+	-	+	+	+	+	+	+	+	+
+	-	+	+	+	+	+	+	+	+
+	-	+	+	+	+	+	+	+	+
+	-	-	-	-	-	+	+	+	+
+	-	+	+	+	-	+	+	+	+
+	-	+	+	+	-	+	+	+	+
+	+	+	+	+	-	+	+	+	+
+	+	-	-	-	-	-	-	+	+
+	+	+	+	+	-	+	+	+	+
+	+	+	+	+	-	+	+	+	+

Words=”LONDON;DELHI;ICELAND;ANKARA”

Prerequistes: - Recursion concept understanding is mandatory

- Backtracking concept understanding is a plus

Algorithm In Steps:

For the 1st word: let's find a proper place for it starting from first row and going down to the last row if we can't find a proper place "Horizontally" let's check for a proper place "Vertically" starting from first column to the last column, if we can't find place too that's mean this crossword puzzle has no full solution.

For the other words:

1- For 2nd word let's find a proper place for it starting from first row and going down to the last row if we can't find a proper place "Horizontally" let's check for a proper place "Vertically" starting from first column to the last column, if we can't find place we **Go Back** (Backtracking) to the previous word (1st word) and try another proper place for it and go again to the 2nd word trying find a proper place for it and so on, if we find a proper place for it we go next to the 3rd word and so on.

Algorithm In Details:

-We need some global variables to help us solving puzzle here they are:

```
- char wordslength[10]={0}; //global array to store each word length
-char numberofoccuipieddashes[10]={0}; //array to store occupied dashes for each word

-char wordoffset[10]={0};//array to store word offset for each word
-char totalnumberofwords=0; //global variable to store total number of words
-char solutionfoundedflag=0; //indicate if a proper complete solution founded or not yet
```

First Step: Let us calculate each word length inside ‘Words’ string and store all lengths inside ‘Wlengths’ array, and also counting number of words in ‘totalnumberofwords’ global variable.

-Words is one line string each word separated by semi column ; and this line is terminated by null character.

```
Getwordlength(wordslength,words); //utility to calculate each word length
```

```
void Getwordslength (char *w)
{
    char index1=0,index2=0,counter=0;
    while((w[index1]!='\0'))
    {
        if(w[index1]==';')
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else
        {
            wordslength[index2]=++counter;
            index1++;
        }
    }
    totalnumberofwords++;
}
```

Words=”LONDON;DELHI;ICELAND;ANKARA”

```
void Getwordslength (char *w)
{
    char index1=0,index2=0,counter=0;
    while((w[index1]!='\0'))→(‘L’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘L’==’;’)→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[0]=1;
            index1++;→index1=1
        }
    }
    totalnumberofwords++;
}
```

```
In This Iteration Index1=1,index2=0,counter=1;
while((w[index1]!='\0'))→(‘O’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘O’==’;’)→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[0]=2;
            index1++;→index1=2
        }
    }
```

```
In This Iteration Index1=2,index2=0,counter=2;
while((w[index1]!='\0'))→(‘N’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘N’==’;’)→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[0]=3;
            index1++;→index1=3
        }
    }
```

```
In This Iteration Index1=3,index2=0,counter=3;
while((w[index1]!='\0'))→('D'!='\0')→TRUE
    {
        if(w[index1]==';') →('D'==';')→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[0]=4;
            index1++;→index1=4
        }
    }
```

```
In This Iteration Index1=4,index2=0,counter=4;
while((w[index1]!='\0'))→('O'!='\0')→TRUE
    {
        if(w[index1]==';') →('O'==';')→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[0]=5;
            index1++;→index1=5
        }
    }
```

```
In This Iteration Index1=5,index2=0,counter=5;
while((w[index1]!='\0'))→(‘N’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘N’==’;’)→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[0]=6;
            index1++;→index1=6
        }
    }
```

```
In This Iteration Index1=6,index2=0,counter=6;
while((w[index1]!='\0'))→(‘;’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘;’==’;’)→TRUE
        {
            index1++;→index1=7;
            index2++;→index2=1;
            counter=0;
            totalnumberofwords++; → totalnumberofwords=1;
        }
        else→FALSE
        {
            wordslength[index2]=++counter;
            index1++;→index1=7
        }
    }
```

```
In This Iteration Index1=7,index2=1,counter=0;
while((w[index1]!='\0'))→('D'!='\0')→TRUE
    {
        if(w[index1]==';') →('D'==';')→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[1]=1;
            index1++;→index1=8
        }
    }
```

```
In This Iteration Index1=8,index2=1,counter=1;
while((w[index1]!='\0'))→(‘E’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘E’==’;’)→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[1]=2;
            index1++;→index1=9
        }
    }
```

```
In This Iteration Index1=9,index2=1,counter=2;
while((w[index1]!='\0'))→(‘L’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘L’==’;’)→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[1]=3;
            index1++;→index1=10
        }
    }
```

```
In This Iteration Index1=10,index2=1,counter=3;
while((w[index1]!='\0'))→('H'!='\0')→TRUE
    {
        if(w[index1]==';') →('H'==';')→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[1]=4;
            index1++;→index1=11
        }
    }
```

```
In This Iteration Index1=11,index2=1,counter=4;
while((w[index1]!='\0'))→('I'!='\0')→TRUE
    {
        if(w[index1]==';') →('I'==';')→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[1]=5;
            index1++;→index1=12
        }
    }
```

```
In This Iteration Index1=12,index2=1,counter=5;
while((w[index1]!='\0'))→(‘;’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘;’==’;’)→TRUE
        {
            index1++;→index1=13
            index2++;→index2=2
            counter=0;
            totalnumberofwords++; → totalnumberofwords=2
        }
        else→FALSE
        {
            wordslength[index2]=++counter;
            index1++;→index1=1
        }
    }
```

```
In This Iteration Index1=13,index2=2,counter=0;
while((w[index1]!='\0'))→('I'!='\0')→TRUE
    {
        if(w[index1]==';') →('I'==';')→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[2]=1;
            index1++;→index1=14
        }
    }
```

```
In This Iteration Index1=14,index2=2,counter=1;
while((w[index1]!='\0'))→(‘C’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘C’==’;’)→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[2]=2;
            index1++;→index1=15
        }
    }
```

```
In This Iteration Index1=15,index2=2,counter=2;
while((w[index1]!='\0'))→(‘E’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘E’==’;’)→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[2]=3;
            index1++;→index1=16
        }
    }
```

```
In This Iteration Index1=16,index2=2,counter=3;
while((w[index1]!='\0'))→(‘L’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘L’==’;’)→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[2]=4;
            index1++;→index1=17
        }
    }
```

```
In This Iteration Index1=17,index2=2,counter=4;
while((w[index1]!='\0'))→(‘A’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘A’==’;’)→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[2]=5;
            index1++;→index1=18
        }
    }
```

```
In This Iteration Index1=18,index2=2,counter=5;
while((w[index1]!='\0'))→(‘N’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘N’==’;’)→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[2]=6;
            index1++;→index1=19
        }
    }
```

```
In This Iteration Index1=19,index2=2,counter=6;
while((w[index1]!='\0'))→(‘D’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘D’==’;’)→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[2]=7;
            index1++;→index1=20
        }
    }
```

```
In This Iteration Index1=20,index2=2,counter=6;
while((w[index1]!='\0'))→(‘;’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘;’==’;’)→TRUE
        {
            index1++;→index1=21
            index2++;→index2=3
            counter=0;
            totalnumberofwords++;→totalnumberofwords=3
        }
        else→FALSE
        {
            wordslength[index2]=++counter;
            index1++;
        }
    }
```

```
In This Iteration Index1=21,index2=3,counter=0;
while((w[index1]!='\0'))→('A'!='\0')→TRUE
    {
        if(w[index1]==';') →('A'==';')→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[3]=1;
            index1++;→index1=22
        }
    }
```

```
In This Iteration Index1=22,index2=3,counter=1;
while((w[index1]!='\0'))→(‘N’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘N’==’;’)→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[3]=2;
            index1++;→index1=23
        }
    }
```

```
In This Iteration Index1=23,index2=3,counter=2;
while((w[index1]!='\0'))→(‘K’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘K’==’;’)→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[3]=3;
            index1++;→index1=24
        }
    }
```

```
In This Iteration Index1=24,index2=3,counter=3;
while((w[index1]!='\0'))→(‘A’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘A’==’;’)→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[3]=4;
            index1++;→index1=25
        }
    }
```

```
In This Iteration Index1=25,index2=3,counter=4;
while((w[index1]!='\0'))→(‘R’!=’\0’)→TRUE
    {
        if(w[index1]==';') →(‘R’==’;’)→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[3]=5;
            index1++;→index1=26
        }
    }
```

```
In This Iteration Index1=26,index2=3,counter=5;
while((w[index1]!='\0'))→('A'!='\0')→TRUE
    {
        if(w[index1]==';') →('A'==';')→FALSE
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else→TRUE
        {
            wordslength[index2]=++counter;→ wordslength[3]=6;
            index1++;→index1=27
        }
    }
```

```
In This Iteration Index1=27,index2=3,counter=6;
{
      while((w[index1]!='\0'))→('\0'!='\0')→FALSE
          {
              if(w[index1]==';')
              {
                  index1++;
                  index2++;
                  counter=0;
                  totalnumberofwords++;
              }
              else
              {
                  wordslength[index2]=++counter;
                  index1++;→index1=26
              }
          }

      totalnumberofwords++; → totalnumberofwords=4;

}
```

-At the end

wordslength[10]={6,5,7,6,0,0,0,0,0,0}

totalnumberofwords=4

Second Step: - Start calling function Getwordoffset, it has no parameter, it calculate something called word offset which is basically how many steps we need to move from first character in words string in order to get first character of each word separately.

```
void Getwordoffset()
{
    wordoffset[0]=0;
    char wordindex=1,index;
    for(;wordindex<totalnumberofwords;wordindex++)
    {
        index=wordindex;
        while(index>0)
            {
                wordoffset[wordindex]=wordoffset[wordindex]+1+wordslength[index-1];
                index--;
            }
    }
}
```

-As an example let's get third (2^{nd}) word offset

Words="LONDON;DELHI;ICELAND;ANKARA"

```
void Getwordoffset()
{
    wordoffset[0]=0;
    char wordindex=1,index;
    for(;wordindex<totalnumberofwords;wordindex++)→wordindex=2
    {
        index=wordindex;→index=2
        while(index>0) → (2>0) → TRUE
            {
                wordoffset[wordindex]=wordoffset[wordindex]+1+wordslength[index-1];
                ➔ wordoffset[2]= wordoffset[2]+1+wordslength[2-1]
                ➔ wordoffset[2]= 0+1+5=6
                index--; → index=1;
            }
      //again we back to while loop
         while(index>0) → (1>0) → TRUE
            {
                wordoffset[wordindex]=wordoffset[wordindex]+1+wordslength[index-1];
                ➔ wordoffset[2]= wordoffset[2]+1+wordslength[1-1]
                ➔ wordoffset[2]= 6+1+6=13
                index--; → index=0;
            }
      //Then we go to the next word
    }
}
```

wordoffset[2]= 6+1+6=13 which means to get to the first letter of third word (which it's index=2), we start at first letter of first word and move 13 steps we found that we reach the first letter of third word.
So *(words+wordoffset[2])= *(words+13)='I' which is first letter of ICELAND which is third word

And So On For All Words, Except First Word Which Has Offset Of Zero

Third Step: - Start calling function SolveCrossWord, it has 3 parameters

```
solvecrossword(crossword,words,0);
```

1-Crossword Puzzle Board

2-Words

3- Current Word Index

And also this function will need more variables which is globally declared no need to pass them to the function

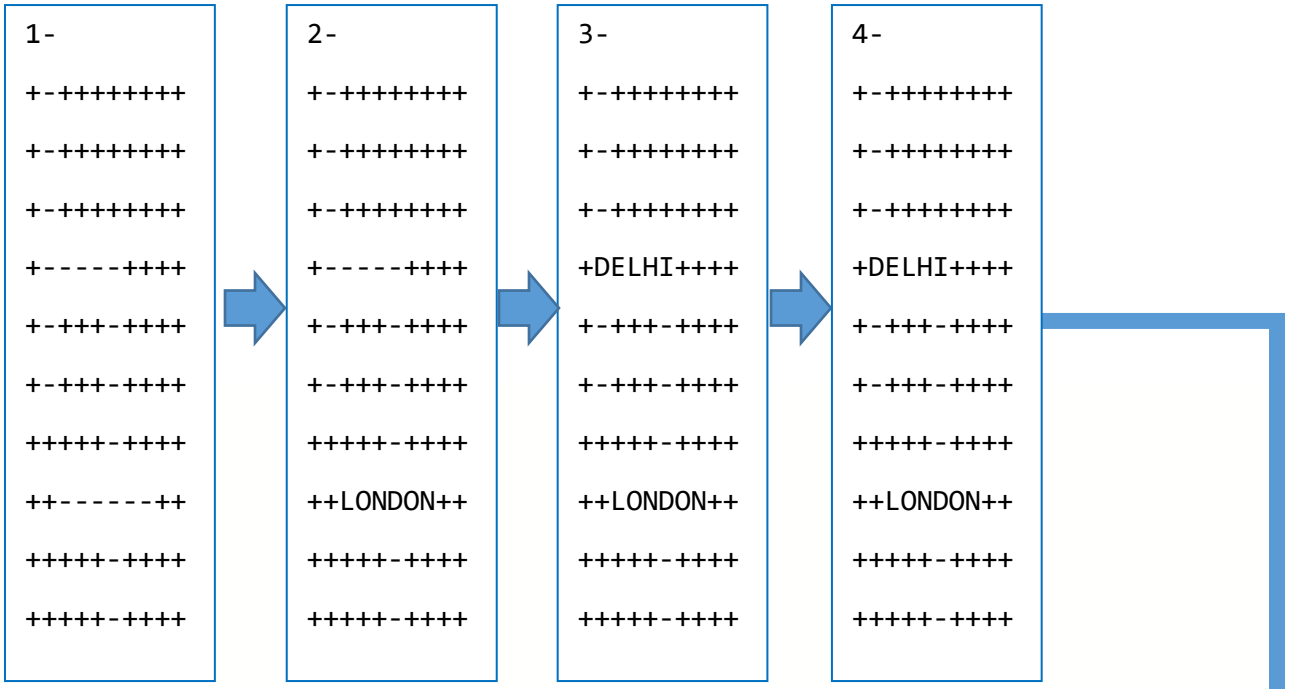
A-Total Number Of Words //which is global variable no need to pass it to function

B- Current Word Length //which can be obtained as word length array is global and we

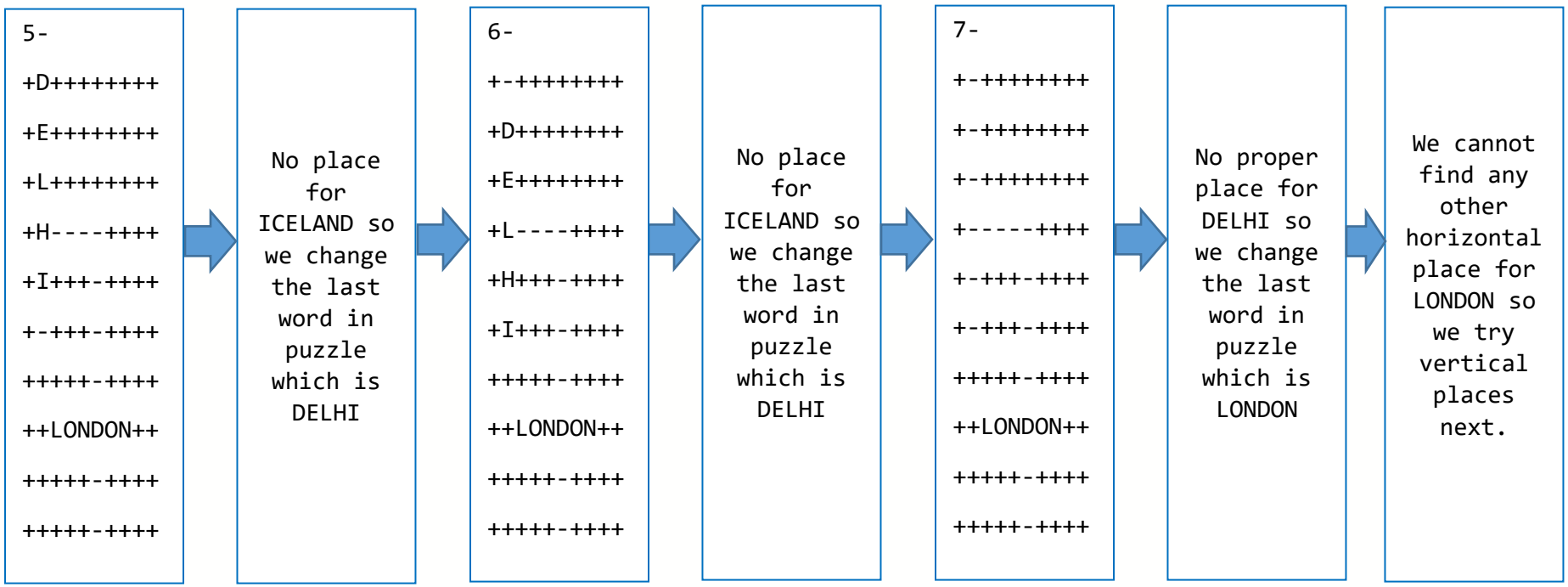
//previously pass the current word index to the function

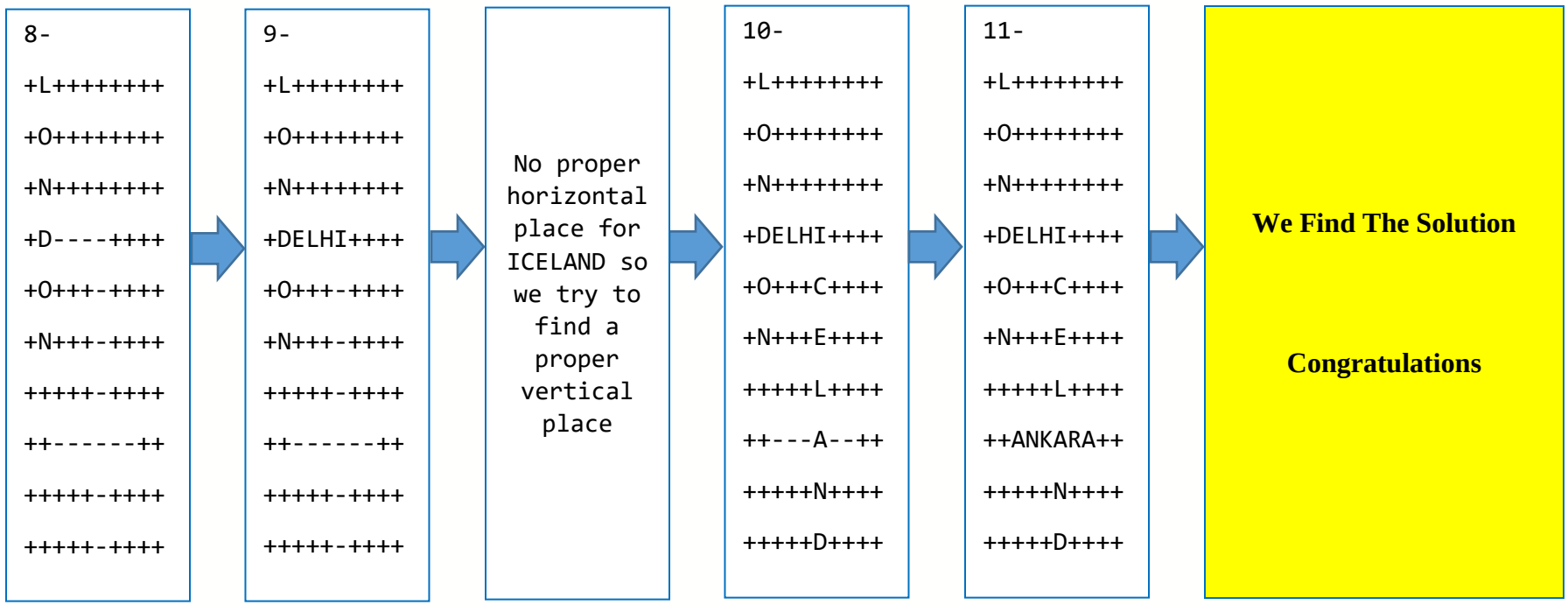
And it return Flag from type char that is indicate if current word has/not a proper place 1/0

➔ Algorithm Emulation:



We cannot find proper place for “ICELAND” horizontally or vertically we backtrack and try another place for last word in puzzle which is DELHI, we cannot find any other horizontal place for DELHI so we try vertical places next.


5-
+D++++++++
+E++++++++
+L++++++++
+H----++++
+I+++-++++
+-+++-++++
+++++-++++
++LONDON++
+++++-++++
+++++-++++
No place for ICELAND so we change the last word in puzzle which is DELHI
6-
+-++++++++
+D++++++++
+E++++++++
+L----++++
+H+++-++++
+I+++-++++
+++++-++++
++LONDON++
+++++-++++
+++++-++++
No place for ICELAND so we change the last word in puzzle which is DELHI
7-
+-++++++++
+-++++++++
+-++++++++
+-----++++
+-+++-++++
+-+++-++++
+++++-++++
++LONDON++
+++++-++++
+++++-++++
No proper place for DELHI so we change the last word in puzzle which is LONDON
We cannot find any other horizontal place for LONDON so we try vertical places next.


8-
+L++++++++
+O++++++++
+N++++++++
+D----++++
+O+++-++++
+N+++-++++
+++++-++++
++------++
+++++-++++
+++++-++++
9-
+L++++++++
+O++++++++
+N++++++++
+DELHI++++
+O+++-++++
+N+++-++++
+++++-++++
++------++
+++++-++++
+++++-++++
No proper horizontal place for ICELAND so we try to find a proper vertical place
10-
+L++++++++
+O++++++++
+N++++++++
+DELHI++++
+O+++C++++
+N+++E++++
+++++L++++
++---A--++
+++++N++++
+++++D++++
11-
+L++++++++
+O++++++++
+N++++++++
+DELHI++++
+O+++C++++
+N+++E++++
+++++L++++
++ANKARA++
+++++N++++
+++++D++++
We Find The Solution
Congratulations

➔ Code:

```
char wordslength[10]={0}; //global array to store each word length
char numberofoccuipieddashes[10]={0}; //array to store occuipied dashes for each word
char wordoffset[10]={0}; //array to store word offset for each word
char totalnumberofwords=0; //global variable to store number of words
char solutionfoundedflag=0; //indicate if a proper solution founded or not yet
void clearcolumn(char** crossword,char* startrowindex,char wordindex, char* columnindex)
{
    char index=wordslength[wordindex]+(*startrowindex)-1;
    char index2=numberofoccuipieddashes[wordindex];
     for(;index2>0;index2--)
    {
        crossword[index][*columnindex]='-';
        index--;
    }
    if(crossword[(*startrowindex+wordslength[wordindex]+1)%10][*columnindex]=='-')
    {
        *columnindex=*columnindex-1;
        *startrowindex=(*startrowindex+1)%10;
    }
    else
    {
        *columnindex=*columnindex;
        *startrowindex=0;
    }
}
void clearrow(char* crosswordrow,char* startcolumnindex,char wordindex,char *rowindex)
{
    char index=wordslength[wordindex]+(*startcolumnindex)-1;
    char index2=numberofoccuipieddashes[wordindex];
    for(;index2>0;index2--)
```

```
    {
        crosswordrow[index]='-';
        index--;
    }
    if(crosswordrow[*startcolumnindex+wordslength[wordindex]]=='-')
    {
        *rowindex=*rowindex-1;
        *startcolumnindex=(*startcolumnindex+1)%10;
    }
    else
    {
        *rowindex=*rowindex;
        *startcolumnindex=0;
    }
}
char try_to_solve_vr(char** crossword,char* words,char wordindex,char columnindex,char* startrowindex)
{
    char index=0,numberofavailabledashes=0,start=*startrowindex;
    for(index=0;index<10;index++)
    {
        numberofavailabledashes=0;
        if((crossword[index][columnindex]=='
')||(crossword[index][columnindex]==*(words+wordoffset[wordindex])))
        {
            start=index;
            while((index<10)&&((crossword[index][columnindex]=='
')||(crossword[index][columnindex]==*(words+wordoffset[wordindex]+index-start))))
            {
                numberofavailabledashes++;
                index++;
            }
```

```
            if(numberofavailabledashes>=wordslength[wordindex])
            {
                numberofoccuipieddashes[wordindex]=0;
                break;
            }
        }
    }
    if(numberofavailabledashes>=wordslength[wordindex])
    {
        *startrowindex=start;
        for(index=0;index<wordslength[wordindex];index++)
        {
            if((crossword[start+index][columnindex]=='-
')||(crossword[start+index][columnindex]==*(words+wordoffset[wordindex]+index)))
            {
                if(crossword[start+index][columnindex]=='-')
                {
                    numberofoccuipieddashes[wordindex]++;
                }
                crossword[start+index][columnindex]=*(words+wordoffset[wordindex]+index);
            }
        }
        return 1;
    }
    else
    {
        return 0;
    }
    return 0;
}
char try_to_solve_hr(char* crosswordrow,char* words,char wordindex,char rowindex,
```

```
char *startcolumnindex)
{
    char index=0,numberofavailabledashes=0,start=*startcolumnindex;
    for(index=start;index<10;index++)
    {
        numberofavailabledashes=0;
        if((crosswordrow[index]=='-')||(crosswordrow[index]==*(words+wordoffset[wordindex])))
        {
            start=index;
            while(((crosswordrow[index]=='-
')||(crosswordrow[index]==*(words+wordoffset[wordindex]+index-start)))&&(index<10))
            {
                 numberofavailabledashes++;
                 index++;
            }
            if(numberofavailabledashes>=wordslength[wordindex])
            {
                numberofoccuipieddashes[wordindex]=0;
                break;
            }
        }
    }
    if(numberofavailabledashes>=wordslength[wordindex])
    {
        *startcolumnindex=start;
        for(index=0;index<wordslength[wordindex];index++)
        {
            if((crosswordrow[start+index]=='-
')||(crosswordrow[start+index]==*(words+wordoffset[wordindex]+index)))
            {
                if(crosswordrow[start+index]=='-')
```

```
                {
                    numberofoccuipieddashes[wordindex]++;
                }
                crosswordrow[start+index]=*(words+wordoffset[wordindex]+index);
            }
        }
        return 1;
    }
    else
    {
        return 0;
    }
    return 0;
}
void Getwordslength (char *w)
{
    char index1=0,index2=0,counter=0;
    while((w[index1]!='\0'))
    {
        if(w[index1]==';')
        {
            index1++;
            index2++;
            counter=0;
            totalnumberofwords++;
        }
        else
        {
            wordslength[index2]=++counter;
            index1++;
        }
```

```
    }
    totalnumberofwords++;
}
void Getwordoffset()
{
    wordoffset[0]=0;
    char wordindex=1,index;
    for(;wordindex<totalnumberofwords;wordindex++)
    {
        index=wordindex;
        while(index>0)
            {
                wordoffset[wordindex]=wordoffset[wordindex]+1+wordslength[index-1];
                index--;
            }
    }
}
char solvecrossword(char** crossword,char* words,char wordindex)
{
    char rowindex,startcolumnindex=0,columnindex,startrowindex=0,foundflag=0,index=0;
    if(wordindex>totalnumberofwords) //base case to stop recursion
    {
        solutionfoundedflag=1; //indication that a proper solution founded
        return 0; //return and stop recursion
    }
    else //recursive case
    {
        if(solutionfoundedflag==0) //check if a prorper solution founded or not
        {
            for(rowindex=0;rowindex<10;rowindex++) //horizontal
            {
```

```
        foundflag=try_to_solve_hr(crossword[rowindex],words,wordindex,
                    rowindex,&startcolumnindex);
        if((foundflag!=0)&&(solutionfoundedflag==0))
        {
            solvecrossword(crossword,words,wordindex+1);
            if(solutionfoundedflag==0)
            {
            clearrow(crossword[rowindex],&startcolumnindex,wordindex,&rowindex);
            foundflag=0;
            continue;
            }
        }
    }
    if(foundflag==0) //if no solution founded in horizontal try vertical
    {
        for(columnindex=0;columnindex<10;columnindex++) //vertical
        {
            foundflag=try_to_solve_vr(crossword,words,wordindex,
               columnindex,&startrowindex);
            if((foundflag!=0)&&(solutionfoundedflag==0))
            {
                solvecrossword(crossword,words,wordindex+1);
                if(solutionfoundedflag==0)
                {
                clearcolumn(crossword,&startrowindex,wordindex,&columnindex);
                foundflag=0;
                continue;
                }
            }
        }
    }
```

```
        }
    }
    return foundflag; //just for the sake of preventg compilation error
}
char** crosswordPuzzle(int crossword_count, char** crossword, char* words, int* result_count) {
Getwordslength(words); //utility to calculate each word length
Getwordoffset(); //utility to calculate each word offset
solvecrossword(crossword,words,0); //utility to solve the crossword puzzle
*result_count=crossword_count;
return crossword;
}
```