THIS IS THE DOCUMENTATION OF A PROGRAM WRITTEN IN C THAT ACCEPTS 1 TO 7 LETTER WORDS AND PRINTS OUT ALL THE POSSIBLE COMBINATION OF THAT LETTER

1.0 The following is a step by step explanation of the code.

Step 1: Accept input form user, which must not be more than 7 characters.
Step 2: Print the string.
Step 3: Obtain the length of the string(L).
Step 4: Obtain the factorial (fac). Let initial fac be (fac=L). Using a loop of "1 to L", let new fac be "fac=fac*(- - L)".
Step 5: Create a function that reverses the string to the right starting from 'k'. 'k' is the index number of the position you want the reversal to start. E.g GR**ACE** has an index no of 01234, 'k' being '1' starts the reversal from 'R' to produce G**ECAR**. Print the new string. Repeat this process from 1 to (fac-1) times using a systematic sequence of 'k' each time, which will produce all the possible combinations of the string by the time (fac-1) approches.
Step 6: Create a function that produces a unique sequence of integers (K). To get the perfect systematic sequence of unique array of 'k' in which if fed into the reverse function sequentially, produces all the possible combinations of the string. Consider the following illustrations:

Illustration 1:

- 0 1 2(array index)
  B**OY** let k = 1 => **BYO**
  BYO let k = 0 => **OYB**
  O**YB** let k = 1 => **OBY**
  OBY let k = 0 => **YBO**
  Y**BO let** k = 1 => **YOB**
  YOB (Reversing this will produce the already existing BOY)

Therefore, the unique array "10101" fed into the function that reverses, produces the possible combinations as shown above. Note, this array combination is unique for every three letter word. Consider the four letter word 'FATE'.

Illustration 2:

- 0 1 2 3(array index)

FATE let k=2	AEFT let k=2	TAEF let k=2
FAET let k=1	AETF let k=1	TAFE let k=0
FTEA let k=2	AFTE let k=2	EFAT let k=2
FTAE let k=1	AFET let k=0	EFTA let k=1
FEAT let k=2	TEFA let k=2	EATF let k=2
FETA let k=0	TEAF let k=1	EAFT let k=1
ATEF let k=2	TFAE let k=2	ETFA let k=2
ATFE let k=1	TFEA let k=1	ETAF

Therefore the unique array "21212021212021212021212" fed into the fxn that reverses, produces the possible combinations. Note, this array combination is unique for every four letter word.

Illustration 3:

- For five letter word, the combination is
  3232313232313232313232303232313232313232313232303232313232313232313232303232313232313232313
  2323.

NOTICE THE UNIQUENESS OF THESE NUMBER SEQUENCE!!!

Code of Snippet:

```
#include <stdio.h>    #include <ctype.h>    #include <string.h>//include the necessary libraries
reverse(char[],int,int);    rev_arr_no(int[],int,int);//defined function prototype
main()
{
        char arr[8]; int i;
        up:printf("Enter any 1-7 letter words\n");
        for(i=0; (arr[i]=getchar())!='\n'; ++i)
        if(i==7)goto up;
        arr[i]='\0';    printf("%10s",arr);      fac=i;    int a = fac;
        for(int j=1; j<i; ++j) fac = fac * --a;
        int ar[fac-1];    rev_arr_no(ar,fac,i);
        for(int j=0; j<=(fac-2); ++j)
        {
              reverse(arr,ar[j],i); printf("%10s",arr);
        }
  }
```

2.0 reverse (char arr[], int k, int i)

This function receives an array of characters ”arr” and two integer variables, ‘k’ and l. Where ‘l’ is the length of the array, and ‘K’ is the index number of the array in which the reversal should begin.

1. Create another character array “g” of size (k+1). Let the last character be the ‘end of string character (‘\0’).
2. Copy the first ‘k’ characters of ‘arr’ into ‘g’. (if k is ‘0’, it copies no character)
3. Create a character array “f” of size (l - k).
4. Create a loop that puts the ‘kth’ character of array ‘arr’ into the last variable of array ‘f’. I.e first to last, 2nd to (2nd to the last) and so on.
5. Cartenate array ‘g’ and ‘f’ together. Puting the result is array arr and return arr.

Code Snippet:

```
reverse(char arr[], int k, int i)
{
        int r=i+1;   char g[k+1];    g[k]='\0';
        strncpy(g,arr,k);    int x=(i-k)+1;
        char f[x];    f[x-1]='\0';
        for (int v=(x-2); v>=0; --v)
        {
              f[v]=arr[k]; ++k;
        }
        strcat(g,f); strcpy(arr,g);
}
```

3.0 rev_arr_no (int ar[], int fac, int l)

This array receives an array “ar”, and two integer variable fac and L. And returns the array “ar” containing the unique sequence of integers suitable for the reverse function to produce all the possible combination.

Array ar[] is the array needed to store the required sequence of numbers. The length of the array is just one less than the factorial (fac) of the word. This is the array passed into the function along with the factorial (fac) and the length (l). Consider the following table:

Letter word	**Unique sequence**
One letter word	"NONE"
Two letter word	0
Three letter word	10101
Four letter word	21212021212021212021212
Five letter word	32323132323132323132323032323132323132323132323032323132323132323132323032323132323132323132323

1. The highest number ('z^0') in the sequence is twice less than the length of the word, (E.g. a five letter word has a unique sequence that starts out at number '3'. Four letter word starts out at number '2'. Three letter word starts out at number '1') and decreases sequentially until it gets to '0'. Let $z^n = a - n$. where n begins at '0' to 'a'. Thus, for a five letter word $z^0 = 3 - 0$, $z^1 = 3 - 1$, $z^2 = 3 - 2$, $z^3 = 3 - 3$. Giving the outputs 'z'= (3,2,1,0) respectively.
2. The initial 'z' (i.e. '3', for a 5-letter word) fills up all even index in the array starting from index '0'.
3. We proceed to fill up all odd index will the next 'z' beginning from index '1'. Although, some odd index will later be replaced by some other values of 'z' in the subsequent iterations. ***(This is the trick!!!)***
4. We now find a way to fill up the array with the correct sequence of 'z'. Consider the following explanations:

NOTE: FOR THE PURPOSE OF PROPER EXPLANATION

ITERATION: Means Looping. *COUNT: Means the number of time a looping process is exited and re-entered.*

- Declare and Initialize the variables a=2, b=1, c=1.
- Create two nested for_loops (preferably).
- The first loop (Let's say loop 'z') represents 'z'. It counts from (L-2) to 0 step -1. i.e, For four letter words, start iterating from 2,1,0 (consider illustration 2 above: for a four letter word only numbers 2,1,0 forms the unique sequence. While that of three and five letter words are 1,0 and 3,2,1,0 respectively).
- The second loop (Let's say loop 'j') represents the array index that would receive a particular set of value. It begins iteration from 'j = (c – 1)' to 'j = (fac – 2)' step 'j = j + a' such that for the first 2 counts, the iterations will increment by '2' i.e. repeats itself after one proceeding index (2,1,2,1,2,0,2,1,2,1,2,0,2….). Where (a = 2) for the first two counts and (a = a * (b + 1)) for subsequent counts (check out the last point for explanations of 'b'). Which makes 'j' begin iteration at index '0' and '1' for the first two counts.
- Let 'c = a' such that each count of loop 'j' will begin iterating at 'c – 1' which is the unique starting index for each count. **Illustrating this;** *Count 1* for loop 'j': initial value of 'a' = 2, initial values of 'c' = 1, then j = 0. *Count 2:* a = 2, c = 2, j = 1.
- Let 'b' (which represents loop 'j' counts) increment sequentially such that it is used to decide when to stop using 'a' as a = 2 and starting using 'a' as a = a*(b + 1). this occurs when 'b' = 2.

Code Snippet:

```
rev_arr_no(int ar[], int fac, int l)
{       int    a=2; int b =1; int c=1;
        for(int i=(l-2); i>=0; --i)
        {       for(int j=(c-1); j<=(fac-2); j+=a)
                {       ar[j]=i;
                }
                if(b<=1)a=2; else a*=(b+1); ++b; c=a;
        }
}
```