The best programs are written so that computing machines can perform them quickly and so that human beings can understand them clearly. A programmer is ideally an essayist who works with traditional aesthetic and literary forms as well as mathematical concepts, to communicate the way that an algorithm works and to convince a reader that the results will be correct. — **Donald E. Knuth**

Algorithms

Dynamic Programming

Problem :-

Splitting a String into minimum number of **palindromes**.
Given a string, find the minimum no. of palidromes into which the string can be broken.
Consider the string **"civic"** (String is Palindrome by itself, so min no. of palindromes = **1**)
The string **"abcde"** cannot be splitted into less than **5** palindromes (each character is a palindrome)
Consider the string **"civicacaramanamaracacammacdeified"**.
It can be broken into 4 palindromes **" civic " + " acaramanamaraca " + " cammac " + " deified "**

Solution :-

Suppose the size of the string is **n**.
We maintain a 2D array **palins [n][n]** where **palins [i][j]** stores the minimum number of palindromes in the string **str [i ... j]** i.e from index **' i '** to index **' j '** of the string. So, the final solution of breaking the string of size **n** into minimum no. of palindromes is given by **palins [0][n - 1]**.
Let us determine the structure of the optimal solution.
Let **palins (i , j)** denotes the min no. of palindromes in the string **str (i ... j)**.
We will try to break the string at all possible positions **k** such that **i <= k < j** and sum up the minimum no. of palindromes in **palins (i , k)** and **palins (k , j)**.
Thus, the structure of optimal solution is determined recursively as :
palins (i , j) = palins (i , k) + palins (k , j) for all i <= k < j
Finally, **palins (0 , n - 1)** gives the required solution. See the implementation below.

```
#include<iostream>
#include<cstring>
using namespace std;

// check if a string is palindrome, given the start and end index of the str
bool isPalindrome(char str[], int i, int j) {
   while (i < j) {
      if (str[i++] != str[j--])
         return false;
   }
   return true;
}

/* Find the minimum no. of palindromes in the input string i.e the
   string has to broken down into min. no. of palindromes
   Suppose palins[i][j] -> min no. of palindromes in the string str[i...j]
   Then, palins[0][n-1] is the final solution
   Standard Recursive Solution :-
   palins(i,j) = min(palins(i,k) + palins(k,j)) for i <= k < j
*/
int minPalindromes(char str[]) {
   int i, j, k;
   int substr_size; //substring size
   int n = strlen(str);
   // Matrix where cell (i,j) stores the minimum no. of palindromes in str[i
   int palins[n][n];

   // iterating over all possible substring sizes
   for (substr_size = 1; substr_size <= n; substr_size++) {
      // 'i' and 'j' tracks the start and end index of the current substring
      for (i = 0; i <= n - substr_size; i++) {
         j = i + substr_size - 1;
         if (isPalindrome(str, i, j)) {
            // substring is a palindrome
            palins[i][j] = 1;
         }
         else {
            // search for palindromes within the substring
            // worst case (min. no of palindromes = length of the substring)
            // each character is a palindrome
```

```
41              palins[i][j] = substr_size;
42              // find min no. of palindromes within substring
43              for (k = i; k < j; k++) {
44                 // try splitting substring at each position
45                 int sum = palins[i][k] + palins[k+1][j];
46                 if (sum < palins[i][j])
47                    palins[i][j] = sum;
48              }
49           }
50        }
51     }
52     // return min no. of palindromes in the input string
53     return palins[0][n-1];
54  }
55
56  // main
57  int main() {
58     char str[] = "civicacaramanamaracacammacdeified";
59     //civic + acaramanamaraca + cammac + deified (4 palindromes)
60     int min_palin = minPalindromes(str);
61     cout<<"\nMinimum no. of palindromes :: "<<min_palin;
62     cout<<endl;
63     return 0;
64  }
```

Compile & Run

Back | Next

All problems on Dynamic Programming

* Find the nth term of fibonacci series
* Evaluate combination(n, r)
* Solve the Edit-Distance problem
* Longest Common Subsequence (LCS) problem
* Given a set of coin denominations, find the minimum number of coins required to make a change for a target value
* Longest Increasing Subsequence (LIS) problem
* Unbounded Knapsack problem
* 0/1 Knapsack problem
* Splitting a string into minimum number of palindromes