// A utility function to check if str is palindroem

bool isPalindrome(string str, int low, int high)

{

    while (low < high)

    {

        if (str[low] != str[high])

            return false;

        low++;

        high--;

    }

    return true;

}

// Recursive function to find all palindromic partitions of str[start..n-1]

// allPart --> A vector of vector of strings. Every vector inside it stores

//             a partition

// currPart --> A vector of strings to store current partition

void allPalPartUtil(vector<vector<string> >&allPart, vector<string> &currPart,

                   int start, int n, string str)

{

    // If 'start' has reached len

    if (start >= n)

    {

        allPart.push\_back(currPart);

        return;

    }

    // Pick all possible ending points for substrings

    for (int i=start; i<n; i++)

    {

        // If substring str[start..i] is palindrome

        if (isPalindrome(str, start, i))

        {

            // Add the substring to result

            currPart.push\_back(str.substr(start, i-start+1));

            // Recur for remaining remaining substring

            allPalPartUtil(allPart, currPart, i+1, n, str);

            // Remove substring str[start..i] from current

            // partition

            currPart.pop\_back();

        }

    }

}

// Function to print all possible palindromic partitions of

// str. It mainly creates vectors and calls allPalPartUtil()

void allPalPartitions(string str)

{

    int n = str.length();

    // To Store all palindromic partitions

    vector<vector<string> > allPart;

    // To store current palindromic partition

    vector<string> currPart;

    // Call recursive function to generate all partiions

    // and store in allPart

    allPalPartUtil(allPart, currPart, 0, n, str);

    // Print all partitions generated by above call

    for (int i=0; i< allPart.size(); i++ )

    {

        for (int j=0; j<allPart[i].size(); j++)

            cout << allPart[i][j] << " ";

        cout << "\n";

    }

}