```
 1: /*
 2:  *      wordbench2.h
 3:  *
 4: */
 5:
 6: #include <xstring.h>
 7: #include <list.h>
 8: #include <oaa.h>
 9:
10: class WordBench
11: {
12: public:
13:   WordBench           (){}
14:   virtual ~WordBench  (){}
15:   bool   ReadText     (const fsu::String& infile);
16:   bool   WriteReport  (const fsu::String& outfile,
17:                        unsigned short kw = 15,   // key col width
18:                        unsigned short dw = 15   // data col width
19:                        )  const;
20:   void   ShowSummary  () const;
21:   void   ClearData    ();
22:
23:   //User Included Functions
24:   bool inRange(const char&);
25:   bool Frequency(fsu::List<fsu::String>&);
26:
27: private:
28:   typedef fsu::String             KeyType;
29:   typedef size_t                  DataType;
30:
31:   size_t totalSize;
32:   size_t uniqueWords;
33:   fsu::OAA  < KeyType , DataType > frequency_;
34:   fsu::List < fsu::String >        infiles_;
35:   static void Wordify  (fsu::String& s); // optional
36: } ;
```

```
#include <wordbench2.h>
#include <cstring>
#include <fstream>

bool WordBench::ReadText(const fsu::String& infile)
{
  //Variable Declarations
  fsu::String word_i;
  char charB[1] = {'0'};
  const char* blank = charB;
  fsu::String blankWord(blank);
  fsu::List<char>* charList = new fsu::List<char>{};
  fsu::List<fsu::String>* greatWords = new fsu::List<fsu::String>{};
  std::ifstream inStream;
  char* goodChars = nullptr;
  char selection;

  inStream.open(infile.Cstr());
  if(!inStream)
  {
    std::cout << "File Failed To Open: " << std::endl;
    return true;
  }

  //Read Information
  while(!inStream.eof()){
  word_i.GetNext(inStream, ' ');
  infiles_.Insert(word_i);
  }

  for(fsu::List<fsu::String>::Iterator j = infiles_.Begin(); j != infiles_.End(); j++)
  {
    goodChars = new char[(*j).Size()];
    for(size_t count = 0, charCounter = 0; count < (*j).Size(); count++)
    {
    selection = tolower((*j).Element(count));
      if(count > 0)
      {
        if(inRange((*j).Element(count))){
              charList->Insert(tolower((*j).Element(count)));
              goodChars[charCounter++] = tolower((*j).Element(count));}
        else if(selection == '-' || selection == '"')
          {
            switch(selection)
            {
              case '-':
                if(inRange((*j).Element(count+1)) && inRange((*j).Element(count-1)))
                {
                  charList->Insert(tolower((*j).Element(count)));
                  goodChars[charCounter++] = tolower((*j).Element(count));
                }
                break;
              case '"':
                if(inRange((*j).Element(count+1)) && inRange((*j).Element(count-1)))
                {
                  charList->Insert(tolower((*j).Element(count)));
                  goodChars[charCounter++] = tolower((*j).Element(count));
                }
                break;
            }
          }
        else
          continue;
      }

      else if(count == 0)
      {
        if(inRange((*j).Element(count)))
        {
          charList->Insert(tolower((*j).Element(count)));
          goodChars[charCounter++] = tolower((*j).Element(count));
        }
      }
    }
    fsu::String temp(goodChars);
    greatWords->Insert(temp);
  }
```

```
  78:
  79:   fsu::List<fsu::String>::Iterator temp;
  80:   for(fsu::List<fsu::String>::Iterator beg = greatWords->Begin(); beg != greatWords->End(); ++beg){
  81:       fsu::String currentWord = *(beg);
  82:       if((currentWord.Element(0) == ' ') || (currentWord.Element(0) == '\0')){
  83:         temp = beg;
  84:         --beg;
  85:         greatWords->Remove(temp);
  86:       }
  87:   }
  88:
  89:   totalSize = greatWords->Size();
  90:   if(Frequency(*greatWords)){
  91:     return true;
  92:   }
  93:   return true;
  94: }
  95:
  96: bool WordBench::WriteReport(const fsu::String& outfile, unsigned short c1, unsigned short c2) const
  97: {
  98:   fsu::String word;
  99:   std::ofstream outputStream;
 100:
 101:   //Open File to Write Summary too
 102:   outputStream.open(outfile.Cstr());
 103:
 104:   outputStream << std::setw(c1)  << "Number of Words Read: " << totalSize << std::endl;
 105:   outputStream << std::setw(c2)  << "Number of Different Words: " << uniqueWords << std::endl;
 106:
 107:   outputStream.close();
 108:
 109:   return true;
 110: }
 111:
 112: void WordBench::ShowSummary() const
 113: {
 114:    std::cout << "Number of Words Read " << totalSize << std::endl;
 115:    std::cout << "Number of Different Words " << uniqueWords << std::endl;
 116: }
 117:
 118: void WordBench::ClearData()
 119: {
 120:   frequency_.Clear();
 121:   uniqueWords = 0;
 122:   totalSize = 0;
 123: }
 124:
 125: bool WordBench::inRange(const char& ch)
 126: {
 127:   return(((48 <= ch) && (ch <= 57)) ||
 128:          ((65 <= ch) && (ch <= 90)) ||
 129:          ((97 <= ch) && (ch <= 122)));
 130: }
 131:
 132: bool WordBench::Frequency(fsu::List<fsu::String>& fsuList)
 133: {
 134: for(fsu::List<fsu::String>::Iterator currentElement = fsuList.Begin();
 135: currentElement != fsuList.End(); currentElement++){
 136:       size_t freq = 1;
 137:       for(fsu::List<fsu::String>::Iterator comparativeElement = fsuList.Begin();
 138:       comparativeElement != fsuList.End(); comparativeElement++){
 139:        if((currentElement != comparativeElement) && ((*currentElement) == (*comparativeElement)))
 140:         ++freq;
 141:      }
 142:   if(!((*currentElement).Element(0) == '\0')){
 143:     frequency_.Get(*currentElement);
 144:     frequency_.Put(*currentElement, freq);
 145:   }
 146: }
 147: uniqueWords = frequency_.Size();
 148: return true;
 149: }
```