Project Description:

Linguists have often noticed that natural languages can be compressed, but they are still fairly easy to understand because humans are fairly good at pattern seeking. For instances, if I mistyped the phrase "in this issue here" as "in thissue here", you can still understand what I was trying to say.
For this project, you will need to write a tool that can take a sentence, and squeeze the words that have the same ending as the beginning of the next word.
The input (from standard input) will consist of a sentence on each line, you can assume that single spaces separate each word. Squeeze each individual line and output the result (each on their own line). You should not squeeze across lines. Your program should process all input until the End-Of-File (EOF) is encountered.

```
Example Input:
Ferrets are really super cool.
But they aren't great at eating dessert.
I like triking around campus with my dog Mal but he likes scaring squirrels.
Things can get tricky yet, time me means answer error.

Expected Output:
Ferrets areally super cool.
Buthey aren't greating dessert.
I like triking around campus with my dog Mal but he likescaring squirrels.
Things can getrickyet, timeanswerror.
```

Your program should be divided into short, clear, legible functions. It is up to you to determine what these sub-component functions should be, with one exception. You need to write a function, named “NumberOfCharactersOverlap“ that takes two strings and returns the number of characters that overlap from the end of one word to the beginning of word. For example, `NumberOfCharactersOverlap(“VSCode“, “developer“)` should return `2`, because the “de” is at the end of the left word matches the beginning of the right word. This function will be useful in determining if two words can be squeezed together. You should (of you want full credit for this project) be writing additional functions beyond this one.

```cpp
#include <string>
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <iterator>
#include <sstream>

int NumberOfCharactersOverlap(std::string const &lhs, std::string const &rhs) {
    int upBound = static_cast<int>(std::min(lhs.length(), rhs.length()));
    bool match;

    // check end elements of lhs and beginning elements of rhs
    for (int overlap = upBound; overlap > 0; --overlap) {
        match = std::equal(
            lhs.end() - overlap,
            lhs.end(),
            rhs.begin()
        );

        if (match) { return overlap; }
    }

    return 0;
}

std::string CombineString(std::string const &curr, std::string const &add) {
    int overlap = NumberOfCharactersOverlap(curr, add);

    if (overlap == 0) { return curr + " " + add; }
    else { return curr + add.substr(overlap); }
}

std::string CompressLine(std::string const &line) {
    std::istringstream iss(line);

    return std::accumulate(
        std::istream_iterator<std::string>(iss),
        std::istream_iterator<std::string>(),
        std::string(),
        CombineString
    );
}

int main() {
    // read in lines to compress
    std::vector<std::string> lines;
    std::string line;

    while (std::getline(std::cin, line)) {
        lines.push_back(line);
    }

    // compress lines and output
    std::transform(
        lines.begin(),
        lines.end(),
        std::ostream_iterator<std::string>(std::cout, "\n"),
        CompressLine
    );

    return 0;
}
```

Will’s Solution

```
#include <iostream>
#include <string>
#include <sstream>
using std::istringstream;
using std::ostringstream;
using std::string;
using std::cin;
using std::cout;
using std::endl;

int NumberOfCharactersOverlap(const string &word1, const string &word2) {
    string temporary1, temporary2;
    int result = 0;
    for (int i = word1.size()-1; i >= 0; --i) {
        temporary1 = word1.substr(i, string::npos);
        for (int j = 1; j < static_cast<int>(word2.size()+1); ++j){
            temporary2 = word2.substr(0, j);
            if (temporary1.compare(temporary2) == 0) {
                result = j;
            }
        }
    }
    return result;
}

void Combine(bool &combined, ostringstream &oss, string &left, string &right) {
    int overlapNumber = NumberOfCharactersOverlap(left, right);
    if (overlapNumber != 0){
      left += right.substr(overlapNumber);
      combined = true;
    } else {
      oss << left << " ";
      left = right;
      combined = false;
    }
}

void Output(bool &combined, ostringstream &oss, string &left, string &right) {
  if (combined){
    oss << left << " ";
  } else if (right == "") {
    oss << left;
  } else {
    oss << right << " ";
  }
}

string Compress(const string &line){
  istringstream iss(line);
  ostringstream oss;
  string left, right;
  bool combined = false;
  iss >> left;
  while (iss >> right) {
    Combine(combined, oss, left, right);
  }
  Output(combined, oss, left, right);
  return oss.str();
}

int main() {
  string line;
  while (getline(cin, line)){
    cout << Compress(line) << endl;
  }
}
```

Jennifers solution

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

```cpp
template <typename I>
auto OverlapPoint(I begin_first, I end_first, I begin_second, I end_second)
{
  // start at the earliest possible overlap in the first range
  auto i =
      end_first - std::min(end_first - begin_first, end_second - begin_second);

  // return the first possible complete match
  for (; i != end_first; ++i)
    if (std::equal(i, end_first, begin_second))
      break;

  return i;
}

template <typename I>
auto SqueezeLine(I begin, I end)
{
  auto space = std::find(begin, end, ' ');

  if (space == end)
    return end; // if no space is found, we're done squeezing

  // find the best overlap point indicated by the space
  auto overlap = OverlapPoint(begin, space, space + 1, end);

  if (overlap == space)
    begin = space + 1; // if no overlap is found, go to the next word
  else
    end = std::move(space + 1, end, overlap); // otherwise squeeze the overlap

  return SqueezeLine(begin, end); // squeeze what remains, recursively
}

int NumberOfCharactersOverlap(std::string const &first,
                              std::string const &second)
{
  return std::distance(OverlapPoint(std::begin(first), std::end(first),
                                    std::begin(second), std::end(second)),
                       std::end(first));
}

int main()
{
  for (std::string line; std::getline(std::cin, line);)
  {
    // erase-remove idiom to efficiently remove squeezed letters
    line.erase(SqueezeLine(std::begin(line), std::end(line)), std::end(line));
    std::cout << line << std::endl;
  }
}
```

Nitash Solution

```cpp
// Author Francis Kasmikha
// CSE 232, Section 07
```

```cpp
#include <iostream>
using std::cin;
using std::cout;
#include <vector>
using std::vector;
#include <string>
using std::string;

// This function takes in a string that we recieve from the user,
// It will then take the string and find each word until whitespace, when it
// reaches whitespace it will then take the word
// It will put that word into an element of the vector. In the end we will have
// a vector called v that turned the input into seperate words
// https://slaystudy.com/c-split-string-by-space-into-vector/
// This is where I got this function from, I did not change it and it works very
// well

void SplitString(string s, vector<string>& v) {
  string temp = "";
  for (int i = 0; i < static_cast<int>(s.length()); ++i) {
    if (s[i] == ' ') {
      v.push_back(temp);
      temp = "";
    } else {
      temp.push_back(s[i]);
    }
  }
  v.push_back(temp);
}
// This function will find the occurence of the letter in the word based on what
// I put in.
// For example If I would like to get the occurence of e the second time it is
// shown in there
// I would There for str, and e for char, and then 2 for my nth integer. returns
// 4 due to the index value of the second e
// This is helpful because later in my code I have a function that will find the
// last occurence of a letter
// I use that to help me test every occurence of the letter to see if it
// matches.
// https://stackoverflow.com/questions/18972258/index-of-nth-occurrence-of-the-string
// I got this from this website, and I did change the function a bit because it
// did not work how I wanted it to
int nthOccurrence(const std::string& str, const char& findMe, int nth) {
  size_t pos = 0;
  int cnt = 0;
  if (nth == 1) {
    int result = str.find(findMe);
    return result;
  } else if (nth == 0) {
    return -1;
  }
  while (cnt != nth) {
    pos = str.find(findMe, pos);
    cnt++;
    pos += 1;
  }
  return pos - 1;
}
// This function takes in a string and a character, it will find the very last
// occurence of the letter in the word.
// This will how many times a letter is in a word. For example for there, e is
// in the word twice. It will return 2 then.
// How I have it set up basically the count is finding the 1st occurence of the
// word, then it will find the 2nd occurence.
// It will keep looping, until -1 is returned, when that is returned it will
// cause a flag to go off and I will save the value
// of times the letter appeared.
//
int Occurrence(const std::string& str, const char& findMe) {
  int count = 1;
  int indexval = nthOccurrence(str, findMe, count);
  while (indexval >= 0) {
    indexval = nthOccurrence(str, findMe, count);
    if (indexval == -1) {
      return count - 1;
    }
    count++;
  }
  return count;
}
// This function is taking our two words we would like to Merge, and it will
// return the final word.
// How it acheives this is by taking our first word and erasing however many
// letters should be combined
// for example abab bab, we have 4 letters in first word, and then we take away
// 3 letters, and add on the second word.
// Then it will return the final word

string Merge(string s1, string s2, int letsize) {
  int size_s1 = (static_cast<int>(s1.size()));
  int indexval = size_s1 - letsize;
  s1.erase(indexval);
  string finalstr = s1 + s2;
  return finalstr;
}
// This will take in the words we want to look for overlapped words in, then the
// charcater that they will use to see if they can merge.
// It also takes in which occurence of the letter we would like to look for, It
// will first do the very last time it is there and
// then it will give how ever many characters overlap in a string and make a new
// string. This is helpful because if this is equal
// to the size of how ever many letters we toke from the second word then it
// will use the value

string OverlapedWords(string s1, string s2, char letter, int last_occured) {
  int indexval_s2 = nthOccurrence(s2, letter, last_occured);
  int size_s2 = (indexval_s2 + 1);
  int indexval_s1 = (static_cast<int>(s1.size()) - 1);
  int size_s1 = (static_cast<int>(s1.size()));
  string newstr = "";
  if (size_s2 < 1) {
    size_s1 = 0;
  }
  for (int i = 0,j=0; j < size_s1; i++, j++) {
    if (s1.at(indexval_s1) == s2.at(indexval_s2)) {
      newstr.push_back(s1.at(indexval_s1));
    }
    indexval_s1--;
    indexval_s2--;
    if ((indexval_s2 < 0) || (indexval_s1 < 0)) {
      size_s1 = 0;
      indexval_s2 = 0;
    }
  }
  return newstr;
}
// Finally we have approched my Number of Characters function, there is not too
// much to look at here
// It will first see if the strings are empty, and if they are not then it will
// continue
// After checking if the strings are empty it will get the character at the last
// index value in our first string.
// After getting this character it will check the occurence of the character at
// the last location in our second word.
// It will then check that word and see if all the letters before the index
// value match with the first word
// If the letters do not match, it will trying searching for the occurence of
// the word again. but this time in one less spot.
// then it will try and see if it matches, and will continue looping until the
// words are matching and if not it will return 0.
int NumberOfCharactersOverlap(const string& s1, const string& s2) {
  if (!s1.empty() && !s2.empty()) {
    char last_occur_char = s1.at(static_cast<int>(s1.size()) - 1);
    int last_occured = Occurrence(s2, last_occur_char);
    while (last_occured > 0) {
      string newstr = OverlapedWords(s1, s2, last_occur_char, last_occured);
      int size_s2 = ((nthOccurrence(s2, last_occur_char, last_occured)) + 1);
      int finalsize = (static_cast<int>(newstr.size()));
      if ((size_s2 > 0) && (finalsize == size_s2)) {
        return finalsize;
      }
      last_occured--;
    }
    return 0;
  }
  return 0;
}

// in my main function, It will ask the user for an input and then it will take
// that string and make it a vector, using that vector
// Check to see if any characters overlap in both words and if they do overlap
// then I will change the value inside the vector
// and the for loop will continue going through each element in the vector
// checking for any characters that overlap.
int main() {
  string string_user;
  while (getline(cin, string_user)) {
    vector<string> v;
    SplitString(string_user, v);
    for (int i = 0; i < (static_cast<int>(v.size()) - 1); i++) {
      string word1 = v[i];
      string word2 = v[i + 1];
      int overlap = NumberOfCharactersOverlap(word1, word2);
      if (overlap > 0) {
        string finalstring = Merge(word1, word2, overlap);
        v[i + 1] = finalstring;
        v.erase(v.begin() + i);
        if (i >= 0) {
          i--;
        }
      }
    }
    for (int j = 0; j < static_cast<int>(v.size()); ++j) cout << v[j] << " ";
    cout << "\n";
  }
  return 0;
}
```

My solution