTree<T>*& pobjBSTreeDriverTree, char& charCommand, Word*& pobjWordSelectedWord )
103  {
104      switch( charCommand )
105      {
106          case '#':
107              break;
108
109          case 'C':
110          case 'c':
111              if( pobjBSTreeDriverTree != NULL )
112              {
113                  delete pobjBSTreeDriverTree;
114                  pobjBSTreeDriverTree    = NULL;
115              }
116              pobjBSTreeDriverTree = new BSTree<Word>;
117
118              cout << "TREE CREATED" << endl;
119              break;
120      }
121
122      if( pobjBSTreeDriverTree != NULL )
123      {
124          switch( charCommand )
125          {
126              case 'X':
127              case 'x':
128                  pobjBSTreeDriverTree->clear();
129                  cout << "TREE CLEARED" << endl;
130                  break;
131
132              case 'D':
133              case 'd':
134                  delete pobjBSTreeDriverTree;
135                  pobjBSTreeDriverTree    = NULL;
136                  cout << "TREE DELETED" << endl;
137                  break;
138
139              case 'I':
140              case 'i':
141                  if( pobjBSTreeDriverTree->find( *pobjWordSelectedWord ) )
142                  {
143                      pobjBSTreeDriverTree->insert( *pobjWordSelectedWord );
144                      cout << "WORD " << pobjWordSelectedWord->getWord() << " INCREMENTED" << endl;
145                  }
146                  else
147                  {
148                      pobjBSTreeDriverTree->insert( *pobjWordSelectedWord );
149                      cout << "WORD " << pobjWordSelectedWord->getWord() << " INSERTED" << endl;
150                  }
```

```
151
152                 break;
153
154             case 'F':
155             case 'f':
156                 if( pobjBSTreeDriverTree->getCount() != 0 )
157                 {
158                     if( pobjBSTreeDriverTree->find( *pobjWordSelectedWord ) )
159                     {
160                         cout << "FOUND " << pobjWordSelectedWord->getWord() << endl;
161                     }
162                     else
163                     {
164                         cout << pobjWordSelectedWord->getWord() << " NOT FOUND" << endl;
165                     }
166                 }
167                 else
168                 {
169                     cout << "TREE EMPTY" << endl;
170                 }
171
172                 break;
173
174             case 'R':
175             case 'r':
176                 if( pobjBSTreeDriverTree->getCount() != 0 )
177                 {
178                     if( pobjBSTreeDriverTree->find( *pobjWordSelectedWord ) )
179                     {
180                         pobjBSTreeDriverTree->remove( *pobjWordSelectedWord );
181                         cout << "REMOVED " << pobjWordSelectedWord->getWord() << endl;
182                     }
183                     else
184                     {
185                         cout << pobjWordSelectedWord->getWord() << " NOT FOUND" << endl;
186                     }
187                 }
188                 else
189                 {
190                     cout << "TREE EMPTY" << endl;
191                 }
192
193                 break;
194
195             case 'G':
196             case 'g':
197                 if( pobjBSTreeDriverTree->getCount() != 0 )
198                 {
199                     if( pobjBSTreeDriverTree->find( *pobjWordSelectedWord ) )
200                     {
```

```
                        Word* pobjWordGottenWord    = pobjBSTreeDriverTree->get( *pobjWordSelectedWord );
                        cout << "GOT " << pobjWordGottenWord->getWord() << " " << pobjWordGottenWord->getCount() << endl;
                    }
                    else
                    {
                        cout << pobjWordSelectedWord->getWord() << " NOT FOUND" << endl;
                    }
                }
                else
                {
                    cout << "TREE EMPTY" << endl;
                }

                break;

            case 'N':
            case 'n':
                cout << "TREE SIZE IS " << pobjBSTreeDriverTree->getCount() << endl;

                break;

            case 'O':
            case 'o':
                if( pobjBSTreeDriverTree->getCount() != 0 )
                {
                    pobjBSTreeDriverTree->inOrder();
                }
                else
                {
                    cout << "TREE EMPTY" << endl;
                }

                break;

            case 'E':
            case 'e':
                if( pobjBSTreeDriverTree->getCount() != 0 )
                {
                    pobjBSTreeDriverTree->reverseOrder();
                }
                else
                {
                    cout << "TREE EMPTY" << endl;
                }

                break;
        }
    }
    else if( charCommand != '#' )
    {
```

```
        cout << "MUST CREATE TREE INSTANCE" << endl;
    }
}

```