Computer Programming Olympiad

A project of the Institute of IT Professionals South Africa
Ph: 021-448 7864 • Fax: 021-447 8410 • PO Box 13013, MOWBRAY, 7705 • info@olympiad.org.za • www.olympiad.org.za

COMPUTER PROGRAMMING OLYMPIAD

2016

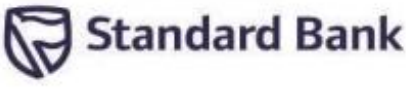
ROUND 2

POSSIBLE SOLUTIONS

Supported by Oracle and the University of Cape Town.

ORACLE ACADEMY

Sponsored by



NOTE:

Solutions to the problems have been tested using the programming languages and IDEs listed below. Those languages and IDE's identified with an asterisk are those that are used during the International Olympiad in Informatics (IOI).

	IDE	Language Version
C++ solutions	jGrasp 2.0.2_02	GCC 4.6.3*
Delphi solution	Delphi	2010
Java solutions	jGrasp 2.0.2_02	Java 1.8.0*
Pascal solutions	Lazarus 1.4.2*	FPC 2.6.4*
Python solutions	Idle	Python 3.4.3*

CONTRIBUTORS:

Max Brock	IT Curriculum Adviser: Western Cape Education Department
Shamiel Dramat	IT Curriculum Adviser: Western Cape Education Department
OER Foss	Educator and open-source advocate
Robert Spencer	Programming Olympiad medal winner: Bronze (2010, 2011) and Gold (2012) International Olympiad in Informatics (IOI): Bronze medal winner (2013), deputy leader (2014, 2015) and delegation leader (2016)
Robin Visser	Programming Olympiad medal winner: Bronze (2013) and Silver (2014) International Olympiad in Informatics (IOI): Bronze medal winner (2015), deputy leader (2016)

CODED SOLUTIONS:

Coded solutions to each of the questions using each of the above programming languages can be found by navigating to the following Dropbox folder:

https://www.dropbox.com/sh/fztvhe5saqp3nj0/AAAshfGT4maQ0dEV_vtVD0Hea?dl=0

The solutions can be downloaded to your computer by clicking on the "Download" button top right of the screen.

QUESTION 1: PALINDROMIC SUM

Adding any natural number to its reverse and repeating the operation with the resulting sum will, in nearly all cases, eventually give a palindromic number. (A palindromic number is a number which reads the same from right to left or from left to right. 323 is palindromic while 321 is not).

For example:
38 + 83 = 121 (a palindromic number in one step)

69 + 96 = 165
165 + 561 = 726
726 + 627 = 1353
1353 + 3531 = 4884 (a palindromic number in 4 steps)

Write a program that asks for a natural number and then calculates how many steps it will take to convert that number into a palindromic number as described above.

Examples:
Input: 38
Output: 1

Input: 69
Output: 4

Test your program with the following and type or paste the answers in the correct block on the web page:

a) 42
b) 527
c) 739
d) 89

[For the mathematicians: It is not known if this procedure will ever terminate for the number 196.]

Answers:

a) 1
b) 2
c) 17
d) 24

SAMPLE C++ SOLUTION

```
#include <iostream>
#include <string>
using namespace std;

int num_digits(long long x)
{
  int ans = 1;
  long long t = 10;
  while (x>=t)
  {
    ans++;
    t*=10;
  }
  return ans;
}

long long reverse(long long x)
{
  long long ans = 0;
  long long d = 1;
  long long s = 1;
  for (int i = 0;i < num_digits(x)-1; ++i) d*=10;
  while (d>0)
  {
    ans += s * ((x/d)%10);
    s *= 10;
    d /= 10;
  }
  return ans;
}

int main()
{
  long long N;
cout<<"Enter the number:  ";
cin>>N;
  int count = 0;
  while (N != reverse(N))
  {
    N = N + reverse(N);
    count++;
  }
  cout<<count<<endl;
}
```

[Adapted by OER Foss from the original solution produced by Robert Spencer]

SAMPLE JAVA SOLUTION

```
import java.util.Scanner;

public class PalindromicSum {
  public static void main(String[] args) {
    int count = 0;
    Scanner keybd = new Scanner(System.in);
    System.out.print("Enter the number: ");
    int input = keybd.nextInt();
    long sum = input;
    do {
      sum = sum + Long.parseLong(reverse(sum + ""));
      count++;
    } while (!isPalindrome(sum + ""));
    System.out.println("Count: " + count);
  }

  static String reverse(String in) {
    String re = "";
    for (int i = in.length() - 1; i >= 0; i--) {
      re = re + in.charAt(i);
    }
    return re;
  }

  static boolean isPalindrome(String s) {
    String p1 = "";
    String p2 = "";
    p1 = s.substring(0, s.length() / 2);
    if (s.length() % 2 == 0) {
      p2 = s.substring(s.length() / 2);
    }
    else {
      p2 = s.substring(s.length() / 2 + 1);
    }
    return p1.equals(reverse(p2));
  }
}
```

[Adapted by OER Foss from original solution produced by Max Brock]

SAMPLE PASCAL SOLUTION (USING CONSOLE MODE)

```
program PalindromicSum;   {$APPTYPE CONSOLE}

uses
   SysUtils;

var
   input, count: integer;
   sum: int64;

function reverse(s: string): string;

var
   i: integer;
   re: string;

begin
   for i := length(s) downto 1 do
      re := re + copy(s, i, 1);
   result := re;
end;

function isPalindromic(s: string): boolean;

var
   p1, p2: string;
   half: integer;

begin
   half := length(s) div 2;
   p1 := copy(s, 0, half);
   if (odd(length(s))) then
      p2 := copy(s, half + 2, half)
   else
      p2 := copy(s, half + 1, half);
   result := p1 = reverse(p2);
end;

begin
   try
      input := 89;
      count := 0;
      sum := input;
      repeat
         sum := sum + strtoint64(reverse(inttostr(sum)));
         count := count + 1;
```

```
      until (isPalindromic(inttostr(sum)));
      writeln(count);
      readln;
   except
      on E: Exception do
        writeln(E.ClassName, ': ', E.Message);
   end;
end.
```

[Adapted by OER Foss from original solution produced by Max Brock]

SAMPLE PYTHON SOLUTION

```
#!/usr/bin/env python3

n = input("Input: ")

step_counter = 0
while(n != n[::-1]):
   n = str(int(n) + int(n[::-1]))
   step_counter += 1

print("Output:", step_counter)
```

[Solution produced by Robin Visser]

QUESTION 2: AD INFINITUM

When converting ordinary fractions into decimal fractions one sometimes ends up with a set of decimal digits that keep on repeating (so-called recurring decimals). For example ⅓ = 0.3333333 etc.

Write a program that will convert an ordinary fraction into a decimal fraction and that places the recurring digits in brackets.

For example:

22/5 = 4.4 (no recurring decimal digits so no brackets)
1/3 = 0.(3) (three recurs ad infinitum)

Examples:
Input: Fraction? 1/3
Output: 0.(3)

Input: Fraction? 9/7
Output: 1.(285714)

Test your program with the following and type or paste the answers in the correct block on the web page:

a) 3/8
b) 3/7
c) 99/43
d) 45/46

Answers:

a) 0.375
b) 0.(428571)
c) 2.(302325581395348837209)
d) 0.9(782608695652173913043 4)

A possible method for obtaining the answer:

The solution can easily be found by reverting back to long division of old. The trick in solving this problem is to keep track of the remainders in each division because as soon as a remainder that one has seen before occurs the sequence of answers and remainders starts repeating itself.

Let's take the fraction 19/33 as an example.

Step 1: dividing 19/33 gives an answer of 0. with a remainder of 19. Keep a record of this remainder, in a list or an array {19}.

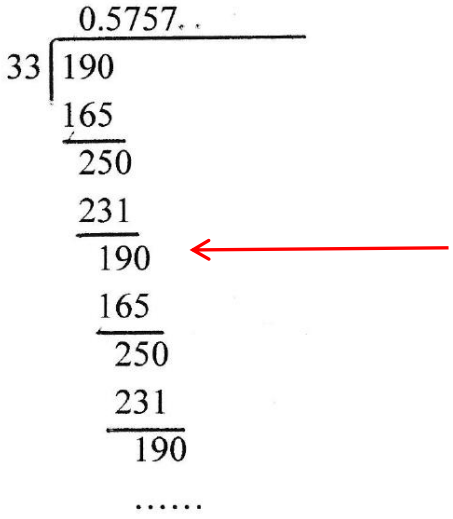
Step 2: multiply the remainder from the above step, i.e. 19, by 10 and divide the answer by 33. This gives an answer of 5 with a remainder of 25. Keep a record of this remainder {19 25}.

Step 3: multiply the remainder from the above step, i.e. 25, by 10 and divide the answer by 33. This gives an answer of 7 with a remainder of 19. We've already seen this remainder (see list in step 2

above) so if we continue with the long division we will do what we did in step 2 followed by what we did in step 3 and so on.

So as soon as the remainder of any of the divisions in a step is the same as one of the remainders already noted, the decimals in the answer will start repeating (recurring) so we need to immediately stop the long division. Studying the example below, one notes that after the point indicated by the arrow the values start repeating themselves and will do so ad infinitum.

In this case the answer will be 0.57575757.. or 0.(57)



SAMPLE C++ SOLUTION

```
#include <iostream>
#include <map>
#include <vector>
using namespace std;

int main()
{
  int numerator, denominator;
  cout<<"Numerator? ";
  cin>>numerator;
  cout<<"Denominator? ";
  cin>>denominator;

  map<int,int> remainders;
  vector<int> digits;

  cout<< numerator/denominator<<'.';
  numerator %= denominator;

  int i = 0;
  while (remainders.count(numerator) == 0)
```

```
 {
  remainders[numerator] = i++;
  numerator *= 10;
  digits.push_back(numerator/denominator);
  numerator %= denominator;
 }
 if (numerator == 0)
  digits.pop_back();

 for (int i = 0;i<digits.size();++i)
 {
  if (numerator!=0)
   if (remainders[numerator]==i)
    cout<<'(';
  cout<<digits[i];
 }
 if (numerator!=0)
  cout<<')';
 cout<<endl;
}
```

[Solution produced by Robert Spencer]

SAMPLE JAVA SOLUTION

```
import java.util.Scanner;
import java.util.ArrayList;

public class AdInfinitum {
  public static void main(String[] args) {
    ArrayList modList= new ArrayList();
    boolean done = false;
    int mod;
    int pos;
    int digit;
    Scanner keybd = new Scanner(System.in);
    System.out.print("Enter the numerator: ");
    int numerator = keybd.nextInt();
    System.out.print("Enter the denominator: ");
    int denominator = keybd.nextInt();
    String answer = "" + numerator / denominator + ".";
    mod = numerator % denominator;
    int lengthToDot = answer.length();
    do {
      modList.add(mod);
      digit = (mod * 10)/denominator;
      answer += digit;
```

```
        mod = (mod * 10) % denominator;
        if(modList.contains(mod)){
          done = true;
          pos = modList.indexOf(mod);
          answer = answer.substring(0, lengthToDot + pos) + "(" + answer.substring(lengthToDot + pos) +
")";
          System.out.println(answer);
        }
        else if(mod == 0){
          done = true;
          System.out.println(answer);
        }
      } while (!done);
   }
}
```

[Adapted by OER Foss from original solution produced by Max Brock]

SAMPLE PASCAL SOLUTION (USING CONSOLE MODE)

```
program AdInfinitum;   {$APPTYPE CONSOLE}

uses
   SysUtils;

var
   numerator, denominator: integer;
   m, p, d, l, remIndex, count: integer;
   done: boolean = false;
   strFraction: string;
   answer: string = '';
   remainders : array[1..100] of integer;

begin
   try
      write('Enter fraction, e.g. 3/8: ');
      readln(strFraction);
      p := pos('/', strFraction);
      numerator := StrToInt(copy(strFraction, 1, p-1));
      denominator := StrToInt(copy(strFraction, p+1, length(strFraction)));
      answer := IntToStr(numerator div denominator) + '.' ;
      m := numerator mod denominator;
      l := length(answer);
      remIndex := 1;
      p := 0;
      while (not done) do
         begin
```

```
            remainders[remIndex] := m;
            inc(remIndex);
            d := (m * 10) div denominator;
            answer := answer + IntToStr(d);
            m := (m * 10) mod denominator;
            for count := 1 to remIndex - 1do
               begin
                  if (m = remainders[count]) then
                     begin
                        p := count;
                     end;
               end;
            if (p > 0) then
               begin
                  done := true;
                  answer := copy(answer, 1, l + p - 1) + '(' + copy(answer, l + p, length(answer)) + ')';
                  writeln(answer);
               end
            else
               if (m = 0) then
                  begin
                     done := true;
                     writeln(answer);
                  end;
         end;
      readln;
  except
     on E: Exception do
        writeln(E.ClassName, ': ', E.Message);
   end;
end.
```

[Produced by OER Foss, based on the Java solution produced by Shamiel Dramat]

SAMPLE PYTHON SOLUTION

```
#!/usr/bin/env python3

# Input
frac = input("Input fraction: ")
numerator = int(frac.split('/')[0])
denominator = int(frac.split('/')[1])

out_buffer = str(numerator//denominator)
decimalstring = ""
lastseen = {}    # Creates map of when remainders where last seen

# Calculate remainders repeatedly
remainder = numerator%denominator
if remainder != 0:
    out_buffer += "."

    i = 0
    while(remainder != 0):
        decimalstring += str((remainder*10)//denominator)

        if remainder in lastseen:
            # We have a repeat
            out_buffer += decimalstring[:lastseen[remainder]] # Add non-repeat
            out_buffer += "("
            out_buffer += decimalstring[lastseen[remainder]:len(decimalstring)-1]
            out_buffer += ")"
            break
        # Calculate new remainder and store
        lastseen[remainder] = i
        remainder = (remainder*10)%denominator
        i += 1

    # Non-repeating case
    if (remainder == 0):
        out_buffer += decimalstring

print("Output:", out_buffer)
```

[Solution produced by Robin Visser]

QUESTION 3: WORD CHAIN

A *word chain* is an ordering of a set of words such that each word is exactly one letter different from the previous word.

For example, in the set of words {CARE HARE CART}, CARE and CART only differ in the last letter, and HARE and CARE differ in the first. Thus a valid word chain for this set of words might be HARE CARE CART

Write a program that, given a set of such words, outputs a valid word chain.

Output the word chain as a sequence of words (in the correct order) separated by one space.

If there is no correct answer, output the word "impossible".

For test case (a), if there are multiple valid chains, you may output any one of them.

For test cases (b), (c) and (d), if there are multiple valid chains, output the chain (string) that comes first alphabetically.

Examples:

Input: DOG CAT COW
Output: Impossible

Input: CARE HARE CART
Output: CART CARE HARE
[Note: HARE CARE CART would also be a valid word chain, but does not come first alphabetically.]

Input: PAT BUT PET BUF POT RAT PUT
Output: BUF BUT PUT PET POT PAT RAT
[Note: BUF BUT PUT POT PET PAT RAT would also be a valid word chain, but does not come first alphabetically; The two strings are identical until the 11^{th} letter; then the one has an E and the other an O. E comes before O alphabetically.

Test your program with the following and type or paste the answers in the correct block on the web page:

a) POP MAP MOP

b) BOB HOT FOB COT FOR LOT NOT NOR

c) CUT HOP COT CAR BAT HUT CAT HAT BAR

d) WALL TAIL TALL WALK BALL WAIL TALK

Answers:

a) MAP MOP POP or POP MOP MAP
b) BOB FOB FOR NOR NOT COT HOT LOT
c) impossible
d) BALL TALL TAIL WAIL WALL WALK TALK

A possible method for obtaining the answer:

Step 1: sort the words in ascending order.

Step 2: for each word in the sorted list of words taken in alphabetical order, do the following

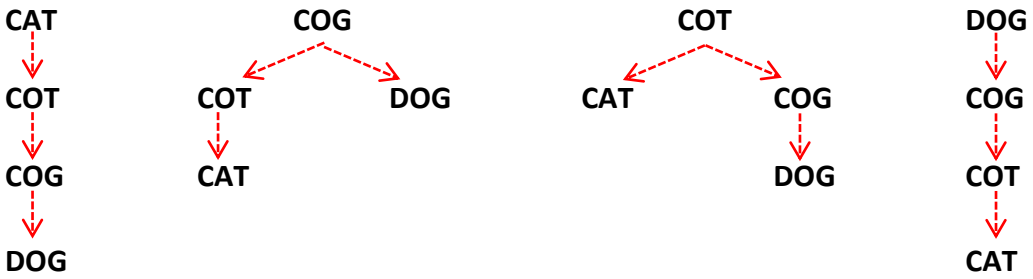
- Create a new list of words that starts with the selected word from the sorted list of words
- Compare this word with the other words in the sorted list of words, in the same order. You may need to repeat this a few times until either all the original words appear in the chain (in other words the length of the new list of words is the same length as the original list of words) or no more words can be added to the chain.
  - = If the two words differ by one letter then add the word to the new list of words. The word that has been added now becomes the word with which other words in the original list are to be compared. Note that any words appearing in the new list should be skipped in this comparison process.
  - = If the two words differ by more than one letter ignore the word and move on to the next word in the sorted list of words

Step 3: having followed step 2 you could have at least one new list of words that contains all of the original words. If there is no new list of words that contains all of the original words, then it is not possible to form a word chain that satisfies the criteria.

Step 4: in the event you have two or more new lists of words that contain all of the original words then these lists need to be sorted in alphabetical order. The list that occurs first in the ordered set of lists is the word chain that satisfies the specifications.

Example 1

Original list of words: COG DOG CAT COT



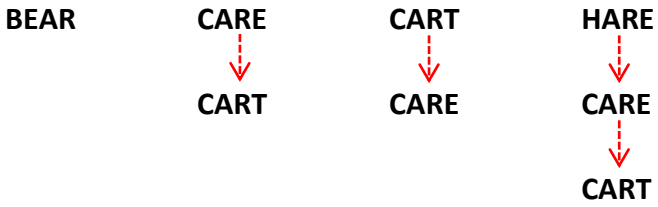
Six word chains can be created from the original four words, only two of which contain all four of the original words, viz.

- CAT COT COG DOG
- DOG COG COT CAT

Of these the first word chain would come before the second word chain in the "dictionary" (because the letter C in CAT comes before the letter D in DOG in the dictionary) and so the first word chain is the answer to the question.

Example 2:

Original list of words: CARE BEAR HARE CART



In this example three word chains are created out of the original four words but none of these word chains contain all four of the original words. This being the case the requirements of the question are not met and so there is no answer, or it is impossible to create a word chain containing all four of the original words.

SAMPLE C++ SOLUTION

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

using namespace std;

bool adjacent(string a, string b)
{
  if (a.length() != b.length()) return false;
  int wrong = 0;
  for (int i = 0; i < a.length(); ++i)
    if (a[i] != b[i])
      wrong++;
  return wrong==1;
}
vector<bool> visited;
vector<int> path;
vector<int> *neigh;
int N=1;

bool dfs(int curr,int depth)
{
  visited[curr] = true;
  if (depth==N-1)
  {
    path.push_back(curr);
    return true;
```

```
  }
  for (int i = 0; i<neigh[curr].size(); ++i)
    if (not visited[neigh[curr][i]])
      if (dfs(neigh[curr][i],depth+1))
      {
        path.push_back(curr);
        return true;
      }
  visited[curr] = false;
  return false;
}

int main()
{
  cout<<"Enter words separated by one space each"<<endl;
  string input;
  getline(cin,input);
  std::transform(input.begin(), input.end(), input.begin(), ::toupper);
  for (int i = 0;i<input.length();++i)
    if (input[i]==' ')
      N++;
  visited.resize(N,false);
  vector<string> words (N);
  stringstream sstream;
  sstream << input;
  int i = 0;
  while(sstream >> words[i]) i++;
  sort(words.begin(),words.end());

  neigh = new vector<int>[N];
  for (int i = 0;i<N;++i)
  {
    for (int j = 0;j<N;++j)
      if (adjacent(words[i],words[j]))
        neigh[j].push_back(i);
  }
  for (int i = 0;i<N;++i)
  {
    if (dfs(i,0))
    {
      for (int j = 0;j < N;++j)
        cout<<words[path[N-j-1]]<<' ';
      cout<<endl;
      return 0;
    }
  }
```

```
 cout<<"impossible"<<endl;
}
```

[Adapted by OER Foss from the original solution produced by Robert Spencer]

SAMPLE JAVA SOLUTION

```
import java.util.Scanner;
import java.util.ArrayList;
import java.util.Arrays;

public class WordChain {

  private static    boolean differsByOne(String word1, String          word2)
  {
    int count =    0;
    if     (word1.equals(word2))
    {
      return false;       }
    for (int   i = 0; i    < word1.length();       i++)
    {
      if  (word1.charAt(i) != word2.charAt(i))
      {         count++;           }
      if  (count > 1)
      {
        return false;    }
    }
    return true;
  }

  public static void main(String[]      args)    {
    Scanner keybd = new Scanner(System.in);
    System.out.print("Enter the sequence of words separated by a space: ");
    String input =       "";
    input      = keybd.nextLine();
    String[]  words     = input.split(" ");
    Arrays.sort(words);
    ArrayList chained= new   ArrayList();
    int startingPoint   = 0;
    boolean done =     false;
    chained.add(words[startingPoint]);
    String word1 =     words[startingPoint];
    boolean nextWordFound =          false;
    do    {
      for (intj = 0; j    < words.length; j++)
```

```
      {
        if     (differsByOne(word1,    words[j]) && !chained.contains(words[j]))
        {
          chained.add(words[j]);
          word1   = words[j];
          nextWordFound = true;
          break;
        }
      }
      if(chained.size()      == words.length)
      {
        done = true;
      }
      else
      {
        if(!nextWordFound    &&     startingPoint != words.length-1)
        {
          startingPoint++;
          chained.clear();
          word1   = words[startingPoint];
          chained.add(word1);
          nextWordFound = false;
        }
        if(!nextWordFound    &&     (startingPoint == words.length-1))
        {
          System.out.println("impossible");
          System.exit(0);
        }
      }
      nextWordFound = false;
    } while (!done);
    System.out.println("");
    for (Object    word : chained)
    {System.out.print(word + " ");
    }
    System.out.println("");
    System.exit(0);
  }
}
```

[Adapted by OER Foss from original solution produced by Shamiel Dramat]

SAMPLE PASCAL SOLUTION (USING CONSOLE MODE)

```
program WordChain;      {$APPTYPE  CONSOLE}

uses
   SysUtils;

var
   i, j, noWords, p: integer;
   startingPoint, wordsInChain : integer;
   done : boolean = false;
   nextWordFound : boolean;
   word1, wordSeq : string;
   words : array[1..100] of string;
   chained : array[1..100] of  string;

function differsByOne (word1, word2: string): boolean;
var
   count: integer = 0;
   i: integer;
begin
   differsByOne := true;
   if (word1 = word2) then
      differsByOne := false;
   for i := 1 to length(word1) do
      begin
         if (word1[i] <> word2[i]) then
            inc(count);
         if (count > 1) then
            differsByOne := false;
      end;
end;

function contains (word: string): boolean;
var
   i: integer;
begin
     contains := false;
     for i := 1 to noWords do
        begin
           if (word = chained[i]) then
              contains := true;
        end;
end;
```

```
procedure sortWords;
var
   i, gap : integer;
   temp : string;
begin
   gap := 8;
   repeat
      for i := 1 to noWords - gap do
         if words[i] > words [i + gap] then
            begin
               temp := words[i];
               words[i] := words[i + gap];
               words[i + gap] := temp;
            end;
         gap := gap div 2;
      until gap = 0;
end;

begin
   try
      write('Enter the sequence of words separated by a space: ');
      readln(wordSeq);
      noWords := 0;
      while wordSeq <> '' do
        begin
           p := pos(' ', wordSeq);
           if (p = 0) then
              begin
                 inc(noWords);
                 words[noWords] := wordSeq;
                 wordSeq := ''
              end
           else
              begin
                 inc(noWords);
                 words[noWords] := copy(wordSeq, 1, p-1);
                 wordSeq := copy (wordSeq, p+ , length (wordSeq)-p);
              end;
         end;
      sortWords;
      startingPoint := 1;
      word1 := words[startingPoint];
      nextWordFound := false;
      wordsInChain := 1;
      chained[wordsInChain] := words[startingPoint];
      while (not done) do
```

```
        begin
          for j := 1 to noWords do
            begin
              if (differsByOne(word1, words[j]) and (not(contains(words[j])))) then
                begin
                  inc(wordsInChain);
                  chained[wordsInChain] := words[j];
                  word1 := words[j];
                  nextWordFound := true;
                  break;
                end;
            end;
          if (wordsInChain = noWords) then
            done := true
          else
            if ((not nextWordFound) and (startingPoint <> noWords - 1)) then
              begin
                inc(startingPoint);
                word1 := words[startingPoint];
                wordsInChain := 1;
                chained[wordsInChain] := word1;
                nextWordFound := false;
              end;
          if ((not nextWordFound) and (startingPoint = noWords - 1)) then
            begin
              writeln('impossible');
              readln;
              exit;
            end;
          nextWordFound := false;
        end;
      for i := 1 to wordsInChain do
        write(chained[i], ' ');
      writeln;
      readln;
    except
      on E: Exception do
        writeln(E.ClassName, ': ', E.Message);
    end;
end.
```

[Produced by OER Foss, based on the original Java solution produced by Shamiel Dramat]

SAMPLE PYTHON 3 SOLUTION

```
#!/usr/bin/env python3

import itertools

def valid_chain(case):
    """Check if given list is a valid word chain."""
    for i in range(1, len(case)):
        if (len(case[i]) != len(case[i-1])):
            return False

        num_different = 0
        for j in range(len(case[i])):
            if (case[i][j] != case[i-1][j]):
                num_different += 1

        if (num_different != 1):
            return False

    return True

# Input (as space seperated words)
chain = input("Input as space separated words: ")
chain = chain.upper()

# Try all permutations
solutions = []
brute = itertools.permutations(chain.split())
for case in brute:
    if valid_chain(case):
        solutions.append(case)

# Output
if (len(solutions) == 0):
    # No solution
    print("impossible")
else:
    # Print out first alphabetically
    solutions.sort()
    for word in solutions[0]:
        print(word, end=' ')
```

[Solution produced by Robin Visser]