```
// Project 5 solution

  // The following line makes Visual C++ shut up; clang++ and g++ ignore it.
#define _CRT_SECURE_NO_WARNINGS

#include <cstring>
#include <cctype>
using namespace std;

const int MAX_WORD_LENGTH = 20;
const int MAX_DOCUMENT_LENGTH = 250;

  // Convert word to lower case, returning true if the word is non-empty
  // and contains only letters.
bool makeWordProper(char s[])
{
    if (s[0] == '\0')  // Is word empty?
        return false;
    for (int k = 0; s[k] != '\0'; k++)
    {
        if (isalpha(s[k]))
            s[k] = tolower(s[k]);
        else // Any non-alphabetic character means the word is bad.
            return false;
    }
    return true;  // Everything was OK
}

bool isPairSame(const char aw1[], const char aw2[], const char bw1[], const char bw2[])
{
    return (strcmp(aw1, bw1) == 0  &&  strcmp(aw2, bw2) == 0);
}

int makeProper(char word1[][MAX_WORD_LENGTH+1],
               char word2[][MAX_WORD_LENGTH+1],
               int separation[],
               int nPatterns)
{
    if (nPatterns <= 0)
        return 0;
    int p = 0;
    while (p < nPatterns)
    {
        bool retainThisPattern = true;
        if (separation[p] < 0  ||  ! makeWordProper(word1[p])  ||
                                              ! makeWordProper(word2[p]))
            retainThisPattern = false;
        else
        {
              // Does an earlier pattern have the same words?  (There will
              // be at most one, since we will have previously eliminated
              // others.)
            for (int p2 = 0; p2 < p; p2++)
            {
                if (isPairSame(word1[p2], word2[p2], word1[p], word2[p]) ||
                    isPairSame(word1[p2], word2[p2], word2[p], word1[p]))
                {
                      // Words match, so retain in position p2 the pattern
                      // with the greater separation.
                    if (separation[p2] < separation[p])
                        separation[p2] = separation[p];
                    retainThisPattern = false;
                    break;
                }
            }
        }
        if (retainThisPattern)
            p++;  // go on to the next pattern
        else
        {
              // Copy the last pattern into this one.  Don't increment p,
              // so that we examine that pattern on the next iteration.
            nPatterns--;
            separation[p] = separation[nPatterns];
            strcpy(word1[p], word1[nPatterns]);
            strcpy(word2[p], word2[nPatterns]);
        }
    }
    return nPatterns;
}

int rate(const char document[],
         const char word1[][MAX_WORD_LENGTH+1],
         const char word2[][MAX_WORD_LENGTH+1],
         const int separation[],
         int nPatterns)
{
      // Get document words

      // There can't be more than this many words.  (Worst case is a
      // document like "a a a ... a"
    const int MAX_DOC_WORDS = (1+MAX_DOCUMENT_LENGTH) / 2;

      // We'll store the document words here.  If a document word is more
      // than MAX_WORD_LENGTH letters long, we'll store only the first
      // MAX_WORD_LENGTH+1 letters; since the long word can't possibly
      // match a pattern word (which is limited to MAX_WORD_LENGTH
      // characters), we'll store only enough to ensure we don't match.
    char docWord[MAX_DOC_WORDS][MAX_WORD_LENGTH+1+1];
```

```
  // Visit each character of the document, transferring letters into
  // the docWord array.
int nDocWords = 0;
int docWordPos = 0;
for (int pos = 0; document[pos] != '\0'; pos++)
{
    if (isalpha(document[pos]))
    {
          // Append letter to the end of the current docWord
        if (docWordPos < MAX_WORD_LENGTH+1)
        {
            docWord[nDocWords][docWordPos] = tolower(document[pos]);
            docWordPos++;
        }
    }
    else if (document[pos] == ' ')
    {
          // If a word was started, mark the end and prepare for the next
        if (docWordPos > 0)
        {
            docWord[nDocWords][docWordPos] = '\0';
            nDocWords++;
            docWordPos = 0;
        }
    }
      // Non-letter, non-blank characters aren't transferred and don't
      // end a word, so do nothing with them.
}
  // If the document didn't end with a blank, end the last word
if (docWordPos > 0)
{
    docWord[nDocWords][docWordPos] = '\0';
    nDocWords++;
}

  // Count matching patterns

int nMatches = 0;

  // For each pattern
for (int p = 0; p < nPatterns; p++)
{
      // For each occurrence of the w1 word
    for (int pos1 = 1; pos1 < nDocWords; pos1++)
    {
        if (strcmp(docWord[pos1], word1[p]) != 0)
            continue;

          // Find the w2 word within the separation
        int pos2 = pos1 - separation[p] - 1;
        if (pos2 < 0)  // Don't start before the first word of the doc.
            pos2 = 0;
        int end = pos1 + separation[p] + 1;
        if (end >= nDocWords)  // Don't end after the last word of the doc.
            end = nDocWords - 1;
        for ( ; pos2 <= end  &&
                  (pos2 == pos1  ||  strcmp(docWord[pos2], word2[p]) != 0);
                                                                  pos2++)
            ;
        if (pos2 <= end)  // found
        {
            nMatches++;
            break;  // Don't find any more matches for this pattern.
        }
    }
}
return nMatches;
}
```