```
//==========================================================================
// Kevin Benson and Taylor Heilman
// May 2, 2016
// dynamic.cpp
//
// This file contains both the recursive and dynamically-programmed solutions
// to problem 15.2 of the textbook.
//==========================================================================
#include <string.h>
#include <iostream>
#include<stdio.h>
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <sys/time.h>
using namespace std;
//=======================================
// reverse helper function
//=======================================
string reverse(string s)
{
        int n = s.length();
        if (n == 0)
                return string("");
        return s[n - 1] + reverse(s.substr(0, n - 1));
}
//=======================================
// max helper function
//=======================================
int max (int x, int y) {
        if(x > y)
                return x;
        else
                return y;
}
//=======================================
// Recursive palindrome function
// Returns the length of the longest palindromic subsequence in s
//=======================================
string palindrome(string s, int i, int j){
        if(i==j){
                string c;
                c += s[i];
                return c;
        }
        if(i+1 == j and s[i] == s[j]){
                string  toreturn;
                toreturn = toreturn + s[i];
                toreturn = toreturn + s[j];
                return toreturn;
        }
        if(s[i] == s[j]){
                string toreturn;
                toreturn = toreturn + s[i];
                toreturn = toreturn + palindrome(s, i+1, j-1);
                toreturn = toreturn + s[j];
                return toreturn;
        }

        // Cannot use max to compare strings based on length, so do it the long way
        if((palindrome(s, i+1, j).length()) > (palindrome(s, i, j-1).length()))
                return palindrome(s, i+1, j);
        else
```

```
                return palindrome(s, i, j-1);
}

//========================================
// Dynamic Programming palindrome function
// Returns the length of the longest palindromic subsequence in s
//========================================
string dynamicPalindrome(string str)
{
        int l = str.length();
        int c[l][l];      // value table, which is square as we're funcitonally comparing two
identical strings
        // *** INITIALIZE THE TABLES ***

        for (int i = 0; i < l; i++){
                for (int j = 0; j < l; j++){
                        c[i][j] = 0;
                }
        }
        // *** FILL DIAGONAL OF TABLE WITH ONES ***
        for (int i = 0; i < l; i++){
                c[i][i] = 1;
        }
        for (int i = 1; i < l; i++)
        {
                for (int j = i-1; j >=0; j--)
                {
                        if (str[i] == str[j]){
                                c[i][j] = c[i-1][j+1] + 2;
                        }
                        else
                        {
                                c[i][j] = max(c[i-1][j], c[i][j+1]);
                        }
                }
        }

        int i = l-1;
        int j = 0;

        // Solution to subproblem f(m,n) is at index [m][m-n], so our final
        // solution is at the top right of our table
        int palLength = c[l-1][0];

        // Backtrace our array to find the palindrome itself
        string palindrome;
        while(c[i][j] != 0){
                if(c[i][j] != c[i-1][j] and c[i][j] != c[i][j+1]){
                        palindrome = palindrome + str[i];
                        i--;
                        j++;
                }
                else if (c[i][j] == c[i-1][j])
                        i--;
                else
                        j++;
        }
        // Reconstruct palindrome based on the parity of its length
        if(palLength % 2 == 0){
                palindrome = palindrome + reverse(palindrome);
        }
        else{
                for(int i = palindrome.length()-2; i >=0; i--){
```

```
                        palindrome += palindrome[i];
                }
        }

        return palindrome;
}

int main(){
        string q = "antidisestablishment";
        int i = q.length();

        // Recursive solution time analysis
        long diffSeconds, diffUSeconds;
        timeval timeBefore, timeAfter;
        gettimeofday(&timeBefore, NULL);

        string pal = palindrome(q, 0, i-1);

        gettimeofday(&timeAfter, NULL);
        diffSeconds = timeAfter.tv_sec - timeBefore.tv_sec;
        diffUSeconds = timeAfter.tv_usec - timeBefore.tv_usec;

        cout << "Recursive palindrome discovery time: " << diffSeconds + diffUSeconds/1000000.
0 << " seconds" << endl;
        cout << "palindrome is: " << pal << endl;
        cout << "palindrome has length = " << pal.length() << endl << endl;

        // Dynamic solution time analysis
        gettimeofday(&timeBefore, NULL);

        pal = dynamicPalindrome(q);

        gettimeofday(&timeAfter, NULL);

        diffSeconds = timeAfter.tv_sec - timeBefore.tv_sec;
        diffUSeconds = timeAfter.tv_usec - timeBefore.tv_usec;

        cout << "dynamic programming palindrome discovery time: " << diffSeconds + diffUSecond
s/1000000.0 << " seconds" << endl;
        cout << "palindrome is: " << pal << endl;
        cout << "palindrome has length = " << pal.length() << endl;

        return 0;
}
```