```
function [str2, str1] = complement(str1)
% complement find the reverse complement of the given nucleotide string as str2
% str is the input string
%
%   See also cleanString.m

  % % For testing:
  % clc, clear all
  % str1 =↙
['ACCCCCCGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG↙
GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG↙
GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG↙
GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA↙
AA'];


  %
  %
  % For given sequence, take the complememnt and store in reverse order
  %
  %

  % Find length of input sequence
  lenStr1 = length(str1);

  % For length of given sequence
  for i = 1:lenStr1
    % Check for nucleotide base and store pair in tmp data structure
    if str1(i) == 'C'
      tmp = 'G';
    elseif str1(i) == 'G'
      tmp = 'C';
    elseif str1(i) == 'A'
      tmp = 'T';
    elseif str1(i) == 'T'
      tmp = 'A';
    else
      % If no nucleotide is detected or protein sequence is run. Return error code:
      disp('Double check that this a DNA sequence (in all CAPS) and remove any non-↙
ACTG characters. Please!')
      return % end
    end

    % Make the reverse complement by filling in the string backwards (i.e. Final↙
position - current)
    tmpPos = lenStr1 - i + 1; % '+1' is added to make the math work
    str2(tmpPos) = tmp; % Write to reverseComlement
  end

  % Use cleanstring file to chunk long sequence into readable text (30 bp long)
  [cleanStr1, cleanExtStr1] = cleanString(str1);
  [cleanStr2, cleanExtStr2] = cleanString(str2);

  %
  %
  % For reference, the m-function file is included below:
  %
  %
  % function [cleanStr2, cleanExtStr2] = cleanString(str2)
  % % complement find the reverse complement of the given nucleotide string
  % %
  % %   See also complement.m
  %   % Manipulate the length of the string
  %   lenStr2 = length(str2);
  %   lines = lenStr2/30; % arbitrary number that looked good for max width
  %   % Round the sequence
  %   ceilLines = ceil(lines);
  %   flrLines = floor(lines);

  %   %
  %   %
  %   % Truncate sequence into 30 bp shorter sequences
  %   %
  %   %

  %   % If seq chunk is less than 30, run extension to avoid range issues
  %   extension = lenStr2 - 30*flrLines; % Find remainder of dividing by 30
  %   if (extension ~= 0)
  %     jlastLast = 30*flrLines + 1; % find start of last 30 base chunk
  %     cleanExtStr2 = str2(jlastLast:(flrLines*30 + extension)); % store it
  %   else
  %     cleanExtStr2 = ''; % empty second header is divisible in 30 base chunks
  %   end

  %   % If seq chunk is = to 30, run standard analysis
  %   if flrLines >= 1 % Check to see if the sequence is longer than 30 characters
  %     jlast = 1; % initialize counter variable
  %     for i=1:flrLines
  %       % For each 30 base chunk (i)
  %       tempStr2 = str2(jlast:i*30); % length(tempStr2) % for testing
  %       cleanStr2(i, :) = tempStr2; % save short sequence as new line
  %       jlast = jlast + 30; % shift to next 30 base string
  %     end
  %   else % If not > 30 characters, just display the extension sequence
  %     cleanStr2 = cleanExtStr2; % Swap short sequence to first header
  %     cleanExtStr2 = ''; % then empty second header
  %   end
  % end


  % Display original string
  disp([13 'Given string:' 13])
  disp(cleanStr1)
  if length(cleanExtStr1) ~= 0
    disp(cleanExtStr1)
  end

  % Display output of string manipulation
  disp([13 'The reverse complement is:' 13])
  disp(cleanStr2)
  if length(cleanExtStr2) ~= 0
    disp(cleanExtStr2)
  end

  return % end
end
```