```
#include <algorithm>
#include <cctype>
#include <iostream>
#include <string>

using namespace std;

bool is_palindrome( const string& line );

int
main()
{
  cout << "This program will read input, one line at a time, and\n"
       << "determine which input lines are palindromes.  It will also\n"
       << "output a count of palindromes\n";

  unsigned int count=0;
  string line;
  while ( getline( cin, line ) )
    {
      if ( is_palindrome( line ) )
        {
          count ++ ;
          cout << line << endl;
        }
    }

  cout << "There were " << count << " lines containing palindromes.\n";
}


bool
is_palindrome( const string& line )
{
  string temp;
  string::const_iterator i;

  //  Pull out letters and place them in the temp string.


  //  Determine if the letters are the same in the first half and the
  //  second half.  Return false as soon as a difference is found.




  return true;
}
```

```
//  Program:  text_count.cpp
//  Author:   Chuck Stewart
//  Purpose:  Count the lines, characters (non-whitespace), and words
//    in a text file.   Count the occurrences of each letter and the
//    occurrences of each word.  Ignore punctuation in the words.

#include <algorithm>
#include <cctype>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
#include <vector>
using namespace std;

unsigned int count_characters( const string& a_line );
void add_to_letter_counts( const string & a_line,
                           vector<int>& letter_counters );
vector<string> break_up_line( const string& a_line );
void add_to_word_counts( vector<string> const& line_words,
                         map<string,int> & word_counts );

int
main( int argc, char* argv[] )
{
  if ( argc != 3)
    {
      cerr << "Usage: " << argv[0] << " text-file results-file";
      return 1;
    }

  ifstream in_str(argv[1]);
  if ( !in_str )
    {
      cerr << "Couldn’t open " << argv[1] << " to read.\n";
      return 1;
    }

  ofstream out_str(argv[2]);
  if ( !out_str )
    {
      cerr << "Couldn’t open " << argv[2] << " to write the results.\n";
      return 1;
    }

  unsigned int character_count = 0;
  unsigned int line_count = 0;
  vector<int> letter_counters(26, 0 );  //  Counts for the individual letters
  int all_word_counts = 0;              //  Total number of words
  map<string,int> word_counts;          //  Occurrence count for each word

  //  Handle one line at a time...

  string a_line;
  while ( getline( in_str, a_line ) )
    {
      line_count ++ ;
      character_count += count_characters( a_line );
      add_to_letter_counts( a_line, letter_counters );
      vector<string> words_in_line = break_up_line( a_line );
      add_to_word_counts( words_in_line, word_counts );
      all_word_counts += words_in_line.size();
    }

  //  Output char, word and line counters

  out_str << "\nHere are the statistics on the input text file:\n"
          << "  char count = " << character_count << "\n"
          << "  word count = " << all_word_counts << "\n"
```

```
        << "  line count = " << line_count << "\n";

  //  Output the letter counts

  out_str << "\nHere are the letter counts:\n";
  for ( unsigned int i = 0; i < 26; ++ i )
    {
      out_str << "  " << char( 'a' + i ) << ":  " << letter_counters[ i ] << "\n";
    }

  //  Output word occurrences
  out_str << '\n' << "Here are the word occurrence counts\n";
  for ( map<string,int>::iterator mp = word_counts.begin();
        mp != word_counts.end(); ++ mp )
    out_str << mp->first << '\t' << mp->second << '\n';
}


// Return the number of non-whitespace characters on the line

unsigned int
count_characters( const string& a_line )
{
  //  To be implemented in lecture









}


//  For each letter seen add to the appropriate count in the vector.

void
add_to_letter_counts( const string & a_line,
                      vector<int>& letter_counters )
{
  //  To be implemented in lecture










}


//  Break up a string storing a line of input into a vector of strings
//  storing the words from the line

vector<string>
break_up_line( const string& a_line )
{
  //  To be implemented in lecture
```

```
}


//   For each word from the input line add to the map, increasing the
//   word's count by 1 (implicitly starting from 0 if the word was not
//   in the map).

void add_to_word_counts( vector<string> const& line_words,
                         map<string,int> & word_counts )
{
  for ( vector<string>::const_iterator p = line_words.begin(); p != line_words.end();
    word_counts[ *p ] ++ ;
}
```