Given three strings A, B and C. Write a function that checks whether C is an interleaving of A and B. C is said to be interleaving A and B, if it contains all characters of A and B and order of all characters in individual strings is preserved.

We have discussed a simple solution of this problem here. The simple solution doesn't work if strings A and B have some common characters. For example A = "XXY", string B = "XXZ" and string C = "XXZXXXY". To handle all cases, two possibilities need to be considered.

a) If first character of C matches with first character of A, we move one character ahead in A and C and recursively check.

b) If first character of C matches with first character of B, we move one character ahead in B and C and recursively check.

If any of the above two cases is true, we return true, else false. Following is simple recursive implementation of this approach (Thanks to Frederic for suggesting this)

```
// A simple recursive function to check whether C is an interleaving of A and B
bool isInterleaved(char *A, char *B, char *C)
{
    // Base Case: If all strings are empty
    if (!(*A || *B || *C))
        return true;

    // If C is empty and any of the two strings is not empty
    if (*C == '\0')
        return false;

    // If any of the above mentioned two possibilities is true,
    // then return true, otherwise false
    return ( (*C == *A) && isInterleaved(A+1, B, C+1))
           || ((*C == *B) && isInterleaved(A, B+1, C+1));
}
```

Dynamic Programming

The worst case time complexity of recursive solution is O(2^n). The above recursive solution certainly has many overlapping subproblems. For example, if wee consider A = "XXX", B = "XXX" and C = "XXXXXX" and draw recursion tree, there will be many overlapping subproblems.
Therefore, like other typical Dynamic Programming problems, we can solve it by creating a table and store results of subproblems in bottom up manner. Thanks to Abhinav Ramana for suggesting this method and implementation.

```
// A Dynamic Programming based program to check whether a string C is
// an interleaving of two other strings A and B.
#include <iostream>
#include <string.h>
using namespace std;

// The main function that returns true if C is
// an interleaving of A and B, otherwise false.
bool isInterleaved(char* A, char* B, char* C)
{
    // Find lengths of the two strings
    int M = strlen(A), N = strlen(B);

    // Let us create a 2D table to store solutions of
    // subproblems.  C[i][j] will be true if C[0..i+j-1]
    // is an interleaving of A[0..i-1] and B[0..j-1].
    bool IL[M+1][N+1];

    memset(IL, 0, sizeof(IL)); // Initialize all values as false.

    // C can be an interleaving of A and B only of sum
    // of lengths of A & B is equal to length of C.
    if ((M+N) != strlen(C))
       return false;

    // Process all characters of A and B
    for (int i=0; i<=M; ++i)
    {
        for (int j=0; j<=N; ++j)
        {
            // two empty strings have an empty string
            // as interleaving
            if (i==0 && j==0)
                IL[i][j] = true;

            // A is empty
            else if (i==0 && B[j-1]==C[j-1])
                IL[i][j] = IL[i][j-1];

            // B is empty
            else if (j==0 && A[i-1]==C[i-1])
                IL[i][j] = IL[i-1][j];

            // Current character of C matches with current character of A,
            // but doesn't match with current character of B
            else if(A[i-1]==C[i+j-1] && B[j-1]!=C[i+j-1])
                IL[i][j] = IL[i-1][j];

            // Current character of C matches with current character of B,
            // but doesn't match with current character of A
            else if (A[i-1]!=C[i+j-1] && B[j-1]==C[i+j-1])
                IL[i][j] = IL[i][j-1];

            // Current character of C matches with that of both A and B
            else if (A[i-1]==C[i+j-1] && B[j-1]==C[i+j-1])
                IL[i][j]=(IL[i-1][j] || IL[i][j-1]) ;
        }
    }

    return IL[M][N];
}

// A function to run test cases
void test(char *A, char *B, char *C)
{
    if (isInterleaved(A, B, C))
        cout << C <<" is interleaved of " << A <<" and " << B << endl;
    else
        cout << C <<" is not interleaved of " << A <<" and " << B << endl;
}


// Driver program to test above functions
int main()
{
    test("XXY", "XXZ", "XXZXXXY");
    test("XY" ,"WZ" ,"WZXY");
    test ("XY", "X", "XXY");
    test ("YX", "X", "XXY");
    test ("XXY", "XXZ", "XXXXZY");
    return 0;
}
```

Output:

```
XXZXXXY is not interleaved of XXY and XXZ
WZXY is interleaved of XY and WZ
XXY is interleaved of XY and X
XXY is not interleaved of YX and X
```

```
XXXXZY is interleaved of XXY and XXZ
```

See this for more test cases.

Time Complexity: O(MN)
Auxiliary Space: O(MN)